ANEXOS

En esta sección se incluyen documentos que pueden ser de interés. En concreto un presupuesto, los planos de hadware y el código más importante de Matlab y Arduino.

1. Presupuesto.

Se ha creado una tabla orientativa de presupuesto de componentes del hardware. Al tratarse de un prototipo no tiene sentido hablar más que de los precios de componentes individuales y no de fabricación.

La posibilidad de fabricar no esta siendo planteada por la empresa a corto o medio plazo. En el mejor de los casos se llegaría a esa cuestión al finalizar el desarrollo completo y testado de todo el sistema.

Elemento	**Proveedor**	**Precio unitario**	**Precio total**
Placa Arduino Duemilanove	Cooking-hacks (Libelium)	22€	22€
LSM303DLH	Digi-key	10.5€	10.5€
Cond. Electrolítico 10uF	Farnell	1.48€	1.48€
Cond. Cerámico 0.1uF	Farnell	0.023€	0.023€
Cond. Cerámico 0.22uF	Farnell	0.031€	0.031€

Cond. Cerámico 4.7uF	Farnell	0.14€	0.14€
Cond. Cerámico 2.2uF	Farnell	0.052€	0.104€
Cond. Cerámico 470pF	Farnell	0.004€	0.008€
Resistencia 10k	Farnell	0.023€	0.046€
Regulador LDO 1.8v	Digi-key	0.1942€	0.1942€
Regulador LDO 3.3v	Digi-key	0.23€	0.23€
Jumper de conexión	Farnell	0.22€	0.22€
Regleta	Farnell	0.32€	0.64€
TOTAL			35.62€

2. Código.

En esta sección se va a adjuntar el código fundamental creado en el proyecto, no así el auxiliar que se ha utilizado para pruebas, scripts o pasos intermedios.

El código se estructura de la misma forma en como se explica en la memoria. En ella se encuentra información detallada de cada una de las partes y funciones.

2.1 Librería de lectura de datos mediante protocolo I2C entre Arduino y sensores.

```
#include <LSM303DLH.h>
#include <Wire.h>
#include <math.h>

// Defines /////////////////////////////////////////////////////////////


//A continuacion todas las direcciones quedan definidas con una etiqueta mas intuitiva.
//El mapa de registros esta en el datasheet pag.27

#define ACC_ADDRESS            (0x30 >> 1)  //WRITE    si se pone en la 0x32 debera poder leer un segundo acelerometro
#define MAG_ADDRESS            (0x3C >> 1)  //WRITE

//Registros del acelerometro (ver datasheet pag 28)
#define CTRL_REG1_A            (0x20) //configuracion general
#define CTRL_REG2_A            (0x21) //filtros paso alto
#define CTRL_REG3_A            (0x22) //interrupciones
#define CTRL_REG4_A                   (0x23) //lectura de datos (little indians) y auto-test
#define CTRL_REG5_A            (0x24) //control del sleep-to-wake
#define HP_FILTER_RESET_A      (0x25)
#define REFERENCE_A            (0x26)
```

```
#define STATUS_REG_A        (0x27) // Dice si ha habido overrun o si han datos nuevos disponibles
//Los OUT de adelerometro y magnetometro tienen parte High y Low, por lo tanto 2 Bytes de precisi—n por dato.
#define OUT_X_L_A           (0x28)      //Todos expresados en complemento a 2
#define OUT_X_H_A           (0x29)
#define OUT_Y_L_A           (0x2A)
#define OUT_Y_H_A           (0x2B)
#define OUT_Z_L_A           (0x2C)
#define OUT_Z_H_A           (0x2D)
//gestion de interrupciones
#define INT1_CFG_A          (0x30)
#define INT1_SOURCE_A       (0x31)
#define INT1_THS_A          (0x32)
#define INT1_DURATION_A     (0x33)
#define INT2_CFG_A          (0x34)
#define INT2_SOURCE_A       (0x35)
#define INT2_THS_A          (0x36)
#define INT2_DURATION_A     (0x37)

//Registros del magnetometro (ver datasheet pag 38)
#define CRA_REG_M           (0x00) //Configuracion del registro A
#define CRB_REG_M           (0x01) //Configuracion del registro B
#define MR_REG_M            (0x02) //Registro de modo de operacion

#define OUT_X_H_M           (0x03)
#define OUT_X_L_M           (0x04)
#define OUT_Y_H_M           (0x05)
#define OUT_Y_L_M           (0x06)
#define OUT_Z_H_M           (0x07)
#define OUT_Z_L_M           (0x08)

#define SR_REG_M            (0x09) //Registro de estado (bloqueo de salida, bit listo...)
#define IRA_REG_M           (0x0A) //"identification registers" para identificar el dispositivo
#define IRB_REG_M           (0x0B)
#define IRC_REG_M           (0x0C)

// Constructor ////////////////////////////////////////////////////////////////
```

```
LSM303DLH::LSM303DLH()
{

}

// Metodos publicos //////////////////////////////////////////////////////////

// Activa acelerometro y magnetometro, y los pone en modo normal.

void LSM303DLH::enable(void)
{
  //Activar acelerometro
  Wire.beginTransmission(ACC_ADDRESS);  //Abre la transmision con el esclavo de la direccion dada
  Wire.send(CTRL_REG1_A); //Envia datos desde un esclavo en respuesta a una peticion de un maestro o PREPARA
  //                      para recibir datos.
  //0x27 = 0b00100111
  // Modo de alimentación normar. todos los ejes (x,y,z) activados. Velocidad seleccionada 50 Hz
  Wire.send(0x27);
  Wire.endTransmission();

  //Activar Magnetometro
  Wire.beginTransmission(MAG_ADDRESS);
  Wire.send(MR_REG_M);
  //0x00 = 0b00000000
  // Modo conversión continua
  Wire.send(0x00);
  Wire.endTransmission();
}

// Lee los 6 canales del LSM303DLH y guarda los valores en variables del objeto
void LSM303DLH::read()
{
  //leer acelerometro
  Wire.beginTransmission(ACC_ADDRESS);

  Wire.send(OUT_X_L_A | (1 << 7)); //pone las salidas de acelerometro listas para enviar
  Wire.endTransmission();
  Wire.requestFrom(ACC_ADDRESS,6); //peticion de 6 bytes a acelerometro
```

```
  while (Wire.available() < 6); //coge los 6 bytes y los guarda en 6 registros

  uint8_t xla = Wire.receive();
  uint8_t xha = Wire.receive();
  uint8_t yla = Wire.receive();
  uint8_t yha = Wire.receive();
  uint8_t zla = Wire.receive();
  uint8_t zha = Wire.receive();

  ax = (xha << 8 | xla) >> 4; //los agrupa en tres registros del vector a (porque eran little-indians)
  ay = (yha << 8 | yla) >> 4;
  az = (zha << 8 | zla) >> 4;

  //leer magnetometro
  Wire.beginTransmission(MAG_ADDRESS);
  Wire.send(OUT_X_H_M);
  Wire.endTransmission();
  Wire.requestFrom(MAG_ADDRESS,6);

  while (Wire.available() < 6);

  uint8_t xhm = Wire.receive();
  uint8_t xlm = Wire.receive();
  uint8_t yhm = Wire.receive();
  uint8_t ylm = Wire.receive();
  uint8_t zhm = Wire.receive();
  uint8_t zlm = Wire.receive();

  mx = (xhm << 8 | xlm);
  my = (yhm << 8 | ylm);
  mz = (zhm << 8 | zlm);
}
```

