



TinkerBlocks: Code, build, and drive!

An Educational Tool for Teaching Programming Concepts Through
Physical Blocks

Amr Badran
Izzat Alsharif

Supervisor: Dr. Ashraf Armoush

Department: Computer Engineering
Faculty: Engineering and Information Technology
University: An-Najah National University

June 10, 2025

ACKNOWLEDGMENT

We would like to express our sincere gratitude to our project supervisor **Dr. Ashraf Armoush** for his invaluable guidance, continuous support, and encouragement throughout this project. His expertise and insights have been instrumental in shaping this work and helping us overcome various technical challenges.

We are grateful to the Computer Engineering Department at An-Najah National University for providing us with the necessary resources and facilities to complete this project successfully.

We would also like to thank our families and friends for their unwavering support and patience during the development of this project. Their encouragement motivated us to push through difficult times and achieve our goals.

Finally, we acknowledge all the researchers and educators whose work in the fields of educational technology, computer vision, and robotics inspired and informed our approach to this project.

Amr Badran
Izzat Alsharif
June 2025

ABSTRACT

Learning programming can be challenging for children due to abstract concepts and complex syntax. TinkerBlocks addresses this challenge by introducing a tangible programming approach where children arrange physical blocks on a grid to control a robotic car, making programming concepts fun and engaging.

This project developed an integrated system combining computer vision, robotics, and educational design. Children place programming blocks representing commands like movement, loops, and conditionals on a physical grid. A camera captures the arrangement, computer vision algorithms recognize the blocks and their positions, and the system translates this into executable code that controls a robotic car.

The implementation consists of three main components: a Raspberry Pi system that handles image processing and command interpretation, a robotic car equipped with sensors and actuators for autonomous movement, and a mobile application for more engaging features. The car can execute complex programs including loops, conditional statements, and sensor-based decisions while providing visual feedback through drawing capabilities.

Evaluation demonstrates successful block recognition with high accuracy, precise car movement control, and effective execution of programming concepts. Children can create programs ranging from simple movement sequences to complex algorithms involving variables and conditional logic.

TinkerBlocks successfully bridges the gap between abstract programming concepts and tangible interaction, providing an intuitive platform for programming education.

Keywords: Educational Technology, Computer Vision, Robotics, Programming Education

Contents

1 INTRODUCTION	1
1.1 Background and Motivation	1
1.2 Project Overview	1
1.3 Problem Statement	2
1.4 Proposed Solution	2
1.5 Objectives	3
1.5.1 Educational Objectives	3
1.5.2 Technical Objectives	3
1.6 Scope and Contributions	3
1.6.1 Project Scope	3
1.6.2 Key Contributions	4
1.7 Report Structure	4
2 LITERATURE REVIEW	5
2.1 Educational Robotics in Programming Education	5
2.2 Tangible Programming Approaches	5
2.2.1 Evolution of Tangible Programming	6
2.2.2 Current Tangible Programming Systems	6
2.2.3 Advantages and Limitations	6
2.3 Computer Vision in Educational Technology	6
2.4 Visual Programming Languages	7
2.5 Integration Challenges in Educational Systems	7
2.6 Research Gaps and Opportunities	8
2.7 TinkerBlocks in Context	8
3 SYSTEM DESIGN AND IMPLEMENTATION	9
3.1 System Overview	9
3.2 System Architecture	11
3.2.1 Distributed Component Architecture	11
3.2.2 Communication Flow	11
3.3 Hardware Design and Implementation	12
3.3.1 Robotic Car	12

3.3.2	Programming Grid Interface	14
3.3.3	Control Station	16
3.4	Software Architecture and Implementation	16
3.4.1	Core Module	16
3.4.2	Vision Module	17
3.4.3	Engine Module	18
3.5	Arduino Firmware Implementation	20
3.5.1	Class-Based Architecture	20
3.5.2	Advanced Movement Algorithms	21
3.5.3	Serial API Implementation	23
3.6	Mobile Application Implementation	23
3.6.1	React Native Architecture	23
3.6.2	User Interface Design	23
3.7	Development Methodology	24
3.7.1	Iterative Development Process	24
3.7.2	Testing and Validation	24
3.8	System Integration and Deployment	25
3.8.1	Communication Protocols	25
3.8.2	Deployment Considerations	26
4	RESULTS AND CONCLUSION	27
4.1	System Demonstration	27
4.1.1	Successful System Integration	27
4.1.2	Programming Language Capabilities	28
4.2	Technical Achievements	29
4.2.1	Computer Vision Success	29
4.2.2	Arduino Firmware Excellence	29
4.2.3	Communication Architecture	30
4.2.4	Mobile Application Success	31
4.3	Educational Impact and Value	31
4.3.1	Learning Benefits	31
4.3.2	Accessibility and Engagement	32
4.4	Challenges and Limitations	32
4.4.1	Technical Challenges	32
4.4.2	Educational Considerations	33
4.5	Future Work and Improvements	34
4.5.1	Technical Enhancements	34
4.5.2	Educational Enhancements	34
4.6	Broader Impact and Significance	35

4.6.1	Contribution to Educational Technology	35
4.6.2	Potential for Widespread Adoption	36
4.7	Conclusion	36
4.7.1	Key Achievements	36
4.7.2	Educational Impact	37
4.7.3	Technical Contribution	37
4.7.4	Future Potential	38
4.7.5	Final Reflection	38
A	API DOCUMENTATION AND COMMAND REFERENCE	39
A.1	Arduino Serial API	39
A.1.1	Movement Commands	39
A.1.2	Sensor Commands	41
A.1.3	Pen Control Commands	41
A.2	ESP32 HTTP API	42
A.2.1	HTTP Endpoints	42
A.3	WebSocket Communication Protocol	42
A.3.1	Client Commands	43
A.3.2	Server Responses	44
A.4	Programming Language Reference	45
A.4.1	Movement Commands	45
A.4.2	Control Flow Examples	45
A.4.3	Variable Operations	46
A.4.4	Sensor Integration	46
A.4.5	Drawing Commands	47
A.5	Error Codes and Troubleshooting	47
A.5.1	Common Error Codes	47
A.5.2	Troubleshooting Guide	48

Chapter 1

INTRODUCTION

1.1 Background and Motivation

In an increasingly digital world, programming skills have become essential for students across all disciplines. However, traditional programming education often presents significant barriers for young learners. Abstract concepts, complex syntax, and screen-based interfaces can make programming feel disconnected from the physical world that children naturally explore and understand.

Educational technology has evolved to address these challenges, with hands-on learning approaches showing particular promise for engaging young minds. Physical manipulation and immediate visual feedback help children grasp abstract concepts more effectively than traditional lecture-based methods. This has led to growing interest in educational tools that bridge the gap between digital programming and physical interaction.

While various educational programming tools exist, most still rely on screen-based interfaces or require significant technical setup. There remains a need for intuitive systems that allow children to program through natural physical interaction while providing immediate, engaging feedback through real-world actions.

1.2 Project Overview

TinkerBlocks is an integrated educational system that transforms programming education through physical interaction. Children arrange programming blocks on a physical grid to control a robotic car, making programming concepts tangible and immediately visible through car movement and drawing.

The system consists of four main components working together:

- Physical programming blocks representing commands, loops, and conditions
- Computer vision system that recognizes block arrangements in real-time

- Intelligent robotic car that executes programs with sensor feedback
- Mobile application for more engaging features

This integration creates a complete learning environment where children can progress from simple movement commands to complex algorithms involving variables, loops, and sensor-based decision making.

1.3 Problem Statement

Current programming education tools for children face several key challenges:

1. **Abstract Learning Environment:** Programming concepts remain disconnected from physical reality, making them difficult for young learners to grasp
2. **Limited Immediate Feedback:** Most tools provide only visual feedback on screens rather than real-world interaction
3. **Complex Setup Requirements:** Many educational robotics solutions require significant technical knowledge to operate
4. **Fragmented Learning Experience:** Existing tools often focus on single aspects (either programming OR robotics) rather than integrated learning
5. **Scalability Issues:** Most solutions are designed for individual use rather than classroom environments

1.4 Proposed Solution

TinkerBlocks addresses these challenges through an innovative approach that combines multiple technologies into a cohesive educational experience:

Physical Programming Interface: Children arrange blocks representing programming commands on a grid, making abstract concepts concrete and manipulable.

Real-time Computer Vision: Advanced image processing recognizes block arrangements and translates them into executable programs without requiring manual input.

Intelligent Robotic Execution: A sophisticated car equipped with sensors executes programs while providing immediate visual feedback through movement and drawing capabilities.

Integrated System Architecture: All components communicate seamlessly, creating a unified experience from block placement to program execution.

This approach allows children to learn programming concepts naturally through physical manipulation while seeing immediate results in the real world.

