TritonBots Team Description Paper for RoboCup 2026

Raymond Rada, Lukas Cao, Pedro Pinela, Akhil Ram Shankar, Minghan Wu, Nikitha Maderamitla, Yash Tandon, Adnan Barwaniwala, Evan Chou, Alex Meng, Wing Huang, Matvey Matsko, Chitraansh Chaudhary, Areeba Balkhi, Sihan Guo, Gabriel Haresco, AJ Jones, Diego Lechuga, Rafael Coyazo, and Chen Yifei

Institute of Electrical and Electronics Engineers (IEEE),
University of California San Diego (UC San Diego),
La Jolla, CA, USA
robocup@ieeeatucsd.org
https://ieeeatucsd.org/projects/robocup

Abstract. This paper will review the v2026 hardware and software changes to the TritonBots RoboCup Small Size League team at the University of California, San Diego. The team intends to compete at RoboCup 2026 in Incheon, South Korea. Changes and new developments to the electronics, mechanical design, AI systems and control systems will be discussed in detail, highlighting the improvements and progression the team has made over the course of the RoboCup 2026 season.

1 Introduction

This year the team has developed the first iteration of the robot with full competition capabilities. Due to team management, leadership issues, poor work quality, and subpar documentation past designs have failed to carry over from previous years. Robot design has been overhauled for more custom hardware that builds on work from other teams' research and development in the league.

A major outcome of this year's work is custom PCB design and physical analysis of the robot's functionality for mechanical design. This allowed for reliable operation of our kicker and dribbler. The software for making robot decisions has shifted from traditional programming to a machine learning approach. Following the software shift, the communication protocol to microcontrollers has been simplified, producing a more robust robot hardware control system.

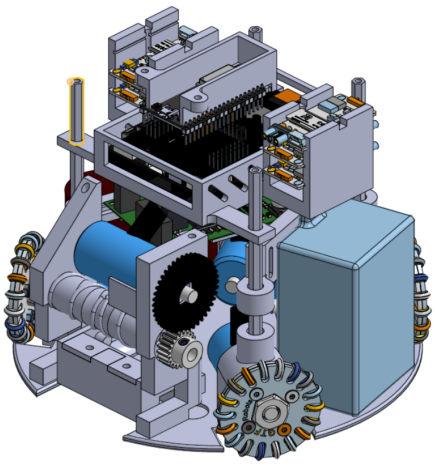

Fig. 1. CAD rendering of v2026 MkII robot. Cover is transparent for view of components.

Table 1. Robot Hardware Specifications

Robot Version	**MkII**	**MkIII**
Dimensions	170 mm × 150 mm	170 mm × 150 mm
Total weight	1722 g	TBD
Max. ball coverage	Variable (adaptable mechanism height)	9.6 mm
Wheel diameter	50 mm	50 mm
Drivetrain motor	M2006 P36 BLDC Motor	EC-45 flat 50 W
Gear ratio (drivetrain)	1:1	1:1
Roller diameter	18.5 mm	17 mm
Dribbler motor	M2006 P36 BLDC Motor	TBD
Gear ratio (dribbler)	2:1	1:1
Kicker charge	2700 μF @ 180 V (43.7 J)	2700 μF @ 240 V (77.8 J)
Microcontroller (Motor)	STM32F427	STM32F427
Microcontroller (Comms)	ESP32-D0WD	ESP32-S2-SOLO2
Battery	Ovonic 6s1p 1550 mAh	Ovonic 6s1p 1550 mAh

2 Mechanical Design

2.1 Overview

The team designed hardware for two development platforms. MkII was designed primarily to immediately address v2025 issues [1]. Focus was on improving mounting and housing of electronic components, dribbling efficiency, ease of assembly, and structural integrity. MkIII was designed to prepare our team for competition. Kicking, passing, dribbling, and movement is refined for new hardware and chipping capability is in development. The MkII currently uses the Robomaster M2006 P36 Brushless DC Gear Motor for the drivetrain and dribbler used in previous seasons for time and cost efficiency.

2.2 Chassis/Drivetrain

The second version of the baseplate possesses a similar cutoff profile to last year's version, with different hole positioning. The support for the electronic housing and the robot coverage is now provided, not directly by attachment to the baseplate, but rather by pins on top of the motor mounts. This reduces the deformation of the base plate, as less load and moment is applied to the baseplate and more towards the wheels.

