An Overview of a Framework for Designing Robot Autonomy Stacks in Simulation

Thomas Hansen[1], Shouvik Chatterjee[1], Sriram Ashokkumar[1], Ishaan Mahajan[1], Harry Zhang[1], Stefan Caldararu[1], Abhiraj Dashora[1], <u>Aaron Young</u>[2], He Shen[3], Luning Fang[1], Dan Negrut[1]

[1] Department of Mechanical Engineering
University of Wisconsin-Madison
Madison, WI, 53705, USA
{thansen8,schatterjee9,ashokkumar2,
imahajan,hzhang699,scaldararu,dashora,
lfang9,negrut}@wisc.edu

[2] Department of Mechanical Engineering
Massachusetts Institute of Technology
Boston, MA, 02139, USA
aryoung@mit.edu

[3] Department of Mechanical Engineering,
The California State University - Los Angeles,
Los Angeles, California, United States
he.shen@calstate.edu

ABSTRACT

We describe a software framework and a hardware platform used in tandem for the design and analysis of robot autonomy algorithms in simulation and reality. The software, which is open source, containerized, and operating system (OS) independent, has three main components: a ROS 2 interface to a vehicle simulation framework (Chrono), which provides high-fidelity wheeled/tracked vehicle and sensor simulation; a basic ROS 2-based autonomy stack for algorithm design and testing; and, a development ecosystem which enables visualization, and hardware-in-the-loop experimentation in perception, state estimation, path planning, and controls. The accompanying hardware platform is a 1/6th scale vehicle augmented with reconfigurable mountings for computing, sensing, and tracking. Its purpose is to allow algorithms and sensor configurations to be physically tested and improved. Since this vehicle platform has a digital twin within the simulation environment, one can test and compare the same algorithms and autonomy stack in simulation and reality. This platform has been built with an eye towards characterizing and managing the simulation-to-reality gap. Herein, we describe how this platform is set up, deployed, and used to improve autonomy for mobility applications.

Keywords: High-fidelity Simulation, Robotics, Sim2real Gap, Robot sensing systems

1 INTRODUCTION

1.1 Motivation

This contribution is concerned with the idea of using simulation to facilitate the task of developing algorithms that improve the autonomy of wheeled and tracked robots operating on rigid or deformable terrains. The value proposition is tied to the cost-effective manner in which data can be generated in simulation as well as to the ease and safety with which candidate solutions can be tested and iterated upon. Using simulation in robotics requires a dynamics engine and a model. Should one make the necessary investment to understand both how the simulator and model should be configured, simulation provides insights that are difficult to obtain in physical testing, e.g., complete state information and quantitative insights about the interaction between the robot and environment it operates in. Unfortunately, these insights do not always lead to decisions that work well in reality due to the simulation-to-reality gap [1]. How to close this sim-to-real gap remains an open problem, and solutions have been proposed that include randomizing the

Figure 1: 1/6th scale autonomous vehicle (V-1) showing 3D printed reconfigurable mountings and support of camera, imu/gnss, and Jetson AGX, the later running the autonomy stack.

experience of the robot inside the simulator [2, 3], using adversarial learning to capture unknown components of the model via ghost external perturbations in an adversarial reinforcement learning framework [4], using ensembles of models [5], using a mix of simulation-generated and real-world data to train robots [6, 7], etc.

Despite these and other similarly valuable contributions, the community lacks an objective understanding of what exactly produces the sim-to-real gap. Our contribution is motivated by this observation and inspired by the belief that an open source autonomy research testbed can be a catalyst for research in two areas: improvement of algorithms for autonomy in mobility; and understanding and mitigating the sim-to-real gap. To that end, we describe herein an infrastructure, i.e., a physical and software platform, that allows one to experiment with an autonomy algorithm, e.g. a sensor-fusion approach, and have the opportunity to do so first in simulation and then quickly on the actual robot. The principle is to have *one* ROS 2 [8] autonomy stack that uses the same collection of perception, state estimation, planning, and control algorithms both in simulation *and* in the real world. This can enable one to perform controlled tests to understand and evaluate the performance of the algorithms.

1.2 Contribution

Our contribution is twofold. First, we established an autonomy development environment (called ART, from "autonomy research testbed") and an accompanying hardware platform (called V-1, for Vehicle 1) that in tandem facilitate two types of technical pursuits: research into algorithms for autonomy in mobility, in the context of wheeled and tracked vehicles; and quantitative characterization of the sim-to-real gap in robotics. Second, we developed an autonomy toolkit (ATK) which is a set of command line tools for building the container system that underpins ART.

