Proyecto final. Delta Robot.

Sistemas Embebidos en Tiempo Real.

Cervantes Solorio, Álvaro
Garnica Cortés, Román
Granillo Martínez, Mariela Itzel

IECSA09

Universidad Aeronáutica en Querétaro
México

4 de marzo de 2025

Índice

Índice de figuras

1. Objetivo

Realizar un programa de control de un sistema embebido mediante el uso de sistemas operativos en tiempo real y normas de software determinista, para controlar un robot delta que ejecute tareas programadas.

2. Introducción

El proyecto a documentar en este reporte se va a enfocar en la programación y diseño de sistemas embebidos en tiempo real, el proyecto consiste en programar rutinas en nuestro robot delta para que este las siga. El robot delta consta de 3 brazos, existen robots delta hasta con 8 brazos, dichos brazos serán los que se encarguen de realizar el movimiento de una plataforma plana que unirá los 3 brazos. El otro extremo de los brazos estará sujeto a una base de madera fija.

En la base de madera fija se colocarán 3 motores reductores con encoder, uno para cada brazo, dichos motores serán los encargados del movimiento de los brazos. Son los motores que se programarán con el objetivo de que el robot realice ciertos movimientos con un orden. La estructura debe estar diseñada correctamente para que la plataforma movible sostenida por los 3 brazos siempre esté paralela al suelo. El objetivo es realizar un programa con el que el robot se mueva en ciertas direcciones en un orden.

El programa para el control de dicha rutina se hará a través de Code Composer Studio y HALCoGen en una tarjeta Hércules RM42, el programa estará basado en FreeRTOS. FreeRTOS es un sistema operativo en tiempo real diseñado para sistemas embebidos y que se utiliza en más de 30 plataformas para microcontroladores. Está desarrollado en lenguaje C y su actual dueño es Amazon Web Services.

3. Marco Teórico

En este marco teórico se darán los conceptos necesarios para la comprensión de lo que se realizará en este proyecto, desde conceptos relacionados con la materia hasta conceptos de algunos materiales usados. Los conceptos serán breves, con la información necesaria para comprenderlos temas y no confundir.

3.1. Sistema Operativo en Tiempo Real. RTOS.

Un sistema operativo en tiempo real posee dos características clave, es predecible y determinista. Con predecible nos referimos a que sabemos que salida va a producir cierta entrada, y siempre será la misma; por otro lado, determinista significa que sabremos exactamente cuanto durará haciendo alguna tarea, no podrá durar más ni menos del tiempo establecido. Los RTOS se dividen en Soft real-time y Hard real-time, los primeros operan dentro de un margen de cientos de milisegundos, mientras que los segundos operan dentro de decenas de milisegundos o menos. Estos sistema utilizan una planificación de tareas basada en prioridades, es decir, primero ejecuta la tarea de mayor prioridad y así sucesivamente. Una ventaja de estos sistemas operativos es que su peso es bastante pequeño en relación con sistemas operativos como Windows o Linux.

3.2. FreeRTOS.

FreeRTOS es un sistema operativo en tiempo real, actualmente es el más usado por empresas y proyectos individuales. Fue desarrollado durante 18 años por compañías líderes en la fabricación de chips y actualmente es el más usado para microcontroladores. FreeRTOS provee gran robustez y soporte en una gran cantidad de dispositivos, al ser desarrollado por Amazon Web Services prometen un buen soporte para el sistema por un largo periodo de tiempo.

Figura 1: FreeRTOS.

3.3. Robot Delta.

Los robots delta son un tipo de robot paralelo, el cual se forma por, mínimo dos brazos, los cuales sostienen una plataforma paralela a la superficie inferior y la mueven en distintos sentidos, mientras más brazos tendrá más grados de libertar para mover a la plataforma. Dichos brazos son accionados por motores reductores, los cuales a su vez son controlador por algún microcontrolador o microporcesador. Uno de los principales usos de estos robots, es en las cadenas de producción, ya que su programación les permite realizar rutinas cada cierto tiempo, lo cual es característico de las cadenas de producción donde cada cierto tiempo llega una pieza o producto y se tiene que realizar una tarea con dicho objeto.

Figura 2: Robot delta.

3.4. Motor DC con encoder.

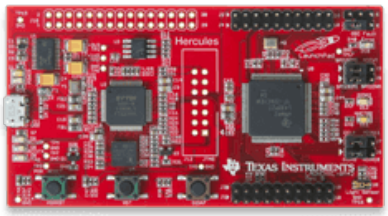
Un motor DC con encoder es básicamente un motor reductor con un encoder para poder medir la posición y velocidad del motor. Una de las características principales de estos motores es la capacidad de poder ejecutar y controlar movimientos precisos, esto es exáctamente lo que se necesita para mover los brazos del robot delta, por lo tanto se usarán 3 motores de este tipo.

Figura 3: Motor DC con encoder.

3.5. Hércules RM42

La Hércules RM42 es un dispositivo de evaluación económico para que los usuarios comiencen a familiarizarse con el desarrollo con microcontroladores Hércules. La RM42 no es la Hércules con mayores prestaciones, pero para nuestras necesidades es suficiente, es un microcontrolador con el cual podemos realizar la programación de distintos sistemas de control discreto. Esta basada en el microcontrolador dual core lock-step ARM® Cortex®-R4 el cual contiene un ADC de 12 bits, HET y pins programables para timers y también cuenta con interfaces de comunicación serial.

Figura 4: Hércules RM42.

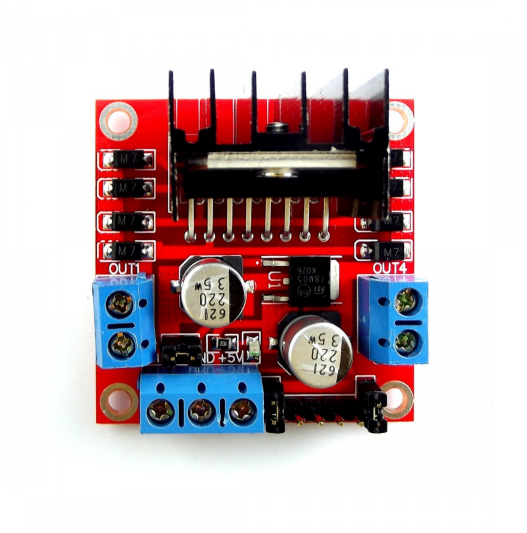


3.6. Puente H.

El puente H es un circuito electrónico que se utiliza para controlar a un motor eléctrico, tanto en sus movimiento en avance y retroceso. El circuito principal está compuesto por 4 interruptores o transistores que forman una figura en forma de

H, de ahí su nombre. Actualmente ya se venden puentes H en circuitos con distintos puertos para facilitar su su uso.

Figura 5: Puente H.



4. Desarrollo.

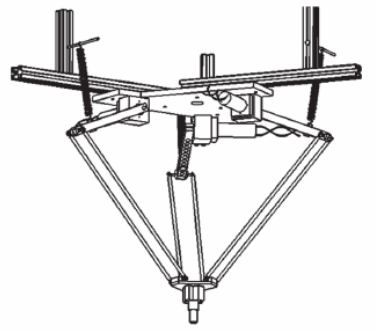
4.1. Materiales.

1. Hércules RM42
2. Estructura de Robot Delta
3. Motor reductor con encoder
4. Cables
5. Puentes H.

4.2. Estructura de robot delta.

La estructura del robot paralelo utilzado para este proyecto es de tipo delta, la cual cuenta con 3 motores reductores que mueven 3 brazos interconectados en su punto final, en la imagen se puede comprender de mejor forma la estructura de un robot paralelo tipo delta.

Figura 6: Diagrama de robot delta.



