A LOW-COST, REAL-TIME ALGORITHM FOR EMBEDDED DEVICES BASED ON FREERTOS KERNEL

Van-Khanh Nguyen[1]

[1]*Automation Department, College of Engineering Technology, Cantho University, Vietnam*
vankhanh@ctu.edu.vn

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, the 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 C code generated to build and run on embedded targets. 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 a famous ARM Cortex M4 core microcontroller STM32F407VTG. The experimental results show that algorithm designed operated well on embedded systems. The tracking position and rotation angle response are so good with low steady error (i.e. 0.01 [m] and less than 1.15 degrees, respectively) while stabilizing the two-wheeled at the upright. The real-time system designed is a low cost methodology and suitable for embedded system designers.

Key Words: *Open source, RTOS, embedded system, ARM Cortex and fuzzy PID controller.*

---***---

1. INTRODUCTION

Real-time embedded systems are widely used for various purposes such as aerospace, industrial and many other fields. Real-time character is required to meet certain deadlines at the right time. To achieve this purpose, real-time operating system (RTOS) is utilized in embedded system. There are a lot of available RTOSes such as freeRTOS, uCOS, CMSIS-RTOS, Salvo RTOS, ChibiOS/RT. Most of them are open source and admitted by many famous semiconductor companies like TI, ST, Atmel, etc. A lot of performance comparision among RTOSes have published. Hrushit Shah and et al published their studies about the comparison of three famous open source RTOSes like RT-Linux, freeRTOS and eCOS based on processors and requirements [1]. This research gives a useful technical document which figures out some important parameters used to choose a suitable RTOS for a specific application. In addition, Douglas P. B. Renaux compared the performance among RTOSes which were suited with CMSIS-RTOS standard on two classes of RTOS: commercial FOSS-Free and Open Source Software-RTOS [2]. These results show that FOSS RTOS has a reasonable time parameters although it is free of charge.

RTOS has also deployed in many applications on different studies. A research published by Douglass in ultilzing the CMSIS standard into the Internet of Things (IoTs) that appealed many researchers [3]. This study shown that CMSIS-RTOS is supported by many HAL drivers, middlewares for utilizing into IoTs technologies. This can reduce design time and bring product to market with effective cost. Another research by Jorge Cabrera-Gasmez [4] was utilized a real-time sailboat controller operated based on ARM, SAM3X8E microcontroller.

On the other hand, many applications have been tending to embedded hardware application in information technology field. In fact, a lot of courses about embedded system programming for beginners are taught in many universities using MSP430 LaunchPad Value Line Development Kit designed by Texas Instrument Corporations. It is an easy-to-use microcontroller development board based on low-power and low-cost MSP430G2x microcontroller [5]. In addition, Thanh M.T and et al developed a tracking Pioneer P3-DX robot used image processing algorithms combining with ARCOS, an embedded operating system, run on Pioneer P3-DX [6]. In this system, the camera integrated on the robot, captured images and then sent to PC to identify objects and sent command back to the robot for tracking objects. The results shown that the robot can track objects well. However, these real-time studies need a hardware platform which is often expensive to develop applications.

Consequently, this study presents a new method which employs an algorithm designed by Matlab embedded into a real-time system hardware with low power, low cost and easy to use. The hardware platform is designed based on 32-bit ARM architecture employed an open source real-time operating system, FreeRTOS, a reliable and stable operating system for research and industry.

2. ALGORITHM

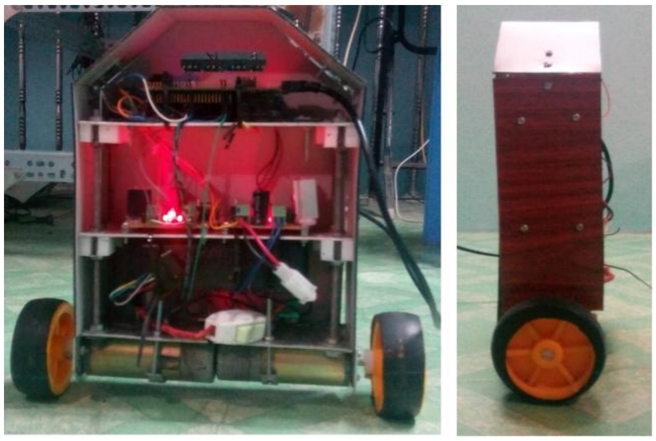
A fuzzy PID self-tunning controlled a two-wheeled self-balancing robot is nominated for implementing the proposed methodology. The body structure of robot is designed and shown in Fig. 1. For controlling robot, an control algorithm is developed as shown in Fig. 2. There are three major control loops. The first loop Fuzzy PD1 is to use a fuzzy PD controller to compute the tilt angle reference r_θ for Fuzzy