The autonomy research testbed is containerized to provide an OS agnostic platform that fosters collaboration and accelerates the research effort since it: 1) mitigates the startup time for setting up an autonomy stack by providing its bridges and interfaces to other system components, e.g., sensors and bridge to simulation; 2) abstracts the components of the autonomy stack thus enabling researchers to quickly replace the algorithm(s) it embeds; 3) makes available a proven simulation environment for tracked/wheeled vehicles, operating on/off-road conditions by allowing communication with Chrono [9]; and, 4) leverages a 1/6th scale vehicle for real world testing that anchors the algorithm validation exercises as well as the sim-to-real inquiry efforts. The hardware component of V-1 (the 1/6th scale vehicle shown in Fig. 1) has been chosen for its topology, payload capacity, and off-road ability. In term of its topology, the suspension and steering mechanism closely mirror those of a full-size vehicle. Finally, with a wider wheel base and tire format, the

vehicle can operate in off-road conditions.

From a high vantage point, our main accomplishment is that a basic autonomy stack has been set up and Docker-containerized. It can be used either with the associated vehicle, to test autonomy algorithms, or in a simulator using the Chrono digital twin. This is the core of ART. Additionally, ATK provides the tools to build the ART ecosystem and collect, visualize, and analyze data coming from ART, which others can leverage for custom containerized environments. ART and ATK allow one to (a) carry out research in the sim-to-real gap; (b) generate data for data-driven solutions in robotics; and (c) improve autonomy algorithms (perception, state estimation, planning, controls) in plug-and-play fashion by working exclusively on the V-1 vehicle, or by combining physical testing and physics-based simulation. The simulator which ART leverages, has the following features: easy wheeled/tracked vehicle model set up via templates [10]; sensor simulation [11]; terradynamics support [12, 13]; support for multiple agents [14]; Python bindings [15].

The subsequent portions of the contribution are organized as follows. To contextualize this contribution, we provide next an overview of similar community efforts. Section 2 highlights the container build tools associated with this project. Section 3 describes the autonomy research testbed by outlining the autonomy stack and its structure, the bridge to the simulator [16], the companion vehicle platform, and its digital twin. This information, along with the online documentation referenced herein should be sufficient for an interesting party to replicate and deploy V-1 at a modest cost should one have access to a 3D printer.

1.3 State of the Art

MuSHR [17], an open-source race-car project, comes closest in spirit to ART. For simulation, MuSHR draws on Gazebo [18]. It uses a 1/10th form-factor vehicle, has one 2D lidar sensor, one stereo camera, and a Jetson Nano processor. MuSHR is ROS-based and runs off a Linux Ubuntu distribution. It has comprehensive documentation and has been used in classes at University of Washington. Another well known platform is MIT RACECAR [19, 20]. It uses a 1/10th form factor, ZED stereo camera, 2D lidar. The software stack, which is ROS-anchored, is provided as a Docker image. Simulation support comes via Gazebo. MuSHR and RACECAR share the same pros and cons – open source platform, ROS-backend, proven software stack; and, respectively, small form factor, indoors use, software and hardware infrastructure relatively difficult to expand beyond the current vehicles.

Karr [21] is designed to only use a depth camera, and requires walls on either side of the car to guide the vehicle along the path. It is a 1/10th form factor, uses NVIDIA's Jetson TX1 chip that comes with a developer kit targeting visual computing running in a pre-flashed Linux environment. Karr relies on ROS and trains a neural net using Keras with a Theano backend using real-life videos. There is no simulator bundled, yet Carla [22] is mentioned as a next logical step towards the goal of training in simulation.