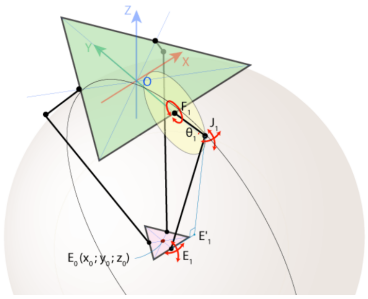
Con base en el diagrama del robot delta se modificó una estructura ya existente de un robot similar, se cambiaron los motores reductores por el modelo jgy-370. Por lo tanto se realizó una base para cada motor para poder fijarlo a la estructura, de igual forma se diseñaron brazos para cada motor, dichos brazos se conectarían a otros más largos mediante una unión de tipo bola.

Figura 7: Motorreductor utilizado.

Para poder verificar que un robot delta funciona de forma correcta, al momento de que los 3 motores estén en movimiento, el efector que es la parte final donde los 3 motores se interconectan, siempre debe estar paralela a la superficie inferior, de ahí el nombre de robot paralelo.

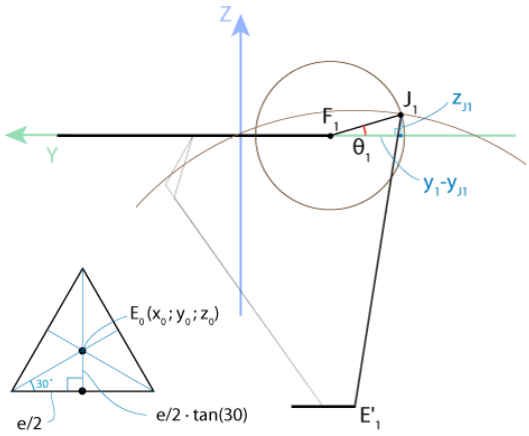
4.3. Modelo cinemático inverso.

Figura 8: Diagrama robot delta.



Para la programación y análisis del robot delta existen dos modelos, el modelo cinemático directo y el modelo cinemático inverso, en este proyecto se utilizó el modelo cinemático inverso. El modelo cinemático inverso o MCI, consiste en, partiendo de un mismo ángulo todos los motores, encontrar el ángulo al cual se deben de mover los motores para que el efector llegue a una posición deseada, la cual se da en el plano cartesiano de 3 ejes. Es decir, este modelo nos sirve para pasar de una posición en el plano a los ángulo de nuestros motores. En la siguiente imagen se puede observar gráficamente la lógica detrás del funcionamiento de las ecuaciones del MCI, el método consiste en encontrar la posición en el eje Y y Z del extremo del brazo más corto, y con esa posición se calcula el ángulo θ del motor. Cabe resaltar que el modelo es para un solo motor, por lo tanto para calcular los otros ángulos debemos recurrir a métodos de traslación y rotación.

Figura 9: Diagrama de modelo cinemático inverso.



$$E(x_0, y_0, z_0) \tag{1}$$

$$EE_1 = \frac{e}{2}tan(30) \tag{2}$$

$$EE_1 = \frac{e}{2}tan(30) = \frac{e}{2\sqrt{3}} \tag{3}$$

$$E_1(x-0, y_0 - \frac{e}{2\sqrt{3}}, z_0) = E_1(0, y_0 - \frac{e}{2\sqrt{3}}, z_0) \tag{4}$$

$$E_1E_1 = E_1J_1 = \sqrt{E_1J_1^2 - E_1E_1^2} = \sqrt{r_e^2 - x_0^2} \tag{5}$$

$$F_1(0, -\frac{f}{2\sqrt{3}}, 0) \tag{6}$$

$$(y_j1 - y_F1)^2 + (z_j1 - z_F1)^2 = r_f^2 \tag{7}$$

$$(y_j1 - y_F1)^2 + (z_j1 - z_F1)^2 = r_E^2 - X_0^2 \tag{8}$$

$$(y_j1 + \frac{f}{2\sqrt{3}})^2 + z_J1^2 = r_f^2 \tag{9}$$

$$(y_j1 - y_0 + \frac{e}{2\sqrt{3}})^2 + (z_j1 - z_0)^2 = r_e^2 - x_0^2 \tag{10}$$

$$J_1(0, y_j1, z_j1) \tag{11}$$

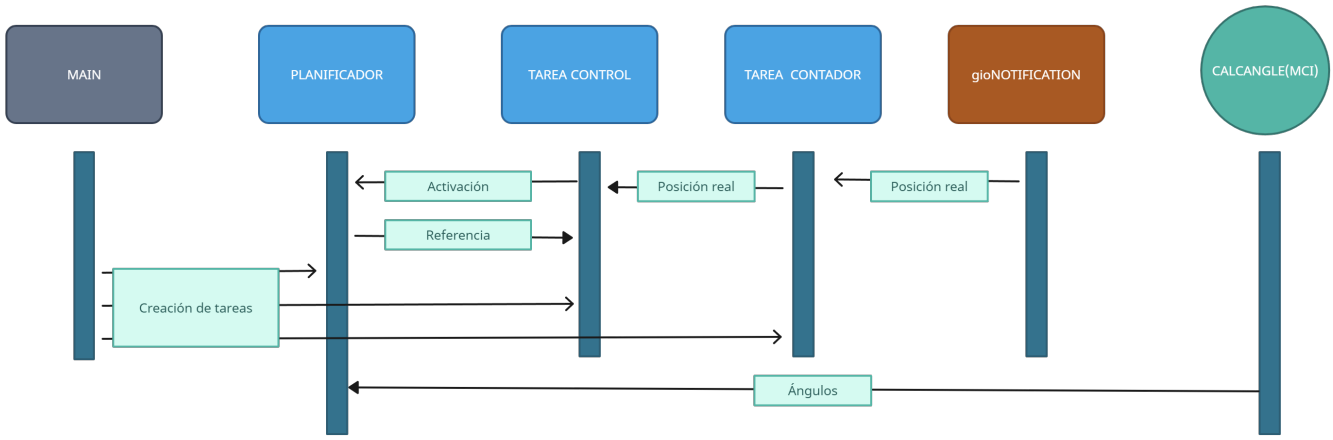
$$\theta_1 = arctan(\frac{z_j1}{y_F1 - y_J1}) \tag{12}$$

Las ecuaciones anteriores se pueden transcribir a un algoritmo para poder utilizarlas en en el programa del robot delta. Dicho algoritmo se encuentra en el código final en la sección de anexos.

2

4.4. Diagrama UML.

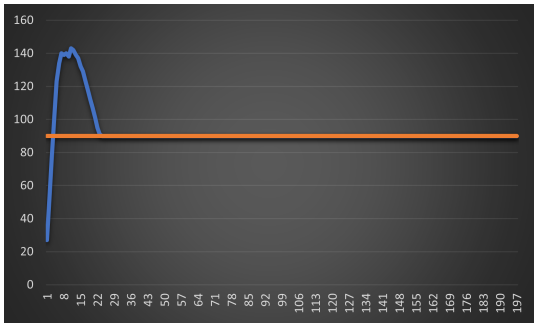
Figura 10: Diagrama UML secuencial.



5. Gráficas de sintonización de control

En la siguiente gráfica colocamos una referencia de 90 pulsos hacia arriba, y declaramos tres arreglos donde se guardaban las posiciones actuales de cada motor, en la siguiente solo se muestra el comportamiento de un motor.

Figura 11: Sintonización de control.



