Xây dựng chương trình điều khiển

A screenshot of a computer code

AI-generated content may be incorrect.

Figure 1. Cấu trúc chương trình

1. **Chương trình quét mã Aruco trên Raspberry Pi 3**

Mục tiêu

Xây dựng một hệ thống trên Raspberry Pi 3 sử dụng Raspberry Pi Camera Module V1.3 để:

* Phát hiện mã ArUco trong ảnh.
* Trích xuất tọa độ trung tâm và góc quay của mã.
* Gửi thông tin này qua cổng UART tới vi điều khiển khác ESP32.

Quá trình hoạt động

1. Thu nhận hình ảnh:  
   Camera trên Raspberry Pi liên tục chụp ảnh theo thời gian thực. Hình ảnh được xử lý ở độ phân giải phù hợp để cân bằng giữa tốc độ và độ chính xác.
2. Phát hiện mã ArUco:  
   Mã ArUco là một loại mã ma trận (giống QR code), dùng để đánh dấu và định vị trong robot. Sau đó sử dụng thuật toán xử lý ảnh của openCV để tìm các mã này trong khung hình.
3. Trích xuất thông tin:  
   Khi một mã ArUco được phát hiện, hệ thống sẽ xác định:
   * Mã ID: số hiệu duy nhất của mỗi marker.
   * Vị trí tương đối (tọa độ tâm mã trên ảnh).
   * Góc xoay của mã so với phương ngang.
4. Giao tiếp UART:  
   Sau khi có thông tin, Raspberry Pi sẽ gửi chuỗi dữ liệu "raspi\_camera,{marker\_id},{cx},{cy},{angle:.2f}\n" qua giao tiếp UART (giao tiếp nối tiếp) tới vi điều khiển ESP32 để xử lý.

Chi tiết quy trình xử lý ảnh mã ArUco trong OpenCV

Bước 1: Tiền xử lý ảnh

* Ảnh từ camera (RGB/BGR) được chuyển sang ảnh xám (grayscale).
* Sau đó áp dụng ngưỡng hóa (threshold) hoặc ngưỡng hóa thích ứng để làm nổi bật các khối đen trắng.

Bước 2: Phát hiện các hình vuông trong ảnh

* OpenCV tìm tất cả các contour trong ảnh.
* Sàng lọc các contour có hình dáng gần giống tứ giác:
  + Có bốn cạnh.
  + Góc gần 90°.
  + Cạnh tương đối bằng nhau.
* Lưu lại các vùng có khả năng là mã ArUco (gọi là candidate markers).

Bước 3: Chuẩn hóa góc nhìn (Perspective Transform)

* Mỗi candidate marker được biến đổi phối cảnh để làm thẳng mã.
* Kết quả là một hình vuông vuông vức, dù mã trong ảnh ban đầu có bị nghiêng hay biến dạng.

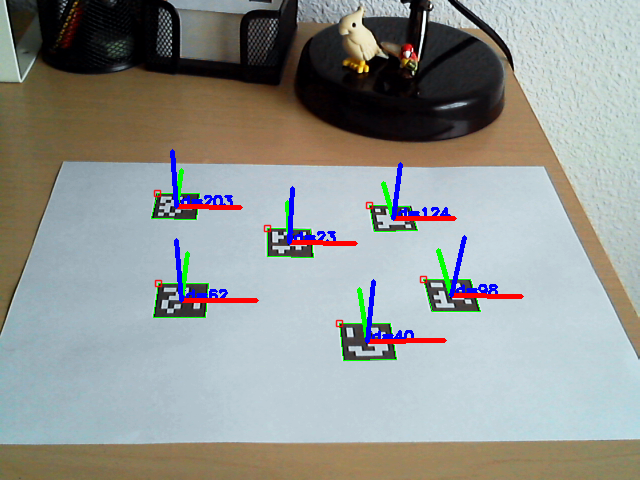


Figure 2. Chuẩn hóa góc nhìn

Bước 4: Đọc nội dung mã

* Hình vuông được chia thành một lưới ô vuông (tùy vào loại mã: 4x4, 5x5, 6x6...).
* Mỗi ô được đọc là đen (0) hoặc trắng (1) → tạo ra một ma trận nhị phân.