2.2 Sketch de la placa Arduino

```
//INCLUDES:
#include <Wire.h>
#include <LSM303DLH.h>

LSM303DLH compass;
```

```
int info=0;
int tipoPromedio=0;
int numMedidas=100;
int arrayAx[100];
int arrayAy[100];
int arrayAz[100];
int arrayMx[100];
int arrayMy[100];
int arrayMz[100];
   long time1=0;
   long time2=0;
   long time=0;

   int j = 0; //orden anterior, j+1 es orden actual


void setup() {
  Serial.begin(115200);
  Wire.begin();
  compass.enable();
}

void loop() {
  if (Serial.available() >0) {

     delay(1); //Para esperar que llegue todo, por seguridad
                //se podría hasta quitar
        info=ToInteger()*10; //Lee el primer byte del puerto serie y lo convierte en
decenas
       info=info+ToInteger();  //Le el siguiente byte y lo suma (unidades)
      switch (info) {

       case 40:

        printValoresSensor(); //Función para devolver los valores a matlab

        break;

       case 41:
       //setup de medidas 41X(nºdigitos)XX(nºmedidas dos cifras)
       setupMedidas();
```

```
          break;

         case 42:
          //setup tipo de promediado 42X (0=medida bruta, 1= media simple, etc)
          tipoPromedio=ToInteger();

          break;
        }//end switch

       Serial.flush(); //Limpio lo que pueda haber en  el Puerto serie
       //Serial.println("Puerto Libre"); **Si se quieren mandar un comando al final **
                                  // **para decir que arduino ya está libre aqui sería**
      } //end if principal


      else {
      //LECTURA DE DATOS y HACER MEDIA
          // time1 = millis();

          compass.read(); //Me remito a la clase en LSM303DLH.h y .ccp.  Lee I2C y lo
guarda.
        arrayAx[j] = compass.ax;
        arrayAy[j] = compass.ay;
        arrayAz[j] = compass.az;
        arrayMx[j] = compass.mx;
        arrayMy[j] = compass.my;
        arrayMz[j] = compass.mz;
        delay(1); //Este delay diezma el numero de datos recogido
        j++;
        if (j==numMedidas){
          j=0;
        }

       }// end else (bucle continuo)

     } //end loop

     //lee byte del puerto serie y devuelve el entero a
      int ToInteger(){
        int a=0;
        if (Serial.available() > 0) {
```

```
        a=Serial.read();
        a=a-48; //convierte ascii en decimal
        return a;
      }
    }

    void printValoresSensor(){
      mediaSimple();

          Serial.print("40"); //Devuelve el mismo valor de protocolo para saber que devuelve valores de sensores
          Serial.print(" ");
          Serial.print(compass.ax); //devuelve el resto de variables
          Serial.print(" ");
          Serial.print(compass.ay);
          Serial.print(" ");
          Serial.print(compass.az);
          Serial.print(" ");
          Serial.print(compass.mx);
          Serial.print(" ");
          Serial.print(compass.my);
          Serial.print(" ");
          Serial.print(compass.mz);
          Serial.println(" "); //-->tiempo total de prints 2ms



          j = 0;

    }

    void setupMedidas(){
      int numDigitos = 0;
      int i = 0;
      int j = 0;
       numDigitos=ToInteger(); //lee el byte que da el numero de digitos que va a tener "numMedidas"

      //obtiene el valor de numMedidas
      int aux=0; //variable auxiliar para los loops
      int multiplicador=1; //variable auxiliar para multiplicar
```

```
  for (i=numDigitos; i>0; i--){
    multiplicador=1;
    aux=ToInteger();
    for (j=i-1; j>0;j--){
      multiplicador=multiplicador*10;
    }
    numMedidas=numMedidas + aux*multiplicador;
  }
}//end setupMedidas



void mediaSimple(){

   int i=0;
   int aux1=0;
   int aux2=0;
   int aux3=0;
   int aux4=0;
   int aux5=0;
   int aux6=0;

   for (i=0; i < j; i++){
     aux1 += arrayAx[i];
     aux2 += arrayAy[i];
     aux3 += arrayAz[i];
     aux4 += arrayMx[i];
     aux5 += arrayMy[i];
     aux6 += arrayMz[i];
   }//end for
   compass.ax = aux1/(j+1);
   compass.ay = aux2/(j+1);
   compass.az = aux3/(j+1);
   compass.mx = aux4/(j+1);
   compass.my = aux5/(j+1);
   compass.mz = aux6/(j+1);

}//end mediaSimple
```

