CPS WS24 Final Project - Air Hockey Robot

Felix Aker
Technical University of Munich
Heilbronn, Germany
felix.aker@tum.de

Begüm Kara
Technical University of Munich
Heilbronn, Germany
begum.kara@tum.de

Dzan Burzic
Technical University of Munich
Heilbronn, Germany
dzan.burzic@tum.de

Vrushabh Jain
Technical University of Munich
Heilbronn, Germany
vrushabh.jain@tum.de

Esmir Kićo
Technical University of Munich
Heilbronn, Germany
e.kico@tum.de

Gift Enabulele Asemota
Technical University of Munich
Heilbronn, Germany
gift.enabulele@tum.de

Samuel Kemmler
Technical University of Munich
Heilbronn, Germany
go34taw@mytum.de

***Abstract*—This project presents the design and implementation of an autonomous air hockey system aimed at providing a training partner for human players. By modifying a small ice hockey table with custom 3D-printed components, motors, and elastic bands, electronics and more, the system enables a puck to be mechanically moved across the playing field. An overhead camera provides real-time tracking of the player's puck, allowing a program to anticipate movements and execute defensive actions to defend its own goal. Unlike a traditional opponent, the system does not attempt to score goals but instead focuses on blocking shots and returning the puck to the player. The resulting system successfully combines hardware prototyping, computer vision, and control algorithms into a cyber-physical platform that demonstrates both reliable functionality and potential for further development.**

I. Introduction

Air hockey is a game that demands speed, precision, and quick reflexes. Its straightforward rules and competitive nature have contributed to its popularity in both recreational and competitive settings. However, a limitation arises from the requirement of two human players, which restricts the opportunities for training when an opponent is not available. Digital simulations and computer games provide an alternative, yet they lack the realism of physical games, where friction, inertia, and human unpredictability play a decisive role.

This project aims to design and implement an autonomous system that serves as a practice opponent on a physical air hockey table. Rather than focusing on scoring goals, the system emphasizes defensive play, blocking incoming shots, and returning the puck to the player. This approach creates a realistic training environment for practicing offensive strategies and improving reaction speed.

Realizing such a system involves solving multiple subproblems:

Mechanical construction: The robot requires actuators and a reliable puck-handling mechanism. We designed and manufactured 3D-printed parts that, in combination with motors, aluminum beams, and elastic bands, allow precise control of the puck.

Electronics System Implementation: Connect different electrical systems, such as wires, batteries, and Arduino, in a way that makes the motors communicate with our program so that it executes actions in a fast and precise way.

Computer vision: A camera positioned above the table is used to continuously monitor the playing field, enabling the program to track the player's movements and respond accordingly.

System integration: The mechanical hardware, vision system, and control algorithms must operate in real time and with sufficient reliability to provide smooth gameplay.

To this end, a small air hockey table was retrofitted with the necessary mechanical and electronic components. The system leverages an overhead camera to detect the puck's position, while control logic determines the appropriate response to the player's actions.

This report is structured as follows:

- Section 2 describes the system architecture and mechanical design,
- Section 3 details the control system and electronics,
- Section 4 explains the computer vision setup,
- Section 5 presents the defensive algorithm behind programming,
- Section 6 concludes with final remarks.

II. Mechanics Implementation

The air hockey robot required a fast and lightweight two-dimensional motion system to follow and intercept the puck. After considering different options, an *H-Bot* design was selected instead of a traditional Cartesian gantry.

A. The H-Bot System

The H-Bot is a belt-driven XY motion system that uses two fixed stepper motors and one continuous belt arranged in an "H" shape. The belt is attached to the moving carriage in the middle of the frame. Because the motors are fixed to the frame, the carriage does not carry their weight, which reduces moving mass. Motion is achieved by driving the motors in different combinations:

- Both motors rotate in the same direction $\rightarrow$ the carriage moves horizontally.
- Motors rotate in opposite directions $\rightarrow$ the carriage moves vertically.
- Only one motor rotates $\rightarrow$ the carriage moves diagonally.

This arrangement allows smooth motion in two dimensions while keeping the construction simple and lightweight. The main challenge is that the belt must be aligned carefully, otherwise twisting forces ("racking") can occur on the carriage.