Bước 5: So khớp với từ điển mã

* So sánh ma trận nhị phân với danh sách mã đã biết (predefined dictionary).
* Nếu khớp, hệ thống xác định được:
  + ID của mã
  + Hướng xoay của mã (OpenCV tự động nhận diện các hướng xoay 0°, 90°, 180°, 270°).

Bước 6: Trả về thông tin mã

* Hệ thống trả về cho mỗi mã:
  + ID
  + Tọa độ 4 đỉnh trên ảnh gốc
  + Góc quay (hướng)
* Dữ liệu này có thể dùng để:
  + Tính tọa độ tâm.
  + Tính góc nghiêng.
  + Ước lượng vị trí và hướng của vật thể trong không gian.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 3. Kết quả nhận diện mã ArUco 4x4

1. **Chương trình điều khiển trên Esp32**

A diagram of a vehicle

AI-generated content may be incorrect.

Figure 4. Chương trình điều khiển trên Esp32

2.1. Khối UART

Giao tiếp UART (Universal Asynchronous Receiver/Transmitter) là phương pháp truyền dữ liệu nối tiếp bất đồng bộ giữa hai thiết bị, thường qua hai đường tín hiệu TX (transmit) và RX (receive). Mỗi lần truyền một khung dữ liệu bao gồm một bit khởi đầu (start bit) để thông báo bắt đầu, lần lượt 5–9 bit dữ liệu, tùy cấu hình, có thể kèm một bit kiểm tra chẵn lẻ (parity) để phát hiện lỗi đơn, và kết thúc bằng một hoặc hai stop bit để đảm bảo thiết bị nhận có thời gian kịp nắm được toàn khung. Vì là bất đồng bộ, UART không dùng xung nhịp chung; hai thiết bị chỉ cần thống nhất tốc độ baud rate và cấu hình khung (số bit dữ liệu, parity, stop bit) là có thể trao đổi mà không cần đường clock riêng.

Về mặt điện lý, tín hiệu UART ở mức TTL (0–3.3 V hoặc 0–5 V) khá đơn giản: mức thấp (0 V) đại diện cho bit “1” và mức cao (3.3 V hoặc 5 V) cho bit “0” (hoặc ngược lại tùy chuẩn), đảm bảo tương thích trực tiếp với vi điều khiển như ESP32 hay Raspberry Pi. Khi dữ liệu được gửi đi, chân TX của thiết bị phát sẽ hạ mức định sẵn để ghi start bit, sau đó chuyển trạng thái liên tiếp theo từng bit dữ liệu rồi đưa về mức nghỉ (idle) qua stop bit; chân RX của thiết bị nhận ngược lại sẽ “bắt” từng bit ở những thời điểm cách đều nhau theo baud rate đã chọn, giải mã khung và đưa lên bộ đệm để phần mềm có thể đọc.

Ưu điểm của UART là độ phức tạp phần cứng rất thấp, dễ triển khai trên bất kỳ vi điều khiển nào hỗ trợ ngoại vi UART, và phù hợp cho truyền tải các chuỗi ký tự ASCII nhẹ nhàng, như trong hệ thống ArUco trên Raspberry Pi gửi ID, tọa độ và góc dấu chấm câu qua UART về ESP32 để xử lý tiếp. Vì không cần xung nhịp chung, UART rất linh hoạt cho kết nối điểm–điểm trong những ứng dụng nhúng, robot hay IoT.

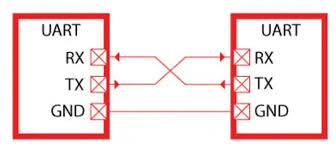


Figure 5. Giao tiếp UART

Phần mềm trên Raspberry Pi khởi tạo cổng UART0 với tốc độ 115 200 baud, sử dụng khung dữ liệu 8 bit, không parity và 1 stop bit. Ngay sau khi mở cổng, chương trình tạm dừng khoảng 2 giây để đảm bảo đường truyền ổn định trước khi bắt đầu gửi dữ liệu. Mỗi khi một mã ArUco được phát hiện, hệ thống sẽ đóng gói ID, tọa độ tâm và góc nghiêng của marker thành một chuỗi ký tự ASCII, trong đó các giá trị được ngăn cách bằng dấu phẩy và kết thúc bởi ký tự xuống dòng. Chuỗi này sau đó được gửi ngay lập tức xuống cổng UART, đảm bảo ESP32 hay bất kỳ thiết bị nào nối với Pi sẽ nhận được thông tin gần như tức thời.

