Runtime Monitoring on Real-World Embedded Systems in Simulator-Free Environments

Kai-Chun Chang, Aniruddha Joshi, Charles Cai
Department of Electrical Engineering and Computer Science, University of California Berkeley, Berkeley, USA
kaichunchang, aniruddhajoshi, charles_cai@berkeley.edu

ABSTRACT

In this work, we explore runtime monitoring on a real-world embedded system. We consider a Simplex structure and focus on the training of the monitor. As the simulators are not always available in real-world applications, large sim-to-real gaps occur and thus degrades the quality of the monitor. To address this challenge, we propose a two-stage framework. In the first stage, we construct a simulation environment, generate an RL controller and a safe controller, and train the monitor in the simulator. In the second stage, the monitor and the controllers are deployed to the real-world environment. Then, we can collect training data from the real world and polish the monitor accordingly. In the experiments and demos, the effectiveness of our system is validated.

KEYWORDS

Runtime monitoring, embedded system, sim-to-real gap, reinforcement learning

1 INTRODUCTION

Machine learning (ML) components are increasingly integrated into autonomous systems to improve perception, planning, and control. However, achieving a balance between safety and performance can be challenging. To address this, the use of runtime monitors is growing in popularity to ensure system safety, particularly learning-based runtime monitors for cyber-physical systems.

In monitor training, simulators often play an important role. However, when it comes to real-world settings, simulators may not always be available. Creating simulators relies on assumptions and can be time-consuming, leading to sim-to-real gaps. To address this challenge, we propose gathering real-world data and training the monitor accordingly. This approach demonstrates the feasibility of developing high-quality monitors for real-world applications even if simulators are not perfect.

2 PROBLEM SETTINGS

2.1 Problem Formulation

We consider the Simplex architecture [6], as illustrated in Figure 1. In a Simplex architecture, two kinds of controllers are integrated. An advanced controller aims to maximize the performance of the system with less concern about safety. On the other hand, a safe controller tries to satisfy the safety specification of the system and thus sacrifices performance a lot. To balance safety and performance, a monitor is adopted to determine which controller to use at each time step. For example, if the system is about to run into unsafe regions, the monitor will select the safe controller to ensure safety. Otherwise, the monitor will choose the advanced controller to profit from its aggressiveness.

Kai-Chun Chang, Aniruddha Joshi, and Charles Cai contributed equally.

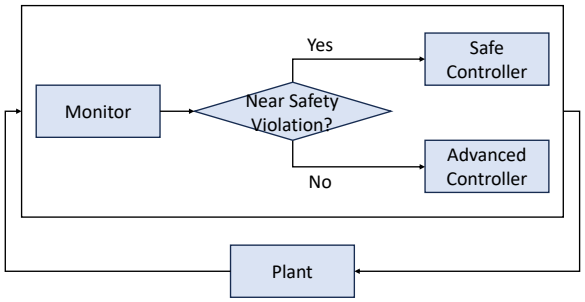

Figure 1: The Simplex architecture.

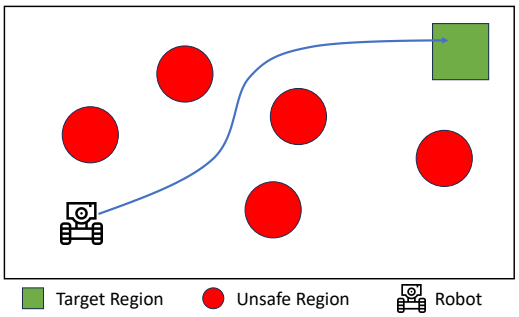

Figure 2: The robot navigation scenario.

In this work, we focus on the learning of the monitor in the Simplex architecture. Formally, our problem can be defined as following:

- **Inputs:** A safe controller and an advanced controller.
- **Outputs:** A learning-based monitor.
- **Objective:** The goal of the monitor is to optimize the overall performance and guarantee the safety for the system.

The details regarding the generation of the input safe controller and the advanced controller are described in Section 3.

2.2 Our Scenario

In this work, we consider a robot navigation task, as depicted in Figure 2. The objective for the robot is to reach a designated target region (the green region) while avoiding unsafe regions (the red circles). The locations of these regions are randomly established at the beginning. In this task, the performance of our framework can be defined as the time taken for the robot to reach the target region. On the other hand, entering an unsafe region can be viewed as a safety violation.