1.5 Objectives

The primary objectives of this project are:

1.5.1 Educational Objectives

1. Create an intuitive programming interface accessible to children without prior technical knowledge
2. Provide immediate visual feedback to reinforce learning of programming concepts
3. Support progressive learning from basic commands to advanced programming constructs
4. Enable collaborative programming activities in classroom settings

1.5.2 Technical Objectives

1. Develop accurate computer vision algorithms for real-time block recognition
2. Implement robust communication between multiple system components
3. Create a comprehensive command interpreter supporting loops, conditionals, and variables
4. Design reliable robotic hardware capable of precise movement and sensor integration

1.6 Scope and Contributions

1.6.1 Project Scope

This project delivers a complete working system including:

- Hardware design and construction of the robotic car with sensor integration
- Computer vision pipeline for block recognition and grid mapping
- Command interpreter supporting complex programming constructs
- Mobile application for more engaging features
- Integration protocols enabling seamless communication between all components
- Testing and validation in real-world educational scenarios

1.6.2 Key Contributions

- Integration of computer vision, robotics, and mobile technology for educational purposes
- Development of an accessible physical programming interface for young learners
- Creation of a scalable system architecture suitable for classroom deployment
- Demonstration of effective tangible programming for computational thinking education

1.7 Report Structure

This report documents the complete development and implementation of the TinkerBlocks system:

Chapter 2 reviews existing educational technology and programming tools, positioning TinkerBlocks within the current landscape of educational robotics.

Chapter 3 presents the system architecture and design decisions, including hardware specifications, software architecture, and integration strategies.

Chapter 4 details the implementation of all system components, from computer vision algorithms to robotic control systems.

Chapter 5 concludes with project achievements, educational impact, and recommendations for future development.

Appendices provide technical documentation, code examples, and detailed system specifications for implementation reference.

Chapter 2

LITERATURE REVIEW

2.1 Educational Robotics in Programming Education

Educational robotics has become a cornerstone of modern STEM education, offering students hands-on experience with programming concepts through immediate physical feedback. Early pioneers like Seymour Papert's Logo turtle demonstrated that children could learn programming through controlling physical objects, establishing the foundation for today's educational robotics platforms.

Modern systems like LEGO Mindstorms, VEX Robotics, and Arduino-based platforms have made robotics accessible to classrooms worldwide. These platforms typically combine programmable controllers with sensors and actuators, allowing students to build and program robots for various tasks. Research consistently shows that robotics education improves problem-solving skills and makes abstract programming concepts more concrete through visual and physical feedback.

However, most current educational robotics platforms still rely on traditional screen-based programming interfaces. Students must learn specific programming languages or visual programming environments before they can control their robots, creating a barrier between the physical robot and the programming logic.

2.2 Tangible Programming Approaches

Tangible programming represents a shift away from screen-based interfaces toward physical manipulation of programming concepts. This approach recognizes that children learn effectively through hands-on interaction with physical objects, making abstract programming concepts more accessible.

2.2.1 Evolution of Tangible Programming

Early examples include simple programmable toys like Big Trak, which allowed children to program movement sequences using physical buttons. More sophisticated systems emerged from research institutions, with MIT's AlgoBlock system demonstrating that children could understand programming logic through physical block arrangements long before mastering text-based programming.

2.2.2 Current Tangible Programming Systems

Several commercial and research systems have explored tangible programming:

Cubetto uses colored wooden blocks placed in slots to program a robot's movement. While intuitive for young children, it's limited to basic movement commands and lacks support for complex programming constructs like loops and conditionals.

KIBO allows children to build robots using construction pieces and program them with wooden blocks. The system supports some programming concepts but requires scanning blocks with a barcode reader, interrupting the natural flow of programming.

Osmo Coding uses a tablet camera to detect physical blocks, but relies on screen-based feedback and is limited to predefined programming challenges rather than open-ended exploration.

Code & Go takes a board game approach to programming logic but lacks robotic execution, limiting the immediate feedback that makes programming concepts concrete.

2.2.3 Advantages and Limitations

Tangible programming offers clear benefits: it eliminates syntax errors, supports collaborative work, and makes programming logic visible and manipulable. Children can focus on computational thinking rather than remembering specific commands or dealing with typing errors.

However, existing systems face significant limitations. Most support only basic programming concepts, lack scalability for complex programs, and often require manual processes (like barcode scanning) that interrupt the programming flow. Additionally, many systems separate the programming interface from the execution environment, reducing the immediate feedback that makes robotics education effective.

2.3 Computer Vision in Educational Technology

Computer vision has enabled new possibilities in educational technology by allowing systems to interpret physical student interactions automatically. In tangible programming

systems, computer vision serves as the crucial bridge between physical block arrangements and digital program execution.

Early computer vision applications in education focused on gesture recognition and simple object detection. Modern systems can perform sophisticated real-time analysis of complex scenes, enabling more natural and fluid interactions between students and educational technology.

However, educational applications of computer vision face unique challenges. Classroom environments have variable lighting conditions, multiple users, and changing physical setups. Systems must be robust enough to work reliably in these conditions while being simple enough for teachers and students to use without technical expertise.

2.4 Visual Programming Languages

Visual programming languages like Scratch, Blockly, and Alice have transformed programming education by replacing text-based syntax with drag-and-drop interfaces. These tools have demonstrated that visual approaches can effectively teach programming concepts while eliminating many barriers that frustrate beginning programmers.

Scratch, developed at MIT, has been particularly influential in showing how visual programming can engage children in creating interactive stories, games, and animations. Research has validated the effectiveness of block-based visual programming for teaching computational thinking skills including decomposition, pattern recognition, abstraction, and algorithm design.

However, visual programming languages still require screen-based interaction and can become unwieldy for complex programs. They also lack the physical manipulation that research shows enhances learning for many children, particularly younger learners who benefit from hands-on interaction.

2.5 Integration Challenges in Educational Systems

Most educational technology tools excel in specific areas but struggle with integration. Robotics platforms provide excellent physical feedback but often require complex programming interfaces. Visual programming languages make programming logic accessible but lack physical execution. Tangible programming systems offer intuitive interaction but often support only limited programming concepts.

This fragmentation creates challenges for educators who must choose between different tools for different aspects of programming education. Students may learn programming logic in one system, robotics in another, and struggle to connect these concepts into a unified understanding of how programming controls physical systems.

2.6 Research Gaps and Opportunities

Current educational programming tools leave several important gaps:

Seamless Integration: Few systems successfully combine tangible programming interfaces with sophisticated robotic execution and complex programming concepts in a single, unified experience.

Real-time Feedback: Most tangible programming systems require manual steps (scanning, uploading, etc.) that interrupt the natural flow from programming to execution.

Scalable Complexity: Many systems work well for simple concepts but cannot scale to support the full range of programming constructs needed for comprehensive programming education.

Classroom Practicality: Research prototypes often require controlled conditions or technical expertise that make them impractical for real classroom use.

2.7 TinkerBlocks in Context

TinkerBlocks addresses these gaps by combining insights from educational robotics, tangible programming, and computer vision into an integrated system designed for practical classroom use.

Unlike existing tangible programming systems that support only basic commands, TinkerBlocks implements a complete programming language with loops, conditionals, variables, and sensor integration. The computer vision system provides real-time block recognition without requiring manual scanning or setup, enabling natural programming flow from block arrangement to robot execution.

The system's modular architecture allows it to scale from simple movement commands for young children to complex algorithms suitable for older students, addressing the full spectrum of programming education needs within a single platform.

By integrating multiple technologies into a cohesive educational experience, TinkerBlocks demonstrates how modern educational technology can overcome the limitations of existing approaches while maintaining the hands-on, engaging qualities that make learning effective and enjoyable.

Chapter 3

SYSTEM DESIGN AND IMPLEMENTATION

3.1 System Overview

TinkerBlocks is a comprehensive educational system that integrates computer vision, robotics, and mobile technology to create an intuitive programming learning environment. Children arrange physical programming blocks on a grid, and the system uses real-time computer vision to recognize these arrangements and execute them on a robotic car.

