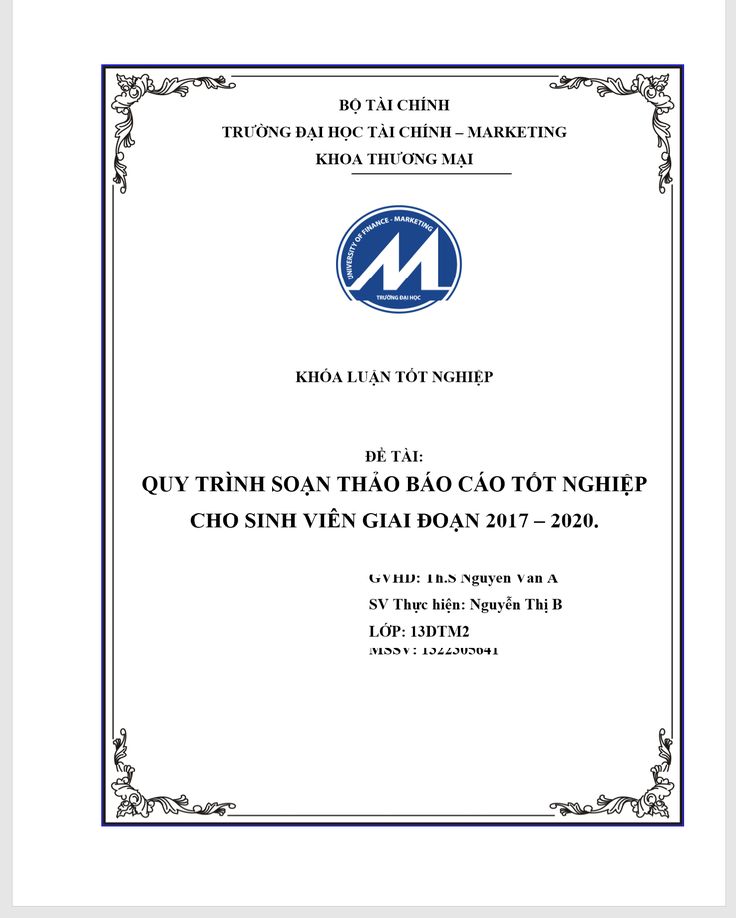
­­**HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG**



**CƠ SỞ TẠI THÀNH PHỐ HỒ CHÍ MINH**

**KHOA KỸ THUẬT ĐIỆN TỬ II**



**BÁO CÁO MÔN HỌC**

**NIÊN KHÓA: 2020 – 2025**

**Môn học: Đồ án hệ thống nhúng**

**ĐỀ TÀI: CÂN BẰNG BÓNG SỬ DỤNG THUẬT TOÁN ĐIỀU KHIỂN PID**

Giảng viên hướng dẫn : **GV.TS Nguyễn Trọng Kiên**

Sinh viên thực hiên :

* **Mạnh Trọng Hải (N20DCDT004)**
* **Trần Trọng Phúc (N20DCDT0018)**
* **Võ Thị Ngân (N20DCDT0014)**

Lớp : **D20CQDT01-N**

**MỤC LỤC**

[**MỞ ĐẦU** 4](#_Toc168657272)

[**CHƯƠNG 1: CƠ SỞ LÝ THUYẾT VỀ PHẦN CỨNG** 6](#_Toc168657273)

[**1. Cấu Hình ESP32** 6](#_Toc168657274)

[1.1 ESP32 6](#_Toc168657275)

[1.2 ESP32 Peripherals 6](#_Toc168657276)

[**2. Cấu hình Servo** 7](#_Toc168657277)

[**CHƯƠNG 2: CƠ SỞ LÝ THUYẾT VỀ PHẦN MỀM** 9](#_Toc168657278)

[**1. Bộ điều khiển PID** 9](#_Toc168657279)

[1.1 Bộ điều khiển PID là gì ? 9](#_Toc168657280)

[1.2 Nguyên lý hoạt động của bộ điều khiển PID 10](#_Toc168657281)

[1.3 Phân loại bộ điều khiển PID 10](#_Toc168657282)

