Stepper Motor Control Script: Detailed Documentation and Explanation

This Arduino script is designed to control a stepper motor using four digital pins connected to a motor driver board. It allows the motor to rotate in both clockwise (CW) and counterclockwise (CCW) directions, either continuously or for a specified number of steps. Commands are received via the serial interface, allowing for remote control over a serial connection.

Global Definitions and Variables

```
#define STEPPER_PIN_1 9
#define STEPPER_PIN_2 10
#define STEPPER_PIN_3 11
#define STEPPER_PIN_4 12

int step_number = 0;
bool motor_running = false;  // Flag to keep track of motor state
bool direction = true;       // True for clockwise, false for
counterclockwise
```

- **STEPPER_PIN_1 to STEPPER_PIN_4:** Define the digital pins (9, 10, 11, 12) connected to the stepper motor driver board to control the motor's four coils.
- **step_number:** Keeps track of the current step in the sequence (ranges from 0 to 3). This determines which coils to energize for the motor to step correctly.
- **motor_running:** A flag indicating whether the motor is running. It is set to `true` when the motor should rotate continuously and `false` when it should stop.
- **direction:** A boolean variable indicating the direction of rotation. `true` for clockwise (CW) and `false` for counterclockwise (CCW).

Setup Function

```
  void setup() {
  pinMode(STEPPER_PIN_1, OUTPUT);
  pinMode(STEPPER_PIN_2, OUTPUT);
  pinMode(STEPPER_PIN_3, OUTPUT);
  pinMode(STEPPER_PIN_4, OUTPUT);

  Serial.begin(9600);  // Initialize serial communication at 9600 baud
rate
  Serial.println("Stepper motor control ready.");
}
```

- **pinMode:** Configures the four digital pins as outputs to control the stepper motor's coils.
- **Serial.begin(9600):** Initializes the serial communication at a baud rate of 9600. This allows the Arduino to communicate with a computer or another serial device.
- **Serial.println("Stepper motor control ready.");** Prints a message to the serial monitor to indicate that the motor control setup is ready.

```
void loop() {
  if (Serial.available() > 0) {
    String command = Serial.readStringUntil('\n');  // Read the
command until newline
    command.trim();  // Remove any leading or trailing whitespace

    Serial.println("Received command: " + command); // Debug output

    if (command == "CW") {
      motor_running = true;
      direction = true;  // Set direction to clockwise
      Serial.println("Motor running clockwise");
    } else if (command == "CCW") {
      motor_running = true;
      direction = false; // Set direction to counterclockwise
      Serial.println("Motor running counterclockwise");
    } else if (command == "STOP") {
      motor_running = false;  // Stop the motor
      Serial.println("Motor stopped");
    } else if (command.startsWith("STEP,CW,")) {
      int steps = command.substring(8).toInt();  // Extract the number
of steps after "STEP,CW,"
      if (steps > 0) {
        Serial.println("Moving motor " + String(steps) + " steps
clockwise");
        for (int i = 0; i < steps; i++) {
          OneStep(true);  // Move clockwise
          delay(5);  // Adjust delay to control speed
        }
        Serial.println("Completed " + String(steps) + " steps
clockwise");
      } else {
```

```
        Serial.println("Invalid step number.");
      }
    } else if (command.startsWith("STEP,CCW,")) {
      int steps = command.substring(9).toInt();  // Extract the number
of steps after "STEP,CCW,"
      if (steps > 0) {
        Serial.println("Moving motor " + String(steps) + " steps
counterclockwise");
        for (int i = 0; i < steps; i++) {
          OneStep(false);  // Move counterclockwise
          delay(5);  // Adjust delay to control speed
        }
        Serial.println("Completed " + String(steps) + " steps
counterclockwise");
      } else {
        Serial.println("Invalid step number.");
      }
    } else if (command.startsWith("STEP,")) {
      String stepValueStr = command.substring(5);  // Extract
everything after "STEP,"
      int steps = stepValueStr.toInt();  // Convert the extracted part
to an integer

      Serial.println("Parsed step number: " + String(steps)); // Debug
output

      if (steps > 0) {
        Serial.println("Moving motor " + String(steps) + " steps");
        for (int i = 0; i < steps; i++) {
          OneStep(direction);  // Adjust the direction if needed
          delay(5);  // Adjust delay to control speed
        }
        Serial.println("Completed " + String(steps) + " steps");
      } else {
        Serial.println("Invalid step number."); // Invalid step number
if conversion fails
      }
    } else {
```

```
      Serial.println("Unknown command: " + command);  // Handle
unknown commands
    }
  }

  // Run the motor if it is set to running state
  if (motor_running) {
    OneStep(direction);  // Make one step in the current direction
    delay(2);            // Control the speed
  }
}
```

- **Serial.available() > 0:** Checks if there is data available to read from the serial buffer. If there is, it processes the command.
- **Serial.readStringUntil('\n'):** Reads the incoming serial data until a newline character (`\n`) is detected.
- **command.trim():** Removes any leading or trailing whitespace from the command.
- **Serial.println("Received command: " + command):** Prints the received command to the serial monitor for debugging purposes.

Command Handling:

1. **CW:** Sets the motor to run clockwise continuously.
2. **CCW:** Sets the motor to run counterclockwise continuously.
3. **STOP:** Stops the motor.
4. **STEP,CW,<number>:** Moves the motor clockwise for a specified number of steps.
5. **STEP,CCW,<number>:** Moves the motor counterclockwise for a specified number of steps.
6. **STEP,<number>:** Moves the motor a specified number of steps in the current direction (either CW or CCW).

If the command is not recognized, it prints an error message to the serial monitor.

