

## THUẬT TOÁN THỜI GIAN THỰC, CHI PHÍ THẤP CHO HỆ THỐNG NHÚNG DỰA TRÊN NHÂN FREERTOS

Nguyễn Văn Khanh<sup>1</sup> và Trần Trọng Hiếu<sup>1</sup>

<sup>1</sup> Khoa Công nghệ, Trường Đại học Cần Thơ

**Thông tin chung:**

Ngày nhận: 19/09/2015

Ngày chấp nhận: 10/10/2015

**Title:**

*A Low-cost, Real-time methodology for embedded devices based on open source FreeRTOS kernel*

**Từ khóa:**

*Mã nguồn mở, hệ điều hành thời gian thực, hệ thống nhúng, ARM Cortex, bộ điều khiển PID mờ*

**Keywords:**

*Open source, RTOS, embedded system, ARM Cortex, fuzzy PID controller*

**ABSTRACT**

*A low-cost, real-time methodology for embedded devices based on well-known open source kernel - freeRTOS is presented in this study. The real-time algorithm designed consists of three main steps. Firstly, algorithm is designed and evaluated by utilizing Matlab/Simulink toolboxes. Secondly, the generic embedded C code is generated by Matlab program. Finally, freeRTOS Tasks code is utilized based on the generated C code to build and run on an embedded target. This real-time algorithm is demonstrated on a two-wheeled self-balancing robot which is employed a fuzzy PID self-tuning controller. The designed controller is executed on famous ARM Cortex M4 core microcontroller STM32F407VTG. The experimental results show that algorithm designed operated well on embedded system. The tracking position and rotation angle response in maximum delay 1.5 seconds which is fast enough while stabilizing the two-wheeled at upright. The real-time system designed is a low cost methodology and suitable for fresh embedded system designers.*

**TÓM TẮT**

*Bài báo trình bày một phương pháp thực hiện thuật toán điều khiển nhúng với chi phí thấp sử dụng nhân hệ điều hành thời gian thực nhúng freeRTOS. Phương pháp đề nghị gồm ba bước chính. Bước thứ nhất, thuật toán sẽ được thiết kế và kiểm chứng sử dụng Matlab/Simulink toolboxes. Bước thứ hai, thuật toán này sẽ được phát sinh mã nguồn ngôn ngữ C cho hệ thống nhúng bằng công cụ Matlab Realtime Embedded Coder. Bước cuối cùng, các tác vụ freeRTOS được lập trình dựa trên mã lệnh sinh ra để biên dịch và thực thi thuật toán trên hệ thống nhúng. Thuật toán PID mờ tự động hiệu chỉnh thông số điều khiển một robot hai bánh tự cân bằng được chọn để minh chứng phương pháp được đề nghị. Thuật toán này được chạy trên vi điều khiển nhân ARM Cortex M4 nổi tiếng STM32F407VTG. Kết quả thực nghiệm cho thấy thuật toán điều khiển nhúng đã hoạt động chính xác, robot vừa cân bằng vừa cho đáp ứng bám góc và vị trí nhanh với độ trễ tối đa 1.5 giây. Giải pháp đề nghị giúp có thể phát triển thuật toán điều khiển nhúng phức tạp với chi phí thấp phù hợp cho hầu hết các nhà phát triển hệ thống nhúng.*

## 1 GIỚI THIỆU

Hệ thống nhúng thời gian thực đã được sử dụng rộng rãi trong nhiều ứng dụng như hàng không, công nghiệp và nhiều lĩnh vực khác. Hệ thống thời gian thực đòi hỏi phải xử lý xong công việc trong một khoảng thời gian yêu cầu, khoảng thời gian này gọi là “deadtime”. Để đạt được mục đích này, hệ điều hành thời gian thực RTOS (Real-Time Operating System) thường được sử dụng để thiết kế các hệ thống thời gian thực này. Hiện tại, có nhiều RTOS đang được các hãng thiết kế và cung cấp cho các nhà phát triển như FreeRTOS,  $\mu$ COS, CMSIS-RTOS, Pumpkin OS, ChibiOS/RT. Phần lớn các RTOS này được cung cấp với bản quyền mã nguồn mở và được chấp nhận bởi nhiều hãng sản xuất vi mạch lớn hiện nay như TI, ST, Atmel. Hrushit Shah *et al.* đã công bố một nghiên cứu về việc so sánh ba hệ điều hành thời gian thực mã nguồn mở nổi tiếng hiện nay là RT-Linux, FreeRTOS và eCOS dựa trên cùng bộ xử lý và một số tác vụ kiểm tra [1]. Nghiên cứu này cung cấp tài liệu kỹ thuật hữu dụng và tìm ra nhiều thông số quan trọng giúp các nhà phát triển lựa chọn được một RTOS phù hợp nhất cho các ứng dụng của họ. Tương tự, Douglas P. B. Renaux đã thực hiện so sánh khả năng thực thi giữa các RTOS được sử dụng để tùy chỉnh thành CMSIS-RTOS dựa trên hai nhóm RTOS chính: thương mại và FOSS (Free and Open Source Software) [2]. Các kết quả nghiên cứu cho thấy FOSS RTOS có các thông số về thời gian trội hơn nhóm còn lại mặc dù nó được cung cấp miễn phí. RTOS cũng được sử dụng trong nhiều nghiên cứu khác. Cụ thể là một nghiên cứu sử dụng chuẩn CMSIS-RTOS để phát triển IoTs (Internet of Things) được thực hiện bởi Douglass đã hấp dẫn nhiều nhà nghiên cứu khác [3]. Nghiên cứu này chỉ ra rằng CMSIS-RTOS hỗ trợ nhiều thư viện HAL, middlewares phù hợp để ứng dụng vào công nghệ IoTs. Nó giúp giảm thời gian thiết kế và đưa hệ thống vào thương mại với giá thành hợp lý. Một nghiên cứu khác được thực hiện bởi Jorge Cabrera-Gasmez [4] đã thiết kế thành công bộ điều khiển thời gian thực một thuyền bươm sử dụng ChibiOS/RT hoạt động trên vi điều khiển SAM3X8E.

