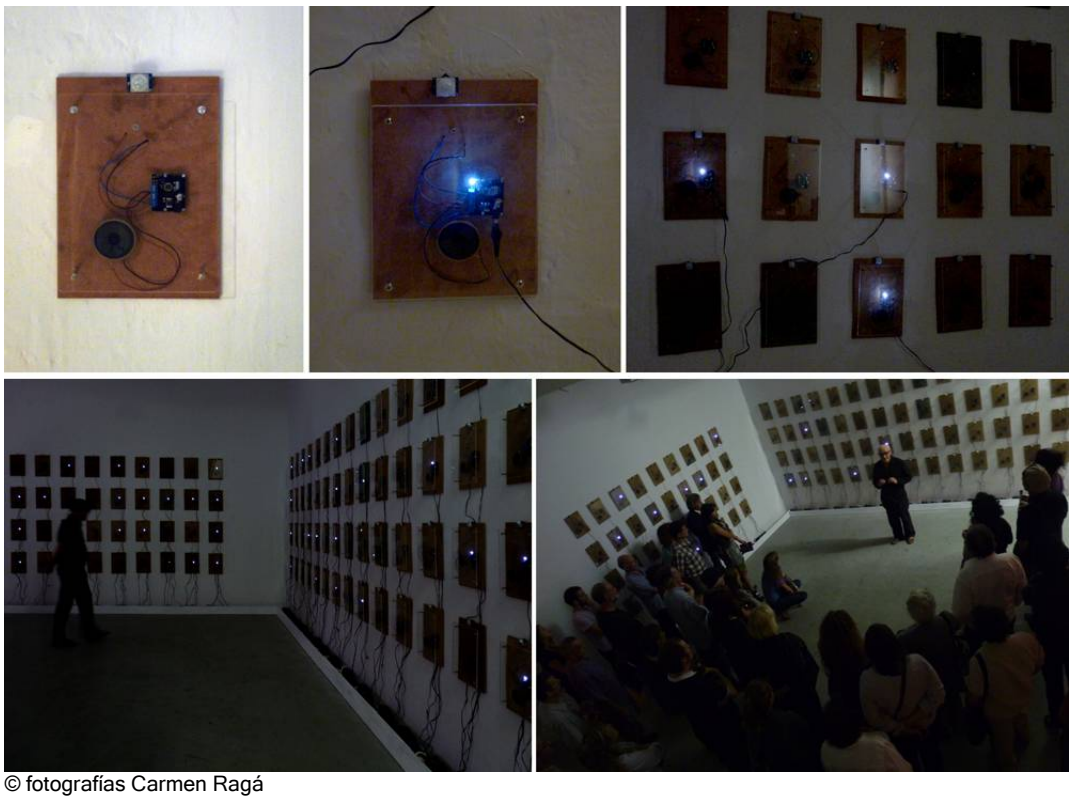
TERNURAS DE GUERRA

Concepto, diseño y realización: Antonio Alvarado.
Implementación del software y el hardware: Monster Electronic (Juan Fabián), Antonio Alvarado y galería Weber-Lutgen.
2010-2011-2012.

Instalación electrónica interactiva sobre planchas de DM y plástico.
100 Planchas tipo. Cada plancha consta de DM (Din A4), una placa basada en arduino, altavoz, sensor PIR y potenciómetro.
100 Transformadores de potencia.
25 Regletas de diversificación eléctrica.
Software especifico de arduino.
Dimensiones y distribución variables según la sala.

Ha sido presentada en:
Cruce. Madrid, España, octubre 2011.
Galería Weber-Lutgen, Sevilla, España. 2012-2013.
La Neomudejar. Madrid: España. 2016.
Museo Zapadores: Madrid. España. 2018-2020
Galería Nacional. San José. CostaRica. 2021.



© fotografías Carmen Ragá

El objetivo de una batalla es aniquilar al enemigo. Aniquilar no significa destruir físicamente pero si dejarle inutilizado para la batalla.
Hay tácticas que recuerdan la práctica amorosa, encierran un alto grado de ternura, una ternura envolvente que va cerrando caminos y que destruye, mas que por las armas, por el espíritu. Es este tipo de guerra, la mas aniquilante, es la que he quiero mostrar en esta obra.

Ternuras de guerra, igual que un regimiento, está formada por elementos independientes, que pueden morir independientemente, fallar y ser sustituidos independientemente pero que actúan en grupo. Si uno de los elementos se estropea o no funciona, el resto sigue cumpliendo su misión de acoso moral.

http://youtu.be/TA_ZzO2a_KE

Descripción funcional

Ternuras de guerra es una obra interactiva que se vale de sensores de movilidad, de placas electrónicas, de altavoces y otros elementos auxiliares.