2.3 Librería para control de placas Arduino (arduino.m)

```
classdef arduino < handle
```

```
    % This class defines an "arduino" object
    % Giampiero Campa, Aug 2010, Copyright 2009 The MathWorks, Inc.

    % Modificación por Javier Matínez, Álvaro Arrúe y Francisco Irache
    % MODIFICACIONES
    % Se han eliminado las funciones relacionadas con la asignación de
    % pines en tiempo real, ya que estas vendrán dadas por el sketch
    % Se han eliminado todas las funciones referentes a steppers y motores
    % Se han añadido nuevas funciones


    %NOTA: La parte comentada en inglés es de la clase original. Contiene
    %la conexión a puerto serie, constructor, destructor, funciones de
    %seguridad, avisos de error, etc.
    properties (SetAccess=private,GetAccess=private)
        aser   % Serial Connection
        pins   % Pin Status Vector
    end

    properties (Hidden=true)
        chks = false;  % Checks serial connection before every operation
        chkp = true;   % Checks parameters before every operation
        ArduinoFree = true; %Checks that Arduino is free to receive new orders
    end

    methods

        % constructor, connects to the board and creates an arduino object
        function a=arduino(comPort)

            % check nargin
            if nargin<1,
                comPort='DEMO';
                disp('Note: a DEMO connection will be created');
                disp('Use a the com port, e.g. ''COM5'' as input argument to
connect to the real board');
            end

            % check port
            if ~ischar(comPort),
                error('The input argument must be a string, e.g. ''COM8'' ');
            end

            % check if we are already connected
            if isa(a.aser,'serial') && isvalid(a.aser) &&
strcmpi(get(a.aser,'Status'),'open'),
                disp(['It looks like Arduino is already connected to port '
comPort ]);
                disp('Delete the object to force disconnection');
                disp('before attempting a connection to a different port.');
                return;
            end

            % check whether serial port is currently used by MATLAB
            if ~isempty(instrfind({'Port'},{comPort})),
                disp(['The port ' comPort ' is already used by MATLAB']);
                disp(['If you are sure that Arduino is connected to ' comPort]);
                disp('then delete the object, execute:');
                disp(['  delete(instrfind({''Port''},{''' comPort '''}))']);
                disp('to delete the port, disconnect the cable, reconnect it,');
```

```
                disp('and then create a new arduino object');
                error(['Port ' comPort ' already used by MATLAB']);
            end

            % define serial object
            a.aser=serial(comPort,'BaudRate',115200);

            % connection
            if strcmpi(get(a.aser,'Port'),'DEMO'),
                % handle demo mode

                fprintf(1,'Demo mode connection ..');
                for i=1:4,
                    fprintf(1,'.');
                    pause(1);
                end
                fprintf(1,'\n');
                pause(1);

                % chk is equal to 3, (general server running)
                chk=3; % Legacy

            else
                % actual connection

                % open port
                try
                    fopen(a.aser);
                catch ME,
                    disp(ME.message)
                    delete(a);
                    error(['Could not open port: ' comPort]);
                end

                % it takes several seconds before any operation could be
attempted

                fprintf(1,'Attempting connection ..');
                chk=1; % INDICO QUE ESTÁ ABIERTO! PERO SIN COMPROBAR

            end

            a.aser.Tag='ok';
            disp('Arduino successfully connected !');

        end % arduino

        % distructor, deletes the object
        function delete(a)

            % if it is a serial, valid and open then close it
            if isa(a.aser,'serial') && isvalid(a.aser) &&
strcmpi(get(a.aser,'Status'),'open'),
                if ~isempty(a.aser.Tag),
                end
                fclose(a.aser);
            end

            % if it's an object delete it
            if isobject(a.aser),
                delete(a.aser);
```

```
            end


        end % delete

%         % disp, displays the object
        function disp(a) % display
            if isvalid(a),
                if isa(a.aser,'serial') && isvalid(a.aser),
                    disp(['<a href="matlab:help arduino">arduino</a> object
connected to ' a.aser.port ' port']);
                    disp(' ');
                else
                    disp('<a href="matlab:help arduino">arduino</a> object
connected to an invalid serial port');
                    disp('Please delete the arduino object');
                    disp(' ');
                end
            else
                    disp('Invalid <a href="matlab:help arduino">arduino</a>
object');
                    disp('Please clear the object and instantiate another one');
                    disp(' ');
            end
        end

% ////////////FUNCIÓN PARA MOVER MOTORES////////////////
% ////////////////////////////////////////////////////

function moverMotor(a,mot_ID, dir, usteps, delay)
    % mot_ID: nº de motor de 1 a 4;
    % dir: Dirección 1 o 2
    % usteps: nº de micropasos de 8 a 9999 (ampliable)
    % delay: delay entre upasos de 200 a 9999 (ampliable)

    % Saco vector con los digitos del nº de upasos
    % El tamaño del vector es el número de digitos
    usteps_vect=int2digits(usteps);
    usteps_num=length(usteps_vect);

    % Saco vector con los digitos del nº de delay
    % El tamaño del vector es el número de digitos
    delay_vect=int2digits(delay);
    delay_num=length(delay_vect);

    %Preparando string
    p0=num2str(0);
    p1=num2str(1);
    pmot=num2str(mot_ID);
    pdir=num2str(dir);
    pdigstep=num2str(usteps_num);
    pstep=num2str(usteps);
    pdigdelay=num2str(delay_num);
    pdelay=num2str(delay);

    % STRING A ENVIAR
    P=[p0 p1 pmot pdir pdigstep pstep pdigdelay pdelay p0];

    % Preparados para enviar
    fwrite(a.aser,P,'uchar');
```

```
    % Leer por puerto serie la respuesta del inf
    % [mot_1,mot_2,mot_3,mot_4,tiempo,ArduinoFree]=SerialRead(a);
end
%--------------------------------------------------------------------
%--------------------------------------------------------------------

% ////////////FUNCIÓN PARA LEER INFO DEL ARDUINO/////////////////
% ////////////////////////////////////////////////////////////////

function [mot_1,mot_2,mot_3,mot_4,tiempo,ArduinoFree]=SerialRead(a)

    p=fscanf(a.aser,'%c');
    disp(['Lectura: ' p]);
    op=[p(1) p(2)];

    switch op,
        case '01'
            disp('01');
            command=op;
        case '11'
            disp('11');
            command=op;
        otherwise
            disp('Comando no reconocido')
    end

    % Me salto los dos espacios
    % Posición de Motor 1
    count=5;
    c=p(count);
    val=[c];

    while ~isspace(c)
        count=count+1;
        c=p(count);
        val=[val c];
    end

    mot_1=str2num(val);

    % Posición de Motor 2
    count=count+1;
    c=p(count);
    val=[c];

    while ~isspace(c)
        count=count+1;
        c=p(count);
        val=[val c];
    end
    mot_2=str2num(val);

    % Posición de Motor 3
    count=count+1;
    c=p(count);
    val=[c];
    while ~isspace(c)
        count=count+1;
        c=p(count);
        val=[val c];
    end
```

```
    mot_3=str2num(val);

    % Posición de Motor 4
    count=count+1;
    c=p(count);
    val=[c];
    while ~isspace(c)
        count=count+1;
        c=p(count);
        val=[val c];
    end
    mot_4=str2num(val);

    % Tiempo de trabajo del motor
    count=count+1;
    c=p(count);
    val=[c];
    while ~isspace(c)
        count=count+1;
        c=p(count);
        val=[val c];
    end
    tiempo=str2num(val);

    % Buscamos si está libre el puerto
    count=count+5;
    c=p(count);
    val=[c];
    while ~isspace(c)
        count=count+1;
        c=p(count);
        val=[val c];
    end
    count=count+1;
    c=p(count);
    val=[val c];
    while ~isspace(c)
        count=count+1;
        c=p(count);
        val=[val c];
    end
    val=val(1:length(val)-1);
    ArduinoFree=strcmp(val, 'Puerto Libre');

end

%-----------------------------------------------------------------
%-----------------------------------------------------------------

%-----------------------------------------------------------------
% ////////FUNCIÓN PARA LEER INFO DEL ARDUINO de LOS SENSORES/////
% ////////////////////////////////////////////////////////////////
function [cabecera,ax,ay,az,mx,my,mz,t_readsensor]=SerialReadSensor(a)

    % Preparados para enviar

    fwrite(a.aser,'40','uchar'); %tiempo= 0.0014
    t1=toc;
    p=fscanf(a.aser,'%c'); %t_readsensor se implementó para recibir por
                           % parámetro el tiempo de lectura. Actualmente
                           % está en desuso desde que se empezo a emplear
```

```
                        % la utilidad Profiler para medida de tiempos.
t2=toc;
t_readsensor=t2-t1;
%disp(['Lectura: ' p]);

% Cabecera, devuelve 40
count=1;
c=p(count);
val=[c];

while ~isspace(c)
    count=count+1;
    c=p(count);
    val=[val c];
end
cabecera=str2num(val);

% Acelerometro eje x
count=count+1;
c=p(count);
val=[c];

while ~isspace(c)
    count=count+1;
    c=p(count);
    val=[val c];
end
ax=str2num(val);

% Acelerometro eje y
count=count+1;
c=p(count);
val=[c];

while ~isspace(c)
    count=count+1;
    c=p(count);
    val=[val c];
end
ay=str2num(val);

% Acelerometro eje z
count=count+1;
c=p(count);
val=[c];
while ~isspace(c)
    count=count+1;
    c=p(count);
    val=[val c];
end
az=str2num(val);

% Magnetometro eje x
count=count+1;
c=p(count);
val=[c];
while ~isspace(c)
    count=count+1;
    c=p(count);
    val=[val c];
end
```

```
    mx=str2num(val);

    % Magnetometro eje y
    count=count+1;
    c=p(count);
    val=[c];
    while ~isspace(c)
        count=count+1;
        c=p(count);
        val=[val c];
    end
    my=str2num(val);

      % Magnetometro eje z
    count=count+1;
    c=p(count);
    val=[c];
    while ~isspace(c)
        count=count+1;
        c=p(count);
        val=[val c];
    end
    mz=str2num(val);

%     % Buscamos si está libre el puerto
%     count=count+1;
%     c=p(count);
%     val=[c];
%     while ~isspace(c)
%         count=count+1;
%         c=p(count);
%         val=[val c];
%     end
%     count=count+1;
%     c=p(count);
%     val=[val c];
%     while ~isspace(c)
%         count=count+1;
%         c=p(count);
%         val=[val c];
%     end
%     val=val(1:length(val)-1);
%     ArduinoFree=strcmp(val, 'Puerto Libre');

end


%-------------------------------------------------------------------
%-------------------------------------------------------------------

% ////////FUNCIONES SETUP PARA ARDUINO DE SENSORES///////////////
% ////////////////////////////////////////////////////////////////

%Funciones para modificar ciertas variables en el Arduino que controla los
%sensores. Actualmente en deshuso, se implementaron para probar distintos
%filtros de media en el sketch.

function setupNumMedidas(a, numMedidas)

    usteps_vect=int2digits(numMedidas);
```

```
    numDigitos=length(usteps_vect);
    numMedidas=num2str(numMedidas);
    numDigitos=num2str(numDigitos);
    p0=num2str(4);
    p1=num2str(1);

    P=[p0 p1 numDigitos numMedidas]

    fwrite(a.aser,'P','uchar');
end


function setupTipoPromediado(a, tipoPromedio)   %tipoPromedio 0: medida bruta
                                                %             1: media aritmetica
    tipoPromedio=num2str(tipoPromedio)          %             2: filtro mediana
    p0=num2str(4);
    p1=num2str(2);

    P=[p0 p1 tipoPromedio]

    fwrite(a.aser,'P','uchar');
end


%-----------------------------------------------------------------
%-----------------------------------------------------------------


% ////////FUNCIÓN PARA LEER SI EL ARDUINO ESTÁ LIBRE//////////////
% ////////////////////////////////////////////////////////////////

function val=isArduinoFree(a)



end

% ////////FUNCIÓN PARA MARCAR POSICIONES 0 POR SOFT //////////////
% ////////////////////////////////////////////////////////////////

%Se resetean las variables que indican la posición del motor. Así se
%establece el origen en la posición actual.

function [mot_1,mot_2,mot_3,mot_4,tiempo,ArduinoFree]=ResetPosicion(a)
    fwrite(a.aser,'03','uchar');
    [mot_1,mot_2,mot_3,mot_4,tiempo,ArduinoFree]=SerialRead(a);

end




%---------------FUNCIONES DE LA CLASE ORIGINAL-------------------
%----------------------------------------------------------------

%Estas funciones no han sido modificadas ni tampoco son utilizadas en el
%proyecto. No obstante se conservan por si son útiles en el futuro.
```