Donkey Car [23] is a popular hobbyist project, which is concerned with racing small, 1/16th or 1/10th, format vehicles that mostly draw on Raspberry Pi 3b+ and a wide-angle camera. It can be fitted with Jetson Nano or TX2 with access to the NVIDIA developer kit and its ecosystem. Donkeycar uses Python for controlling the car, and Unity [24] as a simulator, the latter building off the NVIDIA PhysX simulation engine [25]. The Donkeycar infrastructure is popular, see, for instance [26] for an offshoot. One project that stands out in terms of form factor is PARV/MPAD; at 1/8th is larger than all the rest, but smaller than the platform discussed herein. While the project has modest goals on the autonomy component, it is remarkable in that it is a 3D printed hardware platform [27] that comes with a basic autonomy stack called MPAD (Modular Package for Autonomous Driving), which is Python based, uses OpenCV with a Rasberry Pi camera, has Ultrasonic sensors, an IMU, and runs off a Rasberry Pi 4 [28]. Vehicle control draws on an Arduino Mega. A Raspberry Pi-based solution with Arduino for vehicle control is described in [29]; it is small form factor and works by neural-net-enabled image recognition in conjunction with

scaled-down traffic signs (data not collected in simulation, neural net trained with data collected by driving car around manually). Another hobbyist solution is PiCar of SunFounder [30], which at less than 1/10th format is an inexpensive platform that uses Raspberry Pi 4 and a wide-angle USB camera, with controls designed in EzBlock [31], an open platform for building intro-level electronic projects.

All solutions mentioned above are different in at least one of the following four aspects: (1) project breadth of scope; (2) scale and flexibility of hardware platform; (3) degree to which project embraces and promotes a simulation component; (4) documentation and support. For (1), ART is a researcher's (instead of hobbyist's) platform that uses, but is not limited to, the vehicle shown in Fig. 1. Of the solutions mentioned above, only MuSHR has a similar research purpose. For (2), we adopted a larger form factor since the interest is on- and off-road mobility. Moreover, the goal is enabling sensor fusion, experimenting with various sensor packages, etc., which requires additional payload capacity. For (3), this group is interested in simulation first and foremost, and the vehicle is not the goal, but rather a means to an end. The ends are enabling research in autonomy for mobility and better understanding and management of the sim-to-real gap. For (4), a documentation infrastructure details our solution [32].

2 AUTONOMY TOOLKIT

The objective of ATK is to provide a modular and portable framework for developing, testing, and deploying autonomous algorithms in simulation and reality. This toolkit is a Python package that leverages Docker by wrapping Docker Compose to build and deploy a multi-container system specifically designed for autonomy research. ATK wraps Docker Compose in that all of the functionality of Compose is still available; however, ATK also provides utilities, defaults, documentation, and examples specifically for applications relating to autonomous algorithm development. ATK is open-source, available on GitHub, and has been made available through the Python Package Index (PyPI) [33].

A requirement of this toolkit is to be OS portable. To that end, a containerized system, set up to leverage Docker, is separated into "services" that can be combined to produce a container network that supports complex interactions across a variety of projects. To expedite the deployment process to the physical hardware under test, the same containerized system is used. The Docker interface, especially on Linux systems, is extremely lightweight and introduces negligible overhead for most applications. The development framework is built into two main components: the ATK Python package and the Docker containers themselves. The ATK package, which helps generate the containers, is described in Section 2.1; the container system is described in Section 2.2.

2.1 autonomy-toolkit Package

The package which generates the container system is, in its simplest form, a wrapper of Docker Compose. Called *autonomy-toolkit*, or *ATK* for short, this Python package is publicly available and provides a cross-platform command-line interface. ATK leverages the multi-container architecture to provide optional, extensible, and customizable containers to facilitate development of AVs that greatly increases the generality of the development workflow. The only non-Python requirement for the autonomy-toolkit is Docker. With Docker available, the toolkit can be installed through typical python mechanisms like `pip`. ATK works as follows: YAML configuration files are combined with defaults provided through ATK into a configuration file readable by Docker Compose. From there, ATK simply makes calls to Docker Compose based on the resulting output. All defaults provided through ATK are customizable. Furthermore, the generic Docker Compose commands can be used in conjunction with the generated config file, if desired, without ATK.

ATK was built with modularity and expandability in mind. This means the functionality of the toolkit itself is agnostic of the autonomy stack it is used for, i.e. it is not tied to ART. Individual hardware platforms and control stacks may implement their own containers and customize the

(a) ART environment, setup used on the physical vehicle.