The system consists of four main integrated components:

1. **Physical Programming Interface:** Grid board with blocks representing programming commands
2. **Raspberry Pi Control System:** Computer vision processing and command interpretation
3. **Robotic Car:** Arduino-controlled vehicle with sensors and actuators
4. **Mobile Application:** Real-time communication and system monitoring

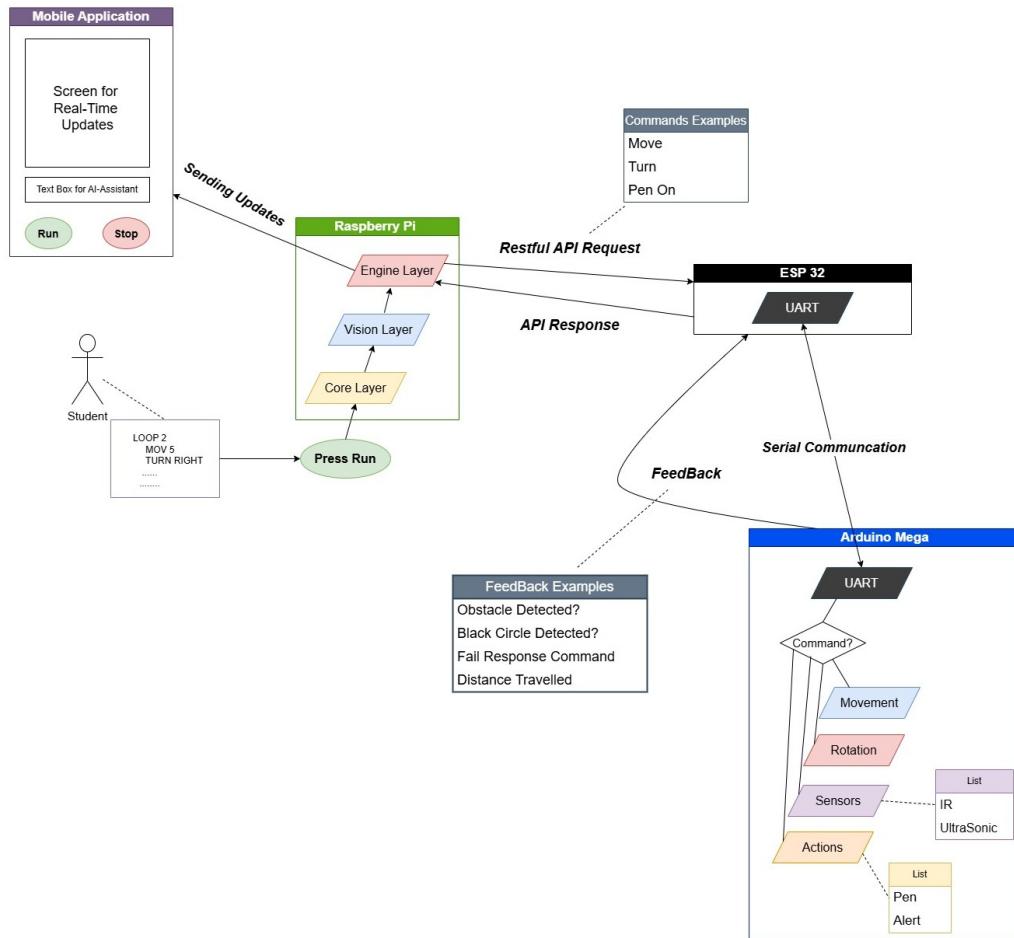


Figure 3.1: Complete System Communication Flow and Component Integration

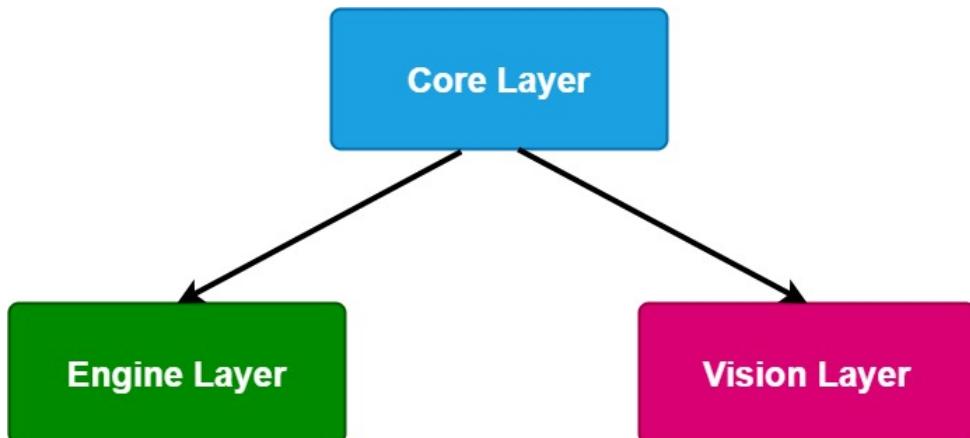


Figure 3.2: Raspberry Pi Software Architecture - Core, Vision, and Engine Modules

3.2 System Architecture

The TinkerBlocks architecture follows a distributed, modular design that enables real-time communication between multiple specialized components.

3.2.1 Distributed Component Architecture

Raspberry Pi Hub: Serves as the central intelligence, running three specialized modules:

- **Core Module:** WebSocket server, process control, and configuration management
- **Vision Module:** Camera capture, OCR processing, and grid mapping
- **Engine Module:** Command interpretation and program execution logic

Arduino Car Controller: Handles direct hardware control with sophisticated firmware:

- Class-based architecture for motor control, sensor management, and movement algorithms
- Real-time sensor processing and feedback control systems
- Serial API for communication with ESP32 bridge

ESP32 WiFi Bridge: Provides wireless communication between Raspberry Pi and Arduino:

- HTTP REST API server for remote car control
- Serial communication bridge to Arduino
- JSON protocol translation between HTTP and serial interfaces

Mobile Application: React Native interface for user interaction:

- WebSocket communication with Raspberry Pi
- Real-time chat interface and system control
- Cross-platform compatibility for iOS and Android

3.2.2 Communication Flow

The system operates through a complete workflow:

1. Children arrange programming blocks on the physical grid
2. Camera captures the grid arrangement

3. Computer vision processes the image and extracts block positions and text
4. Command interpreter translates the visual program into executable commands
5. Commands are sent via WebSocket to ESP32, then via serial to Arduino
6. Arduino executes commands while providing sensor feedback
7. Mobile app displays real-time status and execution progress

3.3 Hardware Design and Implementation

3.3.1 Robotic Car

The robotic car serves as the primary execution platform, designed for precision, reliability, and educational engagement.



Figure 3.3: TinkerBlocks Robotic Car - Complete Assembly

Mechanical Design

- **Chassis:** Custom wooden frame (10cm × 20cm) for durability and cost-effectiveness
- **Propulsion:** Four DC motors with differential steering (tank-style turning)
- **Motor Control:** Two H-bridge modules controlling motor pairs
- **Weight Distribution:** Balanced design for stable movement and turning

Electronic Systems

Primary Controller - Arduino Mega 2560:

- Manages all sensors, motors, and actuators

- Implements sophisticated movement algorithms with sensor feedback
- Provides comprehensive serial API for external control
- Handles real-time control loops for precise movement

WiFi Communication - ESP32:

- Exposes HTTP REST API for wireless car control
- Bridges between WiFi commands and Arduino serial interface
- Handles JSON protocol translation and error management
- Provides reliable communication with timeout and retry mechanisms

Power Management:

- Three 3.7V lithium-ion batteries providing 11.1V total
- Voltage regulator stepping down to 5V for Arduino and logic circuits
- Approximately 2-3 hours of continuous operation
- Battery monitoring and protection circuits

Sensor Integration

Navigation and Orientation:

- **MPU-6050:** 6-axis gyroscope and accelerometer for precise orientation tracking
- **Yaw Correction:** Gyroscope-based straight-line movement compensation
- **PID Control:** Feedback-controlled rotation for accurate turning angles

Environmental Sensing:

- **HC-SR04 Ultrasonic Sensor:** Front-mounted obstacle detection with configurable thresholds
- **IR Sensors:** Black line detection for boundary sensing and target recognition
- **Real-time Processing:** Continuous sensor monitoring during movement

Interactive Features:

- **Servo-Controlled Pen:** Precision drawing mechanism for creative programming tasks
- **Position Tracking:** Coordinate system for drawing and navigation
- **Path Recording:** Movement history for visualization and debugging

3.3.2 Programming Grid Interface

Physical Structure

- **Grid Configuration:** 16 columns × 10 rows (configurable for different complexity levels)
- **Block Placement:** Physical holes for precise block positioning and stability
- **Material:** Durable wooden construction suitable for classroom use
- **Portability:** Lightweight design for easy setup and storage

