TinkerBlocks

TinkerBlocks: Code, Build, and Drive! is a tangible educational platform for teaching programming concepts.

Project Title

TinkerBlocks: Code, Build, and Drive!

By

Amr Badran & Izzat Alsharif

Supervisor

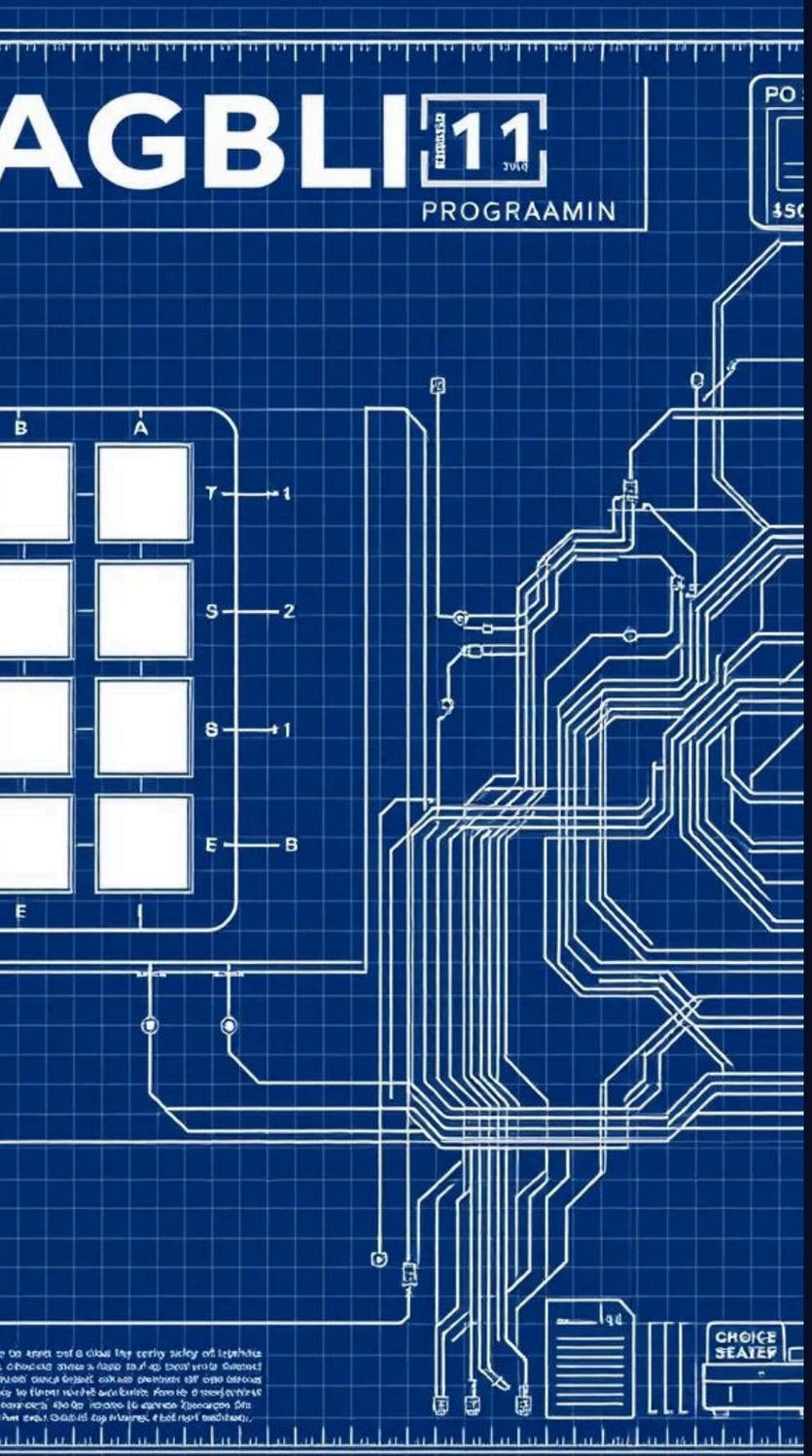
Dr. Ashraf Armoush

University

An-Najah National University, June 2025

TinkerBlocks: Programming Education

Welcome to our presentation on TinkerBlocks, an innovative tangible educational project designed to teach programming concepts to children through hands-on interaction.



Project Objectives: What We Aimed To Achieve

Our primary objectives were to create a robust and engaging educational platform.

Fun & Tangible

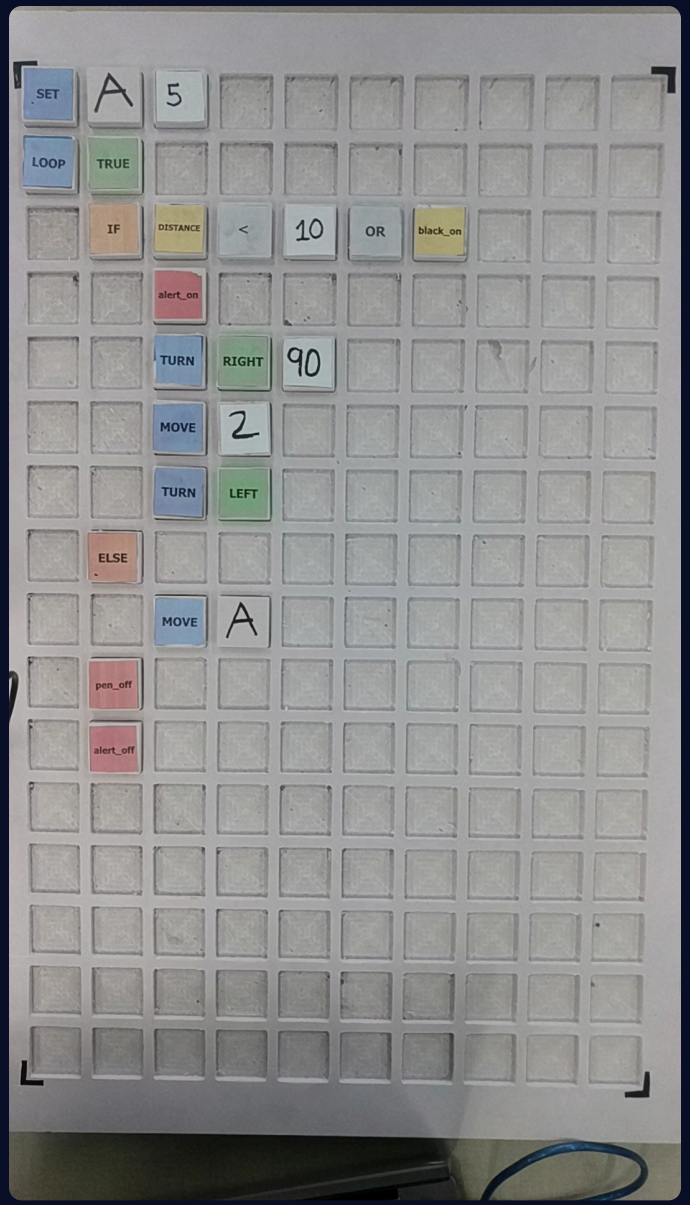
Gameboard to teach students about programming in a physical fun way