2.3 Kicker

MKII makes use of the same overall mechanism previously used in the 2024-2025 season, using a linear guide rail and bearings to constrain the plunger to linear motion and reduce friction, with a couple minor dimensional differences to account for different housing and components that would interfere with the previous design.

The kicker design for MKIII is adapted to the Adafruit Large Push Pull Solenoid. The plunger surface profile is smaller than the previous version to allow more space for the chipper and contacts the ball at its center. Rotation about the solenoid's shaft is restricted by the back end attachment to the solenoid.

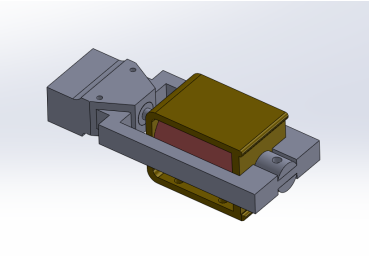

Fig. 2. MkII kicker.

2.4 Chipper

This year the team decided to integrate a chipper for the robot to increase its capabilities. The chipper gives the robot the ability to move the ball in a vertical dimension to be able to pass the ball with less risk of the ball being intercepted. At this time, the chipper has not been included in the Mkll robot and is planned to be integrated into the Mklll robot.

The chipper is to be laser cut and cnc machined from aluminum 6061 alloy and designed to match the curvature of the ball with a 21.5 mm arc, instead of a flat, angled plate. This design increases the contact area of the chipper with the ball, allowing it to distribute load more efficiently. The chipper makes contact with the ball 16 mm from the ground at a 40 degree angle. The chipper plate is attached to the dribbler body with a fixed pin, meaning that the chipper plate itself only has a rotational motion and is placed slightly below the base plate, 1 mm off the ground.

In terms of functionality, it will be powered by a similar solenoid to the kicker, and controlled by a linear transmission that transfers energy from the middle of the robot to the ball-contact surface on the back of the chipper. The frame is planned to be made from aluminium 6061 alloy and manually machined. The chipper solenoid mount is placed in the middle layer of the robot, right above the kicker solenoid, with the mount being attached to the standoff and also used as a battery mount.

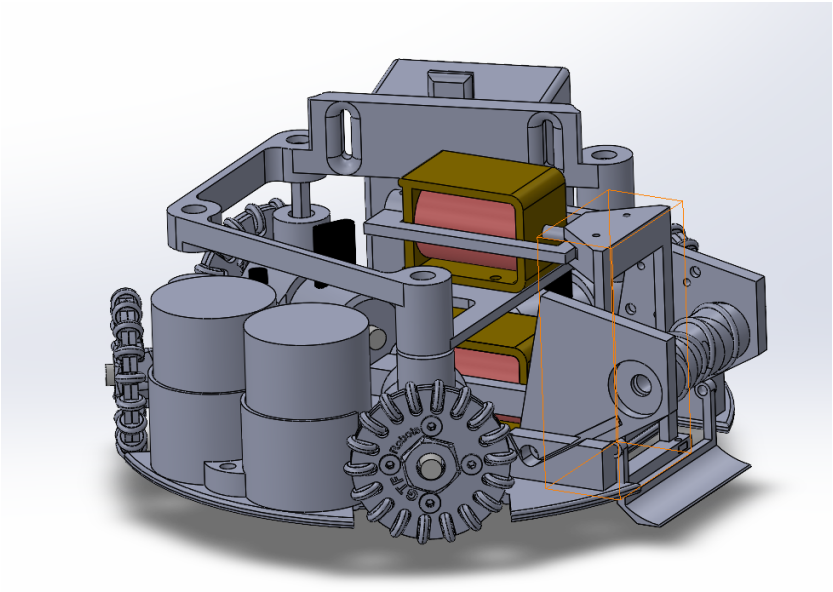

Fig. 3. MkII lower assembly and chipper highlighted.

2.5 Dribbler

The dribbler subsystem is responsible for receiving and maintaining control of the ball while working in conjunction with the kicker. It is composed of three main parts: a roller that directly contacts the ball, a body that supports the roller shaft, bearings, and motor, and a mount that attaches the assembly to the base plate. The MkII design supported only adjustable heights for testing of optimal heights based on playing surface while MkIII design features a torsion spring damping system for reliable intake of the ball.

