ACTUATOR25

Complete Robotics Textbook

A Comprehensive Guide to Arduino, ESP8266, and Autonomous Robotics

ROBOCEK, Government College of Engineering Kannur

September 1, 2025

Contents

Chapter 1

Arduino Fundamentals

1.1 What is Arduino?

Arduino is an open-source electronics platform based on easy-to-use hardware and software. It consists of a programmable circuit board (microcontroller) and a development environment for writing software.

1.1.1 Key Features

- **Microcontroller**: Atmel AVR chip (ATmega328P in Arduino Uno)
- **Operating Voltage**: 5V
- **Digital I/O Pins**: 14 (6 can be used as PWM outputs)
- **Analog Input Pins**: 6
- **Flash Memory**: 32KB (0.5KB used by bootloader)
- **SRAM**: 2KB
- **EEPROM**: 1KB
- **Clock Speed**: 16MHz

1.2 Arduino IDE and Programming

1.2.1 Arduino IDE Structure

```
void setup() {
    // Code that runs once at startup
    // Initialize pins, variables, libraries
}

void loop() {
    // Code that runs continuously
    // Main program logic
```

```
}
```

Listing 1.1: Basic Arduino Structure

1.2.2 Key Functions

- `pinMode(pin, mode)`: Sets pin as INPUT or OUTPUT
- `digitalWrite(pin, value)`: Writes HIGH or LOW to digital pin
- `digitalRead(pin)`: Reads HIGH or LOW from digital pin
- `analogWrite(pin, value)`: Writes analog value (0-255) to PWM pin
- `analogRead(pin)`: Reads analog value (0-1023) from analog pin
- `Serial.begin(baud)`: Initializes serial communication
- `Serial.print()`: Prints data to serial monitor

1.3 Digital vs Analog

1.3.1 Digital Signals

- Only two states: HIGH (5V) or LOW (0V)
- Used for: switches, LEDs, digital sensors
- Functions: `digitalWrite()`, `digitalRead()`

1.3.2 Analog Signals

- Continuous range of values
- PWM (Pulse Width Modulation): Simulates analog output
- Analog input: 0-1023 range (10-bit resolution)
- Functions: `analogWrite()`, `analogRead()`

1.4 Basic Circuit Concepts

$$\text{Voltage (V)} : \text{Electrical pressure} \tag{1.1}$$
$$\text{Current (A)} : \text{Flow of electrons} \tag{1.2}$$
$$\text{Resistance } (\Omega) : \text{Opposition to current flow} \tag{1.3}$$
$$\text{Power (W)} : V \times A \tag{1.4}$$

Ohm's Law: $V = I \times R$

Chapter 2

Electronics and Components

2.1 Essential Components

2.1.1 Arduino Uno

- Main microcontroller board
- USB connection for programming
- Power regulation and protection

2.1.2 Breadboard

- Prototyping platform
- Connected rows and columns
- No soldering required

2.1.3 Jumper Wires

- Male-to-male, male-to-female, female-to-female
- Connect components to Arduino

2.1.4 Resistors

- Limit current flow
- Color-coded values
- Common values: 220Ω, 1kΩ, 10kΩ

2.2 Motor Driver (L298N/L293D)

2.2.1 Purpose

Control DC motors with Arduino. Arduino can't provide enough current for motors directly.

2.2.2 Pin Functions

- **ENA/ENB**: Enable pins (PWM for speed control)
- **IN1/IN2**: Control left motor direction
- **IN3/IN4**: Control right motor direction
- **VCC**: Logic power (5V)
- **VM**: Motor power (6-12V)
- **GND**: Ground connection

2.2.3 Motor Control Logic

IN1	IN2	Left Motor
HIGH	LOW	Forward
LOW	HIGH	Backward
LOW	LOW	Stop

Table 2.1: Left Motor Control

IN3	IN4	Right Motor
HIGH	LOW	Forward
LOW	HIGH	Backward
LOW	LOW	Stop

Table 2.2: Right Motor Control

2.3 DC Motors

2.3.1 Working Principle

- Electromagnetic induction
- Commutator switches current direction
- Brushes maintain electrical contact

2.3.2 Specifications

- **Voltage**: 6-12V typical
- **Current**: 100-500mA per motor
- **Speed**: 100-300 RPM
- **Torque**: Varies by motor type