ESTAS FUNCIONES NO SE VAN A INCLUIR EN ANEXOS PUESTO QUE NO HAN SIDO USADAS DURANTE EL PROYECTO.

2.4 Funciones de calibración de sensores en Matlab.

```
function [ACC]=getAccCalibMatrix(w1, w2, w3, w4, w5, w6)

% procedimiento para calibrar el sensor del acelerometro. Se trata de hacer una
% serie de medidas en posiciones fijas para conseguir la matriz de
% acondicionamiento:


% MATRIZ DE ACONDICIONAMIENTO (CALIBRACIÓN + NORMALIZACIÓN)
% ACC=zeros(3,4);


% A=[Ax, Ay, Az, 1]*ACC; Calculo de valores normalizados del acelerómetro.
% Ax1=A(1,1); OJO! Es un vector!
% Ay1=A(1,2);
% Az1=A(1,3);

%PROCEDIMIENTO
% Para cada posición fija cogemos n medidas de Ax, Ay,y Az, y promediamos
% Obtenemos así la matriz wn que nos permitirá calcular los coefs. de
% calibración.

% P1
y1=[0 0 1];

% P2
y2=[0 0 -1];

% P3
y3=[0 1 0];

% P4
y4=[0 -1 0];

% P5
y5=[1 0 0];

% P6
y6=[-1 0 0];

Y=[y1;y2;y3;y4;y5;y6];

% Ahora las medidas en forma de vector de 4 componentes

% INICIALIZO IGUAL QUE Y, por inicializar...
w1=[w1; 1];
w2=[w2; 1];
w3=[w3; 1];
w4=[w4; 1];
w5=[w5; 1];
w6=[w6; 1];

W=[w1';w2';w3';w4';w5';w6'];
```

```
%Calculamos la matriz ACC

%ACC=inv(W'*W)*W'*Y;
ACC=(W'*W)\W'*Y;

end
```

```
function [Ax1,Ay1,Az1]=normalize_acc(Ax, Ay, Az, ACC)
    % Param de entrada: Medidas crudas y matriz de calibración
    % Output: Medidas normalizadas y calibradas.
    A=[Ax Ay Az 1]*ACC;
    Ax1=A(1);
    Ay1=A(2);
    Az1=A(3);

      %Si se sale de +-1 truncar

     if abs(Ax1)>1,
      Ax1=sign(Ax1);
    end
    if abs(Ay1)>1,
      Ay1=sign(Ay1);
    end
    if abs(Az1)>1,
      Az1=sign(Az1);
    end

end

%------------------------------
```

```
function [X,M_OS,M_SC]=getMagCalibMatrix(M)

% procedimiento para calibrar el sensor magnético. Se trata de hacer una
% serie de medidas en posiciones fijas para conseguir la matriz de
% acondicionamiento:

% Cada medida m es un vector mi=[mx, my, mz] con medidas crudas en
% diferentes posiciones


% MATRIZ DE ACONDICIONAMIENTO (CALIBRACIÓN + NORMALIZACIÓN)
%X=zeros(6,1);
```

```
%PROCEDIMIENTO
% Necesitamos una serie de medidas "aleatorias" en 3D
% Según el application datasheet, son 3 rotaciones circulares 2D
% Podemos utilizar las de las 6 posiciones calculadas para el acelerómetro.
% No es necesario que tengan un ritmo regular o que estén paradas. Pero lo
% aplicaremos así

% PARA UNA PRIMERA VERSIÓN UTILIZAREMOS UN SISTEMA CON 6 MEDIDAS.
% CABE LA POSIBILIDAD DE NECESITAR AUMENTAR EL NUMERO DE MEDIDAS PARA
% AUMENTAR LA FINURA DE LA CALIBRACIÓN

% NOTA: El datasheet hace una calibración estimando que no hay influencia
% de Soft Iron, con lo que hace que sea una matriz identidad.


% Diezmado de la matriz de valores
diezmado=1; % Uno es sin diezmado ya que cojo todas las muestras..
M=M(:,3:diezmado:end);

% Preparación con matrices completas
W=(M(1,:).^2)';
H=[M(1,:) ; M(2,:) ; M(3,:) ; -M(2,:).^2 ; -M(3,:).^2 ; ones(1, length(M))]';

% Calculamos la matriz X

X=((H'*H)\H')*W;

% Seguimos calculando valores
M_OSx=X(1)/2;
M_OSy=X(2)/(2*X(4));
M_OSz=X(3)/(2*X(5));

A=X(6) + M_OSx^2 + X(4)*M_OSy^2 + X(5)*M_OSz^2;
B=A/X(4);
C=A/X(5);

M_SCx=sqrt(A);
M_SCy=sqrt(B);
M_SCz=sqrt(C);

% MATRIZ DE CORRECCIÓN DE HARD IRON OFFSET
M_OS=[M_OSx;...
      M_OSy;...
      M_OSz];

% MATRIZ DE CORRECCIÓN DE SCALE FACTOR
M_SC=[1/M_SCx   0      0;...
      0      1/M_SCy   0;...
      0        0    1/M_SCz];

end


function [Mx1,My1,Mz1]=normalize_mag(Mx, My, Mz, M_OS, M_SC)
    % Param de entrada: Medidas crudas y matriz de calibración
    % Output: Medidas normalizadas y calibradas.

    % Parece ser que es el único método descrito.
    % Ya que si el efecto del soft Iron es despreciable,
    % se aproxima por la matriz identidad.
```

```
    M_SI=eye(3);

    MxOS=[Mx - M_OS(1); My - M_OS(2); Mz - M_OS(3)];
    M=M_SC * M_SI * MxOS;

    Mx1=M(1);
    My1=M(2);
    Mz1=M(3);
end

%--------------------------------
```

```
function [pitch,roll]=position(Ax1, Ay1)
    % Calculo del pitch y el roll
    % La entrada deben ser las medidas calibradas pasadas por normalize_acc

    pitch = asin(-Ax1);
    if abs(pitch)==pi/2, %evita el error en +-pi/2
        pitch=0;
    end
    roll = asin(Ay1/cos(pitch));

end

%------------------------------
```

```
function [heading,check_mag]=heading(Mx1, My1, Mz1, pitch, roll)
    % Param de entrada: Valores normalizados y calibrados
    % Output: Heading!

    Mx2=Mx1*cos(pitch) + Mz1*sin(pitch);
    My2=My1*sin(roll)*sin(pitch) + My1*cos(roll) - Mz1*sin(roll)*cos(pitch);
    Mz2=-Mx1*cos(roll)*sin(pitch) + My1*sin(pitch) + Mz1*cos(roll)*cos(pitch);

    check_mag=test_mag(Mx2,My2,Mz2);
    heading=0;

    % GRADOS
%     if Mx2>0,
%         if My2>=0,
%             heading=atand(My2/Mx2);
%         else
%             heading=360+atand(My2/Mx2);
```

```
%         end
%     else
%         if Mx2<0,
%                 heading=180 + atand(My2/Mx2);
%         else
%             if My2>=0,
%                 heading=270;
%             else
%                 heading=90;
%             end
%         end
%     end
%
    % RADIANES
    if Mx2>0,
        if My2>=0,
            heading=atan(My2/Mx2);
        else
            heading=2*pi+atan(My2/Mx2);
        end
    else
        if Mx2<0,
            heading=pi + atan(My2/Mx2);
        else
            if My2>=0,
                heading=3*pi/2;
            else
                heading=pi/2;
            end
        end
    end


end

%------------------------------
```

