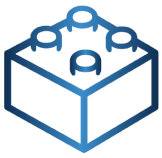

TinkerBlocks: Code, build, and drive!

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

Chapter 1

INTRODUCTION

1.1 Background and Motivation

In an increasingly digital world, programming skills have become essential for students across all disciplines [1]. 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. The concept of tangible programming interfaces has emerged as a promising solution to bridge this gap [2]. Physical manipulation and immediate visual feedback help children grasp abstract concepts more effectively than traditional lecture-based methods. This approach aligns with constructivist learning theories, which emphasize learning through hands-on exploration and manipulation of physical objects.

Educational robotics has proven to be an effective tool for engaging students in learning [3]. The immediate visual and physical feedback provided by robotic systems helps students understand the consequences of their programming decisions. However, most existing educational robotics platforms still rely on screen-based programming interfaces, which may not be suitable for younger learners or those who learn better through physical interaction.

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

Educational robotics has shown as a powerful tool for teaching STEM concepts, particularly programming and computational thinking [3]. The field has evolved significantly since the introduction of Logo and the turtle graphics system in the 1960s, which first demonstrated the potential of robotics in education.

Modern educational robotics platforms like LEGO Mindstorms, VEX Robotics, and Arduino-based systems have made robotics more accessible to students and educators.[4] These platforms typically combine programmable microcontrollers with sensors, actuators, and mechanical components, allowing students to build and program their own robots.

Research has shown that educational robotics can improve student thinking, problem-solving skills, and understanding of programming concepts [5]. The immediate visual and touchable feedback provided by robotic systems helps students understand abstract programming concepts and debug their code more effectively.

2.2 Tangible Programming Interfaces

Tangible programming interfaces represent a paradigm shift from traditional screen-based programming environments to physical manipulation of programming constructs [2]. This approach is rooted in the theory of embodied cognition, which suggests that physical interaction with objects enhances learning and understanding.

2.2.1 Historical Development

The concept of tangible programming can be traced back to early educational toys like the Big Trak programmable robot and more recent systems like the AlgoBlock system developed at MIT [6]. These early systems demonstrated that children could understand

programming concepts through physical manipulation long before they could master text-based programming languages [7].

2.2.2 Current Approaches

Several systems have explored tangible programming interfaces:

- **Cubetto**: A wooden robot that uses colored blocks to represent programming commands
- **Code & Go**: A board game approach to teaching programming logic
- **Osmo Coding**: Uses physical blocks detected by tablet cameras
- **KIBO**: A construction kit for children to build and program robots using wooden blocks [8]

2.2.3 Advantages and Challenges

Tangible programming interfaces offer several advantages:

- Reduced cognitive load by eliminating syntax requirements
- Enhanced spatial understanding of program flow
- Support for collaborative programming activities
- Accessibility for learners with different abilities

However, they also present challenges:

- Limited scalability for complex programs
- Physical constraints on program size
- Recognition accuracy in computer vision-based systems
- Higher hardware costs compared to software-only solutions

2.3 Computer Vision in Educational Applications

Computer vision has become increasingly important in educational technology, enabling systems to interpret and respond to physical student interactions. In the context of tangible programming interfaces, computer vision serves as the bridge between physical manipulation and digital execution.

2.3.1 Constructivist Learning Theory

Based on the work of Jean Piaget and Seymour Papert, constructivist learning theory emphasizes learning through construction and experimentation. This theory has been particularly influential in the design of programming tools for children, promoting hands-on exploration over direct instruction.

2.3.2 Block-Based Programming Languages

Visual programming languages like Scratch, Blockly, and Alice have revolutionized programming education by providing drag-and-drop interfaces that eliminate syntax errors and focus on logic and computational thinking [9]. These tools have demonstrated the effectiveness of visual approaches to programming education [10]

2.3.3 Computational Thinking

Computational thinking encompasses four key skills:

- **Decomposition**: Breaking complex problems into smaller parts
- **Pattern Recognition**: Identifying similarities and patterns
- **Abstraction**: Focusing on essential features while ignoring irrelevant details
- **Algorithm Design**: Creating step-by-step solutions

Effective programming education tools should support the development of these skills through appropriate progression.