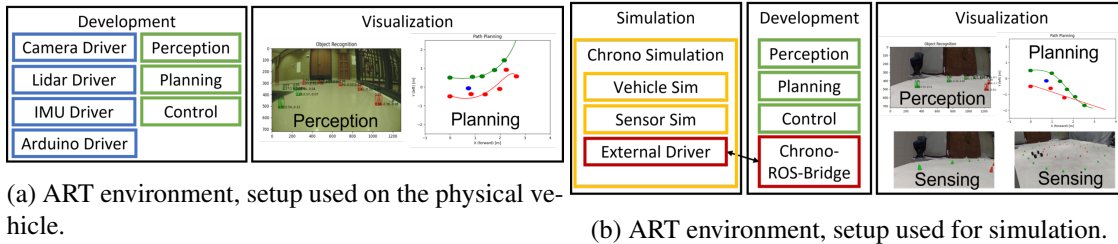(b) ART environment, setup used for simulation.

Figure 2: ART environment setup

default configurations, provided they follow the few requirements outlined in the documentation [32]. As ATK is agnostic of the hardware platform and usage, it can also be used to generate custom containers outside the original scope of the toolkit. For instance, the simulation container to be discussed further in Section 3.1 leverages ATK to create its own container for Chrono. Other common use cases could include a container for training data or an HD map, a deployment container for specific hardware, an automated testing environment, etc.

2.2 Container System

Distributed with the ATK package itself are many predefined and customizable utilities that are used to generate the images and containers. In general, there are two primary services that come directly with the ATK package: `dev` and `vnc`.

`dev`: This is the primary component that is used for algorithm development. It is defined by a custom Dockerfile which utilizes build arguments specified through a configuration file to generate an image specific to the project being developed. By default, `dev` builds on the ROS 2 Galactic image, the most recent ROS 2 distribution at the time of writing. By default, `dev` is assumed to be the primary container and, when launched, initializes a shell environment. The directory that holds the configuration file is mounted into the container for data persistence upon container termination. Other utilities and configuration are performed to further enhance the generality of the toolkit and remove OS dependence.

`vnc`: To make cross-platform visualization easier, we have created a `vnc` container which requires no host setup which allows the user to visualize their applications. The implemented solution leverages Virtual Network Computing (VNC) within a separate service, making its usage optional. To use VNC visualization, `dev` (or any other container run using ATK) must simply attach to the same network and configure its `DISPLAY` variable to that of the `vnc` hostname. NoVNC, a browser based VNC client, can then be used to view any displayed windows from `dev` from any internet browser.

Note that a networking layer is set up to facilitate distributed container interaction. This feature supports hardware in the loop experiments, where the simulation container is deployed on separate hardware from the autonomy stack. To facilitate this, we leverage Docker and Docker Compose networking features. When services are deployed on the same system, the default network configurations are used. However, when the services are deployed in a distributed fashion, the network is customized to expose the necessary ports to facilitate inter-computer communication.

Since the environment is containerized, it can be deployed to a physical vehicle, with optional hardware-specific optimizations. While ATK is general purpose, an example of the containerized system can be seen in Figs. 2a and 2b, which illustrate the services when running on the real vehicle vs. simulation, respectively.

3 AUTONOMY RESEARCH TESTBED

3.1 Simulation

One of the motivations behind this effort was the use of simulation as a means of designing and testing autonomous algorithms. To support this, a bridge to Chrono [16] was developed to allow direct integration within the containerized system for ART. The bridge, along with the custom Chrono container, is an example of the modular environment produced by ATK. Not only can we test in Chrono on a workstation, but this setup paired with ATK allows us to test with hardware in the loop, running the simulation on a workstation while running the autonomy stack on the Jetson.

The Chrono ROS Bridge is a ROS 2 package that integrates with Chrono to feed messages to and from the simulation scenarios. The interface is built on JSON messages so users can send *any* data between ROS and Chrono. Generic publishers and subscribers (a feature of ROS 2) are leveraged to allow for user defined message types and topic names to be used. The package, written in C++, is open source and available under a BSD3 license.

On the Chrono side, the bridge builds directly on core functionality available through the utility class ChSocket. Additional hooks were provided that wrap the JSON generation from user code. This was done to facilitate Python wrapping with PyChrono, the Python bindings to the C++ Chrono API. This allows the researchers to leverage the rapid development process enabled by Python. Additionally, to allow custom message types to be sent between ROS and Chrono, custom functors may be implemented to generate and/or parse custom message formats.

