

*INSTITUTO POLITECNICO NACIONAL*

*Unidad Profesional Interdisciplinaria en Ingeniería y Tecnologías Avanzadas*

*Proyecto de Sistemas Operativos en Tiempo Real*

***Control de Péndulo Invertido utilizando FreeRTOS***

*Presentan:*

*José Arturo Clemente Pérez*

*Arturo Valdez Ochoa*

*Maestro: Lamberto Maza Casas*

*Grupo: 3MV10*

*Fecha: 07/12/2018*

# Introducción

Un sistema en tiempo real ha sido descrito como aquel que “controla un entorno recibiendo datos, procesándolos y devolviendo los resultados lo suficientemente rápido para afectar el entorno en ese momento”. El término “tiempo real” se utiliza en el control de procesos y sistemas empresariales significa “sin demora significativa”.

El software en tiempo real puede usar:

* lenguajes de programación síncronos
* sistemas operativos en tiempo real (**RTOS**)

**RTOS en Arduino UNO**

Con Arduino hacemos una programación en tiempo real recibiendo continuamente datos de los sensores o de los puertos de comunicación, analizandolos y respondiendo al entorno mediante los actuadores o las comunicaciones en un tiempo muy rápido.

Los sistemas operativos en tiempo real (**RTOS**) son sistemas que responden a la entrada inmediatamente. Se utilizan para tareas tales como la navegación, en las que el ordenador debe reaccionar a un flujo constante de nueva información sin interrupción. La mayoría de los sistemas operativos de uso general no son en tiempo real porque pueden tardar unos segundos, o incluso minutos, en reaccionar.

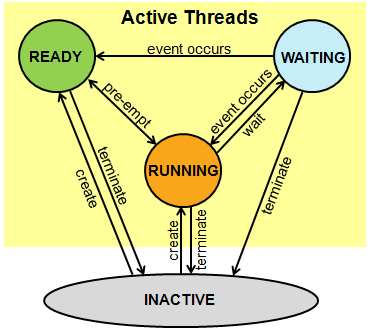


Ilustración 1 Implementación de multitarea

**FreeRTOS en arduino**

FreeRTOS proporciona un kernell mínimo para realizar aplicaciones en tiempo real, está diseñado para trabajar también con microcontroladores y es por ello que se puede implementar en la familia de los AVR´s, aunque la tarjeta de desarrollo Arduino UNO no puede soportar un sistema operativo en tiempo real completo, puede trabajar con algunas funciones que ofrece FreeRTOS, por ejemplo, se puede trabajar con tareas “Task” y la mayoría de las funciones que esto implica (eliminar tareas o crearlas en plena ejecución, cambiar de prioridad las tareas), también se puede hacer uso de semáforos que logren la mutua exclusión de las tareas, evitando el desordenado acceso a la región crítica y también evitando la inanición.

# Desarrollo mecánico y electrónico

En el siguiente proyecto se van a utilizar algunas características de FreeRTOS para controlar un péndulo invertido, el material utilizado fue:

* Motor Namiki 12Vdc 5Kg\*cm
* Encoder por cuadratura
* Fuente de 12Vdc
* Eje de aluminio 5mm de diámetro
* Varilla de aluminio de 9.8mm de diámetro a 40cm de largo
* Acopladores mecánicos
* Tabla y soporte de motor
* Tarjeta de desarrollo ArduinoUNO
* Tarjeta controladora de motores L293D

Para construir el mecanismo, se manufacturaron las piezas como los acopladores mecánicos y el resto de los componentes se compraron, a continuación se muestra una imagen del diseño final.



