UNIVERSITAT POLITÈCNICA DE VALÈNCIA

SEPTIEMBRE 2019

TRABAJO FIN DE MÁSTER

DESARROLLO DE UN SISTEMA DE VISIÓN ACTIVO EN ROBOT BIOLOID

AUTOR:

ANDRÉS MESEGUER VALENZUELA

TUTOR:

JUAN FRANCISCO BLANES NOGUERA

MÁSTER EN AUTOMÁTICA E INFORMÁTICA INDUSTRIAL

Agradecimientos

Debo dar gracias a mi tutor Paco Blanes por su paciencia y ayuda en todo lo concernido a este trabajo y al máster en general. También a mi familia por tener paciencia conmigo al llenar de trastos la casa y a las demás personas que han puesto su granito de arena en hacer realidad este trabajo.

Va por ti Alguñanera, va por ti mi pequeño rubiales.

Vencer es convencer, y hay que convencer sobre todo. Pero no puede convencer el odio que no deja lugar a la compasión, ese odio a la inteligencia, que es crítica y diferenciadora, inquisitiva (mas no de inquisición).

Miguel de Unamuno. Núñez Florencio, Rafael (2014). «Encontronazo en Salamanca: "Venceréis pero no convenceréis"». La Aventura de la Historia 184: 35-39. ISSN 1579-427X

RESUMEN

En el presente proyecto se pretende desarrollar un sistema de percepción para un robot humanoide Bioloid, basado en una cámara como sensor principal, complementado con una IMU para reforzar la información de visión. Para el mismo se ha trabajado tanto en el desarrollo de software de control sobre la Beaglebone Black, como en el diseño de sistemas electrónicos específicos creando una placa donde integrarlos.

RESUM

En aquest projecte, es pretén desenvolupar un sistema de percepció per a un robot humanoide Bioloid, basat en una càmera com a sensor principal, complementat amb una IMU per a reforçar la informació de visió. Per a aquest mateix, s'ha treballat tant en el desenvolupament del software de control sobre la Beaglebone Black, com en el disseny de sistemes electrònics específics creant una placa on integrar-los.

ABSTRACT

This project aims to develop a perception system for a humanoid Bioloid robot, based on a camera as main sensor, and complemented by an IMU to reinforce vision information.

For it, we have focused on the development of control software on the Beaglebone Black, and also on the design of specific electronic systems in order to create a board where to integrate them.

Contenido

ILUSTRACIONES

1. Introducción

1.1. Visión general

Durante el pasado siglo XX se han ido desarrollando diversos modelos de robots humanoides [1], hasta el punto de que cualquier persona puede comprar una versión sencilla para poder poner en práctica conocimientos de robótica, además de personalizar al gusto el robot, extendiendo así, los campos de conocimientos que se emplean entorno a la robótica, como puede ser en este caso, la electrónica y la visión artificial.

En este trabajo, por lo tanto, no se van a explicar conceptos o desarrollos relacionados con la dinámica o cinemática del robot empleado, sino que se va a partir de la base: **sabiendo como mover el robot, llevar a cabo una personalización de este para obtener la mayor información del entorno que sea posible**.

Por lo tanto, se va a hablar de circuitos electrónicos para sensorizar (*Electrónica*). De obtención y tratamiento de imágenes (*Visión Artificial*), además de las reacciones ante toda esta información en un microcontrolador (*Informática Industrial*) comunicándose este con el robot humanoide.

1.2. Objetivo general

Ante esta situación, se pretende llevar a cabo una relación de causa y acción entre la cámara y el circuito de sensorización, el microcontrolador y la salida que será tanto el movimiento o comportamiento del robot como la publicación o extracción de la información recibida.

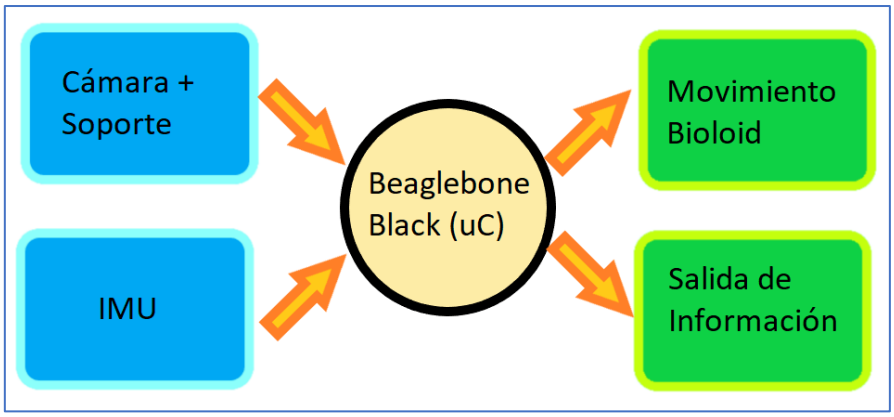

Ilustración 1: Esquema del Trabajo

Por lo tanto, al apartado de Visión del título del TFM, hay que añadir una etapa de sensorización.

1.3. Estado del Arte

El concepto de robot humanoide empezó a desarrollarse a principios del siglo XX, en el mundo del cine, en películas como **Metropolis** (1926), hasta que pasaron 47 años para poder observar uno de los primeros modelos reales bípedos, como lo fue el **Wabot 1** (1973) [1].

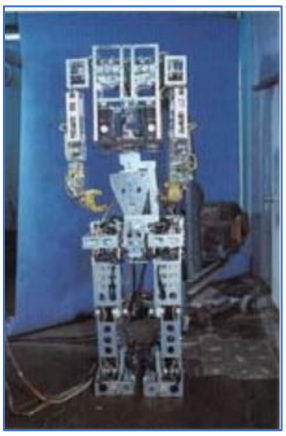

Ilustración 2: Robot Wabot 1

Desde entonces se han creado o descubierto nuevos materiales a la hora de fabricarlos, además de nuevos lenguajes de programación y sobre todo, una mejora en la potencia del cálculo, de modo que a día de hoy, se disponen de Sistemas Empotrados con CPU que trabajan a 1 GHz e incluso frecuencias más altas. Por lo tanto, es posible incorporar todo tipo de sensores y dispositivos de adquisición de imágenes.

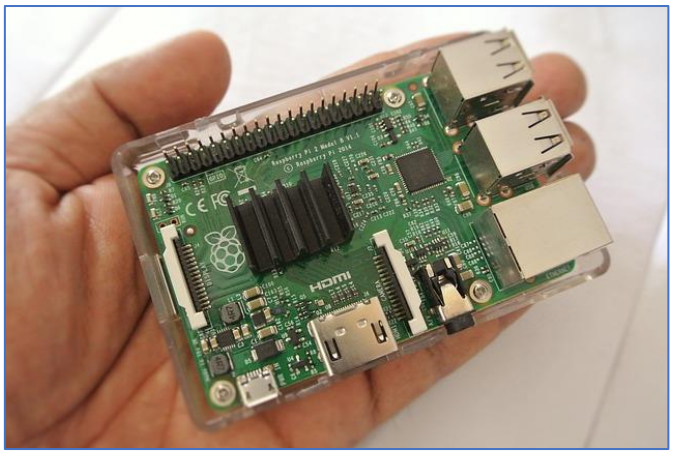

Ilustración 3: Sistema Empotrado Raspberry Pi

Las mejoras en el campo del robot humanoide, hacen posible el abaratamiento de estos robots para poder emplearlo en prácticas académicas, como pueden ser torneos, clases prácticas y para hacer trabajos.

Ilustración 4: Kit Bioloid [2]

Por lo tanto, se dispone de un Kit de un robot ya montado, el cual debe personalizarse. Además, el hecho de que haya sido utilizado previamente permite emplear material desarrollado por otros compañeros en anteriores cursos, en cuanto a cuestiones del movimiento y programación de la Visión Artificial.

1.4. Justificación

Dado que se ya se dispone de un estado de la cuestión, con proyectos anteriores para manejar la comunicación con el Robot y su movimiento, crear la personalización del robot va a poner en práctica los conocimientos adquiridos no solo en el máster de Automática e Informática Industrial, sino también, los adquiridos en el Grado de Ingeniería Electrónica Industrial y Automática.

1.5. Alcances y Límites

- Potencia de cálculo limitada del microcontrolador.
- Precisión de las matrices de rotación del robot. Rozamientos del entorno.
- Alcance del componente o módulo bluetooth empleado.
- Precisión del sensor IMU.
- Estabilidad del soporte de la cámara.

1.6. Capítulos

A continuación, se explican las diversas partes que componen este trabajo para poder entender la estructura de este documento:

- **Objetivos**: Un breve listado de las metas que se pretenden alcanzar en el TFM.
- **Desarrollo**: Este capítulo se encarga de descomponer las distintas partes del trabajo de modo que se pueda entender como se han podido realizar las partes más esenciales del proyecto.
- **Pruebas Realizadas**: Un breve resumen de las pruebas acometidas durante el proceso de desarrollo y la prueba final.
- **Manual de Usuario**: Se trata de una corta explicación para entender como se usa la aplicación de escritorio.
- **Presupuesto de la Cape**: Es, en definitiva, un cálculo del coste aproximado que ha tenido el hacer, un prototipo de la PCB personalizada.
- **Conclusión**: Una breve reflexión para finalizar con el trabajo y destacar detalles importantes.

- **Anexos**: Son documentos importantes que forman parte del desarrollo para poder explicar de forma más exhaustiva la creación del proyecto.

2. Objetivos

Diseño y desarrollo de un sistema de percepción para un robot Bioloid que permita tomar decisiones de control (específicamente sobre trayectoria de movimientos) a partir de los mismos. A partir de este objetivo se derivan unos subobjetivos:

2.1. Subobjetivos

- Diseño y desarrollo de un sistema de adquisición de imágenes usando la cámara UI-1007XS-C con la Beaglebone Black.
- Diseño de electrónica específica (cape) para la Beaglebone Black que soporte una IMU.
- Planteamiento y despliegue de sistema de comunicación entre el robot y aplicaciones externas.
- Proyección de un soporte para la cámara que permita cambios en su configuración física.
- Validar y realizar pruebas con la solución construida.

Establecidos estos objetivos, es posible empezar a explicar como se han logrado o no.

3. Desarrollo

Son varias las partes a desarrollar, de modo que, para visualizarlas, se muestra la siguiente ilustración:

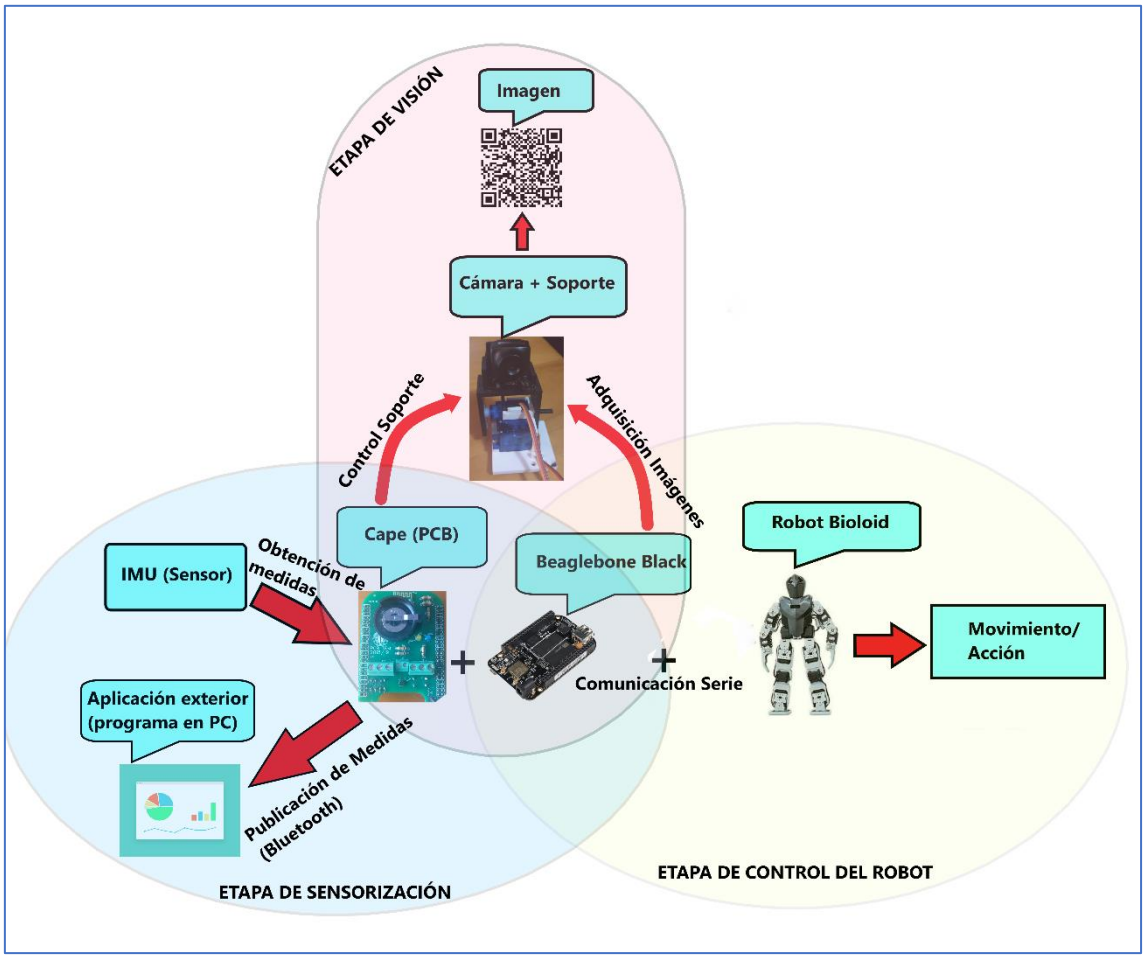

Ilustración 5: Esquema del proyecto

A partir de aquí se explican los desarrollos de las diferentes partes, haciendo hincapié tanto en el código como en la comunicación entre las partes.

