Kiến trúc Hệ thống Simulation Mode Tích hợp và Virtual Digital Twin cho Điều khiển Robot Công nghiệp trên nền tảng C# WPF và Helix Toolkit

Tóm tắt Điều hành

Trong kỷ nguyên Công nghiệp 4.0, yêu cầu về việc xác minh chương trình điều khiển trước khi vận hành thực tế (First Time Right) đã trở thành một tiêu chuẩn bắt buộc nhằm giảm thiểu rủi ro va chạm, tối ưu hóa thời gian chu kỳ và đảm bảo chất lượng đường hàn. Báo cáo này trình bày một kiến trúc phần mềm chuyên sâu để xây dựng tính năng "Simulation Mode" (Chế độ Mô phỏng) tích hợp trực tiếp bên trong phần mềm điều khiển C# WPF. Thay vì dựa vào các giải pháp phần mềm bên thứ ba đắt đỏ và rời rạc, giải pháp này đề xuất việc xây dựng một "Virtual Controller" (Bộ điều khiển ảo) hoạt động song song với bộ điều khiển vật lý, sử dụng Helix Toolkit để hiển thị 3D thời gian thực.

Kiến trúc được đề xuất xoay quanh việc tách biệt hoàn toàn lớp logic điều khiển khỏi lớp phần cứng vật lý thông qua các mẫu thiết kế Mock Object nâng cao, kết hợp với các thuật toán động học nghịch (Inverse Kinematics - IK) và động học thuận (Forward Kinematics - FK) để điều khiển các thực thể "Ghost Robot" (Robot bóng ma) và Robot thực. Đặc biệt, báo cáo đi sâu vào các kỹ thuật xử lý đồ họa nâng cao để giải quyết các thách thức về hiệu năng khi hiển thị đường dẫn (Trail Rendering) mật độ cao và logic toán học vector để mô phỏng các mẫu dao động hàn (Weaving patterns) phức tạp.

1. Kiến trúc Tổng thể: VirtualController và Mẫu thiết kế Mock Object

Cốt lõi của việc tạo ra một chế độ mô phỏng phản hồi "y hệt robot thật" nằm ở việc thiết kế kiến trúc phần mềm sao cho lớp giao diện người dùng (UI) không phân biệt được nó đang giao tiếp với phần cứng hay phần mềm mô phỏng. Điều này đòi hỏi sự tuân thủ nghiêm ngặt nguyên lý Đảo ngược Phụ thuộc (Dependency Inversion Principle) và việc xây dựng một VirtualController hoạt động như một Digital Twin (Bản sao số) của bộ điều khiển vật lý.

1.1. Lớp Trừu tượng Phần cứng (Hardware Abstraction Layer - HAL)

Để phần mềm có thể chuyển đổi mượt mà giữa chế độ "Real" và "Simulation", cần thiết lập một giao diện chung (Interface) định nghĩa mọi hành vi của robot. Giao diện này, tạm gọi là IRobotController, đóng vai trò là hợp đồng ràng buộc cho cả driver điều khiển thật (ví dụ:

GrblDriver) và trình điều khiển ảo.

Các phương thức và thuộc tính trong IRobotController phải bao quát toàn bộ vòng đời hoạt động của robot:

- **Điều khiển chuyển động:** Các lệnh như Jog, MoveTo, Home, ArcTo.
- **Quản lý trạng thái:** Các thuộc tính phản ánh trạng thái máy (Idle, Run, Alarm), tọa độ hiện tại (MachinePosition, WorkPosition), và trạng thái I/O.[1]
- **Phản hồi thời gian thực:** Các sự kiện (Events) hoặc luồng dữ liệu (Observables) cung cấp vị trí khớp (Joint Angles) với tần số cao (ví dụ: 10Hz - 50Hz) để phục vụ việc vẽ đồ họa.

Trong kiến trúc này, khi người dùng kích hoạt "Simulation Mode", hệ thống Dependency Injection (DI) sẽ thay thế thể hiện của RealRobotDriver bằng VirtualController. Vì cả hai đều thực thi IRobotController, lớp ViewModel và View (WPF) sẽ tiếp tục hoạt động mà không cần thay đổi bất kỳ dòng mã nào, đảm bảo tính nhất quán tuyệt đối trong trải nghiệm vận hành.[2]

1.2. VirtualController: Mock Object hay State Simulator?

Một sai lầm phổ biến khi xây dựng chế độ mô phỏng là sử dụng các Mock Object tĩnh (chỉ trả về giá trị cố định). Để "phản hồi y hệt robot thật", VirtualController phải là một **Stateful Simulator** (Bộ mô phỏng có trạng thái). Nó không chỉ giả lập dữ liệu trả về mà còn phải giả lập *thời gian thực thi* và *vật lý chuyển động*.

VirtualController cần bao gồm các thành phần con sau:

1. **G-Code Interpreter (Trình thông dịch G-Code):** Khả năng phân tích cú pháp các lệnh G-code (G0, G1, G2, G3) và chuyển đổi chúng thành các quỹ đạo chuyển động. Việc sử dụng các thư viện phân tích cú pháp mạnh mẽ giúp đảm bảo logic nội suy của robot ảo khớp với robot thật.[3]
2. **Trajectory Planner (Bộ quy hoạch quỹ đạo):** Tính toán biên dạng vận tốc (Velocity Profile), bao gồm gia tốc và giảm tốc. Nếu robot thật mất 5 giây để di chuyển từ A đến B, robot ảo cũng phải mất đúng 5 giây để di chuyển trên màn hình. Điều này đòi hỏi VirtualController phải chạy trên một luồng (Thread) riêng biệt với độ chính xác cao, tách biệt hoàn toàn với luồng UI của WPF.[5]
3. **Collision Detector (Bộ phát hiện va chạm):** Kiểm tra giới hạn mềm (Soft Limits) và va chạm với vật thể trong môi trường ảo trước khi thực thi lệnh.