3 OUR FRAMEWORK

3.1 Overview of Our Framework

Figure 3 shows our framework. In the simulator, we first construct the simulation environment and make it aligned with the real-world environment. After that, we can train our advanced controller

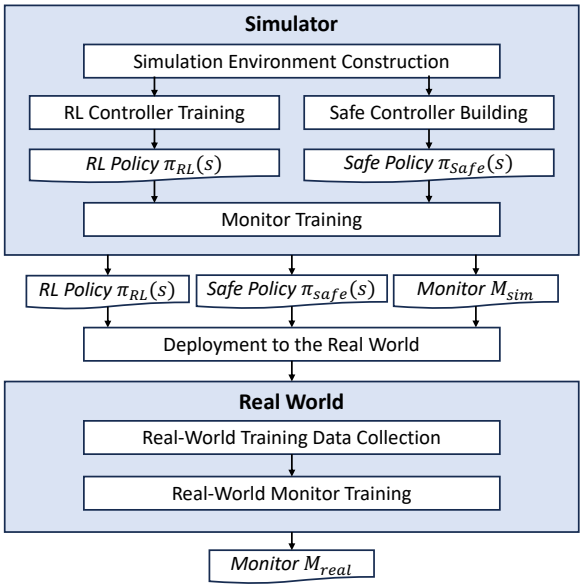

Figure 3: Our framework.

(we adopt an RL-based controller in this work) and build our safe controller within the simulation environment. Once the RL policy $\pi_{RL}(s)$ and the safe policy $\pi_{safe}(s)$ are derived, we can generate a bunch of trajectories from the controllers as the training data for the monitor and perform monitor learning accordingly.

After the monitor M_{sim} is generated, we can deploy the monitor along with the two controllers $\pi_{RL}(s)$ and $\pi_{safe}(s)$ to the real world. Then, we can collect trajectories directly from the real world as new training data, which can be leveraged to polish the monitor. Finally, our framework generates a monitor M_{real} that is based on not only the simulation data but also the real-world trajectories.

3.2 Simulation Environment

We adopt Webots [1][4] as our simulator, as illustrated in Figure 4. This platform provides a user-friendly interface for simulating robotic scenarios, enabling the seamless integration of various obstacles (depicted as red circles) and a designated target region (illustrated by the green rectangle). Webots offers a diverse selection of robot models, facilitating the replication of our physical robot within the virtual environment. Moreover, the simulator allows for the random initialization of the positions and orientations of robots, obstacles, and target regions at the beginning of each simulation. This feature streamlines the process of training RL controllers, gathering trajectories for monitor learning, and conducting verification for our system.

3.3 Safe Controller

To realize our safe controller, we adopt the rapidly exploring random tree (RRT) algorithm [2, 3], which is a one of the most common robot path planning algorithms. The RRT algorithm starts growing a tree from the initial position. In each iteration, it randomly samples a new point in the searching space and connects it to the nearest point in the tree. If there exists no obstacle on the connection path, the new point is included to the tree. The algorithm continually grows the tree until a point within the target region or a searching limit is reached. Figure 5 illustrates an example of the RRT algorithm. The starting point is $(0, 0)$, and the target point is $(6, 10)$. The blue

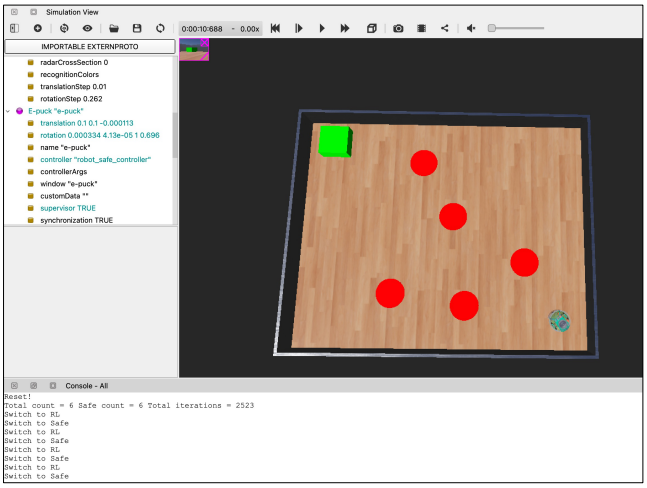

Figure 4: The Webots simulator.