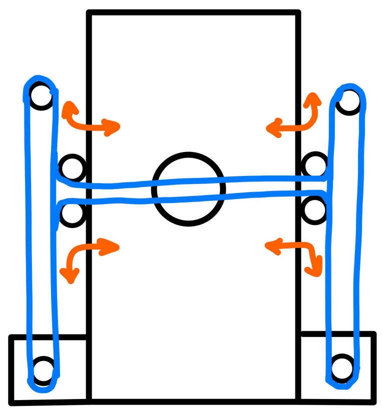

Fig. 1. H-Bot system: two fixed motors drive a continuous belt in an H-shape to move the carriage.

B. Comparison to Cartesian System

A Cartesian gantry also enables XY motion, but it stacks one axis on top of the other. This means one motor and rail system has to carry the full weight of the second axis. While mechanically simple, this leads to several drawbacks:

- Higher moving mass, making acceleration slower.
- More mechanical stress on the supporting axis.
- Bulkier construction, which is less suitable for a compact air hockey table.

For these reasons, the Cartesian setup was rejected in favor of the H-Bot, which provides faster, more efficient movement with fewer moving parts.

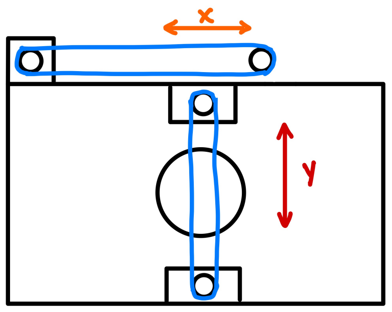

Fig. 2. Cartesian system: stacked X and Y axes, where one axis carries the other's weight.

C. Construction

The H-Bot system was mounted on top of a small air hockey table using aluminum rods and 3D-printed parts. The carriage itself acts as the robot mallet and makes direct contact with the puck. The structure was assembled in the following way:

- Two rigid steel corners were attached to the round corners of the air hockey table.
- Linear rods were installed to guide the carriage smoothly in both directions.
- Stepper motors were mounted to the frame with 3D-printed motor brackets.
- Idler pulleys and belt guides were added to route the continuous H-shaped belt.
- The carriage was 3D-printed and fixed to the belt, allowing it to slide along the rods and act as the striking surface.

D. 3D-Printed Parts and Fastening

Most parts were 3D-printed with 20% infill, which was sufficient for strength while keeping weight low. The slider pieces, however, required 100% infill to prevent breakage, since they carried the highest mechanical load. All parts were assembled with M3 screws and nuts, which allowed easy disassembly and adjustments.[3]

E. Challenges and Adjustments

During assembly, the main challenge was belt alignment and stiffness of the carriage movement. Small misalignments caused friction, which had to be corrected by re-adjusting the rods and tightening the belt evenly. Once tuned, the carriage moved smoothly across the table and provided a reliable mechanical base for the rest of the system.

III. Electronics System Implementation

A. Introduction

This part of the report documents the electrical subsystem design and implementation for our ice hockey project. Our team was responsible for developing a reliable stepper motor control system using modern driver technology and precise current management to ensure optimal performance and component longevity.

B. Component Selection and Specifications

The electrical system was built around carefully selected components optimized for reliability and performance. We implemented TMC2209 stepper drivers, chosen specifically for their superior thermal characteristics and improved reliability compared to traditional stepper driver solutions. These drivers were paired with Arduino Uno E3 microcontrollers, providing robust digital control capabilities through well-documented GPIO interfaces.

For mechanical actuation, we selected Nema 17 42BYGH48 stepper motors with a 1.7A current rating. [1] These motors provide the necessary torque and precision required for our application while maintaining compatibility with our driver selection. The entire system is powered by a 12V AC-to-DC power adapter, ensuring adequate power delivery for sustained operation.

C. Current Limiting and Thermal Management

Proper current management was critical for the reliability of the system. The TMC2209 drivers utilize current sensing resistors positioned adjacent to the main chip for accurate current measurement. Our drivers featured R110 configuration with $R_{cs} = 0.11\Omega$, enabling precise current control through the relationship:

$$\text{Current Limit} = \frac{V_{ref}}{8 \times R_{cs}} \tag{1}$$

To achieve our target motor rating of 1.7A, the theoretical reference voltage requirement was calculated as $V_{ref} = 1.496$V. However, to prevent overheating and component stress, we implemented a conservative 80% power setting, reducing the reference voltage to 1.19V. This de-rating strategy significantly improves long-term reliability while maintaining adequate performance for our application requirements.