3.1. Etapa de visión artificial

El proceso de adquisición de imágenes se realiza durante un movimiento que ejerce el soporte de la cámara para poder observar el entorno desde tres ángulos diferentes. De modo que durante este proceso se toman tres imágenes enumeradas en el siguiente esquema.

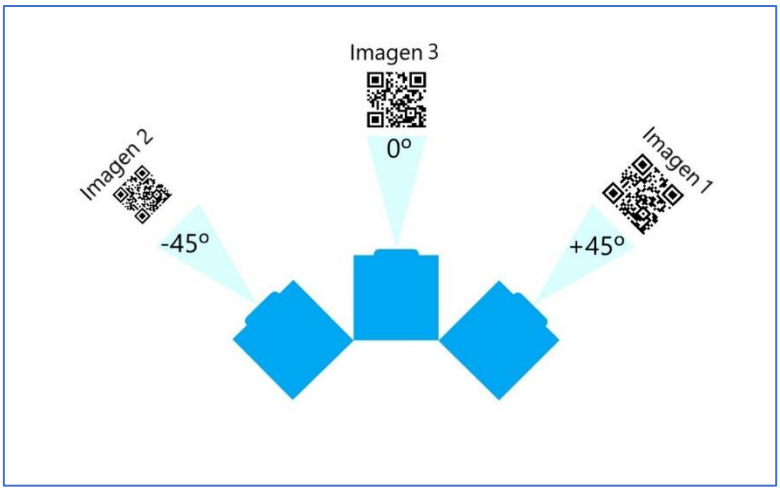

Ilustración 6: Posición de la cámara en el plano horizontal.

Primero se toma la imagen situada en el grado 45 para posteriormente rotar en contra de las agujas del reloj hasta los 45 grados negativos, tomando en esta segunda posición la segunda imagen. Finalmente se rotan 45 grados en sentido a favor de las agujas del reloj para tomar la tercera imagen.

Dicho procedimiento que combina la toma de imágenes con el movimiento de la cámara y el posterior movimiento del robot, hace que se consiga un **control guiado por visión**.

Este procedimiento se repite cada vez que se para el robot y se estabiliza. Para poder llevarlo a cabo se ha diseñado un soporte para la cámara que lleva integrado dos servomotores que ofrecen por lo tanto dos grados de libertad.

Además del movimiento en el plano observado en la Imagen 5 hay que añadir el movimiento en el plano vertical a este, de modo que el segundo grado de libertad ofrece la posibilidad de reclinar hacia delante o dejar el soporte centrado, tal y como indica la siguiente imagen.

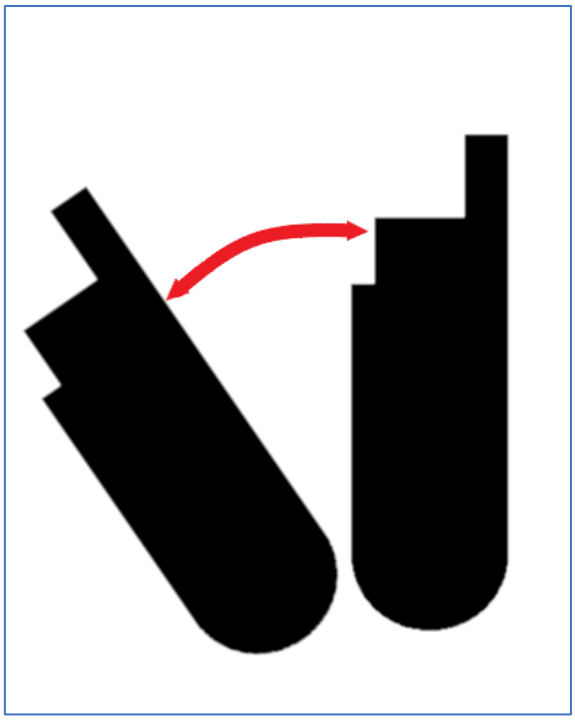

Ilustración 7:Rotación del soporte en el plano Vertical

3.1.1. Proceso de captación de imagen

Tal y como se indica en el apartado de Instalación de la Beaglebone, es necesario instalar los drivers de la cámara para poder crear código empleable con ella. El modelo de cámara empleado es el UI-1007XS-C fabricado por iDS [3].

Ilustración 8: Cámara utilizada

El código para poder obtener imágenes constantemente sigue estos pasos:

En primer lugar, se inicializa la cámara y se seleccionan los parámetros principales de configuración como pueden ser el formato de salida *(png, jpg...*), la calidad (*de 0 a 100*), la anchura de los píxeles de entrada (*Pixel Clock*) o la resolución cuyo valor escogido es 680 x 480 debido a que es la más pequeña que permite adquirir la cámara, de modo que se pierda el menor tiempo posible a la hora de por ejemplo, decodificar códigos QR a expensas de definición en la imagen.

```
nRet = is_InitCamera(&hCam, NULL); //Inicia el driver y la comunicación con la cámara

if(nRet != IS_SUCCESS){ //Si no se ha iniciado correctamente, sal del programa
    cout << "Error al iniciar la camara. " << endl;
    cout << nRet << endl;
    return 0;
}
else{cout << "Camara Iniciada." << endl;}
   //Ajuste de anchura de los píxels de entrada
   nRet = is_PixelClock(hCam, IS_PIXELCLOCK_CMD_SET,(void*)&val_pclock, sizeof(val_pclock));

   if(nRet == IS_SUCCESS) cout << "Exito al modificar el valor de pixelclock" << endl;
   else cout << "Fallo al modificar el valor de pixelclock. Error no: " <<nRet << endl;

   //Se consigue una lista de los formatos posibles
   nRet = is_ImageFormat(hCam, IMGFRMT_CMD_GET_NUM_ENTRIES, &count,4);
   bytesNeeded += (count - 1) * sizeof(IMAGE_FORMAT_INFO);
   void* ptr = malloc(bytesNeeded);
   IMAGE_FORMAT_LIST* pformatList = (IMAGE_FORMAT_LIST*) ptr;
   pformatList->nSizeOfListEntry = sizeof(IMAGE_FORMAT_INFO);
   pformatList->nNumListElements = count;
   nRet = is_ImageFormat(hCam, IMGFRMT_CMD_GET_LIST, pformatList, bytesNeeded);


   nRet = is_PixelClock(hCam,IS_PIXELCLOCK_CMD_GET,(void*)&val_pclock,sizeof(val_pclock));
   cout << "El valor actual del pixelclock es: " << val_pclock;
   cout << endl << " Comenzando captura..." << endl << endl;
   IMAGE_FORMAT_INFO formatInfo;
   IMAGE_FILE_PARAMS file_param;
   //Especificación del nombre de la imagen
   sprintf(nombre, "./captura.png",captura);

   const size_t tam = strlen(nombre)+1;
   wchar_t* file_name = new wchar_t[tam];
   mbstowcs (file_name, nombre, tam);

   //Propiedades del archivo
```

```
file_param.pwchFileName = file_name;
file_param.pnImageID = NULL;
file_param.ppcImageMem = NULL;
file_param.nQuality = 100; //80; //Mejora un poco la nitidez
file_param.nFileType = IS_IMG_PNG;
//Establecimiento de anchura y altura según el formato de imagen
formatInfo = pformatList->FormatInfo[9];

int width = formatInfo.nWidth;
int height = formatInfo.nHeight;
cout << formatInfo.nWidth << "x" << formatInfo.nHeight << endl;
```

El segundo paso es la acción en sí de tomar la imagen, donde se reserva un espacio de memoria y en caso de que el alojamiento u obtención de la imagen no se haya podido realizar satisfactoriamente, se sale del bucle para finalizar con el programa:

```
while (nRet == IS_SUCCESS){



    nRet = is_AllocImageMem(hCam, width, height, 24, &pMem,(int*) &memID);
    if (nRet == IS_SUCCESS){
    //cout << "Memoria localizada con exito. ID no: " << memID << endl;
    }
    else break;

    //Activación de la memoria
    nRet = is_SetImageMem(hCam, pMem, memID);
    if (nRet == IS_SUCCESS)
       //cout << "Memoria activada ID no: " << memID << endl;

    //Se especifica el formato de la nueva imagen
    nRet = is_ImageFormat(hCam, IMGFRMT_CMD_SET_FORMAT, &formatInfo.nFormatID, 4);
    //Adquisición de la imagen
    nRet = is_FreezeVideo(hCam, IS_WAIT);



    //Guardado de la imagen
    nRet = is_ImageFile(hCam, IS_IMAGE_FILE_CMD_SAVE, (void*)&file_param,
sizeof(file_param));
    if(nRet != 0){
       //cout << "Error al guardar la imagen. No: "<< nRet << endl << endl;
       is_FreeImageMem (hCam,pMem,memID);
       is_ExitCamera(hCam);
       break;
    }
```

El tercer y último paso consiste en la aplicación del apartado de Visión, que se explica en el punto de Pruebas Realizadas. Dicha aplicación se basa en la decodificación de QRs obteniendo un valor hexadecimal a enviar mediante el cable serie al Bioloid, indicando así el movimiento que debe hacer el robot.

```
        MSG->Comunicacion=false;
        MSG->Values=ObtenerQRData(); //Función obtención del código QR
        MSG->Comunicacion=true;
        usleep(3E6);
      }
```

La selección del valor hexadecimal puede observarse en este ejemplo donde el vector Values es el mensaje a enviar:

```
if(data.length()>0)
   {
     //cout << "Decoded Data : " << data <<endl;
     if(data.length()==9){ //9
        Values[2]=(char)0x04;
        Values[3]=(char)(~0x04);
        Values[4]=(char)0x09;
        Values[5]=(char)~(0x09);


        cout << "Enviando Izquierda "<<endl;

     }else if(data.length() == 7){ //Derecha
        Values[2]=(char)0x02;
        Values[3]=(char)~(0x02);
        Values[4]=(char)0x05;
        Values[5]=(char)~(0x05);


        cout << "Enviando Derecha "<<endl;
}
   Values[0]=(char)0xff;
   Values[1]=(char)0x55;
```

3.2. Control del Bioloid

Con el objetivo de que la Beaglebone pueda decirle al Robot que hay que hacer en todo momento, se ha necesitado un medio de comunicación y un protocolo. Para todo ello se ha recurrido al TFM escrito por Antoni Benavent Puertos [4], de modo que en este trabajo se incluyen las matrices de movimiento del Bioloid y las funciones de recepción y emisión de mensajes mediante el puerto serie del Robot. Estas funciones llevan integradas un protocolo de comunicación de modo que el Robot sepa cuando está recibiendo un mensaje.

El protocolo es el siguiente:

0xFF	0x55	Low Byte	~Low Byte	High Byte	~High Byte

Tal y como indica la tabla anterior, todo mensaje empieza con dos bytes (0xFF y 0x55) para dar a entender que los cuatro siguientes bytes representan la información. De estos cuatro, solo son útiles el primero y el tercero, de modo que el segundo y el cuarto son los mismos pero negados.

Esta negación de la información se hace de modo que el protocolo confirma que el mensaje se ha enviado correctamente. Por lo tanto, el Bioloid se hace las siguientes preguntas sobre los cuatro bytes de información:

- ¿El primer byte es el segundo pero negado?
- ¿El tercer byte es el cuarto pero negado?

Si las dos respuestas son correctas y la información viene precedida de un 0xff y un 0x55, el Bioloid entiende el mensaje y lo almacena, ordenando los bytes antes de almacenarlos definitivamente.

