Báo cáo Nghiên cứu Chuyên sâu: Kiến trúc Điều khiển Robot Hàn 6-DOF Hiệu năng cao trên nền tảng PC và Teensy 4.1 grblHAL

Tóm tắt Điều hành

Báo cáo này trình bày một phân tích kỹ thuật toàn diện và lộ trình triển khai chi tiết cho việc xây dựng bộ điều khiển cánh tay robot công nghiệp 6 bậc tự do (6-DOF), được tối ưu hóa cho ứng dụng hàn hồ quang. Kiến trúc được đề xuất sử dụng mô hình "Split-Brain" (Phân tách Xử lý), trong đó PC (viết bằng C#) chịu trách nhiệm tính toán Động học Nghịch (Inverse Kinematics - IK) và Hoạch định Quỹ đạo (Trajectory Planning) thông qua thư viện Ruckig, trong khi vi điều khiển Teensy 4.1 chạy firmware grblHAL đóng vai trò là bộ tạo xung bước thời gian thực (Real-time Step Generator).

Tài liệu này giải quyết trực tiếp các thách thức kỹ thuật cốt lõi: cấu hình phần cứng cho 6 trục đồng bộ trên vi xử lý iMXRT1062, chiến lược "Planner Bypass" để vô hiệu hóa bộ hoạch định chuyển động nội tại của grblHAL nhằm nhường quyền kiểm soát cho Ruckig, và tối ưu hóa giao thức truyền thông USB CDC để đạt được mật độ lệnh cao mà không gây ra hiện tượng trễ (latency) hoặc cạn kiệt bộ đệm (starvation).

---

1. Tổng quan Kiến trúc Hệ thống: Mô hình Điều khiển Lai PC-MCU

1.1 Sự chuyển dịch từ MCU-Centric sang PC-Centric

Trong các hệ thống CNC truyền thống (như máy phay 3 trục chạy Grbl trên Arduino Uno), toàn bộ quy trình từ phân tích G-code, hoạch định biên dạng vận tốc (velocity profile), đến tạo xung bước đều diễn ra trên vi điều khiển (MCU). Mô hình này hoạt động hiệu quả cho các trục tuyến tính độc lập. Tuy nhiên, với robot hàn 6-DOF, mối quan hệ giữa không gian khớp (joint space) và không gian làm việc (Cartesian space) là phi tuyến tính và phức tạp. Việc thực hiện giải thuật IK và xử lý các điểm kỳ dị (singularities) trong thời gian thực vượt quá khả năng của các MCU 8-bit hoặc thậm chí 32-bit tầm trung nếu phải duy trì tần số phát xung cao.

Do đó, kiến trúc **PC-based** kết hợp với **Teensy 4.1** là giải pháp tối ưu.

- **Tầng PC (C# + Ruckig):** Đóng vai trò "Vỏ não" (Cortex). Nó xử lý các thuật toán bậc cao như phát hiện va chạm, nội suy đường hàn (weaving), và đặc biệt là tính toán quỹ đạo giới

hạn giật (jerk-limited trajectory) sử dụng thư viện Ruckig.[1]

- **Tầng MCU (Teensy 4.1 + grblHAL):** Đóng vai trò "Thân não" (Brainstem). Nhiệm vụ duy nhất là nhận luồng dữ liệu vị trí/vận tốc từ PC và chuyển đổi chúng thành tín hiệu Step/Dir chính xác đến microsecond, sử dụng các bộ định thời phần cứng (hardware timers) mạnh mẽ của chip iMXRT1062.[3]

1.2 Yêu cầu Thời gian thực cho Ứng dụng Hàn

Khác với các ứng dụng pick-and-place, robot hàn yêu cầu chuyển động liên tục (continuous path) với vận tốc không đổi (constant velocity) để đảm bảo độ ngấu và bề mặt mối hàn đồng nhất. Bất kỳ sự gián đoạn nào trong luồng dữ liệu (buffer starvation) sẽ dẫn đến việc robot dừng lại, tạo ra các khuyết tật hàn. Do đó, trọng tâm của báo cáo này không chỉ là cấu hình 6 trục mà còn là **độ ổn định của độ trễ (latency stability)** và **quản lý bộ đệm (buffer management)**.[4]

---

2. Nền tảng Phần cứng và Firmware: Teensy 4.1 và grblHAL

2.1 Sức mạnh của Vi xử lý iMXRT1062

Teensy 4.1 được trang bị vi xử lý NXP iMXRT1062 (ARM Cortex-M7) chạy ở xung nhịp 600 MHz. Đây là yếu tố then chốt cho phép grblHAL xử lý 6 trục đồng thời.