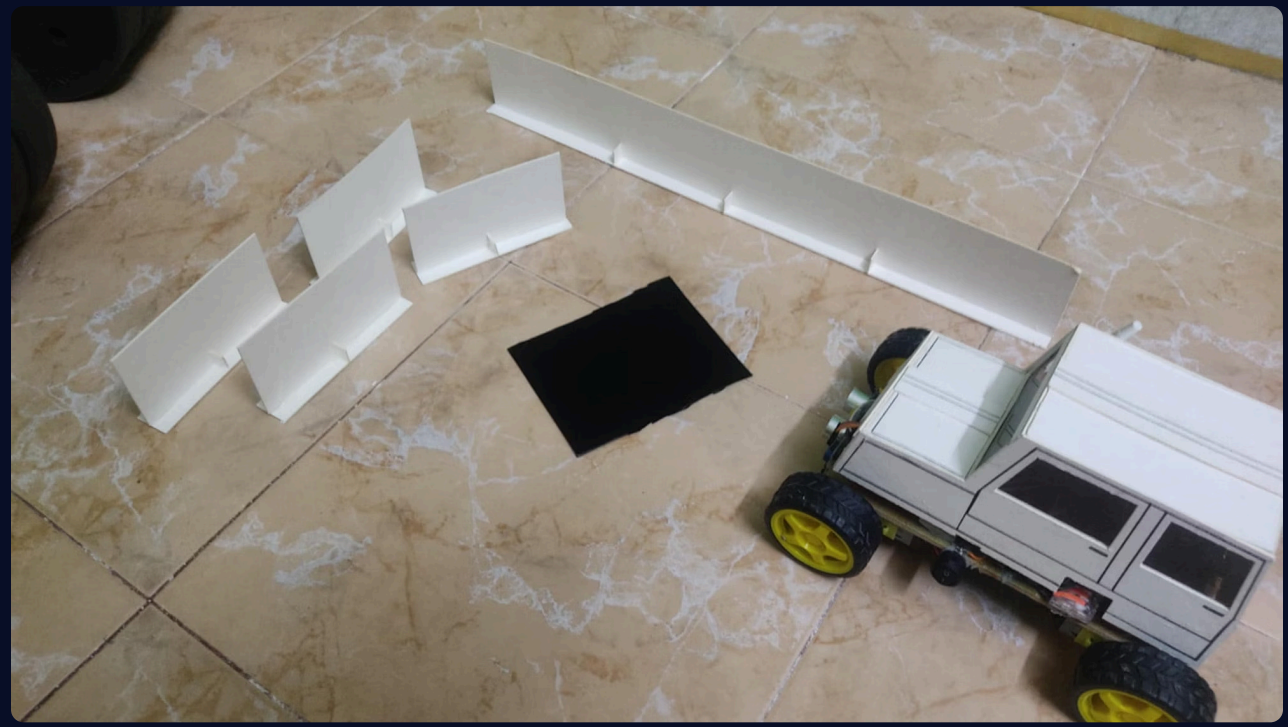
Programmable Robot

A robotic car with sensors & actuators that solves many tasks and achieves differe

Visual Programming

Support many common patterns in programming languages like control flows (loop, if, etc.) in a simple board game

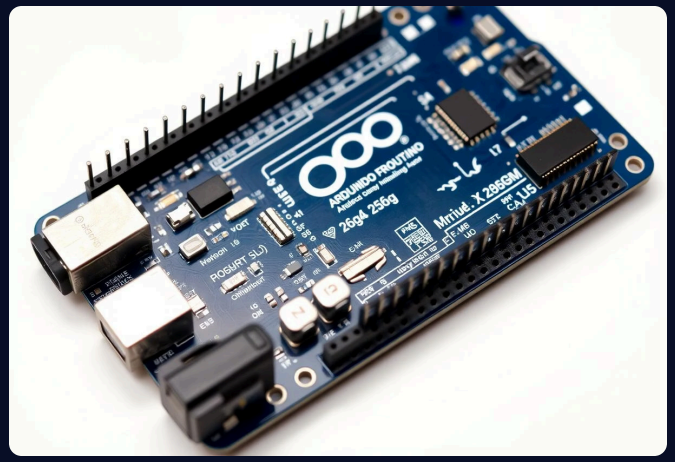




Hardware Components

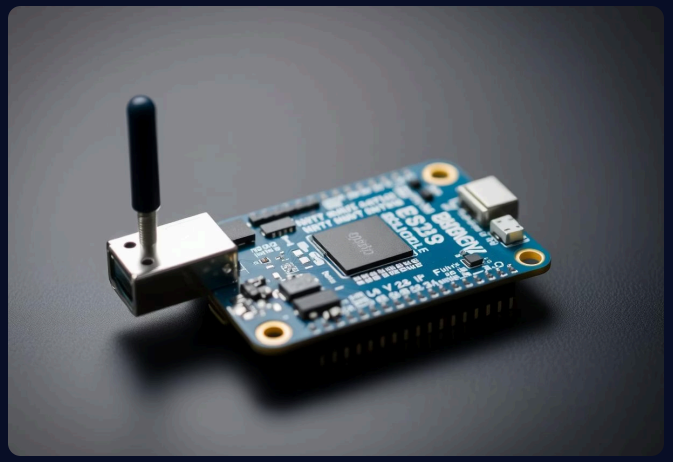
TinkerBlocks integrates essential hardware components to deliver seamless interaction and robust educational experiences for users.