2.3.3 Speed Control

- PWM (Pulse Width Modulation)
- Duty cycle determines average voltage
- Higher duty cycle = faster speed

Chapter 3

Motor Control and Movement

3.1 Basic Movement Functions

3.1.1 Forward Movement

```
void moveForward() {
    digitalWrite(LEFT_MOTOR_1, HIGH);
    digitalWrite(LEFT_MOTOR_2, LOW);
    digitalWrite(RIGHT_MOTOR_1, HIGH);
    digitalWrite(RIGHT_MOTOR_2, LOW);
    analogWrite(LMOTOR_PWM, LEFT_SPEED);
    analogWrite(RMOTOR_PWM, RIGHT_SPEED);
}
```

Listing 3.1: Forward Movement Function

3.1.2 Backward Movement

```
void moveBackward() {
    digitalWrite(LEFT_MOTOR_1, LOW);
    digitalWrite(LEFT_MOTOR_2, HIGH);
    digitalWrite(RIGHT_MOTOR_1, LOW);
    digitalWrite(RIGHT_MOTOR_2, HIGH);
    analogWrite(LMOTOR_PWM, LEFT_SPEED);
    analogWrite(RMOTOR_PWM, RIGHT_SPEED);
}
```

Listing 3.2: Backward Movement Function

3.1.3 Turning Logic

- **Left Turn**: Left motor backward, right motor forward
- **Right Turn**: Left motor forward, right motor backward
- **Stop**: Both motors off

3.2 Differential Drive

3.2.1 Principle

Two independently controlled wheels

3.2.2 Advantages

- Simple construction
- Good maneuverability
- Can turn in place

3.2.3 Movement Types

1. **Forward/Backward**: Both wheels same direction
2. **Turning**: Wheels opposite directions
3. **Curved Path**: Different wheel speeds

Chapter 4

Sensors and Sensing

4.1 IR Sensors (Infrared)

4.1.1 Working Principle

- IR LED emits infrared light
- Photodiode detects reflected light
- Black surfaces absorb IR, white surfaces reflect
- Output: Digital (HIGH/LOW) or Analog

4.1.2 Line Detection

$$\text{Black Line:} \quad \text{IR absorbed} \rightarrow \text{LOW output} \tag{4.1}$$

$$\text{White Surface:} \quad \text{IR reflected} \rightarrow \text{HIGH output} \tag{4.2}$$

4.1.3 Calibration

- Test on black and white surfaces
- Find threshold value
- Adjust sensitivity if needed

4.2 Ultrasonic Sensors (HC-SR04)

4.2.1 Working Principle

- Trig pin sends ultrasonic pulse
- Echo pin receives reflected pulse
- Time difference = distance calculation
- Formula: Distance = (Time $\times$ Speed of Sound) / 2

4.2.2 Pin Functions

- **VCC**: 5V power
- **Trig**: Trigger pin (send pulse)
- **Echo**: Echo pin (receive pulse)
- **GND**: Ground

4.2.3 Distance Calculation

```
float measure_distance(int trig, int echo) {
    digitalWrite(trig, HIGH);
    delayMicroseconds(10);
    digitalWrite(trig, LOW);
    long duration = pulseIn(echo, HIGH);
    float distance = duration * 0.034 / 2;
    return distance;
}
```

Listing 4.1: Ultrasonic Distance Measurement

4.3 Light Sensors (LDR - Light Dependent Resistor)

4.3.1 Working Principle

- Resistance changes with light intensity
- More light = lower resistance
- Voltage divider circuit
- Analog output (0-1023)

4.3.2 Applications

- Light following robots
- Automatic lighting systems
- Solar tracking

Chapter 5

Line Follower Robot

5.1 Concept and Working Principle

5.1.1 Objective

Follow a black line on white surface

5.1.2 Sensors

Two IR sensors (left and right)