1.3. Đồng bộ hóa Luồng Mô phỏng và Luồng Hiển thị

Một thách thức lớn trong C# WPF là cơ chế render của Helix Toolkit phụ thuộc vào CompositionTarget.Rendering (thường là 60Hz), trong khi bộ điều khiển CNC thực tế hoạt động ở tần số cao hơn nhiều (1kHz hoặc cao hơn). Nếu tích hợp logic điều khiển vào luồng render, tốc độ mô phỏng sẽ bị phụ thuộc vào tốc độ khung hình (FPS), dẫn đến sai lệch thời

gian.[7]

Kiến trúc đề xuất:

- **Simulation Thread:** Sử dụng System.Threading.Timer hoặc một vòng lặp Stopwatch độ phân giải cao để cập nhật trạng thái robot, tính toán vị trí tiếp theo dựa trên vận tốc F (Feedrate) và thời gian trôi qua dt. Luồng này tính toán vị trí lý thuyết $P(t)$.
- **Render Thread:** Tại mỗi khung hình hiển thị, Viewport sẽ truy xuất vị trí $P(t)$ mới nhất từ Simulation Thread để cập nhật biến đổi (Transform) của mô hình 3D. Việc tách biệt này đảm bảo rằng ngay cả khi FPS giảm xuống thấp do render đường hàn phức tạp, logic điều khiển vẫn đảm bảo độ chính xác về thời gian và quãng đường.[6]

2. Thiết kế Máy Trạng thái Hữu hạn (Finite State Machine - FSM) cho Robot

Để mô phỏng chính xác hành vi của bộ điều khiển công nghiệp (như Grbl), VirtualController không thể chỉ là một tập hợp các biến rời rạc. Nó phải được xây dựng dựa trên một Máy Trạng thái Hữu hạn (FSM) chặt chẽ, quản lý các quy tắc chuyển đổi giữa các trạng thái như Idle, Run, Hold, Door, và Alarm.

2.1. Định nghĩa Các Trạng thái và Chuyển đổi

Dựa trên kiến trúc của Grbl, FSM trong C# cần định nghĩa rõ ràng các trạng thái thông qua enum và sử dụng một bảng chuyển đổi (State Transition Table) để kiểm soát logic.[10]

Bảng 1: Ma trận Chuyển đổi Trạng thái Mô phỏng (Grbl-Standard)

Trạng thái Hiện tại	Sự kiện Kích hoạt (Trigger)	Trạng thái Tiếp theo	Hành động Kèm theo (Action)
Idle (Chờ)	Tải file G-Code	**Idle**	Kiểm tra cú pháp, tính toán hộp bao (Bounding Box).
Idle	Lệnh Start Cycle	**Run**	Bắt đầu bộ đếm thời gian nội suy, kích hoạt Spindle ảo.
Run (Chạy)	Lệnh Feed Hold	**Hold**	Giảm tốc độ về 0 theo đường cong

			S-curve, lưu vị trí dừng.
Hold (Tạm dừng)	Lệnh Cycle Start	**Run**	Tăng tốc trở lại từ vận tốc 0 đến Feedrate cài đặt.
Run	Kích hoạt Cảm biến (Soft Limit)	**Alarm**	Dừng khẩn cấp, khóa tất cả chuyển động, báo lỗi UI.
Alarm (Báo động)	Lệnh Reset ($X)	**Idle**	Xóa bộ đệm lệnh, đặt lại tọa độ làm việc nếu cần.
Jog (Chạy tay)	Thả phím Jog	**Idle**	Dừng động cơ trơn tru (Deceleration).

Việc cài đặt FSM này có thể sử dụng thư viện Stateless trong.NET để giảm thiểu sự phức tạp của các câu lệnh if/else lồng nhau, giúp mã nguồn dễ bảo trì và mở rộng.[12]

2.2. Logic Xử lý trong Từng Trạng thái

- **Trạng thái Run (Mô phỏng Nội suy):** Đây là trạng thái phức tạp nhất. VirtualController phải thực hiện nội suy tuyến tính (Linear Interpolation) cho G1 và nội suy cung tròn (Circular Interpolation) cho G2/G3. Thuật toán phải tính toán vị trí tức thời P_i tại mỗi bước thời gian Δt:

$$P_i = P_{start} + (P_{end} - P_{start}) \times \frac{t_{elapsed}}{T_{total}}$$

  Trong đó T_{total} là tổng thời gian dự kiến để hoàn thành đoạn lệnh dựa trên Feedrate.[13]
- **Trạng thái Hold (Mô phỏng Quán tính):** Khi chuyển từ Run sang Hold, robot thật không dừng lại ngay lập tức do quán tính vật lý. VirtualController phải mô phỏng hành vi này bằng cách áp dụng một gia tốc âm (deceleration) thay vì dừng đột ngột, đảm bảo trải nghiệm người dùng (UX) giống hệt thực tế.[10]

3. Kiến trúc Ghost Robot và Kỹ thuật Hiển thị Trong

suốt

"Ghost Robot" (Robot bóng ma) là một khái niệm quan trọng trong mô phỏng, cho phép người vận hành nhìn thấy trước vị trí tương lai của robot hoặc vị trí đích (Target Position) trong khi robot thực (hoặc mô phỏng thực) đang di chuyển tới đó. Điều này đòi hỏi hiển thị đồng thời hai mô hình robot: một mô hình đặc (Solid) và một mô hình trong suốt (Transparent/Ghost).

