An-Najah National University

Faculty Of Engineering and Information Technology

Computer Engineering Department

Prepared By:

Khaled Yaish Zaid Balout

Supervised By:

Dr. Anas Toma

A Graduation Project submitted to the Department of Computer Engineering in partial fulfillment of the requirements for the degree of B.Sc. in Computer Engineering

Aug 20, 2025

Acknowledgement

"We would like to express our heartfelt gratitude to Dr. Anas Toma, our supervisor, for his dedication and continuous support throughout our project. His valuable scientific guidance has been invaluable to our success. We would also like to thank the academics in the Department of Computer Engineering for their expertise and assistance, as well as our friends and family for their belief in our abilities. We are deeply grateful for their contributions, support, and belief in our capabilities."

Disclaimer

The following report has been authored by students from the Computer Engineering Department, Faculty of Engineering, An-Najah National University. The report has undergone minimal modifications, limited to editorial corrections, and may still contain errors in language and content. It is important to note that the opinions expressed within the report, including any conclusions and recommendations, solely belong to the students. An-Najah National University bears no responsibility or liability for any consequences arising from the utilization of this report for purposes other than its intended commission.

Contents

Abstract

BotStrike is a tabletop arcade game that combines a **hexagon** arena, four robot carriages, and laptop-based vision and AI. Each robot has its own ESP32 control board with a buck regulator, an 8255-stepper driver, protected sensor inputs, and a 12 V solenoid for the kick action. The laptop receives an overhead camera stream, crops the region of interest, applies a pink color mask to detect the ball, maps the detection to board coordinates, and sends real-time move or push commands. Goals are detected with a simple and reliable laser–LDR pair, while an Arduino manages scoring and audio using a DFPlayer, amplifier, and speaker. An ESP "Coordinator" exposes a small web page for player names, fixtures, and live standings. The final prototype achieves stable power, responsive control, and enjoyable gameplay, and it allows the AI agent to substitute for **one missing player** so matches can still run.

1. Introduction

This project brings the energy of Crash Bash–style competitive play into the physical world. Instead of a screen-only experience, BotStrike delivers a real arena with moving robots, physical goals, and immediate audio feedback. The motivation was to build a fun, fair, and fast game that also demonstrates solid engineering across mechanics, electronics, firmware, and computer vision.

Our objectives were threefold: design a stable and safe game board and robot mechanism, implement low-latency control with clear sensing and scoring, and integrate a simple AI that can act as a substitute player when humans are missing. The work contributes a custom hexagon arena, per-robot controller boards, a practical vision pipeline that runs on a laptop, a robust goal sensor, and a light web interface for match management.

1.1 Problem Statement

Traditional arcade-style games usually need a full set of players, when people are missing, the match cannot run as intended. BotStrike targets this limitation by turning the experience into a physical tabletop system where an AI agent can replace absent player so the game can always start. Building such a system is non-trivial because it must operate in real time and span mechanics, electronics, firmware, vision, and networking. The core challenges are keeping robot motion tightly synchronized with joystick commands, detecting goals robustly with simple sensors, and preserving fairness and balance on the arena regardless of how many humans are playing. The project addresses these challenges through modular architecture that combines laptop-based vision and decision making with per-robot ESP32 control boards and a lightweight scoring/audio stack.

1.2 Objectives

The project aims to design and implement a fully working tabletop game that blends hardware, software, and AI into one cohesive system. Specifically, we seek to

1. Build a stable hexagon arena and durable robot carriages with accurate stepper-based motion.
2. Develop a reliable electronic control path that connects joysticks and AI decisions to motors and a solenoid kicker with low latency.
3. Integrate a camera and a simple computer-vision policy so an AI agent can play as one robot when a human player is missing.
4. Provide complete game handling—goal detection, scoring, and sound—through Arduino and DFPlayer.
5. Evaluate responsiveness, detection accuracy, and system stability under extended play.