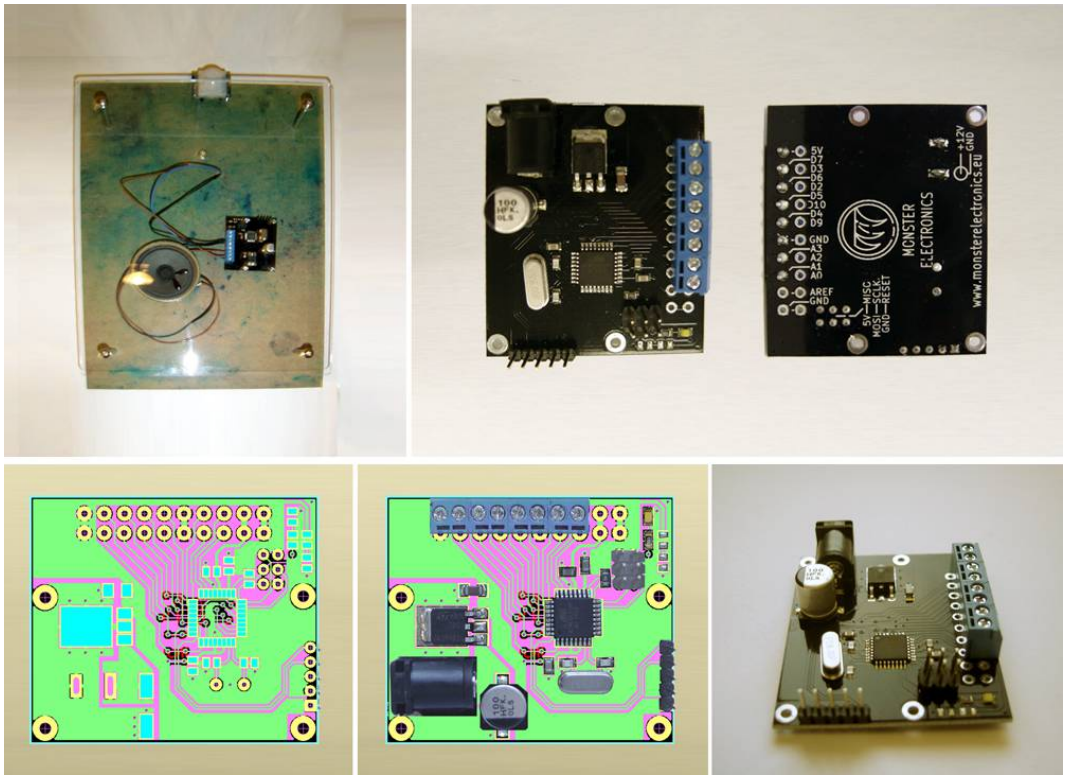
La placa electrónica consiste en un dispositivo programable (microcontrolador) conectado a un sensor de presencia PIR y un pequeño altavoz. El programa que contiene el microcontrolador evalúa los datos que obtiene del sensor para producir sonido por el altavoz. Además se puede conectar un potenciómetro que permite ajustar la frecuencia del sonido que emite el altavoz.

El hardware es un diseño hecho a medida compatible con la plataforma Arduino, permitiendo de este modo que el software pueda ser reescrito con facilidad por una amplia comunidad de programadores. Más adelante, en este documento, se incluye el código fuente, y también todas las herramientas necesarias para seguir o adaptar el desarrollo.

En 2012 , coincidiendo con la muestra en la galería Weber-Lutgen se dotó a cada altavoz de un potenciómetro que regula la potencia de salida del altavoz. Esto mejoró notablemente la sonoridad de la instalación.