3.1. Thách thức về Thứ tự Render (Alpha Sorting) trong WPF

Một vấn đề kỹ thuật lớn khi sử dụng Helix Toolkit WPF là lỗi hiển thị vật thể trong suốt (Transparency Sorting Issue). Do WPF sử dụng thuật toán Z-buffer đơn giản, nếu các bề mặt trong suốt không được vẽ theo thứ tự từ xa đến gần so với camera, các vật thể phía sau sẽ bị che khuất một cách sai lệch hoặc biến mất hoàn toàn.[15]

Nếu không xử lý đúng, Ghost Robot sẽ trông như bị "gãy" hoặc hiển thị sai thứ tự các khớp, làm mất đi tính chân thực cần thiết cho việc kiểm tra va chạm bằng mắt thường.[17]

3.2. Giải pháp SortingVisual3D

Để khắc phục, kiến trúc đề xuất sử dụng lớp SortingVisual3D của Helix Toolkit. Lớp này thực hiện việc sắp xếp lại các tam giác hoặc các đối tượng con dựa trên khoảng cách tới camera trong mỗi khung hình render.

Quy trình triển khai Ghost Robot:

1. **Container:** Đặt toàn bộ các MeshGeometry3D của Ghost Robot vào trong một SortingVisual3D thay vì ModelVisual3D thông thường.
2. **Vật liệu (Material):** Sử dụng DiffuseMaterial với SolidColorBrush có kênh Alpha thấp (ví dụ: Opacity = 0.3 hoặc 0.4). Màu sắc nên được chọn để tương phản với robot thật (ví dụ: Robot thật màu cam, Ghost Robot màu xanh neon hoặc xám nhạt).
   C#
   ```
   // Ví dụ định nghĩa vật liệu cho Ghost Robot
   var ghostMaterial = new DiffuseMaterial(new SolidColorBrush(Color.FromArgb(80, 200, 200, 200)));
   ```
3. **Tách biệt Scenegraph:** Scenegraph (cây hiển thị) nên được tổ chức sao cho Ghost Robot và Real Robot là hai nhánh riêng biệt nhưng chia sẻ cùng một cấu trúc hình học (Geometry) để tiết kiệm bộ nhớ, chỉ khác nhau về Transform (vị trí các khớp) và Material.[18]

3.3. Động học Nghịch (Inverse Kinematics - IK) cho Ghost Robot

Để Ghost Robot hiển thị được vị trí đích khi người dùng Jog (di chuyển) theo tọa độ Descartes (X, Y, Z, Rx, Ry, Rz), hệ thống cần một bộ giải Động học Nghịch (IK) thời gian thực.

- **Analytical IK (IK Giải tích):** Đối với robot 6 trục chuẩn công nghiệp (như Fanuc, ABB,

hoặc robot tự chế theo chuẩn DH), nên sử dụng phương pháp giải tích thay vì phương pháp số (Numerical methods như Jacobian). Giải pháp giải tích cho kết quả tức thì và chính xác tuyệt đối về mặt toán học, giúp Ghost Robot di chuyển mượt mà không có độ trễ.[20]

- **Xử lý Đa nghiệm:** IK giải tích thường trả về tối đa 8 nghiệm (cấu hình lắp ráp robot: Elbow Up/Down, Wrist Flip/No-Flip). VirtualController cần logic để chọn nghiệm gần nhất với cấu hình hiện tại của robot thật để tránh việc Ghost Robot "nhảy" bất ngờ giữa các cấu hình.[20]

4. Trail Rendering: Hiển thị Đường dẫn Hiệu năng cao

Trong mô phỏng hàn, việc hiển thị đường dẫn (Trail) là cực kỳ quan trọng để kiểm tra biên dạng mối hàn. Tuy nhiên, việc vẽ hàng nghìn điểm trong WPF có thể gây nghẽn cổ chai (bottleneck) về hiệu năng CPU.[22]

4.1. Vấn đề của LinesVisual3D

Sử dụng LinesVisual3D mặc định của Helix Toolkit cho đường dẫn động (dynamic trail) thường không hiệu quả vì mỗi khi thêm một điểm mới, WPF phải xây dựng lại toàn bộ visual tree của đối tượng đó. Với đường hàn dài chứa hàng chục nghìn điểm, FPS của ứng dụng sẽ giảm xuống mức không thể chấp nhận được (dưới 10 FPS).[23]

4.2. Giải pháp Tối ưu: SharpDX hoặc MeshBuilder Động

Để đạt hiệu năng cao nhất (High Performance), kiến trúc đề xuất chuyển sang sử dụng **HelixToolkit.Wpf.SharpDX**. Thư viện này render trực tiếp qua DirectX 11, bỏ qua lớp trung gian chậm chạp của WPF 3D Engine.

