Tarea #4 Movimiento de un robot

Materiales:

Arduino

Potenciometros

3 Motores a pasos (Ej. NEMA 17)

3 Drivers para los PaP (Ej. A4988, DRV8825, etc haciendo los cambios apropiados en el codigo)

3 Capacitores electroliticos (minimo de 47µF)

Algunos LED's

Fuente de poder (suficiente para alimentar el arduino y los PaP, ej. 12V 8.5A)

Arduino IDE y rosserial

Para la comunicación e intercambio de información entre arduino y ROS es necesario instalar arduino IDE y rosserial (*stack* de ROS que contiene el *package* rosserial_arduino con las librerias para arduino). Comenzaremos instalando el arduino IDE, para lo que ejecutaremos los siguientes comandos en un terminal:

```
sudo apt-get update
sudo apt-get install arduino arduino-core
```

Una vez realizada la instalación del software de arduino se procedera a la instalación del *package* de ROS ejecutando en un terminal el siguiente comando:

```
sudo apt-get install ros-NUESTRA_VERSIÓN_ROS-rosserial
```

Instalados arduino IDE y el *package* rosserial debemos copiar las librerias del *package* rosserial_arduino al sketchbook de arduino, carpeta que se encuentra habitualmente en la carpeta personal, para lo que ejecutaremos en un terminal los siguientes comandos:

```
roscd rosserial_arduino/libraries
cp -r ros_lib /home/”nombre_sesion”/sketchbook/libraries/ros_lib
```

Creación de un package para nuestros programas

Deberemos tener previamente creado un espacio de trabajo para nuestra versión de ROS. Para poder compilar y enviar nuestros programas a la placa arduino sin tener que pasar por arduino IDE vamos a crear un *package* llamado “Pract4.Arduino” con las dependencias necesarias para nuestros programas, para ello ejecutaremos los siguientes comandos en un terminal:

```
cd ~/ros_workspace
roscreate-pkg Pract4.Arduino rosserial_arduino std_msgs
```

Deberemos sustituir el contenido del fichero "CMakeLists.txt", situado en la carpeta del *package* creado, por el siguiente:

```
cmake_minimum_required(VERSION 2.4.6)
include($ENV{ROS_ROOT}/core/rosbuild/rosbuild.cmake)

rosbuild_find_ros_package(rosserial_arduino)
include(${rosserial_arduino_PACKAGE_PATH}/cmake_scripts/rosserial.cmake)
```

Para finalizar la craci ón del *package* ejecutaremos los siguientes comandos:

```
roscd Pract4.Arduino
cmake .
```

Primer programa (A toda potencia)

El primer programa que vamos a realizar crea un nodo llamado “potencia” que publica un *topic* llamado “cifra” y está suscrito a un *topic* llamado “resultado”. Cuando se publique un número en el *topic* “cifra” lo multiplicará por si mismo y publicará el resultado en el *topic* “resultado”. El código del programa es el siguiente:

```
#include <ros.h>
#include <std_msgs/Int16.h>

ros::NodeHandle nh;
int pot;

void potencia( const std_msgs::Int16& cifra){

::pot = cifra.data*cifra.data;
}

ros::Subscriber<std_msgs::Int16> sub("cifra", &potencia );
std_msgs::Int16 res;

ros::Publisher pub("resultado", &res);

void setup()
{
 nh.initNode();
 nh.subscribe(sub);
 nh.advertise(pub);
}

void loop()
{
 res.data = ::pot;
 pub.publish( &res );
 nh.spinOnce();
 delay(1000);
}
```

Para su compilación y envio a la placa arduino es necesario añadir al archivo "CMakeLists.txt" las siguientes líneas:

```
set(FIRMWARE_NAME potencia)
```

```
set(${FIRMWARE_NAME}_BOARD MODELO_NUESTRA_PLACA)   # Modelo placa arduino
set(${FIRMWARE_NAME}_SRCS src/potencia.cpp )
set(${FIRMWARE_NAME}_PORT /dev/ttyACM0)            # Puerto serie para carga
generate_ros_firmware(${FIRMWARE_NAME})
```

Donde se indica el nombre del programa, tipo de placa arduino (uno, atmega328 o mega2560), nombre y ubicación del archivo y puerto serie del PC empleado para la comunicación con la placa. Deberemos ejecutar los siguientes comandos en un terminal:

```
roscd Pract4.Arduino
make potencia-upload
```

Para la ejecución del programa se debe lanzar en un terminal el núcleo de ROS, si no se encuentra ya en marcha, ejecutando el siguiente comando:

```
roscore
```

En otro terminal se ejecutará el nodo que comunica a ROS con la placa arduino, indicándole el puerto empleado para la comunicación. Para saber el puerto que se está empleando, con la placa arduino conectada al pc a través de su puerto USB, podemos verlo en el menú "Herramientas">"Puerto serie" del software arduino IDE. En nuestro caso es el "ttyACM0":

```
rosrun rosserial_python serial_node.py /dev/ttyACM0
```

Para la publicación de un número en el *topic* “cifra” se ejecutará en otro terminal el siguiente comando:

```
rostopic pub cifra std_msgs/Int16 NUMERO_DESEADO --once
```

Para comprobar que realmente el programa calcula el cuadrado del número publicado en el *topic* “cifra” podemos ejecutar en otro terminal el siguiente comando para visualizar el *topic* “resultado”:

```
rostopic echo resultado
```

Arduino en Movimiento

Segundo programa, Usando Servomotores (opcional)

El programa publica el *topic* “angulo” con la posición del motor (entero entre 0 y 1023) y está suscrito al *topic* “giro” del que recibe la posición a la que debe moverse (entero entre 0 y 1023), con una velocidad constante de 128 (entero entre 0 y 1023). El código del programa es el siguiente:

```
#include <ros.h>
#include <std_msgs/Int16.h>
#include <ServomotorSerial1.h>
```

```
ros::NodeHandle nh;


void mover( const std_msgs::Int16& giro){

 Servomotor.moveSpeed(1,giro.data,128);

}

ros::Subscriber<std_msgs::Int16> sub("giro", &mover );

std_msgs::Int16 ang;
ros::Publisher pub("angulo", &ang);

void setup()
{
 nh.initNode();
 nh.subscribe(sub);
 nh.advertise(pub);
 Servomotor.begin(1000000,2);
 delay(1000);
}

void loop()
{
 int posicion = Servomotor.readPosition(1);
 ang.data = posicion;
 pub.publish( &ang );
 nh.spinOnce();
 delay(10);
}
```

Para poder compilar y enviar el programa a la placa arduino deberemos emplear el software arduino IDE, ya que empleamos una librería externa al sistema ROS y no nos permite hacerlo usando nuestro *package*. Una vez transferido el programa a la placa, para probar el programa debemos ejecutar en un terminal el siguiente comando, que inicia el nucleo de ROS:

```
roscore
```

También debemos lanzar en otro terminal el nodo de comunicación ROS-arduino, ejecutando el siguiente comando:

```
rosrun rosserial_python serial_node.py /dev/ttyACM0
```

En un nuevo terminal publicaremos en el *topic* "giro" la posición de destino del servomotor, ejecutando el siguiente comando:

```
rostopic pub giro std_msgs/Int16 POSICION_DESEADA --once
```

El servomotor de inmediato comenzará a moverse hasta la posición indicada. Podemos visualizar el valor de posición del servomotor en todo momento ejecutando en otro terminal el siguiente comando:

```
rostopic echo angulo
```

Segundo programa (Muévete usando PaP, no es opcional)

En este programa simplemente se realiza un prueba de la comunicación entre la placa arduino con un Motor a pasos.

Entrar al ambiente ROS

```
source /opt/ros/<distro>/setup.bash
```

Inicializar ROS `roscore`

Programar el arduino con el siguiente codigo

```
#include <ros.h>

#include <Arduino.h>
#include <std_msgs/Empty.h>
#include <std_msgs/Int8.h>
#include "A4988.h"


// using a 200-step motor (most common)
#define MOTOR_STEPS 200
// configure the pins connected
#define STEP1 3
#define DIR1 4
#define STEP2 5
#define DIR2 6
#define STEP3 7
#define DIR3 8


ros::NodeHandle nh;


A4988 stepper1(MOTOR_STEPS, DIR1, STEP1);
A4988 stepper2(MOTOR_STEPS, DIR2, STEP2);
A4988 stepper3(MOTOR_STEPS, DIR3, STEP3);


// TODO: build custom ROS message which will contain Int8 rpm
// also Int8 microstep and Int8 degrees
void startMotor1(const std_msgs::Int8& rpm){
// blink the led, to indicate motor1 has started
digitalWrite(0, HIGH);
stepper1.begin(rpm.data, 16);
stepper1.rotate(360);
digitalWrite(0, LOW);
}


void startMotor2(const std_msgs::Empty& toggle_msg){
```

```
// blink the blue led, to indicate motor2 has started
digitalWrite(1, HIGH);
stepper2.begin(15, 16);
stepper2.rotate(360);
digitalWrite(1, LOW);
}

void startMotor3(const std_msgs::Empty& toggle_msg){
// blink the red led, to indicate motor3 has started
digitalWrite(2, HIGH);
stepper3.begin(15, 16);
stepper3.rotate(360);
digitalWrite(2, LOW);
}

ros::Subscriber<std_msgs::Int8> motor1("motor1/start", &startMotor1);
ros::Subscriber<std_msgs::Empty> motor2("motor2/start", &startMotor2);
ros::Subscriber<std_msgs::Empty> motor3("motor3/start", &startMotor3);

void setup() {
pinMode(0, OUTPUT);
pinMode(1, OUTPUT);
pinMode(2, OUTPUT);
nh.initNode();
nh.subscribe(motor1);
nh.subscribe(motor2);
nh.subscribe(motor3);
}

void loop() {
nh.spinOnce();
delay(1);
}
```

(no olvides que debe estar el rosserial_arduino instalado), cargarlo en el arduino y ejecutarlo

Abriendo otra terminal, ejecutar `rosrun rosserial_python serial_node.py /dev/ttyACM0` .