D. System Configuration and Wiring

The reference voltage was set using analog calibration methods, carefully adjusting the onboard potentiometer while simultaneously monitoring the voltage with a precision multimeter. This hands-on approach ensured accurate current limiting within our design specifications.

Motor connections followed standard two-phase stepper configuration protocols. Each Nema 17 motor contains two independent coils, which we connected systematically: the first phase to terminals M1A and M1B, and the second phase to M2A and M2B. Wire phase identification was verified through continuity testing using multimeter measurements to ensure proper electrical connections.

The control interface utilized three primary signal lines per stepper driver connected to Arduino GPIO pins. The DIR pin controls rotational direction, while the STEP pin receives pulse signals that advance the motor one step per pulse. The EN (enable) pin operates with active-low logic, remaining LOW during normal operation and switching HIGH to disable the drivers when motor shutdown is required.

E. Power Distribution Architecture

Our power distribution strategy separated logic and motor power supplies for improved noise isolation and system stability. The stepper drivers VIO and GND pins connect to the Arduino's 5V output and ground, providing clean logic-level power for digital control signals. Meanwhile, the high-current motor power connections (VM and GND) interface directly with the 12V of up to 10A power supply, ensuring adequate current delivery for motor operation without compromising the microcontroller's power integrity.

IV. Camera Vision System Implementation Report

At the start of the project we tried a simple grayscale circle detection method, but it was not reliable. It was difficult to separate the puck, the player paddle, and the robot paddle from each other, and the system often confused objects or lost track completely.

To solve this, we switched to using color detection with the HSV color space, which worked much better. Each object on the field was marked with a unique color: the puck in blue, the human paddle in green, and the robot paddle in orange (see Figure 4). The camera image is converted into HSV, and for each color a mask is created. Noise in the mask is reduced using small filtering steps.

From the mask, the system looks for circular shapes. Only the largest circle of each color is kept, so that each entity always has a clear and unique position. If the robot paddle is not visible for a short moment, the system continues to use its last known position until it is found again.

The result is a set of 2D positions in pixels for all three objects. These positions are updated up to 60 times per second and are sent to the robot control software, which uses them for puck prediction and movement planning. With