- **LineGeometry3D:** Sử dụng LineGeometry3D trong SharpDX cho phép vẽ hàng triệu đoạn thẳng mà không ảnh hưởng đáng kể đến hiệu năng. Dữ liệu đỉnh (Vertex Buffer) có thể được cập nhật trực tiếp trên GPU.
- **Kỹ thuật Ring Buffer (Bộ đệm vòng):** Thay vì liên tục cấp phát bộ nhớ mới cho danh sách điểm (List), hãy khởi tạo một mảng cố định (ví dụ: 50.000 điểm) và sử dụng chỉ số đầu/cuối để cập nhật đường dẫn. Kỹ thuật này loại bỏ chi phí Garbage Collection (GC) trong C#, giữ cho simulation loop luôn mượt mà.[25]
- **Mã hóa thông tin vào màu sắc:** Đường dẫn không chỉ là đường kẻ đơn sắc. Sử dụng tính năng Vertex Color của SharpDX để tô màu đường dẫn theo vận tốc hoặc trạng thái hàn (ví dụ: Màu đỏ cho đoạn hàn G1, màu xanh cho đoạn chạy không tải G0, màu vàng cho đoạn tiếp cận). Điều này cung cấp thông tin trực quan tức thì cho người vận hành về quy trình công nghệ.[27]

5. Logic Mô phỏng Hàn: Thuật toán Weaving và Tạo

hình Mối hàn

Yêu cầu cao nhất của hệ thống là mô phỏng quá trình hàn, đặc biệt là các mẫu dao động (Weaving) như Zigzag, Sin, hoặc Tam giác. Đây không chỉ là hiển thị mà là tính toán toán học phức tạp để tạo ra các tọa độ bù trừ (Offset) thời gian thực.

5.1. Toán học Vector cho Dao động Hàn

Dao động hàn (Weaving) luôn diễn ra trên mặt phẳng vuông góc với hướng di chuyển của mỏ hàn. Để tính toán vị trí dao động chính xác trong không gian 3D, cần xác định Hệ quy chiếu Frenet (Frenet Frame) tại mỗi điểm trên quỹ đạo.[29]

Gọi vector hướng di chuyển là T (Tangent) và vector định hướng của mỏ hàn là O (Tool Orientation).

1. **Vector Binormal (B):** Vector phương ngang (trái/phải) của dao động được tính bằng tích có hướng (Cross Product):

   $$B = \text{Normalize}(T \times O)$$

   Vector này đảm bảo dao động luôn vuông góc với đường hàn và trục mỏ hàn, bất kể robot đang hàn trên mặt phẳng ngang hay leo tường.[30]
2. **Hàm Dao động (Oscillation Function):** Tọa độ lệch δ tại thời gian t được tính dựa trên biên độ A và tần số f.
   - *Dao động Sin:* $\delta(t) = A \cdot \sin(2\pi f t)$
   - *Dao động Tam giác:* Sử dụng hàm tuyến tính đan xen để tạo biên dạng sắc nét.[31]
3. **Vị trí Thực tế:** Vị trí cuối cùng của Ghost Robot (hoặc đầu dây hàn ảo) là:

   $$P_{weaving} = P_{path} + (B \cdot \delta(t))$$

5.2. Hiển thị Mối hàn 3D (Volumetric Rendering)

Thay vì chỉ vẽ đường line, để mô phỏng "y hệt robot thật", hệ thống nên hiển thị mối hàn dưới dạng khối 3D. Sử dụng MeshBuilder.AddTube của Helix Toolkit là phương pháp tối ưu.

- **Dynamic Tube Generation:** Tại mỗi bước mô phỏng, thu thập các điểm $P_{weaving}$ và sử dụng MeshBuilder để tạo ra một đoạn ống (Tube) nối tiếp.
- **Mô phỏng đắp nổi:** Đường kính của ống (Tube Diameter) có thể được lập trình để thay đổi theo tốc độ di chuyển: Di chuyển chậm -> Mối hàn to; Di chuyển nhanh -> Mối hàn nhỏ. Điều này giúp người vận hành phát hiện các điểm lỗi tiềm ẩn (như cháy thủng hoặc

không ngấu) ngay trên mô hình 3D.[32]

Kết luận

Kiến trúc được đề xuất cung cấp một giải pháp toàn diện để xây dựng tính năng Simulation Mode trong C# WPF với độ trung thực cao. Bằng cách kết hợp **VirtualController** dựa trên máy trạng thái FSM (mô phỏng não bộ), **Analytical IK** (mô phỏng cơ học), và **SharpDX/Helix Toolkit** (mô phỏng thị giác), hệ thống có thể tái tạo chính xác hành vi của robot hàn công nghiệp. Việc áp dụng các kỹ thuật như SortingVisual3D cho Ghost Robot và toán học vector cho thuật toán Weaving đảm bảo rằng môi trường ảo không chỉ đẹp mắt mà còn chính xác về mặt kỹ thuật, đáp ứng yêu cầu khắt khe của quy trình sản xuất "First Time Right".

Chi tiết Triển khai Kỹ thuật

1. Phân tích Yêu cầu và Lựa chọn Công nghệ

Dựa trên yêu cầu xây dựng tính năng Simulation Mode trong môi trường C# WPF sử dụng Helix Toolkit, bảng dưới đây tóm tắt các thành phần công nghệ được lựa chọn để đáp ứng các tiêu chí về hiệu năng và độ chính xác.