Arduino Mega



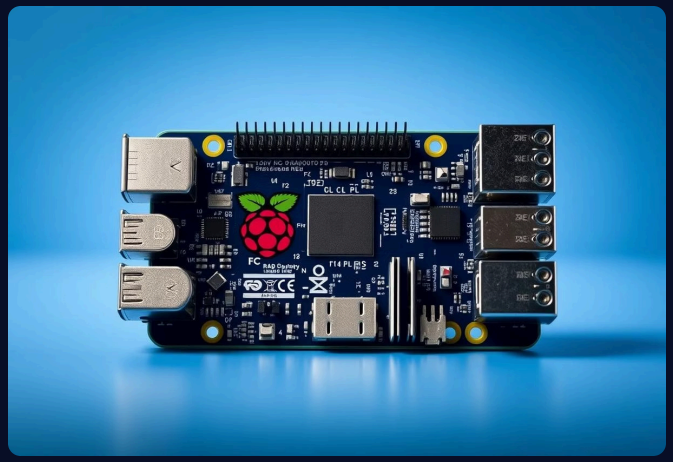
Controls the robot's motors and sensors. Its many input/output pins allow accurate and fast response to physical actions.

ESP32



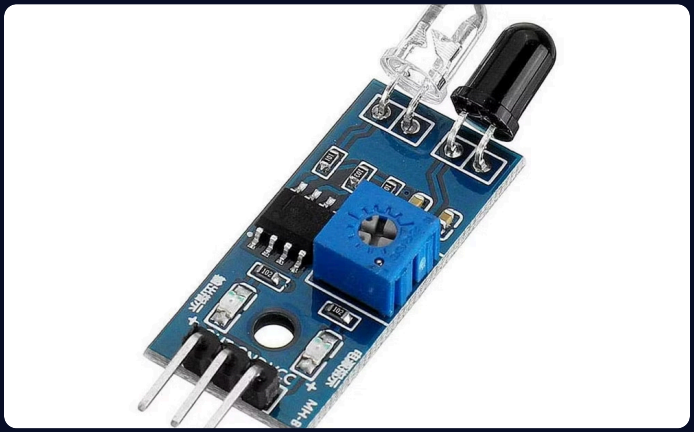
Enables wireless communication (Wi-Fi) between the robot and Raspberry Pi, sending commands and receiving sensor data.

Raspberry Pi



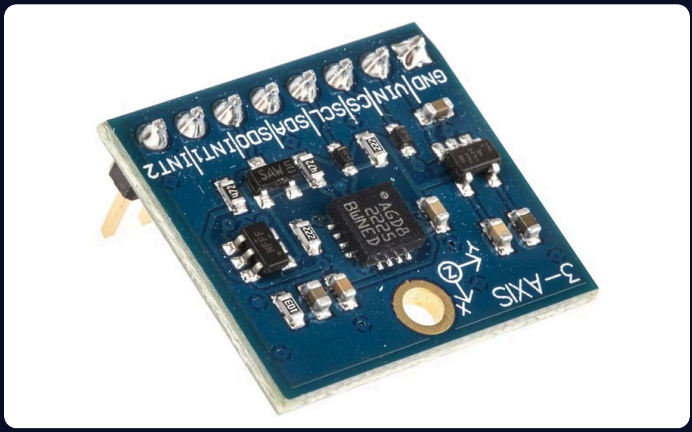
The brain for vision processing and programming logic. It recognises blocks and translates user code into commands for the robot.

Hardware Components



IR Sensor

Detects proximity and obstacles by emitting and receiving infrared light. Crucial black objects detecting in the ground.



Gyro Sensor

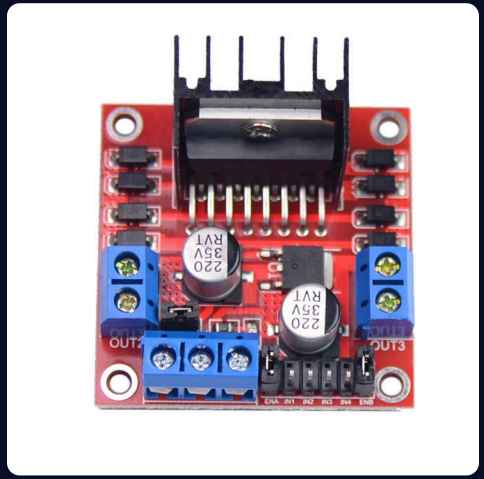
Measures angular velocity and orientation, allowing the robot to maintain balance and execute precise, controlled turns for navigation. yaw z

Ultrasonic Sensor

Uses sound waves to measure accurate distances to objects, providing essential data for dynamic obstacle detection.

Hardware Components

H-Bridge X 2



Drives the DC motors, controlling their speed and direction for robot movement.

DC Motors X 4

Provide the propulsion for the robot car, allowing it to move forward and backward.

Servo Motor 9g

Used for precise angular positioning, typically for controlling the pen.

Buzzer

Produces audible alerts or sounds, providing feedback to the user.

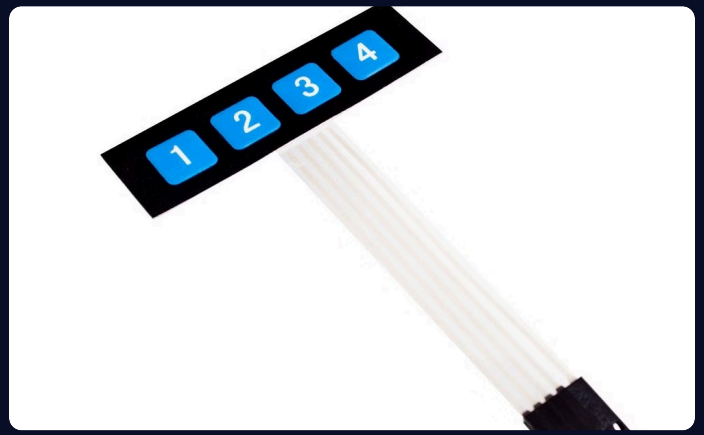
Hardware Components

Oak-D Camera

This camera focuses on vision, providing depth sensing and object detection to help the robot interact accurately with TinkerBlocks.