PD2 based on distance error. The second loop Fuzzy PD2 is also to use a fuzzy PD controller to calculate apart of output signal u_y based on $r_{ø}$. The third loop is to use a self-tuning PID controller to control rotation angle of robot by calculating u_x signal. The output u_R and u_L are figured out by equations (1) and (2), respectively as below [8].

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

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

where u_R, u_L are output signals controlled right and left motors of the robot.



a) b)

Fig -1: Backside and (b) behind of two-wheeled self-balancing robot, respectively.

3. IMPLEMENTATION

FreeRTOS is a powerful real-time operating system and supports a lot of architectures consisting of Intel, ARM, Atmel, PIC, etc. The objective of this work is to implement the real-time algorithm to control two-wheeled that can perform in real-time domain. Introduction about RTOS, hardware platform overview, how to implement this OS to a real-time hardware are described in the following.

3.1 Introduction real-time operating system

RTOS - Realtime Operating System is an operating system designed for developing real-time embedded applications, which process tasks as they come in with a small amount of delay time. This delay time is called deadline time. In real-time system, deadline time is an important parameter, so it must be considered when designing a system [7]. Recently, there are over 30 RTOSes which support a lot of microcontroller families involving very small memory footprint families.

There are two main advantages when designs an application with RTOS. First, designers do not spend too much time to study how interface with microcontroller peripheral resources because they are integrated by developers. In fact, developers always integrated almost supported microcontroller family and peripheral drivers as much as possible. Second, applications will be developed by task based. This designed method is very effective with complicated system and group working projects.
FreeRTOS is an RTOS designed by Real Time Engineer Ltd. Its real-time kernel is a small and simple C-language. Recently, freeRTOS has been distributing by GPL with optional exception, and it has been ported to 35 microcontroller families [6]. The scheduler of freeRTOS can be chosen between pre-emption or cooperation, which can be configured by using freertosconfig.h file in programming. When the scheduler is started, the task with the highest priority will be executed first. If there are more than one priority tasks, the scheduler will use round-robin algorithm to schedule the running of these tasks which are same priority.