En otra terminal, interactuar con el programa usando ROS topics. e.g. `rostopic pub motor2/start std_msgs/Empty --once` ejecutara una vez el siguiente codigo:

`void startMotor2(const std_msgs::Empty& toggle_msg)`

mientras `rostopic pub motor1/start std_msgs/Int8 15 --once` ejecutara una vez el siguiente codigo:

`void startMotor1(const std_msgs::Int8& rpm)`

Al pasar como argumento una rpm de 15

Puedes ver todos los temas disponibles ejecutando en un terminal `rostopic list`

Modelo cinemático del brazo

Para el correcto posicionamiento del brazo es necesario obtener las transformaciones geométricas para determinar el ángulo que deben girar los servomotores para alcanzar un punto (cinemática inversa). Al tratarse de un brazo que trabaja en un solo plano con giro (3 articulaciones) es un proceso sencillo que se uede realizadar de forma directa, sino se puede recurrir al algoritmo de parametrización Denavit- Hartenberg que permite obtener la cinemática directa y a partir de esta obtener la inversa.

Diagramas para el estudio de la cinemática

Cinemática inversa

Siguiendo los diagramas planteados y considerando *x*,*y* y *z* como las coordenadas del punto destino, y ε el ángulo que forma la pinza respecto al plano x-z, las ecuaciones obtenidas para la cinemática inversa son las siguientes:

Brazo ecuaciones.jpg

Teniendo en cuenta los ángulos límite de los servomotores y el rango de valores que debemos usar, los valores que debemos pasar a los servomotores de cada articulación son los siguientes:

Diagrama de ángulos servomotor

Brazo ecuacion base.jpg Se suman 150º al ángulo para situar el 0º coincidente con el eje z.

Brazo ecuacion arti1.jpg Se suman 60º al ángulo para situar el 0º coincidente con el eje z.

Brazo ecuacion arti2.jpg Se restan 30º al ángulo para situar el 0º coincidente con el eje longitudinal del servomotor.

Brazo ecuacion arti3.jpg Se restan 30º al ángulo para situar el 0º coincidente con el eje longitudinal del servomotor.

Para enviar a los servomotores valores dentro del rango de trabajo, se multiplica por 1023 (0x3FF), valor máximo del rango de posiciones del servo, y se divide por 300 (300º), el ángulo total de giro del servomotor.

Cinemática directa

A partir de las ecuaciones anteriores podemos plantear también el proceso inverso, la obtención del punto en que se encuentra el brazo a partir de las posiciones de los servomotores. Las ecuaciones obtenidas son las siguientes:

Brazo ecuaciones directa.jpg

Espacio de trabajo y limitaciones

Para evitar colisiones del brazo con las diferentes partes del robot y con sus propios componentes, es necesario fijar unos límites de giro a los servomotores y restringir el acceso a regiones del espacio. Para esta tarea se puede auxiliar de una hoja de calculo o usando Matlab con las formulas de la cinemática inversa y representar en unos gráficos sencillos las posiciones de los ejes y las articulaciones del brazo en el espacio.

Programas de control

Tercer programa (Ven aquí)

Primero crearemos un programa que actuará de intermediario entre el programa de control y los servomotores del brazo, que situaremos en la placa arduino. Este programa se encargará de enviar las ordenes recibidas a los servomotores y enviar los datos más relevantes de los servomotores. Como se van a *publicar* y *suscribir* los datos de 5 servomotores, vamos a comenzar creando nuestros *mensajes personalizados*.

Como crear un *package*

En el directorio de trabajo creado se creará un *package*, el cual va a depender de otros *packages* del sistema. Las dependencias van en función del tipo de mensajes que queremos enviar o recibir, y el lenguaje empleado en los programas. En este primer programa no se necesitan todas las dependencias, pero se usaran en los siguientes programas, por lo que las añadiremos al crear el *package* que va a contener los programas. Para esto se introducirá la siguiente secuencia de comandos en un terminal:

```
cd ~/ros_workspace
roscreate-pkg turtlebot std_msgs turtlebot_node geometry_msgs nav_msgs
sensor_msgs image_transport roscpp
```

También se pueden añadir dependencias a un *package* ya creado, simplemente se debe editar el archivo "manifest.xml" que se encuentra dentro de la carpeta del *package* creado. El contenido del archivo "manifest.xml" es similar al siguiente, las dependencias se encuentran al final:

```
<package>
  <description brief="turtlebot">

     turtlebot

  </description>
  <author>Ustedes</author>
  <license>BSD</license>
  <review status="unreviewed" notes=""/>
  <url>http://ros.org/wiki/turtlebot_fer</url>
  <depend package="turtlebot_node"/>
  <depend package="geometry_msgs"/>
  <depend package="nav_msgs"/>
  <depend package="std_msgs"/>
  <depend package="sensor_msgs"/>
  <depend package="image_transport"/>
  <depend package="roscpp"/>

</package>
```

Una vez creado el *package*, o modificadas las dependencias de un *package* existente, se debe compilar para que se hagan efectivas las dependencias. Simplemente se debe ejecutar el siguiente comando en el terminal, indicando el nombre del *package*:

```
rosmake turtlebot_fer
```

Creación de mensajes personalizados en ROS

Previamente creamos un package con las dependencias necesarias para nuestros programas (std_msgs geometry_msgs roscpp). Dentro de este *package* crearemos una carpeta llamada "msg", donde colocaremos los siguientes archivos con la descripción de los mensajes (tipo de dato y nombre de campo).

Comenzaremos creando un mensaje base llamado "servos", que contendrá los datos de cada servomotor. Crearemos un archivo llamado "Servos.msg" con el siguiente contenido:

```
int16 base
int16 arti1
int16 arti2
int16 arti3
int16 pinza
```

Vamos a crear un mensaje que contenga los datos que enviaremos a los servomotores (posición, velocidad y par). Ahora el tipo de dato que vamos a manejar es el que hemos creado anteriormente. Creamos un archivo llamado "WriteServos.msg" con el siguiente contenido:

```
Servos posicion
Servos velocidad
Servos par
```

Para la recepción de datos de los servomotores (posición, estado y corriente) crearemos un archivo llamado "ReadServos.mag" con el siguiente contenido:

```
Servos posicion
Servos estado
Servos corriente
```

Para permitir la creación de los mensajes en el sistema ROS debemos editar el archivo "CMakeLists.txt" eliminando el símbolo "#" para que deje de estar comentada la siguiente línea:

```
# rosbuild_gensrv()
```

Para la compilación y creación de los mensaje ejecutaremos la siguiente secuencia de comandos en un terminal, donde "brazo" es el package creado para nuestros programas:

```
roscd brazo
make
```

Como vamos a usar estos mensajes en arduino, necesitamos añadirlos al *package* "rosserial", ejecutando en un terminal el siguiente comando:

```
rosrun rosserial_client make_library.py ~/sketchbook/libraries brazo
```

Programa arduino (Nota: sustituir en los codigos el uso de servomotores por el uso de PaP para la entrega de esta tarea)