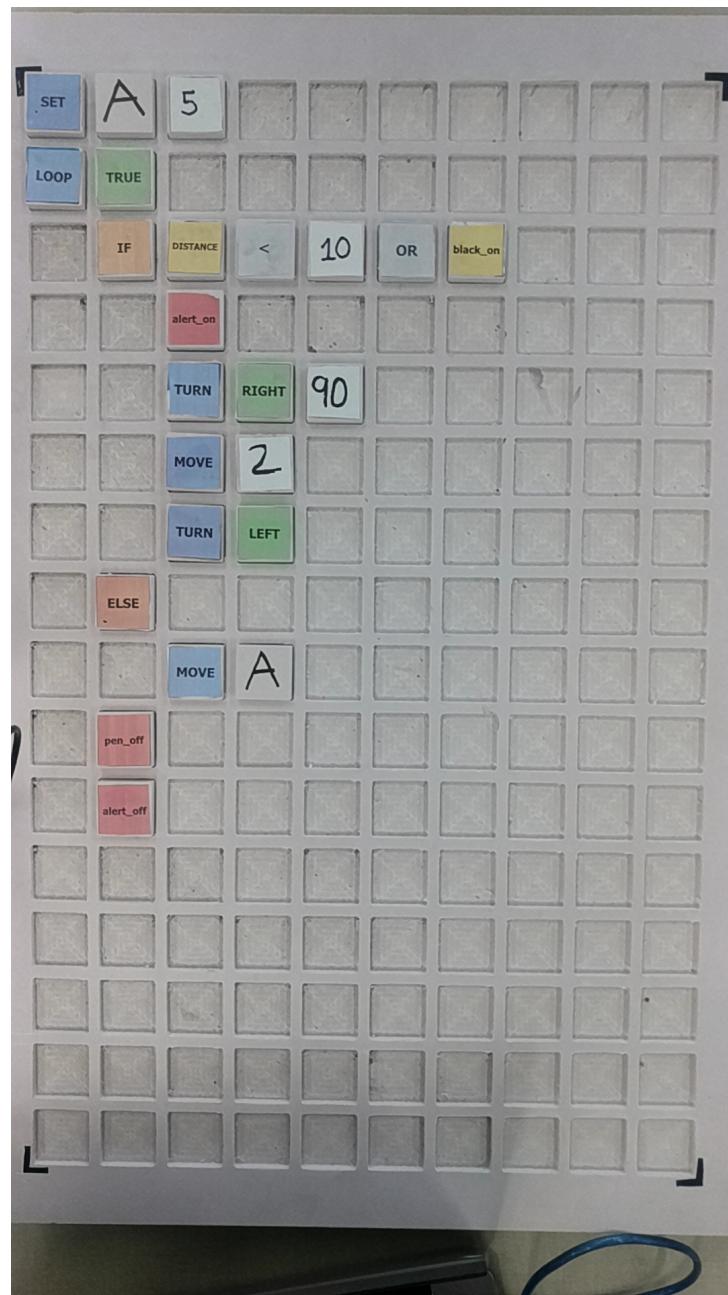


Figure 3.4: Programming Grid with Sample Block Arrangement

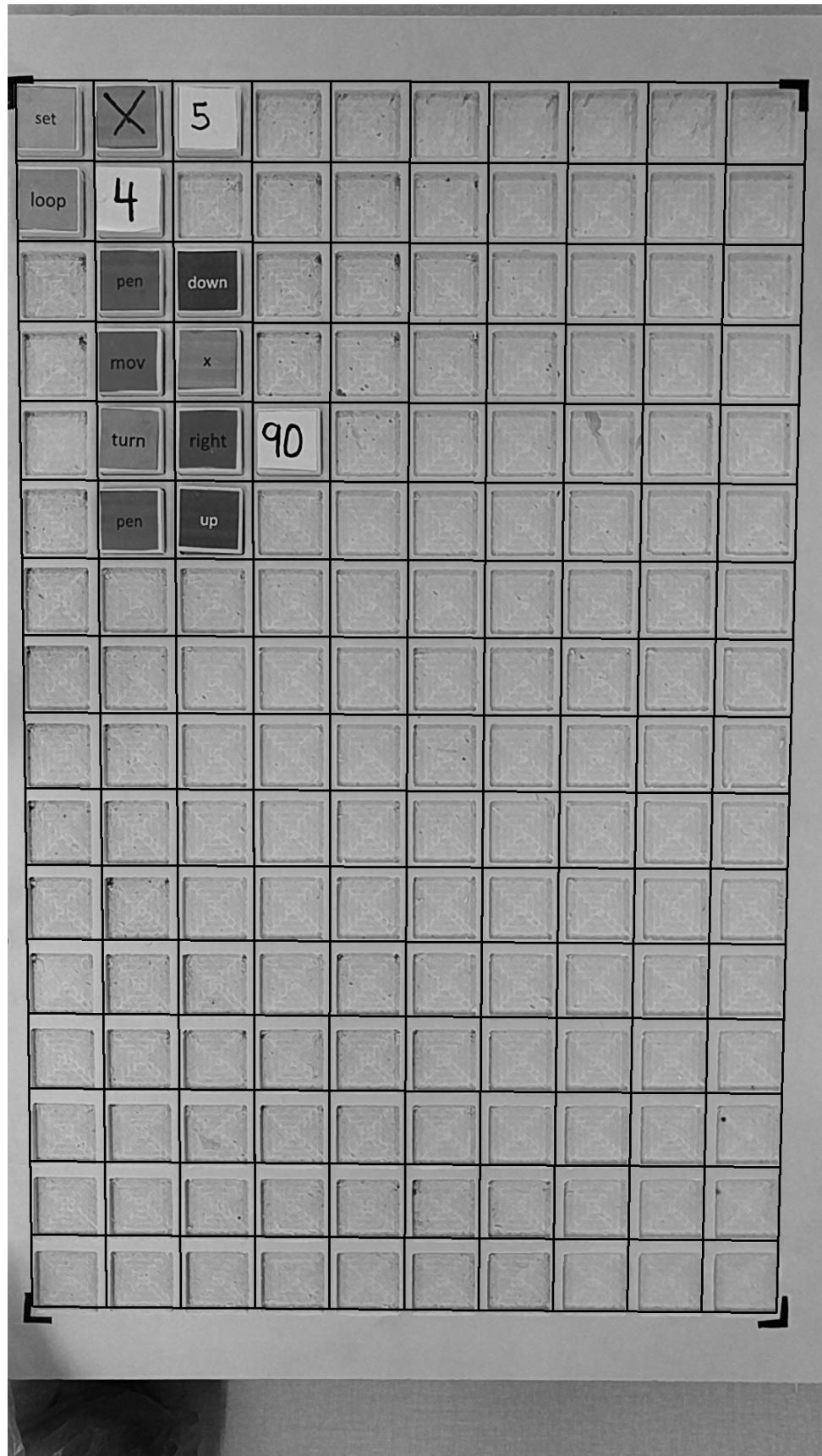


Figure 3.5: Computer Vision Grid Detection and Perspective Transformation

Computer Vision System

Camera Setup:

- **OAK-D Camera:** High-resolution overhead mounting for complete grid visibility
- **Perspective Correction:** Automatic corner detection and transformation
- **Lighting Optimization:** Controlled conditions for consistent OCR performance
- **Real-time Processing:** Live image capture and processing capabilities

3.3.3 Control Station

Raspberry Pi 4 Specifications:

- **Processing Power:** 2GB RAM for computer vision and real-time processing
- **Storage:** 32GB microSD card for system software and data
- **Connectivity:** WiFi for car communication, USB for camera interface
- **Display Interface:** Tablet integration for system status and control

3.4 Software Architecture and Implementation

3.4.1 Core Module

The core module provides foundational infrastructure with zero dependencies on other components, implementing clean architecture principles.

WebSocket Server Implementation

- **Asynchronous Architecture:** Built on asyncio for concurrent client handling
- **JSON Protocol:** Structured command and response messaging
- **Broadcast Support:** System-wide notifications to all connected clients
- **Connection Management:** Automatic reconnection and error recovery

Process Controller

- **Workflow Framework:** Generic execution pattern supporting any async function
- **Cancellation Support:** Clean cancellation through callback mechanisms
- **Progress Reporting:** Real-time status updates via message callbacks
- **Return Value Chaining:** Support for workflow data passing and composition

Configuration Management

- **Pydantic Models:** Type-safe configuration with automatic validation
- **Environment Variables:** Flexible deployment configuration
- **Immutable Design:** Thread-safe configuration access
- **Default Values:** Comprehensive fallback configuration

3.4.2 Vision Module

The vision module handles the complete pipeline from image capture to structured grid data.

Image Processing Pipeline

Multi-Source Capture:

- **Local Capture:** Direct OpenCV camera access for development
- **Remote Capture:** Client-server architecture for Raspberry Pi deployment
- **File Processing:** Static image testing and validation capabilities
- **Format Optimization:** Automatic rotation, scaling, and format conversion

Grid Detection and Transformation:

- **Perspective Correction:** Homography transformation using configurable corner points
- **Grid Cell Mapping:** Precise boundary calculation for block position detection
- **Adaptive Scaling:** Support for different grid sizes and camera positions
- **Quality Validation:** Image quality assessment and optimization

OCR Processing

EasyOCR Integration:

- **GPU Acceleration:** Hardware acceleration when available for improved performance
- **Confidence Scoring:** Recognition quality assessment for filtering
- **Bounding Box Detection:** Precise text localization within grid cells
- **Multi-language Support:** Optimized for English with extensibility

OCR2Grid Mapping

Spatial Analysis:

- **Grid Association:** Geometric analysis to map detected text to specific grid positions
- **Confidence Filtering:** Elimination of false positives based on recognition confidence
- **Multi-word Support:** Handling of complex commands spanning multiple detection regions
- **Empty Cell Detection:** Proper handling of unoccupied grid positions

3.4.3 Engine Module

The engine module implements a comprehensive programming language interpreter using modern design patterns.