3.2 Autonomy Stack

The autonomy stack developed for ART is built on top of ROS 2 and includes basic implementations of autonomy algorithms to enable autonomy experiments in simulation and reality. The stack is basic; there is nothing particularly novel in its implementation. It utilizes publicly available algorithms and packages typically shipped with ROS 2. This is because the focus of ART is not on the autonomy stack itself, but the ability to compare algorithms between sim and reality. Against this backdrop, the perception algorithm is a custom trained instance of Faster-RCNN [34] built on a MobileNetV3 [35] network. MobileNet is intended for mobile phone CPUs, but was adopted for ART considering the limited compute power available on the vehicle. For the example used later in this contribution, in which a vehicle navigates down a lane set up with cones, the perception algorithm was trained using both simulated and real images. Based on the bounding box output from Faster-RCNN, 3D cone positions are generated at each timestep and a simple planner and controller are used to keep the vehicle on the path defined by the cones. To control the car, throttle, braking, and steering inputs are communicated to the car via an Arduino processor.

3.3 Vehicle Chassis

The vehicle platform is built upon a 1/6th scale remote controlled car. With a 47 cm wheel base and a 34 cm track width, the vehicle is large enough to carry multiple cameras and sensors. This is critical since research into the difference between simulation and reality when applied to sensor simulation must perform analysis on the specific sensors of interest in full-scale applications. The base vehicle includes a double wishbone independent suspension at the front and rear. The kinematics of this suspension allow for use in off-road mobility. For actuation, the vehicle includes a 1300 KV brushless motor and a 15 kg-cm servo for steering which we upgraded to a 25 kg-cm servo for durability. The servo controls the steering through a Pitman arm steering mechanism.

3.4 Electronic Hardware

The base RC car was augmented to allow for direct control of the steering and throttle via an onboard computer. The modifications were minimal to allow for duplication of the system. First, the dual-battery setup powering the electronic speed controller (ESC) was reduced to a single two-cell LiPo battery, which lowers the top speed of the vehicle to something more manageable.

The use of a dedicated battery for electronics ensures that voltage spikes and drops induced by the motor do not affect any sensitive computer or sensing electronics. To prevent floating voltage issues, all ground connections are wired to a common ground rail. A power distribution and control wiring diagram along with a bill of material are provided with the hardware documentation online [32].

The computer used on the ART vehicle is a Jetson Xavier AGX, although other compute platforms could also be used. The design of the vehicle and development environment are compute system agnostic as we've used a Jetson Xavier AGX and Orin to run the autonomy stack. To facilitate this, direct control of the servo and ESC is performed through PWM signals from an Arduino chip. On the vehicle herein, an Arduino Uno was leveraged, but an Arduino Nano is considered in the documentation. The Arduino is used exclusively as a device driver, performing no autonomy. The ART vehicle is equipped with a USB camera and a VLP-16 lidar. Further sensors can be mounted on the vehicle to facilitate additional research needs.

3.5 Mounting, Expansion, and Reconfigurability

The ability to reconfigure the vehicle platform was a high priority given the "research-platform mission" this solution must fulfill. To that end, all mounting components were designed to be 3D printed, with a base plate for electronics that could be laser cut or 3D printed. The final setup is shown in Fig. 1 with the electronics mounted above the motor and ESC on the base RC car. While most electronics are mounted directly to the electronics board, some of the sensors require custom mounts. The camera is located in the front of the vehicle above the bumper to provide an unobstructed view. The imu/gnss sensor is mounted above the back wheels, and there is an optional custom lidar mount which provides an unobstructed view of the surroundings from above the Jetson computer. Lastly in addition to electronics and sensors, a motion capture tracking system can be seen mounted to the vehicle in Fig. 1 using some of the holes in the components board. This allows for millimeter level tracking when used in combination with an OptiTrack Motion Capture system.

3.6 Vehicle Digital Twin

The vehicle described in this contribution has a digital counterpart modeled in Chrono using Chrono::Vehicle [10] and Chrono::Sensor [11]. The vehicle model is implemented with a double wishbone suspension and a linear spring-damper with parameters estimated from the vehicle mass. The Chrono model of the car is rendered in Fig. 3 using Chrono::Sensor and highlights the mesh representation of the car along with the double wishbone configuration.

<div align="center">(a) Front view of ART model.   (b) ISO view of ART model.</div>

Figure 3: Chrono::Vehicle model of ART as visualized with Chrono::Sensor.

The steering model uses the Chrono::Vehicle Pitman arm template, allowing actuation of the steering arm. The maximum steering angle was calibrated from a set of minimum radius turn tests using motion tracking. The motor model is a simple linear torque-speed curve, with decreasing power

with motor speed. It also includes a Motor losses curve to model the motors initial resistance and loss as the RPM increases. These parameters were calibrated using Motion Capture data.