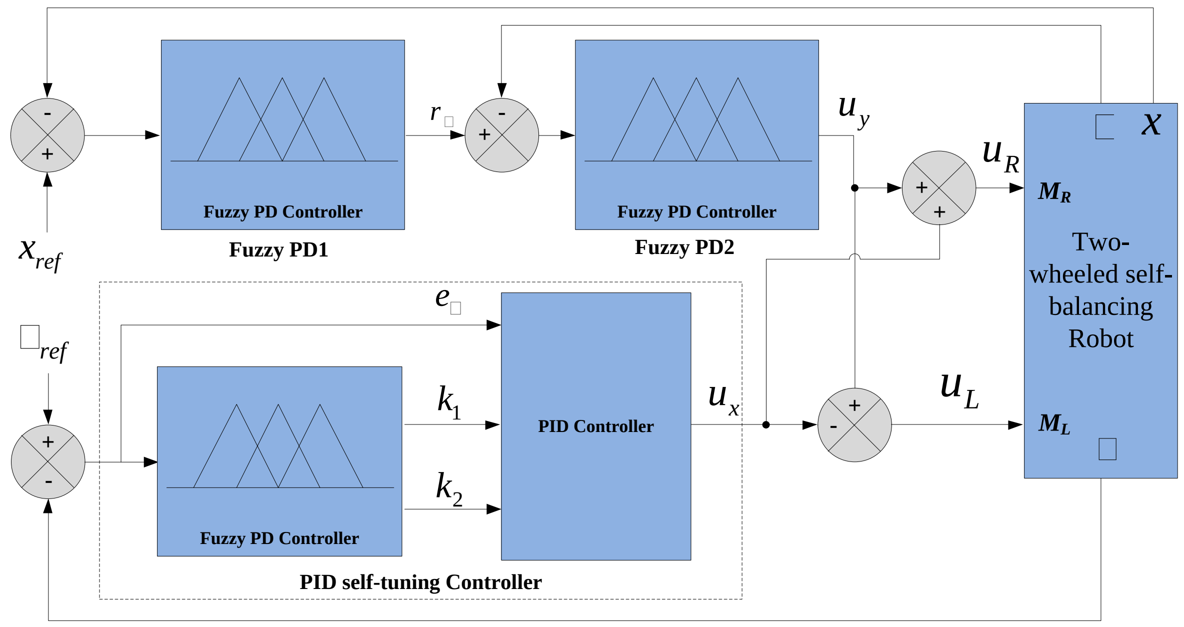


Fig -2: Control algorithm.

3.1 Hardware platform

The microcontroller designed in this study has used the famous STM32 family of 32-bit microcontroller based on the ARM Cortex M4 microprocessor of STMicroelectronics. The STM32F407VG is integrated floating point unit (FPU) with pulse frequency up to 168 MHz and equipped with real-time emulating and tracking. Additionally, it is also built in 12-bit analog-to-digital converter (ADC) with high speed, 7.2 MSPS in interleaved mode, 1 MB high speed flash memory, six-encoder which are suitable for controlling two-wheeled in real-time of low cost and low power consumption. The systematic hardware diagram is shown in Fig. 3.
The tilt angle of the robot is measured by MPU6050 module which is a six axes gyro and accelerometer MEMs sensor. Rotation angle and displacement are calculated based on the moving distance of left and right wheels which a rotatory encoder attached directly to the wheels. The speed of two motors is controlled by Pulse Width Modulation (PWM) technique, an effective method for controlling motor by changing the width of the pulse, via motor driver circuits.

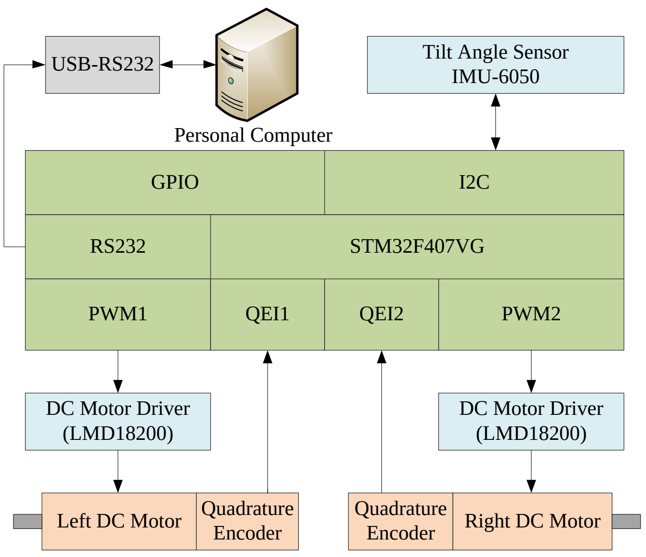


Fig -3: Systematic hardware diagram.

3.1 Two-wheeled Control Implementation

The control algorithm is designed by Matlab/Simulink adhered to three following steps:

- Step one: designs algorithm using Matlab/Simulink. Fig. 4 shows the structure of the Matlab/Simulink program to implement algorithm proposed in this study. It is seen from Fig. 4 that the algorithm designed consists of two kind of blocks. The first block is Input Block which receives data from inputs or others blocks to calculate the output of the block. The second block is Output Block which receives data from the output of Input Blocks to compute the output of the algorithm. If some blocks run independently with sample time, it will be designed in one block. One RTOS task is designed to run this kind of block when sample-time-dependent tasks were finished to reduce the execution time of the system. Some kind of this task can be involed updating task, data communication task.
- Step two: Defines input/output signals for simulink blocks which is generated into embedded C code. Fig. 5a is an example of defined input/output signals of the Fuzzy PD1 block. This block has three inputs and two outputs, so three input signals (TiltAngle, POSFeedback and POSReference) and two output signals (EnableSignal_B1 and TiltAngleError_B1) will be defined. The Storage class property of signals must be selected ExportedGlobal to generate signals to global C variable as represented in Fig. 5b. This will make easily in access signals in C code.
- Step three: Configures parameters of Matlab/Simulinks to build an algorithm to embedded C language code. In this step, three parameters must be configured. First, there are some choices in Solver options like: Type, Solver and Fixed-step size. It is noted that fixed-step size should be chosen 0.01 for this study; however, this value also depends on the specific application. Second, system target file, ert.tlc, defines the rules to generate embedded C code. Last but not least, the Generate code only option must be checked to tell the generator only generates C code.

Based on these steps, embedded C code is ready to generate for employing to RTOS. In this example, the generated embedded C code of input and output blocks are contained in three folders as shown in Fig. 6.

As mentioned in the previous section, an algorithm is designed into several blocks which are also generated into C code separately. To run these codes on embedded system, a freeRTOS program is utilized to combine these codes to execute algorithm on embedded system exactly. The C code of block is called by one freeRTOS task which communicates with others by the message. Based on Matlab/Simulink program designed, there are three kind of tasks. First, tasks run the code of input blocks which read all inputs, run the code and send results for waiting tasks. The template of this task is shown in Listing 1 below.

Listing 1.
```
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);
        }
}
```

Second, tasks execute the code blocks which stand between input and output blocks. This kind of task waits data from other tasks before running their code and sending results to waiting tasks. The template of this task is shown in Listing 2.

Listing 2.
```
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);
            }
    }
}
```

Third, the task calls the code of the output task which waited data from other tasks to run their code and update control signal to plant. There is only one output block, so freeRTOS program also has only one this kind of task. The template of this task is shown in Listing 3.

Listing 3.
```
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);
    }
}
```

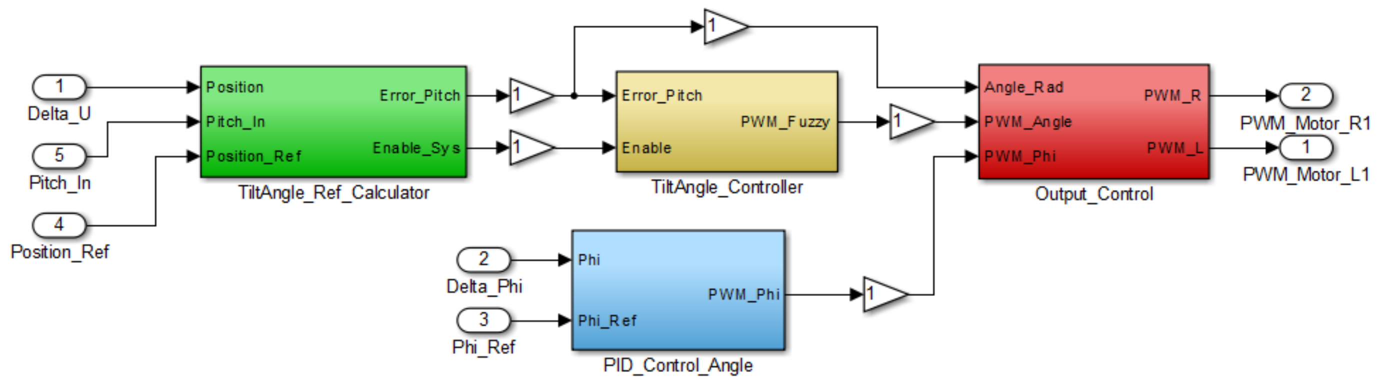
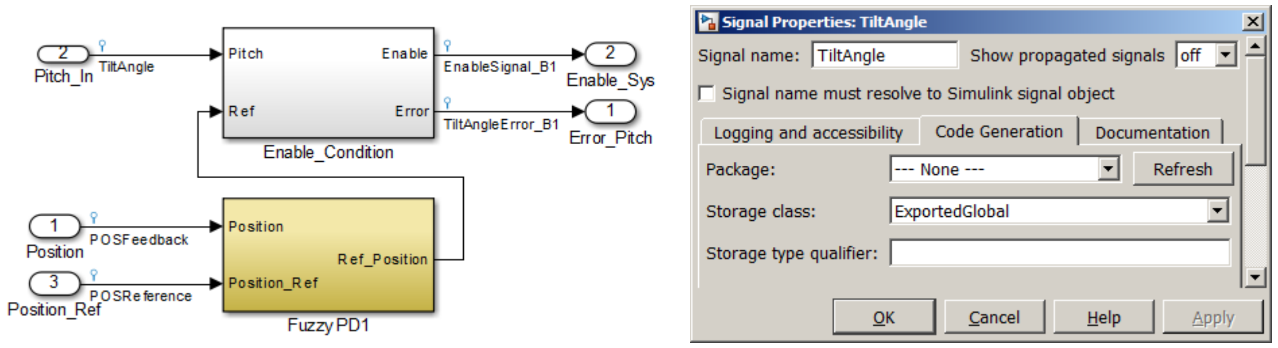