Command System Architecture

Command Registry and Factory:

- **Dynamic Registration:** Automatic command discovery through module imports
- **Type Safety:** Parameter validation and type checking for all commands
- **Extensibility:** Plugin-style architecture for adding new commands
- **Error Handling:** Comprehensive validation and error reporting

Parser Implementation:

- **Grid Traversal:** Left-to-right, top-to-bottom reading order
- **Indentation Scoping:** Column-based nesting similar to Python syntax
- **Special Cases:** Proper ELSE handling at matching indentation levels
- **Command Tree:** Hierarchical structure preserving execution order and nesting

Programming Language Features

Movement Commands:

- **MOVE**: Forward/backward movement with distance parameters and sensor integration
- **TURN**: Directional rotation with degree specifications and conditional execution
- **WAIT**: Time-based pauses with conditional waiting capabilities

Control Flow Constructs:

- **LOOP**: Count-based, conditional, and infinite loops with proper nesting
- **IF/ELSE**: Conditional execution with complex boolean expressions
- **WHILE**: Dynamic conditional loops with real-time sensor evaluation

Data Manipulation:

- **SET**: Variable assignment with arithmetic and logical operations
- **Variables**: Named storage supporting numeric and boolean values
- **Expressions**: Mathematical operations with operator precedence and type coercion

Sensor Integration:

- **DISTANCE**: Real-time ultrasonic sensor readings
- **OBSTACLE**: Boolean obstacle detection with configurable thresholds
- **BLACK_DETECTED/BLOCK_LOST**: IR sensor integration for line following

Value System and Expression Evaluation

Type System:

- **Number**: Integer and floating-point values with automatic conversion
- **Boolean**: Logical values with proper truth evaluation
- **Variable**: Dynamic typing with scope management
- **Direction**: Spatial orientation values (LEFT, RIGHT, FORWARD, BACKWARD)
- **Sensor**: Real-time sensor data access with caching

Expression Processing:

- **Operator Support:** Arithmetic (+, -, *, /), comparison (<, >, =, !=), logical (AND, OR, NOT)
- **Left-to-Right Evaluation:** Consistent evaluation order with proper precedence
- **Type Coercion:** Automatic type conversion for mixed operations
- **Short-Circuit Evaluation:** Optimized logical operation processing

Execution Context**State Management:**

- **Position Tracking:** Coordinate system for car location and orientation
- **Variable Storage:** Scoped variable management with proper lifecycle
- **Drawing State:** Pen position and path tracking for creative tasks
- **Execution Limits:** Step counting and timeout protection
- **Path History:** Movement recording for visualization and debugging

3.5 Arduino Firmware Implementation

3.5.1 Class-Based Architecture

The Arduino firmware implements a sophisticated object-oriented design for maintainable and extensible hardware control.

Core Classes**Motor Class:**

- Direct hardware abstraction for individual motor control
- PWM speed control with direction management
- Consistent interface for all motor operations

MotionController Class:

- High-level movement coordination using motor instances
- Sophisticated algorithms split across multiple files for organization

- Integration with sensor feedback for precise control

GyroSensor Class:

- MPU-6050 integration with calibration and filtering
- Real-time orientation tracking and yaw calculation
- Temperature compensation and drift correction

UltrasonicSensor Class:

- HC-SR04 distance measurement with noise filtering
- Configurable detection thresholds and timing
- Integration with movement algorithms for obstacle avoidance

PenController Class:

- Servo-based pen positioning with smooth movement
- Position feedback and calibration support
- Integration with drawing coordinate system

3.5.2 Advanced Movement Algorithms

Translation Control**Yaw Correction:**

- Gyroscope-based straight-line movement compensation
- Real-time correction for motor imbalances and surface irregularities
- Configurable correction strength and response time

Distance Calculation:

- Wheel circumference and gear ratio compensation
- Time-based movement with precise speed control
- Integration with encoder feedback when available

Rotation Control

PID-Based Turning:

- Feedback control system for accurate angle achievement
- Direction change tracking for complex maneuvers
- Smooth acceleration and deceleration profiles

Absolute and Relative Positioning:

- Support for both relative turns and absolute heading commands
- Reference orientation management and calibration
- Integration with global coordinate system

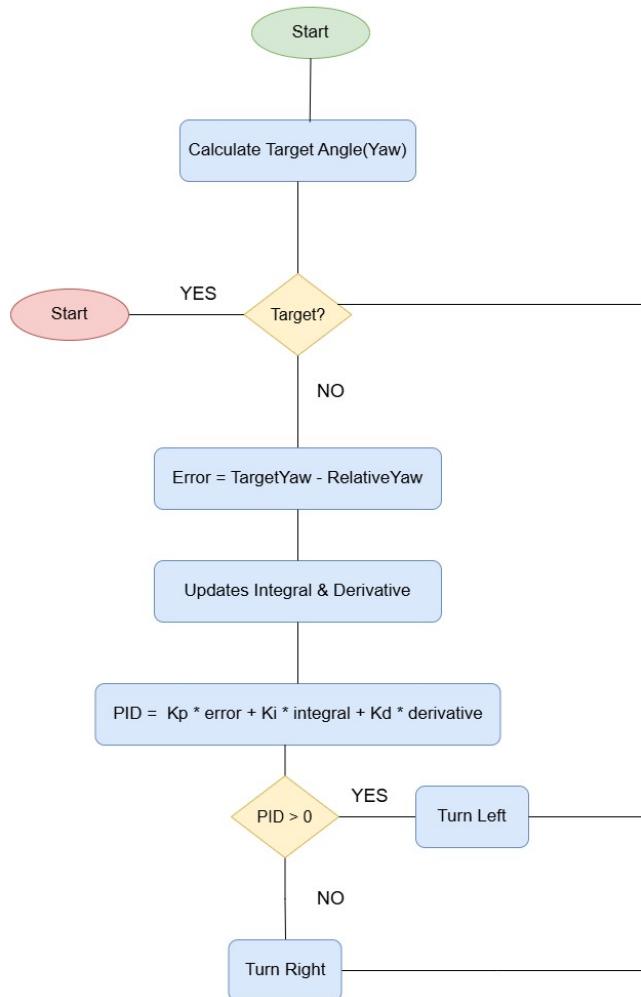


Figure 3.6: PID-Based Rotation Control Algorithm

3.5.3 Serial API Implementation

Command Protocol:

- JSON-based command format: `command:{"param":"value"}`
- Comprehensive parameter validation and error reporting
- Structured response format with success/failure indication
- Support for complex parameter combinations and optional values

API Endpoints:

- **move**: Movement with speed, distance, and time parameters
- **rotate**: Rotation with angle, speed, and absolute positioning
- **pen**: Drawing control with position and action commands
- **gyro**: Sensor calibration and data access
- **sensor**: Distance measurement and obstacle detection

3.6 Mobile Application Implementation

3.6.1 React Native Architecture

Framework and Platform:

- **React Native Expo**: Cross-platform development with native performance
- **iOS and Android**: Responsive design optimized for both platforms
- **WebSocket Integration**: Real-time communication with Raspberry Pi
- **Modern UI**: Built-in React Native components with smooth animations

3.6.2 User Interface Design

Welcome Screen:

- Animated logo with professional design
- Modern solid color backgrounds with decorative elements
- Smooth transitions and call-to-action buttons

Chat Interface:

- Real-time messaging with WebSocket server on Raspberry Pi
- Connection status indicators with pulsing animations
- Dedicated Run/Stop action buttons for program execution
- Markdown support for rich text rendering of system responses
- Auto-scroll functionality for latest message visibility

Communication Features:

- JSON message protocol for command and status exchange
- Error handling and connection recovery mechanisms
- Message queuing for reliable delivery
- Real-time status updates and execution progress monitoring

3.7 Development Methodology

3.7.1 Iterative Development Process

The system was developed using an iterative approach with continuous integration and testing:

Phase-Based Development

Phase 1 - Core Infrastructure: WebSocket server, process control, and basic communication protocols

Phase 2 - Computer Vision: Camera integration, OCR processing, and grid mapping algorithms

Phase 3 - Command Interpreter: Programming language design, parser implementation, and execution engine

Phase 4 - Hardware Integration: Arduino firmware, ESP32 bridge, and sensor calibration

Phase 5 - Mobile Application: User interface development and WebSocket communication integration

3.7.2 Testing and Validation

Component Testing:

- Unit tests for individual modules and classes

- Integration tests for communication protocols
- Hardware-in-the-loop testing for robotic components
- End-to-end workflow validation

Performance Validation:

- Image processing performance under 2 seconds for block recognition
- Real-time program execution with immediate response
- Sub-100ms latency for car movement commands
- Reliable operation under classroom conditions

3.8 System Integration and Deployment

3.8.1 Communication Protocols

WebSocket Communication:

- Real-time bidirectional communication between Raspberry Pi and mobile app
- JSON message format for commands and status updates
- Broadcast support for multiple connected clients
- Automatic reconnection and error recovery

HTTP REST API:

- ESP32-hosted API for car control operations
- Stateless design for reliable remote operation
- Comprehensive error reporting with HTTP status codes
- Timeout management and retry mechanisms

Serial Communication:

- 115200 baud UART between ESP32 and Arduino
- JSON protocol for command translation
- Flow control and error detection
- Command validation and response formatting

3.8.2 Deployment Considerations

Classroom Requirements:

- Portable setup with minimal technical requirements
- Robust operation under variable lighting conditions
- Simple startup and shutdown procedures
- User-friendly error messages and recovery options

Scalability Features:

- Configurable grid sizes for different complexity levels
- Extensible command system for curriculum adaptation
- Multiple car support for collaborative projects
- Modular architecture enabling component upgrades

The implementation successfully delivers a comprehensive educational system that integrates multiple technologies into a cohesive, engaging learning platform. The modular architecture ensures maintainability and extensibility while the robust communication protocols provide reliable operation in educational environments.