4 DEMONSTRATION

To demonstrate ART at work, a path of cones was set up in our motion tracking lab. The cone locations were recorded and injected into a simulation of the same motion. The vehicle then drove along the path in simulation and reality. Images of this setup from a similar location for real and sim are provided in Fig. 4, which shows the same cone paths and the vehicle in a similar location on the path. To navigate, the vehicle exclusively uses its front facing camera. This demonstration is included in the supporting video and can be found online [36].

(a) Real setup
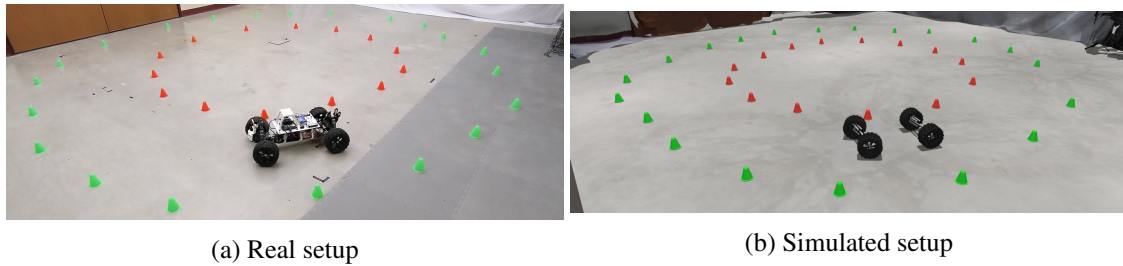
(b) Simulated setup

Figure 4: Real and simulated scenarios using the same cone locations measured by a motion capture system.

To further demonstrate the algorithms used by ART, we show examples of the intermediate output from the perception and planning stages. Perception results from simulation and reality are shown in Fig. 5, and results from the planning stage are shown in Fig. 6.

(a) Perception on a simulated image
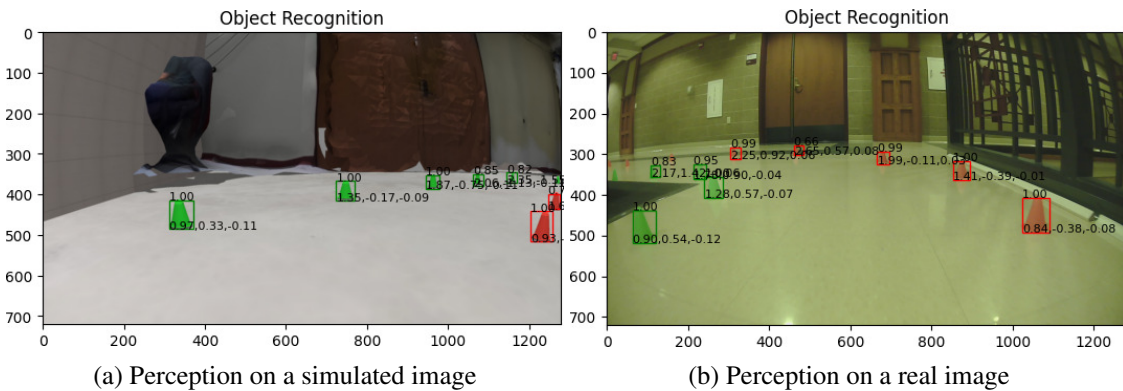
(b) Perception on a real image

Figure 5: Intermediate perception results from the autonomy stack shown for simulation and reality. Overlaid on the image are the detected bounding boxes, classes, confidence, and estimated 3D location relative to the vehicle.