Hiện nay, nhiều ứng dụng trong lĩnh vực công nghệ thông tin cũng đang hướng thực hiện trên hệ thống nhúng. Cụ thể, nhiều khóa học về lập trình hệ thống nhúng cơ bản đã được đưa vào giảng dạy ở các trường đại học. Các khóa này thường được minh họa trên mạch phát triển MSP430 LaunchPad được thiết kế bởi công ty Texas Instrument. Đây là một mạch phát triển được thiết kế theo hướng dễ sử

dụng dựa trên vi điều khiển siêu tiết kiệm năng lượng, giá thành thấp MSP430G2x [5]. Bên cạnh đó, một số nhà nghiên cứu trong lĩnh vực này cũng đã sử dụng robot để phục vụ cho nghiên cứu của họ. Một ví dụ điển hình là Thanh M.T và *ctv.* đã phát triển một ứng dụng bám đối tượng dựa trên robot Pioneer P3-DX [6]. Trong nghiên cứu này, các thuật toán xử lý ảnh và hệ điều hành ARCOS, một hệ điều hành nhúng phát triển chuyên dụng cho robot Pioneer, đã được sử dụng để phát triển ứng dụng. Một camera tích hợp trên robot sẽ liên tục chụp ảnh và gửi về máy tính cá nhân để nhận dạng đối tượng và gửi lệnh ngược lại điều khiển robot bám đối tượng. Kết quả nghiên cứu đã chứng minh được robot đã bám tốt đối tượng. Tuy nhiên, điểm bất lợi của nghiên cứu này là sử dụng một nền tảng phần cứng có sẵn với giá thành cao, tốn nhiều thời gian khảo sát và nghiên cứu để có thể phát triển được ứng dụng.

Song song với các nghiên cứu trên, bài báo này trình bày một phương pháp mới để có thể ứng dụng RTOS đưa các thuật toán được thiết kế bởi Matlab/Simulink vào một hệ thống nhúng chi phí thấp, tiết kiệm năng lượng và dễ sử dụng. Thiết kế phần cứng dựa trên kiến trúc ARM Cortex-M4 32-bit chạy hệ điều hành thời gian thực FreeRTOS. Đây là một hệ điều hành tin cậy và ổn định phù hợp cho nghiên cứu và công nghiệp.

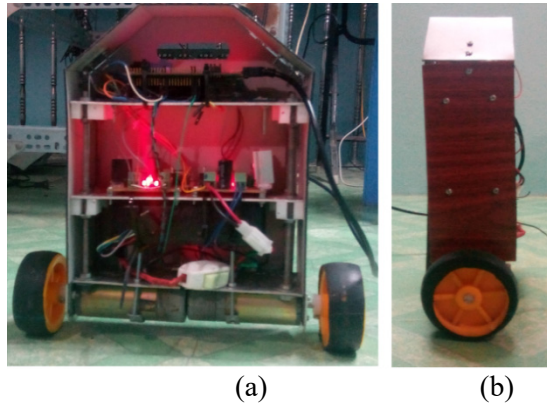
## 2 THUẬT TOÁN

Bộ điều khiển PID mờ tự chỉnh điều khiển một robot hai bánh tự cân bằng được chọn để minh họa cho phương pháp đề nghị trong nghiên cứu này. Mô hình của robot được thiết kế như Hình 1. Thuật toán điều khiển robot được thiết kế như Hình 2. Thuật toán này có ba vòng điều khiển chính. Vòng thứ nhất Fuzzy PD1 sử dụng bộ điều khiển PD mờ để tính toán góc nghiêng tham chiếu  $r_o$  cho bộ điều khiển Fuzzy PD2 dựa trên lỗi khoảng cách so với vị trí đặt của robot. Vòng thứ hai Fuzzy PD2 cũng sử dụng một bộ điều khiển PD mờ để tính toán một phần của tín hiệu ra  $u_y$  dựa vào  $r_o$ . Vòng thứ ba sử dụng bộ điều khiển PID tự chỉnh để điều khiển góc xoay của robot, tính toán tín hiệu ra  $u_x$  còn lại của hệ thống. Để điều khiển robot, hai tín hiệu ra  $u_x$  và  $u_y$  được sử dụng để tính toán hai tín hiệu  $u_R$  và  $u_L$  điều khiển robot theo hai công thức (1) và (2) như sau [8].