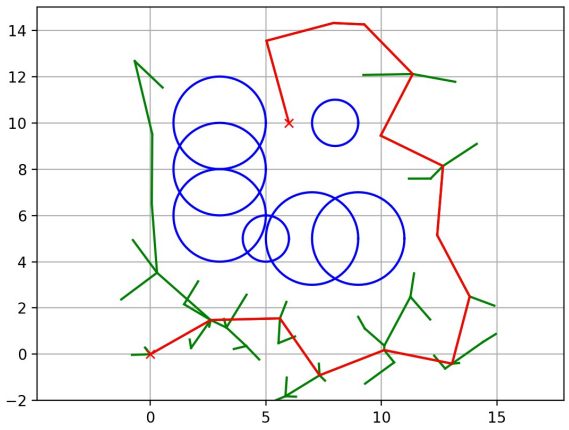

Figure 5: An example of the RRT algorithm.

circles represents the obstacles. The line segments compose the tree maintained during the path searching, where the red ones denote the path found. Although the RRT algorithm is not optimal, it's relatively fast and easy to implement, making it suitable for real-world embedded system applications. In our experiments, we utilize the PythonRobotics [5] library to implement the RRT algorithm.

3.4 Reinforcement Learning-based Controller

We first used the PPO algorithm in the simulator. where we were trying to actuate the wheels of the robot using inputs as position of the robot, orientation of robot, position of the target, and the position of the unsafe regions, but we were unable to acess orientation of robot through lidar, as there were specific angles which created problems . So we switched to having inputs as the position of the robot, the position of the target, the position of the unsafe regions, and the radius of the unsafe regions. The output space here is the x and y coordinates of the next waypoint. We trained the RL controller using DDPG algorithm with an available implementation of deepbots. We randomized the initial position of the robot and the position of the unsafe regions in the environment. We get the following plot of returns, we observe that the dips are because after changing the initial position, and the rewards increase when the inital state is unchanged.

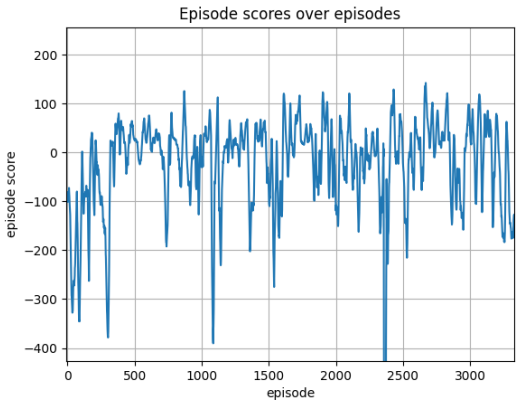

Figure 6: Caption

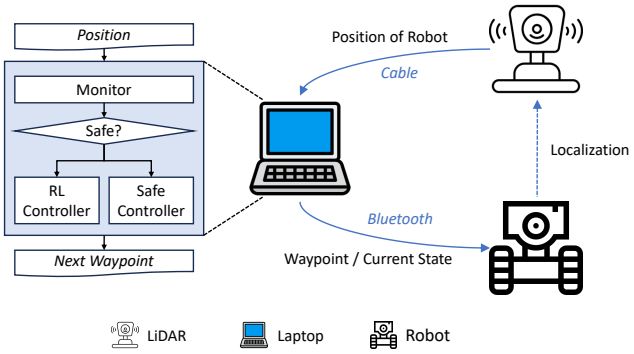

Figure 7: The hardware configuration.

The networks of the actor and the critic are with 3 hidden layers where the first is contains 30 the second contains 50 and the third contains 30 neurons.

3.5 Monitor

We trained the monitor using supervised learning, where we first collect data using the RL controller in the simulator environment, then we process the data such that we add a column that determines whether there is violation within 5 timesteps, and train a network on this data.