1.3 Scope of Work

The scope covers the full stack from mechanics to software. On the hardware side we construct the arena, robot carriages, joystick units, and custom control boards for each robot, including power regulation and wiring. On the embedded side we program ESP32 firmware for motion control and sensing, and an Arduino for scoring and audio. On the software side we implement the laptop client for vision and AI, define the Wi-Fi protocol to the ESP32 servers, and build a small web interface on an ESP coordinator for names, fixtures, and live standings. Finally, we test and validate the system for playability, fairness, and scalability to different player counts.

1.4 Significance

BotStrike is significant because it demonstrates a complete, multidisciplinary computer-engineering solution in a tangible product. It shows how mechanics, microcontrollers, computer vision, and user experience can be integrated to create a responsive and enjoyable game that still works when players are missing. The platform also forms a practical base for future directions such as stronger AI strategies, richer vision and calibration, fully wireless controllers, and eventual packaging as a local arcade offering.

2. Constraints and Earlier Coursework

During development we faced a mix of technical and practical constraints. On the mechanical side, the robot carriages had to balance size, weight, and stability. Using compact stepper motors limited available torque, which in turn constrained top speed and sometimes affected fine positioning. The hexagon arena built from MDF, and aluminum profiles gave useful rigidity, but it also demanded careful alignment of wheels and tracks

the end-stop switches were essential to prevent overruns and keep the motion repeatable under load.

Electrical and wiring constraints were equally significant. With four robot boards, a laptop client, audio hardware, and multiple sensors operating together, power distribution required planning. We split rails into 12 V for solenoids and the amplifier, and regulated 5 V/3.3 V for logic, but still had to watch current draw and noise coupling. Level protection on sensor lines (voltage dividers) was necessary to keep the ESP32's 3.3 V inputs safe, and flyback diodes on the solenoids prevented damage from inductive spikes. Harnessing and labeled connectors reduced wiring mistakes but added assembly time.

Real-time processing imposed another constraint. The system had to keep joystick inputs, laptop vision, and motor commands synchronized so the game felt responsive. Any delay in Bluetooth messaging or stepper response

degraded the experience. We simplified networking by making the ESP32 the server and the laptop the client, used non-blocking firmware loops, and tuned debounce and timing so commands reached the 8255 drivers with minimal jitters.

Integrating the camera and AI depended heavily on lighting and calibration. The vision pipeline cropped the arena and applied a pink color mask, but threshold choice and white balance had to be adjusted to avoid missed detections or false positives. Mapping image coordinates to board coordinates also required geometric calibration. The AI policy was intentionally simple—align left/right and trigger a short push near the ball—to remain competitive without creating an unfair advantage.

Budget and resource limits shaped many design decisions. As a student project, we replaced higher-end kits with DIY mechanics, hand-soldered control boards, and locally available components. This kept costs down and improved our understanding of the hardware, but it also increased the effort needed for alignment, debugging, and reliability testing.

Further constraints arose from external restrictions: importing specific parts was delayed or blocked due to customs and border controls, which forced us to rely on locally available substitutes.

Earlier coursework directly supported the build. Electronics and laboratory classes informed the design of power stages, motor drivers, and protection components. Microcontrollers provided the foundation for Arduino and ESP32 firmware and for writing real-time, event-driven code. Image Processing coursework enabled the camera pipeline—cropping, color masking, ADC interpretation, and thresholding. Digital Design experience helped with board layout decisions and disciplined wiring. Software Engineering principles guided modular code structure, message protocols, and test planning across the laptop client, ESP32 firmware, and the web-based coordinator.

3. Literature Review

Interactive tabletop and arcade-style games have been explored for decades, from commercial cabinets to research prototypes that merge physical hardware with competitive play. Early systems tended to separate the digital logic on a screen from the tangible experience of the players. More recent work blends mechanics and electronics—custom boards, joystick controllers, and motorized units—to create a hands-on environment where users interact with real objects instead of purely virtual avatars. This shift reflects broader trends in tangible user interfaces and embedded robotics, where sensors and actuators mediate fast, repeatable interactions on a constrained playing field.