Ilustración 2 Imagen del péndulo invertido construido

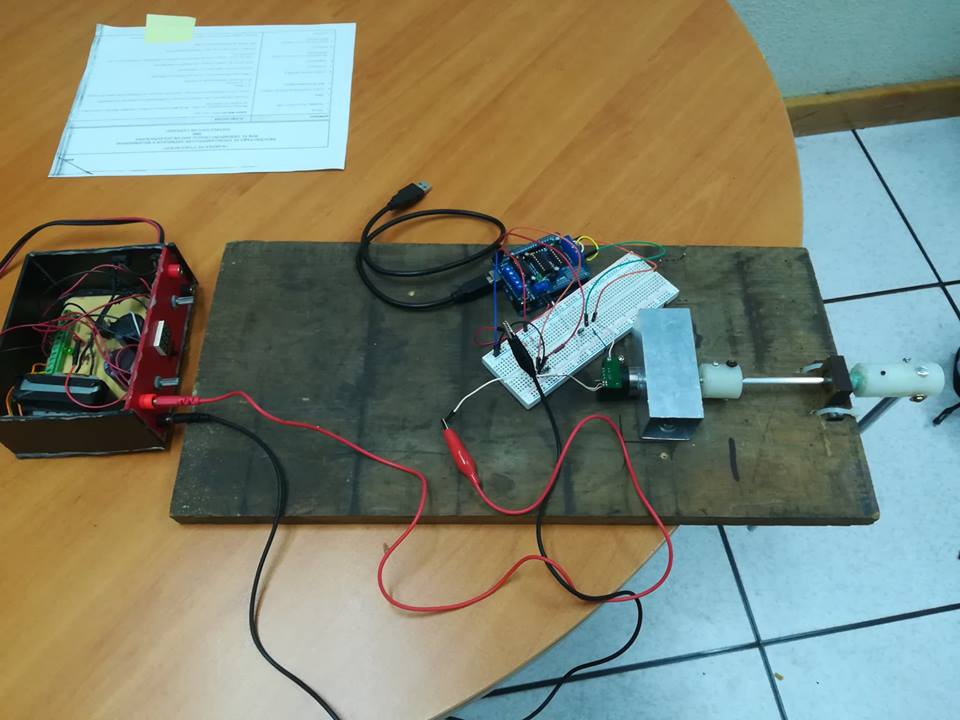


Ilustración 3 visualización de la electrónica

# Programación de Arduino UNO en FreeRTOS

Ahora se describe la programación del microcontrolador. El primer concepto que se maneja es el uso y creación de tareas, una vez que se ha creado una tarea en tiempo real, el scheduler se encargará de ejecutarla repetitivamente, siendo innecesario utilizar otro ciclo en el programa a menos que sea necesario.

Para definir una función se utiliza la siguiente sintaxis:

void TaskControl( void \*pvParameters );

Ahora viene la parte (en la función setup (de configuración) del Arduino) en donde se caracteriza la tarea, se tiene que definir su prioridad, la memoria que utilizará y un nombre para que sea entendible a los humanos. La sintaxis utilizada es.

xTaskCreate( //vamos a crear una tarea

TaskControl //a partir de la función definida anteriormente

, (const portCHAR \*) "LeyDeControl" //nombre para los humanos

, 128 // Stack size

, NULL

, 1 // Prioridad

, NULL );

}

Para resolver el problema del control, se utilizaron dos tareas, una tarea “ociosa” y una tarea con prioridad uno, la tarea ociosa se encargará de la lectura del encoder del motor, la tarea con prioridad 1 se encargara de la ley del control del motor y de variar la velocidad de los motores. En Arduino UNO una tarea de prioridad 0 (que es la prioridad más baja), coincide con las tareas que se ejecutan en el programa principal del microcontrolador, por ello en el programa realizado solo se definió una tarea pues las acciones descritas en el ciclo principal del Arduino funcionan por defecto como una tarea de prioridad 0, por lo que se estarán ejecutando constantemente sin necesidad de sincronización con el scheduler, sin embargo, esto es lo que necesitamos, pues no se deben perder tiempos de espera sincronizando tareas mientras el enconder esté rotando, de otra manera se perderían cuentas y el control sería inefectivo.

