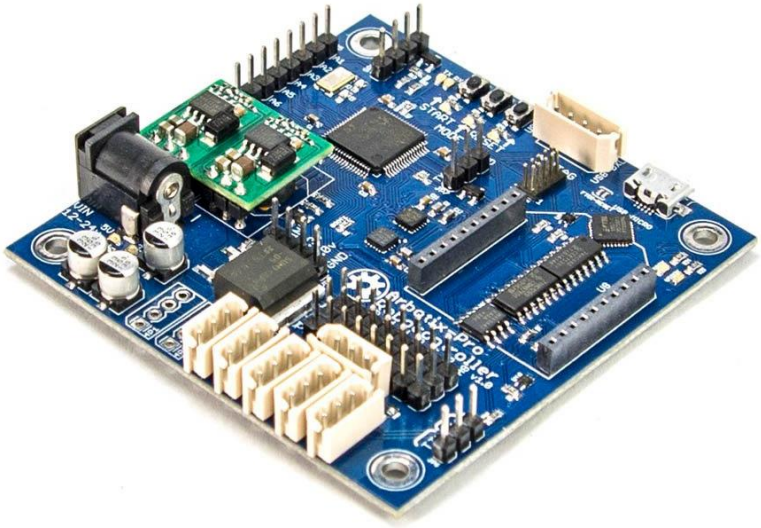
Guía de uso del paquete arbotix_ros



Proyecto de TFG: *Control y simulación en ROS de un PhantomX Reactor Arm en cooperación con un TurtleBot2*

Autor: Daniel Lozano Moreno

Tutora: María del Pilar Arqués Corrales

Grado en Ingeniería Robótica

Escuela Politécnica Superior

Universitat d'Alacant
Universidad de Alicante

Alicante, Junio 2022

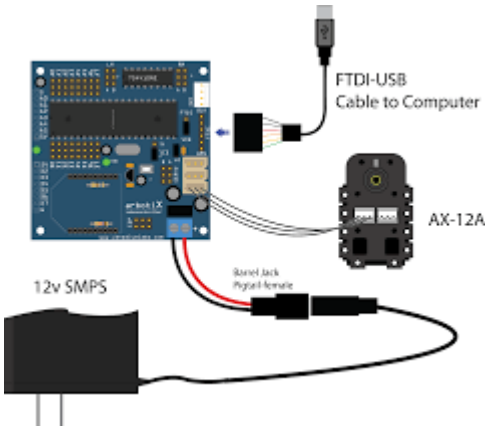
RESUMEN

Se presenta la documentación del estudio del paquete para ROS *arbotix_ros* realizado durante la elaboración del proyecto de Trabajo de Fin de Grado *Control y Simulación de un PhantomX Reactor Arm en cooperación con un TurtleBot2*, por Daniel Lozano Moreno en el Grado en Ingeniería Robótica de la Escuela Politécnica de Superior de la Universidad de Alicante. La motivación para realizar esta guía de uso reside en compartir la información estudiada sobre este paquete de manera sencilla y resumida para facilitar la configuración de una placa ArbotiX-M para su uso con ROS. El estudio se llevó a cabo debido a que los TurtleBots2 que posee el laboratorio de robótica de la Universidad llevan instalados un PhantomX Reactor Arm, cuyos motores Dynamixel son controlados a partir de esta placa, conectada a su vez a la CPU del TurtleBot2 que funciona con el sistema operativo ROS. El paquete ROS *arbotix_ros* es el driver que establece la comunicación entre un ArbotiX-M y ROS, necesario para el control del brazo desde la CPU del TurtleBot. Debido a la poca información sobre este paquete, se ha realizado un gran estudio del código del paquete para conocer su funcionamiento y así realizar las configuraciones que se necesitaban para el proyecto. Debe tenerse en cuenta que no todas las funcionalidades del paquete han sido probadas, únicamente la relacionada con la configuración del ArbotiX-M para el control de seguimiento trayectorias. Aun así, se han intentado entender todas ellas y explicarlas en el documento.

arbotix_ros package

El meta-paquete *arbotix_ros* de ROS ofrece los drivers de comunicación entre una paca ArbotiX-M y ROS. Esta placa fue desarrollada por Trossen Robotics para la programación de aplicaciones dirigidas al control de motores Dynamixel. Este paquete conecta ArbotiX-M y ROS, el sistema operativo estándar en robótica, para controlar motores Dynamixel desde ROS. Para ello, declara multitud de clases que representan al ArbotiX-M con sus motores y ofrece multitud de configuraciones y funcionalidades para su control. En este documento se explicará cada uno de sus paquetes para conocer todo su potencial. El paquete puede ser descargado en: https://github.com/vanadiumlabs/arbotix_ros

La comunicación física entre un ArbotiX-M y ROS se establece mediante un cable FTDI TTL-USB. La CPU con ROS debe tener una entrada USB para conectarse con este cable al ArbotiX-M. Los motores Dynamixel controlados por la placa estarán conectados directamente a ella. Debe usarse Arduino IDE para programar la placa con el código ros.ino, incluido en los drivers para Arduino IDE *arbotix-master* (Hansen, 2015).



Set-up de un ArbotiX-M

Los paquetes que integra *arbotix_ros* son:

- arbotix_firmware: drivers de comunicación
- arbotix_python: clases para la configuración
- arbotix_controllers: controladores para pinzas robóticas
- arbotix_msgs: mensajes de servicios
- arbotix_sensors: sensores de retroalimentación

En los siguientes capítulos, se explicará cada uno de los paquetes.

Contenido

1 arbotix_firmware

arbotix_firmware ofrece los drivers de comunicación que deben ser instalados en la placa ArbotiX para comunicarse con ROS. Estos archivos son:

- *ros.ino*: archivo principal del paquete. Establece la conexión entre el ArbotiX-M desde el lado del ArbotiX. Para su instalación, seguir la guía de ArbotiX-M (Hansen, 2015).
- *ros.h*: declara las constantes del programa.
- *diff_controller.h*: contiene un controlador PID para el control de los motores dynamixel con valores Kp=25, Kd=30, Ki =0.
- *User_hooks.h*: fichero vacío para que los usuarios programen sus códigos de configuración.

2 arbotix_python

Arbotix_python es el paquete pincipal de *arbotix_ros*, encargado de establecer la conexión entre el ArbotiX real y ROS en el lado de ROS y de configurar la placa para el funcionamiento deseado Contiene 3 nodos:

- arbotix_driver: nodo principal. Establece la conexión entre ROS y Arbotix a partir de la clase ArbotixROS.
- arbotix_gui: carga una GUI para controlar a los motores.
- arbotix_terminal: crea comandos para controlar los motores desde una terminal.