El programa para la placa arduino está suscrito al *topic* "move_arm", por el cual recibe los movimientos de los servomotores, y publica el *topic* "pose_arm", con la información de posición, par y corriente de los servomotores. El código del programa es el siguiente:

```
#include <ros.h>
#include <brazo/Servos.h>
#include <brazo/WriteServos.h>
#include <brazo/ReadServos.h>
#include <std_msgs/Bool.h>
#include <math.h>

#include <ServomotorSerial1.h>

ros::NodeHandle nh;

void mover(const brazo::WriteServos& brazo){

 brazo::Servos par = brazo.par;

 brazo::Servos vel = brazo.velocidad;

 brazo::Servos move = brazo.posicion;

 int posicion[4];

 if (par.base == 0){
   Servomotor.torqueStatus(1,OFF);
```

```
 }
 else {
   Servomotor.torqueStatus(1,ON);

   Servomotor.moveSpeed(1,move.base,vel.base);
 }

 if (par.arti1 == 0){
   Servomotor.torqueStatus(2,OFF);
 }
 else {
   Servomotor.torqueStatus(2,ON);

   Servomotor.moveSpeed(2,move.arti1,vel.arti1);
 }

 if (par.arti2 == 0){
   Servomotor.torqueStatus(3,OFF);
 }
 else {
   Servomotor.torqueStatus(3,ON);

   Servomotor.moveSpeed(3,move.arti2,vel.arti2);
 }

 if (par.arti3 == 0){
   Servomotor.torqueStatus(4,OFF);
 }
 else {
   Servomotor.torqueStatus(4,ON);

   Servomotor.moveSpeed(4,move.arti3,vel.arti3);
 }

}

void pinza(const brazo::WriteServos& pinza){

 brazo::Servos par = pinza.par;

 brazo::Servos vel = pinza.velocidad;

 brazo::Servos move = pinza.posicion;

 int posicion;

 if (par.pinza == 0){
   Servomotor.torqueStatus(5,OFF);
 }
 else {
   Servomotor.torqueStatus(5,ON);

   Servomotor.moveSpeed(5,move.pinza,vel.pinza);
 }

}

ros::Subscriber<brazo::WriteServos> move_sub("move_arm", &mover );
ros::Subscriber<brazo::WriteServos> hand_sub("hand_arm", &pinza );

brazo::ReadServos pec;
std_msgs::Bool pulsador;

ros::Publisher pose_pub("pose_arm", &pec);
```

```
void setup()
{
 nh.initNode();
 nh.subscribe(move_sub);
 nh.subscribe(hand_sub);
 nh.advertise(pose_pub);

 Servomotor.begin(1000000,2);

}

void loop()
{
 brazo::Servos pos;
 brazo::Servos est;
 brazo::Servos cor;

 pos.base = Servomotor.readPosition(1);
 pos.arti1 = Servomotor.readPosition(2);
 pos.arti2 = Servomotor.readPosition(3);
 pos.arti3 = Servomotor.readPosition(4);
 pos.pinza = Servomotor.readPosition(5);

 est.base = Servomotor.moving(1);
 est.arti1 = Servomotor.moving(2);
 est.arti2 = Servomotor.moving(3);
 est.arti3 = Servomotor.moving(4);
 est.pinza = Servomotor.moving(5);

 cor.base = Servomotor.readLoad(1);
 cor.arti1 = Servomotor.readLoad(2);
 cor.arti2 = Servomotor.readLoad(3);
 cor.arti3 = Servomotor.readLoad(4);
 cor.pinza = Servomotor.readLoad(5);

 pec.posicion = pos;
 pec.estado = est;
 pec.corriente = cor;

 pose_pub.publish( &pec );

 nh.spinOnce();

}
```

Para compilar y cargar el programa en la placa emplearemos el programa arduino IDE. Este programa nos servirá para otros programas de control que desarrollemos, ya que se trata de un mero intermediario.

Programa ROS

Usando las ecuaciones de la cinemática inversa este programa indica a los servomotores la posición de giro para alcanzar el punto de destino que se encuentre dentro del espacio de trabajo. Está suscrito al *topic* "point", por el cual recibe el punto del espacio en el que tiene que posicionarse, evalúa la viabilidad y obra en consecuencia, tambien esta suscrito al *topic* "pose_arm", donde recibe la información de los servomotores que publica el programa de arduino, y publica el *topic* "move_arm", donde indica al programa de arduino la posición, velocidad y par de los servomotores.

Dentro del *pakage* creado, en el directorio "src" crearemos un fichero llamado "control_v01.cpp" con el siguiente contenido:

```
#include "ros/ros.h"
#include "brazo/Servos.h"
#include "brazo/WriteServos.h"
#include "brazo/ReadServos.h"
#include "geometry_msgs/Point.h"
#include "math.h"

#define PI 3.14159265
#define L1 104
#define L2 104
#define Lp 60


brazo::Servos move, vel, p, e, c;

void posicion(const brazo::ReadServos& pose)
{

  ::p = pose.posicion;
  ::e = pose.estado;
  ::c = pose.corriente;

}

void punto(const geometry_msgs::Point& point)
{
      ros::NodeHandle n;

      ros::Publisher move_pub_=n.advertise<brazo::WriteServos>("move_arm", 1);

      double x = point.x;
      double y = point.y;
      double z = point.z;

      y = y + z*tan(atan2(45,250));

      int coordenadas_correctas = 1;

         double alfa, beta, beta_a, beta_p, beta_pp, gamma, delta, delta_a,
delta_p, epsilon;
      double z_p;
      double L_a, L;

      epsilon = 0; //Ángulo de inclinación de la pinza respecto a la horizontal

      alfa = (atan2(x,z)*180)/PI;

      z_p = sqrt(pow(z,2)+pow(x,2));

      L = sqrt(pow(z_p,2)+pow(y,2));

      L_a = sqrt(pow(y+(Lp*sin(epsilon)),2)+pow(z_p-(Lp*cos(epsilon)),2));

      beta_p = atan2(y+(Lp*sin(epsilon)),z_p-(Lp*cos(epsilon)));

      beta_pp = atan2(y,z_p);

      beta_a = acos((pow(L1,2)+pow(L_a,2)-pow(L2,2))/(2*L1*L_a));

      beta = ((beta_p+beta_a)*180)/PI;
```

```
	gamma = acos((pow(L1,2)+pow(L2,2)-pow(L_a,2))/(2*L1*L2));

	delta_a = PI-(beta_a+gamma);

	gamma = (gamma*180)/PI;

	delta_p = acos((pow(L_a,2)+pow(Lp,2)-pow(L,2))/(2*L_a*Lp));

	if (beta_pp > beta_p) {
		delta = ((2*PI-(delta_p-delta_a))*180)/PI;
	}
	else {
		delta = ((delta_p+delta_a)*180)/PI;

		if (isnan(delta)) {
			delta = ((PI+delta_a)*180)/PI;
		}
	}

	if (isnan(gamma)) // si no hay solución
	{
		coordenadas_correctas = 0;
	}

	::move.base = ((alfa+150)*1023)/300;
	::move.arti1 = ((beta+60)*1023)/300;
	::move.arti2 = ((gamma-30)*1023)/300;
	::move.arti3 = ((delta-30)*1023)/300;
	::move.pinza = 511;

	::vel.base = abs(::move.base - ::p.base)/5;
	::vel.arti1 = abs(::move.arti1 - ::p.arti1)/5;
	::vel.arti2 = abs(::move.arti2 - ::p.arti2)/5;
	::vel.arti3 = abs(::move.arti3 - ::p.arti3)/5;
	::vel.pinza = abs(::move.pinza - ::p.pinza);

	 if (coordenadas_correctas == 1 && (205 <= ::move.base && ::move.base <=
818) && (120 <= ::move.arti1 && ::move.arti1 <= 920) && ::move.arti2 >= 50
&& ::move.arti3 <= 828) {

		brazo::WriteServos mover;

		mover.posicion = ::move;

		mover.velocidad = ::vel;

		mover.par.base = 1;
		mover.par.arti1 = 1;
		mover.par.arti2 = 1;
		mover.par.arti3 = 1;
		mover.par.pinza = 1;

		move_pub_.publish(mover);
	}
	else {
			std::cout<<"error coordenadas no validas o punto fuera del
alcance"<<std::endl;
	}

 }


 int main(int argc, char **argv)
```

```
{
	ros::init(argc, argv, "control_brazo");
	ros::NodeHandle n;

	ros::Subscriber pose_sub_= n.subscribe("pose_arm", 1, posicion);
	ros::Subscriber point_sub_= n.subscribe("point", 1, punto);
	ros::Publisher move_pub_=n.advertise<brazo::WriteServos>("move_arm", 1);

	ros::spin();
	return 0;
}
```

Para compilarlo y generar el ejecutable se debe añadir la siguiente línea de código al archivo "CMakeLists.txt" del *package* creado, donde indicamos el nombre para el ejecutable y la ruta y nombre del archivo a compilar:

```
rosbuild_add_executable(control_v01 src/control_v01.cpp)
```

Para compilar el programa hay que situarse en el directorio del *package*. Simplemente con ejecutar la siguiente secuencia de comandos en un terminal se compilará y creará el ejecutable, siempre que no existan errores:

```
roscd brazo
make
```

Ejecutando el programa

Para ejecutar el programa debemos lanzar el núcleo de ROS, si no se encuentra iniciado, ejecutando en un terminal el siguiente comando:

```
roscore
```

Ahora debemos iniciar el nodo de la placa arduino ejecutando en un nuevo terminal el siguiente comando:

```
rosrun rosserial_python serial_node.py /dev/ttyACM0
```

Para ejecutar el programa de control llamado "control_v01", en un terminal nuevo ejecutaremos el siguiente comando:

```
rosrun brazo control_v01
```

Para poder publicar las coordenadas del punto al que queremos que se desplace el brazo, debemos publicar en el *topic* "point", al que esta suscrito el programa de control para recibir el punto. Ejecutaremos en un nuevo terminal el siguiente comando:

```
rostopic pub place_point geometry_msgs/Point '{x: 50, y: 100, z: 150}' --once
```

El brazo comenzará a desplazarse y se detendrá al llegar al punto que le hemos indicado.

