**Agregar un nuevo comando a los brazos**

(Tarjeta, Módulo, Blackboard).

**Tarjeta CM-700.**

Cada brazo tiene una tarjeta CM-700, la cual realiza todo el control de posición de los brazos, la tarjeta tiene un ciclo principal, en el cuál realiza varias cosas:

* Revisar si tiene un comando serial listo.
* Si el brazo recibió un comando que cambie la posición del brazo, ejecuta el control para llegar a esa posición (actualmente solo divide en pequeñas fracciones la diferencia entre la posición actual y la que se desea que tenga, entonces manda las distintas posiciones intermedias hasta llegar a la deseada).
* Si el brazo recibió la orden de cerrar el gripper, ejecuta el ciclo para cerrar el gripper (La manera en que esto funciona es poner los servos del gripper en modo de rotación continua en el sentido adecuado de giro para cerrar y con el porcentaje indicado del par total, en el rango de 0- 100%).

El código se encuentra repartido en 4 archivos principales:

**ArmsPic.c**: En este archivo se encuentra el ciclo principal en la función int Main(void), junto con el código necesario para la configuración del puerto serie y el llamado a las funciones que inicializan los servos y las constantes para su control.

**cmdATM.c**: En este archivo se encuentran todas las funciones relacionadas con la recepción, interpretación, captura de parámetros y ejecución de las distintas instrucciones seriales a las que responde la tarjeta, desde aquí es que se pone en alto la bandera **cmdReady** para que dentro del ciclo principal se pase a la ejecución del comando recibido, es desde aquí también que se ponen en alto las banderas **doControl** y **doGripper**.

**Calc.c**: Aquí se encuentra todo el código relacionado con la ejecución de las diversas instrucciones que toman acción directa sobre el movimiento y lectura de parámetros de los servomotores de los brazos. En este archivo vienen también la cinemática directa e inversa, y las funciones **executeControl** y **executeGripper.**

**servoControl.c**: En este archivo se encuentran las funciones de más bajo nivel, aquellas que lidian directamente con la manera en la que se envían los datos a los servomotores y la lectura de los distintos registros de los mismos. **NOTA IMPORTANTE[!] En esta clase se encuentran los sentidos de giro de los servos almacenados en el arreglo “cw”, es necesario que SIEMPRE coincidan con los que tiene el módulo ARMS en la clase Manipulator.**

Para **agregar un nuevo comando a la tarjeta CM-700**, primero es necesario establecer lo siguiente:

* Una letra que no haya sido usada por comandos anteriores, para llamar nuestro nuevo comando vía serial\*.
* El número de parámetros MÁXIMO que vayamos a utilizar para la instrucción\*\*.

\*Para checar que la letra que queremos usar no se encuentra actualmente en uso por otro comando, debemos revisar del archivo **cmdATM.c** enla función **execCommand** que en los distintos “case” de “switch(comando.cmd)” no se encuentre ya utilizada la nuestra.

\*\*Tomar en cuenta que el número máximo de parámetros que actualmente se puede recibir es 8, si se requieren más es necesario modificar un poco más el código o generar dos comandos distintos que utilicen menos parámetros.

Una vez establecidos los que vamos a utilizar, seguimos a agregarlos a la lista de comandos validos que la tajeta puede aceptar:

1. Agregamos :

case ‘nuestraLetra’:

nuestraFuncion();

break;

A la lista de posibles “case” del “switch(comando.cmd)” de la función ExecCommand() del archivo **cmdATM.c**

1. En base al número de parámetros que requiere nuestra función, buscamos en la función **fetchCommand()** del archivo **cmdATM.c**, el comentario que coincida con el número de parámetros que necesitamos recibir, si no se encuentra usar el valor más cercano hacia arriba, después de él vendrá un “if” con una serie de comparaciones unidas por “||” (or), a esta se le debe agregar la letra con la cual se identifica su comando por medio de la comparación “||” (or). A continuación viene el código ejemplo, indicando con color rojo lo que se debe agregar.