As these systems evolved, computer vision and basic autonomy began to appear, mainly to track objects, log scores, or assist the referee. Research in robotics and human–machine interaction demonstrates that integrating cameras, motors, and lightweight perception can improve fairness and keep the game state consistent. However, in many prior tabletops projects the "intelligence" remains a background service (e.g., detection and scoring), and the artificial agent is not a full participant on the field. Accessibility constraints and hardware complexity also limit scalability beyond single-purpose demos.

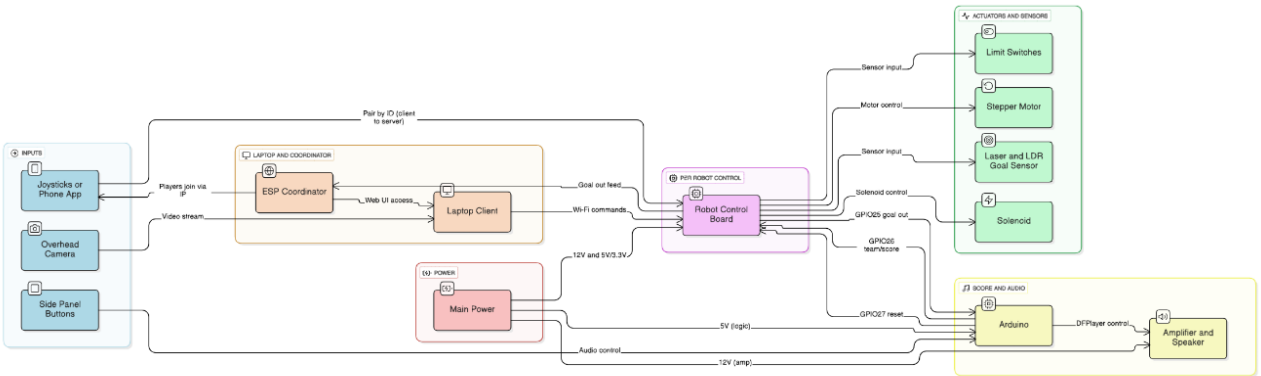
Against this backdrop, **Bot Strike** is built entirely from scratch as a cohesive, end-to-end system. The project combines a rigid hexagon arena and durable robot carriages with embedded control for precise motion, a laptop-based vision pipeline that detects and maps the ball position, and an AI policy that can actively play by issuing move and push commands. Unlike works that confine AI to passive tracking, Bot Strike treats the artificial agent as a substitute player whenever humans are missing, while maintaining stable scoring and audio through a dedicated microcontroller path. This integrated approach positions Bot Strike not only as a competitive game but also as a practical demonstration of applied computer-engineering

concepts spanning mechanics, electronics, firmware, networking, and computer vision.

4. Methodology

4.1 System Architecture & Design

BotStrike follows a layered, modular architecture that links the physical arena and robots with embedded control and a lightweight vision/AI client. A rigid hexagon arena made of MDF, and aluminum profiles hosts four robot carriages and provides mounting points for wiring and an overhead camera. Each robot is driven by its own ESP32 control board using Bluetooth for control and communication. A separate ESP 'Coordinator' opens a Wi-Fi page only for tournaments. and integrates a buck regulator, an **8255-stepper driver**, protected sensor inputs, and a **12 V solenoid** for the kick. The **laptop runs as a client**, receiving the camera stream, detecting the pink ball, mapping its position to board coordinates, and issuing **move/push** commands in real time to the relevant ESP32 board. An **Arduino** handles scoring and audio playback via a DFPlayer and amplifier, while an **ESP "Coordinator"** exposes a small web page for player names, fixtures, and live standings. Power is split between a **12 V rail** (solenoids and audio amp) and regulated **5 V/3.3 V rails** for logic and sensing. *(See the Figure: System Architecture.)*



4.2 Used Processing Units and Components

Each robot uses an ESP32 as its local controller using Bluetooth for communication, generating the DIR/STEP/EN timing for an **8255-stepper driver**, sampling two **active-low limit switches**, reading the **laser–LDR** goal sensor through the ADC, and pulsing the **12 V solenoid** through a transistor with a flyback diode. The **laptop** serves as the vision/AI client that processes the overhead camera feed, computes the ball coordinates, and dispatches movement and kick commands. An **Arduino** board maintains the game state (free-for-all and 2v2), drives the **DFPlayer → amplifier → speaker** audio path, and reacts instantly to goal signals from the ESP32. A separate **ESP Coordinator** runs a simple Wi-Fi web UI for entering players, organizing matches, and tracking the live score. The mechanical platform consists of the MDF/aluminum arena and four metal-frame carriages with wheels; the electrical platform comprises hand-soldered control boards, regulated power rails (12 V / 5 V / 3.3 V), and labeled harnesses for maintainability.

