SPARK: Safe Protective and Assistive Robot Kit

Yifan Sun, Rui Chen, Kai S. Yun, Yikuan Fang, Sebin Jung
Feihan Li, Bowei Li, Weiye Zhao, and Changliu Liu

Robotics Institute, Carnegie Mellon University,
{yifansu2,ruic3,sirkhooy,yikuanf,sebinj,feihanl,sboweili,weiyezha,cliu6}
@andrew.cmu.edu

Abstract: This paper presents the Safe Protective and Assistive Robot Kit (SPARK), a benchmark and toolbox for enhancing safety in humanoid autonomy and teleoperation. Humanoid robots, due to their complex physical structures and interaction with dynamic environments, demand robust safety solutions. SPARK provides a modular safe control framework with representative algorithms, allowing users to configure safety criteria and trade-offs between safety and performance. It includes simulation benchmarks across diverse tasks and supports rapid deployment on real hardware, including the Unitree G1 humanoid robot. SPARK also integrates with external sensors like Apple Vision Pro and Motion Capture Systems. Open-source code is available at https://github.com/intelligent-control-lab/spark.

Keywords: Humanoid robots, safe set algorithm, teleoperation, VR, human robot interaction

1. INTRODUCTION

Safety is a core requirement for robotic systems in both industrial and everyday settings. Safe control ensures task performance while respecting safety constraints, reducing risks and harm. Many approaches cast safe control as a constrained optimization problem, solved either analytically (e.g., via quadratic programs) or with data-driven methods. However, model-free methods often lack safety guarantees in high-dimensional systems, motivating a focus on **model-based safe control**.

In simple scenarios—such as imposing a speed limit on a differential-drive robot—the synthesis of a safe controller is relatively straightforward. However, complexity arises when changing the *robot*, *task*, or *environment*. For example, the safety mechanism for a humanoid navigating open ground is similar to that of a wheel based mobile robot. However, a humanoid performing package delivery must account for both mobility safety and manipulation safety simultaneously. For more complex tasks like assembly, the robot must operate safely as a dual-arm manipulator, while bipedal locomotion requires full-body safety considerations similar to legged robots. These diverse and task-specific safety requirements make case-by-case controller design inefficient and heavily reliant on expert knowledge.

To address this, we introduce the **Safe Protective and Assistive Robot Kit** (SPARK[1]): a modular framework for scalable and efficient synthesis, benchmarking, and deployment of safe controllers. Key features of SPARK include:

Composable. A flexible Python framework for building safe-control scenarios across diverse robot types (manipulators, mobile bases, etc), tasks (manipulation, locomotion, etc), and operating modes (autonomous, teleoperated, etc).

Extensible. A clear template that enables users to integrate their own robots and implement new control algorithms with minimal effort.

Deployable. ROS-based interfaces that connect seamlessly to real robots and AR headsets for real-world deployment.

2. SPARK'S FRAMEWORK FOR TESTING, BENCHMARKING, DEVELOPMENT AND DEPLOYMENT

Figure 1 presents SPARK, a modular collection of Python class templates designed to streamline the synthesis, testing, benchmarking, development, and deployment of safe controllers for robotic systems.

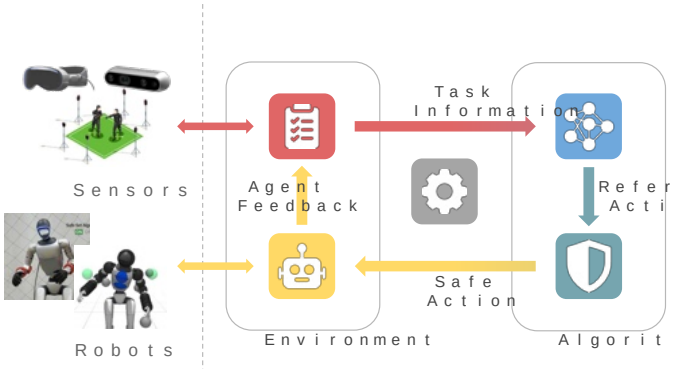

Fig. 1. SPARK system framework.

The SPARK framework is organized around decomposing system components into distinct modules that efficiently

[1] The complete paper is available at https://arxiv.org/pdf/2502.03132.

manage system states, measurements, objectives, dynamics, and controllers. This modular structure enables composable, extensible, and deployable safe controller design.

We first introduce the **Configuration** module, which supports and connects all other modules.

Configuration Module defines the *system dynamics*, encapsulating robot-specific information such as degrees of freedom, motor interfaces, and system models.

The SPARK **Environment** consists of the **Agent** and **Task** modules, serving as the "front-end" responsible for obtaining *system measurements* and updating the *system state* within the context of *system objectives*.

