Engineering Book – Programming

GENIUS Robotics — Genie Firefighter (2026)

Team Name: K.A.D Ballers
Team Members: Kushal Sachdeva, Darsh Goel, Aaryan Singhania
School/Organization: Vasant Valley School, Delhi, India
GitHub: https://github.com/Kushal-Sachdeva78/K.A.D-Ballers-Genius-Olympiad-Robot

---

1. Programming Overview

This programming book documents how our robot is controlled in **tele-operated mode** using:

- **Phone Bluetooth Serial Controller App** (driver input)
- **HC-05 Bluetooth module** (wireless serial link)
- **Arduino Nano** (main microcontroller)
- **2× L298N motor drivers** (drive motor direction control)
- **3× high-torque servos** (arm, gripper, basket flip)

The code reads **single-character commands** from Bluetooth and immediately executes:

- A drivetrain movement function (mecanum patterns)
- Or a servo action (arm up/down, gripper open/close, basket load/unload)

---

2. Software Environment

2.1 Platform

- **Microcontroller:** Arduino Nano (ATmega328P)
- **Programming language:** Arduino C/C++
- **Tooling:** Arduino IDE (or compatible)

2.2 Libraries Used

- SoftwareSerial.h
  Used to communicate with the HC-05 Bluetooth module using pins D2 (RX) and D3 (TX).
- Servo.h
  Used to generate servo control pulses for three servos.

---

3. Control System Architecture (Software)

3.1 Data Flow

Driver phone app → HC-05 Bluetooth → Arduino Nano → output actions

- If command is a movement command: set motor direction pins for mecanum motion
- If command is a manipulator command: set servo angle to a predefined position

3.2 Why Single-Character Commands?

- Very fast, simple, and reliable over Bluetooth serial
- Easy for driver training (each button sends one character)
- Easy debugging (we print commands to Serial monitor)

---

4. Communication Protocol (Bluetooth)

4.1 HC-05 Settings (in our code)

- Baud rate: **9600**
- Module connection: paired with phone via Bluetooth
- Serial link:
  - Arduino **D2** = SoftwareSerial RX (from HC-05 TX)
  - Arduino **D3** = SoftwareSerial TX (to HC-05 RX)

4.2 Command Message Format

- Each message is **one ASCII character**, e.g. 'F', 'L', 'O', 'U'.
- The Arduino reads it using bluetooth.read() and runs a switch(command).

---

5. Hardware Pin Mapping (Code ↔ Wiring)

5.1 Motor Driver Direction Pins (L298N)

We drive motors using direction pins (IN1/IN2 pairs).
(Note: speed control PWM is not implemented in this version; only direction control.)

Front Left Motor

- FL_IN1 = 4
- FL_IN2 = 5

Front Right Motor

- FR_IN1 = 6
- FR_IN2 = 7

Back Left Motor

- BL_IN1 = A0
- BL_IN2 = A1

Back Right Motor

- BR_IN1 = A2
- BR_IN2 = A3

5.2 Servo Signal Pins

- Basket/Holder servo: **D9**
- Arm servo: **D10**
- Gripper servo: **D11**

---

6. Command Map (Driver Controls)

6.1 Movement Commands

- F = move forward
- B = move backward

- L = strafe left
- R = strafe right
- S = stop all motors
- G = diagonal front-left
- H = diagonal front-right
- I = diagonal back-left
- J = diagonal back-right
- X = rotate clockwise
- Y = rotate counterclockwise

6.2 Manipulator Commands

- U = arm up
- D = arm down
- O = gripper open
- C = gripper close
- M = basket/holder load position
- N = basket/holder unload position

---

7. Motion Control Logic (Mecanum Patterns)

Our robot uses mecanum wheels, so different combinations of motor directions produce different motion.

7.1 Forward / Backward

All 4 wheels run forward (or reverse) together.

7.2 Strafing Left / Right

Opposite corners run forward vs backward to generate sideways motion.

7.3 Diagonals

Two motors are driven while the other two are stopped to create diagonal drift (quick driver alignment).

7.4 Rotation

Left-side wheels forward while right-side wheels reverse (or vice versa) to rotate in place.

---

8. Servo Control Logic (Arm + Gripper + Basket)

We use fixed servo angles for repeatable performance:

8.1 Basket/Holder Servo

- HOLDER_LOAD = 0
- HOLDER_UNLOAD = 70

8.2 Arm Servo

- ARM_DOWN = 6
- ARM_UP = 100
  Arm movement is smoothed by stepping one degree at a time.

8.3 Gripper Servo

- GRIP_CLOSE = 0
- GRIP_OPEN = 30

8.4 Why "Preset Angles"?

- Improves consistency under time pressure
- Reduces driver error
- Makes tuning easier (change constants, re-upload)

---

9. Initialization & Startup Behavior

On startup (setup()):

1. Configure all motor pins as outputs
2. Stop motors for safety
3. Attach servos to pins and set them to starting positions
4. Start Bluetooth serial and USB serial output for debugging

Startup servo positions:

- Basket: load
- Arm: down
- Gripper: closed

---

10. Safety and Reliability Features (Current Version)

10.1 Motor Stop Command

- `S` stops all motors immediately.