Bên phía ESP32, lớp tiện ích Uart2 khởi tạo lại cổng nối tiếp với cùng thông số 115 200‑8N1 và đặt thời gian chờ (timeout) ngắn, khoảng 20 ms, để cuộc gọi đọc không bị treo. Khi có dữ liệu đến, ESP32 sẽ đọc toàn bộ chuỗi đến dấu xuống dòng, loại bỏ các ký tự khoảng trắng dư thừa và kiểm tra tiền tố “raspi\_camera” để đảm bảo đây là dữ liệu đúng định dạng. Tiếp theo, chuỗi được tách thành các phần theo dấu phẩy; phần đầu xác định ID, các phần tiếp theo tương ứng với tọa độ và góc nghiêng được chuyển đổi thành số nguyên hoặc số thực. Kết quả là một cấu trúc dữ liệu gồm ID, x, y và góc, sẵn sàng cho các bước xử lý tiếp theo như điều khiển động cơ hay tính toán định vị.

2.2. Khối Encder

Bộ mã hóa xung (incremental encoder) vận hành như một “cây đinh” chuyển đổi chuyển động quay thành xung điện để vi điều khiển có thể “đếm” và tính toán. Về cấu tạo, nó bao gồm một đĩa tròn có khắc xen kẽ các khe hở và phần chắn, gắn cố định vào trục quay, cùng với một cặp đèn LED – photodiode đặt hai bên. Khi trục bánh xe quay, ánh sáng từ LED chỉ có thể xuyên qua những khe, rồi bị chặn lại tại phần đặc, tạo thành một chuỗi tín hiệu số (0–1) tương ứng với từng khe đi qua. Mỗi lần một khe đi qua vị trí cảm biến, xuất hiện một xung điện – và bằng cách đếm liên tục số xung, ta biết được tổng góc quay hay quãng đường di chuyển so với điểm xuất phát.

Ở những ứng dụng yêu cầu xác định cả chiều quay, người ta còn trang bị hai kênh cảm biến A và B đặt lệch nhau 90° pha. Khi đĩa quay, tín hiệu A và B sẽ thay đổi trạng thái xen kẽ, và vi điều khiển so sánh xem kênh nào thay đổi trước để suy ra hướng tiến hoặc lùi. Nhờ vậy, chỉ với việc phân tích mối quan hệ pha và đếm xung, bộ mã hóa incremental encoder vừa cho biết được chính xác góc/quãng đường quay vừa xác định được chiều quay, mà vẫn giữ được độ tin cậy cao và phản hồi thời gian thực.

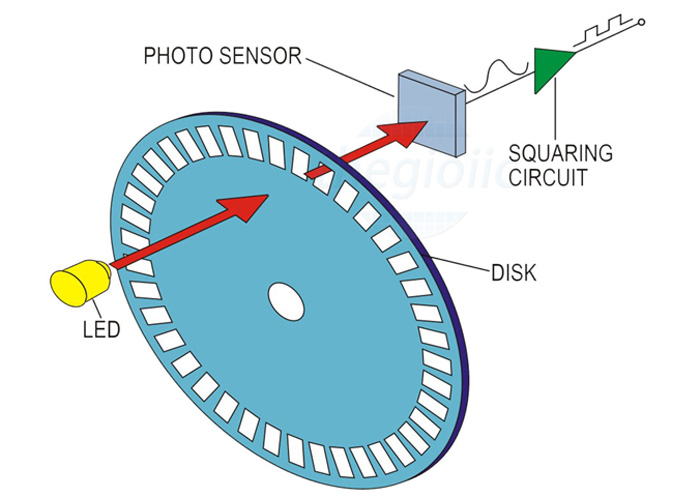


Figure 6. Nguyên lý hoạt động encoder

Khối Encoder trên ESP32 đảm nhiệm nhiệm vụ đếm xung từ hai cảm biến gắn trên trục bánh xe, từ đó xác định số vòng quay và quãng đường di chuyển. Quá trình này diễn ra như sau:

Khi khởi động, hàm begin() cấu hình hai chân tín hiệu encoder ở chế độ đầu vào với điện trở kéo lên nội (INPUT\_PULLUP), giúp giữ mức tín hiệu ổn định khi cảm biến không phát xung .