Agent Module interfaces with either the physical robot or its simulation. It receives control commands from the controller and applies them to the robot's actuators, modifying the robot's *system state*.

Task Module provides task-specific information, including *system measurements* and *system objectives*. It defines mission goals (e.g., reaching a target, obstacle avoidance) and bridges the current robot state (via the **Agent**) with environmental information (e.g., obstacle locations).

The"back-end" of the framework is represented by the **Algorithm**, which unifies the **Policy** and **Safety** modules. Here, the *system controller* is split into two submodules:

Policy Module processes task information to generate reference control actions aimed at achieving performance goals, without explicitly considering safety constraints.

Safety Module modifies the control actions produced by the **Policy** to enforce safety constraints while preserving task performance as much as possible.

Both **Policy** and **Safety** modules support model-based and data-driven designs, maintaining SPARK's core principles of composability and extensibility.

3. SPARK SUITE OPTIONS

In addition to safe control modules, SPARK offers a comprehensive testing suite to evaluate the performance of various safe controllers, as shown in 2. Leveraging SPARK's modularity, users can configure robot models, agents, tasks, policies, and safety algorithms to create diverse benchmark scenarios.

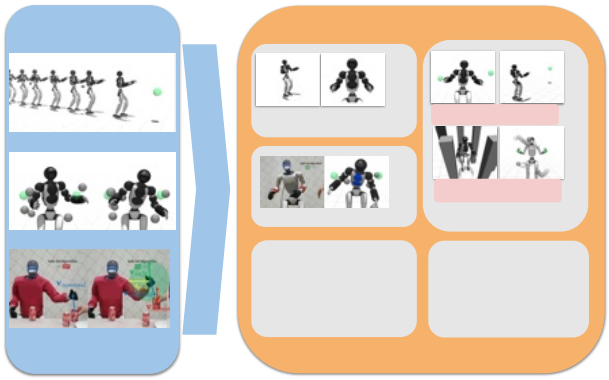

Fig. 2. SPARK Suite Options.

3.1 Configuration Options

Given the complexity of humanoid systems, SPARK provides predefined configurations based on the Unitree G1 humanoid robot, simulated via MuJoCo XML files. The testing suite includes four robot configurations: fixed-base single and dual arm manipulators that target safe manipulation, a wheeled mobile robot for safe mobile-manipulation, and a bipedal robot that tackles the most demanding loco-manipulation scenarios—collectively providing coverage from simple manipulation to complex, integrated locomotion and manipulation tasks. Additionally, users can adjust the order of the dynamic system to obtain more realistic configurations.

3.2 Agent Options

SPARK supports both simulation agents and real robot agents based on Unitree G1 configurations. The G1 robot, which has 29 physical degrees of freedom (DOFs), is simplified in different MuJoCo XML files to represent various robot configurations.

3.3 Task Options

SPARK's testing suite provides predesigned tasks to benchmark safe controllers across various scenarios.

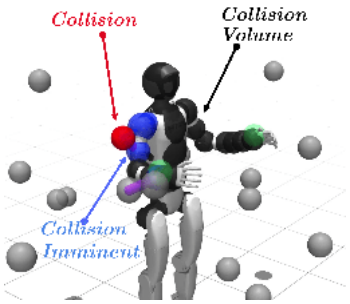

Fig. 3. SPARK whole body task environment.

Task Objectives The framework supports two types of task goals: 1) Arm Goals, in which each hand must reach designated static or dynamic 3D targets, and 2) Base Goals, where the robot navigates to specified 2D positions while avoiding obstacles. Goal motion can be set to static or dynamic, and all goals are visualized as green spheres. in 3.

Task Constraints With SPARK 's built-in interface, users can tailor task complexity by selecting obstacle shape (sphere or box), motion (static or dynamic), and quantity.

Task Interfaces The task interface supports multiple methods for specifying tasks and constraints in real time. For example, simulation agents enable real-time obstacle manipulation through keyboard control. In contrast, real robot agents integrate Apple Vision Pro to track human hands as dynamic obstacles.

3.4 Policy Options

SPARK provides PID control for locomotion and combined PID + inverse kinematics (IK) for manipulation tasks.

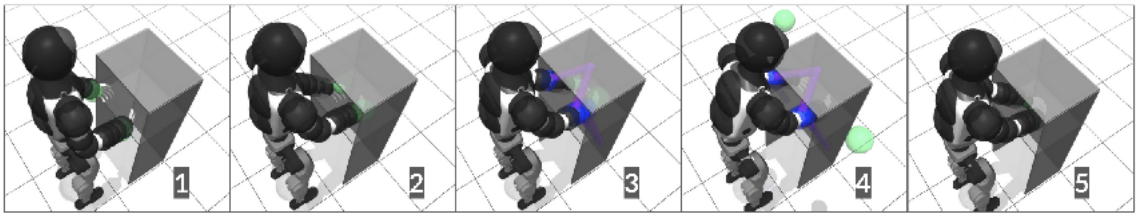

Fig. 4. 4.2 The first two illustrate how the robot's hands successfully reach into a confined cabinet under user teleoperation.