6. Conclusiones.

Al haber sido finalizado el proyecto podemos llegar a varios puntos de vista sobre el funcionamiento del robot delta, al inicio del proyecto la primer etapa que se realizó fue la selección de materiales, una de las partes más importantes son los motorreductores con encoder, en ese momento es difícil decidir cuál es la mejor opción, al estar en la etapa final del proyecto, se puede deducir que la selección de ese componente no fue la mejor opción. Al inicio del proyecto como los motorreductores no recibían tanta demanda funcionaban bien, pero conforme se fue avanzando notábamos irregularidades en un motorreductor y en la etapa final dicho motorreductor afecta demasiado al funcionamiento del robot. En una de las primeras etapas de desarrollo de código, donde se cuentan las posiciones y el cambio de estas de cada motor, todo funcionaba correctamente, cada motor hacía bien sus cuentas.

En la siguiente etapa donde se debía implementar el control para cada motor, así como encontrar su compensador de zona muerta, todos los motorreductores seguían funcionando correctamente, el control de las posiciones era aceptable. Después instalamos los primeros brazos a los motores, los brazos de menor tamaño, aquí fue donde comenzamos a notar irregularidades en un motor, dicho motor cuando rotaba más hacia arriba y cuando debía hacerlo hacia abajo no lo hacía como los otros dos motores. Se aplicaron compensadores distintos a ese motor con la finalidad de igualar su comportamiento al de los otros dos motores y se pudo solucionar un poco pero no del todo. Al ana-

lizar el código con breakpoints para ver mejor el comportamiento de dicho motor, logramos ver que en sus cuentas, dicho motor se comportaba bien, pero en la realidad se encontraba en otra posición, con esto se llegó a la solución de modificar la resolución de dicho motor pero esto tampoco solucionó el problema.

Teniendo en cuenta esto, se avanzó con el proceso por falta de tiempo para cambiar de motores, entonces se instaló toda la estructura completa del robot delta y se implementó en el código el planificador con la función del MCI. Primeramente se aplicaba una posición fija en el planificador, los 3 motores se comportaban bien, incluso el motor que fallaba, entonces se aplicó una rutina de dos posiciones en el eje Z al planificador. Por lo tanto, los ángulos de los motores iban a positivos a negativos por requisito del proyecto, en este punto el motor que fallaba regresó a su protagonismo. Al ser una rutina de posiciones cambiantes, el encoder a veces perdía cuentas y a veces no, pero lo más común era que en software el motor se comportaba bien, pero en la realidad no, lo que afectó y sigue afectando al seguimiento de rutas del robot delta.

El hecho de que un motor no funciona correctamente arruina todo el funcionamiento, y si a esto se suma el hecho de que dicho fallo no es regular, la falla es un más grave. Entonces se llegó una conclusión, ir aumentando o reduciendo la posición del robot solo en el eje Z por medio del botón de reset del microcontrolador, extrañamente cuando avanzaba pausadamente hacia arriba o abajo, el fallo del motor dos desaparecía, por lo que se redujo la velocidad de movimiento de los 3 motores para verificar si ese era el error, pero no fue así. Habiendo probado bastantes soluciones y verificado posibles errores, se llegó a la conclusión de encontrar ganancias y rutas donde el motor que falla no nos afecte tanto, el problema actual es que al momento de seguir una ruta solo lo hace un poco bien la primera vez, pero cuando pasa más tiempo las cuentas perdidas por el motor afectan a toda la trayectoria y el robot delta no puede continuar.