2.4 Related Systems and Technologies

2.4.1 Commercial Educational Robotics Platforms

Several commercial platforms share similarities with TinkerBlocks:

LEGO Mindstorms: A comprehensive robotics platform that combines programmable bricks with sensors and motors. While highly capable, it relies on screen-based programming interfaces and requires significant investment in components.

Bee-Bot and Blue-Bot: Simple programmable robots designed for early childhood education. These systems use button-based programming but lack the flexibility of a visual programming interface.

Dash and Dot: Smartphone-controlled robots with visual programming apps. While accessible, they depend on mobile devices and don't provide tangible programming experiences.

2.4.2 Research Prototypes

Academic research has produced several innovative approaches to tangible programming:

Tern: A tangible programming system that uses physical tiles arranged on a surface, detected by an overhead camera. This system demonstrated the feasibility of camera-based block recognition.

Topobo: A 3D construction kit with kinetic memory, allowing children to teach robots behaviors through physical demonstration.

FlowBlocks: A tangible programming system for controlling robotic devices using magnetic blocks that can be arranged on a whiteboard.

2.5 Gaps in Current Research

Despite significant progress in educational robotics and tangible programming interfaces, several gaps remain:

1. **Limited Integration:** Most systems focus on either robotics or programming, but few successfully integrate both in a easy learning experience.
2. **Scalability Issues:** Many tangible programming systems are limited in the complexity of programs they can represent.
3. **Curriculum Integration:** Most systems exist as standalone tools rather than integrated components of educational curricula.
4. **Accessibility:** Limited research has been conducted on making these systems accessible to learners with different abilities and learning styles.

2.6 Positioning of TinkerBlocks

TinkerBlocks addresses several of these gaps by:

- Providing seamless integration between tangible programming and robotic execution
- Supporting multiple game modes that aims to different learning objectives
- Using modern computer vision techniques for robust block recognition
- Implementing a scalable interpreter architecture that can support complex programming constructs
- Designing a modular system that can be extended and adapted for different educational contexts

The system builds upon the strengths of existing approaches while addressing their limitations through innovative technical solutions and design decisions.