Fig. 5. 4.3 Limb-level collision avoidance with static humanoid reference pose.

Users can also integrate custom policies, including data-driven methods. While SPARK focuses on model-based safe controllers, future work will explore broader control policies.

3.5 Safety Options

Consider the control-affine system

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) + \mathbf{g}(\mathbf{x})\,\mathbf{u}, \qquad \mathbf{u} \in \mathcal{U},$$

where $\mathbf{x}$ is the state, $\mathbf{u}$ the control input, and $\mathbf{U}$ the admissible control set. The safe control problem seeks a safe policy that generates safe control $\mathbf{u}_{\text{safe}}$, keeping the system state trajectories inside a prescribed safe set $\mathbf{X}_{\text{s}}$ while tracking the reference control.

In SPARK, users can choose from a suite of baseline safe-control algorithms. Each algorithm employs an energy function $\phi(\mathbf{x})$ to measure the potential to stay in the prescribed safe set in the future, where states satisfying $\phi(\mathbf{x}) > 0$ are considered potentially *unsafe*.

The first family of methods guarantees safety by solving the constrained optimization problem

$$\begin{aligned} \min_{\mathbf{u}} \quad & \|\mathbf{u} - \mathbf{u}_{\text{ref}}\|^2_{\mathbf{Q}_\mathbf{u}} \\ \text{s.t.} \quad & \dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) + \mathbf{g}(\mathbf{x})\mathbf{u} \\ & \text{Safety Constraint} \\ & \mathbf{u} \in \mathbf{U} \end{aligned} \tag{1}$$

where the Safety Constraint can be formulated based on $\phi(\mathbf{x})$. Among the three representative methods:

Safe Set Algorithm (SSA) Liu and Tomizuka [2014] enforces $\dot{\phi}(\mathbf{x}, \mathbf{u}) \leq -\eta$ whenever $\phi(\mathbf{x}) \geq 0$, where $\eta > 0$ is a user-defined margin.

Control Barrier Functions (CBF) Ames et al. [2019] impose the condition $\dot{\phi}(\mathbf{x}, \mathbf{u}) \leq -\lambda\,\phi(\mathbf{x})$, with a class-$\mathcal{K}$ gain $\lambda > 0$.

Sublevel Safe Set Algorithm (SSS) Wei and Liu [2019] combines the above ideas, applying the CBF-style bound $\dot{\phi}(\mathbf{x}, \mathbf{u}) \leq -\lambda\,\phi(\mathbf{x})$ only when $\phi(\mathbf{x}) \geq 0$.

In addition to the optimization-based baselines, SPARK offers two projection methods that modify the reference input directly, avoiding solving constrained optimizations:

Potential Field Method(PFM) Khatib [1986]. Safety is enforced in Cartesian space via an energy function $\tilde{\phi}(\mathbf{c}_r)$ defined over the Cartesian state $\mathbf{c}_r$, yielding

$$\mathbf{u}_c = \mathbf{u}_{\text{c-ref}} \; - \; c_1\,\nabla\tilde{\phi}, \qquad \text{if } \tilde{\phi}(\mathbf{c}_r) \geq 0,$$

after which the Cartesian command $\mathbf{u}_c$ is mapped back to joint space.

Sliding Mode Algorithm(SMA) Gracia et al. [2013]. The reference joint command is projected in joint space as

$$\mathbf{u} = \mathbf{u}_{\text{ref}} \; - \; c_2\,\mathbf{L}_\mathbf{g}\phi^\top, \qquad \text{if } \phi(\mathbf{x}) \geq 0,$$

where $\mathbf{L}_\mathbf{g}\phi_{\max}$ is the Lie derivative of the energy function along $\mathbf{g}$. c_1 and c_2 are positive constants.

4. USE CASES

4.1 Use Case 1: Benchmarking Safe Control Algorithms

To demonstrate SPARK's capability as a benchmarking toolbox for safe control, we utilize it to systematically evaluate various algorithms under different constraints and objectives. By leveraging SPARK's suite options, we ensure fair comparisons and gain insights into algorithmic performance, safety-efficiency trade-offs, and task complexity effects. Comprehensive experiments conducted using the benchmark are presented in the full version of the paper.