2

A. Anexos

A.1. Código CSS.

```
/** @file sys_main.c
*   @brief Application main file
*   @date 11-Dec-2018
*   @version 04.07.01
*
*   This file contains an empty main function,
*   which can be used for the application.
*/

/*
* Copyright (C) 2009-2018 Texas Instruments Incorporated - www.ti.com
*
*
*  Redistribution and use in source and binary forms, with or without
*  modification, are permitted provided that the following conditions
*  are met:
*
*    Redistributions of source code must retain the above copyright
*    notice, this list of conditions and the following disclaimer.
*
*    Redistributions in binary form must reproduce the above copyright
*    notice, this list of conditions and the following disclaimer in the
*    documentation and/or other materials provided with the
*    distribution.
*
*    Neither the name of Texas Instruments Incorporated nor the names of
*    its contributors may be used to endorse or promote products derived
*    from this software without specific prior written permission.
*
*  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
*  "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
*  LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
*  A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
*  OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
*  SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
*  LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
*  DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
*  THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
*  (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
*  OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*
*/


/* USER CODE BEGIN (0) */
#define mainDELAY_LOOP_COUNT (0xFFFFF)
/* USER CODE END */

/* Include Files */

#include "sys_common.h"

/* USER CODE BEGIN (1) */
//////incluimos las librerias neesarias
#include "FreeRTOS.h"
#include "os_task.h"
#include "os_queue.h"
#include "het.h"
#include "gio.h"
#include "os_semphr.h"
#include "stdlib.h"
#include "math.h"
/* USER CODE END */

/** @fn void main(void)
*   @brief Application main function
*   @note This function is empty by default.
*
```

```
69 *   This function is called after startup.
70 *   The user can use this function to implement the application.
71 */
72
73 /* USER CODE BEGIN (2) */
74 ////declaracion de tareas
75 void vTarea1(void *pvParameters);
76 void vPosition(void *pvParameters);
77 void vPlanif(void *pvParameters);
78 /////declaracion de funciones
79 void gioNotification(gioPORT_t *port, uint32 bit);
80 void limitControl(float MIN, float MAX, float *x);
81 float calcangle(float Angle, float x0, float y0, float z0);
82 int limitpwm(uint32_t MAX, uint32_t x);
83 ///// declaracion de el manejador de cada cola
84 xQueueHandle xQueueGio;
85 xQueueHandle xQueuePos1;
86 xQueueHandle xQueueRef;
87 xQueueHandle xQueueFlag;
88 ///////declaracion de estructuras usadas
89 typedef struct{
90     uint32 giointer;
91     uint32 giocomp;
92 } gioint;
93
94 typedef struct{
95     int32_t countpos;
96     int32_t countpos2;
97     int32_t countpos3;
98 } counts;
99
100 typedef struct{
101     float theta1;
102     float theta2;
103     float theta3;
104 }thetas;
105
106 typedef struct{
107     float ref1;
108     float ref2;
109     float ref3;
110 } refs;
111
112 typedef struct{
113     int i;
114 } flags;
115
116 const float pi = 3.141592 ; //Valor de pi
117 int flag = 0;
118
119
120 /* USER CODE END */
121
122 int main(void)
123 {
124 /* USER CODE BEGIN (3) */
125     gioInit();
126     hetInit();
127     //declaracion de estructuras para inicializar las colas
128     counts count;
129     thetas tets;
130     flags flag;
131
132     ////valores de estructuras que se envian a las colas
133     count.countpos = 0.0;
134     count.countpos2 = 0.0;
135     count.countpos3 = 0.0;
136
137     tets.theta1 = 0.0;
138     tets.theta2 = 0.0;
139     tets.theta3 = 0.0;
140
141     flag.i = 0;
142
143     /////creacion de colas
144     xQueueGio =  xQueueCreate(15, sizeof(gioint));
145     xQueuePos1 = xQueueCreate(1, sizeof(counts));
146     xQueueRef = xQueueCreate(1, sizeof(thetas));
```

```
147     xQueueFlag = xQueueCreate(1, sizeof(flags));
148
149     //////inicializamos las colas
150     xQueueSend(xQueuePos1, &count, 0);
151     xQueueSend(xQueueRef, &tets, 0);
152     xQueueSend(xQueueFlag, &flag, 0);
153
154
155     ////se crean las tareas
156     if(xTaskCreate(vPosition, "Position", configMINIMAL_STACK_SIZE, NULL, 3, NULL)!=pdTRUE)
            {while(1);}
157     if(xTaskCreate(vTarea1, "Tarea1", 4*configMINIMAL_STACK_SIZE, NULL, 4, NULL)!=pdTRUE)
            {while(1);}//control
158     if(xTaskCreate(vPlanif, "Planif", configMINIMAL_STACK_SIZE, NULL, 2, NULL)!=pdTRUE)
            {while(1);}
159
160
161     vTaskStartScheduler();//Se inicializa el scheduler
162
163     while(1);
164
165 /* USER CODE END */
166
167     return 0;
168 }
169
170 int limitpwm(uint32_t MAX, uint32_t x)
171 {
172     uint32_t aux = x;
173
174     if(aux > MAX)
175     {
176         aux = MAX;
177     }
178
179     return aux;
180 }
181
182 /* USER CODE BEGIN (4) */
183 /*
184  * Funcion para limitar el control
185  */
186 void limitControl(float MIN, float MAX, float *x)
187 {
188     float *aux = x;
189     if(*aux < MIN)
190     {
191         *aux = MIN;
192     }
193     if(*aux > MAX)
194     {
195         *aux = MAX;
196     }
197 }
198
199 /*
200  * En esta funcion se ejecuta el MCI,recibe las posiciones
201  * en los 3 ejes y obtiene el angulo en el eje Y y Z, por lo tanto
202  * en el planificador se realizar el proceso de traslacion y
203  * rotacion por la posicion de cada motor.
204  */
205 float calcangle(float Angle, float x0, float y0, float z0)
206 {
207     float pi = 3.14159;
208     ///variables de medidas del robot
209     double PlatformTri = 7;
210     double BassTri = 3;
211     double ArmLength = 6.5;
212     double RodLength = 27;
213
214     ////comienzan ecuaciones de MCI
215     double y1 = -0.5 * tan(30) * BassTri;
216     y0 -= 0.5 * tan(30) * PlatformTri;
217
218     double aV = (x0 * x0 + y0 * y0 + z0 * z0 + ArmLength * ArmLength - RodLength *
            RodLength - y1 * y1) / (2.0 * z0);
219     double bV = (y1 - y0) / z0;
220
```

```
221
222     double dV = -(aV + bV * y1) * (aV + bV * y1) + ArmLength * (bV * bV * ArmLength +
            ArmLength);
223     if (dV < 0)
224     {
225       Angle=0;
226     }
227     else{
228         double yj = (y1 - aV * bV - sqrt(dV)) / (bV * bV + 1);
229         double zj = aV + bV * yj;
230         Angle = atan2(-zj, (y1 - yj)) * 180.0 / pi;
231     }
232
233
234     return Angle; //la variable de retorno es el angulo
235
236 }
237 /*
238  * En la tarea de planificador realizamos las rutas
239  * a seguir para formar las figuras deseadas.
240  */
241 void vPlanif(void * pvParameters)
242 {
243     thetas tets;//estrucuta de referencias
244     flags flag;//estructura de la cola que bloquea la tarea
245     TickType_t puls = 700/portTICK_PERIOD_MS;
246     float  x0 = 0, y0 = 0, z0 = 19.0, Angle;//variables para inicializar la posicion
247     float theta1 = 0, theta2 = 0.0, theta3 = 0.0;
248     int t = 0, i = 1;
249     float resoluc = 3.6, resoluc2 = 3.6;//recoluciones de motores
250
251
252     for(;;)
253     {
254         //al recibir el elemento de la cola la tarea pasa de bloqueada a ejecucion
255         xQueueReceive(xQueueFlag, &flag, portMAX_DELAY);
256
257
258         //switch para ir cambiando de referencia cada vez que se ejecute el planificador
259             switch(i)
260             {
261                 case 1:
262                     z0 = 19.0;
263                     x0 = 1;
264                     y0 = 1;
265                     break;
266
267                 case 2:
268                     z0 = 19.0;
269                     x0 = 1;
270                     y0 = -1;
271                     break;
272
273                 case 3:
274                     z0 = 19.0;
275                     x0 = -1;
276                     y0 = -1;
277                     break;
278
279                 case 4:
280                     z0 = 19.0;
281                     x0 = -1;
282                     y0 = 1;
283                     break;
284                 case 5:
285                     z0 = 20.0;
286                     x0 = 1;
287                     y0 = 1;
288                     break;
289
290                 case 6:
291                     z0 = 20.0;
292                     x0 = 1;
293                     y0 = -1;
294                     break;
295
296                 case 7:
297                     z0 = 20.0;
```

```
298                     x0 = -1;
299                     y0 = -1;
300                     break;
301 
302                 case 8:
303                     z0 = 20.0;
304                     x0 = -1;
305                     y0 = 1;
306                     break;
307                 case 9:
308                     z0 = 21.0;
309                     x0 = 1;
310                     y0 = 1;
311                     break;
312 
313                 case 10:
314                     z0 = 21.0;
315                     x0 = 1;
316                     y0 = -1;
317                     break;
318 
319                 case 11:
320                     z0 = 21.0;
321                     x0 = -1;
322                     y0 = -1;
323                     break;
324 
325                 case 12:
326                     z0 = 21.0;
327                     x0 = -1;
328                     y0 = 1;
329                     i = 0;
330                     break;
331 
332             }
333             i++;
334              //variable para ir cambiando de referencia enviada
335 
336 
337             /*
338              * calculo de los angulo de cada motor segun la posicion deseada
339              * el angulo 2 y 3 tienen las operaciones de translacion y rotacion
340              * debido a las posciones de los motores.
341              */
342           tets.theta1 = calcangle(Angle,x0, y0, z0);
343           tets.theta2 = calcangle(Angle, x0 * cos(2.0944) + y0 * sin(2.0944),
344               y0 * cos(2.0944) - x0 * sin(2.0944),
345               z0);
346           tets.theta3 = calcangle(Angle, x0 * cos(2.0944) - y0 * sin(2.0944),
347               y0 * cos(2.0944) + x0 * sin(2.0944),
348               z0);
349 
350           //convertimos los angulos a conteos de encoder
351           tets.theta1 = tets.theta1 * resoluc;
352           tets.theta2 = tets.theta2 * resoluc;
353           tets.theta3 = tets.theta3 * resoluc;
354 
355 
356           xQueueOverwrite(xQueueRef, &tets);//enviamos la estructura de 3 angulos al control
357 
358           vTaskDelay(puls);
359     }
360 
361 
362 }
363 /*
364  * En esta tarea se realiza el control de los tres motores,
365  * recibe la posicion real de cada motor de la cola sin borrarla
366  * de  dicha cola, igualmente recibe la referencia generada por el
367  * planificador y realiza el control
368  */
369 void vTarea1(void *pvParameters)
370 {
371     //MI: KI=1.5 ERROR 15
372     volatile float kp2 = .4, kp= .4, kp3 = .4, kd = 0.005, ki3 = 2.5, T = .02;//ganancias
            para motores
373     volatile float ki1 = 2.5, ki2 = 2.5;//2.6
374     //variables usadas para las salidas de control de cada motor
```

```
375     volatile float Up1, Ud1, Ui1, Uc1, e1[5] = {0};
376     volatile float Up2, Ud2, Ui2 = 0, Uc2, e2[5] = {0};
377     volatile float Up3, Ud3, Ui3, Uc3, e3[5] = {0};
378     //inicializamos los arreglos usados en cero
379     float step[3] = {0.0,0.0,0.0};
380     float pos_d[3] = {0.0, 0.0, 0.0};
381     float pos_r[3] = {0.0, 0.0, 0.0};
382     float pos_ff[3] = {0.0, 0.0, 0.0};
383     float comp_grav[3]={0.0, 0.0, 0.0};
384     //declaracion de pwm's y su periodo
385     hetSIGNAL_t pwm0het0, pwm1het12, pwm2het18;
386     pwm2het18.period =1000.0;
387     pwm1het12.period = 1000.0;
388     pwm0het0.period = 1000.0;
389     uint32_t i=0, tcount = 0;
390     //variable para el tiempo de espera
391     volatile TickType_t xTicksToWait = 0;
392     //declaracion de estructuras utilizadas
393     counts position;
394     thetas tets;
395     flags flag;
396     //variable para modificar la velocidad de cada motor, suele no utilizarse por falla de
          motor 2
397     int calc_step=0;
398     float pi = 3.14159;
399     for(;;)
400     {
401
402         //recepcion de estructura de posicion real y referecia
403         xQueuePeek(xQueuePos1, &position, xTicksToWait); //TOMA POSICION DE ENCODERS
404         xQueueReceive(xQueueRef, &tets, xTicksToWait);
405
406         //asignacion de referencias al arreglo usado
407         pos_ff[0] = tets.theta1;
408         pos_ff[1] = tets.theta2;
409         pos_ff[2] = tets.theta3;
410
411
412        /*if(calc_step==0){ //CALCULA EL STEP SOLO CUANDO NO ESTA CONTROLANDO
413             for(i = 0; i<3; i++)
414                 {
415                     step[i]=(pos_ff[i]-pos_r[i])/10; //PARA QUE TODOS LLEGUEN AL MISMO
                          TIEMPO**
416                 }
417                 calc_step=1; //BLOQUEAMOS CALCSTEP
418         }
419
420         if(abs(pos_d[0]) < abs(pos_ff[0]) && abs(pos_d[1]) < abs(pos_ff[1]) &&
              abs(pos_d[2]) < abs(pos_ff[2]))//SUMAR STEP SI LA DIFERENCIA ENTRE LAS
              POSICIONES ES MAYOR A 1
421         {
422             for(i = 0; i<3;i++){
423                 pos_d[i] += step[i];
424             }
425         }
426         else{//STEP EN 0 POR SI LAS DUDAS
427                 //step[i]=0;
428         }*/
429
430         //asignacion de posiciones reales al arreglo utilzado
431         pos_r[0] = (float)(position.countpos);//*(resoluc);
432         pos_r[1] = (float)(position.countpos2);//*(resoluc);
433         pos_r[2] = (float)(position.countpos3);//*(resoluc);
434
435         ////////////calculo de errores de cada motor
436         e1[4] = e1[3]; e1[3] = e1[2]; e1[2] = e1[1]; e1[1] = e1[0];
437         e1[0] = (float)(pos_ff[0] - pos_r[0]);
438         e2[4] = e2[3];  e2[3] = e2[2];  e2[2] = e2[1];  e2[1] = e2[0];
439         e2[0] = (float)(pos_ff[1] - pos_r[1]);
440         e3[4] = e3[3];  e3[3] = e3[2];  e3[2] = e3[1];  e3[1] = e3[0];
441         e3[0] =(float)(pos_ff[2] - pos_r[2]);
442
443         //comp_grav[0]=cos((pos_r[0])*pi*0.25714/180);
444         //comp_grav[1]=cos((pos_r[1])*pi*0.25714/180);
445         //comp_grav[2]=cos((pos_r[2])*pi*0.25714/180);
446
447         //////calculo de salidas de control de cada motor////////
448         Up1 = kp *(e1[0]);
```

```
449         Up2 = kp2 * (e2[0]);
450         Up3 = kp3 *(e3[0]);
451         // *****
452         Ui1 = Ui1 + ki1*T*e1[0];
453         Ui2 = Ui2 + ki2*T*e2[0];
454         Ui3 = Ui3 + ki3*T*e3[0];
455
456         limitControl(-60, 60, &Ui1);
457         limitControl(-60, 60, &Ui2);
458         limitControl(-60, 60, &Ui3);
459         // *****
460         Ud1 = (kd/T) * (e1[0] - e1[4]);
461         Ud2 = (kd/T) * (e2[0] - e2[4]);
462         Ud3 = (kd/T) * (e3[0] - e3[4]);
463         // *****
464         Uc1 = Up1 + Ud1 + Ui1;
465         Uc2 = Up2 + Ud2 + Ui2;
466         Uc3 = Up3 + Ud3 + Ui3;
467         // *****
468         ////////limitamos la salida de control de cada motor
469
470
471         limitControl( -150.0, 150.0, &Uc1 );
472         limitControl( -150.0, 150.0, &Uc2 );
473         limitControl( -150.0, 150.0, &Uc3 );
474
475
476
477         ///// para cada motor, cambiamos el sentido de giro segun el signo de su salida de
                control/////////