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

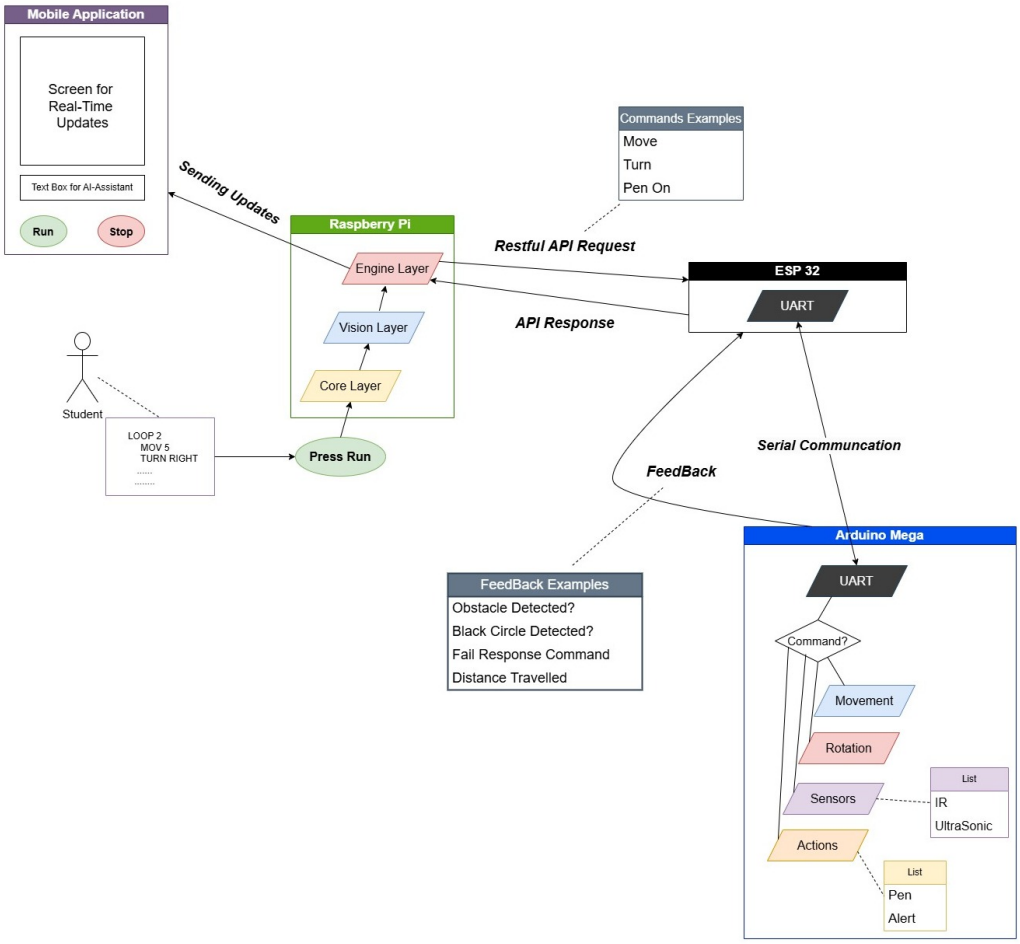

Figure 3.1: Complete System Communication Flow and Component Integration

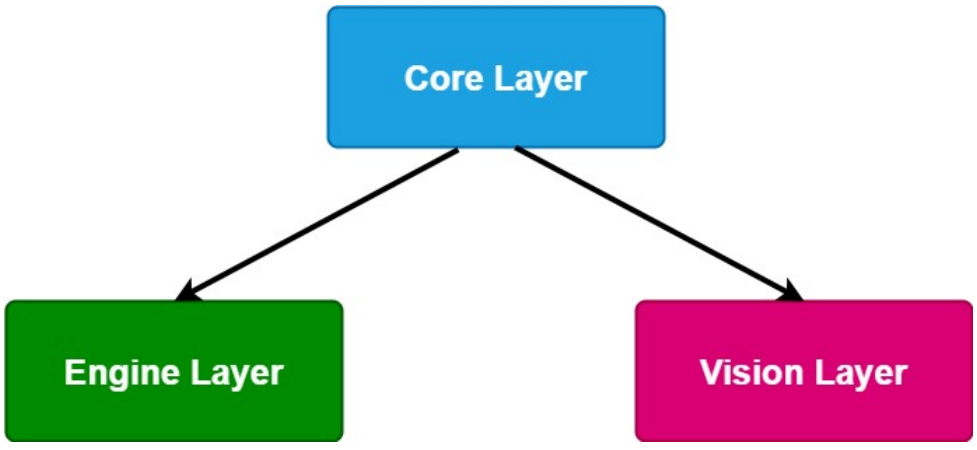

Figure 3.2: Raspberry Pi Software Architecture - Core, Vision, and Engine Modules