- **FPU (Floating Point Unit):** Hỗ trợ tính toán số thực dấu phẩy động bằng phần cứng, loại bỏ sai số làm tròn tích lũy thường thấy trên các hệ thống 8-bit khi tính toán tọa độ cho chuỗi robot dài.[6]
- **Bộ nhớ:** Với 1MB RAM, Teensy 4.1 cho phép thiết lập kích thước bộ đệm lập kế hoạch (planner buffer) lên tới hàng nghìn khối lệnh, tạo ra một "bể chứa" an toàn để hấp thụ các dao động trễ từ hệ điều hành Windows trên PC.[4]
- **Tốc độ xung:** Khả năng tạo xung bước lên đến hàng trăm kHz trên cả 6 trục mà không gây ra hiện tượng jitter (rung pha), đảm bảo chuyển động mượt mà cho các động cơ servo hoặc stepper lai.

2.2 Kiến trúc grblHAL

grblHAL là bản viết lại của Grbl, tách biệt phần lõi (Core) xử lý G-code khỏi phần điều khiển phần cứng (Driver).

- **Core:** Chịu trách nhiệm phân tích cú pháp G-code, quản lý trạng thái máy và lập kế hoạch chuyển động cơ bản.
- **Driver (iMXRT1062):** Cung cấp lớp trừu tượng phần cứng (HAL) để giao tiếp với GPIO, Timer, và các giao thức truyền thông như USB hoặc Ethernet.[7]
- **Khả năng mở rộng:** Hỗ trợ tới 6 trục (hoặc nhiều hơn qua plugin) và các tính năng như

thay đổi công cụ (tool change), điều khiển trục xoay (spindle), và các tín hiệu I/O phụ trợ.[7]

---

3. Giải quyết Vấn đề Cấu hình 6 Trục (X, Y, Z, A, B, C)

Việc kích hoạt 6 trục trên grblHAL đòi hỏi sự can thiệp sâu vào các tệp cấu hình mã nguồn trước khi biên dịch. Mặc định, grblHAL thường được cấu hình cho 3 trục (XYZ) hoặc 4 trục (XYZA).

3.1 Tùy biến tệp my_machine.h

Tệp my_machine.h là nơi định nghĩa các tham số đặc thù của máy, ghi đè lên các giá trị mặc định trong config.h. Để kích hoạt 6 trục, bạn cần thực hiện các thay đổi sau:

C

```
// my_machine.h - Cấu hình cho Robot 6-DOF

// Định nghĩa bo mạch (Sử dụng Phil Barrett's board hoặc custom)
#define BOARD_T41U5XBB // Hoặc BOARD_MY_MACHINE nếu tự thiết kế PCB

// QUAN TRỌNG: Định nghĩa số lượng trục
// Giá trị này kích hoạt các cấu trúc dữ liệu nội bộ cho 6 trục
#define N_AXIS 6

// Định nghĩa tên trục (Tùy chọn, chuẩn robot thường là 1-6 hoặc XYZABC)
#define AXIS_4_NAME 'A' // Trục xoay cổ tay 1
#define AXIS_5_NAME 'B' // Trục xoay cổ tay 2
#define AXIS_6_NAME 'C' // Trục xoay cổ tay 3 (Flange)

// Cấu hình giao tiếp USB CDC (Serial ảo tốc độ cao)
#define USB_SERIAL_CDC 2
```

Theo dữ liệu từ mã nguồn [7], macro N_AXIS là công tắc chính. Nếu không định nghĩa nó là 6, firmware sẽ chỉ cấp phát bộ nhớ cho 3 trục mặc định, dẫn đến lỗi khi gửi lệnh cho trục A, B, C.