A continuación se muestra el código del programa:

1. #include <Arduino\_FreeRTOS.h>
2. #include <MotorDriver.h> //librería para el control de motores
4. MotorDriver m;
6. **int** enc1 = 9, enc2 = 10, contador=0, referencia=320, error, dir, countrol, incrementando=0;
7. unsigned **long** lec1, lec2, cA=0, cD=0, lec1A, lec2A, salida;
8. **int** A0a[] = {
9. 0, -1, 1, 0
10. };
11. **int** A1a[] = {
12. 1, 0, 0, -1
13. };
14. **int** A3a[] = {
15. 0, 1, -1, 0
16. };
17. **int** A2a[] = {
18. -1, 0, 0, 1
19. };
21. **void** TaskControl( **void** \*pvParameters );
23. //En setup se configurará la tarea definida
24. **void** setup() {
25. pinMode(enc1, INPUT);
26. pinMode(enc2, INPUT);
27. //Serial.begin(115200);
28. lec1A=digitalRead(enc1);
29. lec2A=digitalRead(enc2);
31. xTaskCreate(
32. TaskControl
33. ,  (**const** portCHAR \*) "LeyDeControl"
34. ,  128  // Stack size
35. ,  **NULL**
36. ,  1  // Prioridad
37. ,  **NULL** );
38. }
40. **void** loop()
41. {
42. // letura del encoder
43. lec1=digitalRead(enc1);
44. lec2=digitalRead(enc2);
45. //Ley de la cuadratura para determinar sentido de giro
46. **if** ((lec1 != lec1A) | (lec2 != lec2A)){
47. cA=2\*lec2A+lec1A;
48. cD=2\*lec2+lec1;
49. **switch**(cD)
50. {
51. **case** 0:
52. incrementando=A0a[cA];
53. **break**;
54. **case** 1:
55. incrementando=A1a[cA];
56. **break**;
57. **case** 2:
58. incrementando=A2a[cA];
59. **break**;
60. **case** 3:
61. incrementando=A3a[cA];
62. **break**;
63. **default**:
64. **break**;
65. }
66. contador=contador+incrementando;//se altera el contador
67. }
68. lec1A = lec1;
69. lec2A = lec2;
71. }
73. /\*--------------------------------------------------\*/
74. /\*---------------------- Tasks ---------------------\*/
75. /\*--------------------------------------------------\*/
77. **void** TaskControl( **void** \*pvParameters \_\_attribute\_\_((unused)) )  // This is a Task.
78. {
80. **for** (;;)
81. {
82. countrol=contador;
83. //Ley de control P, una ley es para levantar el péndulo y la otra es para estabilizarlo en su región lineal
84. error=referencia-countrol;
85. **if**(abs(error)>25){
86. salida=175;
87. }  **else**
88. salida=10\*abs(error);
90. **if**(error<0)
91. dir=BACKWARD;
92. **else**
93. dir=FORWARD;
94. //salida al motor con dirección indicada
95. m.motor(3,dir,salida);
96. m.motor(4,dir,salida);
98. vTaskDelay(1);  // one tick delay (15ms) in between reads for stability
99. }
100. }

# Conclusiones

Clemente Pérez José Arturo

Los sistemas operativos en tiempo real presentan una ventaja en la implementación de problemas de control como lo es el péndulo invertido, pues se pueden coordinar diferentes tareas sin perder la coherencia del programa, lo que es vital para un problema de teoría del control, sin embargo, es necesario conocer correctamente las funciones del SOTR utilizado, tener en consideración los tiempos de respuesta, la coordinación entre las tareas y el acceso a las regiones críticas, de otra manera el control se verá frustrado. En conclusión se debe entender correctamente el uso de los sistemas operativos en tiempo real y tener claridad en las tareas que se van a utilizar, ordenar la prioridad que se desea asignar a cada una y evitar retardos por acceso a regiones críticas.

Valdez Ochoa Arturo.

La ventaja de implementar un sistema en tiempo real para ciertas tareas es favorable, pues de esta manera, no se desperdician ciertos recursos para otras tareas. Poner en práctica los conocimientos adquiridos en la clase, o investigar otras cuestiones, dio como resultado un péndulo invertido el cual regresa a la posición original. Las dificultades iniciales, respecto al conteo y asignación de tareas se resolvió, reasignando tareas, para poder tener acceso a las zonas críticas, y así, controlar la posición.