…

else if((buffer[0] == 'n')|| (buffer[0] == 'nuestraLetra')) //// 3 PARÁMETROS

{

cc = 1;

while(is\_alpha(buffer[cc]))++cc;

if(!fetchParam(&cc, &p1)) return 0;

if(!fetchParam(&cc, &p2)) return 0;

if(!fetchParam(&cc, &p3)) p3 = 0.0f;

if(buffer[cc] != 0) return 0;

cmdX(buffer[0], p1, p2, p3, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f);

return 1;

}

else if((buffer[0] == 'x')) //// 7 PARÁMETROS

. . . . . . . . . . . . . . . . . . . . . . . .

1. Una vez agregado nuestro comando a esas listas, es recomendable agregarlo a la lista de ayuda, que se encuentra en la función **showHelp()**, esta lista se muestra cuando se manda la letra “h” y un final de línea por el puerto serie.
2. Ya teniendo lista la recepción serial de nuestro comando, lo único restante es agregar nuestraFuncion() al archivo **Calc.c** que es el encargado de la ejecución de todo lo relacionado con los servos. Es necesario tomar en cuenta que los parámetros recibidos se almacenan en la estructura comando, en los valores comando.param1, comando.param2, comando.param3, comando.param4, comando.param5, comando.param6, comando.param7, comando.param8 y deben ser pasados a nuestraFuncion() como parámetros individualmente para poder ser utilizados. Ejemplo:

cmdATM.c :

…

execCommand()

{

…

case ‘k’:

miFuncion(comando.param1);

break;

…

//1 parámetro

else if(buffer[0] == 'k'||(buffer[0] == 'L')||(buffer[0] == 'R')||(buffer[0] == 'G')||(buffer[0] == 't') ||(buffer[0] == 'o')||(buffer[0] == 'c')||(buffer[0] == 'I')) //////// 1 PARÁMETRO

{

…

Calc.c:

…

void miFuncion(float miParametro)

{

//Lo que sea que yo quiera hacer

printf(“k 1\r”);

}

…

Es muy importante que la función regrese el carácter que representa el nuevo comando junto con un espacio, un 1 (éxito) o un 0 (fracaso) y un retorno de carro “\r”, para indicarle a la computadora si es que se puedo o no ejecutar el comando.

Una vez agregado el comando, conviene probarlo con el uso de una terminal serial, para corregir lo que sea necesario.

Para **cargar código propio a la tarjeta CM-700**, una vez compilado el código, abrimos con RoboPlus Terminal el COM al cual está conectada la tarjeta, corroboramos que sea el del brazo correcto presionan “h” y Enter, nos debe mostrar el menú de ayuda y en la parte superior dirá “LEFT ARM FIRMWARE” o “RIGHT ARM FIRMWARE” junto con la fecha de modificación del código que tienen cargado.

Ya que estamos seguros que es la tarjeta correcta, apagamos el switch que tienen integrados, mientras la terminal está seleccionada, desde el teclado mandamos el carácter “#” (Shift + 3) y encendemos la tarjeta.

Si lo hicimos bien, la terminal nos indicará que accedimos al BootLoader de la tarjeta, enviando la palabra “help” nos mostrará la ayuda del BootLoader, en ella encontraremos el comando “ld” que nos permitirá cargarle nuevo código. Una vez enviado dicho comando, presionamos Ctrl +T, lo que nos mostrará el diálogo para transmisión de archivos, nos vamos a la carpeta del proyecto y entramos a la carpeta “default” y ahí encontraremos el archivo “armsPic.hex”, lo seleccionamos y lo enviamos. Hecho esto, nos mostrará un mensaje de éxito si todo se hizo correctamente. Finalmente solo debemos apagar y encender nuevamente la tarjeta, y si mandamos el comando “h” y vemos que nuestro nuevo comando se encuentra en la lista del menú de ayuda, podremos comenzar a probarlo.

Una vez hecho todo esto y que vimos que el comando funciona bien, podemos seguir y agregar nuestro comando al módulo ARMS y posteriormente a BlackBoard.