3.2 Bản đồ Chân (Pin Mapping) trong map.h

Chỉ định nghĩa N_AXIS 6 là chưa đủ; driver cần biết chân vật lý nào trên Teensy điều khiển tín hiệu Step/Dir cho trục 4, 5, và 6.

Trong thư mục driver iMXRT1062, tệp ánh xạ (thường là my_machine_map.h hoặc được nhúng

trong driver.c) cần được chỉnh sửa.

Lưu ý quan trọng: Bo mạch Teensy 4.1 có số lượng chân giới hạn. Nếu sử dụng Ethernet (cần thiết cho các ứng dụng công nghiệp chống nhiễu), một số chân sẽ bị chiếm dụng.[8] Bạn cần tham khảo sơ đồ chân (pinout) của Teensy 4.1 để tránh xung đột.

Ví dụ cấu hình chân cho 6 trục (giả định):

- **Trục 1 (X):** Step Pin 0, Dir Pin 1
- **Trục 2 (Y):** Step Pin 2, Dir Pin 3
- **Trục 3 (Z):** Step Pin 4, Dir Pin 5
- **Trục 4 (A):** Step Pin 6, Dir Pin 7
- **Trục 5 (B):** Step Pin 8, Dir Pin 9
- **Trục 6 (C):** Step Pin 24, Dir Pin 25 (Sử dụng các chân phía sau bo mạch nếu cần).[7]

Nếu bạn sử dụng bo mạch breakout có sẵn như của Phil Barrett (T41U5XBB), bo mạch này thường hỗ trợ tối đa 5 trục.[7] Để có trục thứ 6, bạn có thể cần phải "câu dây" (dead-bug wiring) từ các chân GPIO chưa sử dụng của Teensy hoặc thiết kế một bo mạch PCB tùy chỉnh (Custom Shield).

3.3 Chu trình Homing cho Robot Khớp nối

Chu trình Homing (Về gốc) của máy CNC (Z lên trước, sau đó X/Y) là **nguy hiểm** cho robot 6 trục. Robot có thể va chạm với chính nó hoặc môi trường nếu thứ tự về gốc không đúng.

Trong config.h hoặc my_machine.h, bạn cần định nghĩa lại các macro HOMING_CYCLE_x.

Chiến lược Homing đề xuất cho Robot Hàn:

1. **Bước 1:** Đưa các trục cổ tay (A, B, C) về vị trí trung gian an toàn.
2. **Bước 2:** Thu gọn cánh tay (Trục 2, 3) để tránh va chạm trần/sàn.
3. **Bước 3:** Xoay đế (Trục 1) về vị trí 0.

Mã cấu hình tương ứng [10]:

C

```
// Vô hiệu hóa homing mặc định
//#define HOMING_CYCLE_0 (1<<Z_AXIS)

// Thiết lập chu trình mới
#define HOMING_CYCLE_0 ((1<<AXIS_4)|(1<<AXIS_5)|(1<<AXIS_6)) // Home cổ tay
#define HOMING_CYCLE_1 ((1<<AXIS_2)|(1<<AXIS_3))             // Thu gọn tay
```

```
#define HOMING_CYCLE_2 (1<<AXIS_1)                // Xoay đế
```

Lưu ý: Bạn cần cảm biến giới hạn (Limit Switch) trên tất cả 6 trục để thực hiện điều này.

---

4. Chiến lược 'Planner Bypass' (Vượt qua Bộ lập kế hoạch)

Đây là yêu cầu cốt lõi để tích hợp Ruckig. grblHAL mặc định sử dụng bộ lập kế hoạch hình thang (trapezoidal planner) với khả năng nhìn trước (look-ahead). Tuy nhiên, Ruckig trên PC tạo ra quỹ đạo S-curve (giới hạn đạo hàm bậc 3 - Jerk) mượt mà hơn nhiều.[2] Nếu nạp quỹ đạo Ruckig vào bộ lập kế hoạch của Grbl, hai bộ lọc sẽ xung đột, gây ra sai số và giật cục.