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

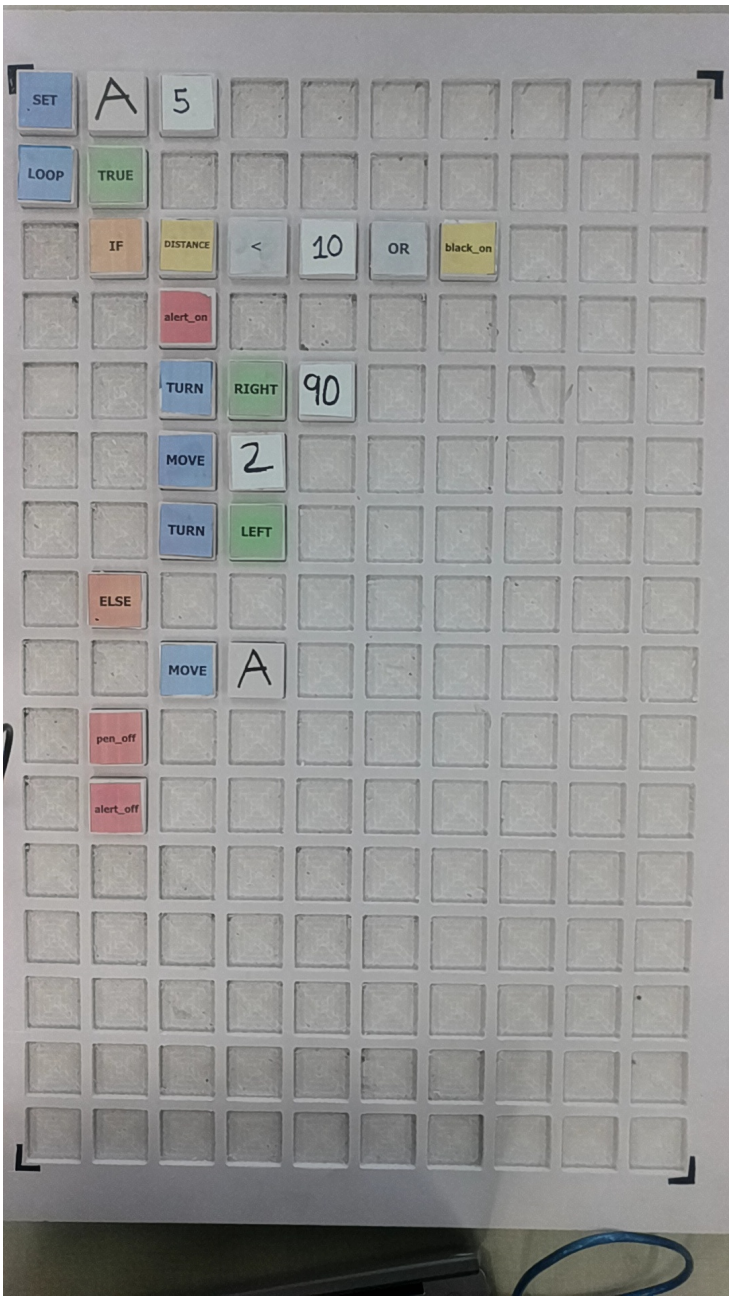

Figure 3.4: Programming Grid with Sample Block Arrangement

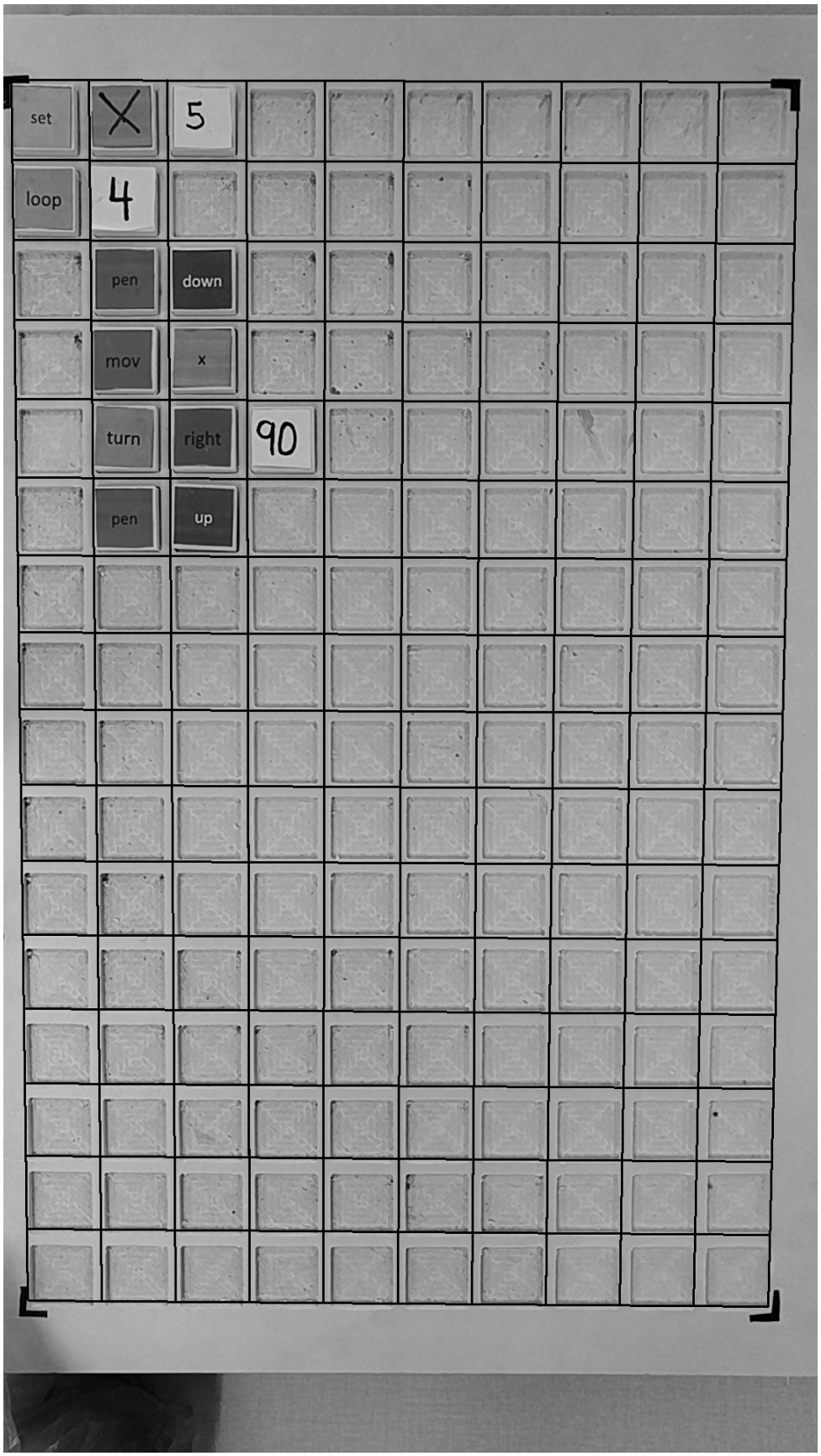

Figure 3.5: Computer Vision Grid Detection and Perspective Transformation

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/BLACK_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