[**2. Thư viện Opencv** 11](#_Toc168657283)

[1.1 Giới Thiệu 11](#_Toc168657284)

[1.2 Các ứng dụng OpenCV 11](#_Toc168657285)

[1.3 Chức năng OpenCV 11](#_Toc168657286)

[1.4 Thư viện và cài đặt 11](#_Toc168657287)

[**3.**  **FreeRTOS** 12](#_Toc168657288)

[3.1 Khái niệm hệ điều hành thời gian thực? 12](#_Toc168657289)

[3.2 RTOS Task 13](#_Toc168657290)

[**CHƯƠNG 3 Đánh giá kết quả** 16](#_Toc168657291)

[**3.1 Kết quả** 16](#_Toc168657292)

[4 19](#_Toc168657293)

**MỞ ĐẦU**

**1. Đặt Vấn Đề**

Ngày nay, kỹ thuật tự động hóa điều khiển được áp dụng rỗng rãi trong tất cả các ngành công nghiệp. Một ứng dụng tiêu biểu của kĩ thuật này là có thể dùng máy tính trực tiếp điều khiển đến từng thiết bị, từng dây chuyền sản xuất cũng như toàn bộ hệ thống sản xuất, một sai sót nhỏ của bộ điều khiển trong quá trình vận hành có thể gây ảnh hưởng lớn đến chất lượng sản phẩm hàng loạt. Với sự đòi hỏi đến mức độ hoàn hảo của thực tế, các thuật toán điều khiển trong quá trình tính toán không những phải đảm bảo độ chính xác của kết quả mà còn phải đảm bảo tính kịp thời của kết quả được đưa ra. Mặt khác, đối với các hệ thống lớn và phức tạp, thực hiện nhiều chức năng, quá trình phân tích và thiết kế firmware hay nâng cấp một firmware hiện có chiếm khá nhiều thời gian của các kỹ sư lập trình. Sự ra đời của hệ điều hành thời gian thực cho phép các kỹ sư lập trình, các nhà phát triển hệ thống có một công cụ đắc lực để giải quyết vấn đề trên.

**2. Mục Đích Nghiên Cứu Đề Tài**

Trong bối cảnh hiện nay, việc tối ưu hóa hiệu suất và giảm chi phí sản xuất là mục tiêu hàng đầu của nhiều doanh nghiệp. Hệ thống nhúng có thể cung cấp các giải pháp hiệu quả và linh hoạt, giúp tự động hóa các quy trình và cải thiện chất lượng sản phẩm. Đề tài này được chọn nhằm khám phá khả năng ứng dụng của hệ thống nhúng trong việc giải quyết một vấn đề cụ thể trong lĩnh vực Embedded.

Đi sâu vào nghiên cứu các dịch vụ của FreeRTOS, nội dung và chức

năng các file mã nguồn, hoạt động của hệ điều hành và đưa ra các đặc tả về

thời gian của hệ điều hành này trên vi điều khiển

**3. Mục Tiêu Của Đồ Án**

Mục tiêu của đồ án này là thiết kế và phát triển một hệ thống nhúng có khả năng cân bằng bóng và sử dụng module AI để dự đoán. Đồ án sẽ tập trung vào các mục tiêu cụ thể sau:

* Nghiên cứu và lựa chọn phần cứng và phần mềm phù hợp.
* Thiết kế và triển khai hệ thống.
* Kiểm tra và đánh giá hiệu suất của hệ thống trong môi trường thực tế.

**4. Nội Dung Nghiên Cứu**

**Nội dung 1**: Tìm hiểu và lựa chọn các giải pháp thiết kế sao cho phù hợp nhất.

**Nội dung 2**: Thu thập tài liệu về các cảm biến, module, bộ vi xử lý.

**Nội dung 3**: Thiết kế, lập trình cho hệ thống điều khiển, chạy thử nghiệm mô hình.

**Nội dung 4**: Đánh giá kết quả thực hiện.

**5. Phạm Vi Nghiên Cứu**

Phạm vi của đề tài bao gồm việc phát triển một hệ thống nhúng hoàn chỉnh từ khâu thiết kế, lập trình đến thử nghiệm. Tuy nhiên, do hạn chế về thời gian và nguồn lực, đề tài sẽ chỉ tập trung vào một số chức năng chính.