Fig -4: Complete Matlab/Simulink algorithm.



Signal Properties: TiltAngle

Signal name: TiltAngle Show propagated signals off

☐ Signal name must resolve to Simulink signal object

Package: --- None ---

Storage class: ExportedGlobal

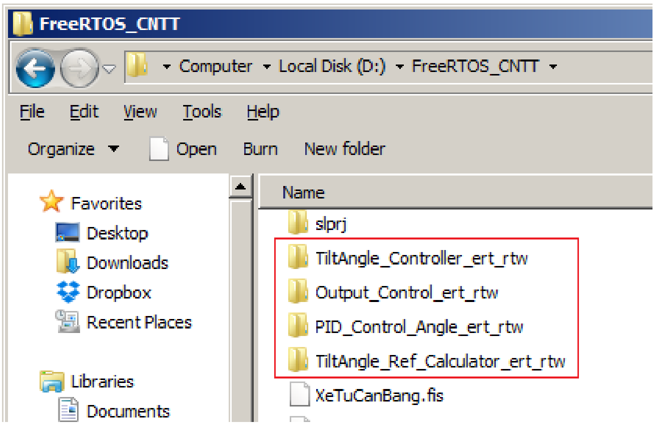
Storage type qualifier:

a) b)

Fig -5: (a) Defination of input/output signals; (b) Properties of signal.

Output task is also a periodic task which is executed regularly each sampling time. This time is created by call TaskDelayUntil() API function which can trigger task execution starts on precise time. Noted that values pass into this function must be equal with fixed-step value. In this case, the sample time of the system, 0.01, is ten mili-seconds or ten ticks of freeRTOS, so 10 is passed to TaskDelayUntil() function.

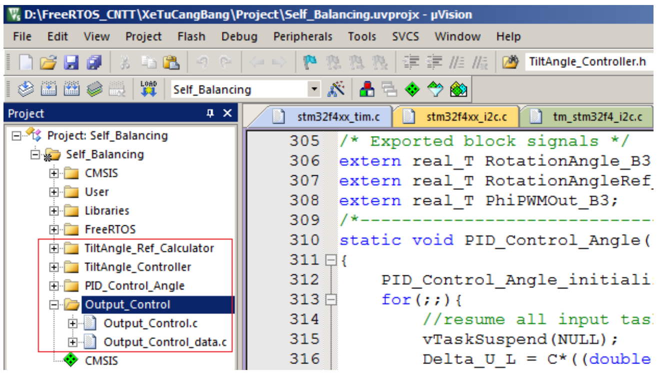
The code designed are integrated to a RealView KeilC for ARM project to build and run on an ARM Cotext M4 platform. The completed project is shown in Fig. 7.


FreeRTOS_CNTT

Computer ▸ Local Disk (D:) ▸ FreeRTOS_CNTT ▸

Name
- slprj
- TiltAngle_Controller_ert_rtw
- Output_Control_ert_rtw
- PID_Control_Angle_ert_rtw
- TiltAngle_Ref_Calculator_ert_rtw
- XeTuCanBang.fis

Fig -6: Output folder of embedded C code.