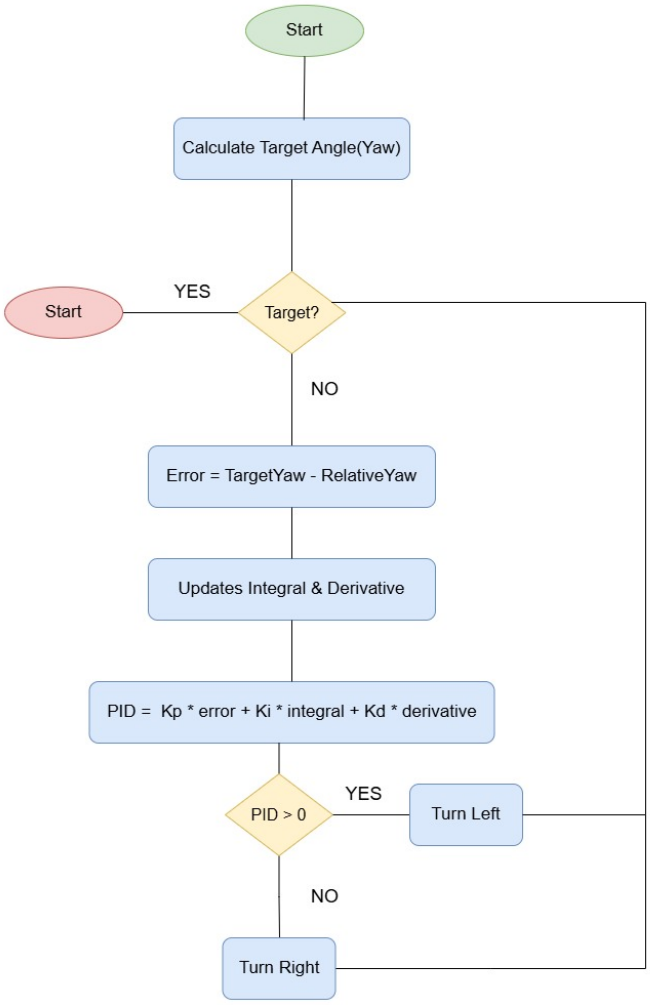

Figure 3.6: PID-Based Rotation Control Algorithm

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.

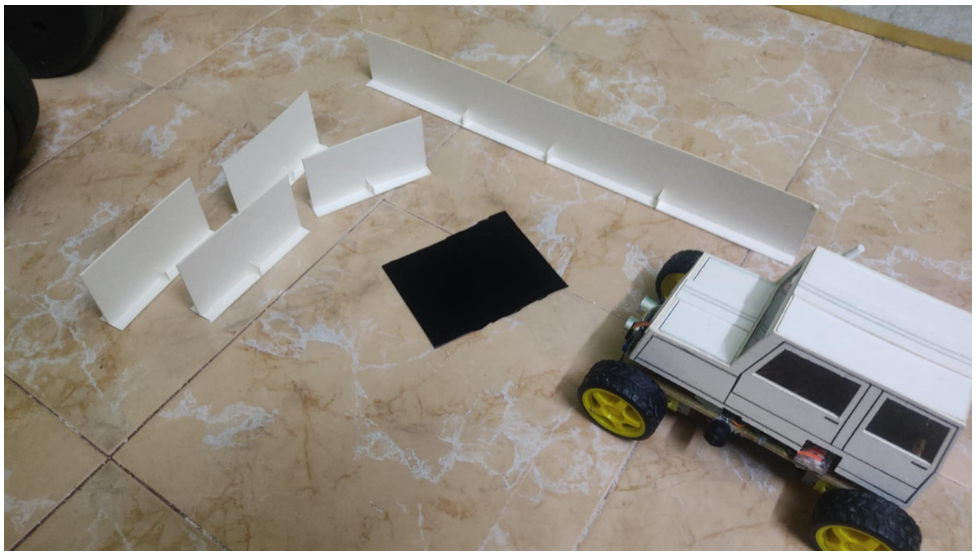

Figure 3.7: TinkerBlocks Car with Obstacle Used in Gameplay

Chapter 4

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.

4.1 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

4.2 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

4.3 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

4.4 Challenges and Limitations

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

4.4.1 Technical Challenges

Computer Vision Constraints

- **Lighting Sensitivity**: OCR performance depends on consistent lighting conditions
- **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 is not as intuitive and easy for students to set up as it should be

4.5 Future Work and Improvements

Several areas for future development have been identified to address current limitations and expand system capabilities:

4.5.1 Technical Enhancements

Computer Vision Improvements

- **Adaptive Lighting**: Develop algorithms that automatically adjust to varying lighting conditions

System Expansion

- **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 lights, or displays

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 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

```
// Move forward 20cm at speed 100
move:{"speed":100,"distance":20}

// Move backward for 1 second at speed 150
move:{"speed":-150,"timeMs":1000}

// Move 30cm in 2 seconds (speed calculated automatically)
move:{"distance":30,"timeMs":2000}

// Move without obstacle checking
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:

```
{
  "success": true,
  "success_result": "{\"distance_traveled\":20.00,\"
   time_taken\":784,\"final_yaw\":0.32}"
}
```

Failure Response:

```
{
  "success": false,
  "failure_reason": "Obstacle detected at 12.45cm"
}
```

Rotation Command

```
// Turn left 90 degrees
rotate:{"angle":90,"speed":100}

// Turn right 45 degrees
rotate:{"angle":-45,"speed":80}

// Rotate to absolute heading (North = 0 degrees)
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

```
// Get distance reading
sensor:{"action":"distance"}

// Check for obstacles within 15cm
sensor:{"action":"obstacle","threshold":15}
```