Este protocolo y modo de comunicación, obviamente necesita un medio físico en el que llevarse a cabo. Para este propósito se ha empleado el mismo cable que empleaba Antoni Benavent, es decir, el siguiente:

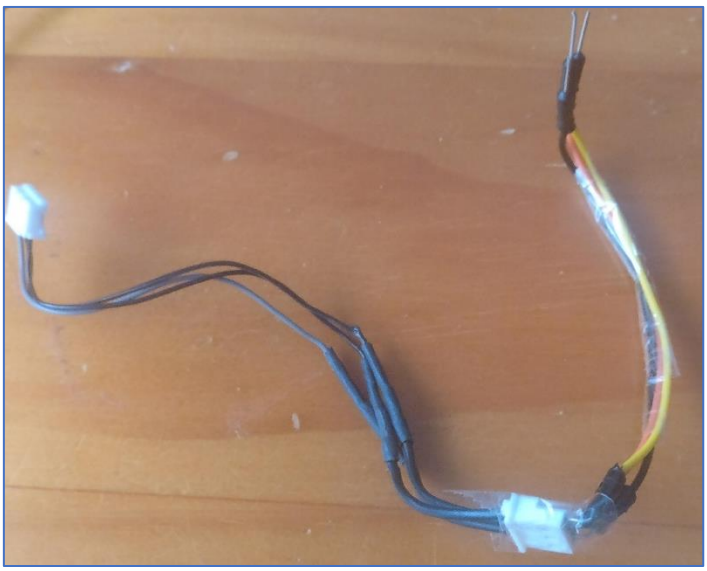

Ilustración 9: Cable de comunicación Bioloid

Tal y como se puede apreciar en la imagen, el cable, aunque tiene una terminación de cuatro pines que se conecta en el Bioloid, de estos cuatro se emplean tres, ya que el cuarto es tensión de alimentación (*Vcc*) que en este caso no es necesaria.

Al fin y al cabo, este utensilio es únicamente una combinación de dos cables unidos de modo que la otra terminación, en vez de conectarse directamente en la Beagle, en este caso terminará por ser unida a las borneras de la PCB.

Haciendo referencia al diseño de esta placa, mencionar que existe una bornera de dos pines destinados a recibir las señales Tx y Rx del Bioloid, por lo tanto, dos de los tres pines del segundo cabezal del cable irán a parar a esa bornera, mientras que el tercer pin del cable, que es la masa, va unido a uno de los terminales de las borneras de los servos. Concretamente al terminal dedicado al GND, ya que el circuito de los servos comparte la masa de la Beaglebone.

Estas conexiones pueden observarse en la siguiente imagen:

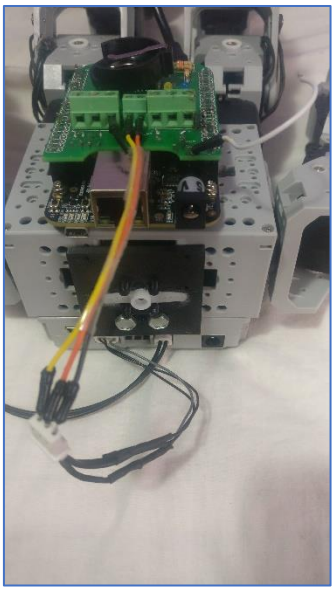

Ilustración 10: Conexión comunicación Bioloid.

Una vez explicada toda la información sobre el modo de comunicación, el protocolo y el medio físico, es necesario hablar del programa que lleva a cabo la comunicación en el Bioloid y su equivalente en la Beaglebone.

3.2.1. Código en la Beagle:

Como se ha explicado anteriormente, en los mensajes entre la Beagle y el Bioloid hay dos bytes de información. Esta es la base de la comunicación implementada por código, porque únicamente se envían y se espera recibir, mensajes de dos bytes. Un movimiento en línea recta para el Bioloid por ejemplo es el mensaje 0x0308, es decir, dos bytes en valor hexadecimal.

Además, como se ha expresado antes, existe una estructura determinada de mensaje, de modo que, la comprobación de que el mensaje recibido en la Beagle tenga la estructura correcta permite que se puedan obtener los valores del mensaje. Esta acción se ha implementado en la siguiente función:

```
bool Decodificar(char msg[128], char Low, char High){
  char ret[2];
  bool reconocido = false;
  for(volatile int i=0;i<128;i++){
    if(msg[i]== 0xff && msg[i+1]==0x55){
        ret[0]=msg[i+4];
        ret[1]=msg[i+2];
        reconocido=true;
```

```
            break;
        }
    }
    if(reconocido){
        if(ret[1]==Low && ret[0]==High)
            return true;
    }else{
        return false;
    }
}
```

Si se cumple con la estructura, dicha función retorna un verdadero y en caso contrario, un valor falso. Esta utilidad se emplea constantemente en el código de la Beaglebone, de modo que este sigue los siguientes dos pasos:

Primero se inicializa el canal UART4 a una velocidad de 57600 baudios por segundo y se espera la recepción de un mensaje enviado por el Bioloid con un valor equivalente a 0x80a0.

```
    string command_pin_RX = "/usr/bin/config-pin P9.11 uart";
    string command_pin_TX = "/usr/bin/config-pin P9.13 uart";
    system(command_pin_RX.c_str());
    system(command_pin_TX.c_str());
    BBB::UART  myUart(BBB::UART4,
                      BBB::Baud57600,
                      BBB::ParityNo,
                      BBB::StopOne,
                      BBB::Char8 );

    myUart.open( BBB::ReadWrite );
    myUart.flush( BBB::bothDirection );
    //cout << "ENVIANDO..." << endl << endl;
    while(!Decodificar(buff,(char)0xA0,(char)0x80)){
        memset(buff, 0, sizeof buff);
        myUart.write(Values, sizeof(Values));
        usleep(1000000);
        myUart.read(buff, sizeof(buff));
        cout<<buff<<endl;
    }
```

El segundo paso es un bucle infinito donde se espera a que la cámara haya detectado cualquier tipo de información (*MSG->Comunicación*) y envía la información detectada. Cada dirección tiene un valor hexadecimal diferente al igual que la nada.

```
    //Bucle enviar orden
      memset(buff, 0, sizeof buff);
      while(!MSG->Comunicacion){}
      myUart.write(Values, sizeof(Values));
      usleep(1000000);
      myUart.read(buff, sizeof(buff));
      cout<<"En bucle envio"<<endl;
```

Enviándose, por lo tanto, dicho valor hexadecimal al Bioloid.

3.2.2. Código en el Bioloid:

Se ha reutilizado código escrito por Antoni Benavent [4], sobre todo el destinado a movimientos del robot y únicamente se ha editado la función *main().*

Dentro de esta función se han implementado otras para detectar mensajes con el protocolo de la comunicación, además de recibir y enviar mensajes. Estas son:

- *zgb_rx_check():* Detecta si se ha recibido un mensaje que cumple con la estructura.
- *zgb_rx_data():* Recibe los datos en cola.
- *zgb_tx_data(int data):* Transmite paquetes de datos de dos bytes (*data*).

Estas funciones son utilizadas frecuentemente de modo que el código las emplea siguiendo este procedimiento:

Primero se configura el robot o, mejor dicho, la controladora del Bioloid llamada CM-530.

```
/* ---------------------- Configuration: ------------------------------------ */
  /* System Clocks Configuration */
  RCC_Configuration();
  /* NVIC configuration */
  NVIC_Configuration();
  /* GPIO configuration */
  GPIO_Configuration();
  SysTick_Configuration();
  /* Timer configuration */
  Timer_Configuration();
  /* ADC configuration */
  ADC_Configuration();
  GPIO_ResetBits(PORT_SIG_MOT1P,PIN_SIG_MOT1P);
  GPIO_ResetBits(PORT_SIG_MOT1M,PIN_SIG_MOT1M);
```

```
/* Puerto Zigbee inicializacion */
dxl_initialize( 0, 1 );
zgb_initialize(0); //Inicalización de la uart
EnableZigbee();
/* UART configuration */
USART1_Configuration(Baudrate_DXL);
USART_Configuration(USART_PC, Baudrate_PC);
```

El segundo paso es esperar a que se pulse el botón Start: *while (!ReadButton(START)){}*

Posteriormente, se sincroniza la comunicación entre la Beaglebone y el Bioloid esperando no recibir valores erróneos por parte de la Beagle (*por ejemplo, 0xabdc)* y enviando el valor 0x80a0 repetidamente para hacer que esta acceda al bucle de enviar las direcciones. dicho bucle es el incluido en el paso dos del procedimiento de la Beagle, anteriormente explicado.

```
while(DATA!=0xabdc){
    while(zgb_rx_check()!=1){
      zgb_tx_data(0x00b0);
    }
    RcvData= zgb_rx_data();
    LSB = dxl_get_lowbyte(RcvData);
    MSB = dxl_get_highbyte(RcvData);
    DATA  = (unsigned short)((MSB << 8) & 0xff00);
    DATA += LSB;
    mDelay(200);
  }
  zgb_tx_data(0x80a0);
  zgb_tx_data(0x80a0);
  zgb_tx_data(0x80a0);
  GPIO_ResetBits(PORT_LED_AUX, PIN_LED_AUX);
  mDelay(2000);
  GPIO_SetBits(PORT_LED_AUX, PIN_LED_AUX);
```

Y el último paso es un bucle infinito donde en función del valor recibido se realiza una acción u otra.

```
    switch(DATA){
      case 0x0502: //Giro derecha
        Robot_giro_derecha(0,5);
        GPIO_ResetBits(PORT_LED_PROGRAM, PIN_LED_PROGRAM);
        mDelay(500);
        GPIO_SetBits(PORT_LED_PROGRAM, PIN_LED_PROGRAM);
        mDelay(500);
```

```
   DATA=0;

   break;
case 0x0904: //Giro Izquierda


   GPIO_ResetBits(PORT_LED_PLAY, PIN_LED_PLAY);
   mDelay(500);
   GPIO_SetBits(PORT_LED_PLAY, PIN_LED_PLAY);
   mDelay(500);
   Robot_giro_izquierda(0,5);
   DATA=0;

   break;
```

El funcionamiento del código se representa gráficamente en el siguiente flujograma:

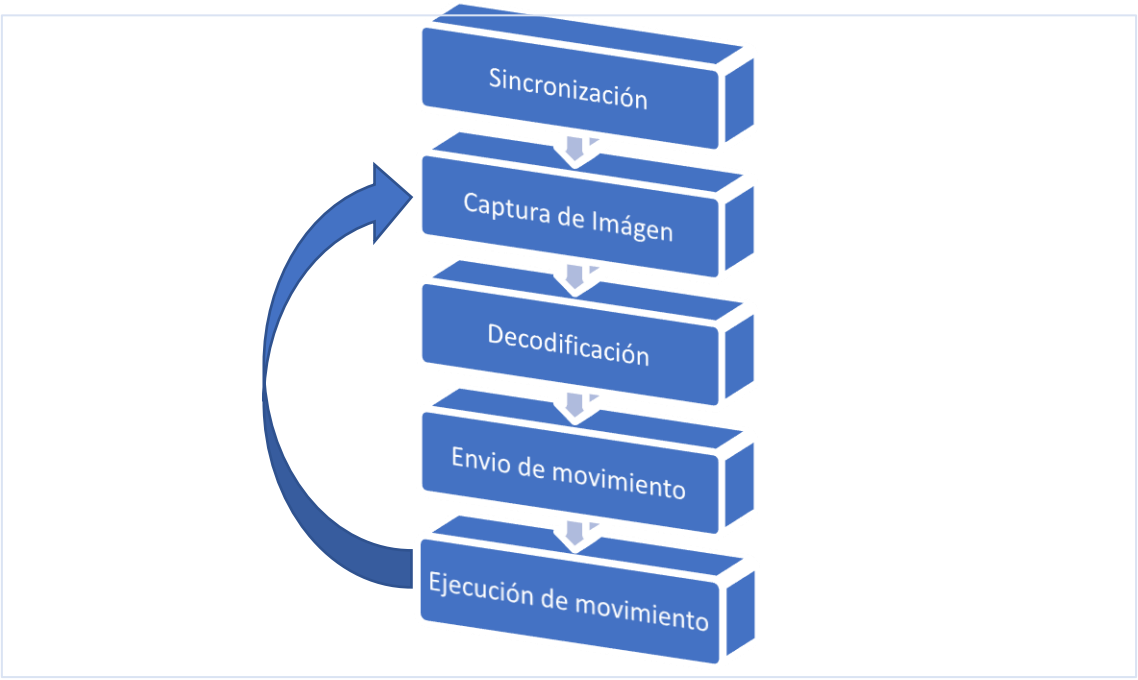

Ilustración 11: Flujograma de la comunicación en la Beaglebone

En último lugar es importante destacar la existencia en el código, de funciones para encender LEDs. Estos están integrados en la controladora CM-530 de modo que permiten indicar el estado del programa del robot.

Ilustración 12: Controladora CM-530 [2]

3.2.3. Entorno desarrollo del Bioloid

Para poder crear, editar y mejorar el código del Bioloid, se ha empleado el entorno de desarrollo Eclipse, utilizando un SDK para programar la controladora CM-530, la cual se encarga de ejercer el control sobre el robot.