**6. Cấu Trúc Của Đồ Án**

Phần báo cáo gồm có 4 chương:

**Chương 1**: Cơ sở lý thuyết phần cứng.

**Chương 2**: Cơ sở lý thuyết phần mềm.

**Chương 3**: Đánh giá kết quả.

**CHƯƠNG 1: CƠ SỞ LÝ THUYẾT VỀ PHẦN CỨNG**

## **1. Cấu Hình ESP32**

### 1.1 ESP32

**ESP32** là một module **Wi-Fi** với khả năng kết nối Internet và được tích hợp sẵn trên một số board nhúng như **NodeMCU**. Nó có thể hoạt động như một điểm truy cập (access point), một client kết nối đến một điểm truy cập khác, hoặc cả hai đều được. ESP32 được sử dụng rộng rãi trong các ứng dụng **IoT** (Internet of Things) như cảm biến thông minh, hệ thống kiểm soát thiết bị, hoặc các ứng dụng điều khiển từ xa.

ESP32 là một nền tảng IoT mã nguồn mở, được phát triển bởi một nhóm kỹ sư tại Trung Quốc. Nền tảng này được xây dựng trên ESP32, một vi điều khiển **Wi-Fi SoC (System on a Chip)** được sản xuất bởi **Espressif Systems**. NodeMCU 32S cung cấp một bộ SDK để lập trình cho ESP32 bằng ngôn ngữ **Lua** hoặc **C++.** Với các tính năng như **Wi-Fi, GPIO, ADC, I2C, SPI, PWM** và một số tính năng khác..