478         if(Uc1 > 00.0)
479         {
480             gioSetBit(hetPORT1, (uint32)2, (uint32)1);//2
481             gioSetBit(hetPORT1, (uint32)4, (uint32)0);//4
482             pwm0het0.duty =(uint32_t)(Uc1) + 35 + (int32_t)(cos((pos_r[0])*pi*0.25714/180)
                    * 30);
483             //pwm0het0.duty = (uint32_t)(limitpwm(80, pwm0het0.duty));
484
485         }
486         else
487         {
488             gioSetBit(hetPORT1, (uint32)2, (uint32)0);//2
489             gioSetBit(hetPORT1, (uint32)4, (uint32)1);//4
490             pwm0het0.duty= (uint32_t)(-Uc1) + 5 +
                    (uint32_t)(cos((pos_r[0])*pi*0.25714/180) * 8);
491             //pwm0het0.duty = (uint32_t)(limitpwm(80, pwm0het0.duty));
492
493         }
494
495
496         if(Uc2>00.0)
497         {
498             gioSetBit(hetPORT1, (uint32)6, (uint32)1);
499             gioSetBit(hetPORT1, (uint32)10, (uint32)0);
500             pwm1het12.duty=  (uint32_t)(Uc2) + 35 +
                    (int32_t)(cos((pos_r[1])*pi*0.25714/180) * 30);
501             //pwm1het12.duty = (uint32_t)(limitpwm(80, pwm1het12.duty));
502
503         }
504         else
505         {
506             gioSetBit(hetPORT1, (uint32)6, (uint32)0);
507             gioSetBit(hetPORT1, (uint32)10, (uint32)1);
508             pwm1het12.duty= (uint32_t)(-Uc2) + 15 +
                    (int32_t)(cos((pos_r[1])*pi*0.25714/180) * 8);
509             //pwm1het12.duty = (uint32_t)(limitpwm(80, pwm1het12.duty));
510
511         }
512
513         if(Uc3>00.0)
514         {
515             gioSetBit(hetPORT1, (uint32)14, (uint32)1);
516             gioSetBit(hetPORT1, (uint32)16, (uint32)0);
517             pwm2het18.duty= (uint32_t)(Uc3) + 35 + (int32_t)(cos((pos_r[2])*pi*0.25714/180)
                    * 30);
518             //pwm2het18.duty = (uint32_t)(limitpwm(80, pwm2het18.duty));
519
520         }
```

```
521         else
522         {
523             gioSetBit(hetPORT1, (uint32)14, (uint32)0);
524             gioSetBit(hetPORT1, (uint32)16, (uint32)1);
525             pwm2het18.duty= (uint32_t)(-Uc3) + 5 + (int32_t)(cos((pos_r[2])*pi*0.25714/180)
                    * 8);
526             //pwm2het18.duty = (uint32_t)(limitpwm(80, pwm2het18.duty));
527         }
528
529         ///inicializamos el pwm de cada motor
530
531
532         //condicional para ejecutarse cuando el error de los motores llegue al deseado
533         if(abs(e1[0]) < 15 &&  abs(e2[0]) < 15 &&  abs(e3[0]) < 15)
534         {
535             pwm0het0.duty = 10;
536             pwm2het18.duty = 10;
537             pwm1het12.duty = 10;
538             Ui1 = 0.0;
539             Ui2 = 0.0;
540             Ui3 = 0.0;
541             calc_step = 0;
542             flag.i = 1;//variable que se envia a la cola que desbloquea al planificador, no
                    importa su valor
543             xQueueSend(xQueueFlag, &flag, 0);//se envia un valor a la cola para que el
                    planificador lo reciba y se active
544
545         }
546         pwmSetSignal(hetRAM1, pwm0,pwm0het0);
547         pwmSetSignal(hetRAM1, pwm1,pwm1het12);
548         pwmSetSignal(hetRAM1, pwm2,pwm2het18);
549
550         vTaskDelay(20/portTICK_PERIOD_MS);//delay del control
551
552     }
553 }
554 /*
555  * En esta tarea realizamos el conteo de posicion de
556  * cada motor, al recibir cada elemento de la cola
557  * generada por gioNotifcation, aumenta o reduce la
558  * posicion real de cada motor segun los valores
559  * de los puertos GIO que reciba, se crea una estructura
560  * la cual cuenta con 3 variables, una para cada posicion
561  * de cada motor. La estructura se envia a una cola
562  * sobreescribiendo la posicion de cada motor, dicha cola
563  * se envia a la tarea de control
564  */
565 void vPosition(void *pvParameters)
566 {
567     volatile int32_t position1;
568     volatile int32_t position2;
569     volatile int32_t position3;
570     counts position;
571     gioint datagio;
572     position.countpos = 0;
573     position.countpos2 = 0;
574     position.countpos3 = 0;
575
576     for(;;)
577     {
578         if(xQueueReceive(xQueueGio, &datagio, portMAX_DELAY) != pdPASS)
579         {
580
581         }
582         else{
583             switch(datagio.giointer)
584             {
585             case 0:
586                 if(datagio.giocomp == 0)
587                 {
588                     position.countpos --;
589
590                 }else
591                 {
592                     position.countpos ++;
593
594                 }
595                 break;
```

```
596             case 1:
597                 if(datagio.giocomp == 0)
598                 {
599                     position.countpos ++;
600 
601                 }else
602                 {
603                     position.countpos --;
604                 }
605                 break;
606             case 3:
607                 if(datagio.giocomp == 0)
608                 {
609                     position.countpos2 --;
610                 }else
611                 {
612                     position.countpos2 ++;
613                 }
614                 break;
615             case 4:
616                 if(datagio.giocomp == 0)
617                 {
618                     position.countpos2 ++;
619                 }else
620                 {
621                     position.countpos2 --;
622                 }
623                 break;
624             case 5:
625                 if(datagio.giocomp == 0)
626                 {
627                     position.countpos3 --;
628                 }else
629                 {
630                     position.countpos3 ++;
631                 }
632                 break;
633             case 6:
634                 if(datagio.giocomp == 0)
635                 {
636                     position.countpos3 ++;
637                 }else
638                 {
639                     position.countpos3 --;
640                 }
641                 break;
642             }
643         }
644 
645 
646         xQueueOverwrite(xQueuePos1, &position);//sobreescritura de la estructura de
                posiciones en la cola
647 
648     }
649 }
650 /*Funcion en la que se almacena el bit del puerto GIO activo
651  *  y el valor del bit de su puerto GIO complemento, todo esto
652  *  en una estructura, la cual se envia a una cola de 16 elementos
653  *  de dicha estructura. La configuracion del microcontrolador
654  *  solo detecta flancos de subida, por lo tanto cuando un GIO se
655  *  activa, almacena ese GIO y el valor de su GIO complementos,
656  *  con esto puede detectar el sentido de giro y la cantidad de giro
657  *  de cada motor.
658  *
659  */
660 void gioNotification(gioPORT_t *port, uint32 bit)
661 {
662     BaseType_t xHigherPriorityTaskWoken;
663     gioint datagiointerr;
664 
665     datagiointerr.giointer = bit;
666 
667     switch(bit){
668     case (uint32)0:
669 
670         datagiointerr.giocomp = gioGetBit(gioPORTA, (uint32)1);
671 
672         break;
```

```
673     case (uint32)1:
674
675         datagiointerr.giocomp = gioGetBit(gioPORTA, (uint32)0);
676
677         break;
678     case (uint32)3:
679
680         datagiointerr.giocomp = gioGetBit(gioPORTA, (uint32)4);
681
682         break;
683     case (uint32)4:
684
685         datagiointerr.giocomp = gioGetBit(gioPORTA, (uint32)3);
686
687         break;
688     case (uint32)5:
689
690         datagiointerr.giocomp = gioGetBit(gioPORTA, (uint32)6);
691
692         break;
693     case (uint32)6:
694
695         datagiointerr.giocomp = gioGetBit(gioPORTA, (uint32)5);
696
697         break;
698     }
699     xQueueSendFromISR(xQueueGio, &datagiointerr, 0);//envio constante de dicha estructura
700 }
701
702
703
704 /* USER CODE END */
```