The MKII design made use of a 1:2 gear ratio to achieve faster rpms at the cost of torque. Testing of the device revealed that the roller would often stall with the setup. MKIII will make use of a 1:1 ratio of gears to solve the issue, while adopting a new dribbler motor capable of providing higher rpm at the same or greater torque.

The team is currently testing the system with torsion springs of different stiffness for reliable intake of the ball by evaluating impact at different speeds and rotations. The springs provide a restorative torque at the pivot point of the dribbler, which resists the incoming ball's push. Once testing is complete, the team will determine if the torsion spring system provides sufficient damping or if it needs to be adapted or substituted.

RoboCup rules limit ball obstruction to 20 percent of the ball's projected area. Modeling the ball as a sphere of radius 21.5 mm, the covered region is treated as a circular segment with area

$$A_{\text{seg}} = \frac{R^2}{2}\left(\theta - \sin\theta\right) \leq 0.2\pi R^2$$

Leads to a conservative solution of 2 rad. This corresponds to a maximum allowable coverage depth of

$$d \approx 0.45R = 9.675\,\text{mm}$$

The roller has an overall cylindrical profile lofted to match the ball surface, increasing the contact area while improving damping of incoming balls. The smallest radius occurs at the roller center (7 mm), with the outer radius increasing to 8.5 mm; the total length is 60 mm, chosen to maximize contact width while remaining compatible with the kicker plunger and surrounding components. Angled grooves along the roller help guide the ball toward the robot's centerline. Based on a free-body analysis of the ball, the roller is positioned as close as possible to the top of the ball (within the 9.675 mm coverage limit) so that the horizontal component of the normal force is reduced while still providing sufficient tangential force to generate backspin, thereby favoring rotational control over forward translation during reception.

The MkIII dribbler further introduces a two-degree-of-freedom damped assembly: the dribbler rotates about an axis and is backed by damping foam to absorb impact when the ball is received or intercepted. Geometric construction using the 7 mm roller radius, 9.6 mm allowable coverage, and 7 mm plunger clearance yields a vertical placement height of approximately 21.3 mm for the roller

relative to the ball center, ensuring both visibility compliance and mechanical clearance with the kicker.

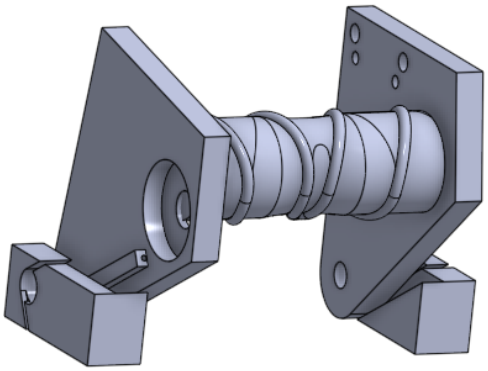

Fig. 4. MkII dribbler.

3 Electronics

3.1 Overview

As mentioned previously, the team has developed the MkII robot for the purposes of qualification and testing AI systems and the MkIII for competition. As a result MkII features electronics used in previous seasons.

The MkII robot relies on the Robomaster Development Board Type A for power distribution and is a central hub for connections with our ESP32 module, electronic speed controllers, and kicker PCB. For motor control the STM32F427 integrated within the Robomaster Development Board communicates with each C610 Motor Speed Controller through CAN protocol. The STM32F27 receives commands from the ESP32-WROOM-32D development board through UART. The ESP32-WROOM-32D allows for wireless communication through Wifi and controls the operation of the kicker.

The MkIII robot builds upon the development of MkIII through implementing ICs, microcontrollers, sensors, and motor drivers on custom PCBs for more reliable and cost effective sourcing and manufacturing of electronics. The majority of our communication protocols and systems remain unchanged.

Due to time constraints preliminary design of motor drivers and encoders is still ongoing. Following our research of BLDC motor driver ICs the team

is considering switching communication protocols for motor control due to the requirement of a CAN bus converter for translating industry standard interfaces (i.e. I2C, SPI, UART) outputted by motor driver ICs.

3.2 Kicker

The kicker electronics are heavily inspired by the Tigers Mannheim v2020 Kicking Module PCB with some notable differences [2]. To maximize the impulse needed for a strong kick of the ball we determined an electronic push-pull solenoid system was necessary.