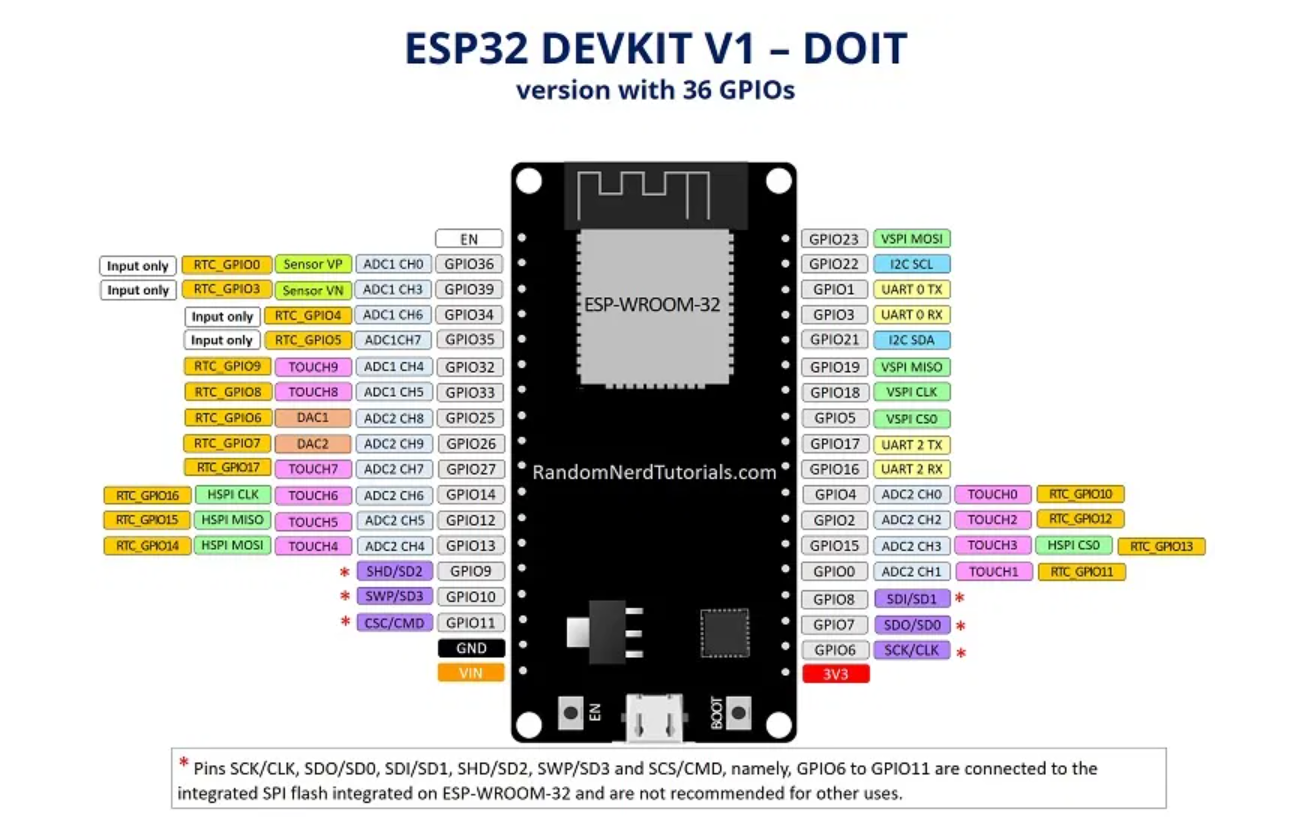
### 1.2 ESP32 Peripherals

Các ngoại vi (peripheral) của ESP32 gồm có:

* 18 Analog-to-Digital Converter (ADC) channels
* 3 SPI interfaces
* 3 UART interfaces
* 2 I2C interfaces
* 16 PWM output channels
* 2 Digital-to-Analog Converters (DAC)
* 2 I2S interfaces
* 10 Capacitive sensing GPIOs

Chức năng ADC (bộ chuyển đổi tương tự sang kỹ thuật số) và DAC (bộ chuyển đổi kỹ thuật số sang tương tự) được gán cho các chân tĩnh cụ thể. Tuy nhiên, bạn có thể quyết định chân nào là UART, I2C, SPI, PWM, v.v. – chỉ cần gán chúng bằng code. Điều này có được là nhờ chức năng ghép kênh của chip ESP32.

Mặc dù bạn có thể định nghĩa các thuộc tính của chân trên phần mềm, có những chân được gán theo mặc định như trong hình sau (đây là một ví dụ cho board mạch [ESP32 DEVKIT V1 DOIT](https://makeradvisor.com/tools/esp32-dev-board-wi-fi-bluetooth/) 36 chân – vị trí chân có thể thay đổi tùy thuộc vào nhà sản xuất).



**2. Cấu hình Servo**

Động cơ RC Servo MG 180 độ là loại servo có mô men xoắn lớn, chạy mượt mà, êm, phù hợp cho những mô hình điều khiển có tải trọng lớn như: mô hình máy bay, trực thăng, thuyền, cánh tay robot, robot nhện,.... Động cơ RC servo MG 180 độ sử dụng chất liệu nhựa có độ bền cao, có bánh răng bằng đồng giúp cho động cơ hoạt động chính xác, đạt độ bền cao. Tương thích hầu hết các kết nối chuẩn Futaba, Hitec, Sanwa, GWS etc...

Thông số kỹ thuật:

- Momen làm việc : 11kg/cm (tại điện áp 6V) , 9.4kg/cm (tại điện áp 4.8V)

- Đây là bản nâng cấp từ servo MG995 về tốc độ, lực kéo và độ chính xác

- Phù hợp với máy bay cánh quạt loại 50 -90 methanol và máy bay cánh cố định xăng 26cc-50cc

- So với MG946R, MG996R nhanh hơn, nhưng hơi nhỏ hơn.

- Tốc độ xoay: 0.17 giây / 60 độ (4.8 v) 0.14 giây / 60 độ (6 v)

- Điện áp làm việc: 4.8-7.2V

- Nhiệt độ hoạt động: -30 ℃ ~ 60 ℃

- Chiều dai dây: 30cm, dây nâu đỏ là 2 dây nguồn, dây vàng là dây tín hiệu

- Vật liệu bánh răng: Kim loại

Việc điều khiển servo thường yêu cầu xung với thời gian khoảng 20ms, xung điều khiển góc trong phạm vi 0.5 – 2.5ms, với tổng thời gian là 2ms.