Bảng 2: Ma trận Lựa chọn Công nghệ cho Simulation Mode

Thành phần	Công nghệ / Thư viện	Lý do Lựa chọn
3D Engine	HelixToolkit.Wpf.SharpDX	Hiệu năng render cao hơn WPF thuần, hỗ trợ DirectX 11, xử lý tốt hàng triệu đỉnh (vertices) cho Trail Rendering.[24]
Logic Điều khiển	Custom VirtualController (C#)	Cần kiểm soát hoàn toàn thời gian thực (deterministic timing) mà các thư viện có sẵn không đáp ứng được.
State Machine	Thư viện Stateless (.NET)	Quản lý trạng thái phân cấp (Hierarchical States),

		dễ dàng định nghĩa trigger và guard conditions.[12]
G-Code Parser	Custom Regex / GcodeParserSharp	Cần tốc độ xử lý nhanh và khả năng tùy biến để hỗ trợ các mã G-code mở rộng cho hàn (G-Weave).[3]
Kinematics	Analytical Solution (Tự triển khai)	Tốc độ tính toán O(1), không cần lặp (iteration) như phương pháp Jacobian, đảm bảo FPS cao cho Ghost Robot.[20]

2. Thiết kế VirtualController: Trái tim của Hệ thống

VirtualController chịu trách nhiệm duy trì trạng thái của toàn bộ hệ thống mô phỏng. Nó hoạt động như một server nội bộ, cung cấp dữ liệu cho giao diện người dùng (GUI).

2.1. Vòng lặp Mô phỏng (Simulation Loop)

Khác với các ứng dụng quản lý thông thường dựa trên sự kiện (Event-driven), phần mềm điều khiển robot đòi hỏi một vòng lặp điều khiển thời gian thực. Tuy nhiên, trong C#, System.Windows.Threading.DispatcherTimer có độ phân giải thấp (khoảng 15-30ms) và không ổn định.

Kiến trúc đề xuất sử dụng **Multimedia Timer** hoặc một luồng Thread riêng biệt với SpinWait để đạt được chu kỳ cập nhật (Tick) ổn định, ví dụ 10ms (100Hz).

C#

```
// Giả mã (Pseudo-code) cho Vòng lặp VirtualController
private void SimulationLoop()
{
    while (_isSimulationRunning)
    {
        long startTime = Stopwatch.GetTimestamp();

        // 1. Cập nhật Máy trạng thái
        _stateMachine.Update();
```

```
        // 2. Nội suy chuyển động (Interpolation)
        if (_stateMachine.State == MachineState.Run)
        {
            var nextPosition = _interpolator.CalculateNextPosition(_deltaTime);
            CurrentPosition = nextPosition;

            // 3. Tính toán Weaving (nếu đang hàn)
            if (_isWelding)
            {
                var weaveOffset = _weavingProcessor.CalculateOffset(nextPosition, _deltaTime);
                CurrentPosition += weaveOffset;
            }
        }

        // 4. Kiểm tra va chạm (Collision Check)
        if (_collisionDetector.Check(CurrentPosition))
        {
            _stateMachine.Fire(Trigger.HardLimit);
        }

        // Duy trì tần số vòng lặp ổn định
        WaitForNextTick(startTime);
    }
}
```

Việc tách biệt luồng tính toán này đảm bảo rằng ngay cả khi giao diện 3D bị giật (lag) do tải nặng, logic robot vẫn di chuyển đúng quãng đường và vận tốc, không bị "trôi" thời gian.[5]

2.2. Xử lý G-Code và Nội suy (Interpolation)

Trình biên dịch G-code trong VirtualController cần chuyển đổi các dòng lệnh văn bản thành danh sách các đoạn chuyển động (Motion Segments).

- **Linear Interpolation (G0/G1):** Sử dụng thuật toán LERP (Linear Interpolation).

  $$P(t) = P_0 + (P_1 - P_0) \cdot t$$

  Trong đó $t \in $. Để mô phỏng vận tốc thực tế, t được tính bằng:

  $$t = \frac{\text{Khoảng cách đã đi}}{\text{Tổng chiều dài đoạn}}$$

  Khoảng cách đã đi được tích lũy dựa trên vận tốc Feedrate (mm/s) nhân với Δt của

vòng lặp.[13]

- **Circular Interpolation (G2/G3):** Phức tạp hơn, yêu cầu sử dụng SLERP (Spherical Linear Interpolation) hoặc tính toán lượng giác trên mặt phẳng 2D (XY, XZ, hoặc YZ) dựa trên tâm cung tròn I, J, K. VirtualController phải xử lý chính xác việc chuyển đổi hệ tọa độ để hiển thị đúng các cung tròn trong không gian 3D.[35]

3. Kỹ thuật Mock Object và Dependency Injection

Để đảm bảo phần mềm có thể chuyển đổi linh hoạt, chúng ta sử dụng mẫu thiết kế **Abstract Factory** hoặc **Strategy Pattern** kết hợp với Dependency Injection (DI).

3.1. Định nghĩa Interface Chung

Giao diện IMotionControlService là cầu nối duy nhất giữa ViewModel (Logic giao diện) và Controller (Logic điều khiển).

C#

```
public interface IMotionControlService
{
    // Các Property quan sát được (Observable)
    IObservable<Point3D> CurrentPosition { get; }
    IObservable<Vector3> JointAngles { get; }
    IObservable<MachineState> State { get; }

    // Các phương thức điều khiển
    void Connect(string portName);
    void LoadGCode(string code);
    void CycleStart();
    void FeedHold();
    void Jog(Axis axis, double step);
}
```

3.2. Cài đặt Mock Object (Virtual Service)

Lớp VirtualMotionService thực thi giao diện trên nhưng không gửi byte xuống cổng COM. Thay vào đó, nó kích hoạt các logic nội bộ đã mô tả ở phần VirtualController.

