Compré este kit de robótica para montarlo con mis hijos con la intención de emplearlo como kit robótico de simulación para robots de cuatro patas.

El kit es realmente económico, teniendo en cuenta que tiene 16 servos con capacidad de posición precisa y movimiento continuo, además de una unidad de control por bluetooth y una batería.

A partir del trabajo referenciado al final del documento se ha continuado con el estudio del protocolo bluetooth hasta conseguir diseccionar el paquete de control de servos para poder establecer la posición de un número arbitrario de servos para controlar un robot como el de la imagen con 16 servos.

Código de ejemplo de conexión Bluetooth:

import bluetooth

from bluetooth import Protocols

from alpha\_1s import Command

def main():

msg = message(b'\x09', [b'\x00',b'\x00',b'\xff',b'\xff',b'\x01',b'\x01',b'\x01',b'\x01',b'\x01',b'\x01',b'\x01',b'\x01',b'\x01',b'\x01',b'\x01',b'\x01',b'\x01',b'\x01',b'\x01',b'\x01',b'\x20',b'\x01',b'\x79'])

print(msg)

bd\_addr = discover()

#bd\_addr = '88:1B:99:0A:B5:4F'

if bd\_addr:

port = 6

sock = bluetooth.BluetoothSocket(Protocols.RFCOMM)

sock.connect((bd\_addr, port))

print('Connected')

sock.settimeout(60.0)

sock.send(msg)

print('Sent data')

response = sock.recv(1024)

print("Salida:")

print(Command().get(response))

sock.close()

def message(command, parameters):

header = b'\xFB\xBF'

end = b'\xED'

parameter = b''.join(parameters)

# len(header + length + command +parameters + check)

length = bytes([len(parameters)+5])

data = [command, length]

data.extend(parameters)

control = [sum(ord(x) for x in data) % 256]

check = bytes(control)

return header+length+command+parameter+check+end

def discover():

print("searching ...")

nearby\_devices = bluetooth.discover\_devices(lookup\_names=True)

print("found %d devices" % len(nearby\_devices))

for addr, name in nearby\_devices:

if name == "Jimu\_B54F":

return addr

if \_\_name\_\_ == '\_\_main\_\_':

main()

Componentes de los paquetes de comunicación:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| header | length | Command | parameters | check | end |

def message(command, parameters):

header = b'\xFB\xBF'

end = b'\xED'

parameter = b''.join(parameters)

# len(header + length + command +parameters + check)

length = bytes([len(parameters)+5])

data = [command, length]

data.extend(parameters)

control = [sum(ord(x) for x in data) % 256]

check = bytes(control)

return header+length+command+parameter+check+end

Ejemplo de payload (command+parameters)

#msg = message(b'\x22', [b'\x00',b'\x10',b'\x70',b'\x10',b'\x00'])

#msg = message(b'\x09', [b'\x00',b'\x00',b'\x00',b'\x1c',b'\x01',b'\x01',b'\x01',b'\x20',b'\x01',b'\x79'])

#msg = message(b'\x09', [b'\x00',b'\x00',b'\x00',b'\x1d',b'\x01',b'\x01',b'\x90',b'\x90',b'\x01',b'\x79'])

#msg = message(b'\x07', [b'\x02',b'\x01',b'\x01',b'\x78'])

#msg = message(b'\x07', [b'\x01',b'\x01',b'\x01',b'\x01',b'\x00'])

#cmd 0x07 (modo x01 (mover s) modo x04 (todos menos s), modo x05 (todos))

#activa todas las patas

#msg = message(b'\x09', [b'\x00',b'\x00',b'\x0f',b'\xff',b'\x01',b'\x01',b'\x01',b'\x01',b'\x01',b'\x01',b'\x01',b'\x01',b'\x01',b'\x01',b'\x01',b'\x01',b'\x20',b'\x01',b'\x79'])

#Activa todas las patas y la trompa del elefante

#msg = message(b'\x09', [b'\x00',b'\x00',b'\xff',b'\xff',b'\x01',b'\x01',b'\x01',b'\x01',b'\x01',b'\x01',b'\x01',b'\x01',b'\x01',b'\x01',b'\x01',b'\x01',b'\x01',b'\x01',b'\x01',b'\x01',b'\x20',b'\x01',b'\x79'])

Contenido de payload:

Control de servos de rotación continua:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Comando | modo | Num.servo | Dirección | Vel1 | Vel2 |
| 0x07 | 0x02 | 0x01 | 0x01 | 0x01 | 0x00 |

Modo

|  |  |
| --- | --- |
| 0x01 | Mueve el servo indicado |
| 0x04 | Mueve todos menos el indicado |
| 0x05 | Mueve todos los servos |