The strength of the kick delivered by the solenoid is proportional to the current resulting from a voltage drop across the coil in a short period of time. This allows for a fast change of electrical potential energy to kinetic energy of the solenoid plunger. To achieve this a 2700uF capacitor is charged to 180V then discharged through the coil using an IGBT for rapid switching. Capacitor voltage was determined experimentally through adjusting the output of the capacitor charger then observing the velocity of the ball following successful kicks. For the purposes of qualification speed was limited to 110V for half-field play.

The design was optimized for two main aspects, charge time and speed of discharge. Fast charging of the capacitors is achieved through the LT3751 Capacitor Charger IC which utilizes a flyback converter topology [3]. The flyback converter topology was chosen for its higher efficiency at high voltage conversion ratios and galvanic isolation between high and low voltage.

For quick turn on of the IGBT the UCC27517 IC Low Side Gate Driver delivers 4A peak current capability for charging of the gate capacitor. Fast discharge times introduce flyback when switching an inductive load however. In response the MBR40250G schottky diode was added in parallel to clamp flyback.

3.3 Power Distribution

The Power Distribution Board was designed to condition the 6S LiPo battery (nominal 24V) and provide regulated power to high-current actuators and sensitive logic. Design prioritized reliability during impacts and brownouts, low EMI, and thermal margin.

The battery enters through an XT-style connector with a blade/MIDI fuse placed close to the entry point to minimize unfused trace length. Inrush current control is achieved through the LTC423 protecting upstream fuses and avoiding brownouts on plug-in. Reverse polarity and transient suppression use ideal-diode MOSFETs and TVS diodes rated for pack voltage and expected surge profiles.

For telemetry, the INA226 was used as it is a high-side current shunt with an I2C monitor that reports current, bus voltage, and power [**?**]. Powered by the +3.3 V rail, the shunt sits on the battery feed before branching rails to capture total robot current. Alarms are programmed for over-current, under-voltage, and over-power in firmware.

Synchronous buck converters step down battery voltage to +12 V and +5 V for efficiency and thermal margin. The LMR23630s (one for each voltage rail)

with soft-start and spread-spectrum features reduce conducted EMI [5]. Its wide 4 - 36 V input range supports the 6S LiPo voltage across charge states, while the integrated loop compensation and peak-current-mode control simplify design and ensure stable operation under dynamic load conditions. PCB layout was optimized for minimizing switching loop area. The +3.3V rail uses the LT3080 Low Dropout Regulator [6] with additional LC filters or ferrites at noise-sensitive loads to provide a low-noise, well-regulated rail for sensitive logic.

3.4 Mainboard Design

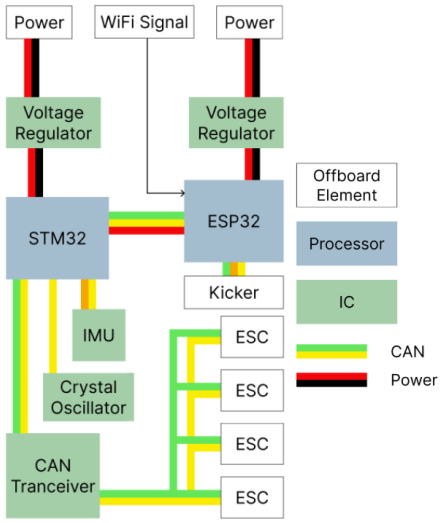

Fig. 5. Overview of mainboard connections.

Instead of relying on the RoboMaster Development Board Type A for motor control and power distribution and the ESP32-WROOM-32D we opted to design our own mainboard containing all subsystems on a single custom PCB. The four-layer board layout allows us to reduce EMI, signal noise, and overheating. The first layer was designated for signals, middle two layers ground, and bottom layer GPIO traces to header pins and low speed signals. Each MCU has its own voltage regulator, allowing each to be powered independently, or at the same time from a single USB port for testing purposes; protection between battery and USB power is also implemented.

The ESP32-S2-SOLO-2, with integrated 2.4 GHz Wi-Fi, is responsible for wireless communication with the AI vision system. Similar to last year, the ESP32 MCU operates as a UDP server to receive vision data and trigger the kicker circuit. To reduce computational and memory load on the less powerful

processor, we integrated peripherals that had previously been connected to the ESP32 MCU with the STM32 MCU directly. The ESP32 MCU subsystem retains USB functionality, a USB-to-UART interface for debugging, in-chip antenna, and GPIO header pins for further development. This design retains basic ESD protection, signal integrity, prevents EMI, and is primed for future designs.