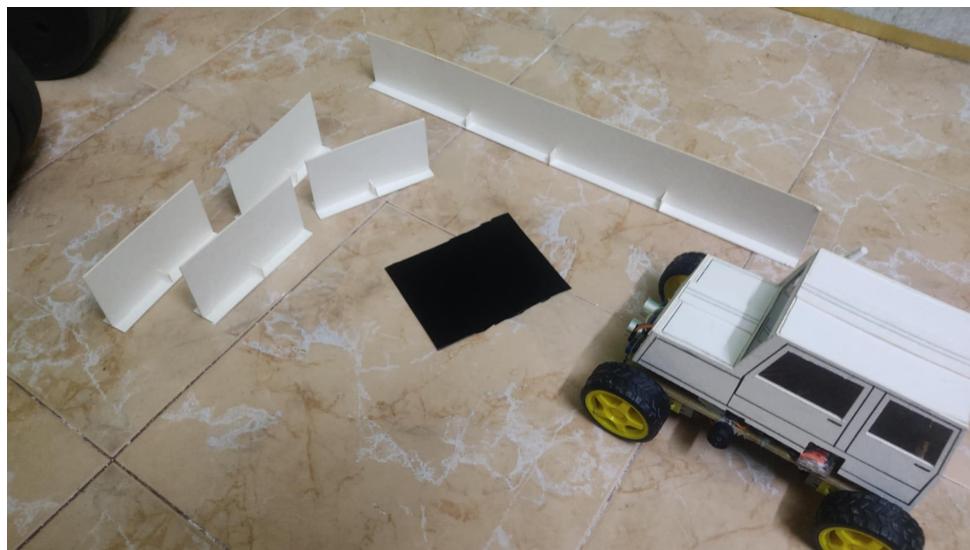


Figure 3.7: TinkerBlocks Car with Obstacle Used in Gameplay

Chapter 4

RESULTS AND CONCLUSION

4.1 System Demonstration

TinkerBlocks has been successfully developed and implemented as a complete educational system that seamlessly integrates physical programming blocks, computer vision, robotics, and mobile technology. The system demonstrates the feasibility and effectiveness of tangible programming interfaces for educational purposes.

4.1.1 Successful System Integration

The most significant achievement of TinkerBlocks is the successful integration of multiple complex technologies into a cohesive, user-friendly educational platform. Children can arrange physical blocks on a grid, and within seconds, see their program executed by a robotic car with real-time feedback through the mobile application.

The complete workflow operates smoothly:

1. Students place programming blocks on the physical grid
2. The overhead camera captures the block arrangement
3. Computer vision algorithms process the image and extract block positions and text
4. The command interpreter translates the visual program into executable instructions
5. Commands are transmitted wirelessly to the robotic car
6. The car executes the program while providing sensor feedback
7. The mobile app displays real-time status and execution progress

4.1.2 Programming Language Capabilities

TinkerBlocks supports a comprehensive set of programming concepts that allow students to create sophisticated programs:

Basic Movement and Control

Students can create simple programs using fundamental commands:

- **Movement:** Forward and backward motion with precise distance control
- **Rotation:** Left and right turns with accurate angle execution
- **Drawing:** Pen control for creative programming tasks
- **Timing:** Wait commands for synchronized operations

Advanced Programming Constructs

The system supports complex programming concepts typically found in advanced programming languages:

- **Loops:** Count-based repetition and conditional loops with WHILE statements
- **Conditionals:** IF/ELSE statements with complex boolean expressions
- **Variables:** Named storage for values with arithmetic operations
- **Sensors:** Real-time integration with distance and line detection sensors
- **Expressions:** Mathematical and logical operations with proper evaluation

Program Examples

Students can create programs ranging from simple movement sequences to complex algorithms:

Simple Square Drawing:

- SET X 4 (set side length)
- PEN_DOWN
- LOOP 4 times: MOVE X, TURN RIGHT
- PEN_UP

Obstacle Avoidance:

- LOOP WHILE TRUE

- IF DISTANCE < 20: TURN RIGHT, MOVE 2, TURN LEFT
- ELSE: MOVE 1

Variable-Based Spiral:

- SET DISTANCE 1
- LOOP 10 times: MOVE DISTANCE, TURN RIGHT, SET DISTANCE DISTANCE + 0.5

4.2 Technical Achievements

4.2.1 Computer Vision Success

The computer vision system successfully recognizes programming blocks and maps them to grid positions in real-time. Key achievements include:

Robust Block Recognition

- **Text Recognition:** EasyOCR integration provides reliable text extraction from physical blocks
- **Perspective Correction:** Automatic detection and correction of camera perspective ensures accurate grid mapping
- **Grid Mapping:** Precise association of detected text with specific grid positions
- **Real-time Processing:** Complete image processing pipeline operates within acceptable time limits

OCR2Grid Mapping Innovation

The custom OCR2Grid mapping system represents a significant technical achievement, translating detected text and positions into structured grid data that preserves programming logic and indentation-based scoping.

4.2.2 Arduino Firmware Excellence

The Arduino firmware demonstrates sophisticated embedded programming with multiple advanced features:

Object-Oriented Architecture

- **Modular Design:** Class-based architecture with separate modules for motors, sensors, and control
- **Advanced Algorithms:** PID-based rotation control and gyroscope-assisted straight-line movement
- **Sensor Integration:** Real-time processing of multiple sensor inputs with filtering and calibration
- **Communication Protocol:** Comprehensive JSON-based API for external control

Movement Precision

The car achieves good movement precision through:

- **Yaw Correction:** Gyroscope feedback maintains straight-line movement despite motor variations
- **PID Control:** Feedback-controlled rotation ensures accurate angle achievement
- **Sensor Feedback:** Real-time obstacle detection and line following capabilities
- **Calibrated Movement:** Precise distance and angle calculations based on wheel characteristics

4.2.3 Communication Architecture

The multi-protocol communication system successfully enables seamless interaction between all components:

Wireless Integration

- **WebSocket Communication:** Real-time bidirectional communication between Raspberry Pi and mobile app
- **HTTP REST API:** ESP32-hosted API provides reliable car control over WiFi
- **Serial Bridge:** Robust communication between ESP32 and Arduino with error handling
- **Protocol Translation:** Seamless translation between different communication protocols

4.2.4 Mobile Application Success

The React Native mobile application provides an intuitive interface that enhances the educational experience:

User Experience

- **Real-time Interaction:** Live communication with the Raspberry Pi system
- **Modern Interface:** Smooth animations and responsive design
- **Cross-platform Support:** Works seamlessly on both iOS and Android devices
- **Educational Features:** Run/stop controls and real-time status updates

4.3 Educational Impact and Value

4.3.1 Learning Benefits

TinkerBlocks addresses fundamental challenges in programming education through its innovative approach:

Tangible Learning

- **Physical Manipulation:** Students learn through hands-on interaction rather than abstract screen-based interfaces
- **Immediate Feedback:** Car movement provides instant visual confirmation of programming logic
- **Spatial Understanding:** Physical arrangement of blocks helps students understand program structure and flow
- **Collaborative Learning:** Multiple students can work together on the same physical grid

Progressive Complexity

The system supports educational progression from simple concepts to advanced programming:

- **Beginner Level:** Basic movement commands and simple sequences
- **Intermediate Level:** Loops, conditionals, and sensor integration
- **Advanced Level:** Variables, expressions, and complex algorithms
- **Creative Projects:** Drawing tasks and open-ended exploration

Computational Thinking Development

TinkerBlocks naturally develops key computational thinking skills:

- **Decomposition:** Breaking complex tasks into smaller, manageable steps
- **Pattern Recognition:** Identifying repetitive sequences and creating loops
- **Abstraction:** Using variables and functions to represent concepts
- **Algorithm Design:** Creating step-by-step solutions to problems

4.3.2 Accessibility and Engagement

Inclusive Design

- **Kinesthetic Learning:** Appeals to students who learn best through physical manipulation
- **Visual Learning:** Car movement provides clear visual representation of program execution
- **Collaborative Environment:** Supports different learning styles and peer interaction
- **Reduced Barriers:** Eliminates syntax requirements that often frustrate beginning programmers

Motivation and Engagement

- **Immediate Results:** Students see their programs executed instantly in the physical world
- **Creative Expression:** Drawing capabilities allow for artistic programming projects
- **Game-like Experience:** Car movement and challenges create engaging learning experiences
- **Social Learning:** Physical interface promotes discussion and collaboration

4.4 Challenges and Limitations

4.4.1 Technical Challenges

While TinkerBlocks successfully achieves its primary objectives, several challenges and limitations have been identified:

Computer Vision Constraints

- **Lighting Sensitivity:** OCR performance depends on consistent lighting conditions
- **Block Quality:** Text recognition accuracy varies with block printing quality and wear
- **Camera Positioning:** Requires proper camera setup and calibration for optimal performance
- **Processing Time:** Complete vision pipeline requires several seconds for complex grids

Physical Limitations

- **Grid Size Constraints:** 16x10 grid limits maximum program complexity
- **Surface Requirements:** Car operation requires smooth, flat surfaces for optimal performance
- **Block Management:** Physical blocks can be lost or damaged, affecting system operation
- **Setup Complexity:** Initial system setup requires technical knowledge and proper configuration

4.4.2 Educational Considerations

Classroom Integration

- **Teacher Training:** Educators need training to effectively integrate the system into curricula
- **Technical Support:** Schools may need ongoing technical support for maintenance and troubleshooting
- **Cost Considerations:** While competitive, the system cost may limit accessibility in some educational settings
- **Curriculum Alignment:** Requires careful integration with existing computer science education standards

4.5 Future Work and Improvements

4.5.1 Technical Enhancements

Several areas for future development have been identified:

Computer Vision Improvements

- **Adaptive Lighting:** Develop algorithms that automatically adjust to varying lighting conditions
- **Enhanced OCR:** Investigate machine learning approaches for improved text recognition accuracy
- **Real-time Processing:** Optimize algorithms for faster image processing and response times
- **Multi-angle Recognition:** Support for different camera angles and perspectives

System Expansion

- **Multiple Robots:** Support for multiple cars operating simultaneously for collaborative projects
- **Larger Grids:** Expandable grid sizes for more complex programming projects
- **Additional Sensors:** Integration of more sensor types for enhanced interaction capabilities
- **Advanced Actuators:** Additional output devices such as speakers, lights, or displays

4.5.2 Educational Enhancements

Curriculum Development

- **Lesson Plans:** Comprehensive curriculum materials for different age groups and skill levels
- **Assessment Tools:** Automated assessment systems for tracking student progress
- **Teacher Resources:** Professional development materials and training programs
- **Standards Alignment:** Explicit alignment with computer science education standards

Advanced Features

- **Programming Constructs:** Support for functions, recursion, and data structures
- **Debugging Tools:** Visual debugging interfaces to help students troubleshoot programs
- **Program Validation:** Pre-execution checking to identify potential program errors
- **Progress Tracking:** Individual student progress monitoring and adaptive learning

4.6 Broader Impact and Significance

4.6.1 Contribution to Educational Technology

TinkerBlocks represents a significant contribution to the field of educational technology and tangible programming interfaces:

Technical Innovation

- **Integration Achievement:** Successful combination of computer vision, robotics, and mobile technology
- **Real-time Processing:** Demonstration of practical real-time computer vision in educational applications
- **Scalable Architecture:** Modular design that can be adapted and extended for various educational contexts
- **Open Source Potential:** Architecture suitable for open source development and community contribution

Educational Research

- **Tangible Programming Validation:** Demonstrates effectiveness of physical programming interfaces
- **Multi-modal Learning:** Shows benefits of combining physical, visual, and digital interaction
- **Collaborative Learning:** Validates benefits of shared physical programming spaces
- **Skill Transfer:** Provides foundation for research on transfer to traditional programming environments

4.6.2 Potential for Widespread Adoption

Scalability

- **Cost Effectiveness:** Competitive pricing compared to existing educational robotics platforms
- **Classroom Ready:** Designed for practical deployment in educational settings
- **Teacher Friendly:** Intuitive operation that doesn't require extensive technical expertise
- **Curriculum Integration:** Flexible system that can adapt to various educational approaches

Global Impact

- **Accessibility:** Physical interface may benefit students with different learning styles and abilities
- **Cultural Adaptability:** System can be adapted for different languages and cultural contexts
- **Educational Equity:** Potential to make programming education more accessible and engaging
- **STEM Promotion:** Engaging platform for promoting interest in science, technology, engineering, and mathematics

4.7 Conclusion

TinkerBlocks successfully demonstrates that tangible programming interfaces can provide an effective and engaging approach to programming education. The system achieves its primary objectives of making programming concepts physical, immediate, and collaborative, while maintaining the sophistication necessary for meaningful learning.

4.7.1 Key Achievements

The project has accomplished several significant milestones:

- **Successful Integration:** Seamless combination of computer vision, robotics, and mobile technology into a cohesive educational platform
- **Technical Excellence:** Robust implementation with sophisticated algorithms for movement control, sensor integration, and real-time communication

- **Educational Value:** Comprehensive programming language support that enables progression from basic concepts to advanced algorithms
- **User Experience:** Intuitive interface that engages students and promotes collaborative learning
- **Practical Deployment:** System designed for real-world classroom use with appropriate consideration for educational constraints

4.7.2 Educational Impact

TinkerBlocks addresses fundamental challenges in programming education by:

- **Reducing Barriers:** Eliminating syntax requirements and abstract interfaces that often frustrate beginning programmers
- **Enhancing Engagement:** Providing immediate, visual feedback through physical robot movement
- **Supporting Collaboration:** Enabling multiple students to work together on shared programming projects
- **Developing Computational Thinking:** Naturally fostering key skills including decomposition, pattern recognition, and algorithm design

4.7.3 Technical Contribution

From a technical perspective, TinkerBlocks contributes to the field through:

- **Computer Vision Innovation:** Practical application of real-time OCR and grid mapping in educational settings
- **Embedded Systems Excellence:** Sophisticated Arduino firmware with advanced control algorithms
- **System Integration:** Successful coordination of multiple technologies and communication protocols
- **Mobile Technology:** Modern cross-platform application with real-time communication capabilities

4.7.4 Future Potential

TinkerBlocks establishes a foundation for future developments in educational technology:

- **Research Platform:** Provides a basis for further research in tangible programming and educational robotics
- **Commercial Viability:** Demonstrates potential for practical educational products that can be deployed at scale
- **Open Source Opportunity:** Architecture suitable for community development and enhancement
- **Educational Innovation:** Model for integrating emerging technologies into educational practice

4.7.5 Final Reflection

The successful development and implementation of TinkerBlocks validates the potential of tangible programming interfaces to transform programming education. By making programming concepts physical, immediate, and collaborative, the system addresses many of the challenges that have historically made programming education difficult for young learners.

The project demonstrates that with careful design, appropriate technology integration, and focus on educational needs, it is possible to create learning tools that are both technically sophisticated and educationally effective. TinkerBlocks represents a step forward in making programming education more accessible, engaging, and effective for the next generation of learners.

As educational technology continues to evolve, systems like TinkerBlocks point toward a future where learning is enhanced by thoughtful integration of physical and digital experiences. The success of this project suggests that the boundary between physical and digital learning environments will continue to blur, creating new opportunities for engaging and effective education.

TinkerBlocks: Code, build, and drive - where programming becomes tangible, learning becomes engaging, and students become creators.

Appendix A

API DOCUMENTATION AND COMMAND REFERENCE

A.1 Arduino Serial API

The Arduino firmware exposes a comprehensive serial API for controlling the robotic car. All commands follow the format: `command:{"param1":"value1","param2":value2}`

A.1.1 Movement Commands

Move Command

```
1 // Move forward 20cm at speed 100
2 move : {"speed":100,"distance":20}
3
4 // Move backward for 1 second at speed 150
5 move : {"speed":-150,"timeMs":1000}
6
7 // Move 30cm in 2 seconds (speed calculated automatically)
8 move : {"distance":30,"timeMs":2000}
9
10 // Move without obstacle checking
11 move : {"speed":100,"distance":20,"checkUltrasonic":false}
```

Listing A.1: Move Command Examples

Parameters:

- `speed` (int): Motor speed (-255 to 255, negative for backward)
- `distance` (float): Distance in centimeters

- **timeMs** (unsigned long): Time in milliseconds
- **checkUltrasonic** (bool): Enable obstacle detection (default: true)
- **enableYawCorrection** (bool): Use gyro correction (default: true)

Success Response:

```

1 {
2   "success": true,
3   "success_result": "{\"distance_traveled\":20.00,\""
4     "time_taken\":784,\"final_yaw\":0.32}"

```

Failure Response:

```

1 {
2   "success": false,
3   "failure_reason": "Obstacle detected at 12.45cm"
4 }

```

Rotation Command

```

1 // Turn left 90 degrees
2 rotate:{ "angle": 90, "speed": 100}
3
4 // Turn right 45 degrees
5 rotate:{ "angle": -45, "speed": 80}
6
7 // Rotate to absolute heading (North = 0 degrees)
8 rotate:{ "angle": 0, "speed": 100, "absolute": true}

```

Listing A.2: Rotation Command Examples

Parameters:

- **angle** (float): Rotation angle in degrees (positive = left/CCW, negative = right/CW)
- **speed** (int): Rotation speed (1-255, default: 100)
- **absolute** (bool): Absolute vs relative rotation (default: false)

A.1.2 Sensor Commands

Ultrasonic Sensor

```
1 // Get distance reading
2 sensor:{ "action": "distance" }
3
4 // Check for obstacles within 15cm
5 sensor:{ "action": "obstacle", "threshold": 15 }
```

Listing A.3: Sensor Command Examples

Gyroscope Commands

```
1 // Calibrate gyroscope
2 gyro:{ "action": "calibrate" }
3
4 // Get current sensor data
5 gyro:{ "action": "data" }
6
7 // Get current yaw angle
8 gyro:{ "action": "yaw" }
9
10 // Set reference orientation
11 gyro:{ "action": "reference" }
```

Listing A.4: Gyroscope Command Examples

A.1.3 Pen Control Commands

```
1 // Lift pen up
2 pen:{ "action": "up" }
3
4 // Put pen down
5 pen:{ "action": "down" }
6
7 // Set custom pen position
8 pen:{ "action": "position", "position": 45 }
```

Listing A.5: Pen Control Examples

A.2 ESP32 HTTP API

The ESP32 serves as a WiFi bridge, exposing HTTP endpoints that translate to Arduino serial commands.