5.1.3 Logic

Keep line between sensors

5.1.4 Algorithm

$$\text{Both sensors on white} \rightarrow \text{Move forward} \tag{5.1}$$

$$\text{Left sensor on black} \rightarrow \text{Turn left} \tag{5.2}$$

$$\text{Right sensor on black} \rightarrow \text{Turn right} \tag{5.3}$$

$$\text{Both sensors on black} \rightarrow \text{Stop (end of line)} \tag{5.4}$$

5.2 Code Analysis

5.2.1 Pin Definitions

```
#define LEFT_IR 4       // Left IR sensor
#define RIGHT_IR 5      // Right IR sensor
#define LEFT_MOTOR_1 6 // Left motor control 1
#define LEFT_MOTOR_2 7 // Left motor control 2
#define RIGHT_MOTOR_1 8 // Right motor control 1
#define RIGHT_MOTOR_2 9 // Right motor control 2
#define LMOTOR_PWM 10  // Left motor speed control
```

```
#define RMOTOR_PWM 11  // Right motor speed control
```

Listing 5.1: Pin Definitions for Line Follower

5.2.2 Setup Function

```
void setup() {
    Serial.begin(9600);  // Initialize serial communication

    // Configure IR sensors as inputs
    pinMode(LEFT_IR, INPUT);
    pinMode(RIGHT_IR, INPUT);

    // Configure motor pins as outputs
    pinMode(LEFT_MOTOR_1, OUTPUT);
    pinMode(LEFT_MOTOR_2, OUTPUT);
    pinMode(RIGHT_MOTOR_1, OUTPUT);
    pinMode(RIGHT_MOTOR_2, OUTPUT);
    pinMode(LMOTOR_PWM, OUTPUT);
    pinMode(RMOTOR_PWM, OUTPUT);
}
```

Listing 5.2: Setup Function

5.2.3 Main Loop Logic

```
void loop() {
    // Read sensor values
    int LDATA = digitalRead(LEFT_IR);
    int RDATA = digitalRead(RIGHT_IR);

    // Decision making
    if (LDATA == 0 && RDATA == 0) {
        moveForward();  // Both sensors on white
    }
    else if (LDATA == 0 && RDATA == 1) {
        moveLeft();     // Right sensor on black
    }
    else if (LDATA == 1 && RDATA == 0) {
        moveRight();    // Left sensor on black
    }
    else if (LDATA == 1 && RDATA == 1) {
        stop();         // Both sensors on black
    }
}
```

Listing 5.3: Main Loop Logic

5.3 Hardware Setup

5.3.1 Components Required

- Arduino Uno
- 2× IR sensors
- L298N motor driver
- 2× DC motors with wheels
- Robot chassis
- Battery pack (6-12V)

5.3.2 Wiring Diagram

Component	Connection
Left IR	Pin 4
Right IR	Pin 5
ENA	Pin 10 (PWM)
IN1	Pin 6
IN2	Pin 7
IN3	Pin 8
IN4	Pin 9
ENB	Pin 11 (PWM)

Table 5.1: Line Follower Wiring

5.4 Troubleshooting

5.4.1 Common Issues

1. **Robot not following line**: Check sensor calibration
2. **Erratic movement**: Adjust motor speeds
3. **Sensors not responding**: Check wiring and power
4. **Line detection issues**: Clean sensors, adjust height

Chapter 6

Obstacle Avoider Robot

6.1 Concept and Working Principle

6.1.1 Objective

Navigate while avoiding obstacles

6.1.2 Sensors

Two ultrasonic sensors (left and right)

6.1.3 Logic

Measure distances and turn away from obstacles

6.1.4 Algorithm

$$\text{Both distances} > \text{threshold} \rightarrow \text{Move forward} \tag{6.1}$$

$$\text{Left distance} < \text{threshold} \rightarrow \text{Turn right} \tag{6.2}$$

$$\text{Right distance} < \text{threshold} \rightarrow \text{Turn left} \tag{6.3}$$

$$\text{Both distances} < \text{threshold} \rightarrow \text{Stop} \tag{6.4}$$

6.2 Code Analysis

6.2.1 Pin Definitions

```
#define LEFT_US_TRIG 4   // Left ultrasonic trigger
#define LEFT_US_ECHO 5   // Left ultrasonic echo
#define RIGHT_US_TRIG 6  // Right ultrasonic trigger
#define RIGHT_US_ECHO 7  // Right ultrasonic echo
#define LEFT_MOTOR_1 8   // Left motor control 1
#define LEFT_MOTOR_2 9   // Left motor control 2
#define RIGHT_MOTOR_1 10 // Right motor control 1
```

```
#define RIGHT_MOTOR_2 11 // Right motor control 2
#define LMOTOR_PWM 12    // Left motor speed
#define RMOTOR_PWM 3     // Right motor speed
```