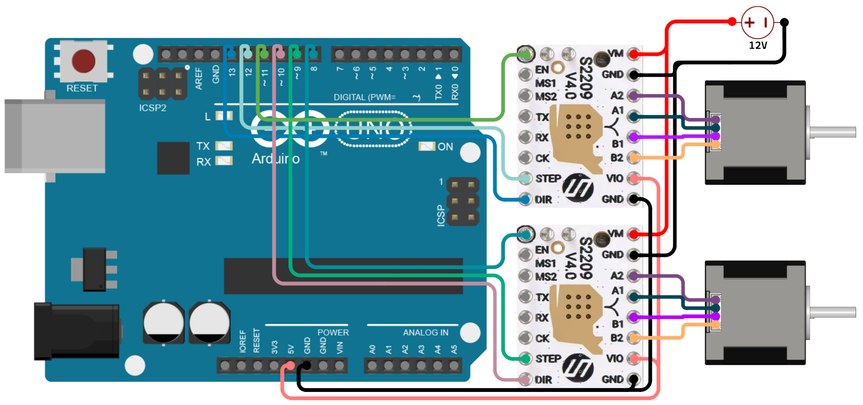

Fig. 3. Circuit scheme

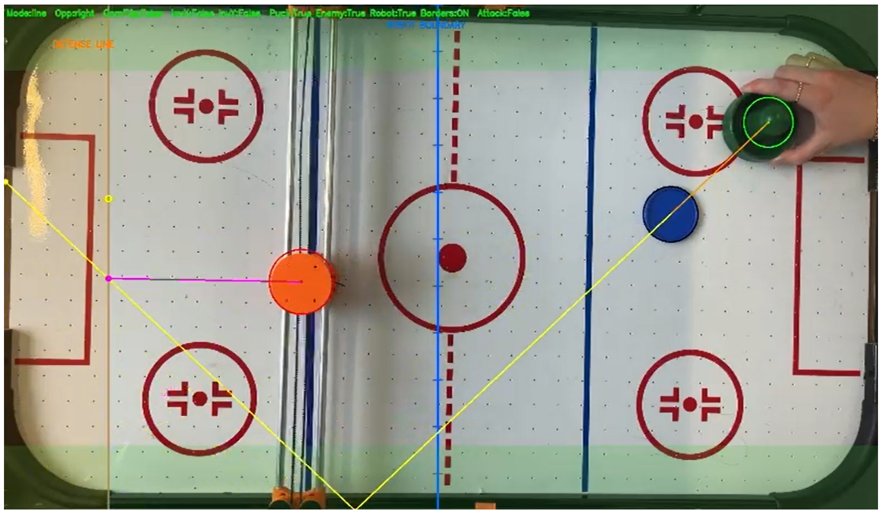
Mode:line Opp:right Puck:True Enemy:True Robot:True Borders:ON Attack:False
DEFENSE LINE

Fig. 4. Player paddle in green, puck in blue, robot paddle in red.
Additional you can see the shot prediction represented as a yellow line and the needed movement to defend represented as a pink line.

the color-based system, tracking became stable and accurate enough for real-time air hockey gameplay.

V. Programming Implementation Report

A. Overview

The programming component of the air hockey robot integrates the camera vision system(puck, human paddle, robot paddle positions), real-time decision making(defend/attack), and robot actuation. The implementation is written in Python and communicates with the Arduino-based motor controller over serial. Two main software versions were developed during the project: the initial `Robot.py` and a refined `RobotV2.py` with improved defensive behavior.

B. Software Architecture

- **I/O**: Frames arrive from the vision process at up to 60 Hz. Each frame provides pixel coordinates for puck and paddles.
- **Decision Layer**: Mode selection (defend vs. attack), shot prediction, target pose generation, safety clamping.
- **Motion Interface**: Target deltas are mapped to the table's motor basis and transmitted via serial as integer steps to the Arduino.
- **Firmware**: The Arduino executes time-accurate step pulses with an acceleration ramp, acknowledges commands, and idles safely.

C. Coordinate System & Motor Kinematics

Our gantry uses two motor axes arranged as $X{-}Y$ and $X{+}Y$ combinations (mechanically at $\pm 45°$). Let $(\Delta x, \Delta y)$ be the desired Cartesian step displacement. The motor step deltas (Δ_1, Δ_2) are:

$$\Delta_1 = \Delta x - \Delta y, \qquad \Delta_2 = \Delta x + \Delta y.$$

Conversely, for diagnostics,

$$\Delta x = \tfrac{1}{2}(\Delta_1 + \Delta_2), \qquad \Delta y = \tfrac{1}{2}(\Delta_2 - \Delta_1).$$

These relations are applied in the Python controller before serial transmission so the firmware remains simple and timing-precise.

D. Main Controller (Robot.py)

The first version established a reliable defensive baseline. The robot was programmed to remain on a fixed vertical defense line in front of its goal. This ensured that the puck could be intercepted whenever it approached the robot's half.

In addition to passive defense, the code included an automatic attack mode. If the puck remained in the robot's half for longer than a predefined threshold, the robot switched from