Điểm quan trọng của Mock Object này là khả năng **Fault Injection** (Tiêm lỗi). Để mô phỏng thực tế, Mock Object nên có khả năng:

- Ngẫu nhiên kích hoạt trạng thái "Alarm" để kiểm tra phản ứng của UI.
- Mô phỏng độ trễ (latency) khi nhận lệnh, giống như giao tiếp Serial thực tế.
- Mô phỏng nhiễu tín hiệu (nếu cần kiểm tra bộ lọc).

Cách tiếp cận này giúp kiểm thử phần mềm (Unit Testing và Integration Testing) trở nên dễ dàng và tin cậy hơn.[1]

4. Ghost Robot: Giải pháp Hiển thị và Tương tác

Ghost Robot không chỉ là hình ảnh mờ ảo, nó là công cụ tương tác chính trong chế độ Simulation.

4.1. Kiến trúc Scenegraph cho Ghost Robot

Trong Helix Toolkit, Scenegraph được tổ chức theo cây. Để tối ưu hóa việc quản lý Ghost Robot:

1. **Model3DGroup:** Tạo một nhóm Model3DGroup riêng cho Ghost Robot.
2. **Shared Geometry:** Các Mesh (Lưới) của từng khâu (Link) robot nên được tải một lần duy nhất vào bộ nhớ (Resource Dictionary) và được chia sẻ giữa Robot thật và Ghost Robot. Chỉ có Transform (Vị trí/Góc quay) và Material (Vật liệu) là khác nhau. Điều này giúp giảm đáng kể lượng bộ nhớ VRAM tiêu thụ.[22]

4.2. Xử lý Transparency (Trong suốt)

Như đã đề cập trong phần Tóm tắt, Helix Toolkit WPF gặp vấn đề với Transparency Sorting. Giải pháp SortingVisual3D là bắt buộc.

Bảng 3: So sánh các phương pháp Render Transparency trong Helix Toolkit

Phương pháp	Ưu điểm	Nhược điểm	Khuyến nghị
Material Opacity	Đơn giản, dễ cài đặt.	Lỗi hiển thị (vật thể sau đè lên vật thể trước) do Z-buffer.	Không dùng cho Ghost Robot phức tạp.
Backface Culling Off	Hiển thị cả mặt trong của robot.	Không giải quyết được vấn đề thứ tự vẽ giữa các khâu.	Chỉ dùng bổ trợ.
SortingVisual3D	Sắp xếp lại thứ tự vẽ theo khoảng cách camera.	Tốn chi phí CPU để tính toán lại mỗi khung hình.	**Khuyên dùng** cho Ghost Robot.

Order Independent Transparency (OIT)	Chính xác tuyệt đối, xử lý trên GPU.	Chỉ hỗ trợ trên HelixToolkit.SharpDX, phức tạp để cài đặt.	Dùng nếu chuyển sang SharpDX hoàn toàn.

Với yêu cầu "C# WPF" thuần túy (không bắt buộc SharpDX nhưng nên dùng nếu có thể), SortingVisual3D là giải pháp an toàn và ổn định nhất để Ghost Robot hiển thị đúng lớp lang.[18]

5. Trail Rendering và Hiệu năng Đồ họa

Hiển thị đường dẫn hàn là một bài toán khó về hiệu năng. Một file hàn phức tạp có thể chứa hàng trăm nghìn điểm lệnh.

5.1. Kỹ thuật Dynamic MeshBuilder

Thay vì tạo hàng ngàn đối tượng LinesVisual3D (mỗi đối tượng gây tốn chi phí quản lý của WPF), ta nên gộp tất cả vào một MeshGeometry3D duy nhất.

Sử dụng HelixToolkit.Wpf.MeshBuilder để xây dựng một đối tượng duy nhất chứa toàn bộ đường dẫn. Tuy nhiên, việc xây dựng lại (Rebuild) lưới này mỗi khi robot di chuyển thêm 1mm là rất tốn kém.

Chiến lược tối ưu:

- **Chunking (Phân mảnh):** Chia đường dẫn thành các đoạn nhỏ (Chunks), ví dụ mỗi 1000 điểm là một Mesh riêng biệt. Khi robot di chuyển, chỉ Mesh cuối cùng (đang active) là được cập nhật. Các Mesh cũ được "đóng băng" (Freeze()) để WPF tối ưu hóa việc vẽ.[22]
- **Decimation (Giảm lược điểm):** Không cần vẽ tất cả các điểm nội suy. Chỉ thêm điểm vào Trail nếu khoảng cách so với điểm trước đó lớn hơn một ngưỡng nhất định (ví dụ: 0.5mm) hoặc góc thay đổi lớn. Điều này giảm số lượng đa giác cần vẽ mà mắt thường không nhận ra sự khác biệt.[39]

6. Logic Mô phỏng Hàn: Weaving và Physics

Để tạo ra "Robot ảo phản hồi y hệt Robot thật", phần mô phỏng hàn phải tính đến các yếu tố vật lý của hồ quang và dao động mỏ hàn.

6.1. Vector Math cho Weaving (Dao động)

Để thực hiện Weaving, ta cần xác định hệ trục tọa độ cục bộ tại điểm hàn.