Cuarto programa (Agarra la botella, si no tienes pinza, usa un gancho)

Archivo include con funciones

Con el objetivo de simplificar y hacer más legibles los programas se ha creado este archivo que contiene las funciones de control del brazo. Crearemos un archivo llamado "brazo.h" dentro del directorio "include/brazo" del *package* creado con el siguiente contenido:

```
#ifndef brazo_H
#define brazo_H

 #include "ros/ros.h"
 #include "brazo/Servos.h"
 #include "brazo/WriteServos.h"
 #include "brazo/ReadServos.h"
 #include "std_msgs/Int16.h"
 #include "geometry_msgs/Point.h"
 #include "math.h"

 #define PI 3.14159265
 #define L1 104
 #define L2 104
 #define Lp 60

geometry_msgs::Point      directa(brazo::Servos      posicion_servos_0,      int
inclinacion_pinza)
 {

          double alfa, beta, beta_a, beta_p, beta_pp, gamma, delta, delta_a,
delta_p1, delta_p2, epsilon;
       double z_p;
       double L_a, L;
       double a, b;
       double x, y, z;

       alfa = (((posicion_servos_0.base*300)/1023)-150)*PI/180;
       beta = (((posicion_servos_0.arti1*300)/1023)-60)*PI/180;
       gamma = (((posicion_servos_0.arti2*300)/1023)+30)*PI/180;
       delta = (((posicion_servos_0.arti3*300)/1023)+30)*PI/180;

       epsilon = (inclinacion_pinza*PI)/180;

       L_a = sqrt(pow(L1,2)+pow(L2,2)-2*L1*L2*cos(gamma));

       beta_a = acos((pow(L1,2)+pow(L_a,2)-pow(L2,2))/(2*L1*L_a));

       beta_p = beta - beta_a;

       delta_a = PI - (beta_a + gamma);

       delta_p1 = delta - delta_a;

       delta_p2 = 2*PI - (delta - delta_a);

       if (delta_p1 < delta_p2)
       {
              L = sqrt(pow(L_a,2)+pow(Lp,2)-2*L_a*Lp*cos(delta_p1));
       }
       else
       {
```

```
                L = sqrt(pow(L_a,2)+pow(Lp,2)-2*L_a*Lp*cos(delta_p2));
        }

        z_p = L_a*cos(beta_p) + Lp*cos(epsilon);

        beta_pp = acos(z_p/L);

        a = L1*sin(beta);

        b = L2*sin(beta+gamma)+Lp*sin(epsilon);

        if (a >= b)
        {
                y = L*sin(beta_pp);
        }
        else
        {
                y = -L*sin(beta_pp);
        }

        z = z_p*cos(alfa);

        x = z_p*sin(alfa);

        geometry_msgs::Point punto;

        punto.x = x;
        punto.y = y;
        punto.z = z;

        return punto;
 }

brazo::WriteServos inversa(geometry_msgs::Point destino, int inclinacion_pinza,
brazo::Servos posicion_servos_0, int velocidad)
 {

        double x = destino.x;
        double y = destino.y;
        double z = destino.z;

        int coordenadas_correctas = 1;

           double alfa, beta, beta_a, beta_p, beta_pp, gamma, delta, delta_a,
delta_p, epsilon;
        double z_p;
        double L_a, L;

        epsilon = (inclinacion_pinza*PI)/180;

        alfa = (atan2(x,z)*180)/PI;

        z_p = sqrt(pow(z,2)+pow(x,2));

        L = sqrt(pow(z_p,2)+pow(y,2));

        L_a = sqrt(pow(y+(Lp*sin(epsilon)),2)+pow(z_p-(Lp*cos(epsilon)),2));

        beta_p = atan2(y+(Lp*sin(epsilon)),z_p-(Lp*cos(epsilon)));

        beta_pp = atan2(y,z_p);

        beta_a = acos((pow(L1,2)+pow(L_a,2)-pow(L2,2))/(2*L1*L_a));
```

```
	beta = ((beta_p+beta_a)*180)/PI;

	gamma = acos((pow(L1,2)+pow(L2,2)-pow(L_a,2))/(2*L1*L2));

	delta_a = PI-(beta_a+gamma);

	gamma = (gamma*180)/PI;

	delta_p = acos((pow(L_a,2)+pow(Lp,2)-pow(L,2))/(2*L_a*Lp));

	if (beta_pp > beta_p) {
		delta = ((2*PI-(delta_p-delta_a))*180)/PI;
	}
	else {
		delta = ((delta_p+delta_a)*180)/PI;

		if (isnan(delta)) {
			delta = ((PI+delta_a)*180)/PI;
		}
	}


	if (isnan(gamma))
	{
		coordenadas_correctas = 0;
	}

	brazo::Servos posicion_servos_1;
	brazo::Servos velocidad_servos;
	brazo::Servos par_servos;

	posicion_servos_1.base = ((alfa+150)*1023)/300;
	posicion_servos_1.arti1 = ((beta+60)*1023)/300;
	posicion_servos_1.arti2 = ((gamma-30)*1023)/300;
	posicion_servos_1.arti3 = ((delta-30)*1023)/300;

	if (velocidad == 0)
	{
		velocidad_servos.base = abs(posicion_servos_1.base - posicion_servos_0.base)/5;
		if (velocidad_servos.base > 1023){velocidad_servos.base = 1023;}
		else if (velocidad_servos.base < 10){velocidad_servos.base = 10;}
		velocidad_servos.arti1 = abs(posicion_servos_1.arti1 - posicion_servos_0.arti1)/5;
		if (velocidad_servos.arti1 > 1023){velocidad_servos.arti1 = 1023;}
		else if (velocidad_servos.arti1 < 10){velocidad_servos.arti1 = 10;}
		velocidad_servos.arti2 = abs(posicion_servos_1.arti2 - posicion_servos_0.arti2)/5;
		if (velocidad_servos.arti2 > 1023){velocidad_servos.arti2 = 1023;}
		else if (velocidad_servos.arti2 < 10){velocidad_servos.arti2 = 10;}
		velocidad_servos.arti3 = abs(posicion_servos_1.arti3 - posicion_servos_0.arti3)/5;
		if (velocidad_servos.arti3 > 1023){velocidad_servos.arti3 = 1023;}
		else if (velocidad_servos.arti3 < 10){velocidad_servos.arti3 = 10;}

	}
	else
	{
```

```
                        velocidad_servos.base  =  abs(posicion_servos_1.base  -
posicion_servos_0.base)*(velocidad/10);
               if (velocidad_servos.base > 1023){velocidad_servos.base = 1023;}
               else if (velocidad_servos.base < 10){velocidad_servos.base = 10;}
                        velocidad_servos.arti1  =  abs(posicion_servos_1.arti1  -
posicion_servos_0.arti1)*(velocidad/10);
                  if (velocidad_servos.arti1 > 1023){velocidad_servos.arti1 =
1023;}
               else if (velocidad_servos.arti1 < 10){velocidad_servos.arti1 =
10;}
                        velocidad_servos.arti2  =  abs(posicion_servos_1.arti2  -
posicion_servos_0.arti2)*(velocidad/10);
                  if (velocidad_servos.arti2 > 1023){velocidad_servos.arti2 =
1023;}
               else if (velocidad_servos.arti2 < 10){velocidad_servos.arti2 =
10;}
                        velocidad_servos.arti3  =  abs(posicion_servos_1.arti3  -
posicion_servos_0.arti3)*(velocidad/10);
                  if (velocidad_servos.arti3 > 1023){velocidad_servos.arti3 =
1023;}
               else if (velocidad_servos.arti3 < 10){velocidad_servos.arti3 =
10;}

      }

      brazo::Servos velocidad_servos_0;

      velocidad_servos_0.base = 0;
      velocidad_servos_0.arti1 = 0;
      velocidad_servos_0.arti2 = 0;
      velocidad_servos_0.arti3 = 0;

      par_servos.base = 1;
      par_servos.arti1 = 1;
      par_servos.arti2 = 1;
      par_servos.arti3 = 1;

      brazo::WriteServos move_arm;

         if (coordenadas_correctas  == 1 && (205 <= posicion_servos_1.base &&
posicion_servos_1.base   <=   818)   &&   (120   <=   posicion_servos_1.arti1   &&
posicion_servos_1.arti1   <=   920)   &&   posicion_servos_1.arti2   >=   50     &&
(posicion_servos_1.arti3 <= 828 && posicion_servos_1.arti3 >= 195)) {
               move_arm.posicion = posicion_servos_1;
               move_arm.velocidad = velocidad_servos;
               move_arm.par = par_servos;
               return move_arm;
      }
      else {
                  std::cout<<"error  coordenadas  no  validas  o  punto  fuera  del
alcance"<<std::endl;
               move_arm.posicion = posicion_servos_0;
               move_arm.velocidad = velocidad_servos_0;
               move_arm.par = par_servos;
               return move_arm;
      }
 }

brazo::WriteServos  control_pinza(brazo::Servos posicion_servos_1, brazo::Servos
posicion_servos_0, brazo::Servos corriente_servos)
 {

      brazo::Servos velocidad_servos;
      brazo::Servos par_servos;
```

```
	velocidad_servos.pinza = 50;

	par_servos.pinza = 1;

	brazo::WriteServos hand_arm;

	 if ((posicion_servos_1.pinza >= 480 && posicion_servos_1.pinza <= 680 &&
corriente_servos.pinza     <=     300)     ||     (posicion_servos_0.pinza     >
posicion_servos_1.pinza && posicion_servos_1.pinza >= 480))
	{
		hand_arm.posicion = posicion_servos_1;
		hand_arm.velocidad = velocidad_servos;
		hand_arm.par = par_servos;
		return hand_arm;
	}
	else
	{
		std::cout<<"Alcanzado límite de la pinza"<<std::endl;
		hand_arm.posicion = posicion_servos_0;
		hand_arm.velocidad = velocidad_servos;
		hand_arm.par = par_servos;
		return hand_arm;
	}
 }

geometry_msgs::Point    home(brazo::Servos    posicion_servos_0,    brazo::Servos
corriente_servos)
 {

	ros::NodeHandle n;

	ros::Publisher move_pub_=n.advertise<brazo::WriteServos>("move_arm", 1);

	ros::Publisher hand_pub_=n.advertise<brazo::WriteServos>("hand_arm", 1);

		geometry_msgs::Point punto_0;

		punto_0.x = 0;
		punto_0.y = 80;
		punto_0.z = 50;

		int inclinacion_pinza = 0;

				brazo::WriteServos  inicio_brazo  =  inversa(punto_0,
inclinacion_pinza, posicion_servos_0, 0);

		brazo::Servos posicion_servos_1;

		posicion_servos_1.pinza = 511;

					brazo::WriteServos   inicio_pinza   =
control_pinza(posicion_servos_1, posicion_servos_0, corriente_servos);

		move_pub_.publish(inicio_brazo);

		hand_pub_.publish(inicio_pinza);


		return punto_0;

 }

#endif
```