**Módulo ARMS**

El módulo de brazos es muy complicado, pero más o menos funciona así:

* El módulo posee la calse ManipulatorManager, la cual se encarga de manejar todo lo relacionado con la conexión a BlackBoard, la ejecución de los comandos que recibe de él, el monitoreo y publicación de variables compartidas, así como del envío de comandos a las tarjetas de los brazos.
* La clase ManipulatorManager tiene dos elementos de la clase Manipulator, los cuales son rightArm y leftArm, los cuales se encargan del movimiento del brazo al que hacen referencia, así como una instancia de cada uno de los CommandExecuters, que se encargan de ejecutar un comando en específico cuando blackboard lo solicita y de enviar la respuesta adecuada, y las listas de posiciones y movimientos predefinidos para cada brazo. También tiene otra clase, que es muy importante, que es el TaskPlanner.
* La clase TaskPlanner se encarga de planear la ejecución de las tareas en uno o ambos brazos a la vez, así como de revisar que las cosas necesarias para su ejecución hayan sido dadas correctamente.
* La clase Manipulator se encarga de todas las funciones de cada brazo, la inicialización de sus variables y el envío de los comandos específicos a la tarjeta CM-700.

**NOTA IMPORTANTE [!]: En esta clase en encuentran establecidos los sentidos de giro(arreglo “senses”) y los centros de los servos (arreglo “centers”, es el valor correspondiente a 0 [rad], 0°), siempre que se modifiquen físicamente, se deben ajustar aquí los valores adecuados y corregirse en el código de la tarjeta por igual en servoControl.c .**

* Los CommandExecuters se encargan de recibir los comandos de blackboard, ejecutarlos y enviar respuesta, por cada comando que se tiene en brazos, es necesario tener un CommandExecuter.

Para **agregar un comando al módulo ARMS**, es necesario que el mismo ya esté programado en la tarjeta CM-700, y de preferencia ya probado y funcionando en esta sin ningún problema.

A continuación se muestra un orden sugerido para agregar el comando al módulo, pero puede hacerse como a uno le plazca:

1. Agregar un nuevo método público a la clase Manipulator, se recomienda que el nombre haga referencia a lo que hace:
   1. Al llamar nuestro método, es necesario tomar en cuenta los parámetros que necesitaremos enviar a la tarjeta CM-700, así como los valores que la misma nos regresará.
   2. Dentro de nuestro método es necesario generar en una cadena el comando específico al que sabemos que sabemos la tarjeta responde.
   3. Enviamos la cadena a través del serialPort que se estableció en el constructor de la clase.
   4. Esperamos la respuesta de la tarjeta.
2. Agregar un nuevo método público en el TaskPlanner:
   1. Este método es el que será llamado directamente por el commandExecuter específico a nuestro comando, por lo que en este se deben meter las cuestiones de seguridad para evitar enviar comandos cuando aun no está listo el brazo o se encuentra ejecutando otra acción.
   2. Para esto resultan muy útiles las variables movingLeftArm y movingRightArm, así como los valores dentro de la instancia armMan.status.
   3. Una vez que uno está seguro que el brazo no se encuentra ocupado y que en el módulo no existe ningún problema que pueda evitar que nuestro comando pueda ser ejecutado, se hace llamada a nuestro método en la instancia adecuada de Manipulator, ya sea leftArm o rightArm.
3. Generar un nuevo CommandExecuter (puede copiarse alguna de las clases más sencillas, como CmdGetVoltage para simplificar el proceso):
   1. Generar una nueva clase.
   2. Agregar “using Robotics.API” y “using Robotics.Controls”.
   3. La clase debe heredar del AsyncCommandExecuter. “class CmdArmsMove : AsyncCommandExecuter”
   4. Solicitar que implemente la clase, colocando el mouse sobre “AsyncCommandExecuter” nos mostrará la opción.
   5. En el constructor debe solicitarse que se le pase como parámetro un TaskPlanner. Igualmente debe agregársele el nombre del comando que solicita la herencia.

“

public CmdArmsMove(TaskPlanner taskPlanner)

: base("arms\_move")

{

…

“

* 1. Agregar la siguiente propiedad y cambiar el valor que regresa, de acuerdo a si nuestro comando requiere parámetros o no.