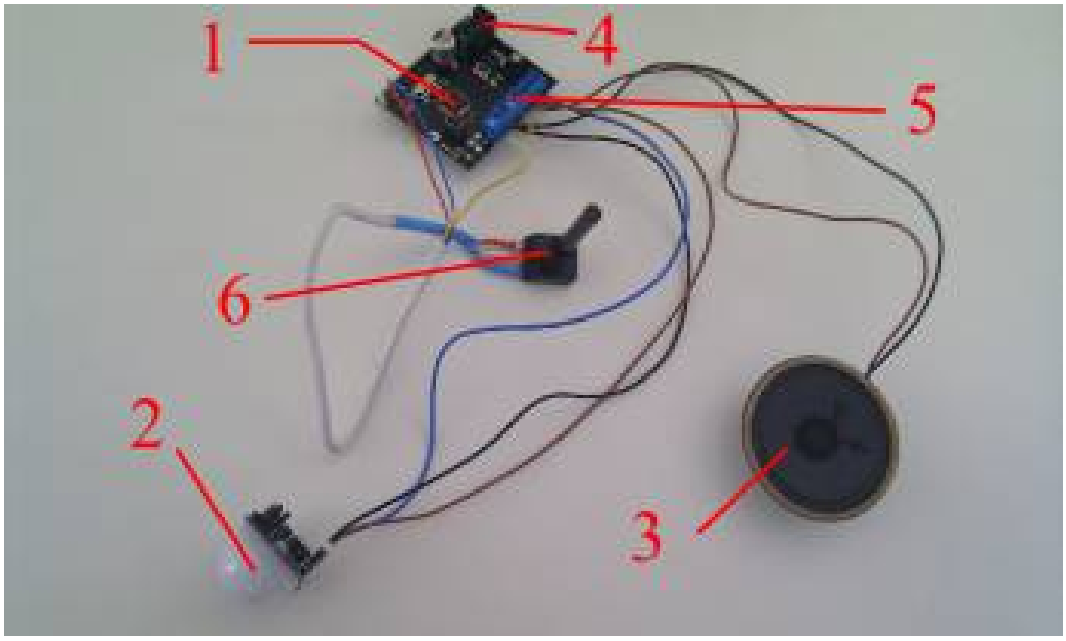
Hardware y esquema de conexiones

La plancha y la placa.



:

La placa. y sus componentes:

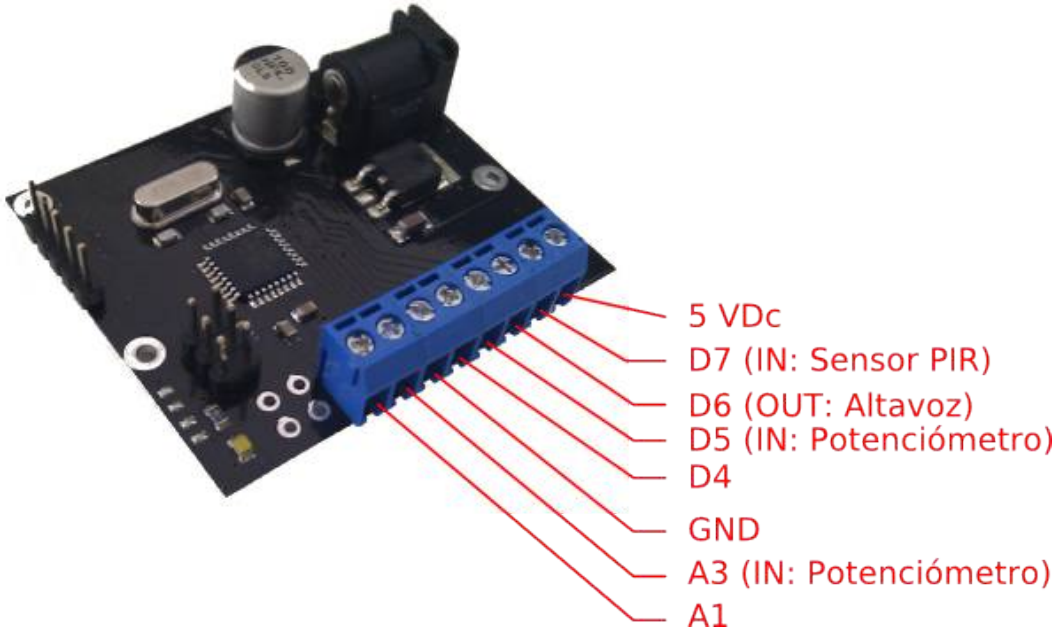


1. Microcontrolador Atmega 328P.
1. Sensor de presencia PIR.
3. Altavoz de 0.2 watios de potencia y 8 ohm de impedancia.
4. Conector de alimentación. Permite conectar fuentes de alimentación de 9 a 15 Vdc.
5. Clemas de conexión.
6. Potenciómetro (solo se usa para regular el software).