Chúng ta không thể xóa bỏ hoàn toàn bộ lập kế hoạch của Grbl (vì nó liên quan đến việc tạo xung), nhưng có thể cấu hình để nó trở nên "trong suốt" (transparent).

4.1 Vô hiệu hóa Giảm tốc tại Góc (Cornering)

Grbl sử dụng tham số "Junction Deviation" (Sai lệch tiếp tuyến - $11) để quyết định tốc độ khi đi qua các góc nối giữa hai đoạn G-code.

- **Vấn đề:** Khi streaming quỹ đạo từ Ruckig, đường cong được xấp xỉ bằng hàng nghìn đoạn thẳng nhỏ (G1). Góc giữa các đoạn này gần như bằng 0 (thẳng hàng), nhưng nhiễu tính toán có thể tạo ra các góc nhỏ. Grbl có thể hiểu nhầm và giảm tốc độ, gây rung động.
- **Giải pháp:** Đặt giá trị $11 cực lớn (ví dụ: 0.1 hoặc 1.0 mm, so với mặc định 0.01). Điều này nói với Grbl rằng "hãy bỏ qua lực ly tâm ảo, cứ chạy hết tốc độ qua các điểm nối".[12]
- **Cơ chế:** Trong mã nguồn planner.c [13], nếu giá trị này đủ lớn, max_junction_speed sẽ tiến tới vô cùng, cho phép robot duy trì vận tốc danh định do Ruckig quy định.

4.2 Cấu hình Giới hạn Gia tốc "Giả"

Để Ruckig toàn quyền kiểm soát động học, giới hạn gia tốc trong grblHAL ($120 đến $125) phải **lớn hơn** giới hạn gia tốc vật lý mà Ruckig sử dụng.

- Ví dụ: Robot chịu được tối đa $500^\circ/s^2$.
- Cấu hình Ruckig: Max Accel = $450^\circ/s^2$ (An toàn).
- Cấu hình grblHAL: Max Accel = $1000^\circ/s^2$ (Hư cấu).
- **Kết quả:** Khi Ruckig gửi lệnh tăng tốc $450^\circ/s^2$, grblHAL kiểm tra thấy $450 < 1000$, nên sẽ chấp nhận thực thi lệnh đó ngay lập tức mà không cắt giảm (clamping). Nếu grblHAL đặt thấp hơn Ruckig, robot sẽ bị trễ so với lệnh PC, gây sai lệch quỹ đạo.

4.3 Chế độ G64 (Continuous) vs G61 (Exact Stop)

Tuyệt đối không sử dụng G61 (Dừng chính xác) cho streaming.[14] G61 buộc robot dừng hẳn lại sau mỗi đoạn lệnh nhỏ (ví dụ mỗi 10ms), gây rung lắc dữ dội. Phải sử dụng **G64** (mặc định) kết hợp với Junction Deviation cao để các đoạn lệnh 10ms được nối liền mạch thành một dòng chảy chuyển động liên tục.

4.4 Kỹ thuật "Stream-Driven"

Thay vì gửi lệnh điểm đích xa (G1 X100), PC sẽ chia nhỏ quỹ đạo thành các đoạn ngắn tương ứng với chu kỳ thời gian cố định (ví dụ: $\Delta t = 10ms$).

Công thức tính Feedrate (F) cho mỗi dòng lệnh G-code:

$$F_i = \frac{\text{Khoảng cách di chuyển trong } \Delta t}{\Delta t} \times 60$$

PC gửi:

G-Code

```
G1 X[pos_t1] Y[pos_t1]... F[vel_t1]
G1 X[pos_t2] Y[pos_t2]... F[vel_t2]
```

GrblHAL sẽ thực thi các đoạn này. Nhờ cấu hình "Planner Bypass", nó sẽ tuân thủ chính xác vận tốc F mà Ruckig đã tính toán.