Alongside the ESP32 MCU, an STM32F427 serves as the main control processor. It receives data from the ESP32, performs real-time control computations, and communicates commands over CAN bus to our ESCs. Operating at up to 120 MHz, the STM32 provides low-latency processing for motor control.

Similar to the ESP32 MCU design, only essential components are retained, including power regulation, clock circuitry, IMU, SWD debugging, and CAN interfaces. The inertial measurement unit (IMU) on the motherboard provides position and acceleration data with a 3-axis gyroscope and 3-axis accelerometer. The MPU-6050 chip is used and communicates to the STM32 MCU over I2C. A TCAN322 CAN transceiver chip facilitates CAN communication between the STM32 MCU and each of the motor controllers.

4 Embedded Systems

4.1 Overview

The robot's embedded architecture utilizes a dual-microcontroller system. An ESP32-WROOM-32 [7] serves as the primary communication node, handling high-level Wi-Fi commands from the AI client and managing the kicker subsystem. A RoboMaster Development Board (STM32F427 [8]) acts as the dedicated motion controller, executing the dribbler and drivetrain control loops. This separation of concerns ensures that the computationally intensive network handling does not interfere with the real-time requirements of motor control.

4.2 ESP32 Firmware Refactor and Communication Protocol Redesign

During the 2025-2026 development cycle, the ESP32 firmware was fully refactored to improve maintainability, determinism, and runtime efficiency. While system functionality remains equivalent to the previous implementation, the internal structure was redesigned to improve modularity, readability, and debugging efficiency.

A major architectural change was the **replacement of Protocol Buffers (Protobuf)** with a lightweight, text-based command protocol for communication between the off-field AI system and the ESP32. In prior implementations, Protobuf parsing introduced unnecessary complexity and memory overhead for a resource-constrained microcontroller environment. The new protocol transmits compact ASCII commands over UDP multicast and is parsed incrementally on the ESP32.

Each command is identified by a single-character opcode, enabling constant-time dispatch without string comparisons. This significantly reduces parsing latency and simplifies extensibility. The current command set uses only a small subset of the available alphabet, leaving room for future expansion without redesigning the protocol.

The ESP32 executes the following operations in a deterministic loop:

1. Receive and buffer incoming UDP packets
2. Discard stale packets and process only the most recent command
3. Decode the command opcode and parameters
4. Convert desired robot velocities into individual wheel velocities
5. Forward motor targets to the STM32 via UART
6. Update actuator state machines (kicker and dribbler)

This "latest-packet-only" strategy minimizes control jitter under high network load and improves responsiveness during rapid command updates.

4.3 ESP32-Based Kicker Control Integration

Kicker control logic was migrated entirely to the ESP32 in the current system. Previously, kicking was handled indirectly through downstream controllers. In the updated design, the ESP32 directly manages solenoid charging and firing using GPIO control.

The kicker logic is implemented as a non-blocking state machine consisting of: Capacitor charging, Ready state, Firing pulse, and Cooldown interval

Timing parameters, including charge duration, firing pulse width, and cooldown time, are configurable at compile time. This design allows high-level software to trigger kicks using a single command while enforcing safety constraints at the embedded level. Centralizing kicker control on the ESP32 also reduces coupling between motor control and high-voltage actuation logic.

4.4 STM32 PID Controller Redesign

The STM32 motor control firmware was revised to improve low-speed controllability and robustness across varying field surfaces. The updated design introduces a **time-aware control formulation** with explicit modeling of low-speed friction behavior.

Let the wheel velocity tracking error be defined as

$$e(t) = \omega_{\text{target}}(t) - \omega_{\text{measured}}(t),$$

where ω_{target} is the commanded wheel velocity and ω_{measured} is obtained from motor encoder feedback.

The controller explicitly accounts for variable loop timing by computing the elapsed time

$$\Delta t = t_k - t_{k-1},$$

and updating internal state accordingly. Finally, a low-speed friction compensation term u_f is applied to mitigate wheel stiction in the near-zero velocity regime, with the compensation reduced during velocity sign changes to limit overshoot.