3. Planos.

A continuación se adjuntan los planos de hardware más importantes, a saber:

1- Esquemático de la placa del sensor

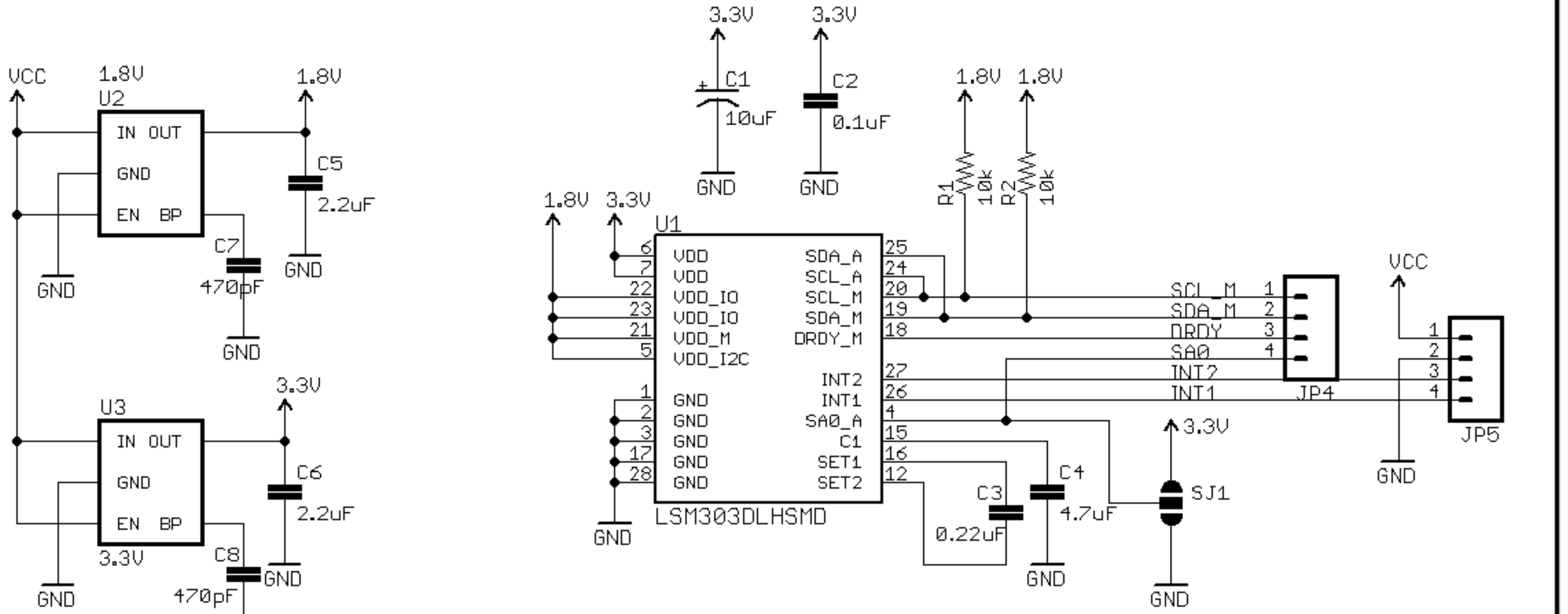
2- Plano PCB *Top* de la placa del sensor