- 0.5ms -> 0 độ

- 1.0ms – > 45 độ

- 1.5ms -> 90 độ

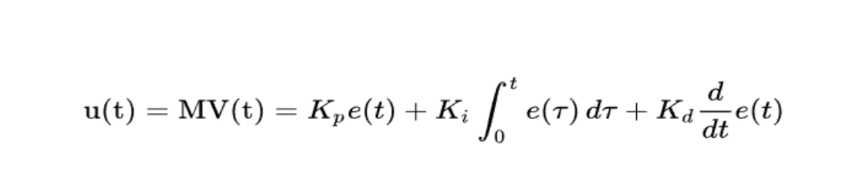
- 2.0ms -> 135 độ

- 2.5ms -> 180 độ

**CHƯƠNG 2: CƠ SỞ LÝ THUYẾT VỀ PHẦN MỀM**

**1. Bộ điều khiển PID**

### 1.1 Bộ điều khiển PID là gì ?



Khâu tỉ lệ, tích phân, vi phân được cộng lại với nhau để tính toán đầu ra của bộ điều khiển PID. Định nghĩa rằng u(t) là đầu ra của bộ điều khiển, biểu thức cuối cùng của giải thuật PID là:

**Độ lợi tỉ lệ: Kp**

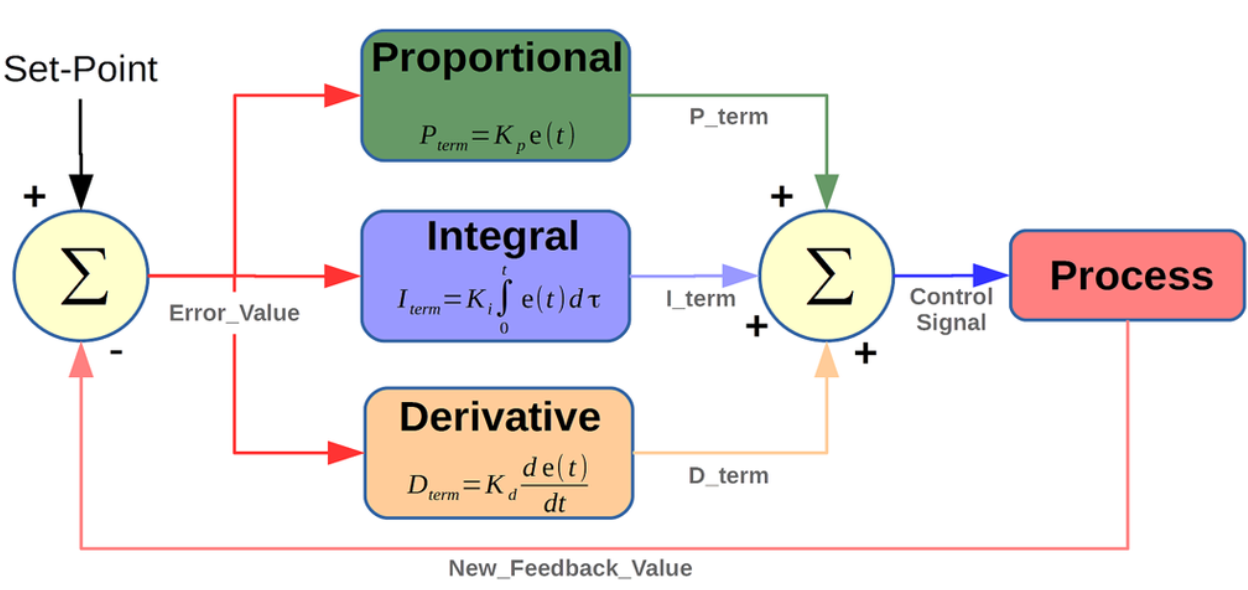
Giá trị càng lớn thì đáp ứng càng nhanh do đó sai số càng lớn, bù khâu tỉ lệ càng lớn. Một giá trị độ lợi tỉ lệ quá lớn sẽ dẫn đến quá trình mất ổn định và dao động.