4.3 Utilized Software

The **ESP32 firmware** implements a small server that parses client messages, drives the stepper through precise DIR/STEP/EN timing, debounces limit switches, samples the LDR via the ADC to detect beam breaks, and produces short, safe pulses for the solenoid. The **laptop vision/AI** pipeline crops the camera image to the arena, applies a **pink color mask** to isolate the ball, computes its centroid, maps it to board coordinates, and applies a simple policy: align left or right; trigger a short push when the ball is near. The **Arduino program** maintains scores and lives for both FFA and 2v2, controls background music and goal sound through the DFPlayer, and coordinates **goal/team/reset** lines with the ESP32 boards. The **Coordinator firmware** serves a minimal **HTML/JavaScript** page that records names and results and keeps the scoreboard synchronized after each reset.

4.4 How the System Works

At startup, each ESP32 advertises itself as a server, and the laptop client (or a joystick/phone client) pairs by device ID. During play, the camera stream reaches the laptop, which crops the view, isolates the pink ball, and produces board coordinates. Based on this state, the laptop sends movement or kick commands over Bluetooth to the ESP32 controlling the target robot; the ESP32 generates stepper timing via the 8255 and fires the solenoid when required. **Limit switches** prevent the carriage from overrunning the track, while each **goal** is monitored by a **laser aligned with an LDR**

when the beam is broken, the ESP32 asserts a goal line to the Arduino. The Arduino immediately updates the score, plays the goal sound, and in 2v2 mirrors a team-sync signal so both teammates remain consistent. After a win condition—last robot standing in FFA or both teammates at zero in 2v2—the reset line returns the system to a clean state for the next match.

In the current prototype, the AI is limited to controlling a single robot at a time, rather than multiple substitutes.

4.5 Implementation and Prototyping

Development proceeded from design to a fully assembled prototype. We first created a **3D model** of the arena and carriages to finalize dimensions and cable routes

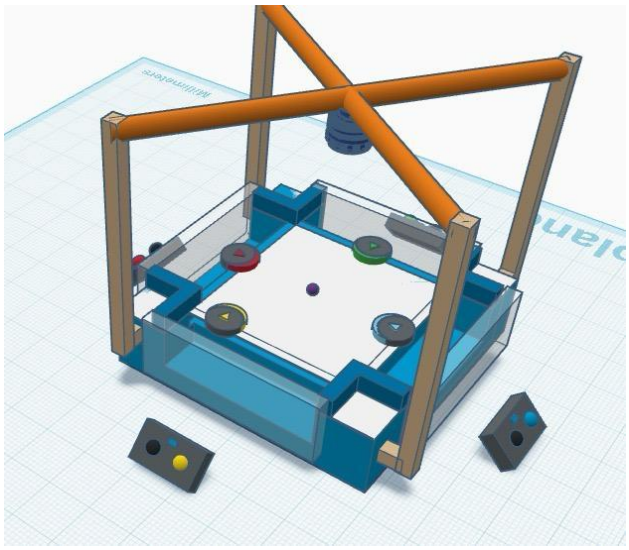


Figure 1: Initial 3D design of the Bot Strike arena and robots.

We then designed and hand-soldered four control boards on perfboard, integrating ESP32s, buck regulation, 8255 drivers, sensor inputs, and

solenoid switching; the boards were iterated until stable timing and clean power were achieved.

We also designed custom PCB layouts for a more integrated solution. However, due to budget and fabrication restrictions, these designs were not manufactured and the system was built on perfboards instead.