3- Plano PCB *Bottom* de la placa del sensor

4- Esquemático de Arduino

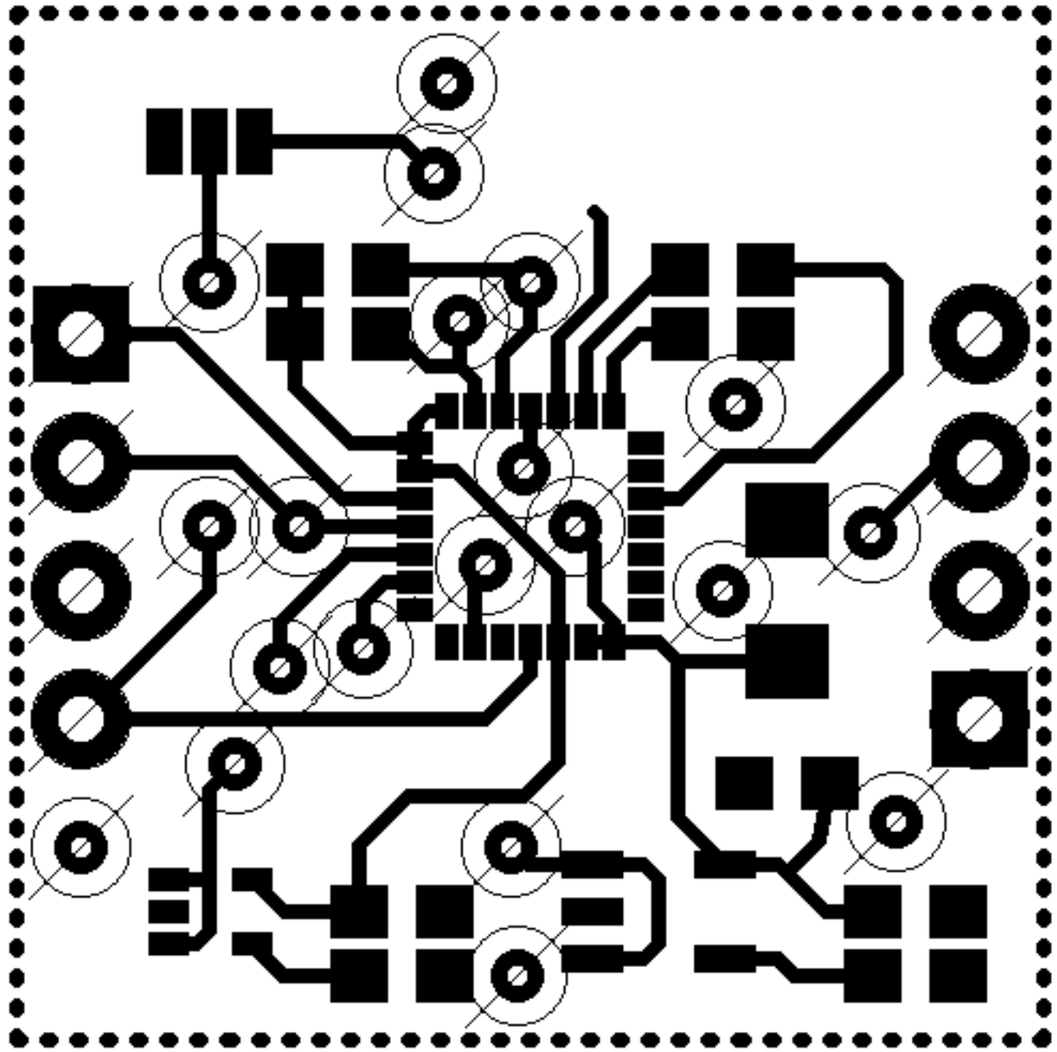
5- Plano PCB *Top* de Arduino

6- Plano PCB *Bottom* de Arduino



	Fecha	Nombre	Firma:	ESCUELA DE INGENIERÍA Y ARQUITECTURA
Dibujado	24/10/11	Javier Martínez Amo		
Comprobado				

Escala:	Plano esquemático de la placa del sensor	Plano n.º 1	
		N.º Alumno:	
		Curso: 3º	

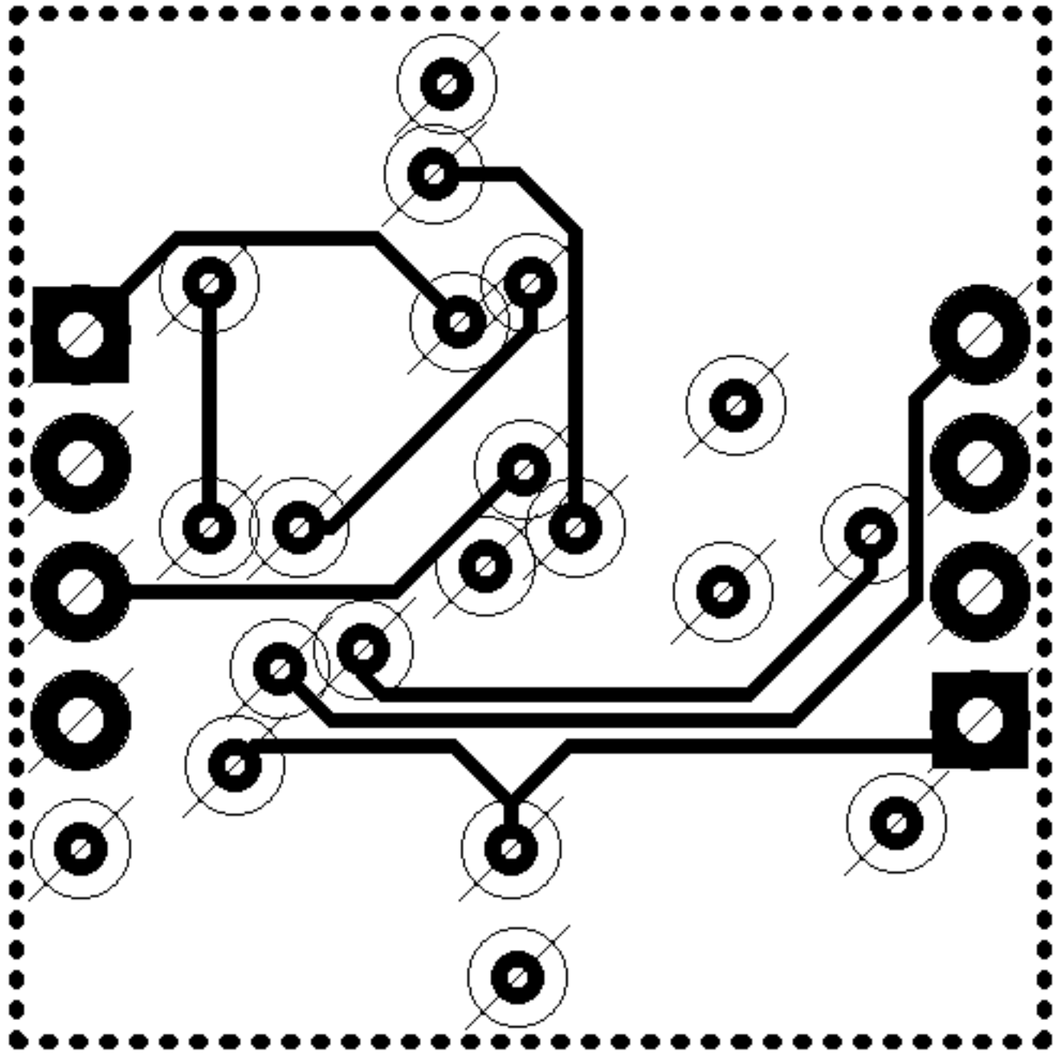


	Fecha	Nombre	Firma:	ESCUELA DE INGENIERÍA Y ARQUITECTURA
Dibujado	24/10/11	Javier Martínez Amo		
Comprobado				