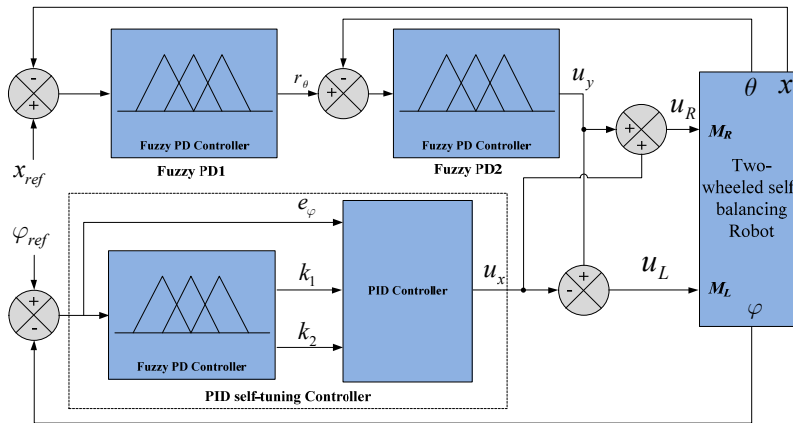
$$u_R = u_y + u_x \quad (1)$$

$$u_L = u_y - u_x \quad (2)$$

Trong đó  $u_R$ ,  $u_L$  là tín hiệu ra điều khiển hai động cơ bên phải và bên trái của robot.



Hình 1: (a) Mặt trước và (b) mặt bên robot



Hình 2: Thuật toán điều khiển robot

### 3 PHƯƠNG PHÁP THỰC HIỆN BỘ ĐIỀU KHIỂN NHÚNG

FreeRTOS là một hệ điều hành thời gian thực mạnh, hỗ trợ rất nhiều kiến trúc phần cứng, một số kiến trúc có thể kể đến như Intel, ARM, 8051, PIC. Mục đích của nghiên cứu này là trình bày một phương pháp để ứng dụng RTOS mà cụ thể là FreeRTOS để thực hiện các thuật toán chạy thời gian thực. Thuật toán điều khiển một robot hai bánh tự cân bằng được đề nghị để minh chứng phương pháp đề nghị. Các phần giới thiệu về RTOS, tổng quan về nền tảng phần cứng và phương pháp thực hiện bộ điều khiển nhúng ứng dụng RTOS sẽ lần lượt được trình bày ở các phần ngay sau đây.

#### 3.1 Giới thiệu hệ điều hành thời gian thực

RTOS - Realtime Operating System là một hệ điều hành được thiết kế để phát triển các ứng dụng nhúng thời gian thực, các ứng dụng này thường xử lý các tác vụ cần thực hiện với một độ trễ nhỏ. Độ trễ này được gọi là “deadline time” của hệ điều hành thời gian thực. Trong hệ thống thời gian thực,

“deadline time” là một thông số quan trọng vì thế nó cần được cân nhắc khi thiết kế hệ thống [7]. Hiện tại, có trên 30 hệ điều hành thời gian thực đã được phát triển, hỗ trợ rất nhiều họ vi điều khiển bao gồm cả những họ vi điều khiển có bộ nhớ rất thấp.

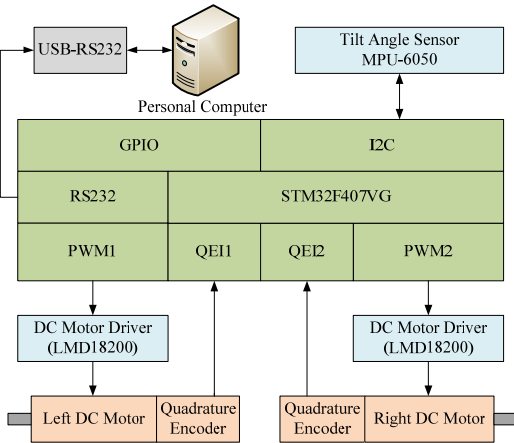
Có hai thuận lợi chính khi thiết kế ứng dụng với RTOS. Thứ nhất, các nhà thiết kế không cần tốn quá nhiều thời gian nghiên cứu, tìm hiểu phương pháp để có thể giao tiếp với các ngoại vi của vi điều khiển vì nó tích hợp gần như hoàn chỉnh các thư viện HAL (Hardware Abstraction Layer) và hỗ trợ rất nhiều họ vi điều khiển khác nhau. Thứ hai, ứng dụng RTOS được thiết kế theo hướng tác vụ rất hiệu quả với các hệ thống phức tạp và dự án làm việc nhóm.