Para iniciar la conexión con ArbotiX-M se debe llamar al nodo arbotix_driver junto a los parámetros de configuración deseados. Si los parámetros de configuración se encuentran en un fichero YAML, el nodo se lanzaría en un launcher de la siguiente manera:

```
1.  <!-- Load arbotix_driver -->
2.  <node name="arbotix" pkg="arbotix_python" type="arbotix_driver" output="screen">
3.     <!-- Load params in ROS parameter server -->
4.     <rosparam file="$(find package_name)/path/to/file/config.yaml" command="load" />
5.  </node>
```

ArbotixROS es la clase en python que establece la conexión entre ArbotiX y ROS en el lado de ROS junto a la configuración deseada. Para ello, como parámetros de construcción es necesario pasarles los siguientes elementos:

- *port*: puerta USB a la que está conectado el ArbotiX-M.
- *rate*: frecuencia de muestreo.
- *joints*: lista de accionamientos, uno por cada motor a controlar.
- *controllers*: controladores para los accionamientos.

La puerta USB se declara como una rule para ArbotiX-M. Resumidamente, rule (regla) es una etiqueta que va ligada al número de serie de un dispositivo. Para que no se necesite escribir la puerta USB cada vez que el dispositivo cambie de entrada, se crea esta rule para nombrar directamente al dispositivo y se buscará automáticamente a que puerta está conectada. Las reglas suelen guardarse en la carpeta dev (device) de Linux. Por defecto, la clase ArbotixROS define esta parámetro como "/dev/ttyUSB0". Para saber cómo crear una rule para el ArbotiX-M, consultar la sección "Creating the udev rule for the device" de (RobotnikAutomation, 2019).

En cuanto al parámetro *joints*, proporciona tres tipos de accionamientos funcionales con ArbotiX:

- dynamixel: servo-motor Dynamixel (por defecto)
- hobby_servo: servo-motor hobby o radio-control (servo-motor tradicional)
- calibrated_linear: servo-motor lineal

A cada accionador se debe especificar el número de identificación del motor 'id'.

En cuanto a *controllers*, se definen cuatro tipos de controladores. Estos controladores se encargan de procesar comandos de entrada especiales para ser ejecutados por los servo-motores. Por defecto, ningún controlador se define para los accionamientos, en cuyo caso, el control de entrada se realiza en posición:

- follow_controller: control de ejecución de trayectorias
- diff_controller: control diferencial
- linear_controller: control lineal con retroalimentación
- linear_controller_i: control lineal sin retroalimentación

Una configuración ArbotiX puede escribirse en un fichero YAML como el siguiente:

```
# Config file for PhantomX Reactor
port: /dev/ttyUSB_REACTOR
rate: 100
joints: {
  arm_shoulder_yaw_joint: {id: 1, max_speed: 50.0},
  arm_shoulder_pitch_joint: {id: 2, max_speed: 50.0},
  arm_shoulder_pitch_mimic_joint: {id: 3, max_speed: 50.0},
  arm_elbow_pitch_joint: {id: 5, max_speed: 50.0},
  arm_elbow_pitch_mimic_joint: {id: 4, max_speed: 50.0},
  arm_wrist_pitch_joint: {id: 6, max_speed: 50.0, invert: true},
  arm_wrist_roll_joint: {id: 7, max_speed: 50.0},
  arm_gripper_revolute_joint: {id: 8, max_speed: 100.0, range: 180, min_angle: -90.0, max_angle: 0},
}
controllers: {
  arm_controller: {
    type: 'follow_controller',
    rate: 100,
    joints: [
            'arm_shoulder_yaw_joint',
            'arm_shoulder_pitch_joint',
            'arm_shoulder_pitch_mimic_joint',
            'arm_elbow_pitch_joint',
            'arm_elbow_pitch_mimic_joint',
            'arm_wrist_pitch_joint',
            'arm_wrist_roll_joint'
            ]
   action_name: 'arm_controller/follow_joint_trajectory'
  }
  grip_controller: {
    type: 'follow_controller',
    rate: 100,
    joints: [
            'arm_gripper_revolute_joint'
            ]
    action_name: 'grip_controller/follow_joint_trajectory'
  }
}
```

Para conocer la sintaxis de cada parámetro de construcción, leer su respectiva clase en el Capítulo 6.

3 arbotix_controllers

Este paquete define los controladores orientados al control de pinzas. Las pinzas pueden ser controladas mediante los controladores ya definidos en arbotix_python, sin embargo estos controladores mejoran la precisión con la que actúan.

Para usar estas funcionalidades, las articulaciones de la pinza primero deben haberse cargado como se explica en el Capítulo 2, pero no debe asignarse ningún controlador a estas articulaciones. Posteriormente, se llama al nodo gripper_controller para cargar los parámetros de configuración de la pinza. Si los parámetros de configuración se encuentra en un fichero YAML, el nodo se lanzaría en un launcher de la siguiente manera:

```
<!-- Load gripper_controller -->
<node name="gripper" pkg="arbotix_controller" type="gripper_controller" output="screen">
   <!-- Load params in ROS parameter server -->
   <rosparam file="$(find package_name)/path/to/file/gripper_config.yaml" command="load" />
</node>
```

Dependiendo del modelo de la pinza usado, los parámetros de configuración serán distintos, por lo que léase con atención las clases del que se desee usar. Siempre será necesario indicar con el parámetro *model* el modelo de la pinza:

- dualservo: pinzas de mordazas no paralelas con dos motores
- parallel: pinza de mordazas paralelas de un motor
- singlesided: pinza paralela de un motor

4 arbotix_msgs

Al cargar un ArbotixROS con el modelo de articulación Dynamixel, se cargarán a su vez los siguientes servicios:

- rospy.Service(name+'/relax', Relax, self.relaxCb): deshabilita los motores.
- rospy.Service(name+'/enable', Enable, self.enableCb): habilita los motores.
- rospy.Service(name+'/set_speed', SetSpeed, self.setSpeedCb): cambia la velocidad de los motores.

Este paquete define los mensajes que pueden ser comandados a estos servicios.

5 arbotix_sensors

Define convertidores de medidas analógicas a medidas de rangos. Se usa internamente para la comunicación entre ROS y ArbotiX-M.

6 Clases del paquete

6.1 arbotix_python

6.1.1 ArbotiX

Representación de un dispositivo ArbotiX para el control de una placa real a través de una conexión serial. Permite la escritura y lectura de posiciones y velocidades de un Dynamixel servo y solo posiciones de un Hobby Servo, y su control de cualquier tipo. Es la clase padre del paquete.