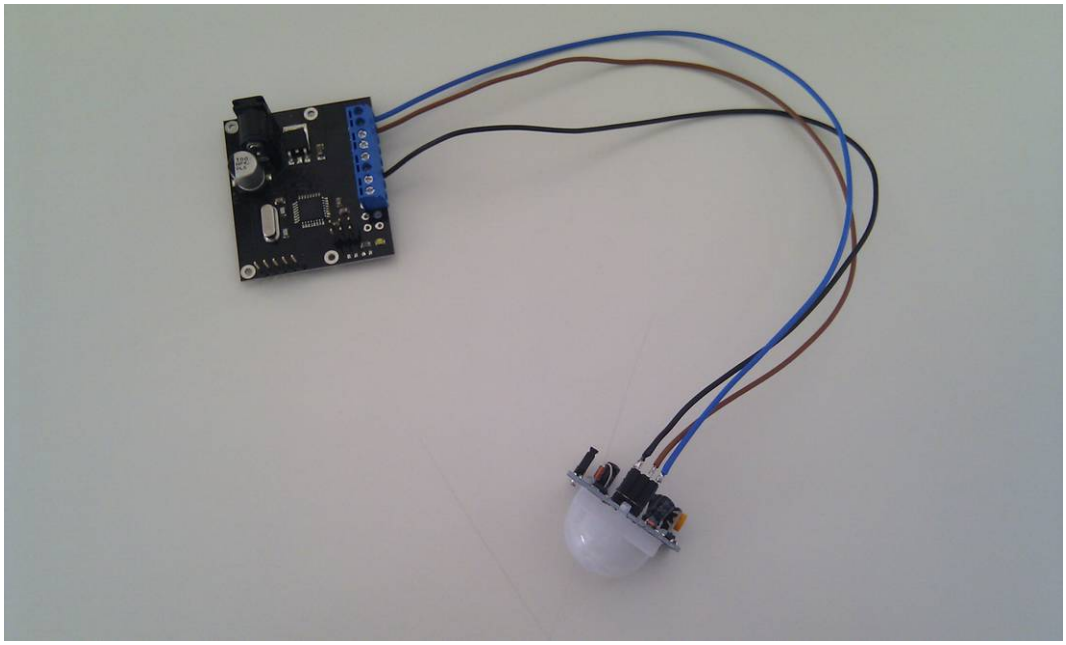
En las secciones siguientes se muestran detalles de las conexiones de todos los elementos al dispositivo Arduino. Hay que prestar cuidado y conectar los elementos siguiendo los colores de los cables y las posiciones indicadas. Si hubiese que sustituir algún elemento, estas conexiones deben realizarse con cuidado pues se podrían dañar tanto los componentes como el dispositivo.

Clemas de conexión

A continuación se muestran las clemas de conexión y las posiciones en las que los diferentes componentes deben ser colocados. Se muestra a su vez la traducción a los nombres de las entradas y salidas tal y como son descritas en la documentación de Arduino:

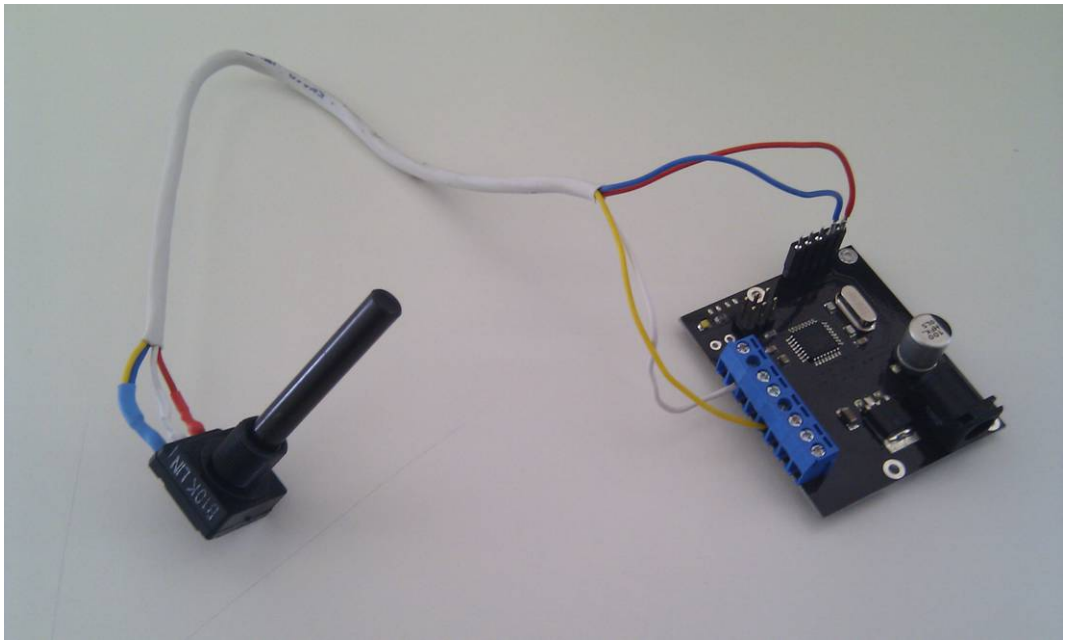


Detalle de conexión del sensor PIR a la placa

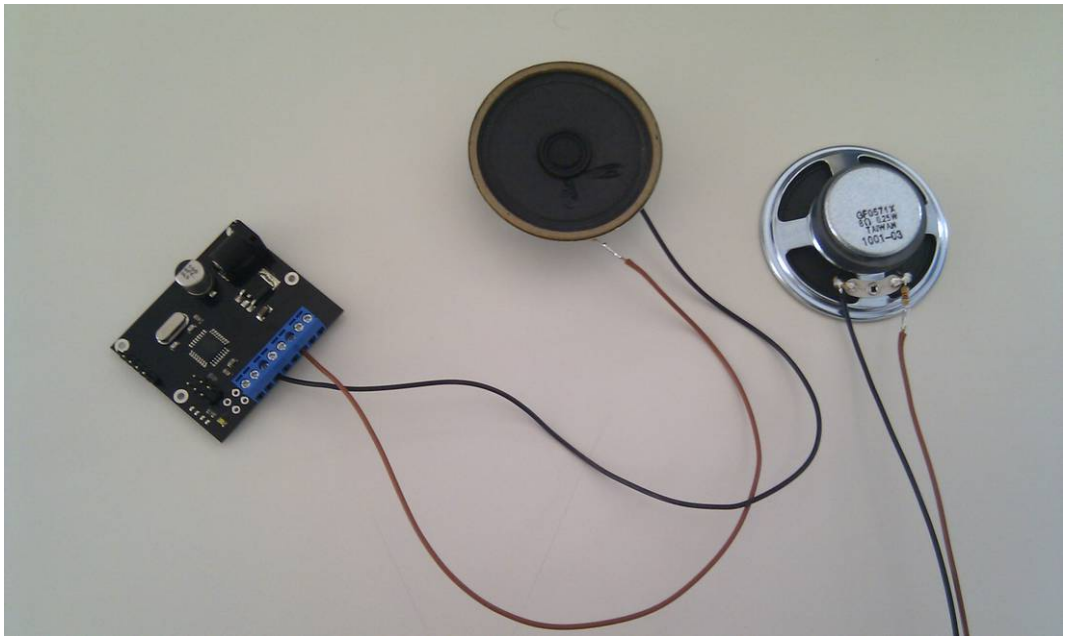


Detalle del sensor PIR

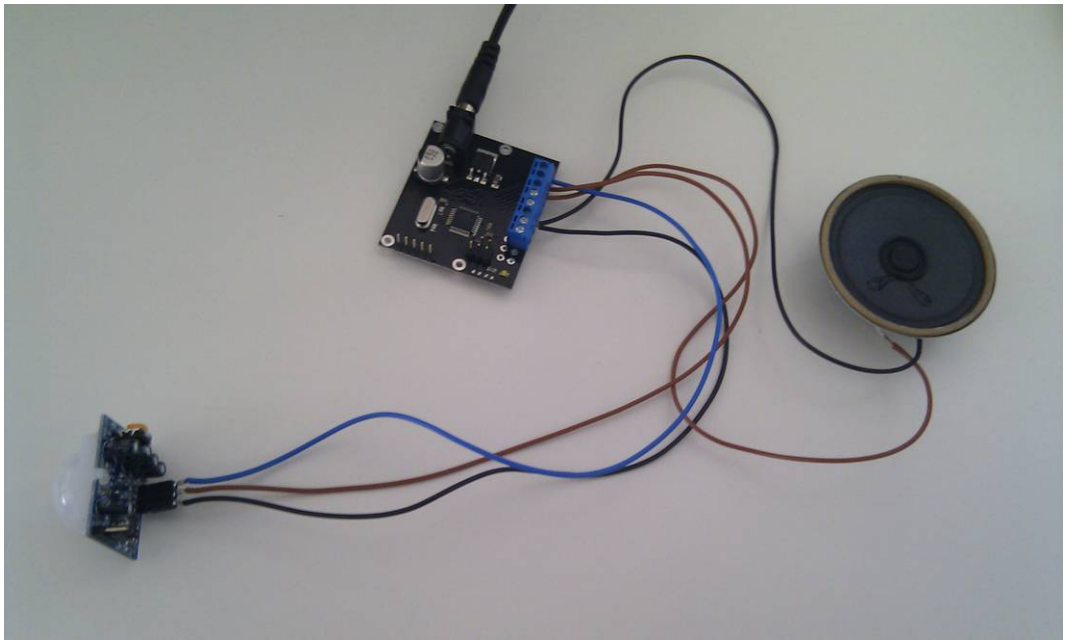
Detalle de conexión del potenciómetro de software