FreeRTOS là một RTOS được thiết kế bởi Real Time Engineer Ltd. Nhân hệ điều hành này rất nhỏ gọn và được viết bằng ngôn ngữ C. Hiện tại, FreeRTOS đã và đang phân phối theo giấy phép mã nguồn mở GPL (General Public License) với các ngoại lệ tùy chọn và nó đã hỗ trợ lên đến 35 họ vi điều khiển của nhiều hãng khác nhau [6]. Bộ lập

lich của FreeRTOS có thể tùy chọn giữa hai thuật toán lập lịch đoạt quyền (pre-emption) và không đoạt quyền (co-operation), điều này được cấu hình trong tập tin freertosconfig.h khi lập trình. Khi bộ lập lịch được bắt đầu, tác vụ có độ ưu tiên cao nhất sẽ chiếm bộ xử lý và thực thi trước. Nếu có nhiều hơn một tác vụ có cùng độ ưu tiên thì bộ lập lịch sẽ sử dụng thuật toán xoay vòng (round-robin) để lập lịch cho các tác vụ này.

### 3.2 Nền tảng phần cứng

Phần cứng được thiết kế trong nghiên cứu này sử dụng họ vi điều khiển nổi tiếng STM32F4xx của hãng STMicroelectronics. Đây là họ vi điều khiển được thiết kế trên nền tảng nhân ARM Cortex M4. Vi điều khiển cụ thể được chọn là STM32F407VG, nó được tích hợp bộ tính toán số dấu chấm động FPU (Floating Point Unit), tần số xung nhịp lên đến 168 MHz cùng với nhiều ngoại vi mạnh mẽ rất thích hợp cho việc thực thi các thuật toán điều khiển chạy thời gian thực. Cụ thể, nó được tích hợp bộ chuyển đổi tương tự sang số 12-bit ADC (Analog-to-Digital Converter) tốc độ cao, bộ nhớ Flash tốc độ cao dung lượng 1 MB, 6 bộ giải mã vòng quay phù hợp cho việc điều khiển thời gian thực robot hai nhánh tự cân bằng với giá chi phí thấp, tiết kiệm năng lượng. Sơ đồ khối phần cứng của hệ thống được mô tả trong Hình 3.



Hình 3: Sơ đồ khối phần cứng hệ thống

Theo Hình 3, góc nghiêng của robot được đo bởi MPU6050 đây là một cảm biến công nghệ MEMs tích hợp cảm biến gia tốc và con quay hồi chuyển 6 trục. Góc nghiêng và khoảng dịch chuyển của robot được tính toán dựa vào khoảng dịch chuyển của hai bánh xe được đo thông qua hai bộ

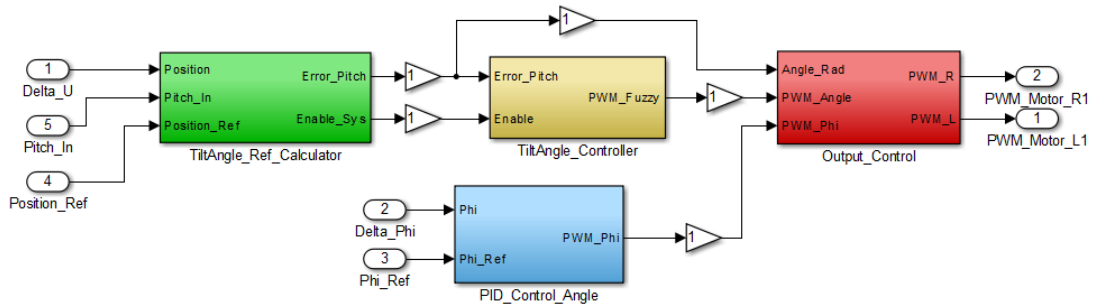
mã hóa vòng quay gắn trực tiếp lên hai bánh xe. Tốc độ quay của hai bánh xe được điều khiển thông qua hai động cơ DC bởi kỹ thuật điều chế độ rộng xung PWM (Pulse Width Modulation), đây là một phương pháp hiệu quả để điều khiển động cơ bởi việc thay đổi độ rộng của xung điều khiển. Xung điều khiển tác động đến động cơ qua một mạch công suất sử dụng vi mạch chuyên dụng LMD18200.

### 3.3 Xây dựng bộ điều khiển robot

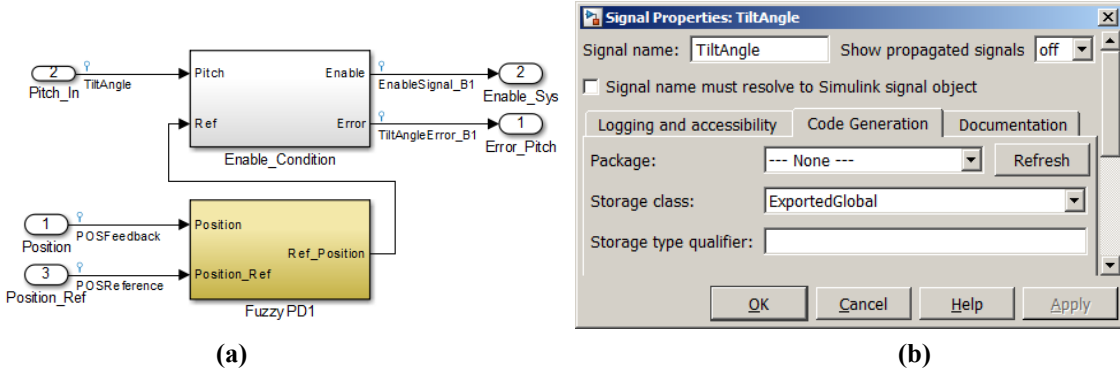
Thuật toán điều khiển được thiết kế bằng Matlab/Simulink thực hiện lần lượt theo 3 bước sau:

- Bước 1: Thiết kế thuật toán sử dụng Matlab/Simulink. Cấu trúc chương trình Matlab/Simulink của thuật toán đề nghị trong nghiên cứu này được thiết kế như Hình 4. Từ hình này cho thấy sơ đồ thiết kế của thuật toán gồm hai loại khối chính. Loại thứ nhất là các khối ngõ vào, các khối này nhận dữ liệu từ các ngõ vào hay từ các khối khác để tính toán giá trị ngõ ra. Loại thứ hai là khối ngõ ra, khối này nhận dữ liệu từ các ngõ ra của các khối ngõ vào đưa vào nó để tính toán các ngõ ra chung của thuật toán. Nếu hệ thống có những khối chạy một cách độc lập với thời gian lấy mẫu nó sẽ được thiết kế riêng thành một khối. Một tác vụ FreeRTOS sẽ được thiết kế để chạy mã lệnh của khối này khi các tác vụ phụ thuộc thời gian lấy mẫu thực thi của hệ thống. Điều này sẽ làm giảm thời gian thực thi của hệ thống. Một số ví dụ cho loại tác vụ này có thể kể đến như tác vụ cập nhập trạng thái, tác vụ truyền dữ liệu.

- Bước 2: Định nghĩa các tín hiệu vào/ra cho các khối trong chương trình Matlab/Simulink, những khối này sẽ được sinh mã sang mã nguồn ngôn ngữ C nhúng. Hình 5a minh họa khối Fuzzy PDI sau khi đã định nghĩa hoàn chỉnh các tín hiệu vào/ra. Khối này có 3 ngõ vào và 2 ngõ ra nên có 3 tín hiệu vào (TiltAngle, POSFeedback and POSReference) và 2 tín hiệu ra (EnableSignal\_B1 and TiltAngleError\_B1) được định nghĩa. Trong khi định nghĩa các tín hiệu, thuộc tính Storage class property của tín hiệu phải được chọn là ExportedGlobal để đảm bảo tín hiệu sẽ được sinh mã thành biến toàn cục. Điều này giúp truy xuất các tín hiệu trong chương trình C được dễ dàng hơn. Việc chọn thuộc tính của tín hiệu được minh họa như Hình 5b.



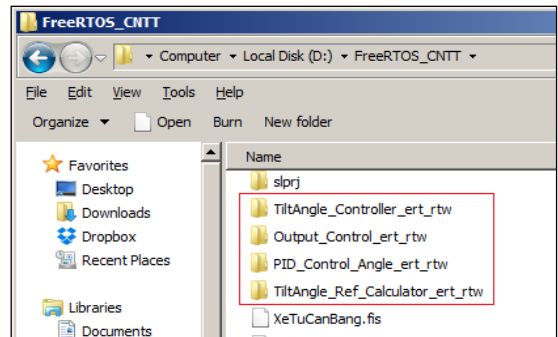
**Hình 4: Thuật toán Matlab/Simulink hoàn chỉnh**



**Hình 5: (a) Định nghĩa các tín hiệu vào/ra; (b). Cấu hình thuộc tính của tín hiệu**

– Bước 3: Cấu hình các thông số của Matlab/Simulinks để biên dịch thuật toán thành mã lệnh ngôn ngữ C nhúng. Ở bước này, có 3 thông số chính phải được cấu hình. Thứ nhất, mục Solver options cần chọn giá trị cho các thông số như Type, Solver và Fixed-step size. Lưu ý thông số Fixed-step size sẽ được chọn là 0.01 cho trường hợp của nghiên cứu này; tuy nhiên, giá trị này tùy thuộc vào từng ứng dụng cụ thể. Đây chính là thông số quy định thời gian lấy mẫu của hệ thống. Thứ hai, tập tin cài đặt cách thức sinh mã hệ thống được chọn là ert.tlc đây là tập tin viết bằng ngôn ngữ TLC (Target Language Compiler) định nghĩa các quy luật để sinh mã ngôn ngữ C cho hệ thống nhúng. Cuối cùng, tùy chọn Generate code only cần được chọn để trình sinh mã chỉ sinh mã ngôn ngữ C mà không sinh ra các tập tin thực thi khác để tránh phát sinh lỗi không cần thiết.

Hoàn thành 3 bước trên, thuật toán thiết kế trên Matlab/Simulink đã sẵn sàng để sinh mã thành ngôn ngữ C nhúng. Từng khối trong thuật toán sẽ được sinh mã riêng biệt. Đối với thuật toán trong nghiên cứu này sẽ có bốn khối được sinh mã và mã lệnh sinh ra sẽ được nằm trong bốn thư mục như Hình 6.