Class ArbotiX() – *abotix_python/src/abotix_python/arbotix.py*			
Construct Params	**Content**	**Default Value**	**Description**
port	Puerta serie	"/dev/ttyUSB0"	Etiqueta udev de la puerta serie a la que está conectada la placa ArbotiX.
baud	Baudrate	115200	Frecuencia de comunicación serie con el ArbotiX
timeout	Timeout	0.1	Tiempo para conectarse a la placa
open_port	Abrir puerta serie	True	True para abrir la puerta serie del ArbotiX al crearse una instancia a esta clase
Attributes	**Content**	**Init Values**	**Description**
self._ser	Serie	serial.Serial()	Instancia a un Serial
self._ser.port	Puerto serie	port	Etiqueta udev de la puerta serie a la que está conectada la placa ArbotiX.
self._ser.baudrate	Baudrate serie	baud	Frecuencia de comunicación serie con el ArbotiX
self._ser.timeout	Timeout serie	timeout	Tiempo para conectarse a la placa
self._ser.error	Error serie	0	El último error leído devuelto
Methods	**Return**	**Params**	**Description**
__write__	void	msg	Envía el mensaje *msg* por la puerta serie
openPort	void	void	Abre la puerta serie
closePort	void	void	Cierra la puerta serie
getPacket	Paquete de datos	mode, id, leng, error, params	Obtiene el paquete de datos ligado a la id. En caso de de no ser posible, devuelve None
execute	getPacket()	index, ins, params, ret	Envía una instrucción al dispositivo. index: id del motor ins: mensaje params: lista de parámetros a enviar
read	Lista de bytes leídos	index, start, length	Lee valores de los registros. index: id del motor start: dirección de registro de comiezo para leer length: numero de bytes de lectura
write	Código de error	index, start, lenght	Escribir valores en los registros. index: id del motor start: dirección de registro de comiezo para leer length: numero de bytes de lectura
syncWrite	void	start, values	Escribe valores en los registros de varios servos. start: dirección de registro de comiezo para leer values: datos a escribir
syncRead	Lista los bytes leídos	servos, start, lenght	Lee valores de los registros de varios servos servos: lista de las id de los motores start: dirección de registro de comiezo para leer length: numero de bytes de lectura de cada servo
setBaud	Código de error	index, baud	Estalece el baudrate de conexión con un servo. index: id del motor aud: baudrate
getReturnLevel	Nivel devuelto	index	Devuelve el nivel de un servo index: id del motor
enableTorque	Código de error	index	Habilita el Torque de un servo index: id del motor
disableTorque	Código de error	index, value	Deshabilita el Torque de un servo index: id del motor value: 0 para apagar el led del robot, >0 para encenderlo
setLed	Código de error	index, value	Apagar/encender el led de un servo

			index: id del motor value: 0 para apagar el led del robot, >0 para encenderlo
`setPosition`	Código de error	index, value	Comandar una posición a un servo index: id del motor value: posición
`setVelocidad`	Código de error	index, value	Comanda una velocidad a un servo index: id del motor value: velocidad
`getPosition`	Posición	index	Obtiene la posición de un servo index: id del motor
`getSpeed`	Velocidad	index	Obtiene la velocidad de un servo index: id del motor
`getGoalSpeed`	Velocidad objetivo	index	Obtiene la velocidad objetivo de un servo index: id del motor
`getVoltaje`	Voltaje	index	Obtiene el voltaje de un servo index: id del motor
`getTemperature`	Temperatura	index	Obtiene la temperatura de un servo index: id del motor
`isMoving`	bool	index	True si el servo se está moviendo index: id del motor
`enableWheelMode`	void	index	Pone un servo en modo rueda (ratación continua) index: id del motor
`disableWheelMode`	void	index, resolution	Quita a un servo del modo rueda (ratación continua) index: id del motor
`setWheelSpeed`	void	index, direction, speed	Establedce la velocidad y dirección de un servo que está en el modo rueda (rotación continua) index: id del motor direction: sentido de giro speed: velocidad
`rescan`	void	void	Fuerza a un Arbotix a reescanear los busses de un Dynamixel.
`getAnalog`	Valor analógico del pin	index, leng	Obtiene el valor analógico de un pin input index: id del motor leng: número de bytes a leer
`getDigital`	Valor digital del pin	index, leng	Obtiene el valor digital de un pin input (0: low; 255: high) index: id del motor
`setDigital`	-1 si error	index, value, direction	Escribe un valor digital a un pin index: id del motor value: valor a escribir direction: dirección puerto (>o es output)
`setServo`	-1 si error	index, value	Establece la posición a un Hobby servo index: id del motor value: valor de posición

6.1.2 ArbotixROS

Drivers de comunicación entre arbotix_ros y la placa ArbotiX. Es hijo de la clase ArbotiX. Inicia el bucle del nodo permitiendo el control de los motores conectados al ArbotiX.

Si el atributo `self.fake` valge True, creará una instancia a la clase ArbotiX y se establecerá la conexión con la placa real llamando a la función `self.connectArbotiX()`. Configura los servicios de entrada y salida analógica y digital de conexión serial.

`Class ArbotixROS(ArbotiX)` – *`abotix_python/bin/arbotix_driver`*			
Attributes	**Content**	**Init Values**	**Description**
`self.rate`	Frecuencia de publicación	`rospy.get_param("~rate", 100.0)`	Frecuencia a la que se publicarán los motores.

`self.fake`	Simulación	`rospy.get_param("~sim", False)`	True si existe un dispositivo ArbotiX conectado, en cuyo caso se llama a self.connectArotiX().

`self.joints`	Articulaciones	`for name in rospy.get_param("~joints"):`
		`  joint_type=get("~joints/"+name+"/type")`
		`  if joint_type == "dynamixel":`
		`  self.joints[name] =`
		`  DynamixelServo(self, name)`
		`  elif joint_type == "hobby_servo":`
		`  self.joints[name] =`
		`  HobbyServo(self, name)`
		`  elif joint_type == "calibrated_linear":`
		`  self.joints[name] =`
		`  LinearJoint(self, name)`
			Diccionario de las articulaciones que posee el ArbotiX.
			Estará formado oor una i
			Cada Artculación será una instancia a una de estos tres tipos de articulaciones:
			• *dynamixel* [DynamixelServo()]
			• *hobby_servo* [HobbyServo()]
			• *calibrated_linear* [LinearJoint()]
			Por defecto, el tipo es *dynamixel*.

`self.controllers`	Controladores	`self.controllers = [ServoController(self, "servos"), ]`
		`controllers = get_param("~controllers", dict())`
		`  for name, params in controllers.items():`
		`  controller = controller_types[params["type"]](self, name)`
		`  self.controllers.append( controller )`
		`  pause = pause or controller.pause`
			Diccionario de los controladores para las articulaciones.
			Está formado por una instancia a ServoController() a la que se concatenan los controladors, que serán instancias a alguno de estos tres tipos de controladores:
			• *follow_controller* [FollowController()]
			• *diff_controller* [DiffController()]
			• *linear_controller* [LinearControllerAbsolute()]
			• *linear_controller_i* [LinearControllerIncremental()]

Methods	**Return**	**Params**	**Description**
`digitalInCb`	void	req	Inicia el servicio digital input.
`digitalOutCb`	void	req	Inicia el servicio digital output.
`analogInCb`	void	req	Inicia el servicio analógico input.
`connectArbotiX`	void	void	Establece la conexión con la placa ArbotiX real

6.1.3 Joint

Clase padre para la representación de una articulación. Los métodos no están declarados, esto se hace en sus clases hijas (tipos de articulación).