Listing 6.1: Pin Definitions for Obstacle Avoider

6.2.2 Distance Measurement Function

```
float measure_distance(int trig, int echo) {
    digitalWrite(trig, HIGH);
    delayMicroseconds(10);
    digitalWrite(trig, LOW);
    long duration = pulseIn(echo, HIGH);
    float distance = duration * 0.034 / 2;
    return distance;
}
```

Listing 6.2: Distance Measurement Function

6.2.3 Main Loop Logic

```
void loop() {
    // Measure distances
    left_distance = measure_distance(LEFT_US_TRIG, LEFT_US_ECHO);
    right_distance = measure_distance(RIGHT_US_TRIG, RIGHT_US_ECHO);

    // Decision making
    if (left_distance > MAX_DISTANCE && right_distance > MAX_DISTANCE) {
        moveForward();  // Clear path ahead
    }
    else if (left_distance < MAX_DISTANCE && right_distance >
   MAX_DISTANCE) {
        moveRight();    // Obstacle on left
    }
    else if (left_distance > MAX_DISTANCE && right_distance <
   MAX_DISTANCE) {
        moveLeft();     // Obstacle on right
    }
    else if (left_distance < MAX_DISTANCE && right_distance <
   MAX_DISTANCE) {
        stop();         // Obstacles on both sides
    }
}
```

Listing 6.3: Obstacle Avoidance Main Loop

6.3 Hardware Setup

6.3.1 Components Required

- Arduino Uno

- 2× Ultrasonic sensors (HC-SR04)
- L298N motor driver
- 2× DC motors with wheels
- Robot chassis
- Battery pack (6-12V)

6.3.2 Wiring Diagram

Component	Connection
Left US Trig	Pin 4
Left US Echo	Pin 5
Right US Trig	Pin 6
Right US Echo	Pin 7
ENA	Pin 12 (PWM)
IN1	Pin 8
IN2	Pin 9
IN3	Pin 10
IN4	Pin 11
ENB	Pin 3 (PWM)

Table 6.1: Obstacle Avoider Wiring

6.4 Optimization Techniques

6.4.1 Distance Threshold

- Adjust `MAX_DISTANCE` based on robot speed
- Typical values: 15-25 cm
- Consider sensor mounting height

6.4.2 Turning Strategy

- Sharp turns for immediate obstacles
- Gradual turns for distant obstacles
- Implement different turn speeds

Chapter 7

ESP8266 and WiFi Robotics

7.1 ESP8266 vs Arduino

7.1.1 ESP8266 Advantages

- Built-in WiFi capability
- Higher clock speed (80MHz)
- More GPIO pins
- Lower cost
- Smaller size

7.1.2 ESP8266 Specifications

- **Microcontroller**: Tensilica L106 32-bit
- **Operating Voltage**: 3.3V
- **Digital I/O Pins**: 11
- **Analog Input Pins**: 1
- **Flash Memory**: 4MB
- **SRAM**: 80KB
- **Clock Speed**: 80MHz

7.2 WiFi Capabilities

7.2.1 Modes of Operation

1. **Station Mode**: Connect to existing WiFi network
2. **Access Point Mode**: Create WiFi network

3. **Station + AP Mode**: Both simultaneously

7.2.2 Key Libraries

```
#include <ESP8266WiFi.h>          // WiFi functionality
#include <ESP8266WebServer.h>     // Web server
#include <ESP8266HTTPClient.h>    // HTTP client
```

Listing 7.1: ESP8266 Libraries

7.3 Pin Mapping

7.3.1 NodeMCU Pin Mapping

NodeMCU Pin	GPIO
D0	GPIO16
D1	GPIO5
D2	GPIO4
D3	GPIO0
D4	GPIO2
D5	GPIO14
D6	GPIO12
D7	GPIO13
D8	GPIO15

Table 7.1: NodeMCU Pin Mapping

7.3.2 Important Notes

- D0, D3, D8 have special functions
- Use D1-D7 for general I/O
- Analog input only on A0