**Độ lợi tích phân: Ki**

Giá trị càng lớn kéo theo sai số ổn định bị khử càng nhanh. Đổi lại là độ vọt lố càng lớn: bất kỳ sai số âm nào được tích phân trong suốt đáp ứng quá độ phải được triệt tiêu tích phân bằng sai số dương trước khi tiến tới trạng thái ổn định.

**Độ lợi vi phân: Kd**

Giá trị càng lớn càng giảm độ vọt lố, nhưng lại làm chậm đáp ứng quá độ và có thể dẫn đến mất ổn định do khuếch đại nhiễu tín hiệu trong phép vi phân sai số.



### 1.2 Nguyên lý hoạt động của bộ điều khiển PID

Bộ điều khiển **PID** được sử dụng để kiểm soát nhiệt độ, áp suất, lưu lượng và các thông số quy trình khác. Để giúp tự động điều chỉnh khi có sự thay đổi trong hệ thống, bộ điều khiển PID kết hợp kiểm soát tỉ lệ (**P**) với điều chỉnh tích phân (**I**) và đạo hàm (**D**):

**P (Proportional):** là phương pháp điều chỉnh tỉ lệ, giúp tạo ra tín hiệu điều chỉnh tỉ lệ với sai lệch đầu vào theo thời gian lấy mẫu.

**I (Integral):** là tích phân của sai lệch theo thời gian lấy mẫu. Điều khiển tích phân là phương pháp điều chỉnh để tạo ra các tín hiệu điều chỉnh sao cho độ sai lệch giảm về 0. Từ đó cho ta biết tổng sai số tức thời theo thời gian hay sai số tích lũy trong quá khứ. Khi thời gian càng nhỏ thể hiện tác động điều chỉnh tích phân càng mạnh, tương ứng với độ lệch càng nhỏ.

**D (Derivative):** là vi phân của sai lệch. Điều khiển vi phân tạo ra tín hiệu điều chỉnh sao cho tỉ lệ với tốc độ thay đổi sai lệch đầu vào. Thời gian càng lớn thì phạm vi điều chỉnh vi phân càng mạnh, tương ứng với bộ điều chỉnh đáp ứng với thay đổi đầu vào càng nhanh.

### 1.3 Phân loại bộ điều khiển PID

PID được chia làm 4 loại chính sau:

- **P: (Proportional Controller)** gọi là Bộ điều khiển tỉ lệ;

- **PI: (Proportinal and Integral Controller)** gọi là Bộ điều khiển tỉ lệ và tích phân;

- **PD: (Proportional and Derivative Controller)** gọi là Bộ điều khiển đạo hàm;

- **PID: (Proportional, Integral and Derivative Controller)** gọi là Bộ điều khiển tỉ lệ, tích phân, đạo hàm.

## **2. Thư viện Opencv**

### 1.1 Giới Thiệu

OpenCV là một thư viện mã nguồn mở hàng đầu cho thị giác máy tính (computer vision), xử lý ảnh và máy học, và các tính năng tăng tốc GPU trong hoạt động thời gian thực.

OpenCV được phát hành theo giấy phép BSD, do đó nó hoàn toàn miễn phí cho cả học thuật và thương mại. Nó có các interface C++, C, Python, Java và hỗ trợ Windows, Linux, Mac OS, iOS và Android. OpenCV được thiết kế để tính toán hiệu quả và với sự tập trung nhiều vào các ứng dụng thời gian thực. Được viết bằng tối ưu hóa C/C++, thư viện có thể tận dụng lợi thế của xử lý đa lõi. Được sử dụng trên khắp thế giới, OpenCV có cộng đồng hơn 47 nghìn người dùng và số lượng download vượt quá 6 triệu lần. Phạm vi sử dụng từ nghệ thuật tương tác, cho đến lĩnh vực khai thác mỏ, bản đồ trên web hoặc công nghệ robot.