`Class Joint()` – *abotix_python/src/abotix_python/joints.py*			
Construct Params	**Content**	**Default Value**	**Description**
`device`	Instancia a ArbotiX		
`name`	Nombre de la articulación		
Attributes	**Content**	**Init Values**	**Description**
`self.name`	Nombre de la articulación	`name`	
`self.device`	Instancia a ArbotiX	`device`	
`self.controller`	True si tiene un controlador ligado	`None`	
`self.position`	Posición	`0.0`	Actual posición, devuelta por retroalimentación (metros)
`self.velocity`	Velocidad	`0.0`	Velocidad de movimiento
`self.last`	Timestamp	`rospy.Time.now()`	Tiemstamp de la última actualización retroalimentada.
Methods	**Return**	**Params**	**Description**
`interpolate`	Nuevo output	frame	Obtiene un nuevo output.

			frame: the frame length in seconds to interpolate forward
`setCurrentFeedback`	Posición actual en rad/metros	raw_data	Establece la actual posición desde la realimentación raw_data: actual dato realimentado
`setControlOutput`	Posición output	position	Establece la posición objetivo: position: posición deseado
`getDiagmosticOutput`	Diagnóstico	void	Obtiene un diagnóstico e la articulación (voltaje, temperatura del motor)

6.1.4 LinearJoint

Tipo de articulación que representa a una articulación lineal. Es una clase heredada de Joint y declara sus métodos.

Subscripciones:

- `rospy.Subscriber(name+'/command', Float64, self.commandCb)`

`class LienarJoint(Joint)` – *abotix_python/src/abotix_python/linear_controller.py*			
Construct Params	**Content**	**Default Value**	**Description**
`device`	Instancia a ArbotiX		
`name`	Nombre de la articulación		
Attributes	**Content**	**Init Value**	**Description**
`self.position`	Posición	`0.0`	Actual posición, devuelta por retroalimentación (metros)
`self.desired`	Deseada	`0.0`	Posición deseada (metros)
`self.velocity`	Velocidad	`0.0`	Velocidad de movimiento
`self.last`	Timestamp	`rospy.Time.now()`	Tiemstamp de la última actualización retroalimentada.
`self.min`	Lower limit	`rospy.get_param('~joints/'+name+'/min_position')` `Default: 0.0`	Estos valores deberían obtenerse del URDF del robot.
`self.max`	Upper limit	`rospy.get_param('~joints/'+name+'/max_position')` `Default: 0.5`	
`self.max_speed`	Máxima velocidad	`rospy.get_param('~joints/'+name+'/max_speed')` `Default: 0.0508`	
`self.dirty`	Dirty	`False`	True si hay alguna nueva posición actualizada
Methods	**Params**	**Return**	**Description**
`commandCb`	`req`	`void`	Comanda una posición

6.1.5 DynamixelServo

Tipo de articulación que representa a una articulación rotacional accionada por un servo-motor Dynamixel. Es una clase heredada de Joint y declara sus métodos.

Subscripciones:

- rospy.Subscriber(name+'/command', Float64, self.commandCb)

Servicios:

- rospy.Service(name+'/relax', Relax, self.relaxCb)
- rospy.Service(name+'/enable', Enable, self.enableCb)
- rospy.Service(name+'/set_speed', SetSpeed, self.setSpeedCb)

class DynamixelServo(Joint) – *abotix_python/src/abotix_python/servo_controller.py*			
Construct Params	**Content**	**Default Value**	**Description**
device	Instancia a ArbotiX		
name	Nombre de la articulación		
ns	Namespace	"~joints"	
Attributes	**Content**	**Init Value**	**Description**
n	Namespace	ns+"/"+name+"/"	Namespace del motor
self.id	Número id	rospy.get_param(n+"id")	Cada motor tiene su propio id
self.ticks	Num ticks	rospy.get_param(n+"ticks", 1024)	Ticks de movimiento del motor
self.neutral	Posición neutral	rospy.get_param(n+"neutral", self.ticks/2)	Tick neutral del motor. Por defecto, la mitad de ticks
self.range	Rango de movimiento	rospy.get_param(n+"range", self.range)	Rango de movimiento del motor en grados
self.rad_per_tick	Radianes por tick	radians(self.range)/self.ticks	Radianes por cada tick
self.min	Lower limit	rospy.get_param('~joints/'+name+'/min_position') Default: 0.0	Estos valores deberían obtenerse del URDF del robot.
self.max	Upper limit	rospy.get_param('~joints/'+name+'/max_position') Default: 0.5	
self.max_speed	Máxima velocidad	rospy.get_param('~joints/'+name+'/max_speed') Default: 0.0508	
self.invert	Sentido invertido	rospy.get_param(n+"invert",False)	Por defecto, sentido levógiro
self.readable	Legible	rospy.get_param(n+"readable",True)	True si el motor es legible
self.dirty	Dirty	False	True si hay alguna nueva posición actualizada
self.position	Posicion	0.0	Actual posición
self.desired	Deseado	0.0	Posición deseada
self.last_cmd	Último comando	0.0	Comando anterior
self.velocity	Velocidad	0.0	Velocidad de movimiento
self.enabled	Habilitado	True	¿Se puede comandar?
self.active	Activado	Fale	¿El torque está activado?
self.last	Tiemsatmp	rospy.Time.now()	Timestamp
self.reads	Lecturas	0.0	Contador lecturas
self.errors	Errores	0	Contador errores
self.total_reads	Lecturas totales	0.0	Lecturas totales
self.total_errors	Errores totales	[0.0]	Errores totals
self.voltage	Volatje	0.0	Voltaje del motor
self.temperature	Temperatura	0.0	Temperatura del motor
Methods	**Params**	**Return**	**Description**
angleToTicks	angle	ticks	Conversor ángulo-tick
ticksToAngle	ticks	angle	Conversor tick-ángulo
speedToTicks	rads_per_sec	ticks_per_sec	Conversor rad/s-ticks/s
enacleCb	req	void	Habilita o deshabilita los torques del motor
relax	req	void	Deshabilita los torques del motor
commandCb	req	void	Comanda al motor

setSpeedCb	req	void	Establece la velocidad del motor.

6.1.6 HobbyServo

Tipo de articulación que representa a una articulación rotacional accionada por un servo-motor Dynamixel. Es una clase heredada de Joint y declara sus métodos.