---

5. Tối ưu hóa Giao thức Streaming Mật độ Cao

Robot hàn yêu cầu độ mượt mà cao. Giao tiếp USB Serial là nút thắt cổ chai tiềm năng do độ trễ (latency) của hệ điều hành Windows/Linux.

5.1 Giao thức "Character Counting" (Đếm ký tự)

Giao thức "Gửi - Chờ phản hồi (ok)" (Send-Response) là quá chậm vì phải chờ trễ khứ hồi (Round-trip time) sau mỗi dòng lệnh.

Bạn cần triển khai giao thức **Character Counting** (Token Bucket) trong phần mềm C#:

1. **Nguyên lý:** PC theo dõi dung lượng bộ đệm RX của Teensy (ví dụ: 2048 bytes).
2. **Thực hiện:**
    - PC gửi lệnh liên tục miễn là Bytes_Sent - Bytes_Acknowledged < Buffer_Size.
    - Khi Teensy xử lý xong một lệnh và gửi ok, PC trừ số byte của lệnh đó ra và gửi tiếp lệnh mới.
3. **Hiệu quả:** Luôn giữ cho đường truyền USB bão hòa dữ liệu, đảm bảo Teensy không bao giờ phải chờ lệnh (Starvation).[5]

5.2 Tăng kích thước Bộ đệm (Buffer Tuning)

Để chịu được các cú "nấc" (jitter) của Windows (ví dụ: Garbage Collection của.NET), bộ đệm trên Teensy phải đủ lớn để chứa ít nhất 1-2 giây chuyển động. Trong my_machine.h hoặc config.h [4]:

- **RX_BUFFER_SIZE**: Tăng lên **4096** bytes (Mặc định thường là 128 hoặc 256). Teensy 4.1 có thừa RAM cho việc này.
- **BLOCK_BUFFER_SIZE**: Tăng lên **512** blocks (Mặc định 16-32). Điều này cho phép bộ lập kế hoạch nhìn trước hàng trăm bước di chuyển nhỏ.

5.3 Định dạng G-code Tinh gọn

Để giảm tải băng thông:

- Cắt bỏ số thập phân thừa: Robot hàn không cần độ chính xác nanomet. 3-4 số sau dấu phẩy là đủ (0.001).
- Loại bỏ dấu cách (Space) và chú thích.
- Sử dụng chế độ Modal: Không cần lặp lại G1 ở mỗi dòng.
    - *Tệ:* G1 X1.000 Y1.000 F1000 (22 bytes)
    - *Tốt:* X1.001Y1.001 (12 bytes)

---

6. Xử lý Lệnh Real-time và Vấn đề Độ trễ (Latency)

6.1 Giám sát Trạng thái Thời gian thực

PC cần biết robot đang ở đâu để đồng bộ hóa đồ họa hoặc xử lý va chạm. grblHAL hỗ trợ báo cáo thời gian thực qua ký tự ?.

- **Cấu hình $10**: Thiết lập mặt nạ bit $10 để báo cáo các thông tin cần thiết (Vị trí máy MPos, Tình trạng bộ đệm Bf).[16]
- **Bf (Buffer State):** PC phải theo dõi trường này. Nếu bộ đệm planner sắp cạn, PC có thể cảnh báo hoặc tối ưu hóa luồng gửi.

6.2 Plugin Real-time Step Generation (Nâng cao)

Nếu phương pháp streaming G-code vẫn gặp giới hạn về băng thông hoặc độ trễ, grblHAL hỗ trợ kiến trúc **Plugin** cho phép can thiệp sâu hơn.

- **Ý tưởng:** Viết một Plugin trên Teensy nhận dữ liệu nhị phân (Binary) thay vì G-code ASCII.
- **Lợi ích:** Giảm tải CPU cho việc phân tích chuỗi (parsing), giảm kích thước gói tin (24 bytes cho 6 số float so với 60+ bytes ASCII).
- **Thực hiện:** Plugin có thể móc vào hàm hal.stream.read để chặn luồng dữ liệu và nạp trực tiếp vào bộ đệm planner plan_buffer_line.[18] Đây là giải pháp triệt để cho vấn đề hiệu năng nhưng đòi hỏi kỹ năng lập trình C nhúng cao.