The controller computes a proportional–integral (PI) control response based on the wheel velocity tracking error. The controller output is computed as

$$u_{\text{out}} = K_p\, e(t) + K_i \int e(t)\, dt + u_f,$$

where the resulting command is clipped to a fixed minimum and maximum value to respect actuator limits. A derivative term is currently omitted from the control output due to sensitivity to encoder noise at low wheel speeds, and is reserved for future evaluation if operating conditions warrant it. [9]

This formulation improves low-speed responsiveness and reduces dead-zone behavior, resulting in more consistent wheel control across robots and field surfaces.

5 Robot Decision Making Software

5.1 Overview

In previous years, our team preferred to use traditional programming to decide on the actions of our robots. However, progress became slow and difficult because of the sole reliance on a control theory approach for traditional programming. The performance of our decision-making algorithms (decision trees) did not generalize well to new scenarios. Additionally, the maintenance and development of the existing convoluted decision tree was no longer sustainable. Hence, our team has shifted to a machine learning (ML) approach to perform robot decision-making this year. Global research efforts to perform robot control through ML rather than traditional control theory inspired our team to develop a reinforcement learning (RL) approach for SSL robot control [10].

Compared to other programming languages, the most active and supported language with machine learning frameworks is Python. Our new approach is written with Python and we use the PyTorch framework. The built-in Python sockets module is used to receive external communication from SSL-Vision, SSL-Game-Controller, and AutoRefs. In order to obtain training data for our RL model's learning, we repurposed the RoboCup Soccer Simulation 2D. To clarify, we edited the C++ source code of the simulator, rcssserver, so that it can be used for our RL model to learn how to control SSL robots. Rcssserver was chosen because it's design is friendly for multi-agent AI research.

In the rest of this section we describe how we repurposed the simulator for our purposes, the definition of a *GameState*, the RL model, and an additional deployment feature used to carefully modify the simulation parameters such that the simulation models the kinematics of the real world as closely as possible.

5.2 Simulator

Repurposing Simulator for SSL

Units and Field Dimensions: The units of the simulator are in decimeters which are one-tenth of a meter. To convert the field dimensions of division B, one simply divides the milimeter measurement by 100. Thus, the simulator's pitch length and width are modified to be 90 dm (9000 mm), 60 dm (6000 mm), respectively. Similarly, the radius of the center circle is 5 dm, the penalty box's length and width are 10 dm and 20 dm, respectively, and the goal's width and depth are 10 dm and 1.8 dm, respectively. The 2025 SSL rulebook also specifies that the ball's diameter is 0.043 meters and the robot must fit in a 0.18 meters wide cylinder. Trivially, in the simulator we edit the ball size to be 0.215 dm and the robot's size to be 0.9 dm.

Ball Catching: In the SSL league, all robots can catch the ball without violating any rules. However, rcssserver only allows the goalie to catch the ball. The simulator's source code was modified so that all players are allowed to catch and dribble the ball. Lastly, if excessive dribbling is detected, a foul is called.

Removed Features: The simulator implements a "stamia" concept to physically limit the kinematics of the robots. Since our physical robots are battery powered, they do not have a "stamia" property. Thus, we discarded the code that updates the "stamia" of the robots. Also, we discard the simulator's offside rule because this rule does not exist in the SSL league.

Receiving a GameState from the Simulator

In order for our RL model to learn from its decisions, we need to also receive information about the game from the simulator. We define the *GameState* as data about the current state of the game. The *GameState* object contains:

- **count**: Simulator server cycle count or SSL-Vision Frame-Dection number
- **timestamp**: Current timestamp
- **ball-pos**: Ball position (x, y)
- **robot-poses**: Dictionary of team robot poses. These are keyed by teamname and the values are the robot ID and a corresponding pose tuple (x, y, orientation)

Notice that this format is interchangeable with data from the simulator or SSL-vision detections. The GameState can easily be obtained through the third-party sslclient Python package which automatically deserializes Protobuf packets.

When working with rcssserver, a "trainer" client is connected to the simulator. Every 100 ms (simulator cycle time) this client receives the current status of the game (server cycle count), position of the ball, and positions of all the robots on the field. The data is an UTF-8 string that be parsed with regular expressions and converted into a *GameState*.