Ngay trước khi bắt đầu đo, khối Encoder gọi hàm start(), trong đó bộ đếm được đưa về giá trị 0 và hai ngắt ngoài (external interrupt) được gán cho hai chân encoder, kích hoạt mỗi khi tín hiệu chuyển từ mức thấp lên cao. Mỗi lần ngắt xảy ra, một trình xử lý ngắt ngắn gọn (ISR) sẽ kiểm tra xem khoảng thời gian tính bằng micro-giây kể từ xung trước đó có vượt quá ngưỡng chống rung (500 µs) hay không. Nếu thỏa, biến đếm tương ứng sẽ được tăng lên và thời điểm xung hiện tại được ghi lại, nhờ vậy hiệu quả loại bỏ nhiễu cơ khí, rung lắc được đảm bảo .

Trong suốt quá trình vận hành, hai biến đếm cho bánh trái và bánh phải được lưu giữ ở dạng volatile, cho phép truy xuất nhanh chóng và nhất quán dù đang ở trong ngắt . Khi kết thúc chu kỳ hoặc cần tạm dừng đo đạc, hàm stop() sẽ tháo gỡ cả hai ngắt, ngăn chặn bất kỳ đếm nhầm nào khi hệ thống không yêu cầu .

Để lấy dữ liệu, hai phương thức getL() và getR() đơn giản trả về số xung đã đếm cho mỗi bánh, sẵn sàng cho các thuật toán điều khiển đường đi hoặc tính toán quãng đường di chuyển chính xác.

Với thiết kế ngắt phần cứng kết hợp cơ chế chống rung, khối Encoder này cho phép ESP32 thu thập thông tin về số vòng quay bánh xe một cách ổn định, ít nhiễu, và thời gian thực, làm cơ sở cho các hệ thống định vị và điều hướng xe.

2.3. Khối HC-SR04

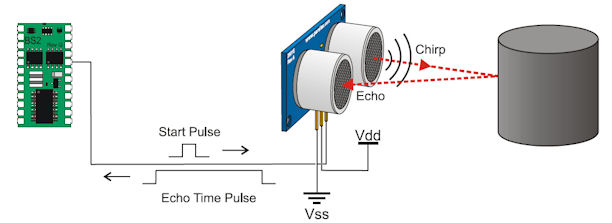


Figure 7. Nguyên lý Hoạt động HC-SR04

Nguyên lý hoạt động của Cảm biến siêu âm HC-SR04

Cảm biến siêu âm HC-SR04 là một module đo khoảng cách sử dụng nguyên lý phản xạ sóng siêu âm (ultrasonic reflection principle). Hoạt động của cảm biến dựa trên việc phát ra một xung sóng siêu âm và đo khoảng thời gian cần thiết để sóng đó truyền tới vật cản và phản xạ trở lại cảm biến. Nguyên lý này còn được gọi là phương pháp đo thời gian bay (Time of Flight - ToF).

Cấu tạo của module HC-SR04 bao gồm một bộ phận phát sóng siêu âm (transmitter) và một bộ phận thu sóng siêu âm (receiver), thường là các đầu dò gốm áp điện hoạt động ở tần số siêu âm (đặc trưng là 40 kHz).

Quá trình đo khoảng cách được thực hiện theo các bước sau:

1. Kích hoạt (Trigger): Để bắt đầu một chu kỳ đo, bộ điều khiển bên ngoài (ví dụ: vi điều khiển) sẽ tạo một xung dương (logic HIGH) có độ rộng tối thiểu 10 micro giây (μs) tại chân Trig của module HC-SR04.
2. Phát sóng siêu âm (Emission): Sau khi nhận tín hiệu kích hoạt, mạch xử lý trên module sẽ điều khiển bộ phát sóng phát ra một loạt (thường là 8 chu kỳ) sóng siêu âm với tần số 40 kHz vào môi trường.
3. Đồng hồ đo thời gian và Tín hiệu Echo (Timing & Echo Signal): Ngay tại thời điểm sóng siêu âm được phát ra, chân Echo của module sẽ tự động chuyển trạng thái từ mức thấp (LOW) lên mức cao (HIGH). Chân Echo này sẽ duy trì ở mức HIGH trong suốt thời gian sóng siêu âm truyền đi và quay trở lại.
4. Phản xạ và Thu sóng (Reflection & Reception): Sóng siêu âm lan truyền trong không khí. Khi gặp một vật cản trên đường đi, một phần năng lượng của sóng sẽ bị phản xạ ngược trở lại theo hướng cảm biến. Bộ phận thu sóng sẽ lắng nghe tín hiệu phản xạ này.
5. Kết thúc Tín hiệu Echo và Ghi nhận Thời gian (Echo Termination & Time Measurement): Khi bộ thu phát hiện và nhận được tín hiệu sóng siêu âm phản xạ quay trở lại, mạch xử lý trên module sẽ kéo chân Echo trở về mức thấp (LOW). Khoảng thời gian mà chân Echo duy trì ở mức HIGH chính là tổng thời gian sóng siêu âm đã truyền đi từ cảm biến đến vật cản và quay trở lại. Đây chính là giá trị Thời gian bay (ToF).

Tính toán Khoảng cách:

Dựa trên giá trị ToF đo được và biết trước vận tốc truyền âm trong không khí (v), khoảng cách d từ cảm biến đến vật cản có thể được xác định bằng công thức sau:



Trong đó:

* d là khoảng cách từ cảm biến đến vật cản.
* ToF là khoảng thời gian chân Echo ở mức HIGH (thời gian sóng đi và về).
* v là vận tốc truyền âm trong không khí, giá trị này phụ thuộc vào nhiệt độ môi trường (ở 20°C, v≈343m/s, tương đương khoảng 29.1μs/cm hoặc 34300cm/s).

Việc chia cho 2 trong công thức là do ToF đo được là thời gian cho cả quãng đường sóng siêu âm di chuyển từ cảm biến đến vật cản và quay ngược lại từ vật cản về cảm biến.

Như vậy, nguyên lý hoạt động của HC-SR04 là một ứng dụng của phương pháp định vị bằng sóng siêu âm, cho phép xác định khoảng cách một cách không tiếp xúc dựa trên việc đo chính xác thời gian phản hồi của tín hiệu.

2.4. Khối Blynk Handler

Chức năng của Khối Xử lý Blynk (Blynk Handler)

Khối xử lý Blynk, được hiện thực hóa bởi lớp BlynkHandler, đóng vai trò là tầng giao tiếp cốt lõi giữa hệ thống điều khiển phương tiện và nền tảng dịch vụ Internet of Things (IoT) Blynk. Nhiệm vụ chính của khối này là thiết lập, duy trì kết nối, và xử lý toàn bộ luồng dữ liệu tương tác giữa thiết bị (phương tiện) và ứng dụng điều khiển trên nền tảng Blynk.

Vai trò cụ thể của khối xử lý Blynk bao gồm:

1. Quản lý kết nối: Chịu trách nhiệm khởi tạo kết nối ban đầu tới máy chủ Blynk, sử dụng các thông số cấu hình như mã token dự án, thông tin mạng Wi-Fi (SSID và mật khẩu). Khối này cũng đảm bảo rằng kết nối được duy trì ổn định trong suốt quá trình hoạt động của hệ thống.
2. Tiếp nhận lệnh điều khiển từ xa: Lắng nghe các sự kiện và dữ liệu được gửi xuống từ ứng dụng Blynk. Các lệnh điều khiển này thường được biểu diễn dưới dạng thay đổi giá trị trên các "chân ảo" (Virtual Pins) trên giao diện người dùng của ứng dụng.
3. Thông dịch lệnh và điều khiển phương tiện: Phân tích các giá trị nhận được từ các chân ảo tương ứng. Dựa trên giá trị này, khối xử lý Blynk sẽ gọi các phương thức hoặc thay đổi trạng thái của đối tượng phương tiện (Vehicle) được liên kết. Điều này bao gồm các tác vụ như điều khiển hướng di chuyển (tiến, lùi, quay), thiết lập tốc độ, kích hoạt các chức năng đặc thù của phương tiện (ví dụ: nâng/hạ một cơ cấu), hoặc bắt đầu/kết thúc một nhiệm vụ tự động.
4. Cập nhật trạng thái lên Blynk (Tiềm năng): Mặc dù mã nguồn cung cấp chủ yếu cho thấy chức năng nhận lệnh, khối này cũng bao gồm một phương thức cho phép gửi dữ liệu từ phương tiện lên các chân ảo trên ứng dụng Blynk. Điều này cho phép hiển thị các thông tin về trạng thái hoạt động, dữ liệu cảm biến, hoặc tiến trình thực hiện nhiệm vụ của phương tiện trên giao diện người dùng từ xa.
5. Đồng bộ hóa hoạt động: Đảm bảo sự phối hợp nhịp nhàng giữa việc xử lý các tác vụ nội bộ của phương tiện và việc giao tiếp với nền tảng Blynk, duy trì tính phản hồi của hệ thống.