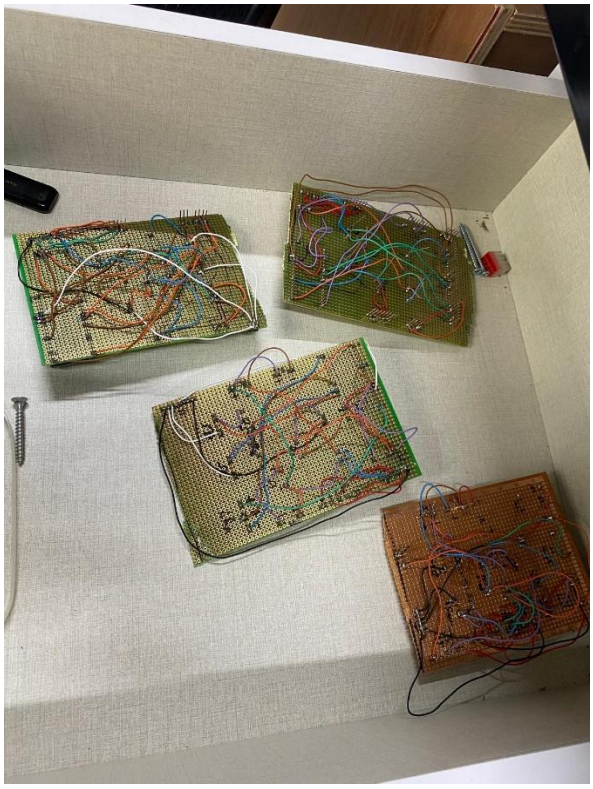
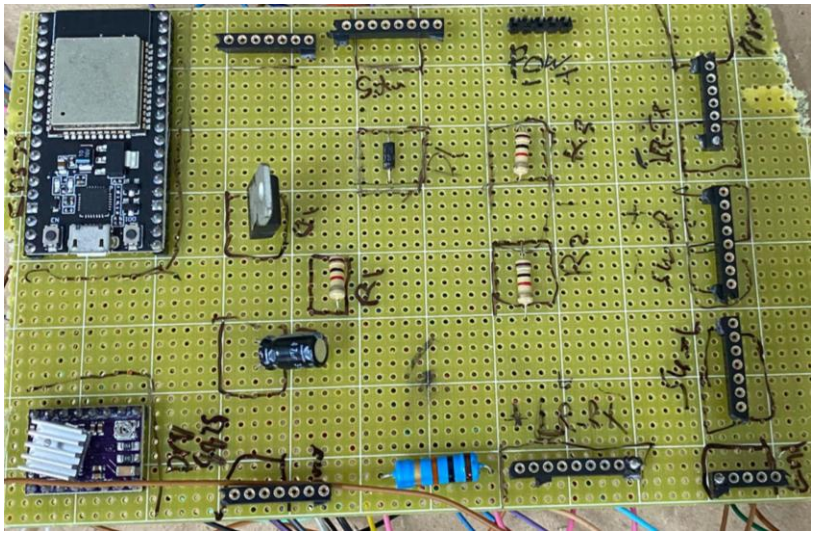


Figure 2: designed and soldered four custom circuits by hand to handle motor drivers, inputs, and power distribution.