4 HARDWARE CONFIGURATION

4.1 Overview of the Real-World Environment

Figure 7 depicts our hardware configuration. We use a LiDAR for robot localization. The LiDAR transmits robot positions to the laptop via a cable. Subsequently, the laptop performs path planning for the robot using our generated monitor and controllers. If the monitor identifies an imminent safety violation, it switches to the safe controller and determines a safe waypoint. Otherwise, the RL controller is engaged to compute the next waypoint. Once generated, the laptop transmits the waypoint, along with the robot's position, to the robot via Bluetooth. The robot, in turn, autonomously executes low-level control. This involves initial rotation to align with the new waypoint, leveraging data from its built-in gyroscope's yaw angle. Following alignment, the robot propels itself towards the designated waypoint until the distance falls below a predefined threshold.

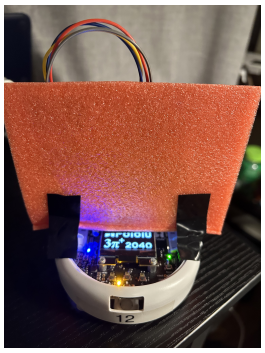

Figure 8: The Pololu Robot.

4.2 Robot

Our selected robotic platform for real-world deployment is the Pololu RP2040 3pi+, serving as the primary agent. We leverage the graphical user interface (GUI) implemented through MicroPython directly on the robot. To enhance the efficacy of LiDAR-based localization and orientation detection, a flat board has been affixed atop the robot, as whown in Figure 8. This augmentation facilitates the LiDAR system in efficiently detecting and determining the robot's spatial location and facing orientation.

4.3 LiDAR and Localization

We employ YD-LiDAR as the primary sensor for ascertaining the precise location information of the robot. Our data acquisition involves obtaining feedback in the form of a tuple (distance, angle) representing the measurements of the two edges of a flat board affixed to the robot. Subsequently, we utilize this data to compute the accurate positions of the robot.

4.4 Wireless Communication

We utilize the HC-05 module for Bluetooth connection, serving as the receiving component integrated into the robot. Data transmission occurs through UART communication via GPIO Pins. During data transfer to the robot, a comprehensive string is transmitted, encompassing essential information such as the current location coordinates, waypoint coordinates, and a flag indicating whether the monitor is switching between controllers.

5 EXPERIMENTAL RESULTS

5.1 Effectiveness of Our Framework

As our hardware setting is not quite stable now, we have not been able to perform real-world data collection and monitor training. Thus, the following experimental results are all based on the monitor trained in the simulator, i.e., M_{sim}.

Table 1 shows the performance of the controllers and the monitor in the simulation environment. We consider three settings: only safe controller, only RL controller, and the Simplex architecture (i.e., monitor with the two controllers). For each settings, we perform 1000 simulations and collect the simulation results, where the positions of the obstacles are randomized in each simulation. The average reaching time is the average number of timesteps the robot needs to reach the target among all the simulations. The safe rate denotes the ratio of the simulations where the robot doesn't enter any obstacle before it achieves the target.

Table 1: Our experimental results in the simulation environment.

Settings	Average Reaching Time	Safe Rate
Only Safe Controller	440.45	95.0%
Only RL Controller	**301.29**	19.9%
Simplex Architecture	425.38	**95.3%**

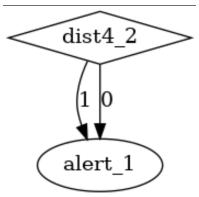

Figure 9: The decision diagram.

As expected, the safe rate of the safe controller is very high, but it's also the most conservative one and thus takes the longest average reaching time. On the other hand, the RL controller is much more aggressive and efficient, while it suffers a lot safety violations. Our Simplex architecture maintains a similar safe rate as the safe controller. Moreover, it also improves the average reaching time compared to the safe controller, indicating that it benefits from the aggressiveness of the RL controller.

5.2 Analysis of the Monitor

In our experiment, we have observed that the monitor selects the safe controller most of the time, thus the average reaching time of our Simplex architecture only improves a subtle margin from the safe controller. To further analyze the behavior of the monitor, we utilize the tool developed in [7]. We get only one decision diagram here. We used nodes of the decision diagram as distance between robot and target $\leq$ radius, and 5 nodes for distance between robot and the five unsafe regions. It gives only this decision diagram.

5.3 Real-World Experimental Results

The video in this [link] and Figure 10 show an example of the behavior of the RL controller in the real world. As the experimental results in the simulator, it aggressively heads toward the target region, but it also violates the safety constraints during its way. On the other hand, the video in this [link] and Figure 11 show the behavior of our monitor with the two controllers, which safely direct the robot toward the target region.