Tóm lại, khối xử lý Blynk là bộ phận trung tâm cho phép hệ thống phương tiện có thể nhận lệnh điều khiển từ người dùng qua ứng dụng di động từ bất kỳ đâu có kết nối Internet và có khả năng gửi ngược thông tin về cho người dùng, tạo nên khả năng điều khiển từ xa hoàn chỉnh.

2.5. Khối thuật toán A\*  
Trong đồ án này, để giải quyết bài toán tìm đường đi ngắn nhất trên lưới ô vuông chứa chướng ngại vật và sử dụng thuật toán A\*. A\* là một phương pháp tìm kiếm có hướng, kết hợp giữa chi phí thực tế đã đi và ước lượng chi phí còn lại đến đích, giúp giảm thiểu số nút phải thăm so với các thuật toán tìm kiếm thuần túy như Dijkstra. Về cơ bản, mỗi ô trên bản đồ được gán một giá trị đánh giá tổng hợp f=g+h, trong đó g là chi phí từ vị trí xuất phát đến ô hiện tại, còn h là hàm heuristic ước lượng khoảng cách từ ô hiện tại đến đích.

Với hàm heuristic chọn là khoảng cách Manhattan (tổng độ chênh theo trục x và y), A\* đảm bảo tính tối ưu và tính nhất quán trong môi trường lưới không cho phép đi chéo hoặc có chi phí không đồng nhất. Thuật toán bắt đầu từ ô nguồn, đưa ô này vào tập “mở” và lặp lại các bước: lấy ô có f nhỏ nhất ra để mở rộng, đánh dấu ô đó đã khám phá, rồi cập nhật chi phí cho các ô lân cận chưa hoặc cần cập nhật lại. Mỗi khi xét một ô láng giềng, nếu chi phí thu được thấp hơn chi phí đã lưu trước đó, A\* sẽ ghi nhận đường dẫn hiệu quả hơn và cập nhật giá trị g, đồng thời tính lại f. Quá trình dừng lại khi ô đích được mở rộng, và đường đi tối ưu được truy ngược qua các liên kết cha – con đã lưu.

Trong thực nghiệm của đồ án, A\* cho phép hệ thống tìm được đường đi ngắn nhất một cách nhanh chóng ngay cả trên những bản đồ lớn với tỷ lệ chướng ngại vật cao. Nhờ hàm heuristic khôn ngoan, số ô được thăm giảm đáng kể, tiết kiệm cả bộ nhớ lẫn thời gian tính toán. Tuy nhiên, khi kích thước lưới quá lớn hoặc heuristic không đủ “gần” giá trị thực, A\* có thể vẫn phải mở rộng nhiều nút, ảnh hưởng đến hiệu năng. Do đó, đồ án cũng thảo luận phương án cải tiến như chia vùng tìm kiếm, hoặc kết hợp với thuật toán Jump Point Search để tăng tốc hơn nữa.

2.6. Khối main

Dự án được triển khai dưới dạng nhiều “tác vụ” (task) chạy song song trên FreeRTOS, mỗi tác vụ đảm nhiệm một phần chức năng cụ thể, phối hợp với nhau qua tập hợp các cờ (flags) và trạng thái (state) của đối tượng quản lý xe (Vehicle). Luồng hoạt động tổng thể của hệ thống có thể được mô tả như sau:

1. Khởi động và khởi tạo  
   Khi hệ thống bật lên, hàm setup() sẽ tuần tự khởi tạo tất cả các thành phần:
   * Giao tiếp UART với Raspberry Pi và Blynk.
   * Bộ đếm xung (Encoder) để ghi nhận quãng đường và góc quay bánh xe.
   * Thuật toán A\* để tính đường đi.
   * Cảm biến siêu âm (HC-SR04) để phát hiện chướng ngại.  
     Cuối cùng, 8 tác vụ FreeRTOS được tạo ra, cho phép chạy đồng thời các chức năng quan trọng như xử lý lệnh từ smartphone (Blynk), điều khiển động cơ, xử lý dữ liệu QR/ArUco, và giám sát chướng ngại .
2. Kích hoạt nhiệm vụ (Mission) qua Blynk  
   Tác vụ taskRunBlynk liên tục lắng nghe lệnh khởi động từ ứng dụng di động. Khi người dùng nhấn “Start”, hệ thống sẽ:
   * Xác định hướng di chuyển ban đầu dựa trên góc ArUco gửi từ Pi.
   * Gọi A\* để sinh ra một chuỗi các bước (steps) từ vị trí xuất phát đến đích.
   * Đặt cờ bắt đầu (running = true) và chuyển trạng thái xe sang “START” để các tác vụ khác tham gia thực thi .
3. Điều khiển chung và phân chia bước  
   Tác vụ taskControl hoạt động như “bộ não trung tâm”, theo dõi các trạng thái:
   * QR Available: khi có dữ liệu ArUco hợp lệ, tính sai số góc so với hướng bước tiếp theo, đặt cờ alignQR nếu cần căn chỉnh.
   * DONE\_STEP → RUN\_QR → DONE\_QR → RUN\_STEP: lần lượt từ dừng sau mỗi bước, căn chỉnh góc, di chuyển tới mã tiếp theo, rồi lặp lại.
   * Khi hoàn tất tất cả bước hoặc chướng ngại xuất hiện, cập nhật trạng thái dừng hoặc tái lập lộ trình mới .
4. Di chuyển thẳng với phản hồi encoder  
   Tác vụ taskForward đảm bảo xe chạy thẳng:
   * Đọc sai lệch số xung giữa hai bánh (qua Encoder).
   * Điều chỉnh tốc độ hai bên để bù chênh lệch, giữ quỹ đạo ổn định.
   * Luôn khởi động/báo hiệu encoder trước khi di chuyển, dừng encoder sau khi hoàn thành .
5. Quay góc cố định  
   Bốn tác vụ taskl90, taskr90, taskl180 và taskBack thực hiện các thao tác quay trái 90°, quay phải 90°, quay 180° và lùi:
   * Mỗi tác vụ dựa vào số xung gom được để xác định đã quay đủ góc hay đi đủ quãng đường.
   * Khi đạt ngưỡng xung định sẵn, dừng động cơ, reset encoder và chuyển cờ về di chuyển thẳng .
6. Căn chỉnh qua mã ArUco (QR)  
   Tác vụ taskAlignQR chịu trách nhiệm điều khiển “mềm” (PD control) để xoay xe cho đến khi sai số góc so với mã ArUco nằm trong khoảng cho phép (±1.5°), sau đó dừng và chuyển sang bước di chuyển tiếp theo .
7. Giám sát chướng ngại  
   Tác vụ taskHCSR04 liên tục đo khoảng cách với hai cảm biến siêu âm. Khi phát hiện vật cản trong khoảng 10 cm và xe đang ở trạng thái đã hoàn thành một bước, hệ thống sẽ:
   * Dừng ngay lập tức, đánh dấu chướng ngại, thông báo lên Blynk.
   * Gọi A\* cập nhật lại lộ trình tránh vật cản, rồi tiếp tục thực hiện nhiệm vụ từ vị trí hiện tại .
8. Hoàn thành nhiệm vụ  
   Khi tác vụ taskControl nhận diện mã đích (goal) đã tới, xe dừng hẳn, phát hiệu hoàn thành lên Blynk và chờ lệnh mới.

Tóm lại, nhờ cơ chế chia nhỏ chức năng vào các tác vụ riêng biệt chạy song song, hệ thống phối hợp nhịp nhàng giữa việc nhận diện ArUco, đếm xung encoder, tính toán đường đi, giám sát chướng ngại và điều khiển động cơ, tạo nên một luồng thực thi mạch lạc, linh hoạt và có khả năng phản ứng thời gian thực trong các kịch bản robot di chuyển trong môi trường có chướng ngại.

1. **Xây dựng giao diện điều khiển Blynk**