### 1.2 Các ứng dụng OpenCV

OpenCV đang được sử dụng rộng rãi trong các ứng dụng bao gồm:

Hình ảnh street view Kiểm tra và giám sát tự động Robot và xe hơi tự lái Phân tích hình ảnh y tế Tìm kiếm và phục hồi hình ảnh/video Phim - cấu trúc 3D từ chuyển động Nghệ thuật sắp đặt tương tác

### 1.3 Chức năng OpenCV

Image/video I/O, xử lý, hiển thị (core, imgproc, highgui) Phát hiện các vật thể (objdetect, features2d, nonfree) Geometry-based monocular or stereo computer vision (calib3d, stitching, videostab) Computational photography (photo, video, superres) Machine learning & clustering (ml, flann) CUDA acceleration (gpu)

### 1.4 Thư viện và cài đặt

Cài đặt python

Hầu như các hệ điều hành đều hỗ trợ Python, bạn có thể cài đặt online, hoặc offline. Chúng ta sẽ thấy các cách phổ biến sau:

* Trên Debian (Ubuntu, Linux Mint, ...) chúng ta sử dụng: apt-get install python
* Trên RedHat (CentOS, RedHat, Asianux, ...): yum install python

Cài đặt thư viện.

Chúng ta sẽ làm việc thông qua nhiều ví dụ Python ở đây. Bắt đầu với ràng buộc Python của OpenCV . Bạn sẽ cần hai thư viện chính, với một tùy chọn thứ ba: python-OpenCV, Numpy, và Matplotlib.

* apt-get install python3-pip
* sau đó chạy
* pip3 install numpy or apt-get install python3-numpy

Tiếp theo chạy lệnh: pip3 install matplotlib or apt-get install python3-matplotlib

Và cuối cùng là: apt-get install python-OpenCV

## **3. FreeRTOS**

### 3.1 Khái niệm hệ điều hành thời gian thực?

Hệ điều hành thời gian thực (Real Time Operating System – RTOS) là một hệ điều hành đa nhiệm hướng đến các ứng dụng thời gian thực. Cũng có thể hiểu, RTOS là một chương trình có nhiệm vụ lập lịch thực hiện cho các tác vụ để đảm bảo tính thời gian thực, quản lý tài nguyên, cung cấp một sự thiết lập phù hợp cho các mã chương trình ứng dụng. Một nhân của hệ điều hành thời gian thực phải đảm nhiệm được các nhiệm vụ chính sau [2]:

- Xử lý ngắt: Lưu trữ ngữ cảnh chương trình tại thời điểm xuất hiện ngắt, nhận dạng và lựa chọn đúng trình phục vụ ngắt.

- Điều khiển quá trình:

+ Tạo và kết thúc các quá trình/ tác vụ

+ Lập lịch và điều phối hoạt động của hệ thống

+ Định thời

– Quản lý và cấp phát tài nguyên:

+ Giải quyết tranh chấp tài nguyên giữa các tác vụ

+ Cấp phát tài nguyên cho các tác vụ

FreeRTOS được thiết kế phù hợp cho nhiều hệ nhúng nhỏ gọn vì nó chỉ triển khai rất ít các chức năng như: cơ chế quản lý bộ nhớ và tác vụ cơ bản, các hàm API quan trọng cho cơ chế đồng bộ. Nó không cung cấp sẵn các giao tiếp mạng, drivers, hay hệ thống quản lý tệp (file system) như những hệ điều hành nhúng cao cấp khác. Tuy vậy, FreeRTOS có nhiều ưu điểm, hỗ trợ nhiều kiến trúc vi điều khiển khác nhau, kích thước nhỏ gọn (4.3 Kbytes sau khi biên dịch trên Arduino), được viết bằng ngôn ngữ C và có thể sử dụng, phát triển với nhiều trình biên dịch C khác nhau (GCC, OpenWatcom, Keil, IAR, Eclipse, …), cho phép không giới hạn các tác vụ chạy đồng thời, không hạn chế quyền ưu tiên thực thi, khả năng khai thác phần cứng. Ngoài ra, nó cũng cho phép triển khai các cơ chế điều độ giữa các tiến trình như: queues, counting semaphore, mutexes.