6.3 Xử lý Lệnh Hàn (M-Code)

Trong hàn hồ quang, độ trễ khi bật/tắt mỏ hàn là quan trọng.

- Sử dụng lệnh M3 (Spindle On) để kích hoạt hồ quang và M5 để tắt.
- grblHAL xử lý M3/M5 gần như tức thời trong luồng chuyển động. Tuy nhiên, nguồn hàn thực tế có độ trễ khí (pre-flow).
- **Ruckig Integration:** Ruckig trên PC cần tính toán thời gian dừng (Dwell - G4 P...) hoặc gửi lệnh M3 sớm hơn một khoảng thời gian dự tính để bù trừ cho độ trễ vật lý của nguồn hàn.

---

7. Kiến trúc Phần mềm Máy tính (C# Application)

7.1 Luồng xử lý Đa luồng (Multithreading)

Ứng dụng C# phải được thiết kế đa luồng để đảm bảo tính Real-time mềm (Soft Real-time):

1. **UI Thread:** Hiển thị giao diện, không được chặn các luồng khác.
2. **Trajectory Thread (Ruckig):** Chạy chu kỳ 10ms (100Hz). Tính toán vị trí tiếp theo $(q_{t+1}, \dot{q}_{t+1})$. Chuyển đổi sang chuỗi G-code và đẩy vào hàng đợi an toàn (ConcurrentQueue).
3. **Communication Thread:** Vòng lặp while(true) ưu tiên cao (High Priority). Lấy lệnh từ hàng đợi, kiểm tra bộ đệm (Token Bucket), và ghi xuống SerialPort.

7.2 Tích hợp Ruckig vào.NET

Ruckig là thư viện C++. Để dùng trong C#, bạn cần:

- Dùng **C++/CLI** để bọc (wrap) class Ruckig.
- Hoặc sử dụng **P/Invoke** để gọi các hàm xuất từ DLL của Ruckig.
- Đảm bảo dữ liệu truyền qua lại (mảng vị trí khớp) được marshaling nhanh chóng để không gây trễ.[1]

---

8. Kết luận và Khuyến nghị

Việc xây dựng bộ điều khiển robot hàn 6-DOF dựa trên PC và Teensy 4.1 grblHAL là hoàn toàn khả thi và mang lại hiệu năng cao với chi phí thấp. Chìa khóa thành công nằm ở sự phối hợp chặt chẽ giữa cấu hình firmware và thuật toán điều khiển trên PC.

Các bước hành động cụ thể:

1. **Phần cứng:** Chế tạo bo mạch kết nối (Shield) hỗ trợ đủ 6 trục Step/Dir, tránh xung đột chân Ethernet/USB.
2. **Firmware:** Biên dịch lại grblHAL với N_AXIS 6, bản đồ chân tùy chỉnh, và bộ đệm (RX/BLOCK) cực lớn.
3. **Bypass:** Đặt $11 (Junction Deviation) cao và $12x (Gia tốc) cao hơn mức Ruckig sử dụng.
4. **Phần mềm:** Viết trình điều khiển C# sử dụng giao thức Character Counting và tích hợp Ruckig qua lớp Wrapper.

Giải pháp này biến Teensy 4.1 thành một "card chuyển động" (motion card) thông minh, tận dụng sức mạnh tính toán của PC cho các thuật toán phức tạp mà vẫn đảm bảo tính ổn định thời gian thực của các xung điều khiển động cơ.