Listado 1: Código CCS

A.2. Configuración de HALCoGen.

Figura 12

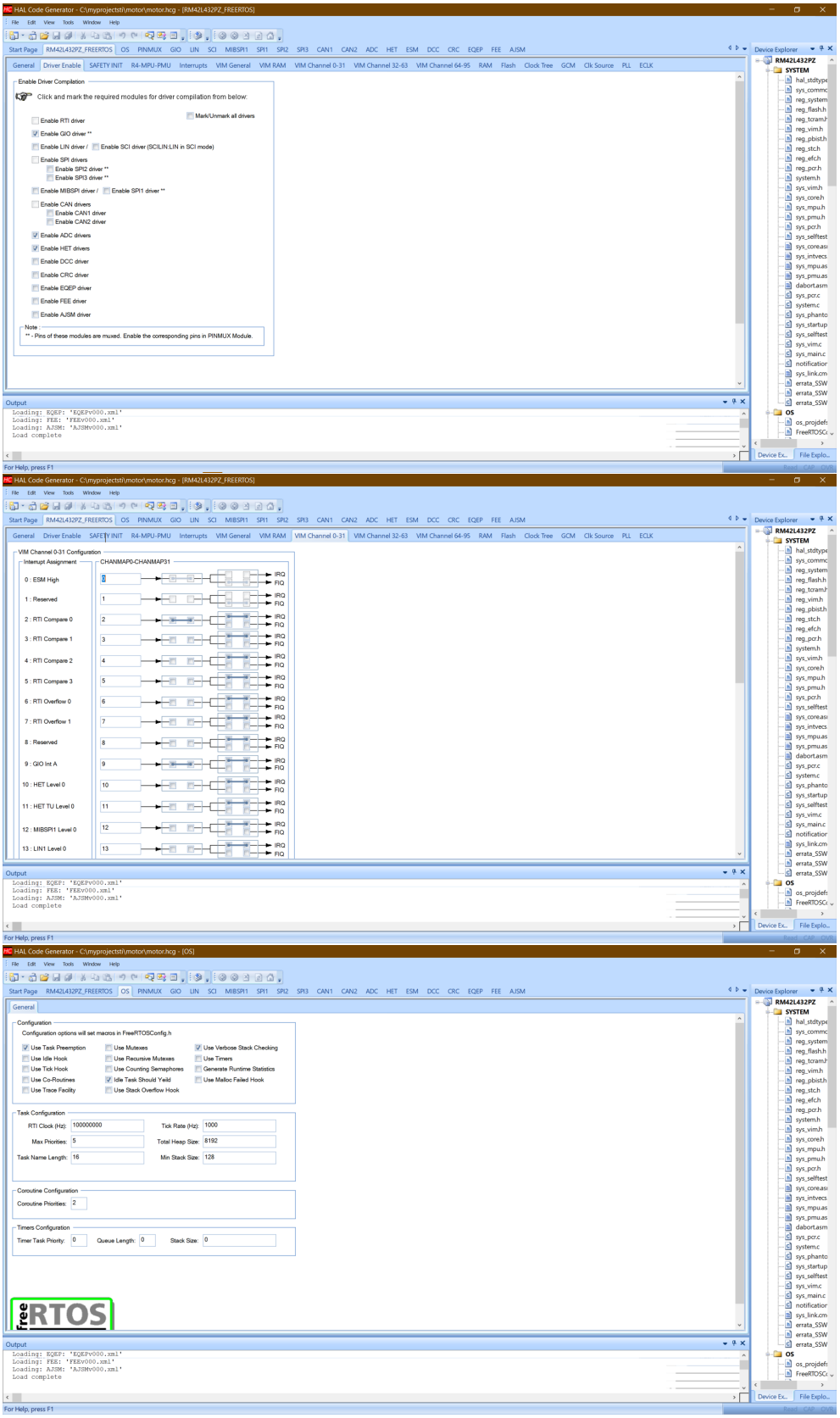


Figura 13

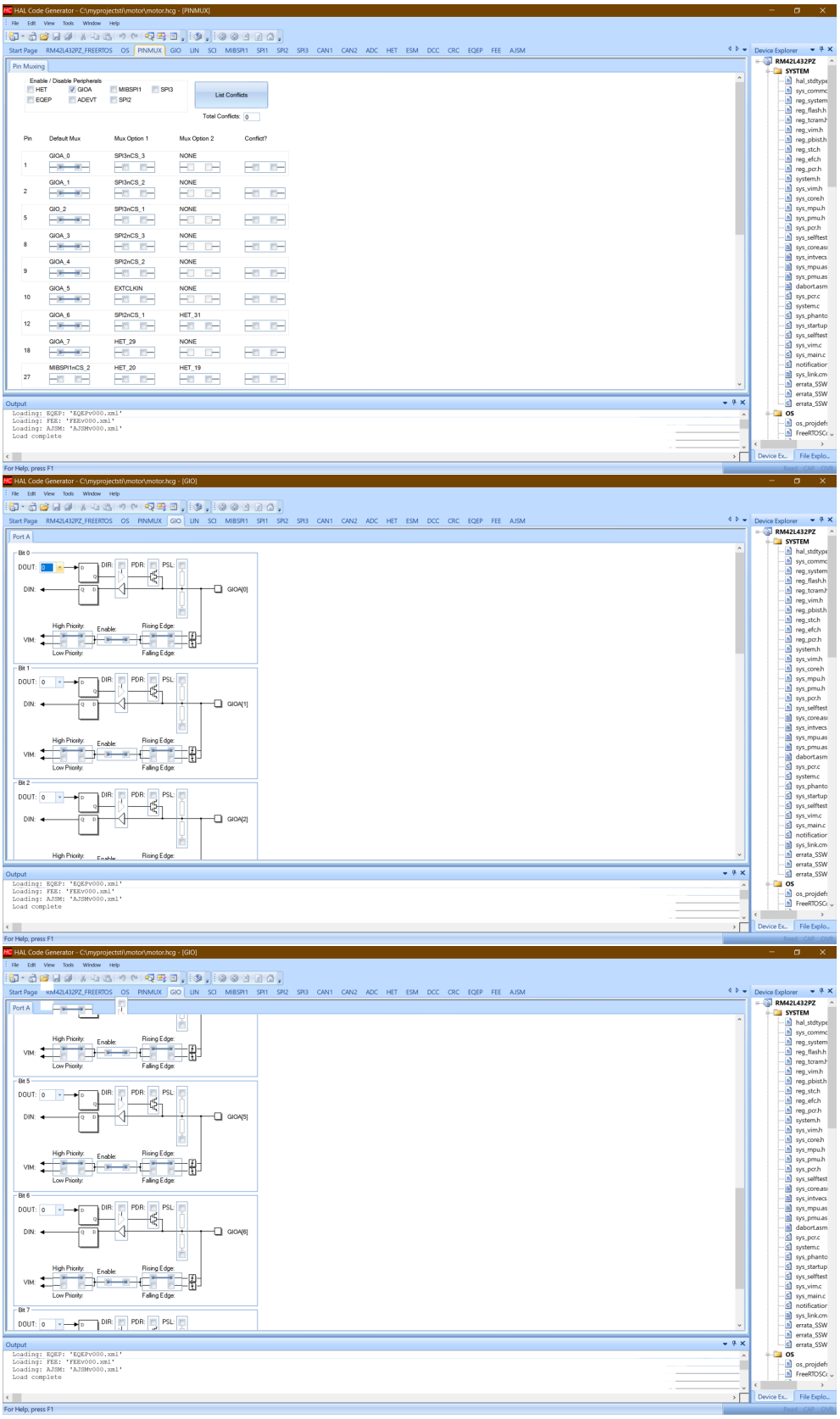

HAL Code Generator - C:\myprojectsti\motor\motor.hcg - [PINMUX]
Pin Muxing
Enable / Disable Peripherals
List Conflicts
Total Conflicts: 0
Pin
Default Mux
Mux Option 1
Mux Option 2
Conflict?
GIOA_0
SPI3nCS_3
NONE
GIOA_1
SPI3nCS_2
NONE
GIO_2
SPI3nCS_1
NONE
GIOA_3
SPI2nCS_3
NONE
GIOA_4
SPI2nCS_2
NONE
GIOA_5
EXTCLKIN
NONE
GIOA_6
SPI2nCS_1
HET_31
GIOA_7
HET_29
NONE
MIBSPI1nCS_2
HET_20
HET_19
HAL Code Generator - C:\myprojectsti\motor\motor.hcg - [GIO]
Port A
Bit 0
Bit 1
Bit 2
Bit 5
Bit 6
Bit 7
DOUT:
DIN:
DIR:
PDR:
PSL:
VIM:
High Priority:
Low Priority:
Enable:
Rising Edge:
Falling Edge:
GIOA[0]
GIOA[1]
GIOA[2]
GIOA[5]
GIOA[6]
Output
Loading: EQEP: 'EQEPv000.xml'
Loading: FEE: 'FEEv000.xml'
Loading: AJSM: 'AJSMv000.xml'
Load complete
For Help, press F1
Device Explorer
RM42L432PZ
SYSTEM
OS