5.3 Deep Reinforcement Learning

Our AI system was developed through deep reinforcement learning (RL) in the RoboCup Soccer Simulator environment, where our policies were trained iteratively by interacting with the soccer ball and other players on the field.

Gym Environment

To create a training environment for our agent to learn how to play soccer, we use the OpenAI Gym framework to initialize several environment functions and parameters, including the action space, step function, reward computations, and simulator reset function.

1. Observation Space. The observation space defines a bounded, continuous representation of the agent's perception based on relative field geometry rather than absolute world coordinates. Each observation is a six-dimensional vector consisting of the ball's position relative to the agent, the goal's position relative to the agent, and the agent's heading with its $\cos(\theta)$ and $\sin(\theta)$ angles. Defining these limits enforces valid observations, improve training stability, and normalization during training.

2. Action Space. Our action space comprises both discrete actions and continuous actions, similar to our basic commands implemented where each discrete action was followed by a continuous power intensity. This hybrid discrete-continuous action space was implemented in preparation for our chosen RL model selections, where we decided to proceed with two-head policy networks to choose and quantify actions within the observation space.

3. Simulator Episode Reset. Each episode in our RL training loop is defined by a full game, from the start to the time the ball is intercepted by the goal keeper, sent out of bounds, or successfully sent into the goal. When either of these conditions are met, the game states from the entire episode are processed in batches for step training, and the simulator resets so that the learned policy can continue learning iteratively and improving itself.

Deep Deterministic Policy Gradient (DDPG)

To ensure an actively learning are controlled robot, we adopt the Deep Deterministic Policy Gradient (DDPG) algorithm. DDPG is an off-policy Actor-Critic learning method designed for continuous action spaces, which makes it well-suited for understanding and modeling real-time soccer behaviors. The DDPG uses two neural networks: an Actor and a Critic.

The Actor is responsible for using the agent's policy and making a choice on certain actions based on its observations of the game environment. This network outputs control values showing how the robots move within the simulator. The

Critic will then evaluate the Actor's decisions by approximating a Q-value of the state-action pair. This network is responsible for providing feedback regarding how beneficial the actor's performance is with respect to long-term reward. The critic's evalaution updates the Actor, encouraging it to choose behaviors that have more successful outcomes.

Reward System

The reward function splits the gameplay into two modes based on possession status: "Hunter" (recover the ball) vs. "Attacker" (score). When the team has the ball, we reward the velocity of the ball towards the opponent's net (V_{prog}) and strictly penalize moving it backward (V_{reg}). When the team loses the ball, the reward shifts to minimizing the distance to the ball (d_{ball}) to encourage immediate recovery. We apply a constant penalty for crowding (teammates within 2 meters) to enforce spacing and a bonus for passing (change in ball owner) to encourage cooperation. A large binary reward or penalty (± 1) is given for scoring or conceding a goal.

$$R_t = \begin{cases} +1.0 & \text{if Goal Scored} \\ -1.0 & \text{if Goal Conceded} \\ (1.2 \cdot v_{\text{prog}} - v_{\text{reg}} + 0.1 \cdot \mathbb{I}_{\text{pass}}) - (0.005 \cdot N_{\text{crowd}}) & \text{if Possession} \\ (-d_{\text{ball}} - 0.1) - (0.005 \cdot N_{\text{crowd}}) & \text{if No Possession} \end{cases}$$

where v_{prog} is the ball's velocity projected towards the opponent's goal, v_{reg} is the ball's velocity projected towards the own goal, $\mathbb{I}_{\text{pass}}$ is a binary indicator that equals 1 when possession transfers between teammates, d_{ball} is the Euclidean distance to the ball, and N_{crowd} is the count of teammates within a 2-meter radius.

Training Methods

We use two pairs of the neural networks in the training process, the main actor-critic and the target actor-critic networks. The weights of the main pair are copied to the weights of the target pair at the start, so we're starting with the same copies of neural networks [11].

We use Experience Replay to train the networks [11]. We start by playing a game on the simulator, and each time step, the main network takes an action based on the current state, causing a next state and reward based on its action. The state, action, next state, and reward received is stored as an experience in a buffer. For training, 64 random experiences are selected from the buffer. The next step for each experience is sent to the target actor, which predicts the actions in those situations. The actions are sent to the target critic, which predicts the q-value for those next states.