D:\FreeRTOS_CNTT\XeTuCangBang\Project\Self_Balancing.uvprojx - µVision

Project: Self_Balancing
- Self_Balancing
  - CMSIS
  - User
  - Libraries
  - FreeRTOS
  - TiltAngle_Ref_Calculator
  - TiltAngle_Controller
  - PID_Control_Angle
  - Output_Control
    - Output_Control.c
    - Output_Control_data.c
  - CMSIS

```
/* Exported block signals */
extern real_T RotationAngle_B3
extern real_T RotationAngleRef_
extern real_T PhiPWMOut_B3;
/*-------------------------------
static void PID_Control_Angle(
{
    PID_Control_Angle_initiali
    for(;;){
        //resume all input tas
        vTaskSuspend(NULL);
        Delta_U_L = C*((double
```

Fig -7: The final realView KeilC project.

4. EXPERIMENT RESULTS

4.1 Sampling and execution time of algorithm

The sampling time is measured to validate that it is generated exactly on the embedded systems. A 32-bit timer of ARM Cortex is used to measure this time. The timer is restarted when the code begins running with a counter value cleared to zero. The counter value is read at the next execution of the code. This value is transformed into time in millisecond unit as illustrated in Fig. 8. It can be seen that the sampling time is generated exactly with error is as small as 0.001ms.

Not only sampling time is measured, the execution time of the algorithm is also calculated to show how the power of this microcontroller. Using the same measurement strategy, the execution time of algorithm as shown in Fig. 9. From the

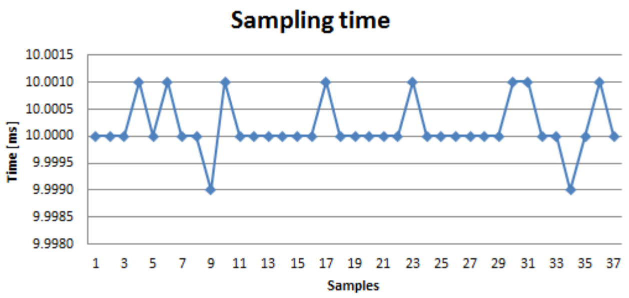


Fig -8: Sampling time of embedded system.

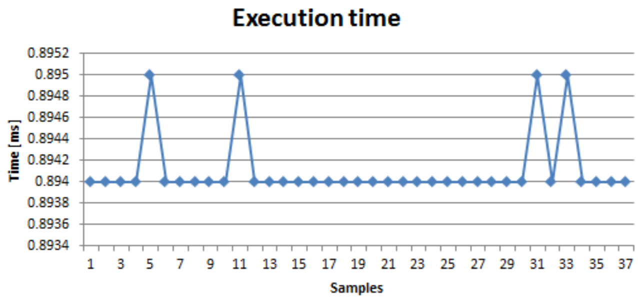


Fig -9: Execution time of algorithm.

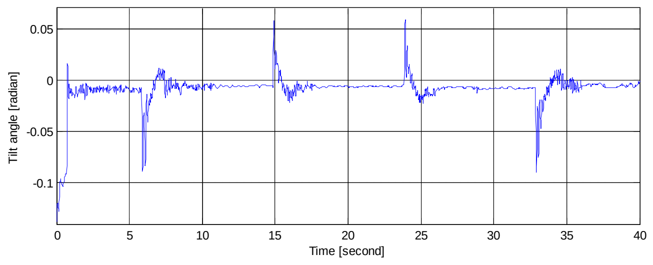
figure can be seen that the whole time to execute algorithm is less than one millisecond.