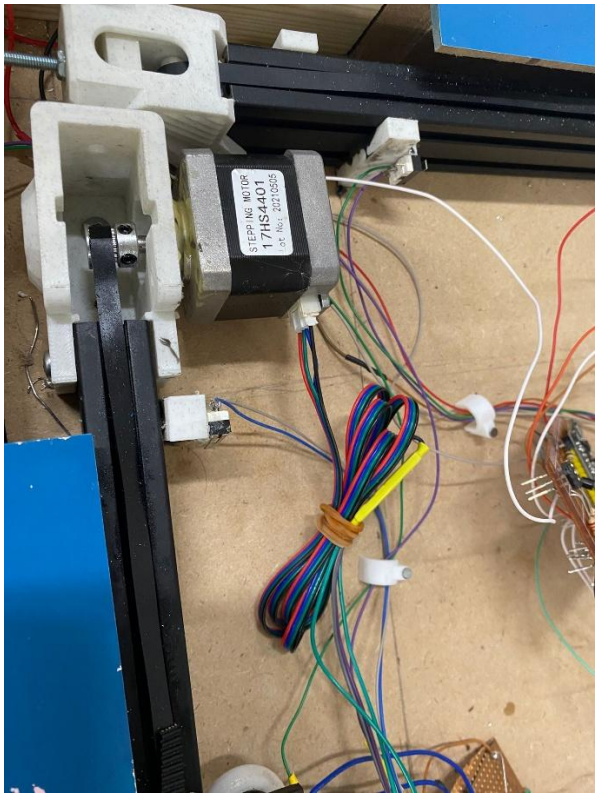


In parallel, we **built the arena** from MDF and aluminum profiles, cutting, assembling, and reinforcing the base and walls to create a durable, low-friction playing surface

Figure 3: Early stage of arena construction using MDF boards.

Figure 4: Cutting and preparing aluminum for the arena frame.

Figure 5: Assembling support structures and joints for stability.


STEPPING MOTOR
17HS4401

With the frame ready, we **installed and aligned the stepper mechanisms** and verified end-stop behavior to ensure accurate, repeatable motion along the tracks

Figure 6: Mechanical assembly of the arena frame with mounted stepper motors.

Finally, we **completed assembly and finishing**, mounted the robots and joystick units, installed the overhead camera frame, and integrated the laptop vision/AI, Arduino audio/score, and web-based coordinator into a cohesive system

Figure 7: Completed Bot Strike arena with player robots in place.

Figure 8: Close-up view of a player robot, showing the stepper motor mechanism.

Figure 9: Final assembly of the overhead metal frame used to mount the vision camera above the arena.

5. Results & Discussion

5.1 Results

After completing the design and build, BotStrike operated as a fully functional tabletop arcade system. The custom hexagon arena provided a rigid, low-friction surface that kept robot motion smooth and the geometry fair between players. Joystick control felt responsive in playtests: commands translated to movement with minimal perceived delay, and the robots maintained balance and direction reliably across repeated matches. The stepper-driver stage supplied sufficient torque for quick changes in direction, while the end-stops prevented overruns and helped preserve repeatability along the tracks. The vision and AI loop on the laptop successfully filled missing seats; when fewer than four humans were available, the agent detected the pink ball, mapped its position to board coordinates, and issued left/right and push actions that kept matches competitive without stalling the game.

This functionality was implemented for one robot only, ensuring that at least a single missing player could be replaced by AI.

Throughout extended sessions the electrical platform remained stable: the 12 V rail for solenoids and audio, together with the regulated 5 V/3.3 V logic rails, showed no disruptive drops, and the system handled simultaneous inputs and scoring/audio events without noticeable glitches.

5.2 Discussion

The results highlight a number of strengths in the implemented architecture. End-to-end integration across mechanics, embedded control, computer vision, and user experience yielded a coherent product rather than a set of isolated demos. The decision to make each robot an ESP32 “server” and to run vision/AI on a laptop “client” simplified pairing, reduced bottlenecks, and kept latency low in practice. Goal detection via a simple laser–LDR pair proved robust after threshold tuning, and the DFPlayer-based audio track contributed to the arcade feel during play.

At the same time, several limitations became clear. Color-based tracking is sensitive to lighting and white balance; despite cropping and threshold calibration, the pipeline occasionally required retuning to avoid false positives or missed detections. Mechanically, the compact stepper setup constrained peak torque, which caps top speed and can affect fine positioning at higher accelerations; stiffer wheels or stronger motors would improve headroom. Wiring complexity increased with four boards and multiple sensors/actuators; careful harnessing, labeling, and separation of power and signal paths mitigated noise, but the layout would benefit from a consolidated PCB revision. Overall, the system met its objectives—delivering a stable, responsive, and enjoyable tabletop game while demonstrating applied computer-engineering concepts across hardware, firmware, and AI—yet it also points clearly to future work on more robust vision, cleaner mechanics, and a more integrated electrical design.