Subscripciones:

- rospy.Subscriber(name+'/command', Float64, self.commandCb)

class HobbyServo(Joint) – *abotix_python/src/abotix_python/servo_controller.py*			
Construct Params	**Content**	**Default Value**	**Description**
device	Instancia a ArbotiX		
name	Nombre de la articulación		
ns	Namespace	"~joints"	
Attributes	**Content**	**Init Value**	**Description**
n	Namespace	ns+"/"+name+"/"	Namespace del motor
self.id	Número id	rospy.get_param(n+"id")	Cada motor tiene su propio id
self.ticks	Num ticks	rospy.get_param(n+"ticks", 1024)	Ticks de movimiento del motor
self.neutral	Posición neutral	rospy.get_param(n+"neutral", self.ticks/2)	Tick neutral del motor. Por defecto, la mitad de ticks
self.range	Rango de movimiento	rospy.get_param(n+"range", self.range)	Rango de movimiento del motor en grados
self.rad_per_tick	Radianes por tick	radians(self.range)/self.ticks	Radianes por cada tick
self.min	Lower limit	rospy.get_param('~joints/'+name+'/min_position') Default: 0.0	Estos valores deberían obtenerse del URDF del robot.
self.max	Upper limit	rospy.get_param('~joints/'+name+'/max_position') Default: 0.5	
self.max_speed	Máxima velocidad	rospy.get_param('~joints/'+name+'/max_speed') Default: 0.0508	
self.invert	Sentido invertido	rospy.get_param(n+"invert",False)	Por defecto, sentido levógiro
self.readable	Legible	rospy.get_param(n+"readable",True)	True si el motor es legible
self.dirty	Dirty	False	True si hay alguna nueva posición actualizada
self.position	Posicion	0.0	Actual posición
self.desired	Deseado	0.0	Posición deseada
self.last_cmd	Último comando	0.0	Comando anterior
self.velocity	Velocidad	0.0	Velocidad de movimiento
self.enabled	Habilitado	True	¿Se puede comandar?
self.active	Activado	Fale	¿El torque está activado?
self.last	Tiemsatmp	rospy.Time.now()	Timestamp
n	Namespace	ns+"/"+name+"/"	Namespace del motor
self.id	Número id	rospy.get_param(n+"id")	Cada motor tiene su propio id
Methods	**Params**	**Return**	**Description**
angleToTicks	angle	ticks	Conversor ángulo-tick

ticksToAngle	ticks	angle	Conversor tick-ángulo
commandCb	req	void	Comanda al motor

6.1.7 Controller

Clase padre para la representación de un controlador.

class Controller() – *abotix_python/src/abotix_python/controllers.py*			
Construct Params	**Content**	**Default Value**	**Description**
device	Instancia a ArbotiX		
name	Nombre del controlador		
Attributes	**Content**	**Init Values**	**Description**
self.name	Nombre	name	Nombre del controlador
self.device	Dispositivo	device	Instancia a ArbotiX
self.fake	Simulación	device.fake	True si el controlador es simulado
self.pause	Pausado	False	True si el controlador está parado
self.joint_names	Nombres de las articulaciones	list()	
self.joint_positions	Posiciones de las articulaciones	list()	
self.joint_velocities	Velocidades de las articulaciones	list()	
Methods	**Params**	**Return**	**Description**
getDiagnostics	self	DiagnosticsStatus()	Obtiene un mensaje tipo diagnóstico de la articulación del controlador
startup	self	RFU	Enciende el controlador
shutdown	self	RFU	Apaga al controlador
update	self	RFU	Hace cualquier lectura/escritura al dispositivo
active	self	False (RFU)	¿El controlador está envíando comandos a la articulación?

6.1.8 FollowController

Tipo de controlador que perimite ejecutar trayectorias de tipo JointTrajectory. Es una clase heredada de Controller.

- Subscrito: `self.name+'/command'`

class FollowController(controller) – *abotix_python/src/abotix_python/follow_controller.py*			
Construct Params	**Content**	**Default Value**	**Description**
device	Instancia a ArbotiX		
name	Nombre del controlador		
Attributes	**Content**	**Init Value**	**Description**
self.interpolating		0	
self.rate	Frecuencia de publicación	rospy.get_param('~controllers/'+name+'/rate', 50.0)	La frecuencia con la que se publicará en el controlador.
self.joints	Articulaciones	rospy.get_param('~controllers/'+name+'/joints')	Lista de articulaciones del controlador
self.index	Índice del controlador	rospy.get_param('~controllers/'+name+'/index', len(device.controllers)	Cada controlador tiene su propio índice. Por defecto es el número al crearse.
self.server	ActionServer	name = rospy.get_param('~controllers/'+name+'/action_name', follow_joint_trajectory)	Creación del ActionServer para el controlador. Tipo: FollowJointTrajectoryAction

		`actionlib.SimpleActionServer ( name, FollowJointTrajectoryAction, execute_cb=self.actionCb, auto_start=False )`	Action_ns por defecto: follow_joint_trajectory
`self.executing`	Trayectoria ejecutandose	`False`	True si el controlador está ejecutando una trayectoria. False en caso contrario.
Methods	**Params**	**Return**	**Description**
`startup`	`self`	`void`	Inicia `self.server`

6.1.9 LinearControllerAbsolute

Tipo de controlador que perimite controlar ariculaciones lineales usando un controlador con posición absoluta retroaliemntada. Es una clase heredada de Controller.

`class LinearControllerAbsolute(controller)` – *abotix_python/src/abotix_python/linear_controller.py*			
Construct Params	**Content**	**Default Value**	**Description**
`device`	Instancia a ArbotiX		
`name`	Nombre del controlador		
Attributes	**Content**	**Init Value**	**Description**
`self.a`	Motor a	`rospy.get_param('~controllers/'+name+'/motor_a',29)`	Pin del motor a
`self.b`	Motor b	`rospy.get_param('~controllers/'+name+'/motor_b',30)`	Pin motor a
`self.p`	PWM	`rospy.get_param('~controllers/'+name+'/motor_pwm',31)`	Pin motor b
`self.analog`	Feedback	`rospy.get_param('~controllers/'+name+'/feedback',0)`	Pin de retroalimentación
`self.last`	Timestamp	`0`	Tiemstamp de la última actualización
`self.last_reading`	Timestamp	`0`	Tiemstamp de la última actualización
`self.delta`	Duración	`rospy.Duration(1.0/rospy.get_param('~controllers/'+ name+'/rate', 10.0))`	Duración del movimiento
`self.next`	Próximo timestamp	`rospy.Time.now() + self.delta`	Timestamp del próximo punto
`self.joint`	Articulación	`device.joints[rospy.get_param('~controllers/'+name+ '/joint')]`	Articulación a la que va ligada. Tipo class Joint()
Methods	**Params**	**Return**	**Description**

6.1.10 LinearControllerIncremental

Tipo de controlador que perimite controlar ariculaciones lineales usando un controlador sin encoder. Es una clase heredada de Controller.