Escala:		Plano n.º
4:1	Plano de la cara Top de la placa del sensor. Pistas, pads y vías	2

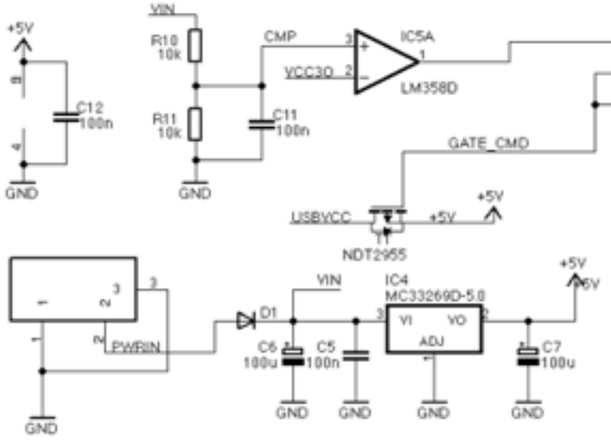
N.º Alumno:

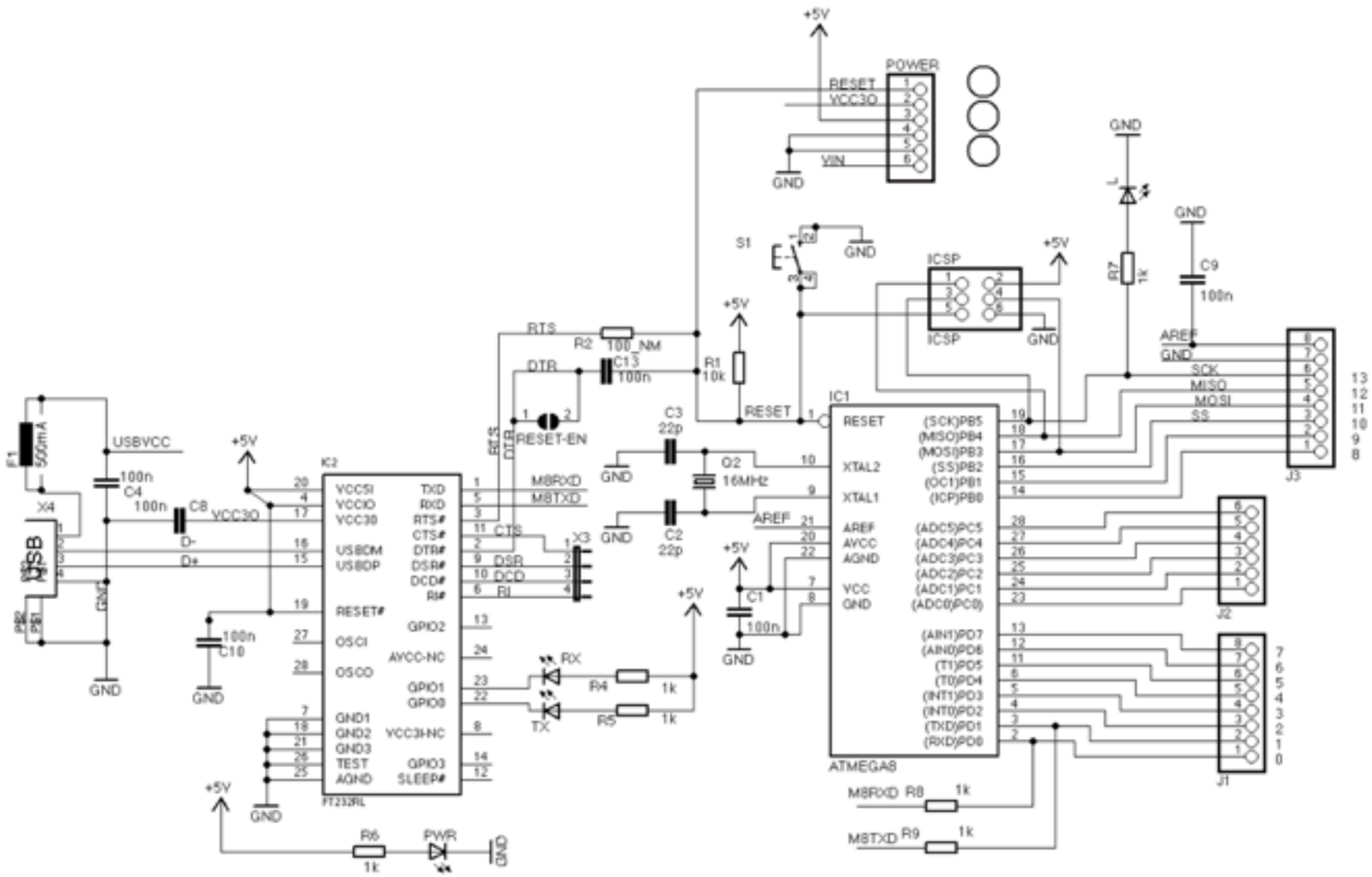
Curso: 3º



	Fecha	Nombre	Firma:	ESCUELA DE INGENIERÍA Y ARQUITECTURA
Dibujado	24/10/11	Javier Martínez Amo		
Comprobado				