**Hình 6: Các thư mục chứa mã lệnh ngôn ngữ C nhúng được sinh mã**

Như đã được đề cập, thuật toán sẽ được thiết kế thành nhiều khối và được sinh mã riêng biệt. Để thực thi mã lệnh của các khối này trên hệ thống nhúng một chương trình FreeRTOS sẽ được phát triển để kết hợp và thực thi các mã lệnh này để phục hồi chính xác thuật toán đã thiết kế trên Matlab/Simulink. Sự chính xác này được đánh giá trên hai yếu tố: sự chính xác của thời gian lấy mẫu và đáp ứng thực tế của đối tượng. Mã lệnh của các khối sẽ được gọi bởi các tác vụ FreeRTOS, các tác vụ giao tiếp với nhau bằng cách truyền thông điệp.

Dựa trên chương trình Matlab/Simulink đã thiết kế, hệ thống sẽ có 3 loại tác vụ. Loại thứ nhất là các tác vụ thực thi mã lệnh của các khối ngõ vào. Các tác vụ này sẽ đọc dữ liệu từ tất cả các ngõ vào, thực thi mã lệnh của khối và gửi thông điệp chứa kết quả đến các tác vụ đang chờ dữ liệu. Khuôn mẫu mã lệnh của loại tác vụ này được đề nghị như sau.

```
static void BlockName( void *pvParameters){
    BlockName_initialize();
    for( ; ; ){
        //waiting for sampletime
        vTaskSuspend(NULL);
        //read all input
        ...
        //run generated code
        BlockName_step();
        //send results to other task
        xQueueSendToBack(xQueue1,&r1,0);
        xQueueSendToBack(xQueue1,&r2,0);
    }
}
```

Loại thứ hai là các tác vụ thực thi các mã lệnh của các khối nằm giữa các khối ngõ vào và ngõ ra. Tác vụ này sẽ chờ dữ liệu từ các tác vụ khác, thường là các tác vụ ngõ vào, trước khi thực thi mã lệnh của khối và gửi thông điệp chứa kết quả đến các tác vụ đang chờ dữ liệu. Khuôn mẫu mã lệnh của loại tác vụ này được đề nghị như sau.

```
static void BlockName( void *pvParameters){
    BlockName_initialize();
    for( ; ; ){
        //waiting for sampling time
        vTaskSuspend(NULL);
        //waiting for message
        if(xQueueReceive(xQueue1,&m1,500))
            okf1=1;
        if(xQueueReceive(xQueue1,&m2,500)){
            okf2=1;
        }
        //run generated code
        if(okf1 && okf2){
            BlockName_step();
            //send messages to other tasks
            xQueueSendToBack(xQueue2,&r1,0);
            xQueueSendToBack(xQueue2,&r2,0);
        }
    }
}
```

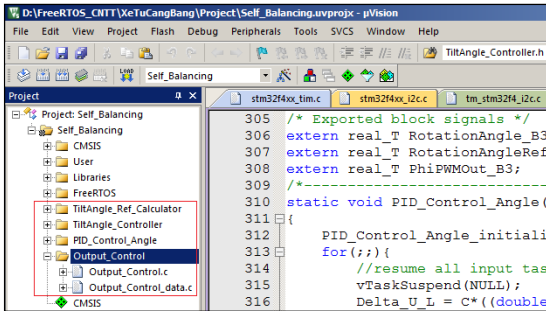
Loại thứ ba là tác vụ thực thi mã lệnh của khối ngõ ra. Tác vụ này cũng đợi dữ liệu từ các tác vụ khác, thực thi mã lệnh của khối và cập nhật tín hiệu của thuật toán để điều khiển đối tượng, trong trường hợp này là hai động cơ DC của robot. Thông thường chỉ cần thiết kế một khối ngõ ra nên chương trình FreeRTOS cũng chỉ có một tác vụ loại này. Khuôn mẫu mã lệnh của loại tác vụ này được đề nghị như sau.

```
static void BlockName( void *pvParameters){
    portTickType xLastWakeTime;
    xLastWakeTime = xTaskGetTickCount();
    BlockName_initialize();
    for( ; ; ){
        //sample time passed
        //resume all inputs tasks
        vTaskResume(hTask1);
        vTaskResume(hTask2);
        vTaskResume(hTask3);
        //waiting for messages
        if(xQueueReceive(xQueue1,&m1,500)){
            okf1=1;
        }
        if(xQueueReceive(xQueue1,&m2,500)){
            okf2=1;
        }
        if(xQueueReceive(xQueue1,&m3,500)){
            okf3=1;
        }
        //run generated code
        if(okf1 && okf2 && okf3){
            BlockName_step();
            //send control signal to plant
            ...
        }
        TaskDelayUntil(10);
    }
}
```

Tác vụ ngõ ra cũng là tác vụ chạy theo chu kỳ, nó được thực thi đều đặn sau mỗi lần thời gian lấy mẫu trôi qua. Thời gian này được tạo bởi hàm API (Application Programming Interface) TaskDelayUntil(). Hàm này sẽ đảm bảo tác vụ được kích hoạt để thực thi sau một khoảng thời gian lấy mẫu chính xác, thời gian này tính cả thời gian thực thi mã lệnh của tác vụ. Một điều cần lưu ý là giá trị truyền vào hàm này phải đảm bảo tạo ra thời gian trù hoãn bằng với giá trị thiết lập cho

thuộc tính Fixed-step size trên Matlab/Simulink, trường hợp này là 0.01[s] hay 10[ms] hay bằng mười nhịp hệ thống của FreeRTOS nên giá trị 10 sẽ được truyền vào hàm TaskDelayUntil().