5 CONCLUSION AND FUTURE WORK

This brief outlines an open-source Autonomy Research Testbed (ART) whose purpose is twofold: conduct research in autonomy for wheeled vehicles in on/off-road conditions; and investigate the sim-to-real gap in robotics – understand what causes it, and how it can be controlled. Looking ahead, we will equip V-1 with additional sensors to test their use both in autonomy algorithms and to help close the sim to real gap in Chrono. Utilizing the available motion capture system and its millimeter position tracking, a richer family of sensors will allow us to investigate better sensor models, as well as perception, state estimation, planning, and controls algorithms, and understand how inaccuracy in different components of the autonomy stack propagate downstream and cause
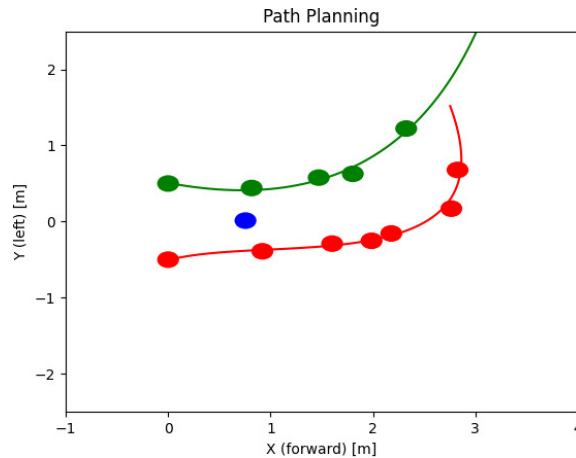
Figure 6: An example of intermediate results from the planning stage. Shown here are the locations of the cones in 2D (circles), the estimated curve of the boundary, and the target point used for control (blue).

the sim-to-real gap. With an increased understanding of the sensors and improved sensor models we will be able to take V-1 into off-road conditions to further build our understanding of the vehicle.

Finally, ATK can be used to facilitate autonomy algorithm development by allowing researchers to configure a custom set of containers to host their development, deployment, simulation, and visualization needs. For instance, this platform is assisting UW-Madison students in their SAE-sponsored AutoDrive Challenge II, and is currently being used in our lab to improve the Chrono sensor package.

ACKNOWLEDGMENT

REFERENCES

[1] Jakobi, N., Husbands, P., Harvey, I.: Noise and the reality gap: The use of simulation in evolutionary robotics. In: European Conference on Artificial Life, Springer (1995) 704–720

[2] Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., Abbeel, P.: Domain randomization for transferring deep neural networks from simulation to the real world. In: 2017 IEEE/RSJ international conference on intelligent robots and systems (IROS), IEEE (2017) 23–30

[3] Peng, X.B., Andrychowicz, M., Zaremba, W., Abbeel, P.: Sim-to-real transfer of robotic control with dynamics randomization. In: 2018 IEEE International Conference on Robotics and Automation (ICRA). (May 2018) 1–8

[4] Pinto, L., Davidson, J., Sukthankar, R., Gupta, A.: Robust adversarial reinforcement learning. In: International Conference on Machine Learning, PMLR (2017) 2817–2826

[5] Rajeswaran, A., Ghotra, S., Ravindran, B., Levine, S.: EPOpt: Learning robust neural network policies using model ensembles (2016)

[6] Farchy, A., Barrett, S., MacAlpine, P., Stone, P.: Humanoid robots learning to walk faster: From the real world to simulation and back. In: Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems. (2013) 39–46

[7] Chebotar, Y., Handa, A., Makoviychuk, V., Macklin, M., Issac, J., Ratliff, N., Fox, D.: Closing the sim-to-real loop: Adapting simulation randomization with real world experience. In: 2019 International Conference on Robotics and Automation (ICRA), IEEE (2019) 8973–8979

[8] Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A.Y.: ROS: an open-source Robot Operating System. In: ICRA workshop on open source software. Volume 3., Kobe, Japan (2009) 5

[9] Project Chrono: Chrono: An open source framework for the physics-based simulation of dynamic systems. `http://projectchrono.org` (2020) Accessed: 2020-03-03.

[10] Serban, R., Taylor, M., Negrut, D., Tasora, A.: Chrono::Vehicle template-based ground vehicle modeling and simulation. Intl. J. Veh. Performance **5**(1) (2019) 18–39

[11] Elmquist, A., Negrut, D.: Sensor modeling and simulation for evaluation of connected and autonomous vehicles. Technical Report TR-2020-07, Simulation-Based Engineering Laboratory, University of Wisconsin-Madison (2020)

[12] Tasora, A., Mangoni, D., Negrut, D., Serban, R., Jayakumar, P.: Deformable soil with adaptive level of detail for tracked and wheeled vehicles. International Journal of Vehicle Performance **5**(1) (2019) 60–76

[13] Hu, W., Zhou, Z., Chandler, S., Apostolopoulos, D., Kamrin, K., Serban, R., Negrut, D.: Traction control design for off-road mobility using an SPH-DAE co-simulation framework. Multibody System Dynamics **55** (2022) 165–188

[14] Taves, J., Elmquist, A., Young, A., Serban, R., Negrut, D.: Synchrono: A scalable, physics-based simulation platform for testing groups of autonomous vehicles and/or robots. In: Proceedings of 2020 International Conference on Intelligent Robots and Systems (IROS) – Las Vegas, USA. (2020)

[15] Project Chrono Development Team: PyChrono: A Python wrapper for the Chrono multi-physics library. `https://anaconda.org/projectchrono/pychrono` Accessed: 2023-01-14.