Programa

Este programa ejecuta la secuencia para coger una botella pequeña de plástico (vacía, ya que la potencia de los servomotores es insuficiente para una llena, deberían emplearse más servomotores por articulación). Los pasos que realiza son los siguientes:

1. Primera aproximación (sitúa el brazo a unos 70 mm del punto indicado).
1. Posicionamiento en el punto indicado.
1. Cierre de la pinza (hasta tener agarrada la botella).

1.1.1.1 Levantamiento y acercamiento (levanta y acerca la botella al Robot)

Dentro del package creado, en el directorio "src" crearemos un fichero llamado "control_v02.cpp" con el siguiente contenido:

```
#include "ros/ros.h"
 #include "brazo/Servos.h"
 #include "brazo/WriteServos.h"
 #include "brazo/ReadServos.h"
 #include "brazo/brazo.h"
 #include "std_msgs/Int16.h"
 #include "std_msgs/String.h"
 #include "geometry_msgs/Point.h"
 #include "math.h"

 geometry_msgs::Point punto_0,punto_1;

 int cont = 0;
 int cont_1 = 0;
 int cont_2 = 0;
 int inclinacion_pinza = 0;
 int start = 0;

 brazo::Servos pg, cg;
 brazo::WriteServos move;
 brazo::Servos pinza;
 brazo::WriteServos pin;
 brazo::WriteServos error_coordenadas;

 brazo::WriteServos acercar(geometry_msgs::Point cerca)
 {
       double x = cerca.x;
       double z = cerca.z;

       cerca.z = cerca.z - 70*cos(atan2(x,z));
       cerca.x = cerca.x - 70*sin(atan2(x,z));

       brazo::WriteServos move = inversa(cerca, ::inclinacion_pinza, ::pg, 0);

       return move;

 }

   brazo::WriteServos levantar_acercar(geometry_msgs::Point la)
 {

       la.x = 0;
       la.y = la.y + 50;
       la.z = 60;
```

```
	if (la.y < 75) {
		la.y = 75;
	}

	brazo::WriteServos move = inversa(la, ::inclinacion_pinza, ::pg, 0);

	::punto_0 = la;

	return move;

}

void posicion_estado_corriente(const brazo::ReadServos& pec)
{

	ros::NodeHandle n;

	ros::Publisher move_pub_=n.advertise<brazo::WriteServos>("move_arm", 1);

	ros::Publisher hand_pub_=n.advertise<brazo::WriteServos>("hand_arm", 1);

	brazo::Servos p = pec.posicion;

	::pg = pec.posicion;
	::cg = pec.corriente;

	brazo::Servos e = pec.estado;

	brazo::Servos c = pec.corriente;


	if (::cont == 0)
	{
		::punto_0 = home(p, c);

		::punto_1 = ::punto_0;

		::cont = ::cont+1;

	}

	      if (::punto_0.x != ::punto_1.x || ::punto_0.y != ::punto_1.y
|| ::punto_0.z != ::punto_1.z ) {



	if (::cont_1 == 0) {

		::move = acercar(::punto_0);

		move_pub_.publish(::move);

		::cont_1 = ::cont_1+1;

	}


	     if (((p.base-15) < ::move.posicion.base && ::move.posicion.base <
(p.base+15)) && ((p.arti1-15) < ::move.posicion.arti1 && ::move.posicion.arti1 <
(p.arti1+15)) && ((p.arti2-15) < ::move.posicion.arti2 && ::move.posicion.arti2
<	(p.arti2+15))	&&	((p.arti3-15)	<	::move.posicion.arti3
&& ::move.posicion.arti3 < (p.arti3+15))) {
```

```
                        if (::cont_2 == 0) {

                                                                        ::move  =
inversa(::punto_0, ::inclinacion_pinza, ::pg, 0);

                                    move_pub_.publish(::move);

                                    ::cont_2 = ::cont_2+1;


                        }
                        else {


                                             if (((p.base-15) < ::move.posicion.base
&& ::move.posicion.base < (p.base+15)) && ((p.arti1-15) < ::move.posicion.arti1
&&      ::move.posicion.arti1      <      (p.arti1+15))      &&      ((p.arti2-15)
< ::move.posicion.arti2 && ::move.posicion.arti2 < (p.arti2+15)) && ((p.arti3-
15) < ::move.posicion.arti3 && ::move.posicion.arti3 < (p.arti3+15))) {


                                    ::pinza.pinza = p.pinza;

                                            if(::pin.posicion.pinza != p.pinza) {

                                                    std::cout<<c.pinza<<std::endl;

                                                      ::pinza.pinza = ::pinza.pinza +
20;

                                                    ::pin = control_pinza(::pinza, p,
c);


                                                    hand_pub_.publish(::pin);

                                            }
                                            else {

                                                    std::cout<<"Agarrado"<<std::endl;

                                                                        ::move  =
levantar_acercar(::punto_0);

                                                    move_pub_.publish(::move);

                                                    ::punto_1 = ::punto_0;

                                                    ::cont_1 = 0;
                                                    ::cont_2 = 0;
                                    }
                                    }
                        }

      }
      else {

                        std::cout<<"En movimiento"<<std::endl;
      }

      }


 }
```

```
 void punto(const geometry_msgs::Point& point)
 {
       ros::NodeHandle n;

        ros::Publisher move_pub_=n.advertise<brazo::WriteServos>("move_arm", 1);
//Publicación del topic "move_arm"

        ros::Publisher hand_pub_=n.advertise<brazo::WriteServos>("hand_arm", 1);
//Publicación del topic "hand_arm"

       ::punto_0 = point;

 }

 int main(int argc, char **argv)
 {

       ros::init(argc, argv, "control_recogida");

       ros::NodeHandle n;


                     ros::Subscriber   pose_sub_=   n.subscribe("pose_arm",   1,
posicion_estado_corriente);

       ros::Subscriber point_sub_= n.subscribe("pick_point", 1, punto);

       ros::Publisher move_pub_=n.advertise<brazo::WriteServos>("move_arm", 1);

       ros::Publisher hand_pub_=n.advertise<brazo::WriteServos>("hand_arm", 1);


       ros::spin();

       return 0;
 }
```

Para compilarlo y generar el ejecutable se debe añadir la siguiente línea de código al archivo "CMakeLists.txt" del package creado, donde indicamos el nombre para el ejecutable y la ruta y nombre del archivo a compilar:

```
rosbuild_add_executable(contro_v02 src/control_v02.cpp)
```

Para compilar el programa hay que situarse en el directorio del package. Simplemente con ejecutar la siguiente secuencia de comandos en un terminal se compilará y creará el ejecutable, siempre que no existan errores:

```
roscd brazo
make
```