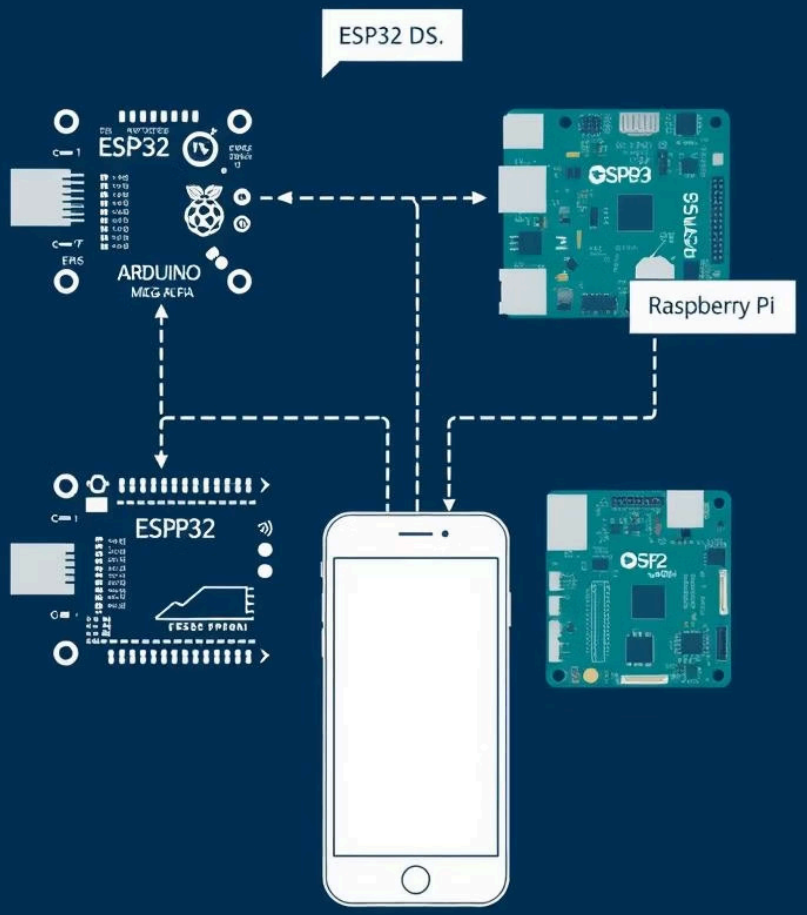
Lithium Batteries X 3

Powerful lithium-ion batteries give the robot long-lasting energy, allowing it to run longer without interruptions.



Keypad

A built-in number keypad lets users enter commands for running the program: run,stop.



Communication Protocols

Seamless communication is vital for TinkerBlocks to function effectively.

WebSocket

Enables real-time communication between Raspberry Pi and the App.