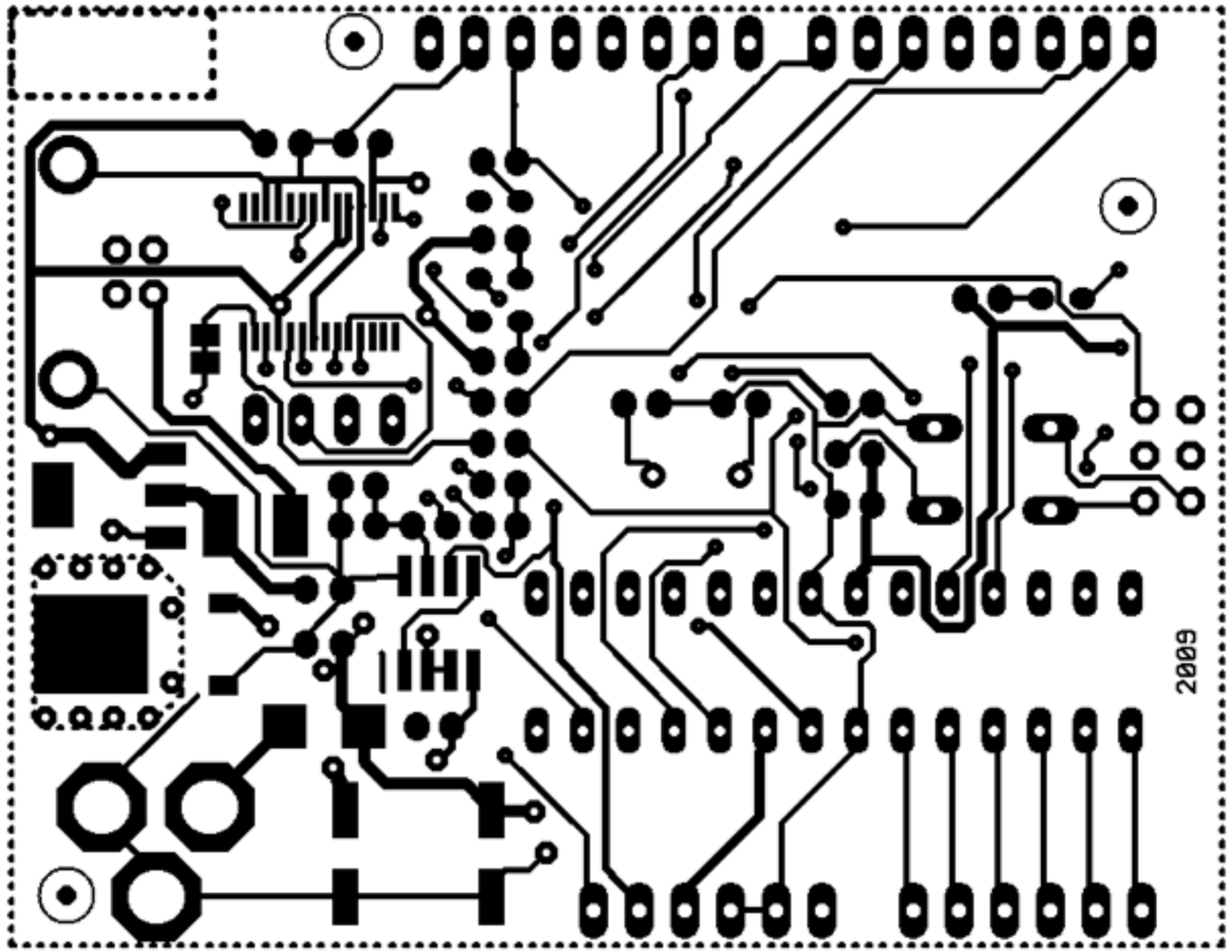
Escala:	Plano de la cara Bottom de la placa del sensor con pistas y vías.	Plano n.º 3
4:1		N.º Alumno:
		Curso: 3º





	Fecha	Nombre	Firma:	ESCUELA DE INGENIERÍA Y ARQUITECTURA
Dibujado	24/10/11	Javier Martínez Amo		
Comprobado				

Escala:	Plano esquemático de la placa Arduino	Plano n.º 4	
		N.º Alumno:	
		Curso: 3º	

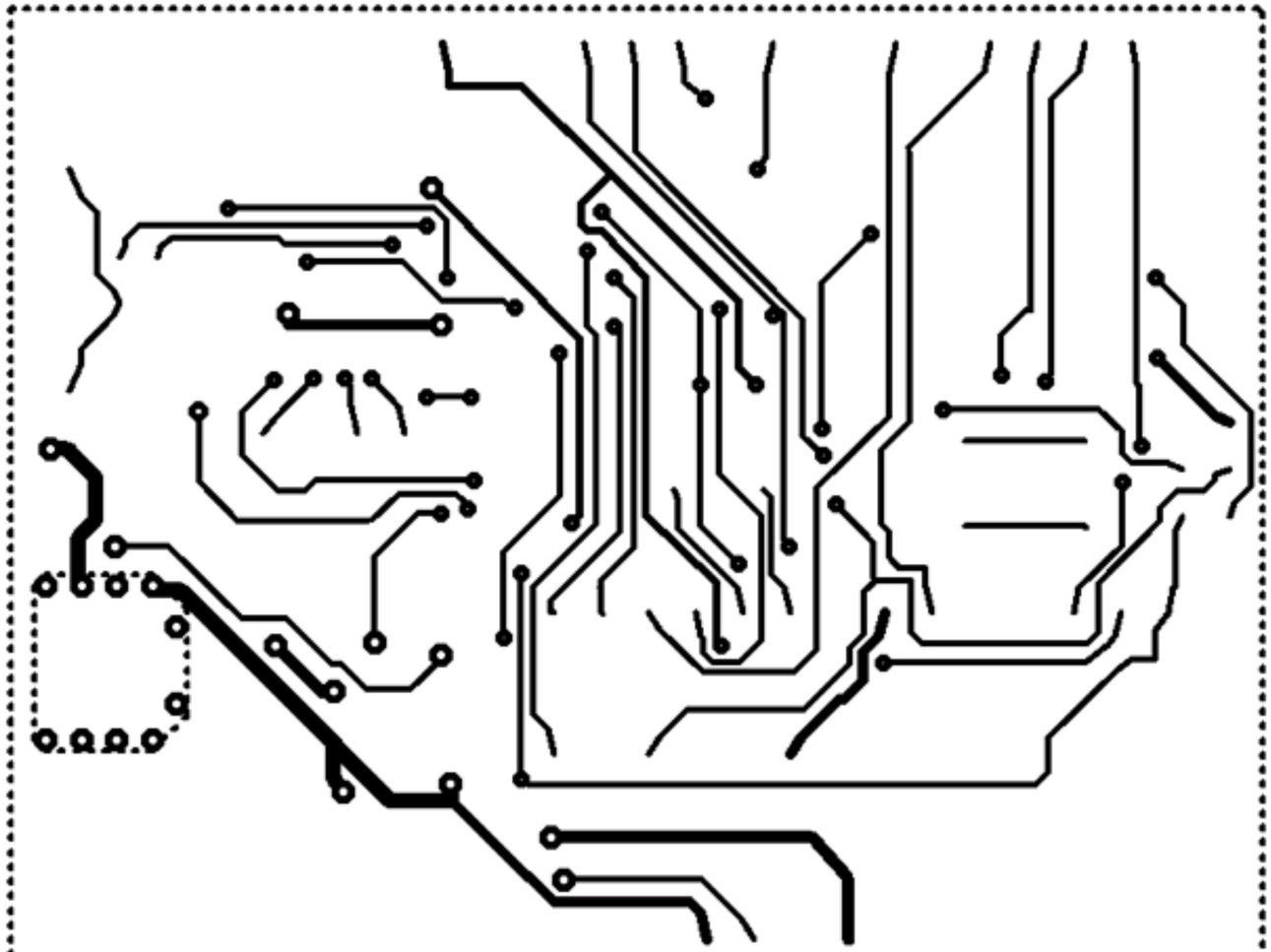


	Fecha	Nombre	Firma:	ESCUELA DE INGENIERÍA Y ARQUITECTURA
Dibujado	24/10/11	Javier Martínez Amo		
Comprobado				

Escala:		Plano n.º
1.5:1	Plano de la cara Top de la placa Arduino. Pistas, pads y vías	5

N.º Alumno:

Curso: 3º



	Fecha	Nombre	Firma:	ESCUELA DE INGENIERÍA Y ARQUITECTURA
Dibujado	24/10/11	Javier Martínez Amo		
Comprobado				

Escala:		Plano n.º
1.5:1	Plano de la cara Bottom de la placa Arduino. Pistas y vías	6

N.º Alumno:

Curso: 3º