`class LinearControllerIncremental(controller)` – *abotix_python/src/abotix_python/linear_controller.py*			
Construct Params	**Content**	**Default Value**	**Description**
`device`	Instancia a ArbotiX		
`name`	Nombre del controlador		
Attributes	**Content**	**Init Value**	**Description**
`self.a`	Motor a	`rospy.get_param('~controllers/'+name+'/motor_a',29)`	Pin del motor a

`self.b`	Motor b	`rospy.get_param('~controllers/'+name+'/motor_b',30)`	Pin motor a
`self.p`	PWM	`rospy.get_param('~controllers/'+name+'/motor_pwm',31)`	Pin motor b
`self.analog`	Feedback	`rospy.get_param('~controllers/'+name+'/motor_a',29)`	Pin de retroalimentación
`self.last`	Timestamp	`0`	Tiemstamp de la última actualización
`self.last_reading`	Timestamp	`0`	
`self.delta`	Duración	`rospy.Duration(1.0/rospy.get_param('~controllers/'+name+'/rate', 10.0))`	Timestamp de la duración del movimiento
`self.next`	Próximo timestamp	`rospy.Time.now() + self.delta`	Timestamp del próximo punto
`self.joint`	Articulación	`device.joints[rospy.get_param('~controllers/'+name+'/joint')]`	Articulación a la que va ligada. Tipo class Joint()
Methods	**Params**	**Return**	**Description**

6.1.11 ServoController

Clase que clasifica a las articulaciones del ArbotiX según el tipo de servomotor utilizado. Permite la interacción con todas al mismo tiempo para acciones como apagar torques, encender torques, mover todas las articulacioens a sus próximas posiciones, etc.

Clase general de controladores. Solo es llamada desde la clase ArbotixROS. Inicializa una instancia a Controller. Es una clase heredada de Controller.

Servicios:

- rospy.Service(name + '/relax_all', Relax, self.relaxCb): relajar todas las articulaciones.
- rospy.Service(name + '/enable_all', Enable, self.enableCb): habilitar todas las articulaciones.

`class ServoController(controller)` – *abotix_python/src/abotix_python/servo_controller.py*			
Construct Params	**Content**	**Default Value**	**Description**
`device`	Instancia a ArbotiX		
`name`	Nombre del controlador		Este valor será "servos" porque es el nombre por defecto a la hora de crearse una instancia a esta clase desde ArbotixROS
Attributes	**Content**	**Init Value**	**Description**
`self.dynamixels`	Dynamixel Servos	`list()`	Lista que contiene las articulaciones instancias a DynamixelServo
`self.hobbyservos`	Hobby Servos	`list()`	Lista que contiene las articulaciones instancias a HobbyServo
`self.iter`	Interacción	`0`	Contador de interacciones con los servos
`self.w_delta`	Duración escritura		Duración de escritura
`self.w_next`	Tmestamp escritura		Tiemstamp de la próxima escritura
`self.r_delta`	Duración lectura		Duración de lectura
`self.r_next`	Timestamp lectura		Tiemstamp de la próxima lectura
Methods	**Params**	**Return**	**Description**
`update`	`void`	void	Actualiza los servo motroes a las posiciones `self.r_next`
`getDiagnostics`	`void`	void	Obtiene un diagnóstico de los servos (voltaje, temperaturas)
`enableCb`	`void`	req	Encender/apagar los torques de los motores
`relaxCb`	`void`	req	Apagar todos los toruqes de los motores

6.1.12 DiffController

Tipo de controlador que perimite controlar motores para una configuración diferencial, frecuentemente usado en robots móviles. Es una clase heredada de Controller.

`class DiffController(controller)` – *abotix_python/src/abotix_python/diff_controller.py*

Construct Params	**Content**	**Default Value**	**Description**
`device`	Instancia a ArbotiX		
`name`	Nombre del controlador		
Attributes	**Content**	**Init Value**	**Description**
`self.rate`	Frecuencia de muestreo	`rospy.get_param('~controllers/'+name+'/rate',10)`	La frecuencia con la que se publicará en el controlador.
`self.timeout`	Timeout	`rospy.get_param('~controllers/'+name+'/timeout',1.0)`	Tiempo para conectarse a la placa
`self.t_delta`	Duración escritura		Duración de escritura
`self.t_next`	Tmestamp escritura		Tiemstamp de la próxima escritura
`self.ticks_meter`	Tics por metro	`rospy.get_param('~controllers/'+name+'/ticks_meter)`	Número de ticks que del motor que corresponde a un metro
`self.base_width`	Anchura de la base	`rospy.get_param('~controllers/'+name+'/base_width)`	Distancia entre las ruedas diferenciales
`self.base_frame_id`	ID de la frame base	`rospy.get_param('~controllers/'+name+'/base_frame_id)`	ID relacionadas con el URDF del robot
`self.odom_frame_id`	ID de la odometría	`rospy.get_param('~controllers/'+name+'/odom_frame_id)`	
`self.kp`	Constante proporcional	`rospy.get_param('~controllers/'+name+'/Kp)`	Constantes del controlador de los motores
`self.kd`	Constante derivativa	`rospy.get_param('~controllers/'+name+'/Kd)`	
`self.ki`	Constante integral	`rospy.get_param('~controllers/'+name+'/Ki)`	
`self.ko`	Constante ¿?	`rospy.get_param('~controllers/'+name+'/Ko)`	
`self.accel_limit`	Límite de aceleración	`rospy.get_param('~controllers/'+name+'/accel_limit, 0.1)`	Límite de aceleración
`self.joint_names`	Nombres de las articulaciones	`["base_l_wheel_joint","base_r_wheel_joint"]`	No son configurables
`self.joint_positions`	Posiciones de las articulaciones	`[0,0]`	Lecturas sobre el estado del robot.
`self.joint_velocities`	Velocidaes de las articulaciones	`[0,0]`	
`self.v_left`	Veloc motor izquierdo	`0`	
`self.v_right`	Veloc motor derecho	`0`	
`self.v_des_left`	Veloc deseada motor izquierdo	`0`	
`self.v_des_right`	Veloc deseada motor derecho	`0`	
`self.enc_left`	Lectura encoder motor izquierdo	`0`	
`self.enc_right`	Lectura encoder motor derecho	`0`	
`self.x`	Posición x del robot	`0`	
`self.y`	Posición y del robot	`0`	
`self.th`	Ángulo theta del robot	`0`	
`self.dx`	Velocidad x del robot	`0`	

`self.dr`	Velocidad y del robot	`0`	

6.2 arbotix_controllers

Se definen tres modelos de pinzas:

- TrapezoidGripperModel: pinzas de mordazas no paralelas con dos motores
- ParallelGripperModel: pinza de mordazas paralelas de un motor
- OneSideGripperModel: pinza paralela de un motor

Se definen tres tipos de controladores:

- OneSideGripperController: controlador para pinzas paralelas de un motor
- ParallelGripperController: controlador para pinzas paralelas de dos motores
- ParallelGripperController: controlador para pinzas de mordazas paralelas de un motor

Por alguna razón, no está definido el controlador para el modelo TrapezoidGripperModel y aparece un nuevo tipo de pinza: pinza paralela de dos motores.