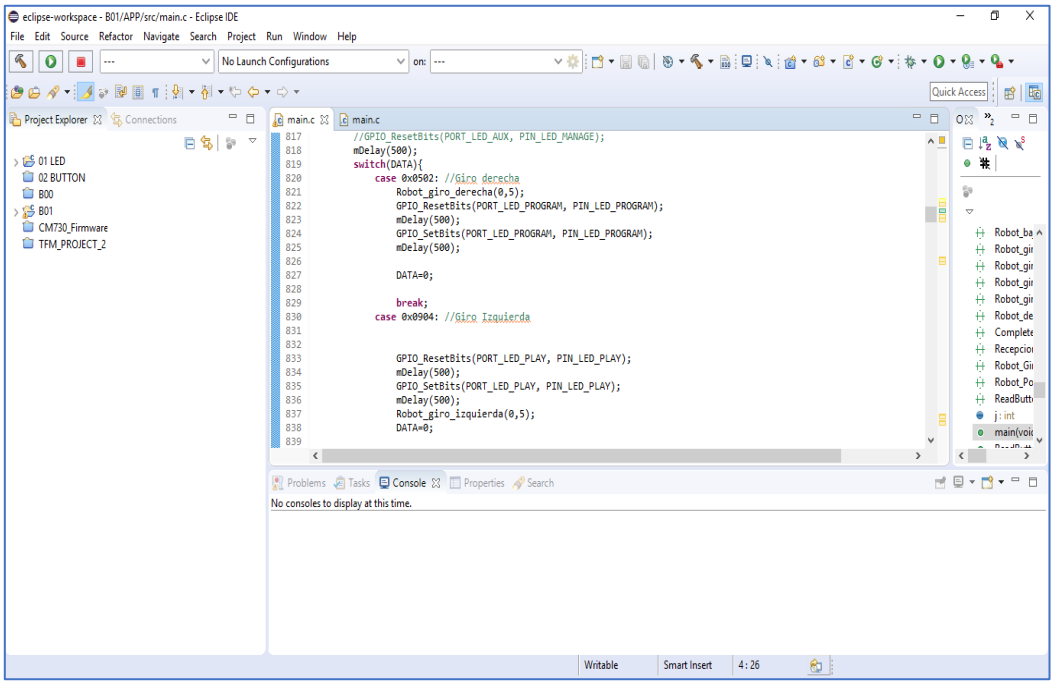

Ilustración 13: Entorno de desarrolo del Bioloid en Eclipse [5]

Una vez creado el código, este se compila creando un archivo *.hex* el cual, para poder probarlo, es necesario descargar el programa RoboPlus. Una vez descargado, instalado y ejecutado, entre las aplicaciones que este posee, se debe seleccionar la aplicación RoboPlus Terminal.

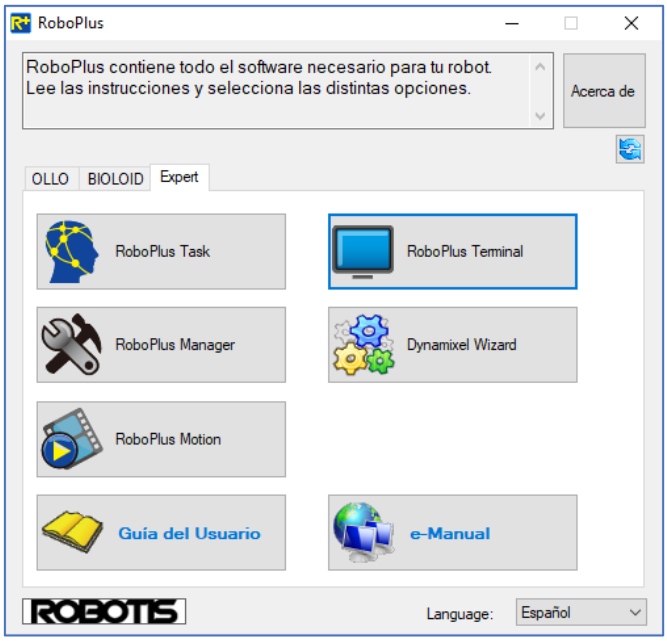

Ilustración 14: Interfaz del programa RoboPlus.

Se conecta mediante un cable, el puerto serie miniUSB a un puerto USB del PC que se esté utilizando y dentro del RoboPlus Terminal, se conecta la aplicación al puerto de conexión al seleccionar el COM y la velocidad de conexión (*57600 bps*). Tras ello se mantienen pulsadas las teclas para escribir una almohadilla mientras se activa el interruptor de alimentación del Bioloid. Haciendo todo esto, aparecen los siguientes mensajes por pantalla:

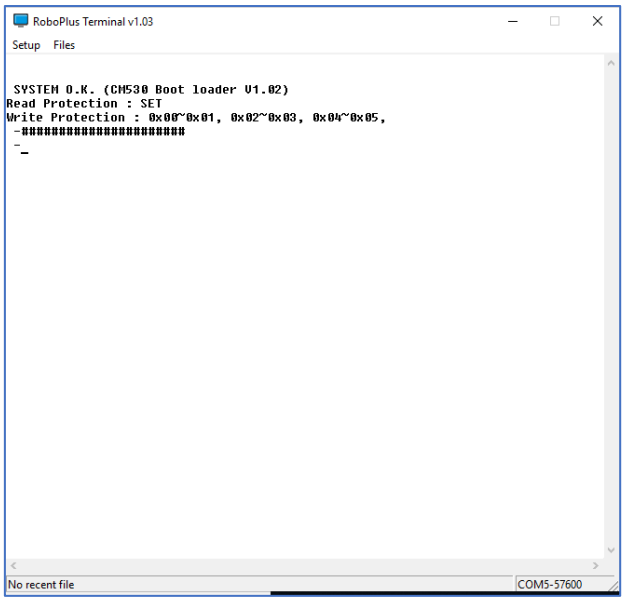

Ilustración 15: Interfaz programa RoboPlus Terminal

A continuación, se escribe la letra "l" de load, de modo que la aplicación se prepara para cargar el archivo *.hex* con el código creado. Para poder subirlo al controlador es necesario pulsar en *Files→Trasmit File*, donde aparecerá un navegador para seleccionar el archivo. A partir de aquí, se carga el *.hex* a la controladora, de modo que hay que esperar todo el proceso de carga para poder desconectar la controladora.

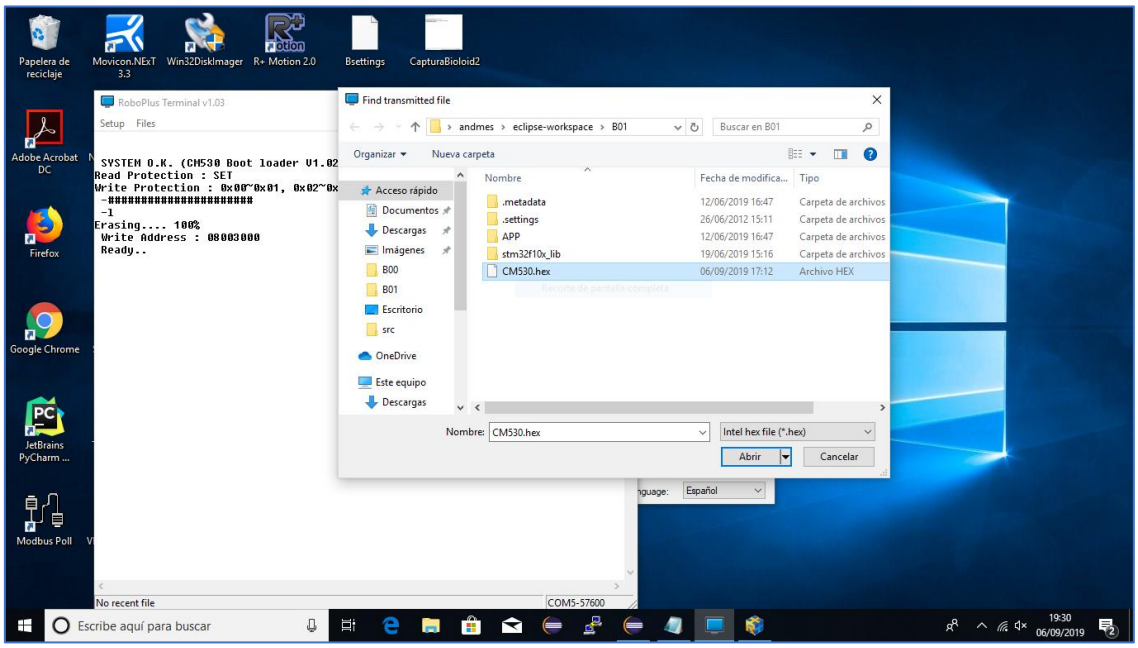

Ilustración 16: Subida del programa a la CM-530

Este es el procedimiento a realizar cada vez que se quiera probar un código nuevo para en el Bioloid y forma una parte importante del desarrollo del proyecto.

3.3. Diseño de la cape

El como se ha diseñado la PCB personalizada ha sido explicado en un documento adjunto en los anexos, pero es necesario expresar el empleo de la cape para poder entender el uso como una parte más del proyecto y poder concebir su desarrollo.

Haciendo referencia a las partes importantes de esta, la placa consta de una IMU con comunicación por I2C, unos servos alimentados por pilas y un módulo Bluetooth comunicado mediante UART. De modo que a continuación se habla del desarrollo de cada parte:

3.3.1. Unidad Inercial de Medida (IMU) MPU-9250

Este componente y parte fundamental de la placa se comunica a la Beaglebone mediante I2C, es decir, un protocolo de comunicación fácil de implementar debido a que se poseen librerías creadas previamente por el profesor José Simó. Del mismo modo, esta IMU es empleada en la asignatura de Sistemas Empotrados impartida en el MAII, por lo tanto, ya existe una librería que, de hecho, se utiliza en el código para poder obtener los valores de interés.

A continuación, se explica el código dedicado a la IMU y los pasos que se siguen:

En primer lugar, se instancian el objeto i2c en el canal 2 (ya que se utiliza en el canal I2C_2) y el objeto IMU, el cual se inicializa.

```
BBB::I2C i2cBus(2);
BBB::MPU9250Lite imu(&i2cBus);
imu.initialize();
```

El segundo paso es utilizar el objeto *imu* el cual se introduce en diferentes funciones como por ejemplo la de calibrar el magnetómetro y la de guardar estos valores de calibración en un archivo .txt. Además, se crean las variables que se van a emplear:

```
calibrateAccelOffsets(&imu);
imu.saveOffsetRegisters("accOffsets.txt");
float accx,accy,accz, moduleAcc;
```

```
float girx, giry, girz, moduleGir;
float mx,my,mz, moduleMag;
```

A partir de aquí, con la IMU calibrada, es posible medir valores reales de forma cíclica. Para hacerlo posible se miden los 9 valores principales y se calculan los módulos mediante la raíz de los cuadrados.

```
imu.get9Axis(&accx, &accy, &accz, &girx, &giry, &girz, &mx, &my, &mz);
moduleAcc = sqrt(accx*accx+accy*accy+accz*accz);
moduleMag = sqrt(mx*mx+my*my+mz*mz);
moduleGir = sqrt(girx*girx+giry*giry+girz*girz);
```

Estos valores se emplean en la parte de Bluetooth para ejecutar su exportación al exterior.

3.3.2. Módulo Bluetooth HC-05

Este dispositivo, que es el que se encarga de transmitir la información de la IMU, necesita una determinada configuración, de modo que se ha creado una pequeña clase con funciones, para poder emplear este componente de forma correcta.

Al momento de encender la Beaglebone, es necesario conectar el pin PIO_11 a 3.3 V. Para hacerlo, se ha soldado un cable a un terminal circular para poder conectarlo manualmente a Vcc. De modo que se conecta a Vcc, se enciende la Beagle y se suelta el cable.

Todo este procedimiento se lleva a cabo para hacer que el dispositivo HC-05 se comunique mediante la UART2 a una velocidad de 38400 baudios por segundo. Una vez alcanzada esta velocidad de comunicación, es posible configurar el módulo siguiendo estos pasos:

1)Se envía un valor cualquiera para estabilizar la comunicación y poder desechar información incorrecta: *ble.Enviar(FUZZY);*

2)Posteriormente se pregunta el estado de comunicación (debe aparecer un ok al observar la salida por pantalla): *ble.Enviar(GET_COM_STATUS);*

3)El tercer paso consta en poner un nombre al dispositivo. El nombre seleccionado es "BLE". *ble.Enviar(SET_NAME);*

4)Es necesario hacer que el dispositivo asuma el rol de esclavo, de modo que aporte valores al maestro. *ble.Enviar(SET_ROLE_SLAVE);*

5) Y como último paso, se cambia la velocidad de comunicación a 9600 baudios por segundo (*ble.Enviar(BAUD_RATE);*) y se reinicia el módulo (*ble.Enviar(RESET);).*

Finalizada la configuración, el módulo se reinicia y ya es posible transmitir información, de modo que se envían valores constantemente mediante la UART2, los cuales son recibidos en el PC al emparejarse. El mensaje que se envía desde la Beaglebone empieza con un "@" y separa los campos obtenidos de la IMU mediante ";". La función empleada para transmitir es: *ble.Transmitir();*.