Listing A.3: Sensor Command Examples

Gyroscope Commands

```
// Calibrate gyroscope
gyro:{"action":"calibrate"}

// Get current sensor data
gyro:{"action":"data"}

// Get current yaw angle
gyro:{"action":"yaw"}

// Set reference orientation
gyro:{"action":"reference"}
```

Listing A.4: Gyroscope Command Examples

A.1.3 Pen Control Commands

```
// Lift pen up
pen:{"action":"up"}

// Put pen down
pen:{"action":"down"}

// Set custom pen position
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

```
# Movement control
POST /api/move
Content-Type: application/json
{"speed": 100, "distance": 20}

# Rotation control
POST /api/rotate
Content-Type: application/json
{"angle": 90, "speed": 100}

# Pen control
POST /api/pen
Content-Type: application/json
{"action": "up"}

# Sensor readings
POST /api/sensor
Content-Type: application/json
{"action": "distance"}

# Gyroscope control
POST /api/gyro
Content-Type: application/json
{"action": "calibrate"}

# IR sensor readings
POST /api/ir
Content-Type: application/json
{"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

```
// Run complete OCR to engine pipeline
{
  "command": "run",
  "params": {
    "workflow": "full"
  }
}

// Run OCR only
{
  "command": "run",
  "params": {
    "workflow": "ocr_grid"
  }
}

// Run OCR with automatic engine execution
{
  "command": "run",
  "params": {
    "workflow": "ocr_grid",
    "chain_engine": true
  }
}

// Run engine with custom grid
{
  "command": "run",
  "params": {
    "workflow": "engine",
    "grid": [
      ["MOVE", "5"],
      ["TURN", "RIGHT"],
      ["LOOP", "3"],
      ["", "MOVE", "2"],
      ["", "TURN", "LEFT"]
    ]
```

```
  }
}

// Stop current process
{
  "command": "stop"
}
```

Listing A.7: WebSocket Client Commands

A.3.2 Server Responses

```
// Status update
{
  "type": "status",
  "message": "Processing image..."
}

// Progress update
{
  "type": "progress",
  "percentage": 45,
  "message": "Running OCR..."
}

// Completion notification
{
  "type": "complete",
  "result": "Success",
  "data": {
    "execution_time": 2.3,
    "commands_executed": 15
  }
}

// Error notification
{
  "type": "error",
  "message": "Block recognition failed",
```

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

Listing A.8: WebSocket Server Responses

A.4 Programming Language Reference

A.4.1 Movement Commands

```
// Basic movement
MOVE                    // Move forward 1 unit
MOVE | 5                // Move forward 5 units
MOVE | -3               // Move backward 3 units
MOVE | X                // Move forward X units (variable)

// Conditional movement
MOVE | WHILE | X < 10  // Move while X is less than 10
MOVE | WHILE | DISTANCE > 30  // Move while distance > 30cm

// Turn commands
TURN | LEFT                     // Turn left 90 degrees
TURN | RIGHT                    // Turn right 90 degrees
TURN | 45                       // Turn right 45 degrees
TURN | LEFT | 30                // Turn left 30 degrees
TURN | RIGHT | WHILE | X < 5    // Turn right while X < 5
```

Listing A.9: Movement Command Examples

A.4.2 Control Flow Examples

```
// Simple loop
LOOP | 5
    MOVE | 2
    TURN | RIGHT

// Conditional loop
LOOP | WHILE | X < 10
    MOVE | 1
    SET | X | X + 1

// Infinite loop with condition
LOOP | TRUE
    MOVE | 1
    IF | OBSTACLE
        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
```

```
IF | OBSTACLE
    MOVE | -2
    TURN | 180

IF | BLACK_DETECTED
    MOVE | 1
ELSE
    TURN | LEFT

// Sensor values in expressions
SET | SAFE_DISTANCE | DISTANCE + 5
SET | TOO_CLOSE | DISTANCE < 15

// Waiting for sensor conditions
WAIT | WHILE | DISTANCE < 20
WAIT | WHILE | BLACK_DETECTED
```