10.2 “No-repeat” Servo Writes

- The code tracks current servo state (`currentHolder/currentArm/currentGrip`) and avoids re-writing if already at that position.
  This reduces jitter and unnecessary servo movement.

10.3 Smooth Arm Motion

- Arm moves gradually (step-by-step) to reduce shaking and dropping objects.

---

11. Testing & Debugging

11.1 Debug Output

We print received commands to Serial Monitor:

- Helps confirm the phone app is sending the expected letters
- Helps identify missed/incorrect commands

11.2 Functional Test Checklist

Bluetooth

- Pair HC-05 to phone
- Confirm app sends characters

- Confirm Arduino receives and prints command

Drive

- Test F/B/L/R and verify motor directions match expected motion
- If motion is inverted, swap motor wires or adjust direction logic

Servos

- Test O/C for gripper, U/D for arm, M/N for basket
- Adjust angle constants if needed for mechanical endpoints

---

12. Version Control & GitHub Repository (Programming Assets)

GitHub Repo Link:
https://github.com/Kushal-Sachdeva78/K.A.D-Ballers-Genius-Olympiad-Robot

13. Future Programming Improvements (Planned)

13.1 PWM Speed Control (Fine driving near borders/buckets)

- Use L298N enable pins (ENA/ENB) with PWM for:
  - slow mode (precision)
  - fast mode (travel)

13.2 Failsafe “Dead-Man Stop”

- Auto-stop motors if no command received for X milliseconds
  (prevents runaway if Bluetooth disconnects)

13.3 Macro Commands (One-button actions)

Example macros:

- “Pick” sequence: arm down → open → close → arm up
- “Dump” sequence: approach bin → basket unload → basket load

13.4 Automatic Intake Logic (Rubber-band roller)

- Add a motor/servo control for intake rollers
- Add a simple toggle command: intake ON/OFF

13.5 Scissor Lift Control (Basket elevation)

- Add lift states: lift down / lift up
- Add optional limit switch support for safety