6 CONCLUSION

In this work, we explore the Simplex architecture on a robot navigation scenario. We generate a robust safe controller and an efficient RL controller, and we also develop a learning-based monitor. In the simulation environment, our Simplex system can maintains a similar safe rate as the safe controller, while improving the robot reaching time simultaneously. We also successfully deploy our system onto the Pololu robot and perform runtime monitoring in the real-world environment. The monitor shows its effectiveness in the real world under some instances.

Future work includes realizing the real-world training data collection and the real-world monitor training flow, so that we can leverage the real-world data to polish our learning-based monitor. In addition, we can introduce falsification techniques into our flow, generate falsified cases for our monitors, and improve the monitor accordingly.

REFERENCES

[1] Cyberbotics Ltd. 1998. Webots. http://www.cyberbotics.com. Open-source Mobile Robot Simulation Software.
[2] Sertac Karaman and Emilio Frazzoli. 2010. Incremental sampling-based algorithms for optimal motion planning. *Robotics Science and Systems VI* 104, 2 (2010), 267–274.
[3] Sertac Karaman and Emilio Frazzoli. 2011. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research* 30, 7 (2011), 846–894.
[4] O. Michel. 2004. Webots: Professional Mobile Robot Simulation. *Journal of Advanced Robotics Systems* 1, 1 (2004), 39–42. http://www.ars-journal.com/International-Journal-of-Advanced-Robotic-Systems/Volume-1/39-42.pdf
[5] Atsushi Sakai, Daniel Ingram, Joseph Dinius, Karan Chawla, Antonin Raffin, and Alexis Paques. 2018. PythonRobotics: a Python code collection of robotics algorithms. arXiv:1808.10703 [cs.RO]
[6] Lui Sha, Ragunathan Rajkumar, and Michael Gagliardi. 1996. Evolving dependable real-time systems. In *1996 IEEE Aerospace Applications Conference. Proceedings*, Vol. 1. IEEE, 335–346.
[7] Hazem Torfah, Shetal Shah, Supratik Chakraborty, S Akshay, and Sanjit A Seshia. 2021. *Synthesizing pareto-optimal interpretations for black-box models.* IEEE.