A.2.1 HTTP Endpoints

```
1 # Movement control
2 POST /api/move
3 Content-Type: application/json
4 {"speed": 100, "distance": 20}
5
6 # Rotation control
7 POST /api/rotate
8 Content-Type: application/json
9 {"angle": 90, "speed": 100}
10
11 # Pen control
12 POST /api/pen
13 Content-Type: application/json
14 {"action": "up"}
15
16 # Sensor readings
17 POST /api/sensor
18 Content-Type: application/json
19 {"action": "distance"}
20
21 # Gyroscope control
22 POST /api/gyro
23 Content-Type: application/json
24 {"action": "calibrate"}
25
26 # IR sensor readings
27 POST /api/ir
28 Content-Type: application/json
29 {"action": "black_obstacle"}
```

Listing A.6: HTTP API Endpoints

A.3 WebSocket Communication Protocol

The Raspberry Pi control system uses WebSocket communication for real-time control and monitoring.

A.3.1 Client Commands

```
1 // Run complete OCR to engine pipeline
2 {
3     "command": "run",
4     "params": {
5         "workflow": "full"
6     }
7 }
8
9 // Run OCR only
10 {
11     "command": "run",
12     "params": {
13         "workflow": "ocr_grid"
14     }
15 }
16
17 // Run OCR with automatic engine execution
18 {
19     "command": "run",
20     "params": {
21         "workflow": "ocr_grid",
22         "chain_engine": true
23     }
24 }
25
26 // Run engine with custom grid
27 {
28     "command": "run",
29     "params": {
30         "workflow": "engine",
31         "grid": [
32             ["MOVE", "5"],
33             ["TURN", "RIGHT"],
34             ["LOOP", "3"],
35             ["", "MOVE", "2"],
36             ["", "TURN", "LEFT"]
37         ]
38     }
39 }
```

```
38     }
39 }
40
41 // Stop current process
42 {
43   "command": "stop"
44 }
```

Listing A.7: WebSocket Client Commands

A.3.2 Server Responses

```
1 // Status update
2 {
3   "type": "status",
4   "message": "Processing image..."
5 }

6
7 // Progress update
8 {
9   "type": "progress",
10  "percentage": 45,
11  "message": "Running OCR..."
12 }

13
14 // Completion notification
15 {
16   "type": "complete",
17   "result": "Success",
18   "data": {
19     "execution_time": 2.3,
20     "commands_executed": 15
21   }
22 }

23
24 // Error notification
25 {
26   "type": "error",
27   "message": "Block recognition failed",
```

```

28     "details": "Insufficient lighting detected"
29 }
```

Listing A.8: WebSocket Server Responses

A.4 Programming Language Reference

A.4.1 Movement Commands

```

1 // Basic movement
2 MOVE                      // Move forward 1 unit
3 MOVE | 5                  // Move forward 5 units
4 MOVE | -3                 // Move backward 3 units
5 MOVE | X                  // Move forward X units (variable)
6
7 // Conditional movement
8 MOVE | WHILE | X < 10    // Move while X is less than 10
9 MOVE | WHILE | DISTANCE > 30 // Move while distance > 30cm
10
11 // Turn commands
12 TURN | LEFT              // Turn left 90 degrees
13 TURN | RIGHT             // Turn right 90 degrees
14 TURN | 45                // Turn right 45 degrees
15 TURN | LEFT | 30         // Turn left 30 degrees
16 TURN | RIGHT | WHILE | X < 5 // Turn right while X < 5
```

Listing A.9: Movement Command Examples

A.4.2 Control Flow Examples

```

1 // Simple loop
2 LOOP | 5
3     MOVE | 2
4     TURN | RIGHT
5
6 // Conditional loop
7 LOOP | WHILE | X < 10
8     MOVE | 1
9     SET | X | X + 1
10
11 // Infinite loop with condition
12 LOOP | TRUE
13     MOVE | 1
14     IF | OBSTACLE
15         TURN | RIGHT
```

```

16
17 // Conditional statements
18 IF | DISTANCE < 30
19     TURN | LEFT
20 ELSE
21     MOVE | 5
22
23 // Nested conditions
24 IF | X > 10
25     IF | Y < 5
26         MOVE | X
27     ELSE
28         TURN | RIGHT
29 ELSE
30     SET | X | 0

```

Listing A.10: Control Flow Examples

A.4.3 Variable Operations

```

1 // Basic assignment
2 SET | X | 5
3 SET | SPEED | 10
4 SET | Y | DISTANCE
5 SET | FLAG | TRUE
6
7 // Arithmetic operations
8 SET | Z | X + 2
9 SET | COUNT | COUNT + 1
10 SET | RESULT | X * Y / 2
11 SET | AVERAGE | (X + Y) / 2
12
13 // Boolean operations
14 SET | FOUND | DISTANCE < 30
15 SET | SAFE | NOT OBSTACLE
16 SET | READY | X > 0 AND Y > 0
17 SET | CONTINUE | DISTANCE > 10 OR COUNT < 5

```

Listing A.11: Variable Operation Examples

A.4.4 Sensor Integration

```

1 // Using sensor values in conditions
2 IF | DISTANCE < 20
3     TURN | RIGHT
4

```

```

5 IF | OBSTACLE
6     MOVE | -2
7     TURN | 180
8
9 IF | BLACK_DETECTED
10    MOVE | 1
11 ELSE
12     TURN | LEFT
13
14 // Sensor values in expressions
15 SET | SAFE_DISTANCE | DISTANCE + 5
16 SET | TOO_CLOSE | DISTANCE < 15
17
18 // Waiting for sensor conditions
19 WAIT | WHILE | DISTANCE < 20
20 WAIT | WHILE | BLACK_DETECTED

```

Listing A.12: Sensor Integration Examples

A.4.5 Drawing Commands

```

1 // Draw a square
2 SET | SIDE | 3
3 PEN_DOWN
4 LOOP | 4
5     MOVE | SIDE
6     TURN | RIGHT
7 PEN_UP
8
9 // Draw with conditional pen control
10 LOOP | 10
11     IF | X % 2 = 0
12         PEN_DOWN
13     ELSE
14         PEN_UP
15     MOVE | 2
16     SET | X | X + 1

```

Listing A.13: Drawing Command Examples

A.5 Error Codes and Troubleshooting

A.5.1 Common Error Codes

```

1 // Arduino Serial API Errors

```

```

2 "Invalid speed. Must be between 1 and 255."
3 "Obstacle detected at 12.45cm"
4 "Missing or invalid 'position' parameter for pen position"
5 "Unknown gyro action: invalid_action"
6 "Unknown sensor action: invalid_action"
7
8 // Vision Processing Errors
9 "Block recognition failed - insufficient lighting"
10 "Grid detection failed - corners not found"
11 "OCR processing timeout"
12 "Invalid grid dimensions"
13
14 // Engine Execution Errors
15 "Unknown command: INVALID_COMMAND"
16 "Maximum steps exceeded (1000)"
17 "Variable 'X' not defined"
18 "Division by zero in expression"
19 "ELSE without matching IF"

```

Listing A.14: Error Code Reference

A.5.2 Troubleshooting Guide

```

1 // Poor block recognition
2 Problem: Low recognition accuracy
3 Solution:
4 - Ensure lighting > 300 lux
5 - Check camera focus and positioning
6 - Verify block text clarity
7 - Clean camera lens
8
9 // Communication timeouts
10 Problem: API timeouts
11 Solution:
12 - Check WiFi connection strength
13 - Verify ESP32 power supply
14 - Restart system components
15 - Check serial cable connections
16
17 // Movement inaccuracy
18 Problem: Car movement imprecise
19 Solution:
20 - Calibrate gyroscope sensor
21 - Check battery voltage level
22 - Verify wheel alignment
23 - Test on smooth surface
24

```

```
25 // Program execution errors
26 Problem: Commands not executing
27 Solution:
28 - Verify grid syntax correctness
29 - Check indentation alignment
30 - Validate parameter ranges
31 - Review variable definitions
```

Listing A.15: Common Solutions

References