### 3.2 RTOS Task

Một task là một chương trình, chương trình này chạy liên tục trong vòng lặp vô tận và không bao giờ dừng lại. Với mỗi task thì có niềm tin duy nhất là chỉ mình nó đang chạy và có thể sử dụng hết nguồn tài nguyên sẵn có của bộ xử lý (mặc dù là thực tế thì nó vẫn phải chia sẻ nguồn tài nguyên này với các task khác).

Một chương trình thường sẽ có nhiều task khác nhau. Ví dụ như máy bán đồ uống tự động sẽ có các thành task sau:

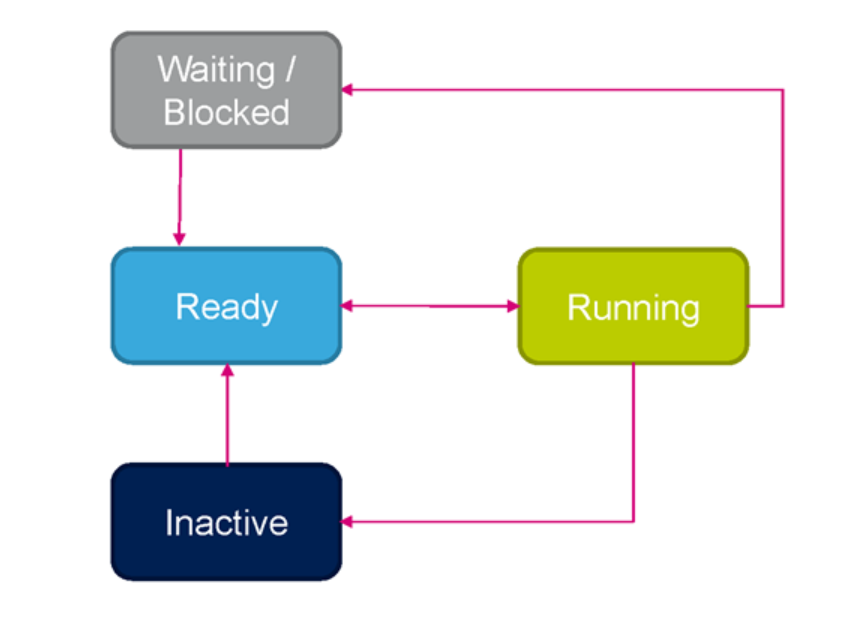
* Task quản lý việc lựa chọn của người dùng
* Task để kiểm tra đúng số tiền người dùng đã trả
* Task để điều khiển động cơ/ cơ cấu cung cấp nước uống.

Kernel sẽ quản lý việc chuyển đổi giữa các task, nó sẽ lưu lại ngữ cảnh của task sắp bị hủy và khôi phục lại ngữ cảnh của task tiếp theo bằng cách:

* Kiểm tra thời gian thực thi đã được định nghĩa trước (time slice được tạo ra bởi ngắt systick)
* Khi có các sự kiện unblocking một task có quyền cao hơn xảy ra (signal, queue, semaphore,…)
* Khi task gọi hàm Yield() để ép Kernel chuyển sang các task khác mà không phải chờ cho hết time slice

Khi khởi động thì kernel sẽ tạo ra một task mặc định gọi là Idle Task

**Task state**



**Ready:**Task đã sẵn sàng để có thể thực thi nhưng chưa được thực thi do có các task khác với độ ưu tiên ngang bằng hoặc hơn đang chạy.

**Running:** khi task thực sự đang chạy

**Blocked(Waiting):**Task đang đợi một event tạm hoặc event từ bên ngoài

**Suspended:**Task không khả dụng để lên lịch (scheduling)

# **CHƯƠNG 3 Đánh giá kết quả**

## **3.1 Kết quả**

* Hoàn thành yêu cầu cân bằng bóng
* Tập trung vào việc tối ưu hóa hoạt động để đạt được hiệu suất tốt nhất và tiết kiệm năng lượng
* Hoàn thành được 80% yêu cầu được đề ra.
* Phần động cơ điều khiển đôi lúc vẫn gặp trục trặc.

# 4