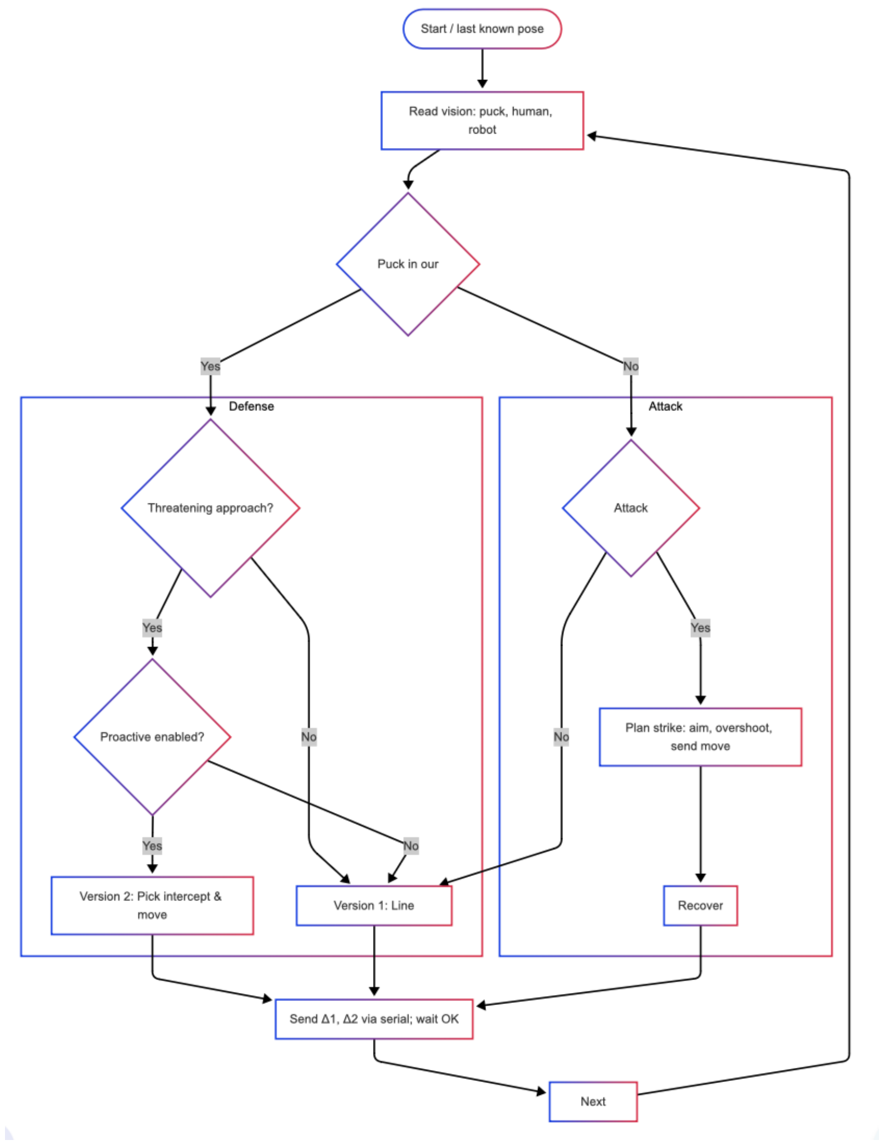

Fig. 5. Clean layout: defense (left), proactive decision (center), and attack (right), all funneled to execution.

defensive mode to offensive play. In this mode, the robot approached the puck and pushed it toward the opponent's side, while staying within legal play boundaries such as the crease and table margins. Safety margins and motor inversion options were also implemented for robust calibration and reliable gameplay.

- **Line Defense**: The mallet holds a vertical defense line in front of the goal. Horizontal (lateral) repositioning tracks the puck to block incoming shots reliably.
- **Attack Trigger**: If the puck dwells in our half beyond a time threshold (and speed falls below a limit), the robot approaches and pushes it across the center line using a bounded forward stroke.
- **Safety & Calibration**: Table margins, crease boundaries, and soft limits are enforced. Parameters exist for motor inversion, max step rate, deadband, and rate limiting so the system remains stable across setups.
- **Portability**: The code auto-detects serial ports across OSes, includes basic loss-of-vision fallbacks (last known pose) and guards against command flooding.

E. Proactive Controller (RobotV2.py)

While the line-based defense was stable, it limited reactivity against angled shots. The second version introduced a more proactive defense strategy. Instead of strictly staying on a fixed axis, the robot actively positioned itself based on puck trajectory and predicted paths. This increased coverage of the goal area and allowed earlier interceptions.

`RobotV2.py` also refined the attack logic. The robot waited for the puck to be relatively still before striking, reducing wasted movements and ensuring stronger attacks. Parameters such as strike overshoot distance, recovery timeout, and required stillness duration were configurable, making the robot more adaptable to different game conditions.

- **Trajectory-Aware Defense**: Rather than staying on a fixed axis, the robot selects an interception point along the predicted puck path and repositions preemptively. This increases save rate against off-axis entries.
- **Smarter Attacks**: Striking waits for low puck velocity and favorable geometry, reducing wasted movements. Overshoot distance, stillness duration, and recovery time are tunable.
- **Behavioral Tuning**: Parameters expose the defense aggressiveness (how far to leave the goal line), cooldowns between strikes, and maximum reposition burst length.