Chapter 8

Web Controlled Robot

8.1 Concept and Working Principle

8.1.1 Objective

Control robot via web browser

8.1.2 Method

ESP8266 creates WiFi access point

8.1.3 Interface

HTML web page with control buttons

8.1.4 Features

- WiFi access point creation
- Web server hosting
- Real-time motor control
- Mobile-friendly interface

8.2 Code Analysis

8.2.1 WiFi Setup

```
const char* ssid = "MY_BOT_AP";
const char* password = "12345678";

void setup() {
    // Create Access Point
    WiFi.mode(WIFI_AP);
    WiFi.softAP(ssid, password);
```

```
8
9       Serial.print("IP Address: ");
10      Serial.println(WiFi.softAPIP()); // Usually 192.168.4.1
11  }
```

Listing 8.1: WiFi Access Point Setup

8.2.2 Web Server Setup

```
1  ESP8266WebServer server(80);
2
3  void setup() {
4      // Define web routes
5      server.on("/", handleRoot);
6      server.on("/forward", moveForward);
7      server.on("/backward", moveBackward);
8      server.on("/left", moveLeft);
9      server.on("/right", moveRight);
10     server.on("/stop", stop);
11
12     server.begin();
13 }
```

Listing 8.2: Web Server Configuration

8.2.3 HTML Interface

```
1  void handleRoot() {
2      String html = "<html><body style='text-align:center;'>";
3      html += "<h1 style='color:white;'>Web Controlled Car</h1>";
4      html += "<button onclick=\"fetch('/forward')\">Forward</button>";
5      html += "<button onclick=\"fetch('/backward')\">Backward</button>";
6      html += "<button onclick=\"fetch('/left')\">Left</button>";
7      html += "<button onclick=\"fetch('/right')\">Right</button>";
8      html += "<button onclick=\"fetch('/stop')\">Stop</button>";
9      html += "</body></html>";
10     server.send(200, "text/html", html);
11 }
```

Listing 8.3: HTML Control Interface

8.2.4 Main Loop

```
1  void loop() {
2      server.handleClient(); // Handle web requests
3  }
```

Listing 8.4: Web Server Main Loop

8.3 Hardware Setup

8.3.1 Components Required

- NodeMCU ESP8266
- L298N motor driver
- 2× DC motors with wheels
- Robot chassis
- Battery pack (3.3V-5V)

8.3.2 Wiring Diagram

Component	Connection
ENA	D1 (PWM)
IN1	D6
IN2	D7
IN3	D8
IN4	D0
ENB	D1 (PWM)

Table 8.1: Web Controlled Robot Wiring

8.4 Usage Instructions

8.4.1 Connection Steps

1. Upload code to NodeMCU
2. Connect to "MY_BOT_AP" WiFi network
3. Open browser and go to `http://192.168.4.1`
4. Use buttons to control robot

8.4.2 Security Considerations

- Change default password
- Implement authentication if needed
- Consider encryption for sensitive applications

Chapter 9

Light Follower Robot

9.1 Concept and Working Principle

9.1.1 Objective

Follow light sources automatically

9.1.2 Sensors

Three light sensors (left, middle, right)

9.1.3 Logic

Move toward brightest light source

9.1.4 Algorithm

$$\text{Middle sensor} > \text{threshold} \rightarrow \text{Move forward} \tag{9.1}$$

$$\text{Left sensor} > \text{right sensor} \rightarrow \text{Turn left} \tag{9.2}$$

$$\text{Right sensor} > \text{left sensor} \rightarrow \text{Turn right} \tag{9.3}$$

$$\text{All sensors} < \text{threshold} \rightarrow \text{Stop} \tag{9.4}$$

9.2 Code Analysis

9.2.1 Sensor Setup

```
#define LEFT_LIGHT_SENSOR D4
#define RIGHT_LIGHT_SENSOR D5
#define MIDDLE_LIGHT_SENSOR D6

// WiFi credentials
const char* ssid = "YOUR_WIFI_SSID";
```

```
const char* password = "YOUR_WIFI_PASSWORD";
```