Appendix A — Full Source Code (Arduino Nano)

```
#include <SoftwareSerial.h>
#include <Servo.h>

SoftwareSerial bluetooth(2, 3);

Servo ballHolderServo;
Servo ballArmServo;
Servo gripperServo;

#define FL_IN1 4
#define FL_IN2 5
#define FR_IN1 6
#define FR_IN2 7

#define BL_IN1 A0
#define BL_IN2 A1
#define BR_IN1 A2
#define BR_IN2 A3

const int HOLDER_LOAD   = 0;
const int HOLDER_UNLOAD = 70;

const int ARM_DOWN = 6;
const int ARM_UP   = 100;
const int GRIP_CLOSE = 0;
const int GRIP_OPEN  = 30;

int currentHolder = HOLDER_LOAD;
int currentArm    = ARM_DOWN;
int currentGrip   = GRIP_CLOSE;

void setup() {
  pinMode(FL_IN1, OUTPUT); pinMode(FL_IN2, OUTPUT);
  pinMode(FR_IN1, OUTPUT); pinMode(FR_IN2, OUTPUT);
  pinMode(BL_IN1, OUTPUT); pinMode(BL_IN2, OUTPUT);
  pinMode(BR_IN1, OUTPUT); pinMode(BR_IN2, OUTPUT);

  stopMotors();

  ballHolderServo.attach(9);
  ballArmServo.attach(10);
  gripperServo.attach(11);

  ballHolderServo.write(HOLDER_LOAD);
  ballArmServo.write(ARM_DOWN);
  gripperServo.write(GRIP_CLOSE);

  bluetooth.begin(9600);
  Serial.begin(9600);
  Serial.println("Mecanum Robot Ready - Directions Updated per Table");
  Serial.println("Movement: F B L R S G H I J X Y");
  Serial.println("Other:    C O U D M N");
}

void loop() {
  if (bluetooth.available()) {
    char command = bluetooth.read();
    Serial.print("Cmd: "); Serial.println(command);

    switch (command) {
      case 'F': moveForward();   break;
      case 'B': moveBackward();  break;
      case 'L': strafeLeft();    break;
      case 'R': strafeRight();   break;
      case 'S': stopMotors();    break;

      case 'G': diagonalFrontLeft();  break;
      case 'H': diagonalFrontRight(); break;
      case 'I': diagonalBackLeft();   break;
      case 'J': diagonalBackRight();  break;

      case 'X': rotateClockwise();        break;
      case 'Y': rotateCounterClockwise(); break;

      case 'C': setGripper(GRIP_CLOSE); break;
      case 'O': setGripper(GRIP_OPEN);  break;

      case 'U': moveArmTo(ARM_UP);   break;
      case 'D': moveArmTo(ARM_DOWN); break;

      case 'M': setBallHolder(HOLDER_LOAD);   break;
      case 'N': setBallHolder(HOLDER_UNLOAD); break;
    }
  }
}

void moveForward() {
  digitalWrite(FL_IN1, HIGH); digitalWrite(FL_IN2, LOW);
  digitalWrite(FR_IN1, HIGH); digitalWrite(FR_IN2, LOW);
  digitalWrite(BL_IN1, HIGH); digitalWrite(BL_IN2, LOW);
  digitalWrite(BR_IN1, HIGH); digitalWrite(BR_IN2, LOW);
}

void moveBackward() {
  digitalWrite(FL_IN1, LOW);  digitalWrite(FL_IN2, HIGH);
  digitalWrite(FR_IN1, LOW);  digitalWrite(FR_IN2, HIGH);
  digitalWrite(BL_IN1, LOW);  digitalWrite(BL_IN2, HIGH);
  digitalWrite(BR_IN1, LOW);  digitalWrite(BR_IN2, HIGH);
}

void strafeLeft() {
  digitalWrite(FL_IN1, LOW);  digitalWrite(FL_IN2, HIGH);
  digitalWrite(FR_IN1, HIGH); digitalWrite(FR_IN2, LOW);
  digitalWrite(BL_IN1, HIGH); digitalWrite(BL_IN2, LOW);
  digitalWrite(BR_IN1, LOW);  digitalWrite(BR_IN2, HIGH);
}

void strafeRight() {
  digitalWrite(FL_IN1, HIGH); digitalWrite(FL_IN2, LOW);
  digitalWrite(FR_IN1, LOW);  digitalWrite(FR_IN2, HIGH);
  digitalWrite(BL_IN1, LOW);  digitalWrite(BL_IN2, HIGH);
  digitalWrite(BR_IN1, HIGH); digitalWrite(BR_IN2, LOW);
}

void diagonalFrontLeft() {
  digitalWrite(FL_IN1, HIGH); digitalWrite(FL_IN2, LOW);
  digitalWrite(FR_IN1, LOW);  digitalWrite(FR_IN2, LOW);
  digitalWrite(BL_IN1, LOW);  digitalWrite(BL_IN2, LOW);
  digitalWrite(BR_IN1, HIGH); digitalWrite(BR_IN2, LOW);
}

void diagonalFrontRight() {
  digitalWrite(FL_IN1, LOW);  digitalWrite(FL_IN2, LOW);
  digitalWrite(FR_IN1, HIGH); digitalWrite(FR_IN2, LOW);
  digitalWrite(BL_IN1, HIGH); digitalWrite(BL_IN2, LOW);
  digitalWrite(BR_IN1, LOW);  digitalWrite(BR_IN2, LOW);
}

void diagonalBackLeft() {
  digitalWrite(FL_IN1, LOW);  digitalWrite(FL_IN2, LOW);
  digitalWrite(FR_IN1, LOW);  digitalWrite(FR_IN2, LOW);
  digitalWrite(BL_IN1, HIGH); digitalWrite(BL_IN2, LOW);
  digitalWrite(BR_IN1, LOW);  digitalWrite(BR_IN2, HIGH);
}

void diagonalBackRight() {
  digitalWrite(FL_IN1, HIGH); digitalWrite(FL_IN2, LOW);
  digitalWrite(FR_IN1, LOW);  digitalWrite(FR_IN2, HIGH);
  digitalWrite(BL_IN1, LOW);  digitalWrite(BL_IN2, LOW);
  digitalWrite(BR_IN1, LOW);  digitalWrite(BR_IN2, LOW);
}

void rotateClockwise() {
  digitalWrite(FL_IN1, HIGH); digitalWrite(FL_IN2, LOW);
  digitalWrite(FR_IN1, LOW);  digitalWrite(FR_IN2, HIGH);
  digitalWrite(BL_IN1, HIGH); digitalWrite(BL_IN2, LOW);
  digitalWrite(BR_IN1, LOW);  digitalWrite(BR_IN2, HIGH);
}

void rotateCounterClockwise() {
  digitalWrite(FL_IN1, LOW);  digitalWrite(FL_IN2, HIGH);
  digitalWrite(FR_IN1, HIGH); digitalWrite(FR_IN2, LOW);
  digitalWrite(BL_IN1, LOW);  digitalWrite(BL_IN2, HIGH);
  digitalWrite(BR_IN1, HIGH); digitalWrite(BR_IN2, LOW);
}

void stopMotors() {
  digitalWrite(FL_IN1, LOW); digitalWrite(FL_IN2, LOW);
  digitalWrite(FR_IN1, LOW); digitalWrite(FR_IN2, LOW);
  digitalWrite(BL_IN1, LOW); digitalWrite(BL_IN2, LOW);
  digitalWrite(BR_IN1, LOW); digitalWrite(BR_IN2, LOW);
}

void setBallHolder(int target) {
  if (target == currentHolder) return;
  ballHolderServo.write(target);
  currentHolder = target;
  delay(300);
}

void setGripper(int target) {
  if (target == currentGrip) return;
  gripperServo.write(target);
  currentGrip = target;
  delay(300);
}

void moveArmTo(int target) {
  if (target == currentArm) return;

  int start = currentArm;
  int step = (target > start) ? 1 : -1;
  int delayMs = 30;
  for (int pos = start; pos != target; pos += step) {
    ballArmServo.write(pos);
    delay(delayMs);
  }
  ballArmServo.write(target);
  currentArm = target;
}
```