Tất cả các mã lệnh sẽ được tích hợp lại thành một dự án RealView KeilC cho vi điều khiển STM32F407VG để biên dịch thành mã lệnh thực thi và chạy thời gian thực để điều khiển robot. Dự án hoàn chỉnh của thuật toán điều khiển robot được minh họa trong Hình 7.

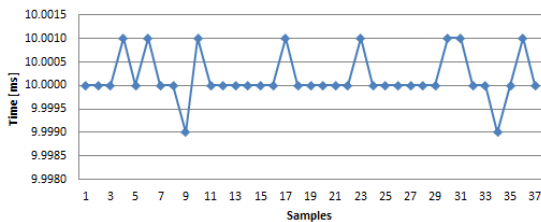


Hình 7: Dự án RealView KeilC sau cùng

## 4 KẾT QUẢ THỰC NGHIỆM

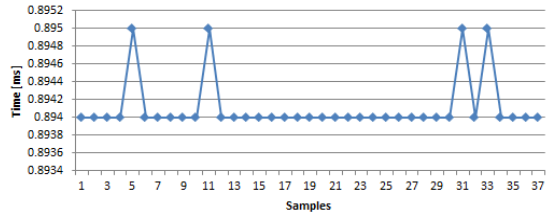
### 4.1 Thời gian lấy mẫu và thực thi thuật toán

Thời gian lấy mẫu được đo để khẳng định nó đã được tạo ra một cách chính xác trên hệ thống nhúng. Thời gian này được đo bằng một bộ định thời 32-bit của vi điều khiển STM32F407VG. Bộ định thời sẽ bắt đầu chạy tại thời điểm mã lệnh bắt đầu thực thi với giá trị bộ đếm bị xóa về 0. Giá trị của bộ đếm sẽ được đọc tại lần thực thi kế tiếp của mã lệnh. Giá trị này sẽ được chuyển thành đơn vị thời gian và có đồ thị như Hình 8 sau 37 lần đọc. Đồ thị cho thấy thời gian lấy mẫu được tạo ra một cách chính xác với sai số rất nhỏ 0.001[ms].



Hình 8: Thời gian lấy mẫu của hệ thống nhúng

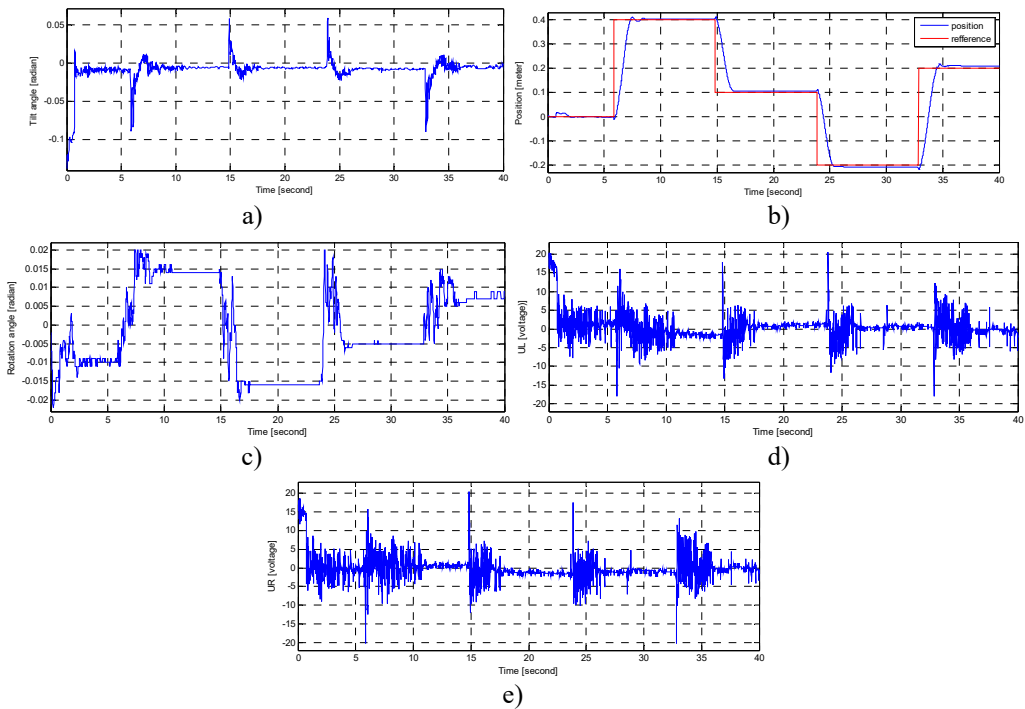
Bên cạnh thời gian lấy mẫu, thời gian thực thi của thuật toán đề nghị cũng được đo để thể hiện được sức mạnh của dòng vi điều khiển được chọn. Sử dụng cùng một phương pháp đo đạc, thời gian thực thi của thuật toán được thể hiện bằng đồ thị Hình 9. Đồ thị này cho thấy tổng thời gian thực thi toàn bộ thuật toán chỉ khoảng 0.9[ms] thấp hơn rất nhiều so với tổng thời gian lấy mẫu. Điều này cho thấy STM32F407VG là vi điều khiển có khả năng tính toán rất mạnh, nó hoàn toàn có thể thực thi các bộ điều khiển khác phức tạp hơn.