Ejecutando el programa

Para ejecutar el programa debemos lanzar el núcleo de ROS, si no se encuentra iniciado, ejecutando en un terminal el siguiente comando:

```
roscore
```

Ahora debemos iniciar el nodo de la placa arduino ejecutando en un nuevo terminal el siguiente comando:

```
rosrun rosserial_python serial_node.py /dev/ttyACM0
```

Para ejecutar el programa de control llamado "control_v02", en un terminal nuevo ejecutaremos el siguiente comando:

```
rosrun brazo control_v02
```

Para poder publicar las coordenadas del punto donde se encuentra la botella (ya que no disponemos de otros medios para calcular la posición, se colocará la botella en una posición conocida), debemos publicar en el topic "pick_point", al que esta suscrito el programa de control para recibir el punto. Ejecutaremos en un nuevo terminal el siguiente comando (en este caso la posición en la que se encuentra la botella):

```
rostopic pub pick_point geometry_msgs/Point '{x: -150, y: 0, z: 150}' --once
```

Quinto programa (posa la botella)

Este programa ejecuta la secuencia para posar la botella que hemos cogido. Los pasos que realiza son los siguientes:

1. **Posicionamiento en el punto indicado.**
1. **Apertura de la pinza (hasta soltar la botella).**

1.1.1Separación (sitúa el brazo a unos 70 mm del punto indicado).

1.1.2Retorno a la posición de inicial.

Dentro del package creado, en el directorio "src" crearemos un fichero llamado "control_v03.cpp" con el siguiente contenido:

```
#include "ros/ros.h"
#include "brazo/Servos.h"
#include "brazo/WriteServos.h"
#include "brazo/ReadServos.h"
#include "brazo/brazo.h"
#include "std_msgs/Int16.h"
#include "std_msgs/String.h"
#include "geometry_msgs/Point.h"
#include "math.h"

geometry_msgs::Point punto_0, punto_1;

int cont = 0;
int cont_1 = 0;
int cont_2 = 0;
int cont_3 = 0;
int inclinacion_pinza = 0;

brazo::Servos pg, cg, pinza;
brazo::WriteServos move;
brazo::WriteServos pin;
brazo::WriteServos error_coordenadas;

brazo::WriteServos separar(geometry_msgs::Point lejos)
{
      double x = lejos.x;
      double z = lejos.z;

      lejos.z = lejos.z - 70*cos(atan2(x,z));
      lejos.x = lejos.x - 70*sin(atan2(x,z));
```

```
	brazo::WriteServos move = inversa(lejos, ::inclinacion_pinza, ::pg, 0);

	return move;

}

void posicion_estado_corriente(const brazo::ReadServos& pec)
{

	ros::NodeHandle n;

	ros::Publisher move_pub_=n.advertise<brazo::WriteServos>("move_arm", 1);

	ros::Publisher hand_pub_=n.advertise<brazo::WriteServos>("hand_arm", 1);

	brazo::Servos p = pec.posicion;

	::pg = pec.posicion;
	::cg = pec.corriente;

	brazo::Servos e = pec.estado;

	brazo::Servos c = pec.corriente;

	if (::cont == 0)
	{
		::punto_0 = directa(p, inclinacion_pinza);

		::punto_1 = ::punto_0;

		::cont = ::cont+1;
	}

	      if (::punto_0.x != ::punto_1.x || ::punto_0.y != ::punto_1.y
|| ::punto_0.z != ::punto_1.z ) {




	if (::cont_1 == 0) {

		::move = inversa(::punto_0, ::inclinacion_pinza, ::pg, 0);

		move_pub_.publish(::move);

		::cont_1 = ::cont_1+1;

	}

	    if (((p.base-15) < ::move.posicion.base && ::move.posicion.base <
(p.base+15)) && ((p.arti1-15) < ::move.posicion.arti1 && ::move.posicion.arti1 <
(p.arti1+15)) && ((p.arti2-15) < ::move.posicion.arti2 && ::move.posicion.arti2
<	(p.arti2+15))	&&	((p.arti3-15)	<	::move.posicion.arti3
&& ::move.posicion.arti3 < (p.arti3+15))) {

			if (::cont_2 == 0) {

				::pinza.pinza = 470;

				::pin = control_pinza(::pinza, p, c);

				hand_pub_.publish(::pin);
```

```
                        ::cont_2 = ::cont_2+1;

                }
                else {


                        if (p.pinza <= 495) {

                                if (::cont_3 == 0) {

                                        ::move = separar(::punto_0);

                                        move_pub_.publish(::move);

                                        ::cont_3 = ::cont_3+1;

                                }

                                                        else if (((p.base-15) < ::move.posicion.base && ::move.posicion.base < (p.base+15)) && ((p.arti1-15) < ::move.posicion.arti1 && ::move.posicion.arti1 < (p.arti1+15)) && ((p.arti2-15) < ::move.posicion.arti2 && ::move.posicion.arti2 < (p.arti2+15)) && ((p.arti3-15) < ::move.posicion.arti3 && ::move.posicion.arti3 < (p.arti3+15))) {

                                        std::cout<<"Suelto"<<std::endl;

                                        ::cont = 0;

                                        ::cont_1 = 0;
                                        ::cont_2 = 0;
                                        ::cont_3 = 0;
                                }
                        }
                }

        }
        else {

                std::cout<<"En movimiento"<<std::endl;
        }
        }



 }

 void punto(const geometry_msgs::Point& point)
 {

        ::punto_0 = point;

 }

 int main(int argc, char **argv)
 {

        ros::init(argc, argv, "control_entrega");

        ros::NodeHandle n;


                ros::Subscriber pose_sub_= n.subscribe("pose_arm", 1, posicion_estado_corriente);
```

```
	ros::Subscriber point_sub_= n.subscribe("place_point", 1, punto);

	ros::Publisher move_pub_=n.advertise<brazo::WriteServos>("move_arm", 1);
	ros::Publisher hand_pub_=n.advertise<brazo::WriteServos>("hand_arm", 1);

	ros::spin();
	return 0;
}
```

Para compilarlo y generar el ejecutable se debe añadir la siguiente línea de código al archivo "CMakeLists.txt" del package creado, donde indicamos el nombre para el ejecutable y la ruta y nombre del archivo a compilar:

```
rosbuild_add_executable(contro_v03 src/control_v03.cpp)
```

Para compilar el programa hay que situarse en el directorio del package. Simplemente con ejecutar la siguiente secuencia de comandos en un terminal se compilará y creará el ejecutable, siempre que no existan errores:

```
roscd brazo
make
```

Ejecutando el programa

Para ejecutar el programa debemos lanzar el núcleo de ROS, si no se encuentra iniciado, ejecutando en un terminal el siguiente comando:

```
roscore
```

Ahora debemos iniciar el nodo de la placa arduino ejecutando en un nuevo terminal el siguiente comando:

```
rosrun rosserial_python serial_node.py /dev/ttyACM0
```

Para ejecutar el programa de control llamado "control_v03", en un terminal nuevo ejecutaremos el siguiente comando:

```
rosrun brazo control_v03
```

Para poder publicar las coordenadas del punto donde queremos posar la botella, debemos publicar en el *topic* "place_point", al que esta suscrito el programa de control para recibir el punto. Ejecutaremos en un nuevo terminal el siguiente comando:

```
rostopic pub point geometry_msgs/Point '{x: 150, y: 0, z: 150}' --once
```

Sexto programa (Sigue el camino recto)