“

public override bool ParametersRequired

{

get

{

return true;

}

}

”

* 1. Al haber implementado la clase, nos generó el siguiente método

“

protected override Response AsyncTask(Command command)

{

throw new NotImplementedException();

}

“

En él deberá utilizarse la instancia de TaskPlanner que se pidió por parámetro para llamar al nuevo método que generamos en ella.

Al finalizar se deberá regresar a blackboard un elemento de la clase Response. La manera más sencilla de enviar una respuesta es generándola en base al comando original que recibimos, esto se hace con la siguiente línea:

“

return Response.CreateFromCommand(command, true);

“

Siendo el segundo parámetro un booleano que nos indique si se logró o no ejecutar el comando.

Si se necesita que el comando regrese parámetros, antes se puede generar un elemento de la clase Response, cambiar el valor de command.Parameters, que es una cadena, por los datos que queremos enviar de regreso.

“

Response resp;

…

command.Parameters = "miRespuesta";

resp = Response.CreateFromCommand(command, false);

return resp;

“

* 1. Una vez hecho esto solo falta agregar el Nuevo CommandExecuter a los commandExecuters del ManipulatorManager, esto se hace antes del constructor, después se debe generar una nueva instancia en el método SetupCommandExecuters().

...

private CmdArmsGoTo cmdArmsGoTo;

private CmdArmsMove cmdArmsMove;

private CmdArmsTorque cmdArmsTorque;

private CmdArmsState cmdArmsState;

private CmdGetVoltage cmdGetVoltage;

...

private void SetupCommandExecuters()

{

this.cmdArmsGoTo = new CmdArmsGoTo(this.taskPlanner);

this.cmdArmsMove = new CmdArmsMove(this.taskPlanner);

this.cmdArmsTorque = new CmdArmsTorque(this.taskPlanner);

this.cmdArmsState = new CmdArmsState(this.taskPlanner);

this.cmdGetVoltage = new CmdGetVoltage(this.taskPlanner);

...

* 1. Ya hecho esto, el comando queda listo para ser llamado desde Blackboard, sólo falta agregarlo a la lista de comando aceptados por Blackboard y agregarlo en los módulos a la lista de comandos por igual. Para agregarlo a la lista de Blackboard, debe buscarse el “xml” de configuración, buscar la parte del módulo ARMS y agregar un comando, siguiendo el mismo formato que tienen en el archivo. El archivo que generalmente se usa es “bbMark.xml”, para realizar pruebas se recomienda generar una copia, ya que sea exitosa, agregarlo a la versión que se esté utilizando.

<module name="**ARMS**">

<ip>**127.0.0.1**</ip>

<port>**2080**</port>

<simulate>**false**</simulate>

<aliveCheck>**true**</aliveCheck>

[**-**](file:///D:\Documentos\Trabajos\2013_2\Servicio\RobotApps\Blackboard\bbMark.xml) <commands>

<command name="**ra\_goto**" answer="**True**" timeout="**20000**" />

<command name="**ra\_abspos**" answer="**True**" timeout="**15000**" parameters="**false**" />

<command name="**ra\_artpos**" answer="**True**" timeout="**15000**" parameters="**false**" />

<command name="**ra\_getabspos**" answer="**True**" timeout="**5000**" parameters="**false**" />

<command name="**ra\_orientation**" answer="**True**" timeout="**10000**" parameters="**false**" />

<command name="**ra\_opengrip**" answer="**True**" timeout="**10000**" parameters="**false**" />

<command name="**ra\_closegrip**" answer="**True**" timeout="**10000**" parameters="**false**" />

<command name="**ra\_getgripstatus**" answer="**True**" timeout="**1000**" parameters="**false**" />

<command name="**ra\_torque**" answer="**True**" timeout="**1000**" />

<command name="**ra\_servotorqueon**" answer="**True**" timeout="**1000**" />

<command name="**ra\_relpos**" answer="**True**" timeout="**15000**" />

<command name="**ra\_move**" answer="**True**" timeout="**25000**" />

</commands>

</module>