[16] Tasora, A., Serban, R., Mazhar, H., Pazouki, A., Melanz, D., Fleischmann, J., Taylor, M., Sugiyama, H., Negrut, D.: Chrono: An open source multi-physics dynamics engine. In Kozubek, T., ed.: High Performance Computing in Science and Engineering – Lecture Notes in Computer Science, Springer International Publishing (2016) 19–49

[17] Srinivasa, S.S., Lancaster, P., Michalove, J., Schmittle, M., Summers, C., Rockett, M., Smith, J.R., Chouhury, S., Mavrogiannis, C., Sadeghi, F.: MuSHR: A low-cost, open-source robotic racecar for education and research. CoRR **abs/1908.08031** (2019)

[18] Open-Source-Robotics-Foundation: A 3D multi-robot simulator with dynamics. `http://gazebosim.org/` Accessed: 2022-03-01.

[19] RACECAR Team: The racecar project. `https://racecar.mit.edu` (2022)

[20] RACECAR Team: Github repo, racecar project. `https://github.com/mit-racecar` (2022)

[21] Dziubiński, M.: The karr platform. `https://tinyurl.com/2p8s35rt` (2019)

[22] Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., Koltun, V.: CARLA: An open urban driving simulator. In: Proceedings of the 1st Annual Conference on Robot Learning. (2017) 1–16

[23] Donkey Car Community: The donkey car. `https://docs.donkeycar.com` (2022)

[24] Unity3D: Real-Time 3D Tools. `https://unity3d.com/` (2016) Accessed: 2022-12-28.

[25] NVIDIA: PhysX simulation engine. Available online at `http://developer.nvidia.com/object/physx.html` (2019)

[26] Orsolic, I.: Donkey car-based self-driving vehicle. `https://tinyurl.com/56wb2ue4` (2020)

[27] Czuba, D., Dandekar, R., Gordon, S., Stultz, E.: Parv: A 3d printed autonomous roboticvehicle platform. `https://digital.wpi.edu/pdfviewer/cj82kb092` (2021) Worcester Polytechnic Institute.

[28] de Azevedo, E.G., Jeanlys, A., Mercer, C., Mugabo, J., Reardon, E., Sel, T.: Modular package for autonomous driving (mpad). `https://digital.wpi.edu/downloads/pc289n23j` (2020) Worcester Polytechnic Institute.

[29] Sharma, S., White, J.: Self driving rc car. `https://github.com/sidroopdaska/SelfDrivingRCCar` (2021)

[30] SunFounder: PiCar. `https://tinyurl.com/4sabb5bk` (2021)

[31] EzBlock Studio: EzBlock. `https://ezblock.cc/` (2022)

[32] UW-Madison Simulation Based Engineering Laboratory: Autonomy Toolkit. `http://projects.sbel.org/autonomy-toolkit/` (2022)

[33] UW-Madison Simulation Based Engineering Laboratory: PyPI: Autonomy-ToolKit. `https://pypi.org/project/autonomy-toolkit/` (2022)

[34] Ren, S., He, K., Girshick, R., Sun, J.: Faster R-CNN: Towards real-time object detection with region proposal networks. IEEE transactions on pattern analysis and machine intelligence **39**(6) (2016) 1137–1149

[35] Howard, A., Sandler, M., Chen, B., Wang, W., Chen, L., Tan, M., Chu, G., Vasudevan, V., Zhu, Y., Pang, R., Adam, H., Le, Q.: Searching for mobilenetv3. In: 2019 IEEE/CVF International Conference on Computer Vision (ICCV), Los Alamitos, CA, USA, IEEE Computer Society (nov 2019) 1314–1324

[36] UW-Madison Simulation Based Engineering Laboratory: ART Supporting Video. `https://uwmadison.box.com/s/iypwo22vj26yqjqcprhc65trbe5vp8vz` (2022)