Giả sử robot đang di chuyển từ điểm P_1 đến P_2.

Vector tiếp tuyến (hướng di chuyển): $T = \text{Normalize}(P_2 - P_1)$.

Vector pháp tuyến bề mặt (thường là hướng trục Z của mỏ hàn): N.

Vector Binormal (hướng dao động ngang) B được tính bằng tích có hướng:

$$B = T \times N$$

Nếu robot đang leo tường (hàn đứng), vector B này sẽ tự động nằm ngang song song với mặt đất hoặc bề mặt hàn, đảm bảo đường hàn không bị méo mó. Đây là logic toán học cốt lõi để tính toán offset cho các mẫu Zigzag hoặc Crescent (Lưỡi liềm).[29]

6.2. Mô phỏng Đắp Mối hàn (Bead Deposition)

Thay vì chỉ vẽ một đường mảnh, tính năng Simulation Mode nên thể hiện được "độ dày" của mối hàn.

Sử dụng phương thức MeshBuilder.AddTube():

- **Input:** Danh sách các điểm $P_{weaving}$ đã tính toán.
- **Diameter:** Đường kính ống. Để tăng tính chân thực, đường kính này có thể tỷ lệ nghịch với tốc độ di chuyển (F).

  $$D_{bead} \approx \frac{K}{F}$$

  Điều này giúp người dùng nhìn thấy trực quan: Chỗ nào robot chạy chậm, mối hàn sẽ phình to ra; chỗ nào chạy nhanh, mối hàn sẽ thu nhỏ lại. Đây là một tính năng cao cấp giúp kiểm soát chất lượng đường hàn ảo.[32]

Kết luận và Khuyến nghị Triển khai

Để xây dựng thành công tính năng Simulation Mode cho phần mềm điều khiển Robot hàn với các yêu cầu đã đặt ra, báo cáo khuyến nghị lộ trình triển khai như sau:

1. **Giai đoạn 1 (Core):** Xây dựng VirtualController và FSM sử dụng thư viện Stateless. Đảm bảo logic nội suy G0/G1 hoạt động chính xác trên luồng riêng.
2. **Giai đoạn 2 (Basic Vis):** Tích hợp Helix Toolkit WPF. Xây dựng lớp RobotVisualizer sử dụng FK để điều khiển mô hình robot dựa trên khớp.
3. **Giai đoạn 3 (Ghost & Interaction):** Cài đặt SortingVisual3D cho Ghost Robot. Tích hợp Analytical IK để Ghost Robot bám theo chuột hoặc tọa độ nhập liệu.
4. **Giai đoạn 4 (Advanced Process):** Cài đặt thuật toán Weaving và Trail Rendering tối ưu

(Chunking/SharpDX).

Kiến trúc này không chỉ đáp ứng yêu cầu "phản hồi y hệt robot thật" mà còn mở ra khả năng mở rộng trong tương lai cho các tính năng như phát hiện va chạm thời gian thực hay mô phỏng đa robot.

Works cited

1. CNC Simulation and Programming Solutions Explained - Radonix, accessed February 1, 2026, https://radonix.com/cnc-simulation-and-programming-solutions/
2. A Scalable Graphics User Interface Architecture for CNC Application Based - on WPF and MVVM - ResearchGate, accessed February 1, 2026, https://www.researchgate.net/publication/269380100_A_Scalable_Graphics_User_Interface_Architecture_for_CNC_Application_Based_-_on_WPF_and_MVVM
3. AndreasReitberger/GcodeParserSharp: A simple C# project to parse gcode files and get the estimated print time and extruded filament volume. - GitHub, accessed February 1, 2026, https://github.com/AndreasReitberger/GcodeParserSharp
4. Michael-F-Bryan/gcodes: A basic C# gcode parser and interpreter - GitHub, accessed February 1, 2026, https://github.com/Michael-F-Bryan/gcodes
5. WPF Architecture - Microsoft Learn, accessed February 1, 2026, https://learn.microsoft.com/en-us/dotnet/desktop/wpf/advanced/wpf-architecture
6. How to: Render on a Per Frame Interval Using CompositionTarget - WPF | Microsoft Learn, accessed February 1, 2026, https://learn.microsoft.com/en-us/dotnet/desktop/wpf/graphics-multimedia/how-to-render-on-a-per-frame-interval-using-compositiontarget
7. How to control frame rate in WPF by using dispatcher timer accurately? - Stack Overflow, accessed February 1, 2026, https://stackoverflow.com/questions/31237266/how-to-control-frame-rate-in-wpf-by-using-dispatcher-timer-accurately
8. What can I do inside CompositionTarget.Rendering? - Stack Overflow, accessed February 1, 2026, https://stackoverflow.com/questions/18353225/what-can-i-do-inside-compositiontarget-rendering
9. HelixToolkit.Wpf.SharpDX performance (High CPU usage) · Issue #585 - GitHub, accessed February 1, 2026, https://github.com/helix-toolkit/helix-toolkit/issues/585
10. Grbl control input buttons., accessed February 1, 2026, https://www.grbl.org/single-post/grbl-control-input-buttons
11. C# Enum: Definition and Best Practices - Stackify, accessed February 1, 2026, https://stackify.com/c-enum-definition-and-best-practices/
12. Implementing State Machines in C# Using Stateless: A Step-by-Step Guide, accessed February 1, 2026,