Todas estas funciones se basan en enviar unas cadenas de caracteres mediante UART, por lo tanto, para recibir y enviar valores se utilizan estas funciones dentro de la librería del HC_05:

```
myUart2->open( BBB::ReadWrite);
myUart2->flush( BBB::bothDirection );
myUart2->write(writeBuffer,
sizeof(writeBuffer));
myUart->read(buff, sizeof(buff));
myUart2->close();
```

Las funciones escritas anteriormente permiten, respectivamente, abrir el canal de comunicación tanto para lectura como escritura, vaciar el canal, escribir una cadena de caracteres "*writeBuffer*" que tiene un tamaño equivalente a *"sizeof(writeBuffer)".* La cuarta función permite leer una cadena *"buff"* de tamaño *"sizeof(buff)"* y la última función cierra el canal de comunicación.

3.3.3. Control Servos.

Para poder controlar los servos alimentados mediante dos pilas de botón en serie, se emplean dos señales PWM. Estas son programadas mediante una librería utilizada en la asignatura de Sistemas Empotrados del máster, de modo

que únicamente se necesitan saber los valores del periodo de las señales y el ciclo de trabajo que es el valor que se varía para poder mover los servos.

Según la hoja de datos, la señal que se envía debe tener un periodo de 20 ms por lo tanto la cuestión es: ¿Cuál debe ser el rango del ciclo de trabajo de cada servo?

La respuesta a esta pregunta se ha obtenido de forma experimental, de modo que se han creado varios bucles con un valor inicial del ciclo de trabajo, de modo que el incremento en cada iteración del bucle varía el ciclo de trabajo, logrando, por lo tanto, el movimiento deseado.

Ejemplo de bucle:

```
float duty = 8.0; //Valor central
pwm2.run();
for (int i=0; i<20; i++) { //Cambiar el 20 per a fer que vaja més avant o mes arrere
                    //i tindre-ho en conter al fer el moviment contrari
    duty -=0.1;
    pwm2.setDutyCycle((float)duty);

    usleep(200000);

}
```

3.4. Aplicación de escritorio

Dado que es necesario llevar a cabo una recaptación de los datos que se envían mediante tecnología Bluetooth, se ha creado un programa para Windows capaz de recibir esta información y representarla gráficamente además de almacenarla en un archivo *.txt*. Posibilitando así el análisis de los diferentes campos que mide la IMU y la observación directa de sus valores.

Para poder crear dicha aplicación se ha empleado Visual Studio. Concretamente, se ha utilizado el apartado de Windows Forms para poder crear un programa con interfaz gráfica. Además, ha sido necesario emplear el paquete libre conocido como *.32Feet*.

Este paquete permite crear programas capaces de interactuar con tecnología bluetooth tanto para publicar como para recibir valores. Su integración ha sido sencilla gracias a los ejemplos libres que se publican en la red.

El proyecto de Visual Studio creado para hacer esta aplicación funciona asíncronamente mediante hilos, de modo que se evita la congelación de la aplicación y lleva incorporada una interfaz gráfica para facilitar la interacción y legibilidad de los datos, haciendo posible la observación del comportamiento de los diferentes valores de estudio de forma periódica.

3.4.1. Interfaz Gráfica

La aplicación se ha desarrollado de modo que únicamente existe una ventana a la que accede el usuario. Esto se debe a que se trata de un programa experimental y dedicado, por lo tanto, únicamente a este trabajo.

Esta ventana puede observarse en la siguiente imagen:

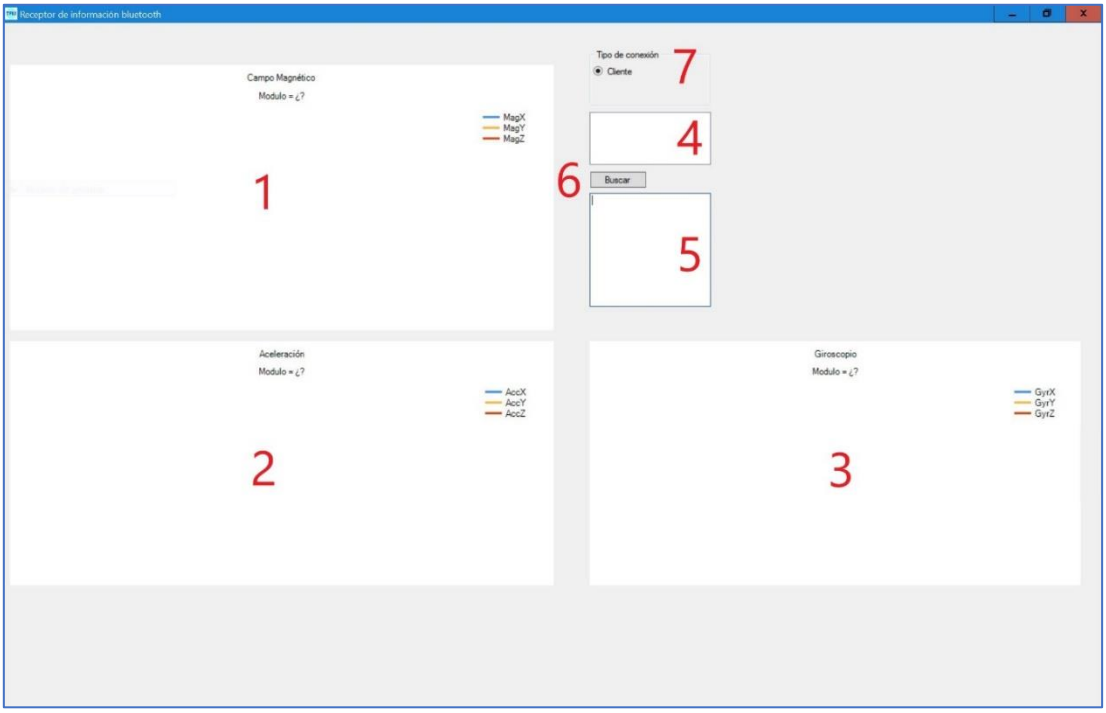

Ilustración 17: Interfaz aplicación PC para Bluetooth.

Se han enumerado en rojo los diferentes elementos que componen la ventana, los cuales son:

1. **Gráfica de campo magnético**: Se representa el valor de cada eje del campo magnético en uT.
2. **Gráfica de aceleración**: Representación de cada eje en unidades g.

3. **Gráfica de giroscopio**: Del mismo modo que en las anteriores. En este caso las unidades son grados por segundo.
4. **Lista de Dispositivo**: Se trata de una caja de texto donde aparecen los diferentes dispositivos bluetooth enlazables, de modo que solo interesa el que lleve como nombre BLE, dado que se trata de la Beaglebone.
5. **Registro de Progreso**: Es otro cuadro de texto para mostrar que está haciendo la aplicación en todo momento y enseñar los mensajes exactos que se reciben.
6. **Botón de Buscar**: Se trata de un pulsador necesario para actualizar la Lista de Dispositivos. Además, permite preguntar al usuario donde ubicar el archivo .txt almacena todos los valores medidos.
7. **Selección de Modo**: Indica el modo de trabajo de la aplicación. Originalmente existía el modo cliente y el modo servidor, pero para este caso solo se ha empleado el modo cliente.

3.4.2. Código: Partes Esenciales

Dado que el código es extenso, se van a explicar las partes más cruciales de él.

La primera parte remarcable es la creación de variables globales de los campos y la configuración de las gráficas para definir las series de valores, la rotulación de los ejes, los títulos y el tipo de representación gráfica en modo línea con un ancho determinado.

```
        delegate void SetChartCallback();

        float MagX, MagY, MagZ, Mod_Mag;


        float AccX, AccY, AccZ, Mod_Acc;

        float GyrX, GyrY, GyrZ, Mod_Gyr;


        string Time, direccion;
        int it = 0;

        List<string> items;
        public Form1()
        {
            items = new List<string>();
```

```
            InitializeComponent();

            chart1.Titles.Add("Campo Magnético");
            chart1.Titles.Add("Modulo = ¿?");


            chart1.ChartAreas[0].AxisX.Title = "Tiempo";
            chart1.ChartAreas[0].AxisY.Title = "Magnetismo (uT)";

            chart1.Series.Add("MagX");
            chart1.Series["MagX"].BorderWidth = 6;
            chart1.Series["MagX"].XValueType =
System.Windows.Forms.DataVisualization.Charting.ChartValueType.String;

            chart1.Series.Add("MagY");
            chart1.Series["MagY"].BorderWidth = 6;
            chart1.Series["MagY"].XValueType =
System.Windows.Forms.DataVisualization.Charting.ChartValueType.String;

            chart1.Series.Add("MagZ");
            chart1.Series["MagZ"].BorderWidth = 6;
            chart1.Series["MagZ"].XValueType =
System.Windows.Forms.DataVisualization.Charting.ChartValueType.String;

            chart1.Series["MagX"].ChartType =
System.Windows.Forms.DataVisualization.Charting.SeriesChartType.Line;
            chart1.Series["MagY"].ChartType =
System.Windows.Forms.DataVisualization.Charting.SeriesChartType.Line;
            chart1.Series["MagZ"].ChartType =
System.Windows.Forms.DataVisualization.Charting.SeriesChartType.Line;
```

La segunda parte más remarcable es la actualización del Registro de Progreso al crear una función llamada updateUI, definida del siguiente modo:

```
        private void updateUI(string message)
        {
            Func<int> del = delegate()
            {
                tbOutput.AppendText(message + System.Environment.NewLine);
                return 0;
            };
            Invoke(del);
        }
```

Además de la función que actualiza la Lista de Dispositivos:

```
        private void scan()
        {

            updateUI("Starting Scan..");
            BluetoothClient client = new BluetoothClient();
            devices = client.DiscoverDevicesInRange();
            updateUI("Scan complete");
            updateUI(devices.Length.ToString()+" devices discovered");
            foreach (BluetoothDeviceInfo d in devices)
            {
                items.Add(d.DeviceName);
            }
```

```
            updateDeviceList();
        }
```

Y la correcta definición de la identificación del dispositivo Bluetooth que existe en el PC.

```
Guid mUUID = BluetoothService.SerialPort;
```

A partir de aquí, solo falta recibir la información y dividirla para poder asignar cada valor a su variable. El mensaje que se recibe desde la Beaglebone empieza con un “Inicio;” y termina con un “Fin;”, de este modo, se puede distinguir cuando comienza y cuando termina el mensaje. Además, cada campo se ha subdividido mediante “;”, de modo que se divide el mensaje según ese carácter obtenido 12 campos en total. La asignación se hace del siguiente modo:

```
String msg = Encoding.ASCII.GetString(received);
if (msg.Contains("Inicio") && msg.Contains("Fin")) {
    //if (it == 1000)
    //{
    //    TrasladarVectores();
    //
    //}
    MagX = float.Parse(msg.Split(';')[1]);
    MagY = float.Parse(msg.Split(';')[2]);
    MagZ = float.Parse(msg.Split(';')[3]);
    Mod_Mag = float.Parse(msg.Split(';')[4]);
    AccX = float.Parse(msg.Split(';')[5]);
    AccY = float.Parse(msg.Split(';')[6]);
    AccZ = float.Parse(msg.Split(';')[7]);
    Mod_Acc= float.Parse(msg.Split(';')[8]);
    GyrX = float.Parse(msg.Split(';')[9]);
    GyrY = float.Parse(msg.Split(';')[10]);
    GyrZ = float.Parse(msg.Split(';')[11]);
    Mod_Gyr = float.Parse(msg.Split(';')[12]);
    Time = DateTime.Now.ToString("HH:mm:ss");
```

El último paso, una vez obtenidos los valores, es enviarlos a la gráfica para que sean representados. Durante el programa hay una variable de valor entero que cuenta el número de veces que se han ido representando valores, de modo que a partir de la vez número 50, se elimina el primer valor de la serie y se desplazan todos los valores de esta una posición, representando el nuevo valor en la posición 49.

Este proceso se hace para que la gráfica no se sature, evitando un aumento en la memoria empleada y para que los cambios en las variables se puedan observar más fácilmente y no se vean como pequeños cambios lejanos.

El código que hace esto para una de las gráficas es el siguiente:

```
                chart1.Series["MagX"].Points.AddXY(Time, MagX);
                chart1.Series["MagY"].Points.AddXY(Time, MagY);
                chart1.Series["MagZ"].Points.AddXY(Time, MagZ);
                chart1.Titles[1].Text = "Modulo = " + Mod_Mag.ToString() +"
uT";
                if (it >= 50) {
                    chart1.Series["MagX"].Points.RemoveAt(0);
                    chart1.Series["MagY"].Points.RemoveAt(0);
                    chart1.Series["MagZ"].Points.RemoveAt(0);
                }
```

3.5. Código Principal

Se ha tenido que juntar todas las partes del proyecto en un mismo código, de modo que este consiste en un programa con 3 hilos principales, que son:

- **BLE_imu**: Hilo empleado para llevar a cabo la lectura de la IMU y la transmisión de los datos mediante el módulo Bluetooth.
- **Bioloid**: Se utiliza para poder establecer la comunicación entre la Beaglebone y el robot.
- **Camara**: Permite tomar imágenes y decodificarlas a la vez de controlar los servomotores del soporte.