Listing 9.1: Light Sensor Pin Definitions

9.2.2 Light Following Algorithm

```
void lightFollowingAlgorithm() {
    // Read light sensor values
    leftLightValue = analogRead(LEFT_LIGHT_SENSOR);
    rightLightValue = analogRead(RIGHT_LIGHT_SENSOR);
    middleLightValue = analogRead(MIDDLE_LIGHT_SENSOR);

    // Decision making
    if (middleLightValue > 500) {
        moveForward();  // Strong light ahead
    }
    else if (leftLightValue > rightLightValue && leftLightValue > 300) {
        turnLeft();     // More light on left
    }
    else if (rightLightValue > leftLightValue && rightLightValue > 300)
    {
        turnRight();    // More light on right
    }
    else {
        stop();         // No significant light
    }
}
```

Listing 9.2: Light Following Algorithm

9.2.3 Web Monitoring Interface

```
void handleRoot() {
    String html = "<html><head><meta name='viewport' content='width=
    device-width, initial-scale=1'>";
    html += "<title>Light Follower Bot Status</title>";
    html += "<style>";
    html += "body{font-family:Arial,sans-serif;text-align:center;
    background:#f0f0f0;margin:20px;}";
    html += ".container{max-width:600px;margin:0 auto;background:white;
    padding:20px;border-radius:10px;box-shadow:0 2px 10px rgba(0,0,0,0.1)
    ;}";
    html += "h1{color:#333;}";
    html += ".status{background:#e8f5e8;padding:15px;border-radius:5px;
    margin:20px 0;font-size:18px;font-weight:bold;}";
    html += ".sensor{background:#f8f8f8;padding:10px;margin:10px 0;
    border-radius:5px;}";
    html += ".auto{background:#fff3cd;padding:10px;margin:10px 0;border-
    radius:5px;}";
    html += "</style></head><body>";
    html += "<div class='container'>";
    html += "<h1>       Light Follower Bot</h1>";
    html += "<div class='status'>Status: " + botStatus + "</div>";
```

```
    html += "<div class='sensor'>Left Light Sensor: " + String(leftLightValue) + "</div>";
    html += "<div class='sensor'>Middle Light Sensor: " + String(middleLightValue) + "</div>";
    html += "<div class='sensor'>Right Light Sensor: " + String(rightLightValue) + "</div>";
    html += "<div class='auto'>Auto Mode: Active - Following Light</div>";
    html += "<p>Page refreshes every 2 seconds</p>";
    html += "</div>";
    html += "<script>setTimeout(function(){location.reload();}, 2000);</script>";
    html += "</body></html>";
    server.send(200, "text/html", html);
}
```

Listing 9.3: Web Monitoring Interface

9.3 Hardware Setup

9.3.1 Components Required

- NodeMCU ESP8266
- 3× Light sensors (LDR)
- L298N motor driver
- 2× DC motors with wheels
- Robot chassis
- Battery pack (3.3V-5V)

9.3.2 Light Sensor Circuit

$$\text{LDR} + 10\text{k}\Omega \text{ Resistor Voltage Divider:} \tag{9.5}$$

$$\text{VCC} \rightarrow \text{LDR} \rightarrow \text{Analog Pin} \tag{9.6}$$

$$10\text{k}\Omega \text{ Resistor} \rightarrow \text{GND} \tag{9.7}$$

9.4 Calibration and Optimization

9.4.1 Sensor Calibration

1. Test in different lighting conditions
2. Adjust threshold values
3. Consider ambient light compensation
4. Implement dynamic threshold adjustment

9.4.2 Performance Optimization

- Filter sensor noise
- Implement hysteresis
- Add speed control based on light intensity
- Consider multiple light source scenarios

Chapter 10

Advanced Concepts and Future

10.1 PID Control

10.1.1 Proportional-Integral-Derivative Control

- **Proportional**: Response proportional to error
- **Integral**: Accumulated error correction
- **Derivative**: Rate of change prediction

10.1.2 Application in Robotics

- Line following with smooth movement
- Distance maintenance
- Speed control
- Position control

10.1.3 PID Implementation Example

```
float Kp = 2.0;  // Proportional gain
float Ki = 0.1;  // Integral gain
float Kd = 0.5;  // Derivative gain

float error, lastError, integral;
float setpoint = 0;  // Target value

float calculatePID(float input) {
    error = setpoint - input;
    integral += error;
    float derivative = error - lastError;

    float output = Kp * error + Ki * integral + Kd * derivative;
    lastError = error;