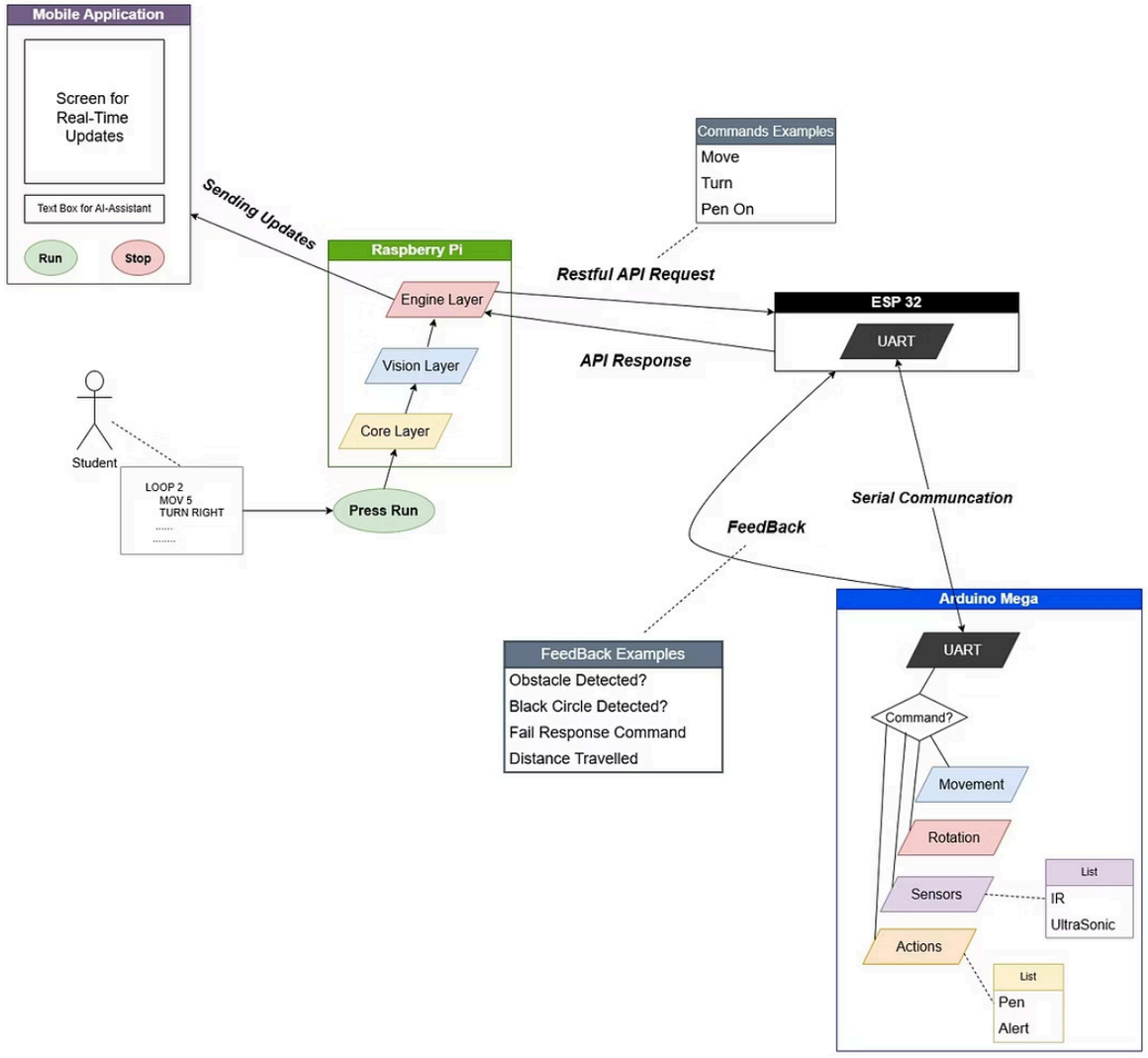
Serial UART

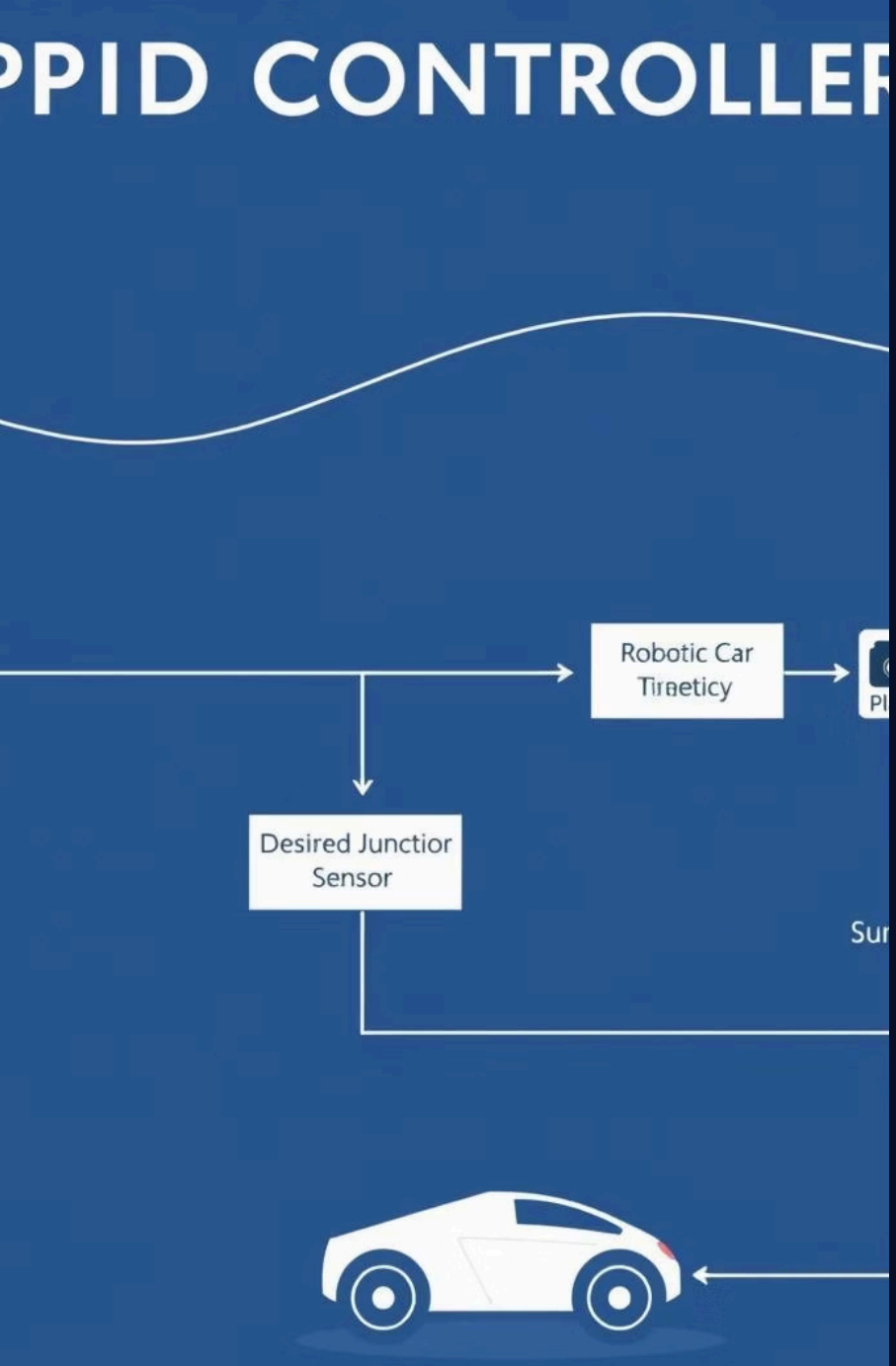
Facilitates data exchange between Arduino and ESP32 for robot control.

HTTP API

ESP32 exposes endpoints for remote command reception and control.

The Program Life Cycle





PID Controller: Precision in Motion

The PID controller is crucial for ensuring the robotic car's precise movement, from maintaining headings to executing accurate turns.

1 Proportional (P)

Corrects the car based on the current error, such as drift.