Para poder trabajar conjuntamente el hilo destinado al Bioloid y el de la cámara, se han utilizado variables globales y compartidas. Permitiendo así que se actualicen los mensajes a enviar al Robot sin interrumpir las tomas de imágenes.

Todo este código emplea las librerías que se comentan en la programación de la PCB y la cámara. De modo que se trabaja con todas ellas en el IDE de Eclipse de C++ en un PC con Linux y luego se crea un archivo *.cpp* con el mismo código en la Beaglebone, de modo que este se compile en este Sistema Empotrado.

Para poder hacerlo, se utiliza el siguiente comando en consola, donde *P15.cpp* es el código con todo el proyecto:

g++ -Wall -I/usr/include -I/home/debian -L /usr/ MPU9250Lite.cpp util.cpp i2c.cpp PWMuniv.cpp HC_05.cpp UART.cpp P15.cpp -o P15 -lueye_api -lpthread `pkg-config --cflags --libs opencv`

4. Pruebas Realizadas

Desde el pasado mes de diciembre de 2018 se ha estado trabajando con la Beaglebone Black, de modo que primero se instalase una imagen con herramientas de tiempo real, permitiendo así la adquisición de experiencia al trabajar con un Sistema Empotrado que emplease Debian.

Se siguieron los pasos explicados en el anejo de Instalación de la Beaglebone para poder instalar los drivers de la cámara y poder experimentar con la toma de imágenes. Para ello, también fue necesaria la instalación de la librería OpenCV.

Con esta herramienta incorporada al microcontrolador, se comenzaron a hacer códigos o archivos *.cpp* en la Beaglebone, descubriendo como compilarlos. Cada prueba principal tiene asociada un número diferente: P0, P1, P2…

Hasta que, al fin, se pudieron obtener imágenes nítidas mediante OpenCV y el driver de la cámara.

Ilustración 18: Fotografía obtenida con la cámara

Una vez hecho esto, el siguiente paso era poder pensar en una aplicación de Visión que permitiera obtener una dirección o orden a enviar al Bioloid.

Una aplicación simple consistía en códigos QR, los cuales contenían las direcciones u órdenes a enviar al robot. Para esta aplicación era necesario tener instalada la versión 4.0.0 como mínimo, de OpenCV, para poder utilizar el módulo QRDecoder que simplifica sustancialmente esta parte del trabajo.

Para poder llevar a cabo el reconocimiento de imágenes, o de forma más concreta, la lectura de códigos QR, se emplea la librería de Visión Artificial conocida como OpenCV. A partir de la versión 4 de dicha herramienta, se incluyó un módulo nuevo llamado como QRDecoder, el cual, permite usar diversas funciones, por ejemplo, en lenguaje C++, para poder leer códigos QR de forma sencilla.

El procedimiento para poder leer los códigos QR resulta ser muy sencillo gracias a este nuevo módulo. Los pasos de este son los siguientes:

- Cargar una imagen cualquiera y pasarla a una variable *Mat*.

cv::Mat inputImage = cv::imread("recto.png", 1); //Carga de la imagen guardada

- Se inicializa una variable **QRCodeDetector**.

QRCodeDetector qrDecoder = QRCodeDetector();

- Se crean las variables *Mat* para la caja en la que está dentro el código QR.

Mat bbox, rectifiedImage, bwsrc;

- Posteriormente, es necesario pasar la imagen original a escala de grises para hacer la identificación más amena.

cvtColor(inputImage, bwsrc, cv::COLOR_RGB2GRAY); //Conversión a grises

- Una vez preparada la imagen, el último paso es emplear la herramienta decodificadora de QRs, la cual, se emplea en una sola línea de código. Las variables que se introducen son la imagen en escala de grises, el rectángulo dentro del cual, está el supuesto código QR y la imagen rectificada con el QR detectado. Dicha función o herramienta lo que hace es primero detectar si hay un código QR en la imagen obtenida y en caso positivo, decodificarlo para devolver una cadena de caracteres con el mensaje del código. Por lo tanto, la variable data contiene el mensaje.

  std::string data = qrDecoder.detectAndDecode(bwsrc, bbox, rectifiedImage);

Con todo ello, ya era posible decodificar los QR, así que a partir de aquí empezó a trabajarse con el Robot y en su personalización, instalando primero el SDK de Robotis para poder programar la controladora CM-530 y poder así, programar el robot para al fin, hacer pruebas de comunicación con la Beagle.

Para hacer esto posible, se hicieron pruebas de comunicación del Bioloid a un PC y al ver que se recibía lo que se esperaba, se pasó a la fase de crear programas en la Beagle y compilarlos. Por lo tanto, fue necesario recurrir a el protocolo de comunicación entre los dispositivos y establecer un orden entre el Bioloid y la Beagle, hasta que, por ejemplo, al leer la cámara un código que significase "*Derecha*", el robot fuese hacia la derecha.

Asumida esta parte, se diseñó la placa PCB. Tardó una semana en diseñarse y fabricarse, de modo que, al llegar, se soldaron los componentes y se hizo un testeo de conectividad de los componentes para evitar cortocircuitos.

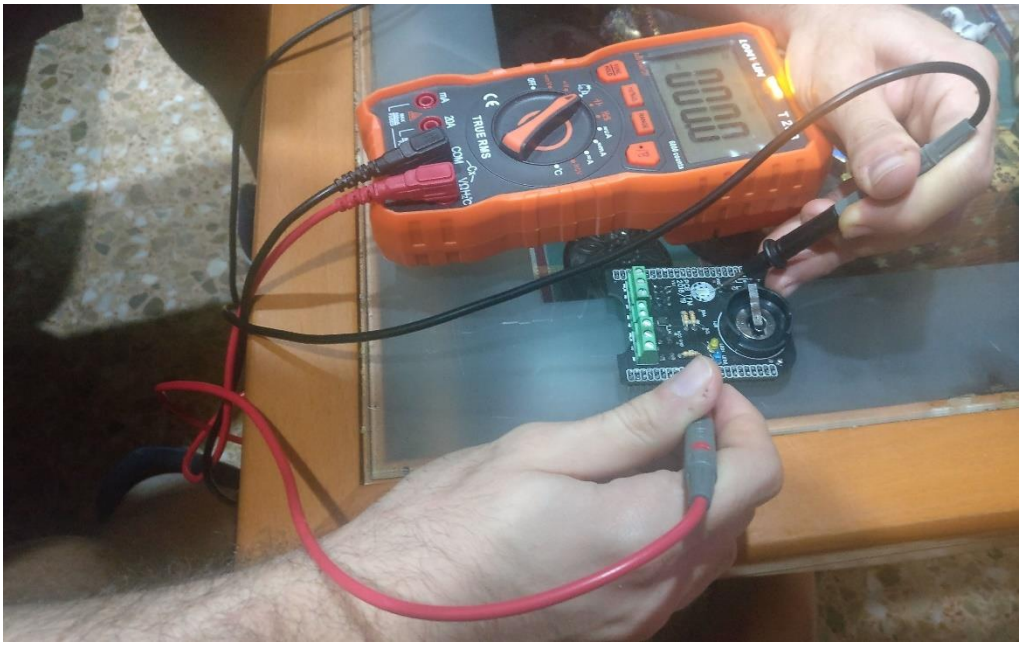

Ilustración 19: Comprobación de la PCB

Con la placa montada, se procedió a crear un programa en la Beagle, de la misma forma que en las otras pruebas, para que se leyesen los valores de la IMU y se enviaran mediante Bluetooth. Con las pruebas fue posible descubrir fallos en la aplicación de PC hasta que, al fin, los datos recibidos empezaban a ser coherentes, observándose del siguiente modo:

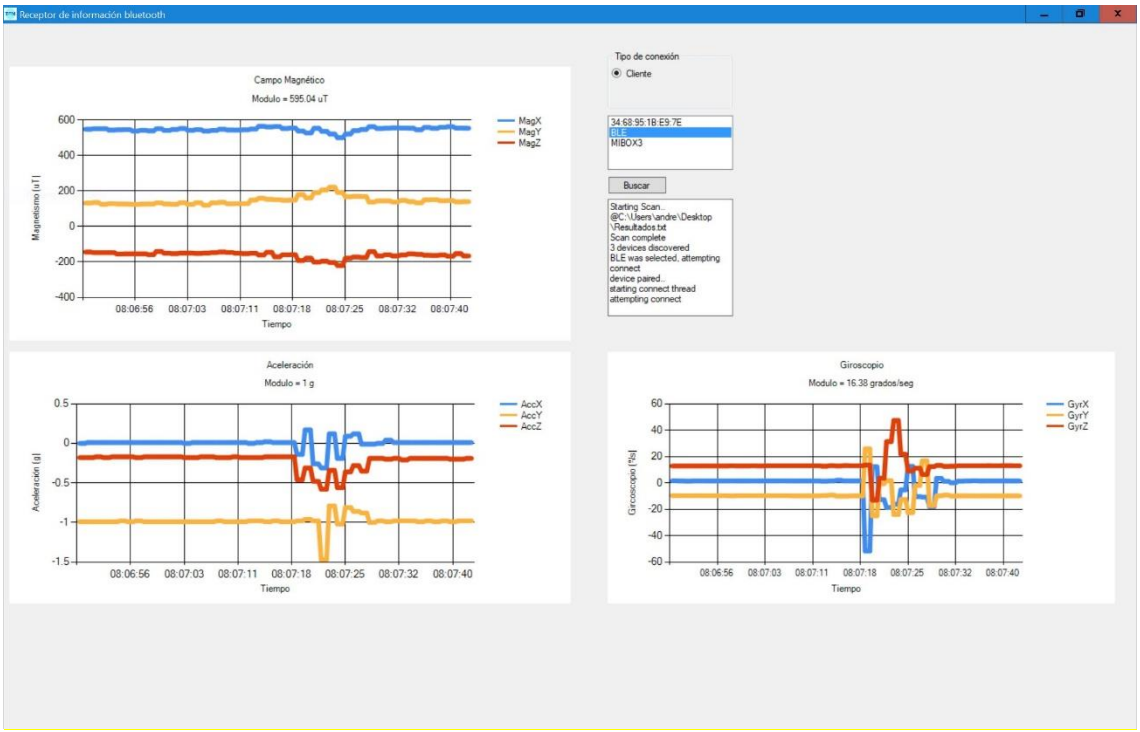

Ilustración 20: Medidas durante la prueba final.

En paralelo al diseño de la placa, se procedió al diseño del soporte de la cámara. Han sido un total de 4 las versiones diferentes del modelo, siguiendo siempre la estructura, pero ajustando el tamaño para que los servomotores pudieran acoplarse de la forma mas ajustada posible. Cada modelo tuvo que ser impreso en una Impresora 3D.

Al final, en cuanto a pruebas, lo más difícil del soporte ha sido establecer el movimiento de los servos, puesto que es diferente al sujetarlo en una zona fija a llevarlo sujeto en el cabezal del Robot. Esto puede deberse a causas de rozamiento, haciendo que el comportamiento en una situación sea diferente respecto a la otra.

Esto se ha solucionado al variar los tiempos de rotación del soporte, ante todo en el plano horizontal (*ilustración 5*).

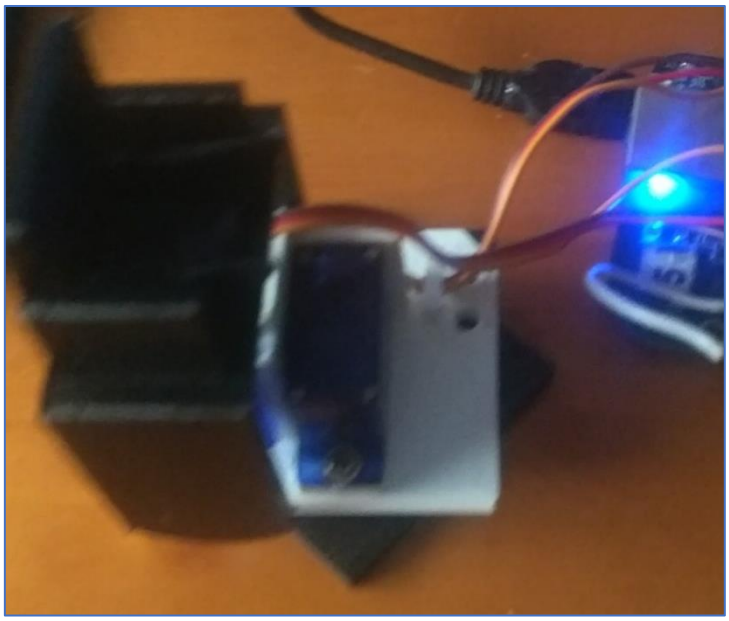

Ilustración 21: Soporte rotado

Con todas estas pruebas, se pasó a hacer una prueba conjunta, con todas las partes conectadas en el robot, ofreciendo los siguientes problemas:

- La fricción del suelo ofrece una gran inestabilidad al Robot dificultando su movimiento.
- La anterior inestabilidad provoca que el soporte de la cámara del robot pueda separarse de la base.
- El movimiento inestable del Robot hace que la IMU detecte valores lejanos a la media en reposo.