```

```
16     return output;
17 }
```

Listing 10.1: PID Control Implementation

10.2 Sensor Fusion

10.2.1 Combining Multiple Sensors

- **Redundancy**: Multiple sensors for same measurement
- **Complementary**: Different sensors for different aspects
- **Kalman Filter**: Optimal estimation from noisy data

10.2.2 Examples

- IMU + GPS for navigation
- Multiple distance sensors for mapping
- Camera + ultrasonic for obstacle detection

10.2.3 Simple Sensor Fusion Example

```
1 float fuseSensors(float sensor1, float sensor2, float weight1, float
    weight2) {
2     return (sensor1 * weight1 + sensor2 * weight2) / (weight1 + weight2)
    ;
3 }
```

Listing 10.2: Simple Sensor Fusion

10.3 Machine Learning in Robotics

10.3.1 Applications

- **Computer Vision**: Object recognition
- **Path Planning**: Optimal route finding
- **Behavior Learning**: Adaptive responses
- **Predictive Maintenance**: Fault detection

10.3.2 Implementation

- TensorFlow Lite for microcontrollers
- Edge AI processing
- Cloud-based learning

10.4 IoT and Cloud Robotics

10.4.1 Cloud Integration

- **Data Logging**: Sensor data storage
- **Remote Monitoring**: Real-time status
- **Fleet Management**: Multiple robot coordination
- **Over-the-Air Updates**: Remote firmware updates

10.4.2 Technologies

- MQTT protocol
- REST APIs
- WebSocket connections
- Cloud platforms (AWS, Google Cloud, Azure)

10.4.3 MQTT Example

```
#include <PubSubClient.h>

WiFiClient espClient;
PubSubClient client(espClient);

void setup() {
    client.setServer("mqtt.example.com", 1883);
    client.setCallback(callback);
}

void publishData() {
    String data = "{\"temperature\":" + String(temp) + ",\"humidity\":"
   + String(hum) + "}";
    client.publish("robot/sensors", data.c_str());
}
```

Listing 10.3: MQTT Implementation

10.5 Advanced Algorithms

10.5.1 Path Planning

- **A* Algorithm**: Optimal path finding
- **RRT (Rapidly-exploring Random Trees)**: Dynamic planning
- **SLAM (Simultaneous Localization and Mapping)**: Environment mapping

10.5.2 Control Systems

- **Fuzzy Logic**: Human-like decision making
- **Neural Networks**: Pattern recognition
- **Genetic Algorithms**: Optimization

10.5.3 Simple A* Implementation

```
struct Node {
    int x, y;
    float g, h, f;
    Node* parent;
};

float heuristic(int x1, int y1, int x2, int y2) {
    return sqrt((x2-x1)*(x2-x1) + (y2-y1)*(y2-y1));
}
```

Listing 10.4: A* Algorithm Structure

10.6 Future Trends

10.6.1 Emerging Technologies

1. **5G Robotics**: Low-latency remote control
2. **Edge Computing**: Local AI processing
3. **Swarm Robotics**: Multi-robot coordination
4. **Soft Robotics**: Flexible and adaptive robots
5. **Bio-inspired Robotics**: Nature-inspired designs

10.6.2 Industry Applications

- **Agriculture**: Autonomous farming
- **Healthcare**: Medical assistance robots
- **Logistics**: Warehouse automation
- **Space Exploration**: Planetary rovers
- **Underwater**: Ocean exploration

10.7 Learning Resources

10.7.1 Online Platforms

- Arduino Official Documentation
- ESP8266 Community Forums
- ROS (Robot Operating System) Tutorials
- Coursera Robotics Courses
- MIT OpenCourseWare

10.7.2 Books

- "Arduino Cookbook" by Michael Margolis
- "Making Embedded Systems" by Elecia White
- "Probabilistic Robotics" by Sebastian Thrun
- "Introduction to Autonomous Mobile Robots" by Roland Siegwart

10.7.3 Projects to Try

1. **Maze Solver**: Advanced path finding
2. **Gesture Control**: Hand gesture recognition
3. **Voice Control**: Speech recognition
4. **Autonomous Navigation**: GPS + compass
5. **Multi-Robot Communication**: Robot-to-robot interaction

Appendix A

Common Error Codes and Solutions

A.1 Arduino Errors

- **"Board not found"**: Check USB connection and drivers
- **"Upload failed"**: Verify board selection and port
- **"Compilation error"**: Check syntax and libraries