4.2 Use Case 2: Safe Teleoperation With Simulated Robot

To present a user scenario where teleoperation is performed in simulation to collect human demonstration data without requiring real hardware, we configure the robot with `G1fixedBase` and use the simulation agent in SPARK. The **Task** module defines a cabinet as an obstacle, with teleoperation targets provided via an external Apple Vision Pro input.

Fig. 6. 4.3 Limb-level collision avoidance with dynamic humanoid reference poses.

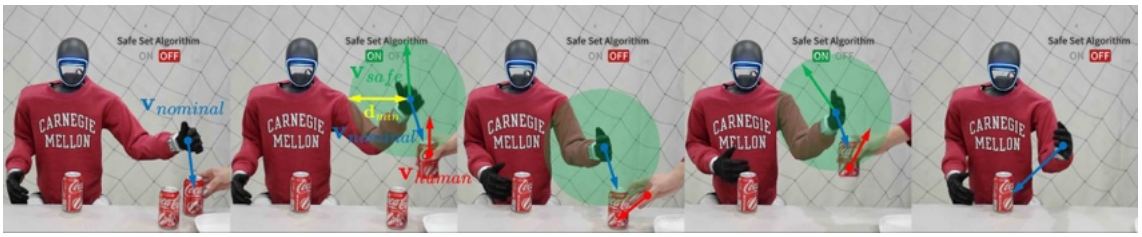

Fig. 7. 4.4 Limb-level collision avoidance with teleoperation commands.

As shown in the first two images of 4, the robot's hands maneuver into the cabinet under user control. In the fourth image, when the green target spheres move outside the cabinet, the robot's hands remain safely inside, demonstrating effective constraint enforcement.

4.3 Use Case 3: Safe Autonomy on Real Robot

Building on the benchmark use case in 4.1, we replaced the simulation agent with the real G1 SDK, allowing direct deployment of previously safe control algorithms.

We first evaluated a static task where the robot maintains a fixed position while avoiding human users. As shown in 5, when a human hand approaches within a minimum safe distance $d_{\min}$, the robot actively moves away. Once the human moves beyond $d_{\min}$, the robot resumes its nominal behavior, maintaining the target static pose.

Next, we assessed dynamic target tracking by commanding the robot's right hand to follow a circular trajectory while ensuring collision avoidance. Unlike the static case, the nominal controller now tracks a moving target $\mathbf{x}^R_{\text{target}}$. As shown in 6, the robot tracks the reference trajectory when safe and actively adjusts its waist and arms to avoid collisions when humans intrude into the $d_{\min}$ region, prioritizing safety over nominal control.

4.4 Use Case 4: Safe Teleoperation With Real Robot

This section evaluates the SPARK safe controller under "Safe Teleoperation" where the nominal controller's target $\mathbf{x}^R_{\text{target}}$ is generated in real time by a teleoperator, introducing greater unpredictability. To implement this, we modified the **Task** module to set the operator's hand positions as goals, while treating other humans as dynamic obstacles.

We tested a realistic scenario where the robot retrieves objects from a table. As shown in 7, when a human reaches for the same object, the safe controller overrides the teleoperation command to avoid collisions, prioritizing safety over task execution and protecting both the human and the robot.

5. DISCUSSION AND CONCLUSION

In this paper, we introduced SPARK, a comprehensive benchmark for advancing safe humanoid autonomy and teleoperation. We presented its modular safe control framework, core algorithms, and simulation environment with diverse benchmark tasks.

SPARK enables configurable trade-offs between safety and performance and supports broad task, hardware, and customization needs through accessible APIs. By providing robust baselines and practical deployment tools, SPARK aims to accelerate humanoid robotics development while ensuring hardware and environmental safety.

6. ACKNOWLEDGEMENT

This work is supported by the National Science Foundation under grant No. 2144489.

REFERENCES

Ames, A.D., Coogan, S., Egerstedt, M., Notomista, G., Sreenath, K., and Tabuada, P. (2019). Control barrier functions: Theory and applications. In *2019 18th European control conference (ECC)*, 3420–3431. IEEE.

Gracia, L., Garelli, F., and Sala, A. (2013). Reactive sliding-mode algorithm for collision avoidance in robotic systems. *IEEE Transactions on Control Systems Technology*, 21(6), 2391–2399.

Khatib, O. (1986). Real-time obstacle avoidance for manipulators and mobile robots. *The international journal of robotics research*, 5(1), 90–98.

Liu, C. and Tomizuka, M. (2014). Control in a safe set: Addressing safety in human-robot interactions. In *2014 ASME Dynamic Systems and Control Conference*. ASME.

Wei, T. and Liu, C. (2019). Safe control algorithms using energy functions: A unified framework, benchmark, and new directions. In *2019 IEEE 58th Conference on Decision and Control (CDC)*, 238–243.