Estos problemas han provocado que sea imposible trabajar con el robot para hacer un circuito completo. Pero ha sido posible ejercer varios movimientos al leer códigos QR (*uno de ellos en una posición equivalente a 45 grados positivos en el plano horizontal*). La configuración del robot durante la prueba se observa en la siguiente imagen:

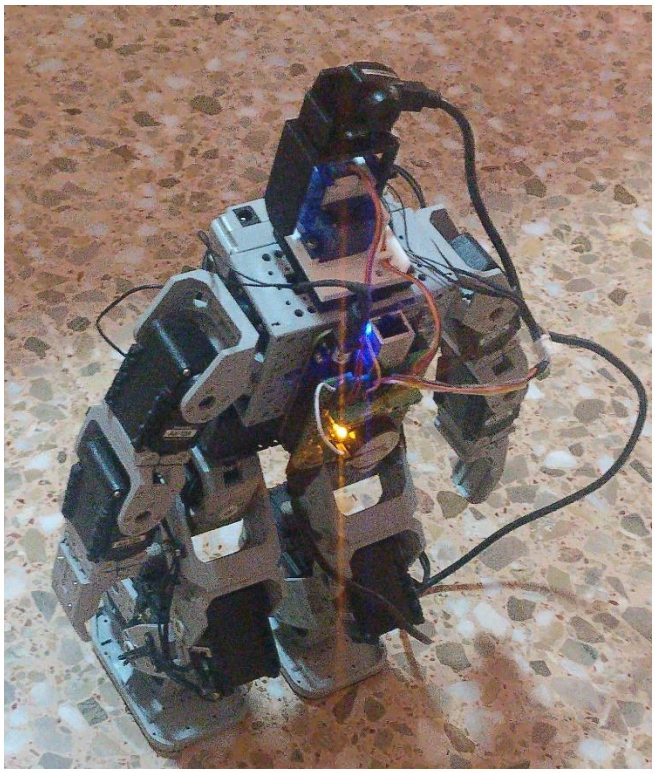

Ilustración 22: Robot Bioloid en pruebas.

5. Manual de Usuario de la aplicación de escritorio

El programa experimental "*Receptor de Información Bluetooth*" creado mediante Visual Studio se emplea siguiendo estos pasos:

- Iniciar la aplicación, de modo que aparece la siguiente ventana.

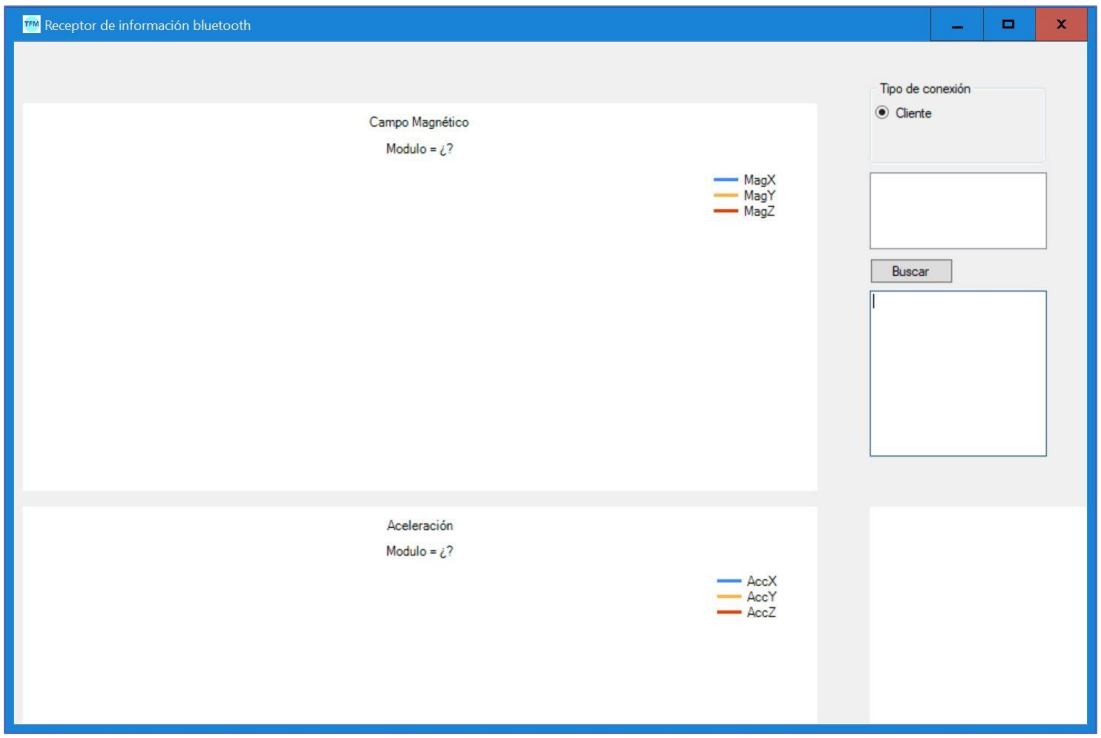

Ilustración 23: Inicio aplicación.

Tal y como se observa, existen tres gráficas donde posteriormente, se graficarán las señales recibidas. Para poder pasar al siguiente paso, es necesario pulsar el botón Buscar y seleccionar un directorio donde se guardará un registro de los valores que se obtengan.

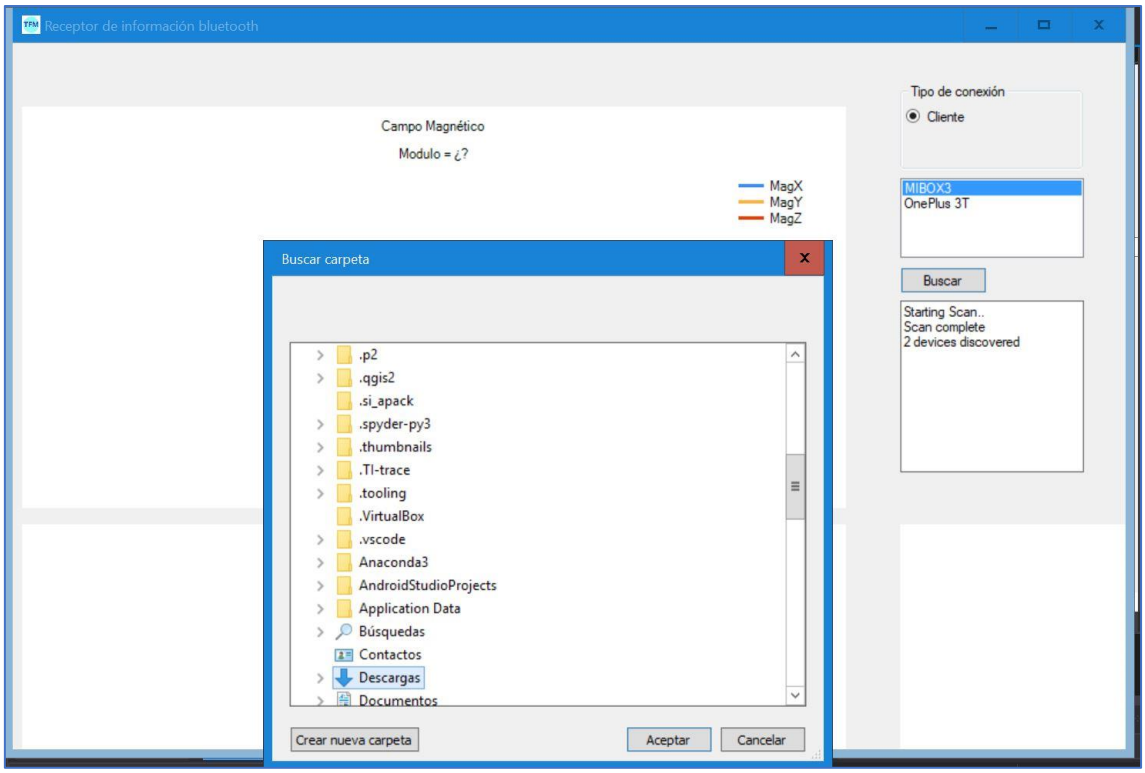

Ilustración 24: Ubicación de archivo de registro Log

- En este segundo paso, se ha iniciado la búsqueda de dispositivos enlazables, de modo que el progreso de la búsqueda se explica en el cuadro de texto situado debajo del botón buscar.

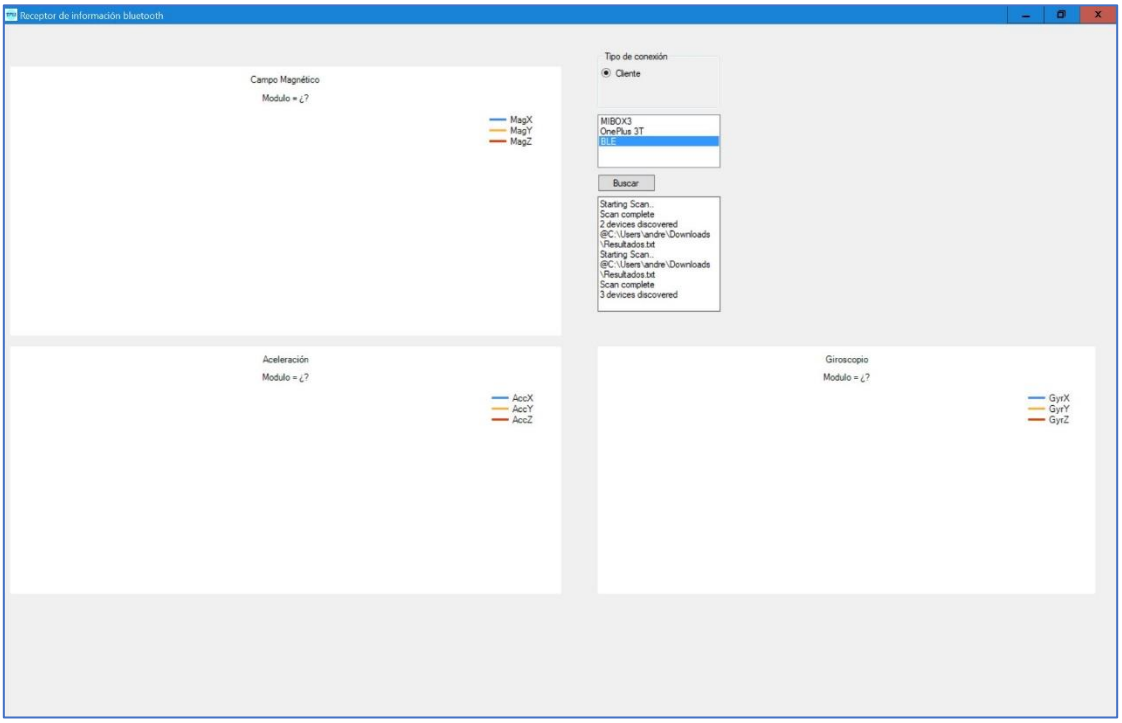

Ilustración 25: Emparejamiento con módulo Bluetooth.

Una vez finalizada la búsqueda, se muestran los dispositivos en el otro cuadro de texto. Para poder pasar al siguiente paso es necesario hacer doble click sobre el dispositivo llamado BLE.

- Tras el paso anterior, se ha iniciado el emparejamiento del dispositivo BLE con la aplicación de escritorio. Una vez el proceso haya concluido satisfactoriamente, las gráficas empiezan a mostrar valores en función del tiempo.