Se definen dos Actions para el control de las pinzas:

- GripperActionController: action para controlar cualquiera de los modelos de pinza
- ParallelGripperActionController: action para controlar una pinza paralela de dos motores

GripperActionController es necesaria para controlar un modelo de pinza. A continuación, se muestran las configuraciones modelo-controlador-acción de una pinza permitida:

Modelo de pinza	Controlador	Action
TrapezoidGripperModel	???	GripperActionController
ParallelGripperModel	ParallelGripperController	GripperActionController
OneSideGripperModel	OneSideGripperController	GripperActionController
???	ParallelGripperController	ParallelGripperActionController

Tabla 6.1 Configuraciones modelo-controlador-acción de una pinza permitidas

El controlador ParallelGripperController se encuentra repetido. Se recomienda ignorar el uso del controlador para pinzas de mordazas paralelas de un motor ya que esta funcionalidad no parece estar terminada. Ignórese por lo tanto a ParallelGripperActionController.

6.2.1 TrapezoidGripperModel

Tipo de modelo de pinza que representa a una pinza de mordazas no paralelas con dos motores.

Publicaciones:

- `rospy.Publisher(self.joint_left+'/command'', Float64, queue_size=5)`
- `rospy.Publisher(self.joint_right+'/command'', Float64, queue_size=5)`

`class TrapezoidGripperModel()` – *abotix_controllers/bin/gripper_controller.py*			
Construct Params	**Content**	**Default Value**	**Description**
Attributes	**Content**	**Init Value**	**Description**
`self.pad_width`	Anchura de la base	`rospy.get_param('~pad_width, 0.01)`	Ancho entre los puntos de rotación de cada dedo
`self.finger_length`	Longitud del dedo	`rospy.get_param('~finger_length', 0.02)`	Longitud de los dedos al punto de cómputo

self.min_opening	Mínima apertura	rospy.get_param('~min_opening', 0.00)	Rango de movilidad de la pinza
self.max_opening	Máxima apertura	rospy.get_param('~max_opening', 0.09)	
self.center_l	Centro del dedo izquierdo	rospy.get_param('~center_left', 0.0)	Centros de cada dedo
self.center_r	Centro del dedo derecho	rospy.get_param('~center_right', 0.0)	
self.invert_l	Dedo izquierdo invertido	rospy.get_param('~invert_left', False)	True el motor de un dedo debe girar de forma invertida al del otro dedo.
self.invert_r	Dedo derecho invertido	rospy.get_param('~invert_right', False)	
self.left_joint	Dedo izquierdo	rospy.get_param('~joint_left', 'l_gripper_joint')	Nombre de la articulación del dedo
self.right_joint	Dedo derecho	rospy.get_param('~joint_right', 'r_gripper_joint')	
Methods	**Params**	**Return**	**Description**
setCommand	Self, command	True or False	Comanda a la pinza el parámetro *command*. Devuelve False si el valor está fuera del rango de movilidad de la pinza.
setPosition	Self, js	0.0	Devuelve la posición de las articulaciones nombradas en el vector js. Función por terminar
getEffort	Self, jointStates	1.0	Devuelve el par articular de la articulación. Función por hacer.

6.2.2 ParallelGripperModel

Tipo de modelo de pinza que representa a una pinza de mordazas paralelas de un motores.

Publicaciones:

- rospy.Publisher(self.joint +'/command'', Float64, queue_size=5)

class ParallelGripperModel() – *abotix_controllers/bin/gripper_controller.py*			
Construct Params	**Content**	**Default Value**	**Description**
Attributes	**Content**	**Init Value**	**Description**
self.center	Centro del rango	rospy.get_param('~center', 0.0)	Centro entre los dos dedos
self.scale	Escala de movimiento	rospy.get_param('~scale', 1.0)	Escala de movimiento
self.joint	Nombre de la articulación	rospy.get_param('~joint', 'gripper_joint')	Nómbre de la articulación
Methods	**Params**	**Return**	**Description**
setCommand	Self, command	None	Comanda a la pinza el parámetro *command*. Devuelve False si la el valor está fuera del rango de movilidad de la pinza.
setPosition	Self, jointStates	0.0	Devuelve la posición de las articulaciones nombradas en el vector jointStates. Función por hacer.
getEffort	Self, jointStates	1.0	Devuelve el par articular de la articulación. Función por hacer.

6.2.3 OneSideGripperModel

Tipo de modelo de pinza que representa a una pinza paralela de un motor.

Publicaciones:

- `rospy.Publisher(self.joint +'/command'', Float64, queue_size=5)`

`class OneSideGripperModel() – abotix_controllers/bin/gripper_controller.py`			
Construct Params	**Content**	**Default Value**	**Description**
Attributes	**Content**	**Init Value**	**Description**
`self.pad_width`	Anchura de la base	`rospy.get_param('~pad_width, 0.01)`	Ancho entre los puntos de rotación de cada dedo
`self.finger_length`	Longitud del dedo	`rospy.get_param('~finger_length, 0.02)`	Longitud de los dedos al punto de cómputo
`self.min_opening`	Mínima apertura	`rospy.get_param('~min_opening, 0.00)`	Rango de movilidad de la pinza
`self.max_opening`	Máxima apertura	`rospy.get_param('~max_opening, 0.09)`	
`self.center`	Centro del dedo izquierdo	`rospy.get_param('~center', 0.0)`	Centro entre los dedos
`self.invert`	Dedo izquierdo invertido	`rospy.get_param('~invert_left', False)`	True si movimiento invertido del motor
`self.joint`	Dedo izquierdo	`rospy.get_param('~joint_left, 'l_gripper_joint')`	Nombre de la articulación del dedo
Methods	**Params**	**Return**	**Description**
`setCommand`	`Self, command`	`True or False`	Comanda a la pinza el parámetro *command*. Devuelve False si la el valor está fuera del rango de movilidad de la pinza.
`setPosition`	`Self, jointStates`	`0.0`	Devuelve la posición de las articulaciones nombradas en el vector `jointStates`. Función por terminar
`getEffort`	`Self, jointStates`	`1.0`	Devuelve el par articular de la articulación. Función por hacer.

6.2.4 OneSideGripperController

Tipo de controlador para pinzas paralelas de un motor.

Subscripciones:

- `rospy.Subscriber('~command', Float64, self.commandCb)`

Publicaciones:

- rospy.Publisher("gripper_joint/command", Float64, queue_size=5)

`class OneSideGripperController() – abotix_controllers/bin/one_side_gripper_controller.py`			
Construct Params	**Content**	**Default Value**	**Description**
Attributes	**Content**	**Init Value**	**Description**
`self.pad_width`	Anchura de la base	`rospy.get_param('~pad_width, 0.01)`	Ancho entre los puntos de rotación de cada dedo
`self.finger_length`	Longitud del dedo	`rospy.get_param('~finger_length, 0.02)`	Longitud de los dedos al punto de cómputo
`self.center`	Centro del dedo izquierdo	`rospy.get_param('~center', 0.0)`	Centro entre los dedos
`self.invert`	Dedo izquierdo invertido	`rospy.get_param('~invert_left', False)`	True si movimiento invertido del motor
Methods	**Params**	**Return**	**Description**

CommandCb	Self, msg	None	Comanda a la pinza el parámetro *msg*. Devuelve None si la el valor está fuera del rango de movilidad de la pinza.