Se ha creado un programa que desplaza el brazo desde el punto en el que se encuentre al que le indiquemos, siguiendo la línea recta que une los puntos. Dentro del *package* creado, en el directorio "src" crearemos un fichero llamado "control_v04.cpp" con el siguiente contenido:

```
#include "ros/ros.h"
#include "brazo/Servos.h"
#include "brazo/WriteServos.h"
#include "brazo/ReadServos.h"
#include "brazo/brazo.h"
#include "geometry_msgs/Point.h"
#include "math.h"

brazo::Servos  p, e, c;
brazo::WriteServos destino, mover;
geometry_msgs::Point punto_0, punto_1, punto_i;
double lambda = 0;
int inclinacion_pinza = 0;
int cont = 0;

void posicion(const brazo::ReadServos& pose)
{
      ros::NodeHandle n;

      ros::Publisher move_pub_=n.advertise<brazo::WriteServos>("move_arm", 1);

      ::p = pose.posicion;
      ::e = pose.estado;
      ::c = pose.corriente;

  if (cont == 0)
  {
              ::punto_0 = home(::p, ::c);

              ::punto_1 = ::punto_0;

              cont = 1;
      }

            if ((::punto_0.x != ::punto_1.x || ::punto_0.y != ::punto_1.y
||  ::punto_0.z  !=  ::punto_1.z)  &&  ((::p.base-15)  <  ::mover.posicion.base
&&     ::mover.posicion.base     <     (::p.base+15))     &&     ((::p.arti1-15)
<  ::mover.posicion.arti1  &&  ::mover.posicion.arti1  <  (::p.arti1+15))  &&
((::p.arti2-15)   <   ::mover.posicion.arti2   &&   ::mover.posicion.arti2   <
(::p.arti2+15))      &&      ((::p.arti3-15)      <      ::mover.posicion.arti3
&& ::mover.posicion.arti3 < (::p.arti3+15)))
      {

                                if  (((::p.base-15)  <  ::destino.posicion.base
&&     ::destino.posicion.base     <     (::p.base+15))     &&     ((::p.arti1-15)
<  ::destino.posicion.arti1  &&  ::destino.posicion.arti1  <  (::p.arti1+15))  &&
((::p.arti2-15)   <   ::destino.posicion.arti2   &&   ::destino.posicion.arti2   <
(::p.arti2+15))      &&      ((::p.arti3-15)      <      ::destino.posicion.arti3
&& ::destino.posicion.arti3 < (::p.arti3+15)))
              {
                      ::punto_0 = ::punto_1;
                      ::lambda = 0;
              }
              else
              {
                      geometry_msgs::Point u;

                      //ecuación de la recta p1 = p0+lambda.u

                      u.x = ::punto_1.x - ::punto_0.x;
                      u.y = ::punto_1.y - ::punto_0.y;
                      u.z = ::punto_1.z - ::punto_0.z;
```

```
                        double paso = abs(sqrt(pow(u.x,2)+pow(u.y,2)+pow(u.z,2))/
10);

                        ::lambda = ::lambda + 1/paso;

                        ::punto_i.x = ::punto_0.x + (::lambda)*u.x;
                        ::punto_i.y = ::punto_0.y + (::lambda)*u.y;
                        ::punto_i.z = ::punto_0.z + (::lambda)*u.z;

                           ::mover = inversa(::punto_i, ::inclinacion_pinza, ::p,
0);

                        move_pub_.publish(::mover);

                }

        }

 }

 void punto(const geometry_msgs::Point& point)
 {
        ros::NodeHandle n;

        ros::Publisher move_pub_=n.advertise<brazo::WriteServos>("move_arm", 1);
        ros::Publisher hand_pub_=n.advertise<brazo::WriteServos>("hand_arm", 1);

        ::lambda = 0;

        ::punto_0 = directa(::p, ::inclinacion_pinza);

        ::punto_1 = point;

        ::destino = inversa(::punto_1, ::inclinacion_pinza, ::p, 0);

 }


 int main(int argc, char **argv)
 {

        ros::init(argc, argv, "control_brazo");

        ros::NodeHandle n;


        ros::Subscriber pose_sub_= n.subscribe("pose_arm", 1, posicion);

        ros::Subscriber point_sub_= n.subscribe("point", 1, punto);

        ros::Publisher move_pub_=n.advertise<brazo::WriteServos>("move_arm", 1);
        ros::Publisher hand_pub_=n.advertise<brazo::WriteServos>("hand_arm", 1);


        ros::spin();

   return 0;
 }
```

Para compilarlo y generar el ejecutable se debe añadir la siguiente línea de código al archivo "CMakeLists.txt" del package creado, donde indicamos el nombre para el ejecutable y la ruta y nombre del archivo a compilar:

```
rosbuild_add_executable(contro_v04 src/control_v04.cpp)
```

Para compilar el programa hay que situarse en el directorio del package. Simplemente con ejecutar la siguiente secuencia de comandos en un terminal se compilará y creará el ejecutable, siempre que no existan errores:

```
roscd brazo
make
```

Ejecutando el programa

Para ejecutar el programa debemos lanzar el núcleo de ROS, si no se encuentra iniciado, ejecutando en un terminal el siguiente comando:

```
roscore
```

Ahora debemos iniciar el nodo de la placa arduino ejecutando en un nuevo terminal el siguiente comando:

```
rosrun rosserial_python serial_node.py /dev/ttyACM0
```

Para ejecutar el programa de control llamado "control_v04", en un terminal nuevo ejecutaremos el siguiente comando:

```
rosrun brazo control_v04
```

Para poder publicar las coordenadas del punto donde queremos que se deplace el brazo, debemos publicar en el topic "point", al que esta suscrito el programa de control para recibir el punto. Ejecutaremos en un nuevo terminal el siguiente comando:

```
rostopic pub point geometry_msgs/Point '{x: 150, y: 100, z: 0}' --once
```

Séptimo programa (Teleoperador)