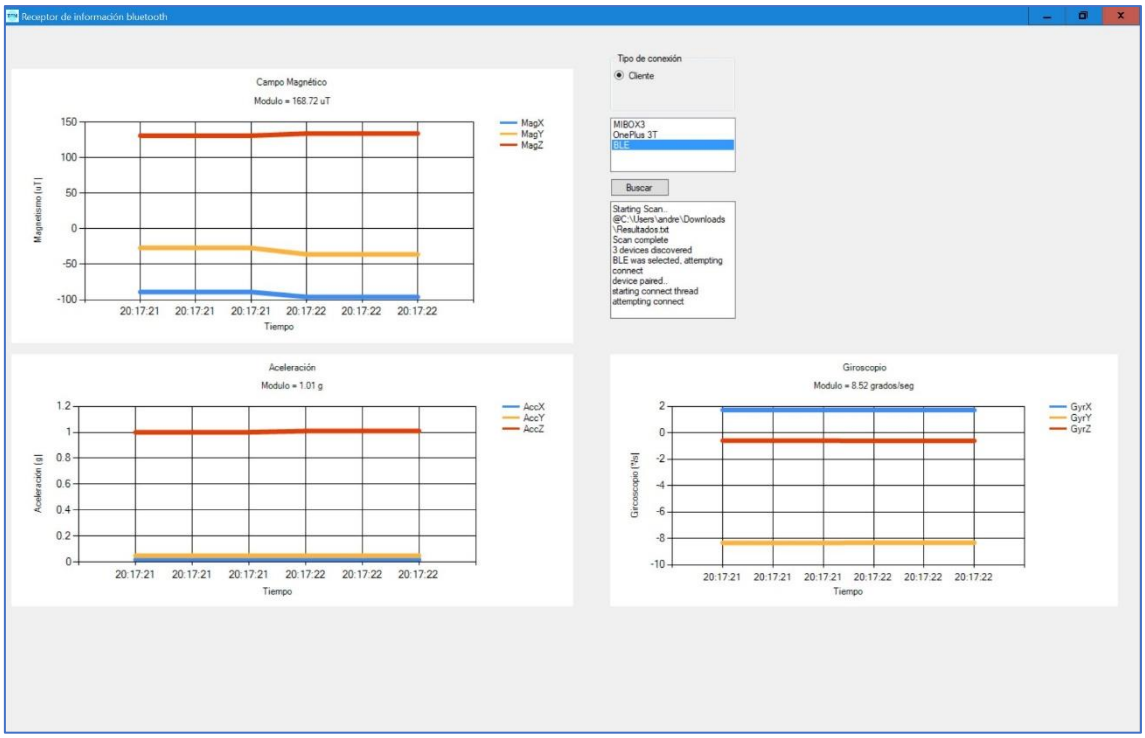

Ilustración 26: Representación de datos obtenidos.

- Una vez haya transcurrido suficiente tiempo se dispone de un histograma con muchos valores, por lo tanto, para finalizar la representación hay que cerrar la aplicación en el Bioloid.

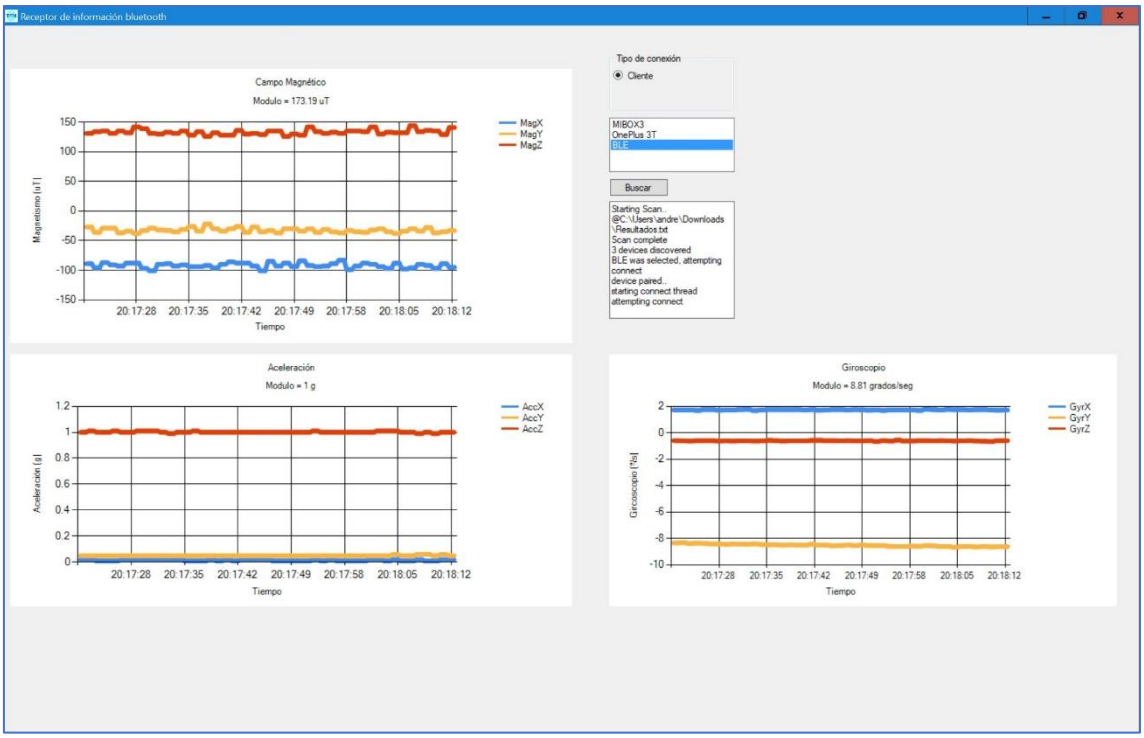

Ilustración 27: Variación de las medidas.

En caso de querer tener valores a mano, la aplicación crea un archivo *.txt* donde se escriben todos los valores.

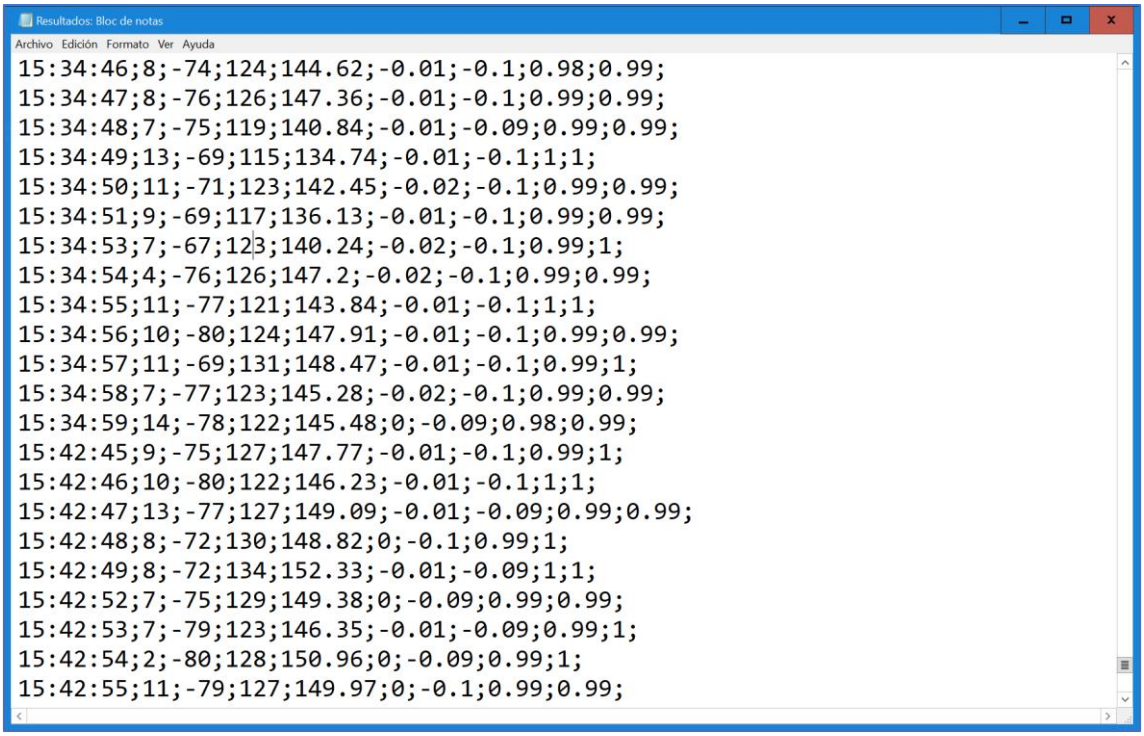

Ilustración 28: Ejemplo archivo de Registro.

6. Presupuesto para la cape

Artículo	Descripción	Valor Unitario (€/Ud)	Cantidad (Ud)	Subtotal (€)
COMPONENTES IMPORTADOS				
Cabecera de 3 Pines	Cabecera 0.100" (2.54mm) Plugin RoHS	0.0091	31	**0.28**
Resistencias de 330 Ohmios	Resistencias de peliculas de carbono 330Ohms ±5% 1/4W Axial RoHS	0.0046	2	**0.01**
Resistencias de 10 KOhmios	Resistencias de peliculas de carbono 10KOhms ±5% 1/4W Axial RoHS	0.0046	6	**0.03**
MPU-9250	IMU QFN-24 RoHS	9.78	1	**9.78**
LED Amarillo	LED amarillo 589nm 200mcd@20mA Through Hole RoHS	0.03	1	**0.03**
LED Azul	LED azul 460~470nm 500~10000mcd@20mA Through Hole RoHS	0.041	1	**0.04**
Condensadores 100nF	Condensador Cerámico 100nF -20%,+80% 50V Through Hole,P=5.08mm RoHS	0.015	4	**0.06**
Portapilas	Portapilas, Through Hole RoHS	1.52	1	**1.52**
Bornera de 3 terminales	Pluggable System Terminal Block P=5.08mm RoHS	0.47	2	**0.94**
Bornera de 2 terminales	Pluggable System Terminal Block P=3.81mm RoHS	0.8	1	**0.80**
Gastos de Transporte		22.57	1	**22.57**
			SUBTOTAL CON IMPUESTOS YA INCLUIDOS	**36.06**
COMPONENTES ADQUIRIDOS EN ESPAÑA				
HC05 Bluetooth	Permite conexiones inalámbricas bluetooth como maestro o esclavo	4.2	1	**4.20**
Pila CR2430	Pila de boton de litio 3V	2.06	1	**2.06**
			SUBTOTAL SIN IMPUESTOS	**6.26**
IMPUESTO	IVA		21%	**1.31**
			SUBTOTAL CON IMPUESTOS	**7.57**
TOTAL COMPONENTES				**43.63**
PLACA PCB				
Manufacturación	Fabricación de un pedido de 5 placas	1.82	1	**1.82**
Gastos de Transporte		5.86	1	**5.86**
TOTAL PLACA (IMPUESTOS YA INCLUIDOS)				**7.68**
				TOTAL (€)
				51.31

7. Conclusiones

Tras explicar las principales partes del proyecto cabe señalar si se han cumplido las metas prestablecidas. La más importante de ellas, es decir, el diseño y desarrollo de un sistema de percepción para un robot Bioloid, se ha cumplido mediante los códigos expuestos en el trabajo y los resultados descritos en el apartado Pruebas Realizadas, aunque con diversos problemas externos al robot principalmente.

A partir de este, se desglosan los subobjetivos, empezando por el diseño y desarrollo de un sistema de adquisición de imágenes usando la cámara *UI-1007XS-C*, conseguido tal y como muestra la ilustración 18, obteniendo una imagen nítida obtenida a partir de dicha cámara.

El siguiente objetivo nombrado como: Diseño de electrónica específica (*cape*) para la Beaglebone Black que soporte una IMU, se ha logrado mediante las herramientas descritas en el apartado 9.2, obteniendo la PCB que se observa en la ilustración 48. Dicha PCB, aunque tiene problemas de diseño en su versión utilizada (*v 1.0*) logra este subobjetivo.

Posteriormente señalar el objetivo: Planteamiento y despliegue de sistema de comunicación entre el robot y aplicaciones externas. Dicho objetivo se ha conseguido tal y como lo indican la imagen 20.

El penúltimo objetivo: Proyección de un soporte para la cámara que permita cambios en su configuración física, se ha realizado adecuadamente tal y como indica la imagen 34, aunque se han sufrido inconvenientes en el diseño haciendo necesaria una modificación manual del objeto tras la impresión 3D. A pesar de que el soporte funciona, es poco robusto, estando demasiado expuesto a soltarse del robot debido al movimiento de este.

Y en último lugar, se ha buscado obtener validación y realización de pruebas. Dicho objetivo se ha cumplido tal y como indica la ilustración 19 y el apartado de Pruebas Realizadas, consiguiendo los resultados para demostrar el cumplimiento de todos los objetivos.

8. Bibliografía

[1] C. Balaguer, «Robots Humanoides: La Evolución,» de *ROBOT'2007: Workshop Español de Robótica*, Zaragoza, 2007.

[2] Robotis, «http://www.robotis.us/robotis-premium/,» [En línea].

[3] iDS, «https://en.ids-imaging.com/xs-features.html,» [En línea].

[4] A. B. Puertos, *Diseño y desarrollo de un módulo de gestión de IMU para Bioloid sobre Beaglebone Black,* Valencia: UPV, 2016.

[5] «Eclipse,» [En línea]. Available: https://www.eclipse.org/downloads/packages/.

[6] «SolidWorks,» [En línea]. Available: https://www.solidworks.com/es.

[7] «EasyEDA,» [En línea]. Available: https://easyeda.com.

[8] «LCSC,» [En línea]. Available: https://lcsc.com.

[9] «iDS Driver,» [En línea]. Available: https://es.ids-imaging.com/download-ueye-emb-hardfloat.html.

[10] «Solaris,» [En línea]. Available: https://solarianprogrammer.com/2018/12/18/cross-compile-opencv-raspberry-pi-raspbian/.