2 Integral (I)

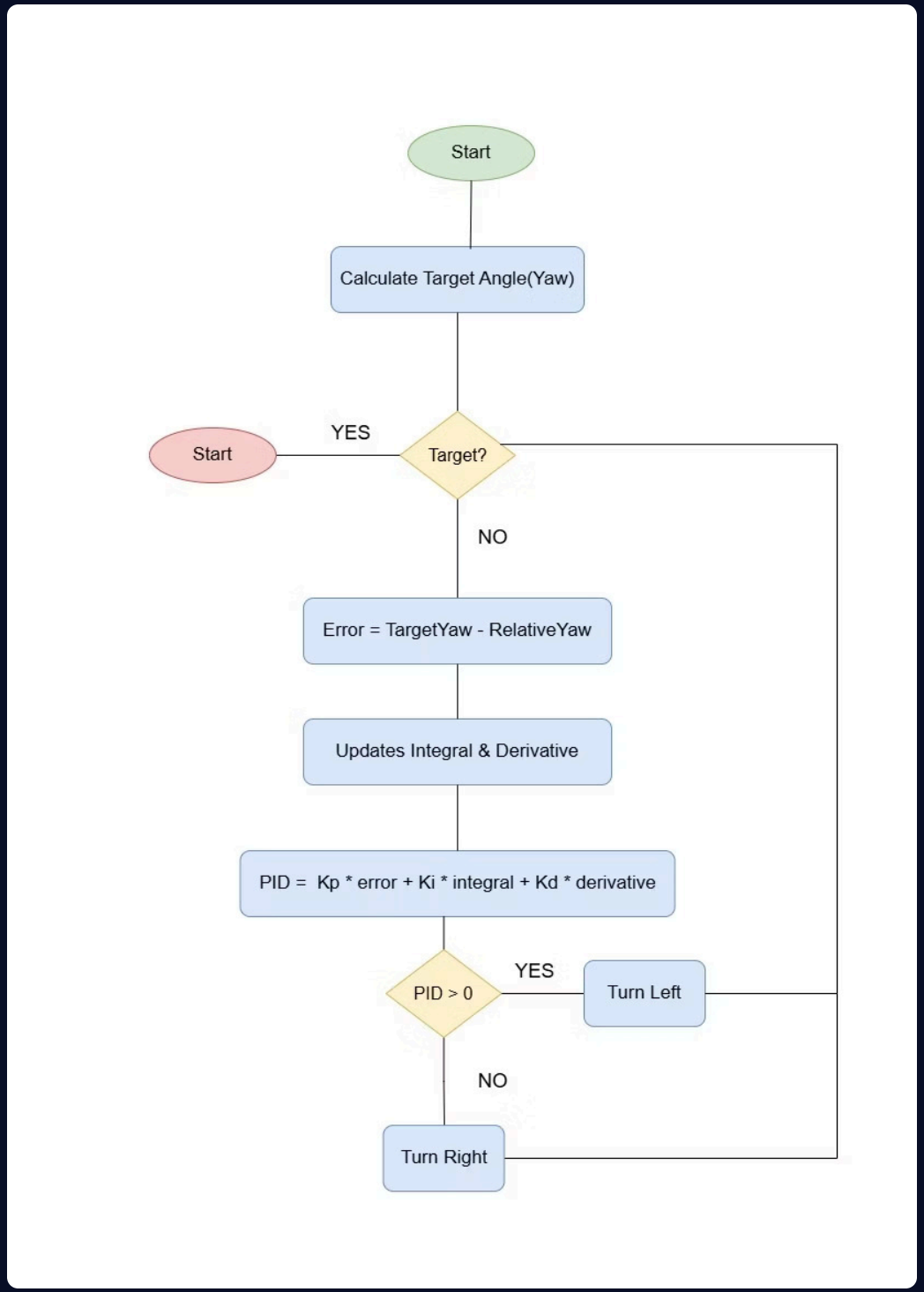
Fixes errors that build up over time to make the system more accurate.

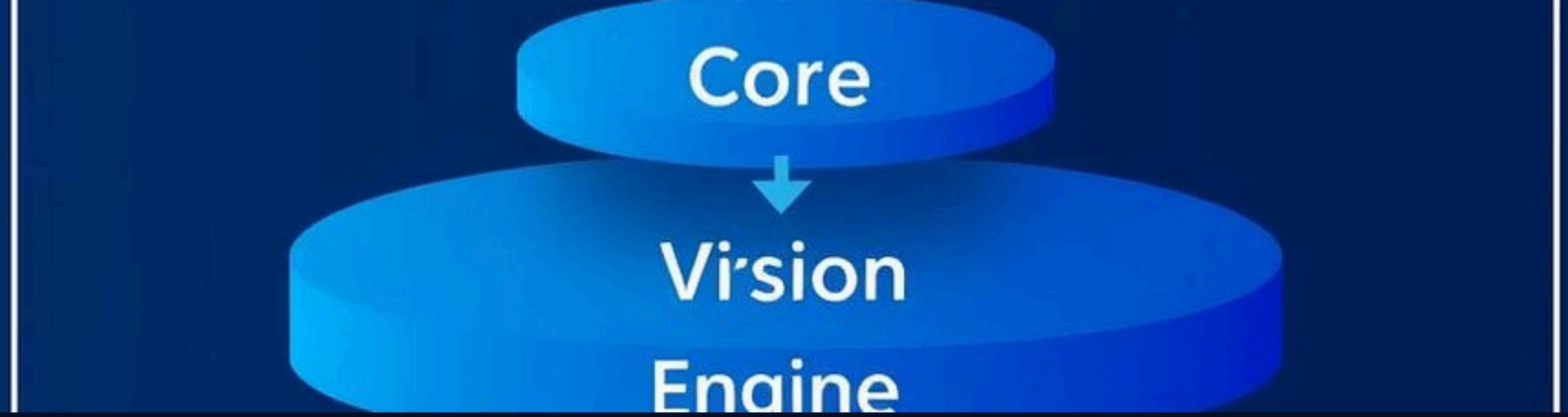
3 Derivative (D)

Predicts and prevents overshoot by observing the rate of change in error.

Example Usage: Adjusting motor speed for heading, accurate 90° turns, line-following stability.

PID Controller





Raspberry Pi Tasks

The Raspberry Pi's software operates through three distinct layers, each designed to handle specific functionalities crucial for the TinkerBlocks system.

Communication

Manages workflows, WebSocket server, and process controller.