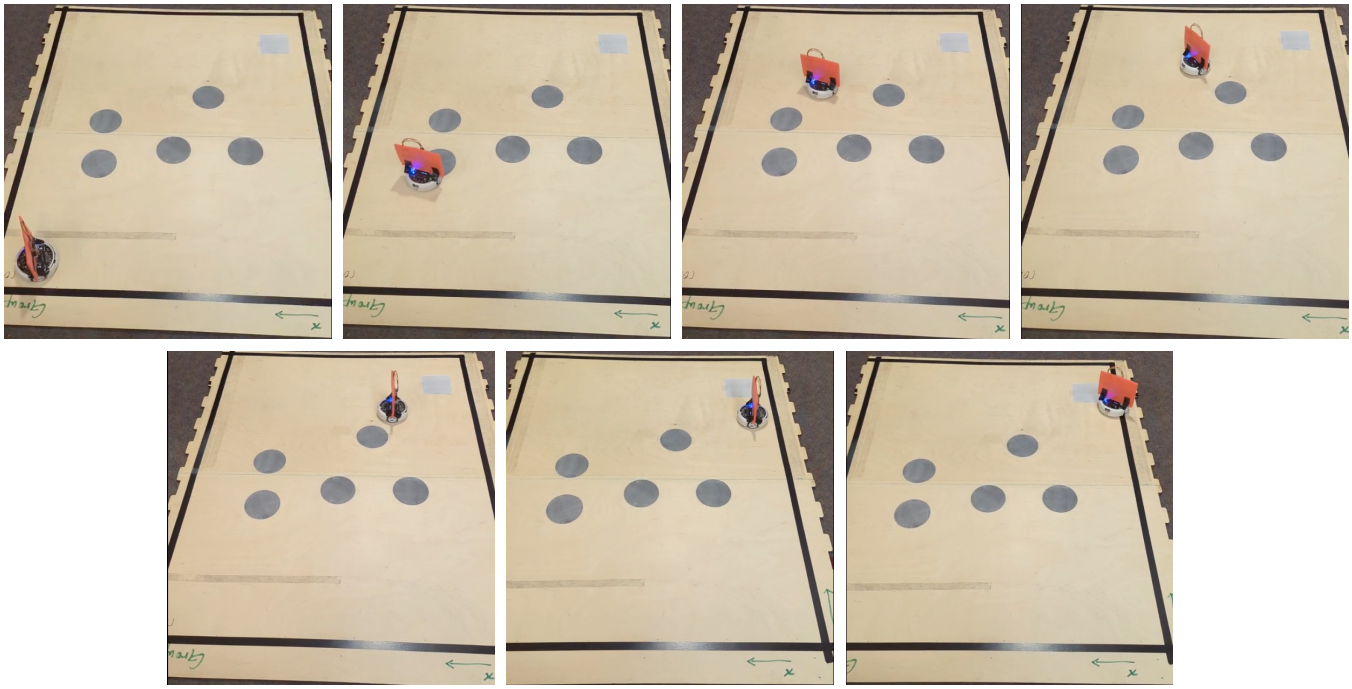

Figure 10: The real-world experimental result of the RL controller.

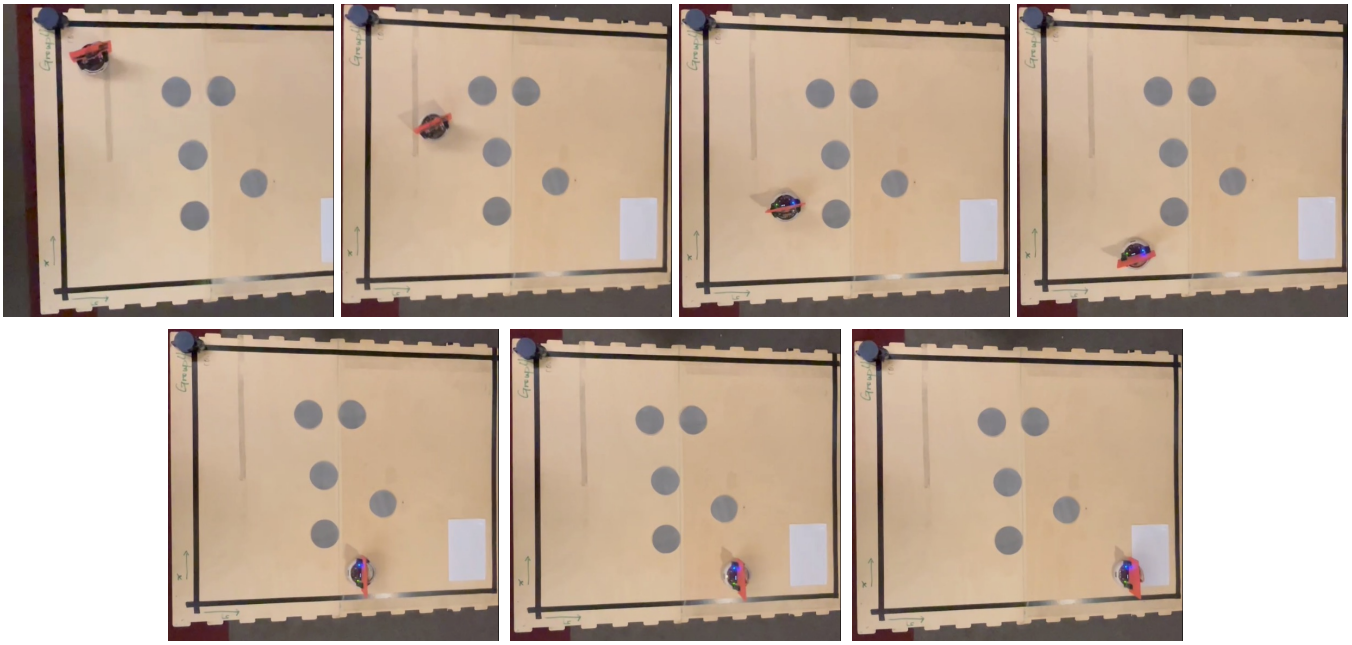

Figure 11: The real-world experimental result of our Simplex architecture.