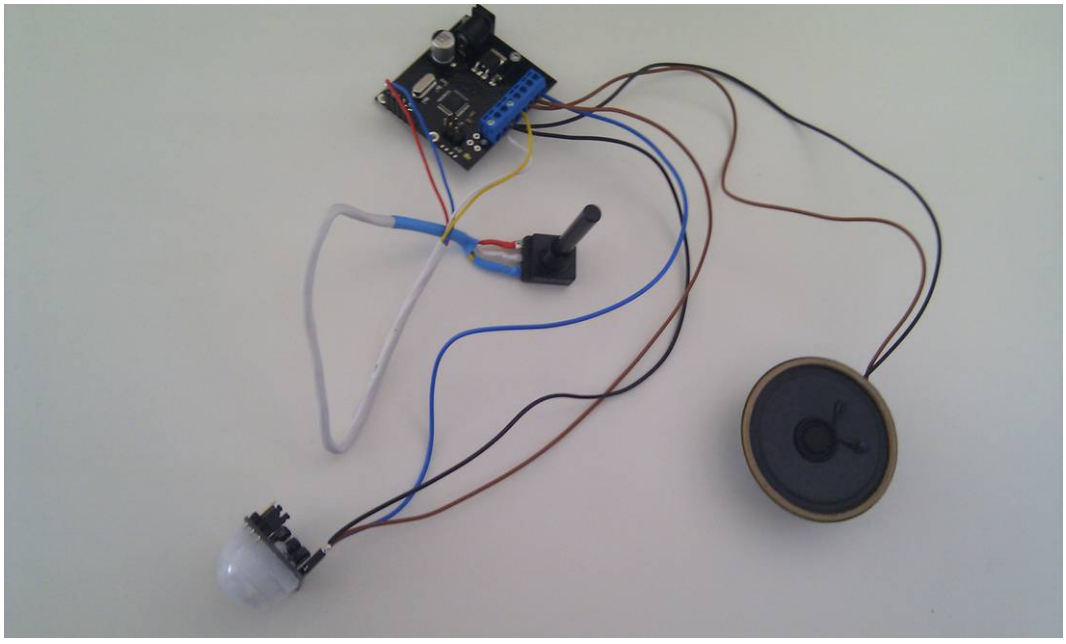
Detalle de conexión del altavoz



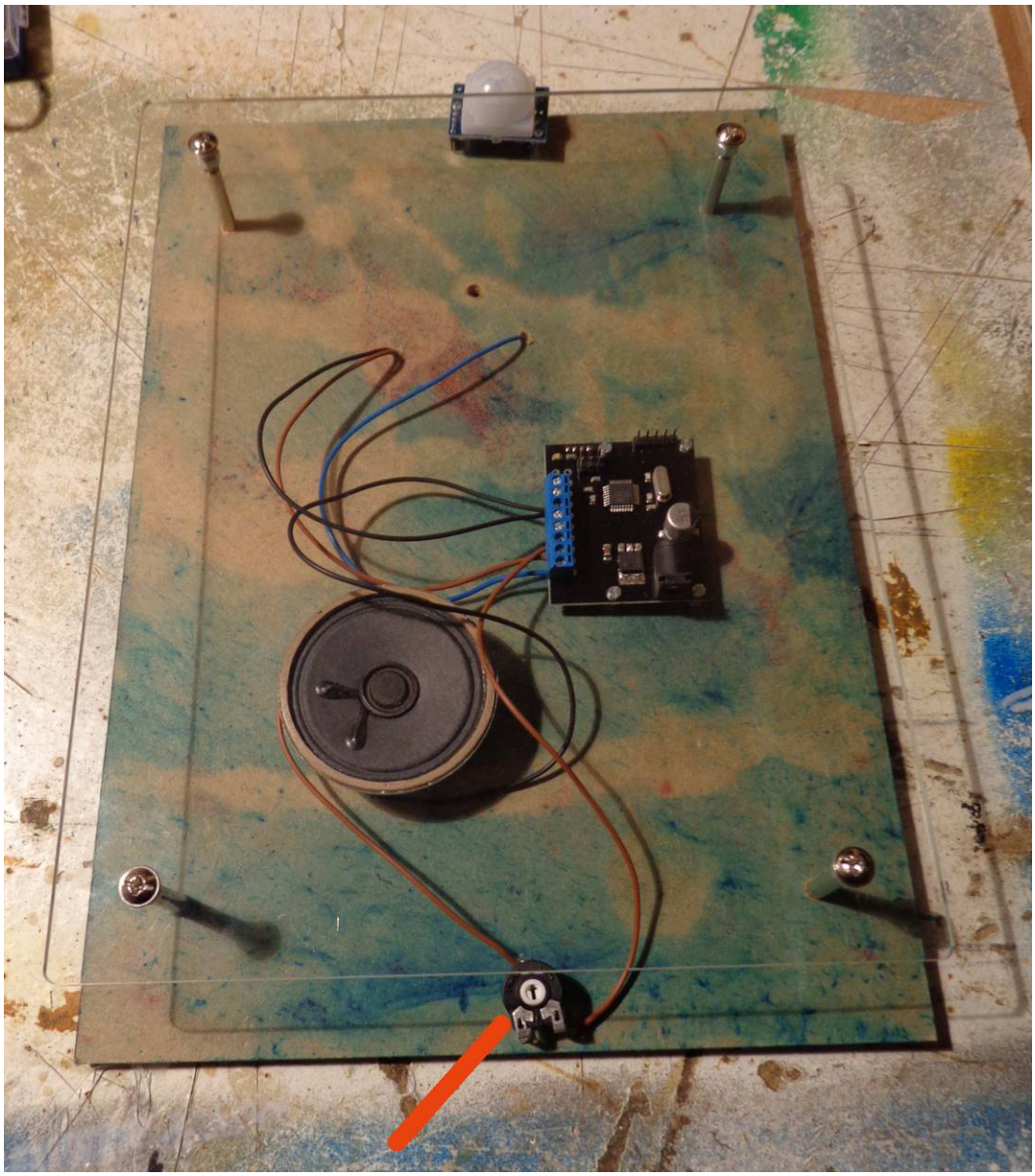
Detalles de conexión generales, antes de poner el potenciometro regulador de volumen.