Se ha desarrollado un programa tele-operador para poder manejar el brazo en modo manual usando el teclado del PC. Dentro del *package* creado, en el directorio "src" crearemos un fichero llamado "teleoperador_teclado.cpp" con el siguiente contenido:

```
#include "ros/ros.h"
#include "brazo/Servos.h"
#include "brazo/WriteServos.h"
#include "brazo/ReadServos.h"
#include "brazo/brazo.h"
#include "geometry_msgs/Point.h"
#include "std_msgs/Int16.h"
#include "std_msgs/String.h"
#include <signal.h>
#include <termios.h>
#include <stdio.h>

#define KEYCODE_Xmas 0x43
#define KEYCODE_Xmenos 0x44
#define KEYCODE_Ymas 0x41
#define KEYCODE_Ymenos 0x42
#define KEYCODE_Zmas 0x77
#define KEYCODE_Zmenos 0x73
#define KEYCODE_Pabrir 0x61
```

```
 #define KEYCODE_Pcerrar 0x64
 #define KEYCODE_PinclinarMas 0x65
 #define KEYCODE_PinclinarMenos 0x71

struct termios cooked, raw;

int cont = 0;

double retardo = 0.11;

 geometry_msgs::Point punto;
 brazo::Servos vel;
 brazo::Servos pinza, p, c;

 int pinza_incli = 0;


void posicion_estado_corriente(const brazo::ReadServos& pec)
 {

       ::p = pec.posicion;
       ::c = pec.corriente;

 }

void quit(int sig)
{
 tcsetattr(0, TCSANOW, &cooked);
 ros::shutdown();
 exit(0);
}

void callback(const ros::TimerEvent&)
{

       ros::NodeHandle n;

       ros::Publisher move_pub_=n.advertise<brazo::WriteServos>("move_arm", 1);

       ros::Publisher hand_pub_=n.advertise<brazo::WriteServos>("hand_arm", 1);

       ros::Time anterior;

       brazo::WriteServos teleop;
       brazo::WriteServos teleop_pinza;

       if (::cont == 0)
   {
               anterior = ros::Time::now();

               ::punto = home(::p, ::c);

               ::cont = 1;

               ::pinza.pinza = 511;

               teleop = inversa(::punto, ::pinza_incli, ::p, 0);
               teleop_pinza = control_pinza(::pinza, ::p, ::c);

               //std::cout<<teleop<<"-"<<teleop_pinza<<std::endl;
       }

 char c;
```

```
 // get the console in raw mode
 tcgetattr(0, &cooked);
 memcpy(&raw, &cooked, sizeof(struct termios));
 raw.c_lflag &=~ (ICANON | ECHO);
 // Setting a new line, then end of file
 raw.c_cc[VEOL] = 1;
 raw.c_cc[VEOF] = 2;
 tcsetattr(0, TCSANOW, &raw);

 puts("");
 puts("########################################################");
 puts("                    CONTROLES BRAZO");
 puts("########################################################");
 puts("");
 puts("Flecha arriba:________ Subir pinza");
 puts("Flecha abajo:_________ Bajar pinza");
 puts("Flecha izquierda:_____ Desplazar pinza a la izquierda");
 puts("Flecha derecha:_______ Desplazar pinza a la derecha");
 puts("Tecla W:______________ Desplazar pinza hacia delante");
 puts("Tecla S:______________ Desplazar pinza hacia atrás");
 puts("Tecla A:______________ Abrir pinza");
 puts("Tecla D:______________ Cerrar pinza");
 puts("Tecla Q:______________ Inclinar pinza hacia arriba");
 puts("Tecla E:______________ Inclinar pinza hacia abajo");


   while (ros::ok())
   {

   // get the next event from the keyboard
   if(read(0, &c, 1) < 0)
   {
     perror("read():");
     exit(-1);
   }

     if (c == KEYCODE_Xmas)
     {
               if (ros::Time::now() > anterior + ros::Duration(::retardo))
               {
               ::punto.x = ::punto.x - 5;
               teleop = inversa(::punto, ::pinza_incli, ::p, 0);
                  if (teleop.posicion.base == ::p.base && teleop.posicion.arti1
==  ::p.arti1  &&  teleop.posicion.arti2  ==  ::p.arti2  &&  teleop.posicion.arti3
== ::p.arti3)
               {::punto.x = ::punto.x + 5;}
               anterior = ros::Time::now();
               }
     }
     if (c == KEYCODE_Xmenos)
     {
               if (ros::Time::now() > anterior + ros::Duration(::retardo))
               {
               ::punto.x = ::punto.x + 5;
               teleop = inversa(::punto, ::pinza_incli, ::p, 0);
                  if (teleop.posicion.base == ::p.base && teleop.posicion.arti1
==  ::p.arti1  &&  teleop.posicion.arti2  ==  ::p.arti2  &&  teleop.posicion.arti3
== ::p.arti3)
               {::punto.x = ::punto.x - 5;}
               anterior = ros::Time::now();
               }
     }
     if (c == KEYCODE_Ymas)
     {
```

```
                if (ros::Time::now() > anterior + ros::Duration(::retardo))
                {
                ::punto.y = ::punto.y + 5;
                teleop = inversa(::punto, ::pinza_incli, ::p, 0);
                   if (teleop.posicion.base == ::p.base && teleop.posicion.arti1
==  ::p.arti1  &&  teleop.posicion.arti2  ==  ::p.arti2  &&  teleop.posicion.arti3
== ::p.arti3)
                {::punto.y = ::punto.y - 5;}
                anterior = ros::Time::now();
                }
     }
     if (c == KEYCODE_Ymenos)
     {
                if (ros::Time::now() > anterior + ros::Duration(::retardo))
                {
                ::punto.y = ::punto.y - 5;
                teleop = inversa(::punto, ::pinza_incli, ::p, 0);
                   if (teleop.posicion.base == ::p.base && teleop.posicion.arti1
==  ::p.arti1  &&  teleop.posicion.arti2  ==  ::p.arti2  &&  teleop.posicion.arti3
== ::p.arti3)
                {::punto.y = ::punto.y + 5;}
                anterior = ros::Time::now();
                }
     }
     if (c == KEYCODE_Zmas)
     {
                if (ros::Time::now() > anterior + ros::Duration(::retardo))
                {
                ::punto.z = ::punto.z + 5;
                teleop = inversa(::punto, ::pinza_incli, ::p, 0);
                   if (teleop.posicion.base == ::p.base && teleop.posicion.arti1
==  ::p.arti1  &&  teleop.posicion.arti2  ==  ::p.arti2  &&  teleop.posicion.arti3
== ::p.arti3)
                {::punto.z = ::punto.z - 5;}
                anterior = ros::Time::now();
                }
     }
     if (c == KEYCODE_Zmenos)
     {
                if (ros::Time::now() > anterior + ros::Duration(::retardo))
                {
                ::punto.z = ::punto.z - 5;
                teleop = inversa(::punto, ::pinza_incli, ::p, 0);
                   if (teleop.posicion.base == ::p.base && teleop.posicion.arti1
==  ::p.arti1  &&  teleop.posicion.arti2  ==  ::p.arti2  &&  teleop.posicion.arti3
== ::p.arti3)
                {::punto.z = ::punto.z + 5;}
                anterior = ros::Time::now();
                }
     }
     if (c == KEYCODE_Pabrir)
     {
                if (ros::Time::now() > anterior + ros::Duration(::retardo))
                {
                ::pinza.pinza = ::pinza.pinza - 5;
                teleop_pinza = control_pinza(::pinza, ::p, ::c);
                    if (teleop_pinza.posicion.pinza == ::p.pinza) {::pinza.pinza
= ::pinza.pinza + 5;}
                anterior = ros::Time::now();
                }
     }
     if (c == KEYCODE_Pcerrar)
     {
                if (ros::Time::now() > anterior + ros::Duration(::retardo))
```

```
                {
                ::pinza.pinza = ::pinza.pinza + 5;
                teleop_pinza = control_pinza(::pinza, ::p, ::c);
                    if (teleop_pinza.posicion.pinza == ::p.pinza) {::pinza.pinza
= ::pinza.pinza - 5;}
                anterior = ros::Time::now();
                }
     }
     if (c == KEYCODE_PinclinarMas)
     {
                if (ros::Time::now() > anterior + ros::Duration(::retardo))
                {
                ::pinza_incli = ::pinza_incli + 5;
                teleop = inversa(::punto, ::pinza_incli, ::p, 0);
                   if (teleop.posicion.base == ::p.base && teleop.posicion.arti1
==  ::p.arti1  &&  teleop.posicion.arti2  ==  ::p.arti2  &&  teleop.posicion.arti3
== ::p.arti3)
                {::pinza_incli = ::pinza_incli - 5;}
                anterior = ros::Time::now();
                }
     }
     if (c == KEYCODE_PinclinarMenos)
     {
                if (ros::Time::now() > anterior + ros::Duration(::retardo))
                {
                ::pinza_incli = ::pinza_incli - 5;
                teleop = inversa(::punto, ::pinza_incli, ::p, 0);
                   if (teleop.posicion.base == ::p.base && teleop.posicion.arti1
==  ::p.arti1  &&  teleop.posicion.arti2  ==  ::p.arti2  &&  teleop.posicion.arti3
== ::p.arti3)
                {::pinza_incli = ::pinza_incli + 5;}
                anterior = ros::Time::now();
                }
     }

                  if (teleop.posicion.base != ::p.base || teleop.posicion.arti1 !
= ::p.arti1 || teleop.posicion.arti2 != ::p.arti2 || teleop.posicion.arti3 !
= ::p.arti3)
                {
                        move_pub_.publish(teleop);
                }
                if (teleop_pinza.posicion.pinza != ::p.pinza)
                {
                        hand_pub_.publish(teleop_pinza);
                }

       }
}

 int main(int argc, char **argv)
 {

       ros::init(argc, argv, "teleoperador_teclado");

       ros::NodeHandle n;

                     ros::Subscriber   pose_sub_=   n.subscribe("pose_arm",   1,
posicion_estado_corriente);

       ros::Publisher move_pub_=n.advertise<brazo::WriteServos>("move_arm", 1);

       ros::Publisher hand_pub_=n.advertise<brazo::WriteServos>("hand_arm", 1);
```

```
        signal(SIGINT,quit);

        ros::Timer timer = n.createTimer(ros::Duration(1), callback);


        ros::spin();

        return 0;
 }
```

Para compilarlo y generar el ejecutable se debe añadir la siguiente línea de código al archivo "CMakeLists.txt" del package creado, donde indicamos el nombre para el ejecutable y la ruta y nombre del archivo a compilar:

```
rosbuild_add_executable(teleoperador_teclado src/teleoperador_teclado.cpp)
```

Para compilar el programa hay que situarse en el directorio del *package*. Simplemente con ejecutar la siguiente secuencia de comandos en un terminal se compilará y creará el ejecutable, siempre que no existan errores:

```
roscd brazo
make
```

Ejecutando el programa

Para ejecutar el programa debemos lanzar el núcleo de ROS, si no se encuentra iniciado, ejecutando en un terminal el siguiente comando:

```
roscore
```

Ahora debemos iniciar el nodo de la placa arduino ejecutando en un nuevo terminal el siguiente comando:

```
rosrun rosserial_python serial_node.py /dev/ttyACM0
```

Para ejecutar el programa de control llamado "teleoperador_teclado", en un terminal nuevo ejecutaremos el siguiente comando:

```
rosrun brazo teleoperador_teclado
```

Una vez entendido todo este proceso, reemplazar la tarjeta de ARDUINO por la de un microcontrolador de 32 bits

En el caso de que estes batallando en el uso de Cypress PSoC4, aquí esta un ejemplo de como realizar una conexión con ROS

RosOnAStick

Una demostración de una Board ARM de $4 dlls con un nodo ROS

Vista general del dispositivo

Estos procesadores Cypress de arquitectura ARM están disponibles en placas de "starter kits" de muy bajo costo

PSoC4 CY8CKIT-049 por menos de $ 5

Entorno de desarrollo

La arquitectura del "Sistema programable en un chip" (PsoC “Programmable System on a Chip") admite periféricos altamente configurables, incluidos componentes de alto nivel (por ejemplo, UART) y de bajo nivel (por ejemplo, FPGA a nivel de compuerta). Cypress proporciona una herramienta de desarrollo gráfico gratuita basada en Windows (PSoC Creator) mediante la cual el diseño se ingresa en forma de diagrama de bloques / esquema y un "instalador" asigna recursos de hardware para implementar el diseño.

Además, PSoC Creator es un IDE completo con editor de programas, compilador, enlazador y capacidad de descarga; utiliza la herramienta arm-gcc. Los desarrolladores que prefieren realizar el desarrollo de código en un entorno Linux pueden hacerlo utilizando una función de "exportación". Esto puede ser conveniente, ya que el desarrollo y la ejecución de ROS requieren un host Linux. En cualquier caso, el paso inicial de configuración de los componentes de "hardware" solo es práctico en la herramienta PSoC Creator y debe ejecutarse en un entorno nativo de Windows.

Los archivos están diseñados con PSoC Creator y puede ser clonado a cualquier ubicación conveniente; en PSoC Creator, navegue y abra el espacio de trabajo RosOnAStick / RosOnAStick.cywrk.

Bibliografia

https://github.com/chuck-h/RosOnAStick
https://github.com/chuck-h/ros-psoc
http://robotica.unileon.es/index.php/Control_brazo_Robot_(bioloid%2Barduino) Carlos Rodríguez Hernández