Figura 14

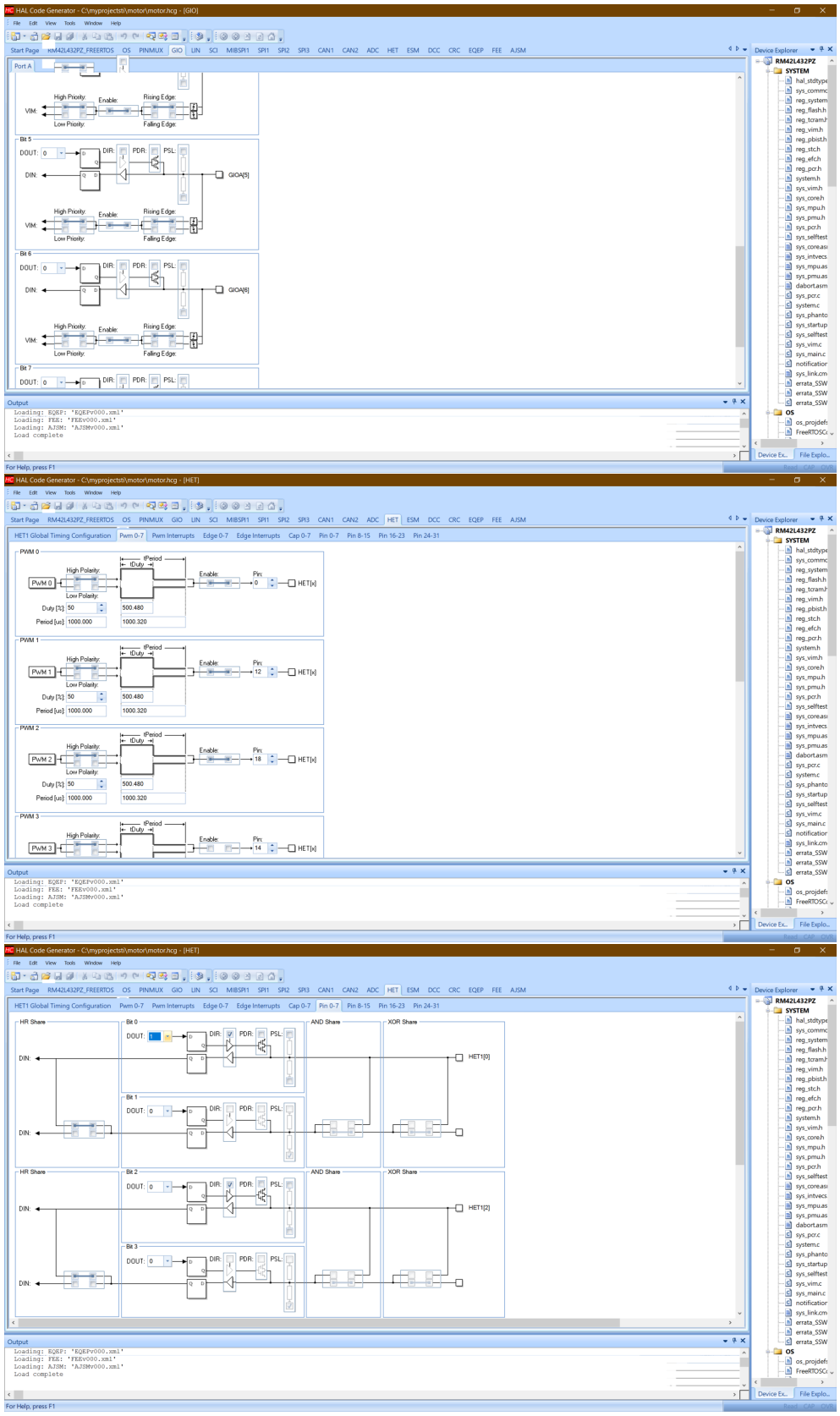
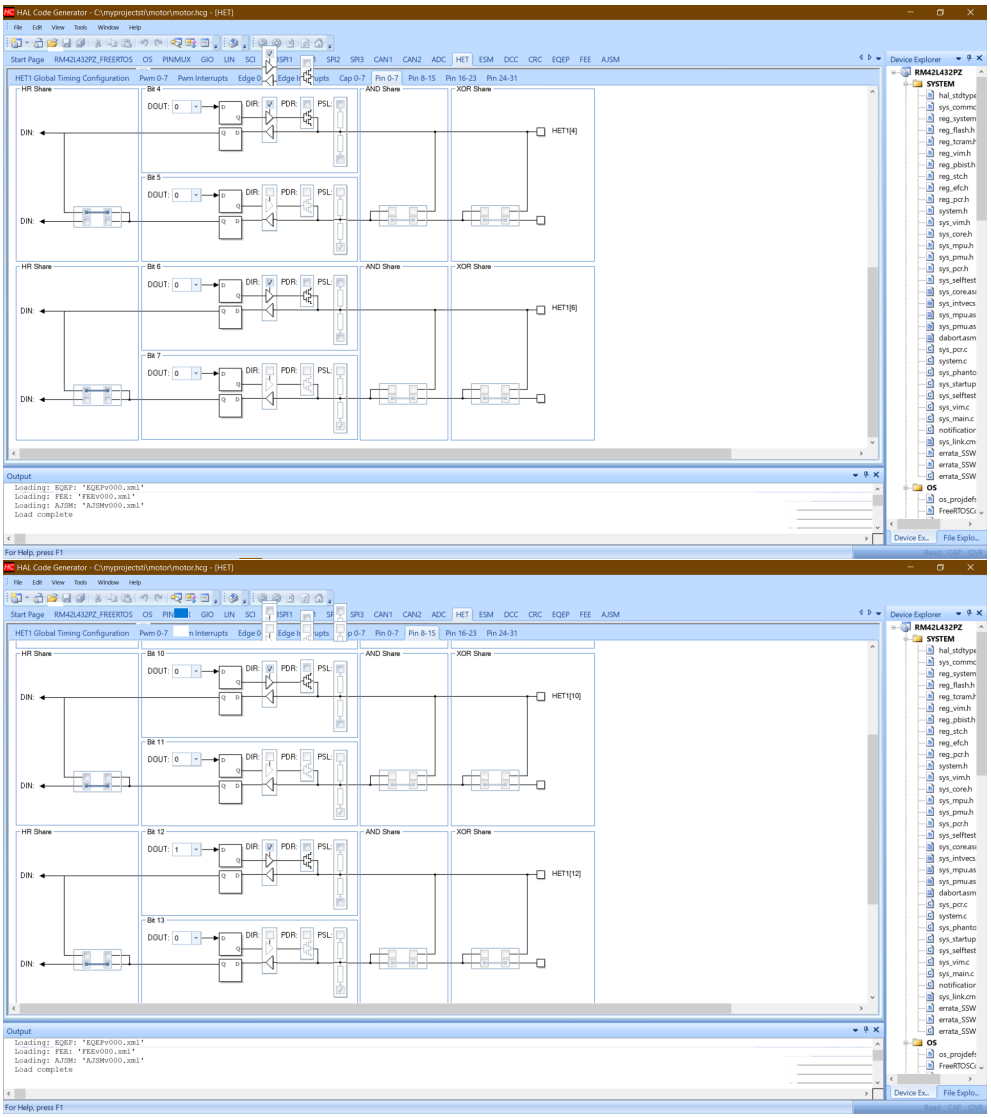

HAL Code Generator - C:\myprojects\ti\motor\motor.hcg - [GIO]
HAL Code Generator - C:\myprojects\ti\motor\motor.hcg - [HET]
HAL Code Generator - C:\myprojects\ti\motor\motor.hcg - [HET]
Output
Loading: EQEP: 'EQEPv000.xml'
Loading: FEE: 'FEEv000.xml'
Loading: AJSM: 'AJSMv000.xml'
Load complete
For Help, press F1

Figura 15


HAL Code Generator - C:\myprojectsti\motor\motor.hcg - [HET]
HR Share
AND Share
XOR Share
HET1[4]
HET1[6]
HET1[10]
HET1[12]
Output
Loading: EQEP: 'EQEPv000.xml'
Loading: FEE: 'FEEv000.xml'
Loading: AJSM: 'AJSMv000.xml'
Load complete
For Help, press F1

Bibliografía

[1] Delta Robot Kinematics - Tutorials. URL https://hypertriangle.com/~alex/delta-robot-tutorial/.

[2] FreeRTOS. FreeRTOS - market leading RTOS (Real time operating system) for embedded systems with internet of things extensions, 5 2023. URL https://www.freertos.org/index.html.

[3] Tinkersprojects. GitHub - Tinkersprojects/Delta-Kinematics-Library: Arduino Forward and Inverse Kinematics Library for Delta Robot. URL https://github.com/tinkersprojects/Delta-Kinematics-Library.