4.2 Responding of robot

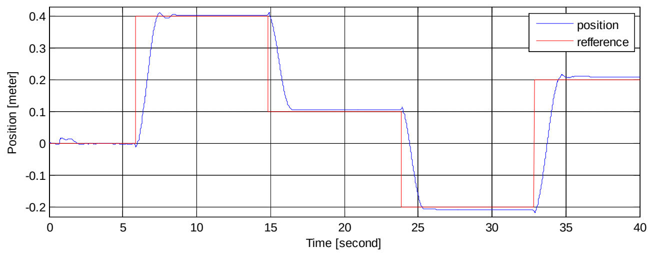
This section validated that the algorithm designed is successfully implemented in embedded systems. In this demonstration, the robot is controlled to move to specific position includes 0.0, 0.4, 0.1, -0.2 and 0.2 [m] sequentially while try keeping rotation angle at 0 [rad] and balancing robot at the up-right equilibrium. Fig. 10 shows that the fuzzy PID self-tuning algorithm operated well on an embedded system with the proposed method. The robot can track to input reference signals. In detail, Fig. 10.b shows that the robot can continuously follow the position reference signal with 0.01[m], 1.5 [s], 0.015 [m] maximum steady error, rising time and overshot, respectively. The tilt angle steady error when the robot is stable at position set-point is not exceeded 1 degree, and rotation angle of robot changed continuously but limit in 1.15 degrees. Fig. 10d and Fig. 10e show the reaction of u_R and u_L when the robot is tracking reference signals.

5. CONCLUSIONS

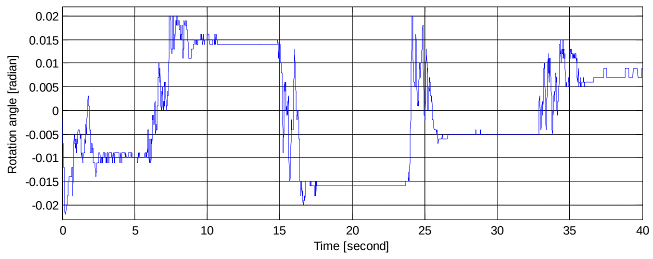
A new method to implement real-time algorithm using a well-known open sources freeRTOS in embedded system is proposed in this study. This method facilitates researchers to employ algorithms designed in Matlab, a powerful designed tools, to embedded system by using FreeRTOS. The experimental results show that the proposed method which is implemented on two-wheeled robot archives fast response with maxima delay time 1.5 seconds. The tilt angle steady state error is less than 1 degree. The sampling time error is as small as 0.001ms. These achievements are as good as other works which used real-time hardware. This proposed method is a low cost solution for embedded system and suitable for fresh embedded designers.



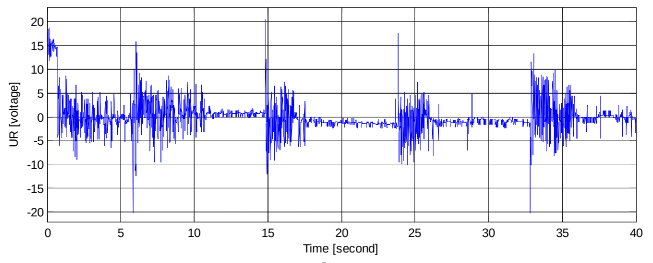
a)



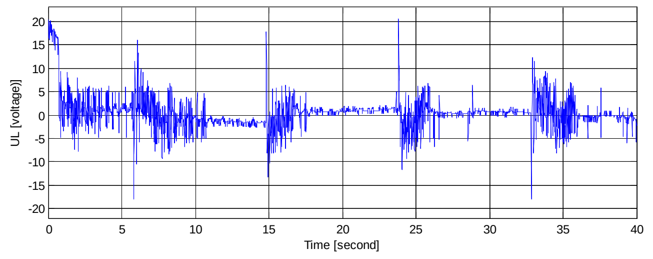
b)



c)



d)



e)

ACKNOWLEDGEMENT

The author would like to express sincere thanks to Cantho University for funding during research and also thanks for Trong-Hieu Tran contributed in revising paper.

REFERENCES

[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]. 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 Stategic Technology: 95-100.

BIOGRAPHIES

Van-Khanh Nguyen received a BE degree from Can Tho University, Vietnam, a MSc degree from Ho Chi Minh City University of Technology, Vietnam, both in Electronic Engineering in 2005, 2014, respectively. He is currently is a lecturer of the College of Engineering, Can Tho University, Vietnam