OneStep Function

```
void OneStep(bool dir) {
  if (dir) {  // Clockwise direction
    switch (step_number) {
      case 0:
        digitalWrite(STEPPER_PIN_1, HIGH);
        digitalWrite(STEPPER_PIN_2, LOW);
```

```
      digitalWrite(STEPPER_PIN_3, LOW);
      digitalWrite(STEPPER_PIN_4, LOW);
      break;
    case 1:
      digitalWrite(STEPPER_PIN_1, LOW);
      digitalWrite(STEPPER_PIN_2, HIGH);
      digitalWrite(STEPPER_PIN_3, LOW);
      digitalWrite(STEPPER_PIN_4, LOW);
      break;
    case 2:
      digitalWrite(STEPPER_PIN_1, LOW);
      digitalWrite(STEPPER_PIN_2, LOW);
      digitalWrite(STEPPER_PIN_3, HIGH);
      digitalWrite(STEPPER_PIN_4, LOW);
      break;
    case 3:
      digitalWrite(STEPPER_PIN_1, LOW);
      digitalWrite(STEPPER_PIN_2, LOW);
      digitalWrite(STEPPER_PIN_3, LOW);
      digitalWrite(STEPPER_PIN_4, HIGH);
      break;
  }
} else {  // Counterclockwise direction
  switch (step_number) {
    case 0:
      digitalWrite(STEPPER_PIN_1, LOW);
      digitalWrite(STEPPER_PIN_2, LOW);
      digitalWrite(STEPPER_PIN_3, LOW);
      digitalWrite(STEPPER_PIN_4, HIGH);
      break;
    case 1:
      digitalWrite(STEPPER_PIN_1, LOW);
      digitalWrite(STEPPER_PIN_2, LOW);
      digitalWrite(STEPPER_PIN_3, HIGH);
      digitalWrite(STEPPER_PIN_4, LOW);
      break;
    case 2:
      digitalWrite(STEPPER_PIN_1, LOW);
```

```
        digitalWrite(STEPPER_PIN_2, HIGH);
        digitalWrite(STEPPER_PIN_3, LOW);
        digitalWrite(STEPPER_PIN_4, LOW);
        break;
      case 3:
        digitalWrite(STEPPER_PIN_1, HIGH);
        digitalWrite(STEPPER_PIN_2, LOW);
        digitalWrite(STEPPER_PIN_3, LOW);
        digitalWrite(STEPPER_PIN_4, LOW);
        break;
    }
  }

  step_number++;
  if (step_number > 3) {
    step_number = 0;
  }
}
```

- **OneStep(bool dir):** This function takes a boolean parameter `dir` (direction). If `dir` is `true`, it steps the motor clockwise; otherwise, it steps counterclockwise.
- **digitalWrite(STEPPER_PIN_X, HIGH/LOW):** Sets the respective stepper motor control pins to `HIGH` or `LOW`, energizing or de-energizing the coils in the correct sequence to move the motor one step in the specified direction.
- **Step Sequence Logic:**
    - The function uses a `switch` statement to determine which coils to energize based on the current step number (`step_number`). This ensures that the motor moves in a precise manner.
    - When `dir` is `true` (clockwise):
        - The sequence of coil activation follows a pattern: `1 -> 2 -> 3 -> 4`. This pattern advances the motor in a clockwise direction.
    - When `dir` is `false` (counterclockwise):
        - The sequence of coil activation is reversed: `4 -> 3 -> 2 -> 1`. This pattern makes the motor rotate in the opposite direction.
- **Incrementing `step_number`:**
    - After each step, `step_number` is incremented by 1.
    - If `step_number` exceeds 3 (the highest step index in the sequence), it is reset to 0. This creates a loop in the step sequence, allowing continuous rotation.

Summary of the Script Functionality

1. **Command Reception:**
    - The script waits for commands via the serial interface. The commands control the motor's behavior, such as starting, stopping, changing direction, or moving a specific number of steps.
2. **Continuous Rotation:**
    - If the motor is set to run continuously (commands `"CW"` or `"CCW"`), the script repeatedly calls `OneStep()` in the main loop to keep the motor rotating in the specified direction.
3. **Controlled Steps:**
    - For commands like `"STEP,CW,<number>"` or `"STEP,CCW,<number>"`, the motor moves the specified number of steps in the given direction. This allows precise positioning or limited rotation.
4. **Feedback and Debugging:**
    - Throughout the script, `Serial.println()` statements provide feedback and debugging information to the user, indicating what the motor is doing or if there are any issues (e.g., an invalid command).

Key Concepts

- **Stepper Motor Control:**
    - Stepper motors move in discrete steps. By carefully controlling the sequence and timing of coil energization, the motor can be made to rotate in precise increments.
- **Direction Control:**
    - By changing the order in which the coils are energized, the motor can be rotated clockwise or counterclockwise.
- **Serial Communication:**
    - The script utilizes serial communication to receive commands from a connected device (e.g., a computer) and provide feedback. This enables remote control of the motor.

Possible Enhancements

- **Speed Control:**
    - The current script uses a fixed delay (`delay(5)`) to control the speed of the motor. This could be enhanced by allowing dynamic speed adjustments based on user input.
- **Acceleration and Deceleration:**
    - Implementing acceleration and deceleration profiles could make the motor's movement smoother, which is particularly useful in applications requiring precise control.

- **Error Handling:**
    - The script already handles some common errors, such as invalid commands or step numbers. More sophisticated error handling could be added, such as checking for motor stall conditions or adding timeout features.

Conclusion

This script provides a basic yet functional control mechanism for a stepper motor using Arduino. By leveraging serial communication, it allows for flexible and dynamic control of the motor's behavior, making it suitable for various applications, such as robotics, CNC machines, or automated systems.