Hình 9: Thời gian thực thi thuật toán

### 4.2 Đáp ứng của robot

Phần này sẽ minh chứng thuật toán đã thiết kế được xây dựng thành công trên hệ thống nhúng. Trong thực nghiệm này, vị trí tham chiếu của robot được đặt lần lượt là 0.0, 0.4, 0.1, -0.2 và 0.2 [m] trong khi góc xoay của robot được giữ cố định ở 0 [rad] và cân bằng robot tại điểm cân bằng hướng lên. Hình 10 là đồ thị đáp ứng của robot khi được điều khiển bằng thuật toán PID mờ tự chỉnh thông số, đáp ứng này cũng đã khẳng định thuật toán đề nghị hoạt động tốt trên hệ thống nhúng khi áp dụng phương pháp được đề nghị trong nghiên cứu này. Robot hoàn toàn có thể vừa cân bằng vừa bám các tín hiệu tham chiếu ngõ vào. Robot đã bám tốt vị trí chiếu với sai số xác lập tối đa không quá 0.01[m], thời gian tăng nhỏ khoảng 1.5 [s] và vọt lố không quá 0.015 [m] (Hình 10b). Góc nghiêng robot tại các điểm đặt khi robot ổn định không vượt quá 1°. Góc xoay của robot bị thay đổi liên tục nhưng nó vẫn không vượt quá giới hạn 1.15° (Hình 10c). Các Hình 10d, 10e là đồ thị của hai tín hiệu  $u_R$  và  $u_L$  khi robot bám các tín hiệu tham chiếu. Ta thấy khi robot thay đổi vị trí  $u_R$  và  $u_L$  có giá trị lớn nhưng khi robot đã ổn định tại vị trí đặt thì hai tín hiệu này có giá trị rất nhỏ, gần như bằng 0.



Hình 10: Đáp ứng của robot khi vị trí tham chiếu thay đổi; a) góc nghiêng; b) vị trí; c) góc xoay; d)  $u_R$ ; e)  $u_L$

## 5 KẾT LUẬN

Một phương pháp xây dựng thuật toán thời gian thực sử dụng hệ điều hành mã mở FreeRTOS chạy trên hệ thống nhúng đã được đề nghị trong nghiên cứu này. Phương pháp này tạo điều kiện thuận lợi cho các nhà nghiên cứu sử dụng các thuật toán được thiết kế trên Matlab/Simulink vào hệ thống nhúng. Các kết quả thực nghiệm đã minh chứng được bộ điều khiển robot hai bánh tự cân bằng được xây bằng phương pháp đề nghị đã hoạt động chính xác, đáp ứng của robot nhanh với thời gian trễ tối đa 1.5[s]. Sai số góc nghiêng không quá  $1^0$ . Lỗi thời gian lấy mẫu rất nhỏ 0.001[ms]. Những kết quả này có thể so sánh được với các nghiên cứu trước đó. Phương pháp đề nghị này là một giải pháp để phát triển các hệ thống nhúng với chi phí thấp và nó phù hợp với hầu hết các nhà thiết kế hệ thống nhúng.

## TÀI LIỆU THAM KHẢO

1. Hrushit Shah, Rahil Shah, Udit Shah and Sanjay Deshmukh, 2013. Performance Parameters of RTOSs; Comparison of Open Source RTOSs and Benchmarking Techniques. International Conference on Advances in Technology and Engineering 101:1-6.
2. Renaux, D.P.B., 2014. Comparative Performance Evaluation of CMSIS-RTOS.

Brazilian Symposium on Computing Systems Engineering: 126-131.

3. Renaux, D.P.B., Pottker, F., 2014. Applicability of the CMSIS-RTOS Standard to the Internet of Things. International Symposium on Object/Component-Oriented Real-Time Distributed Computing: 284-291.
4. Jorge C.G., Angel R.M., 2014. A Real-Time Sailboat Controller Base don ChibiOS. Proceeding of the 7th International Robotic Sailing Conference: 77-84.
5. TI Corporation, 2015. MSP430 LaunchPad Value Line Development kit. Available: <http://www.ti.com/tool/msp-exp430g2>. Accessed 8 September 2015.
6. Wikipedia, 2015. FreeRTOS. Available: <https://en.wikipedia.org/wiki/FreeRTOS>. Accessed 8 September 2015.
7. freeRTOS Real Time Engineers ltd., 2015. Available: <http://www.freertos.org/RTOS.html>. Accessed 8 September 2015.
8. Thao Ng.G.M, Nghia D.H, Phuc Ng.H, 2010. A PID backstepping controller for two-wheeled self-balancing robot. Proceeding of International Forum on Statigic Technology: 95-100.