We then use the reward for the current state and add it to the q-values for the next states to calculate the desired values. The current states are sent to the

main actor which predicts the actions for those situations. The current states and predicted actions are sent to the main critic, which predicts the q-values for the actions. These q-values are then compared with the target values, and their error is minimized using back propagation and gradient descent. We use the Adam optimizer. The actor networks use the tanh activation function, while the critic networks have a linear output.

5.4 Deployment

Parameter Estimation

1. Simulator dynamics. According to the simulator, the per-step discrete-time kinematics is

$$p_{t+1} = p_t + \big(v_t + u_t\big), \quad v_{t+1} = \alpha\big(v_t + u_t\big),$$

where $p_t \in \mathbb{R}^2$ is position and $v_t \in \mathbb{R}^2$ is velocity. For the ball, $u_t = 0$ when there's no kicking. When the ball is kicked with directed kick effort K_t,

$$v_{t+1} = \kappa_k K_t$$

with unknown kick rate κ_k. For the player, the control is the directed power effort P_t:

$$u_t = \kappa_d \, P_t$$

with unknown dash power rate κ_d.

2. Estimated parameters. We estimate ball_decay (α_b) and kick_rate (κ_k) only, as player_decay (α_p) and dash_power_rate (κ) are set to pre-defined values through PID control.

Experiment Script and Estimation Procedure

3. Experiment script. The robot moves behind the ball, kicks with angle 0, then waits until the ball stops before repeating. Ball trajectories are recorded after a short delay following each kick and end when the ball is stationary.

4. Estimation from data. Let pre-decay velocities of the ball be computed from adjacent differences from positions: $v'_t = p_{t+1} - p_t = v_t + u_t$

- **Ball decay:** using ball-only trajectories with $u_t = 0$, estimate α_b via least squares on
$$v'_{t+1} \approx \alpha_b v'_t.$$
- **Kick rate:** Let $(K_1, t_1), (K_2, t_2), \ldots, (K_l, t_l)$ be the (kick rate, time step) pairs of each trajectory. We obtain v'_{t_i} for all i by taking adjacent positional differences, then estimate κ_k via least squares on
$$v_{t_i+1} \approx \kappa_k K_t$$

Goodness-of-fit is reported using RMSE and R^2 on velocity residuals.

References

1. R. Rada, T. Tai, D. Bonkowsky, P. Pinela TritonBots FC 2025 Team Description Paper, 2025.
2. A. Ryll and S. Jut. TIGERs Mannheim - Extended Team Description for RoboCup2020, 2020.
3. Analog Devices. LT3751 Datasheet, July 2017. https://www.analog.com/media/en/technical-documentation/data-sheets/LT3751.pdf.
4. Texas Instruments. INA226 Datasheet, September 2024. https://www.ti.com/lit/ds/symlink/ina226.pdf?ts=1752431299280
5. Texas Instruments. LMR23630 Datasheet, August 2020. https://www.ti.com/general/docs/suppproductinfo.tsp?distId=10
6. Analog Devices. LT3080 Datasheet, October 2023. https://www.analog.com/media/en/technical-documentation/data-sheets/lt3080.pdf
7. ESP32-WROOM-32 Datasheet Version 3.6, August 2025. https://documentation.espressif.com/esp32-wroom-32_datasheet_en.pdf
8. STM32F405/415, STM32F407/417, STM32F427/437 and STM32F429/439 advanced Arm-based 32-bit MCUs. RM0090 Reference manual, June 2024.
9. Åström, K. J., Murray, R. M.: Feedback Systems: An Introduction for Scientists and Engineers. Princeton University Press (2008).
10. Lukas Brunke, Melissa Greeff, Adam W. Hall, Zhaocong Yuan, Siqi Zhou, Jacopo Panerati, Angela P. Schoellig. 2022. Safe Learning in Robotics: From Learning-Based Control to Safe Reinforcement Learning. Annual Review Control, Robotics, and Autonomous Systems. 5:411-444. https://doi.org/10.1146/annurev-control-042920-020211
11. Catacora Ocaña, J. M., Riccio, F., Capobianco, R., Nardi, D. (2019). Cooperative multi-agent deep reinforcement learning in a 2 versus 2 free-kick task [Conference session]. RoboCup 2019, Sydney, Australia. https://2019.robocup.org/downloads/program/OcanaEtAl2019.pdf