Listing A.12: Sensor Integration Examples

A.4.5 Drawing Commands

```
// Draw a square
SET | SIDE | 3
PEN_DOWN
LOOP | 4
    MOVE | SIDE
    TURN | RIGHT
PEN_UP

// Draw with conditional pen control
LOOP | 10
    IF | X % 2 = 0
        PEN_DOWN
    ELSE
        PEN_UP
    MOVE | 2
    SET | X | X + 1
```

Listing A.13: Drawing Command Examples

A.5 Error Codes and Troubleshooting

A.5.1 Common Error Codes

```
// Arduino Serial API Errors
```

```
"Invalid speed. Must be between 1 and 255."
"Obstacle detected at 12.45cm"
"Missing or invalid 'position' parameter for pen position"
"Unknown gyro action: invalid_action"
"Unknown sensor action: invalid_action"

// Vision Processing Errors
"Block recognition failed - insufficient lighting"
"Grid detection failed - corners not found"
"OCR processing timeout"
"Invalid grid dimensions"

// Engine Execution Errors
"Unknown command: INVALID_COMMAND"
"Maximum steps exceeded (1000)"
"Variable 'X' not defined"
"Division by zero in expression"
"ELSE without matching IF"
```

Listing A.14: Error Code Reference

A.5.2 Troubleshooting Guide

```
// Poor block recognition
Problem: Low recognition accuracy
Solution:
- Ensure lighting > 300 lux
- Check camera focus and positioning
- Verify block text clarity
- Clean camera lens

// Communication timeouts
Problem: API timeouts
Solution:
- Check WiFi connection strength
- Verify ESP32 power supply
- Restart system components
- Check serial cable connections

// Movement inaccuracy
Problem: Car movement imprecise
Solution:
- Calibrate gyroscope sensor
- Check battery voltage level
- Verify wheel alignment
- Test on smooth surface

```

```
// Program execution errors
Problem: Commands not executing
Solution:
- Verify grid syntax correctness
- Check indentation alignment
- Validate parameter ranges
- Review variable definitions
```

Listing A.15: Common Solutions

References

[1] J. M. Wing, "Computational thinking," *Communications of the ACM*, vol. 49, no. 3, pp. 33–35, 2006.

[2] H. Ishii, "Tangible user interfaces: past, present, and future directions," *Communications of the ACM*, vol. 51, no. 6, pp. 32–36, 2008.

[3] F. B. V. Benitti, "Exploring the educational potential of robotics in schools: A systematic review," *Computers & Education*, vol. 58, no. 3, pp. 978–988, 2012.

[4] S. Papert, "Mindstorms: Children, computers, and powerful ideas," 1980.

[5] A. Sullivan, E. R. Kazakoff, and M. U. Bers, "Robots in the classroom: Teaching young children to program with robotics," *Childhood Education*, vol. 89, no. 6, pp. 380–389, 2013.

[6] D. Bau, J. Gray, C. Kelleher, J. Sheldon, and F. Turbak, "Alphablocks: a tangible programming language for children," in *Proceedings of the 8th International Conference on Interaction Design and Children*, 2009, pp. 170–177.

[7] H. Suzuki and H. Kato, "Interaction design and children," *Communications of the ACM*, vol. 42, no. 5, pp. 39–41, 1999.

[8] E. R. Kazakoff, A. Sullivan, and M. U. Bers, "Putcode: Creating tangible programming blocks for young children," in *Proceedings of the 10th International Conference on Interaction Design and Children*, 2011, pp. 75–84.

[9] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman *et al.*, "Scratch: Programming for all," in *Communications of the ACM*, vol. 52, no. 11. ACM, 2009, pp. 60–67.

[10] J. Maloney, M. Resnick, N. Rusk, B. Silverman, and E. Eastmond, "The scratch programming language and environment," *ACM Transactions on Computing Education*, vol. 10, no. 4, pp. 1–15, 2010.