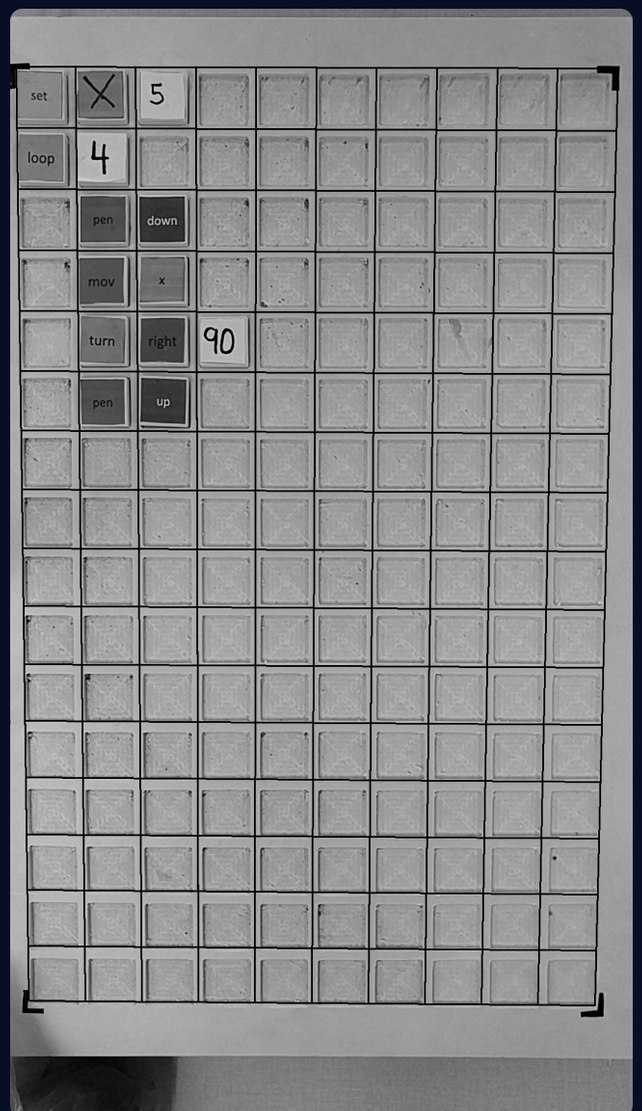
Vision & OCR

Handles image capture, perspective transformation, and OCR (EasyOCR) for block recognition.

Command Interpretation

Parses commands, executes programs, and manages variables and control flow.

Perspective Grid



Movement Commands

MOVE Command

Directs the car forward or backward, with options for specific distances or continuous motion until an obstacle.

Syntax:

```
MOVE
MOVE | distance
```

Examples:

```
MOVE     #Move forward till obstacle
MOVE | 5     #Move forward 5cm
MOVE | -3    #Move backward 3cm
MOVE | X     #X cm (variable)
```

TURN Command

Allows the robot to rotate precisely, supporting specified degrees or standard 90° turns left or right.

Syntax:

```
TURN | direction
TURN | degrees
TURN | direction | degrees
```

Examples:

```
TURN | LEFT  #Turn left 90°
TURN | RIGHT   #Turn right 90°
TURN | 45    #Turn right 45°
TURN | LEFT | 30  #Turn left 30°
```

Control Flow Commands

LOOP Command

Automate repetitive tasks. Repeat actions for a specified count or continuously, ideal for sequences like moving and turning multiple times.

```
LOOP | 5
    MOVE | 2
    TURN | RIGHT
```

WHILE Command

Execute commands conditionally, as long as a specified condition remains true. Useful for dynamic scenarios, for instance, moving until a sensor detects an obstacle.

```
WHILE | NOT OBSTACLE
    MOVE | 1
```

IF / ELSE Commands

Introduce decision-making logic. Execute specific code blocks based on whether a condition is true or false, enabling adaptive robot behaviour.

```
IF | OBSTACLE
    TURN | RIGHT
ELSE
    MOVE | 5
```

Variable Commands

SET Command

The SET command gives values to variables. It can set fixed numbers, use sensor readings, or do math to get new values.

Syntax:

```
SET | variable | value/expression
```

Parameters:

- **variable**: An alphanumeric string, such as **X**, **Y**, **COUNT**, or **SPEED**, serving as the variable's identifier.
- **value/expression**: The data to be assigned, which can be a direct numeric value, a sensor input, or a mathematical expression involving other variables.

Examples:

```
SET | X | 5              #Assigns the number 5 to variable X
SET | SPEED | 10         #Sets the SPEED variable to 10 units
SET | Y | DISTANCE       #Assigns the current DISTANCE sensor reading to Y
SET | Z | X + 2          #Sets Z to the value of X plus 2
SET | COUNT | COUNT + 1  #Increments the COUNT variable by 1
SET | FOUND | DISTANCE < 15 #Sets FOUND to true if DISTANCE is less than 15
```

Utility Commands

PEN_DOWN / PEN_UP

These commands manage the robot's pen. **PEN_DOWN** puts the pen down to draw while moving, and **PEN_UP** lifts it to stop drawing.

Syntax:

```
PEN_DOWN
PEN_UP
```

Example:

```
PEN_DOWN
MOVE | 5
PEN_UP
MOVE | 2
```

WAIT

The **WAIT** command pauses the program for a set number of seconds. It helps with timing and syncing actions with other events.

Syntax:

```
WAIT | seconds
```

Example:

```
MOVE | 1
WAIT | 5
MOVE | 1
```

ALERT_ON / ALERT_OFF

ALERT_ON turns on the buzzer, and **ALERT_OFF** turns it off. Useful for sounds like alerts or errors.

Syntax:

```
ALERT_ON
ALERT_OFF
```

Example:

```
ALERT_ON
WAIT | 2
ALERT_OFF
```

Programming Instructions: Operators

TinkerBlocks uses various operators to perform calculations, compare values, and manage logical conditions, essential for dynamic robot programming.

Arithmetic Operators

- `+`: Addition
- `-`: Subtraction
- `*`: Multiplication
- `/`: Division

Comparison Operators

- `<`: Less than
- `>`: Greater than
- `<=`: Less than or equal
- `>=`: Greater than or equal
- `==`: Equal to
- `!=`: Not equal to

Logical Operators

- **AND**: Both conditions must be true
- **OR**: Either condition must be true
- **NOT**: Inverts the condition

Sensor Integration Commands

DISTANCE

Reads object distance in centimetres via an ultrasonic sensor. Essential for precise navigation and mapping.

```
IF | DISTANCE < 15
   ALERT_ON
```

OBSTACLE Flag

A boolean value (**TRUE** if **DISTANCE** is less than 15cm). Simplifies collision avoidance logic.

```
WHILE | NOT OBSTACLE
   MOVE | 1
TURN | RIGHT
```

Black Detection

BLACK_DETECTED is **TRUE** if the IR sensor finds a black object. **BLACK_LOST** is its inverse, for line following.

```
IF | BLACK_DETECTED
   ALERT_ON
ELSE
   MOVE | 1
```

AI Assistant

The Assistant Module makes programming fun and easy for kids. It takes a child’s question, its essentially a LangChain React Agent that choose tools (functions) to perform actions, like reading blocks, running the robot, and give clear answers to the student.