Empirically, `RobotV2.py` yields earlier blocks and cleaner counter-attacks, while preserving the guardrail checks from the baseline.

F. Serial Protocol (Python $\rightarrow$ Arduino)

Each motion command is a single line with the two motor step deltas in the motor basis:

```
<delta1>,<delta2>\n
```

where `delta1` applies to the $X-Y$ axis and `delta2` to the $X+Y$ axis. The Arduino replies with `OK` after completing the move. The Python side rate-limits transmissions, clamps excessively large deltas, and retries on missing acknowledgements. This minimal, line-oriented protocol proved robust and easy to debug.

G. Communication and Integration

Both versions communicated with the Arduino microcontroller through serial messages, translating pixel coordinates from the camera into stepper motor commands. Commands were rate-limited to ensure stable communication. The code supported automatic detection of serial ports across operating systems, making it portable for development and testing.

H. Arduino Execution Model

The Arduino firmware receives each command, parses the two integers, and generates step pulses on two TMC2209-driven stepper channels:

- **Pinout**: `STEP1/DIR1/EN1` for $X-Y$, `STEP2/DIR2/EN2` for $X+Y$.
- **Motion Block**: For each command, the firmware sets directions from the signs of `delta1`/`delta2` and steps the larger of the two counts while interleaving the smaller, ensuring both axes finish together.
- **Acceleration Ramp**: A simple time-based ramp decreases the inter-step delay from `MAX_STEP_DELAY` to `MIN_STEP_DELAY`, updated every `ACCEL_DELAY` μs. This limits jerk and reduces stalls without heavy computation.
- **Acknowledgement**: On completion, the firmware prints `OK`, enabling the host to pipeline the next command safely.
- **Idle Safety**: After 30 s without new commands the drivers are disabled (`EN`=HIGH) to reduce heat and power; this also acts as a fail-safe halt.

This division of labor—planning in Python, precise pulse timing on the microcontroller—keeps the control loop responsive while preserving step timing integrity.

I. Key Parameters and Safety

- **Speed Limits**: Host clamps target deltas per cycle; firmware enforces per-step minimum delays.
- **Boundaries**: All target positions are clipped to legal play regions (table margins and crease).

- **Watchdogs**: Serial timeouts, missing-vision fallbacks, and idle driver disable reduce failure impact.
- **Tuning**: `MIN/MAX_STEP_DELAY`, `ACCEL_DELAY`, strike overshoot, dwell thresholds, and defense aggressiveness can be adapted to surface friction and camera FPS.

J. Summary

Overall, the programming effort provided the intelligence of the system, combining computer vision input with real-time decision making. The transition from a static defensive line in `Robot.py` to a proactive, prediction-based defense in `RobotV2.py` significantly improved the robot's gameplay. These iterations reflect a balance between stability, responsiveness, and offensive capability, forming the core control logic of the air hockey robot.

VI. Conclusion

The autonomous air hockey system achieved its primary goal of providing a defensive robotic opponent for training and recreational purposes. The system reliably blocks shots and passes the puck back, offering users a realistic physical experience that can be used instead of a real player for practice.

Key insights gained during the development include the importance of precise vision calibration, the role of iterative prototyping using 3D printing, and the benefits of modular design for testing and troubleshooting. While the system fulfills its initial requirements, improvements in puck speed, vision robustness under changing lighting, and more advanced gameplay strategies could significantly expand its capabilities.

Overall, this project demonstrates a successful proof-of-concept for applying cyber-physical principles to interactive gaming and highlights opportunities for future enhancements in both mechanical and algorithmic domains.

References

[1] HandsOn Technology, *42BYGH48 Stepper Motor Datasheet*, https://www.handsontec.com/dataspecs/motor_fan/42BYGH48-4170.pdf

[2] G. D. Greenwade, "The Comprehensive TeX Archive Network (CTAN)," *TUGBoat*, vol. 14, no. 3, pp. 342–351, 1993.

[3] jjrobots, "Air Hockey Robot EVO (SMARTPHONE CONTROLLED)," Thingiverse. [Online]. Available: https://www.thingiverse.com/thing:1804534.