6. Conclusion and Future Work

Conclusions

BotStrike demonstrates the successful integration of mechanics, embedded control, and computer vision into a single, cohesive tabletop arcade system. The final prototype delivers a fully working game built from scratch: a rigid hexagon arena, four robot carriages with stepper-based motion, a laptop-driven vision/AI loop that can substitute for one missing player so matches never stall.

Joystick interaction feels smooth and responsive, while synchronization across the ESP32 robot boards, the 8255 motor drivers, and the Arduino score/audio path remained reliable in extended sessions. The architecture—ESP32 servers on the robots and a laptop client for vision and decisions—proved practical and scalable, and the laser–LDR goal sensor offered a simple, robust way to detect scoring events. As a whole, the project bridges

classroom concepts in electronics, microcontrollers, real-time software, and computer vision with a tangible, enjoyable system.

Future Work

Future iterations should deepen the AI behavior beyond the current policy by adopting richer decision logic and, where appropriate, reinforcement learning so computer-controlled robots adapt to player styles, lighting conditions, and ball dynamics over time. On the mechanical side, upgrading to higher-torque motors or improved wheel and carriage designs would raise speed and precision, while a more modular chassis would reduce downtime for maintenance and part replacement.

Electrically, consolidating the per-robot control board into a manufactured PCB, tightening grounding and cable routing, and expanding protection circuitry would lower noise and further increase reliability.

Wireless controllers are a natural evolution of the joystick units, moving from wired inputs to low-latency wireless links would simplify setup and reduce wiring complexity around the arena. The vision stack can also advance with higher-resolution or multi-camera configurations, better color-invariant features, and automatic calibration routines to keep performance stable across different lighting environments.

From a gameplay perspective, adding new modes—such as obstacle layouts, alternative ball types, or multi-round tournaments with brackets and statistics—would increase replay value and support structured competitions.

Also, Fabricating the PCB designs we prepared would simplify wiring, reduce noise, and improve reliability in future versions.

Extending the AI to handle multiple robots simultaneously would increase flexibility in matches with fewer human players.

Finally, the system is a promising candidate for small-scale commercialization. Refining the industrial design, simplifying assembly, improving safety and serviceability, and packaging the software into a maintainable release would position BotStrike for pilots in arcades, gaming cafés, and educational labs. Together, these steps form a clear roadmap from a successful prototype to a robust, user-friendly product.

7. References

- Ishii, H., & Ullmer, B. "Tangible Bits: Towards Seamless Interfaces between People, Bits and Atoms." *CHI '97.* (Seminal paper on physical interactive systems.) GitHub
- Dietz, P., & Leigh, D. "DiamondTouch: A Multi-User Touch Technology." *UIST 2001.* (Multi-user tabletop groundwork.) Arduino
- RoboCup Small Size League (SSL) — Official rules (overhead vision + global tracking design used in robot soccer). RoboCup Soccer
- SSL-Vision: Open-source vision system for SSL (architecture of camera-over-field tracking). RoboCup Soccer
- OpenCV docs — HSV color spaces & inRange thresholding (what you used to isolate the pink ball). OpenCV+1
- OpenCV camera calibration tutorial (to justify lens calibration / homography for top-down mapping). OpenCV+1
- Arduino-ESP32 Wi-Fi API reference (practical WiFiServer/WiFiClient usage). AllDatasheet
- Espressif — ESP32 ADC programming guide (ADC, attenuation, calibration; supports your LDR/threshold design). Espressif Documentation
- ESP32 datasheet (ADC specs & electrical characteristics to cite numeric limits). DigiKey+1

- Allegro A4988 stepper driver datasheet (canonical STEP/DIR microstepping driver—same class as your driver). Tutorials for Newbies
- TI DRV8825 product page + datasheet (higher-current STEP/DIR driver, widely referenced). Texas Instruments
- Flyback diode (what/why across the solenoid; use as background + figure). Wikipedia
- SparkFun Thumb Joystick Hookup Guide (2-axis analog joystick → ADC mapping). SparkFun Learn
- DFRobot DFPlayer Mini wiki/manual (MP3 module specs, UART control, FAT32 on microSD). DFRobot Wiki+1