Read Blocks

Uses OCR to accurately interpret physical programming blocks, connecting actions to on-screen logic.

Execute Program

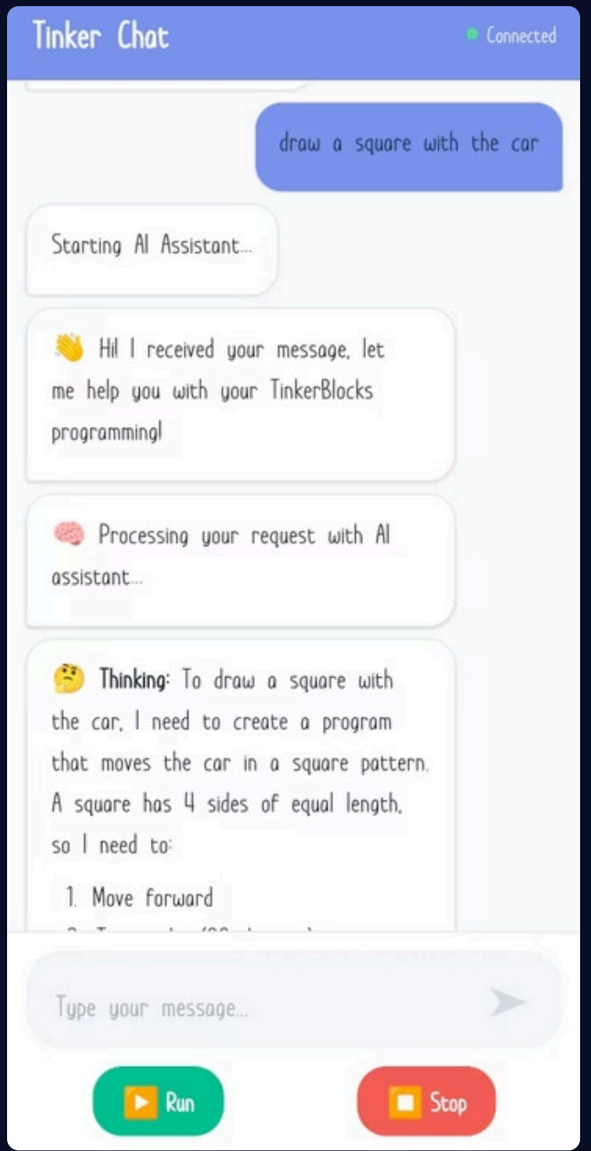
Runs code directly on the robot car, providing immediate visual feedback for children's learning.

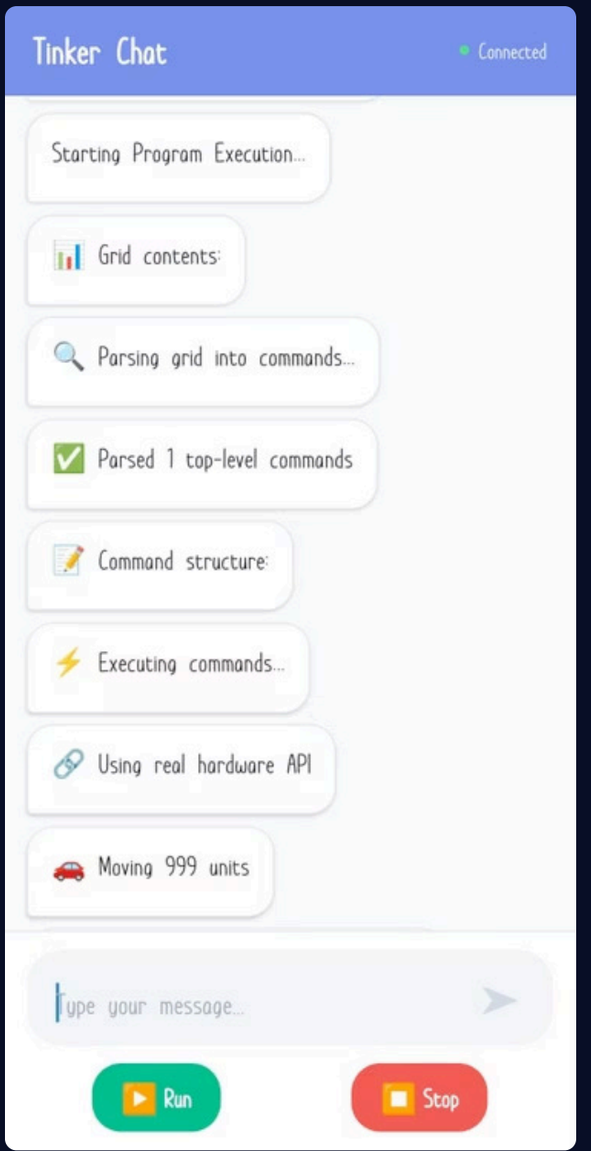
Think

Shares the AI's reasoning, demystifying complex concepts through step-by-step guidance.

Answer

Provides clear, encouraging explanations and guidance, acting as a friendly programming mentor.





Future Work & Innovations

Advanced Sensor Integration

Integrate sophisticated sensors for richer environmental interaction, enabling complex robot behaviours and data collection.

Dual Robot Control

Adds control for a second robot, allowing them to work together and perform more complex tasks in sync.

Enhanced Debugging Tools

Introduce visual debuggers and step-by-step execution to simplify program troubleshooting and accelerate learning.

Limitations & Challenges

Fixed Grid Size

The system uses a fixed 16x10 board, which limits how long and complex programs can be.

Slow Block Recognition

On average, the system takes about 7 seconds to recognize blocks, slowing down learning and testing.

Dependence on External Camera

The system needs a calibrated OAK-D overhead camera, making it dependent on specific hardware.

Thank You

Demo

Google Docs

Demo.mp4