6.2.5 ParallelGripperController

Tipo de controlador para pinzas paralelas de dos motores.

Publicaciones:

- rospy.Publisher("l_gripper_joint/command", Float64, queue_size=5)
- rospy.Publisher("r_gripper_joint/command", Float64, queue_size=5)

Subscripciones:

- rospy.Subscriber("~command", Float64, self.commandCb)

class ParallelGripperController() – *abotix_controllers/bin/parallel_gripper_controller.py*			
Construct Params	**Content**	**Default Value**	**Description**
Attributes	**Content**	**Init Value**	**Description**
self.pad_width	Anchura de la base	rospy.get_param('~pad_width, 0.01)	Ancho entre los puntos de rotación de cada dedo
self.finger_length	Longitud del dedo	rospy.get_param('~finger_length', 0.02)	Longitud de los dedos al punto de cómputo
self.min_opening	Mínima apertura	rospy.get_param('~min_opening', 0.00)	Rango de movilidad de la pinza
self.max_opening	Máxima apertura	rospy.get_param('~max_opening', 0.09)	
self.center_l	Centro del dedo izquierdo	rospy.get_param('~center_left', 0.0)	Centros de cada dedo
self.center_r	Centro del dedo derecho	rospy.get_param('~center_right', 0.0)	
self.invert_l	Dedo izquierdo invertido	rospy.get_param('~invert_left', False)	True el motor de un dedo debe girar de forma invertida al del otro dedo.
self.invert_r	Dedo derecho invertido	rospy.get_param('~invert_right', False)	
Methods	**Params**	**Return**	**Description**
CommandCb	Self, msg	None	Comanda a la pinza el parámetro *msg*. Devuelve None si la el valor está fuera del rango de movilidad de la pinza.

6.2.6 ParallelGripperController

Tipo de controlador para pinzas paralelas de dos motores.

Publicaciones:

- rospy.Publisher('gripper_joint/command'', Float64, queue_size=5)

Subscripciones:

- rospy.Subscriber("~command", Float64, self.commandCb)
- rospy.Subscriber("joint_states", JointState, self.stateCb)

class ParallelGripperController() – *abotix_controllers/bin/parallel_gripper_controller.py*

Construct Params	**Content**	**Default Value**	**Description**
Attributes	**Content**	**Init Value**	**Description**
`self.calib`	Calibración	`{0.0000:1.8097, 0.0159:1.2167, 0.0254:0.8997, 0.0381:0.4499, 0.042:0.1943}`	
`self.min`	Mínimo	`rospy.get_param('~min, 0.0)`	
`self.max`	Máximo	`rospy.get_param('~max, 0.042)`	
`self.center`	Centro	`rospy.get_param('~center, 512)`	
`self.invert`	Invertido	`rospy.get_param('~invert, False)`	
Methods	**Params**	**Return**	**Description**
`getCommand`		None	Obtiene un comando de entrada para comandar a la pinza
`getWigth`		`Apertura de la pinza`	Devuelve la apertura de la pinza
`commandCb`	`Self, msg`	None	Comanda a la pinza el parámetro *msg*. Devuelve None si la el valor está fuera del rango de movilidad de la pinza.
`stateCb`	`Self, msg`	None	Obtiene el estado de la articulación

6.2.7 GripperActionController

Acción necesaria para el control de un modelo de pinza.

Subscripciones:

- `rospy.Subscriber('joint_states', JointState, self.stateCb)`

Acciones:

- actionlib.SimpleActionServer('~gripper_action', GripperCommandAction, execute_cb=self.actionCb, auto_start=False)

`class GripperActionController() – abotix_controllers/bin/gripper_controller.py`			
Construct Params	**Content**	**Default Value**	**Description**
Attributes	**Content**	**Init Value**	**Description**

`self.model`	Modelo de pinza	`rospy.get_param('~model')`
		
		`if model == "dualservo": self.model = TrapezoidGripperModel()`
		
		`elif model == "parallel": self.model = ParallelGripperModel()`
		
		`elif model == "singlesided": self.model = OneSideGripperModel()`
			Modelo de la pinza a controlar. Puede ser uno entre los siguientes:
			• *dualservo* [TrapezoidGripperModel()]
			• *parallel* [ParallelGripperModel()]
			• *singlesided* [OneSideGripperModel()]

Methods	**Params**	**Return**	**Description**
`actionCb`	`Self, goal`	None	Se comanda la entrada *goal* al motor para alcanzar la posición.
`stateCb`	`Self,msg`	None	Devuelve el estado de las articulaciones de la pina.

6.2.8 ParallelGripperActionController

Tipo de modelo de pinza que representa a una pinza de mordazas no paralelas con dos motores.

Publicaciones:

- `rospy.Publisher("l_gripper_joint/command", Float64, queue_size=5)`
- `rospy.Publisher("r_gripper_joint/command", Float64, queue_size=5)`

Acciones:

- `actionlib.SimpleActionServer('~gripper_action', GripperCommandAction, execute_cb=self.actionCb, auto_start=False)`

`class ParallelGripperActionController()` – *abotix_controllers/bin/gripper_gripper_action_controller.py*			
Construct Params	**Content**	**Default Value**	**Description**
Attributes	**Content**	**Init Value**	**Description**
`self.pad_width`	Anchura de la base	`rospy.get_param('~pad_width, 0.01)`	Ancho entre los puntos de rotación de cada dedo
`self.finger_length`	Longitud del dedo	`rospy.get_param('~finger_length', 0.02)`	Longitud de los dedos al punto de cómputo
`self.min_opening`	Mínima apertura	`rospy.get_param('~min_opening', 0.00)`	Rango de movilidad de la pinza
`self.max_opening`	Máxima apertura	`rospy.get_param('~max_opening', 0.09)`	
`self.center_l`	Centro del dedo izquierdo	`rospy.get_param('~center_left', 0.0)`	Centros de cada dedo
`self.center_r`	Centro del dedo derecho	`rospy.get_param('~center_right', 0.0)`	
`self.invert_l`	Dedo izquierdo invertido	`rospy.get_param('~invert_left', False)`	True el motor de un dedo debe girar de forma invertida al del otro dedo.
`self.invert_r`	Dedo derecho invertido	`rospy.get_param('~invert_right', False)`	
Methods	**Params**	**Return**	**Description**
`actionCb`	`Self, goal`	`None`	Se comanda la entrada *goal* al motor para alcanzar la posición.

7 Referencias

Hansen, K. D. (2015). *Getting Started with the Arbotix-M*. Obtenido de https://github.com/AalborgUniversity-ControlLabs/start-here/blob/master/crust-crawler-arms/getting-started-with-arbotix-m.md

RobotnikAutomation. (2019). *https://github.com/RobotnikAutomation/phantomx_reactor_arm*. Obtenido de phantomx_reactor_arm.