Detalles de conexión generales, con potenciómetro de software



Detalles de la placa tipo tras colocar el potenciometro regulador de volumen en 2012.



Software

El programa que gobierna el hardware está desarrollado con el entorno de programación de Arduino. Debe tenerse en cuenta que la compilación del código se realizó en el verano del 2010, utilizando el entorno de programación de Arduino por lo cual se recomienda compilar con la versión "Arduino-0018" que puede bajarse de la página de Arduino: http://arduino.cc/ o consultar con el autor.
Una vez instalado Arduino, se recomienda añadir las siguientes bibliotecas: http://www.antonioalvarado.net/obras/2011/2011_001/libraries.rar

El código fuente se muestra a continuación:

Fichero: PIR_Soldiers_v006.pde

```
//-----------------------------------------------------------------
// PIR_Soldiers
//
// Version:  006
// Date:     21 Jul 2010
// Author:   fabi @ Monster Electronics
//           http://www.monsterelectronics.eu
//-----------------------------------------------------------------

#include "pitches.h"
#include <EEPROM.h>

// DEBUG variable is used just for testing purposes.
// Be sure to set DEBUG to false before burn the chip
// for production. This will reduce code footprint.
const boolean DEBUG = false;

// The time we give the sensor to calibrate (10-60 secs. according
// to the datasheet)
const byte CALIBRATION_TIME = 15; //45; //60;
const unsigned int NOTE_DURATION = 20;

const byte UNCONNECTED_ANALOG_IN = 1;
const byte POT_PIN = 3;
const byte CALIBRATION_PIN = 5;
const byte SPEAKER_PIN = 6;
const byte PIR_PIN = 7;
const byte LED_PIN = 13;

const unsigned int WAVE_CUT = 20; // Minimum 10 ms.

// EEPROM read/write position. Note that we'll read/write integer
// numbers, so we really use EEPROM_Pos and EEPROM_Pos + 1.
// Note that the Atmega328 chip contains 0xFF values for all
// the EEPROM positions by default (and after programming it).
const unsigned int EEPROM_Pos = 0;

const unsigned int MAX_NOTE = NOTE_F1;

// The time when the sensor outputs a low impulse
long unsigned int lowIn;

// The amount of milliseconds the sensor has to be low
// before we assume all motion has stopped
long unsigned int pause = 500; //2000;

boolean lockLow = true;
boolean takeLowTime;

unsigned int note;
unsigned int potValue;

boolean storeToEEPROM = true;


void setup()
{
```

```
  if( DEBUG == true )
  {
    Serial.begin( 9600 );
  }

  pinMode( LED_PIN, OUTPUT );
  pinMode( PIR_PIN, INPUT );
  digitalWrite( PIR_PIN, LOW ); // Turn on pull-down
  pinMode( CALIBRATION_PIN, INPUT );
  digitalWrite( CALIBRATION_PIN, HIGH ); // Turn on pull-up

  // Give the sensor some time to calibrate
  for( int i = 0; i < CALIBRATION_TIME; i++ )
  {
    delay( 1000 );
  }

  // Initialize random number generator
  randomSeed( analogRead( UNCONNECTED_ANALOG_IN ) );

  // Read a pitch from the EEPROM
  note = EEPROMReadInt( EEPROM_Pos );

  // Check if the EEPROM contains a pitch value previously selected
  // by the user (using the potentiometer).
  // If the EEPROM contains 0xFFFF at EEPROM_Pos, then no value was
  // ever stored, so we start with a random pitch value. Also, notes
  // above MAX_NOTE are not wanted because we want low tones.
  if( (note == 0xFFFF) || (note > MAX_NOTE) )
  {
    // Assign a random pitch (value between 0 and 1023)
    note = assignNote( random( 0, 1024 ) );

    // Store it
    EEPROMWriteInt( EEPROM_Pos, note );
  }

  // Initialize some other variables
  storeToEEPROM = true;
  lockLow = true;
}


void loop()
{
  //
  // PIR Stage
  //
  if( digitalRead( PIR_PIN ) == HIGH )
  {
    digitalWrite( LED_PIN, HIGH );
    tone( SPEAKER_PIN, note, NOTE_DURATION );