Dirección

|  |  |
| --- | --- |
| 0x01 | Avance |
| 0x02 | Retroceso |

Control de posición de servos múltiples:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Cmd | Servo1 | Servo2 | Servo3 | Servo4 | posiciones | tiempo | End1 | End2 |
| 0x09 | 0x00 | 0x00 | 0xff | 0xff | 1-16bytes | 1byte | 0x01 | 0x79 |

|  |  |
| --- | --- |
| **Servo3** | **Servo4** |
| **0000 0000** | **0000 0000** |

Los valores de las posiciones oscilan entre 0x01 y 0xB4 (180 grados)

Cada posición binaria representa un servo, si está a 1 se proporciona su posición en la zona de datos de posiciones, si está a 0 no se asigna valor y no se especifica en la zona de posiciones.

Tiempo: tiempo en que los servos alcanzan la posición indicada.

Para el robot jimu elefante, Servo3 asigna los servos de la derecha visto desde atrás y Servo4 asigna los servos de la izquierda.

Ejemplo de payload válido como posición inicial:

# 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

msg = message(b'\x09', [b'\x00',b'\x00',b'\xff',b'\xff',b'\x70',b'\x18',b'\x5a',b'\x70',b'\x18',b'\x5a',b'\x78',b'\x14',b'\x35',b'\x78',b'\x14',b'\x35',b'\x73',b'\xa0',b'\x5a',b'\x5a',b'\x20',b'\x01',b'\x79'])

Hay que desmonta todos los servos, enviar el mensaje de posicionamiento inicial y volver a montar los servos de forma que se consiga la siguiente forma del robot:

Información del contenido de los paquetes de los robots Jimu:

Fuente: <https://github.com/msantang78/node-jimu> (https://www.minds.com/newsfeed/1245908881519476736?referrer=msantangelo)

7,8,9 10,11,12 <- arriba

|-----------|

| |

13--| |

14 |-----------|

15 1,2,3 4,5,6 <- abajo

16

# Packet format

The packages are composed by the header, the payload, the checksum, and an ending byte;

## Header

0xfb, 0xbf

[0xfb, 0xbf]

## Payload

packet length, 1 byte command, ...n bytes params

[0x07, 0x0B, 0x00, 0x00]

## Packet end

checksum + 0xed

The checksum is the sum of all the bytes of the packet.length + payload

[0x9a, 0xed]

## Full package example

[0xfb, 0xbf, 0x07, 0x0b, 0x00, 0x00, 0x0b, 0xed]

# Commands

## Controlling wheels

Command 07

### one servo

params 0x01 [motor 1 byte][direction 1 byte][velocity 2 bytes]

mode 1 control one servo

direction 1-2

velocity 16 bits

cmd mode s d v1 v2

const payload = [0x0A, 0x07, 0x01, 0x01, 0x02, 0x01, 120];

> remember that the servos are inverted so if you send a command to the 2 wheels servos they will run in opposite direction

## Controlling servo position

const payload = [

0x09,

0,

0,

0,

28,

205, // motor central

80, // brazo izquierdo

130, // brazo derecho

30, // velocidad

1, // ?

121 // ?

];

## Control eyes colors

### all lights

const payload = [

0x79,

0x04,

0x03, // eyes 1, 2, 3 both

0x0A, // time

0x01, // number of colors

0xFF, // light mask

0x35, // color

0x35, // color

0x35 // color

]

### custom colors each light

const payload = [

0x79,

0x04,

0x03, // eyes

0xFF, // time

0x05, // number of colors

0x11, // light mask

0xFF, // color

0xF0, // color

0x00, // color

0x0A, // light mask

0xFF, // color

0x80, // color

0x00, // color

0x04, // light mask

0xFF, // color

0x00, // color

0x00, // color

0xA0, // light mask

0x00, // color

0xFF, // color

0xFF, // color

0x40, // light mask

0x35, // color

0x35, // color

0x35, // color

]

### animation eyes

const payload = [

0x78, // command

0x04,

0x03, // eye 1, 2, 3 both

0x02, // animation

0x00, // repetition 16bits?

0x01, // repetitions

0x40, // RR

0x40, // GG

0xFF // BB

]

### IR sensor status request

const payload = [

0x08,

0x7E,

0x01,

0x01,

0x01

]

// read response

characteristics[1].on('data', function (data, isNotification) {

console.log(data)

console.log('NOTIFICATION', data.readInt32BE(10));

});

### read positions

070B 0000

070B 0001

// read response

characteristics[1].on('data', function (data, isNotification) {

// last payload byte is the position of the servo (one response by servo)

console.log(data)

});