9. Tài liệu tham khảo và Trích dẫn

Các thông tin trong báo cáo được tổng hợp và phân tích dựa trên các tài liệu nghiên cứu sau:

- Cấu hình 6 trục và Pin mapping:.[7]
- Vấn đề Ethernet và nhiễu USB:.[8]
- Cấu hình Homing cho nhiều trục:.[10]
- Tối ưu hóa Buffer và Streaming:.[4]
- Junction Deviation và Planner:.[13]
- Ruckig và tích hợp quỹ đạo:.[1]
- Kiến trúc Plugin grblHAL:.[18]

Works cited

1. Tutorial - Ruckig, accessed February 1, 2026, https://docs.ruckig.com/tutorial.html
2. S-Curve - Jerk for Acceleration settings. · grblHAL core · Discussion #384 - GitHub, accessed February 1, 2026, https://github.com/grblHAL/core/discussions/384
3. grblHAL on a Teensy 4.1, accessed February 1, 2026, https://forum.pjrc.com/index.php?threads/grblhal-on-a-teensy-4-1.65331/
4. Is a smaller BLOCK_BUFFER_SIZE better? · Issue #1066 · grbl/grbl - GitHub, accessed February 1, 2026, https://github.com/grbl/grbl/issues/1066
5. Laser stuttering at high speed, ability to increase grbl buffer size - LightBurn Software Forum, accessed February 1, 2026, https://forum.lightburnsoftware.com/t/laser-stuttering-at-high-speed-ability-to-i

ncrease-grbl-buffer-size/87007
6. Slow raster performance · grblHAL core · Discussion #189 - GitHub, accessed February 1, 2026, https://github.com/grblHAL/core/discussions/189
7. iMXRT1062/grblHAL_Teensy4/src/my_machine.h at master - GitHub, accessed February 1, 2026, https://github.com/grblHAL/iMXRT1062/blob/master/grblHAL_Teensy4/src/my_machine.h
8. Using Ethernet with grblHAL on a Teensy 4.1 - The Grbl Project, accessed February 1, 2026, https://www.grbl.org/single-post/using-ethernet-with-grblhal-on-a-teensy-4-1
9. Teensy 4.1 Based CNC Controller, accessed February 1, 2026, https://forum.pjrc.com/index.php?threads/teensy-4-1-based-cnc-controller.61622/
10. Homing with 2 Axis Machine · Issue #224 · grbl/grbl - GitHub, accessed February 1, 2026, https://github.com/grbl/grbl/issues/224
11. Possible typo in driver.c · Issue #11 · grblHAL/STM32F4xx - GitHub, accessed February 1, 2026, https://github.com/grblHAL/STM32F4xx/issues/11
12. GRBL Settings - Pocket Guide - Lupa-CNC, accessed February 1, 2026, https://lupa-cnc.com/wp-content/download/GRBL_Settings_Pocket_Guide.pdf
13. core/planner.c at master · grblHAL/core - GitHub, accessed February 1, 2026, https://github.com/grblHAL/core/blob/master/planner.c
14. G61 & G64 - Path Control Mode Commands, accessed February 1, 2026, https://www.galil.com/gcode/docs/supported-gcodes/g61-g64/
15. PathPilot Quick Tips: G61 Exact Stop and G64 Constant Velocity Differences - YouTube, accessed February 1, 2026, https://www.youtube.com/watch?v=YVIk6Cg3xlM
16. GRBL Settings - Pocket Guide - DIY Machining, accessed February 1, 2026, https://www.diymachining.com/downloads/GRBL_Settings_Pocket_Guide_Rev_B.pdf
17. GRBL Settings Quick Guide - Adam's Bits, accessed February 1, 2026, https://www.endmill.com.au/blog/grbl-settings-quick-guide/
18. grblHAL/plugins: Plugins overview - GitHub, accessed February 1, 2026, https://github.com/grblHAL/plugins
19. Usage with external microcontrollers for closed loop control. · Issue #147 · terjeio/grblHAL, accessed February 1, 2026, https://github.com/terjeio/grblHAL/issues/147
20. 2024-01-17, accessed February 1, 2026, https://incoherency.co.uk/notes/20240117.html
21. Background Information - Ruckig, accessed February 1, 2026, https://docs.ruckig.com/background.html
22. CBeam and a Workbee using grblHAL - OpenBuilds, accessed February 1, 2026, https://builds.openbuilds.com/threads/cbeam-and-a-workbee-using-grblhal.16540/