https://gunesramazan.medium.com/implementing-state-machines-in-c-using-stateless-a-step-by-step-guide-641e35133134
13. Real time trajectory generation and interpolation - UBC Library Open Collections, accessed February 1, 2026, https://open.library.ubc.ca/soa/clRcle/collections/ubctheses/24/items/1.0080691
14. Linear Interpolation - Alan Zucconi, accessed February 1, 2026, https://www.alanzucconi.com/2021/01/24/linear-interpolation/
15. Opacity bug in VS 2015 using Helix Toolkit - Stack Overflow, accessed February 1, 2026, https://stackoverflow.com/questions/52998657/opacity-bug-in-vs-2015-using-helix-toolkit
16. problem with opacity change of objects - Helix Toolkit forum, accessed February 1, 2026, https://helixtoolkit.userecho.com/communities/1/topics/426-problem-with-opacity-change-of-objects
17. HelixToolkit.Wpf.SharpDX - Transparency and Sorting, accessed February 1, 2026, https://helixtoolkit.userecho.com/communities/1/topics/158-helixtoolkitwpfsharpdx-transparency-and-sorting
18. Issues with making a 3D model faces semi-transparent using HelixToolkit - Stack Overflow, accessed February 1, 2026, https://stackoverflow.com/questions/35167471/issues-with-making-a-3d-model-faces-semi-transparent-using-helixtoolkit
19. Helix Toolkit recently added tube objects in WPF application, are not firing events or reacts on mouse interaction? #1696 - GitHub, accessed February 1, 2026, https://github.com/helix-toolkit/helix-toolkit/issues/1696
20. 6-DOF Robot Inverse Kinematics: Analytical Solutions Explained! - YouTube, accessed February 1, 2026, https://www.youtube.com/watch?v=fATIK4tivNU
21. The Ultimate Guide to Inverse Kinematics for 6DOF Robot Arms - Automatic Addison, accessed February 1, 2026, https://automaticaddison.com/the-ultimate-guide-to-inverse-kinematics-for-6dof-robot-arms/
22. Improve WPF rendering performance using Helix Toolkit - Stack Overflow, accessed February 1, 2026, https://stackoverflow.com/questions/33374434/improve-wpf-rendering-performance-using-helix-toolkit
23. wpf 3d performance : r/csharp - Reddit, accessed February 1, 2026, https://www.reddit.com/r/csharp/comments/h92rz5/wpf_3d_performance/
24. How to improve performance of rendering 3D scene using HelixViewport - Stack Overflow, accessed February 1, 2026, https://stackoverflow.com/questions/54429650/how-to-improve-performance-of-rendering-3d-scene-using-helixviewport
25. Load and unload Model3DGroup dynamically? · Issue #547 · helix-toolkit/helix-toolkit, accessed February 1, 2026, https://github.com/helix-toolkit/helix-toolkit/issues/547
26. How to update large number of visuals in HelixViewport3D - Stack Overflow,

accessed February 1, 2026, https://stackoverflow.com/questions/69441790/how-to-update-large-number-of-visuals-in-helixviewport3d
27. Set per-vertex colour · Issue #1062 · helix-toolkit/helix-toolkit - GitHub, accessed February 1, 2026, https://github.com/helix-toolkit/helix-toolkit/issues/1062
28. Drawing a lot of lines of different color · Issue #439 - GitHub, accessed February 1, 2026, https://github.com/helix-toolkit/helix-toolkit/issues/439
29. Circular trajectory weaving welding control algorithm based on space transformation principle - ResearchGate, accessed February 1, 2026, https://www.researchgate.net/publication/353844809_Circular_trajectory_weaving_welding_control_algorithm_based_on_space_transformation_principle
30. Computing a Normal/Perpendicular vector - Unity - Manual, accessed February 1, 2026, https://docs.unity3d.com/2019.3/Documentation/Manual/ComputingNormalPerpendicularVector.html
31. An Algorithm for the Welding Torch Weaving Control of Arc Welding Robot - Semantic Scholar, accessed February 1, 2026, https://pdfs.semanticscholar.org/91d2/5b5cef87617741edce7b7c954f1141293ef7.pdf
32. General / Helix Toolkit forum, accessed February 1, 2026, https://helixtoolkit.userecho.com/communities/1-general/topics?page=58
33. MeshBuilder.AddTube FrontCap and BackCap · Issue #1694 · helix-toolkit/helix-toolkit · GitHub, accessed February 1, 2026, https://github.com/helix-toolkit/helix-toolkit/issues/1694
34. Performance / General / Helix Toolkit forum, accessed February 1, 2026, https://helixtoolkit.userecho.com/communities/1/topics/110-performance
35. Robotics3D Lec11c: Trajectory generation for orientation using LERP and SLERP (Spring 2025) - YouTube, accessed February 1, 2026, https://www.youtube.com/watch?v=og-xde1lmOk
36. Accurate real-time trajectory generation of circular motion using FIR interpolation: a trochoidal milling case study - PMC - NIH, accessed February 1, 2026, https://pmc.ncbi.nlm.nih.gov/articles/PMC12006282/
37. newmatik/grbltest: Testing GRBL format in C# using .NET only - GitHub, accessed February 1, 2026, https://github.com/newmatik/grbltest
38. Helix Toolkit is a collection of 3D components for .NET. - GitHub, accessed February 1, 2026, https://github.com/helix-toolkit/helix-toolkit
39. bodxp/helix-toolkit - Gitee, accessed February 1, 2026, https://gitee.com/lemorlee/helix-toolkit/blob/master/CHANGELOG.md