A.2 Hardware Issues

- **Motors not moving**: Check power supply and connections
- **Sensors not responding**: Verify wiring and voltage
- **WiFi not connecting**: Check credentials and signal strength

Appendix B

Pin Reference Tables

B.1 Arduino Uno Pin Mapping

Pin	Function	Notes
0-1	Serial	TX/RX
2-13	Digital I/O	3,5,6,9,10,11 are PWM
A0-A5	Analog Input	10-bit resolution
5V	Power	5V output
3.3V	Power	3.3V output
GND	Ground	Common ground

Table B.1: Arduino Uno Pin Functions

B.2 NodeMCU Pin Mapping

Pin	Function	Notes
D0	GPIO16	Boot mode
D1	GPIO5	I2C SCL
D2	GPIO4	I2C SDA
D3	GPIO0	Boot mode
D4	GPIO2	Built-in LED
D5	GPIO14	SPI SCK
D6	GPIO12	SPI MISO
D7	GPIO13	SPI MOSI
D8	GPIO15	SPI SS
A0	Analog	10-bit ADC

Table B.2: NodeMCU Pin Functions

Appendix C

Component Specifications

C.1 Common Sensors

Sensor	Voltage	Current	Output	Range
IR Sensor	3.3-5V	20mA	Digital/Analog	2-30cm
Ultrasonic	5V	15mA	Digital	2-400cm
LDR	3.3-5V	1mA	Analog	Variable
Temperature	3.3-5V	1mA	Digital/Analog	-40°C to +125°C

Table C.1: Sensor Specifications

C.2 Motor Specifications

Type	Voltage	Current	Speed	Torque
DC Motor	6-12V	100-500mA	100-300 RPM	Variable
Servo Motor	4.8-6V	100-500mA	60°/sec	1-25 kg-cm
Stepper Motor	5-12V	200-800mA	Variable	High

Table C.2: Motor Specifications

Appendix D

Mathematical Formulas

D.1 Distance Calculation (Ultrasonic)

$$\text{Distance} = \frac{\text{Time} \times \text{Speed of Sound}}{2} \tag{D.1}$$

$$\text{Speed of Sound} = 340 \text{ m/s} = 0.034 \text{ cm/}\mu\text{s} \tag{D.2}$$

D.2 PWM Duty Cycle

$$\text{Duty Cycle} = \frac{\text{Pulse Width}}{\text{Period}} \times 100\% \tag{D.3}$$

$$\text{Average Voltage} = \frac{\text{Duty Cycle}}{100} \times \text{Supply Voltage} \tag{D.4}$$

D.3 Voltage Divider

$$V_{\text{out}} = V_{\text{in}} \times \frac{R_2}{R_1 + R_2} \tag{D.5}$$

D.4 Ohm's Law

$$V = I \times R \tag{D.6}$$

$$P = V \times I \tag{D.7}$$

Appendix E

WiFi Network Analysis

E.1 Network Performance

- **Signal Strength**: -30dBm (excellent) to -90dBm (poor)
- **Channel Congestion**: Use WiFi analyzer apps
- **Interference**: Avoid 2.4GHz interference sources

E.2 Security Best Practices

- Use WPA2/WPA3 encryption
- Change default passwords
- Implement MAC address filtering
- Regular firmware updates

Conclusion

This textbook has covered the fundamental concepts of Arduino robotics, from basic electronics to advanced WiFi-controlled robots. The progression from simple line following to complex light following with web interfaces demonstrates the evolution of robotics technology.

Key Takeaways

1. **Fundamentals First**: Understanding basic electronics and programming is crucial
2. **Hands-on Learning**: Building and testing robots is the best way to learn
3. **Iterative Development**: Start simple and add complexity gradually
4. **Problem Solving**: Robotics involves troubleshooting and optimization
5. **Continuous Learning**: Technology evolves rapidly; stay updated

Next Steps

- Experiment with the provided code
- Modify and improve the robots
- Explore advanced concepts
- Join robotics communities
- Contribute to open-source projects

Remember: The best way to learn robotics is by doing. Start building, keep experimenting, and never stop learning!

Happy Robotics!

This textbook is part of the ACTUATOR25 workshop organized by ROBOCEK, GCEK.