    if( lockLow )
    {
      // Makes sure we wait for a transition to LOW before any
      // further output is made
      lockLow = false;
    }
```

```
  takeLowTime = true;
}

if( digitalRead( PIR_PIN ) == LOW )
{
  digitalWrite( LED_PIN, LOW );
  noTone( SPEAKER_PIN );

  if( takeLowTime )
  {
    lowIn = millis();    // Save the time of the transition from HIGH to LOW
    takeLowTime = false; // Make sure this is only done at the start of a LOW phase
  }

  // If the sensor is LOW for more than the given pause,
  // we assume that no more motion is going to happen
  if( (lockLow == false) && (millis() - lowIn > pause) )
  {
    // Makes sure this block of code is only executed again after
    // a new motion sequence has been detected
    lockLow = true;
  }
}

// Some delay that cut the sound wave to avoid pure tones, so
// the sound seems to be more organic.
delay( WAVE_CUT );

//
// Analog read Stage
//
potValue = analogRead( POT_PIN );

//
// Store to EEPROM Stage
//
// To avoid analog noise, we'll only assign the tone when
// the digital pin CALIBRATION_PIN is driven to LOW using
// the provided "special" potentiometer board with 4 pins.
if( digitalRead( CALIBRATION_PIN ) == LOW )
{
  note = assignNote( potValue );

  // We protect EEPROM write because it's limited to 100000
  // write cycles. Once the value is stored we'll have to reset
  // the board to store new values.
  if( storeToEEPROM == true )
  {
    storeToEEPROM = false;
    EEPROMWriteInt( EEPROM_Pos, note );
  }

  if( DEBUG == true )
  {
    Serial.print( "Calibration = " );
    Serial.println( digitalRead( CALIBRATION_PIN ) );
    delay( 5 ); // Some delay to avoid serial port overhead
  }
}
```

```
  // Some debug information
  if( DEBUG == true )
  {
    //Serial.println( note ); // Print pitch value
    Serial.println( potValue ); // Print potentiomenter value
    delay( 5 ); // Some delay to avoid serial port overhead
  }
}


//
// Helper functions
//

// This function translates an integer value into a note
// (for more information about tones (musical notes) pitches.h)
int assignNote( unsigned int pVal )
{
  if( pVal >= 0 && pVal < 128 )
  {
    return NOTE_G0;
  }
  else if( pVal >= 128 && pVal < 256 )
  {
    return NOTE_B0;
  }
  else if( pVal >= 256 && pVal < 384 )
  {
    return NOTE_C1;
  }
  else if( pVal >= 384 && pVal < 512 )
  {
    return NOTE_D1;
  }
  else if( pVal >= 512 && pVal < 640 )
  {
    return NOTE_E1;
  }
  else if( pVal >= 640 && pVal < 768 )
  {
    return NOTE_F1;
  }
  else
  {
    return NOTE_F0;
  }
}


// This function will write a 2 byte integer to the EEPROM
// at the specified address and address + 1
void EEPROMWriteInt( unsigned int pAddress, int pValue )
{
  byte lowByte = ((pValue >> 0) & 0x00FF);
  byte highByte = ((pValue >> 8) & 0x00FF);

  EEPROM.write( pAddress, lowByte );
  EEPROM.write( pAddress + 1, highByte );
}
```

```
// This function will read a 2 byte integer from the EEPROM
// at the specified address and address + 1
unsigned int EEPROMReadInt( unsigned int pAddress )
{
  byte lowByte = EEPROM.read( pAddress );
  byte highByte = EEPROM.read( pAddress + 1 );

  return( (lowByte << 0) & 0x00FF) + ((highByte << 8) & 0xFF00 );
}
```

Libreria: Pitches.h

```
//
// Pitches.h
//
// This file contains some public constants definitions of musical
// notes frequencies (pitches).
//

#define NOTE_F0  25
#define NOTE_G0  27
#define NOTE_A0  29
#define NOTE_B0  31
#define NOTE_C1  33
#define NOTE_CS1 35
#define NOTE_D1  37
#define NOTE_DS1 39
#define NOTE_E1  41
#define NOTE_F1  44
#define NOTE_FS1 46
#define NOTE_G1  49
#define NOTE_GS1 52
#define NOTE_A1  55
#define NOTE_AS1 58
#define NOTE_B1  62
#define NOTE_C2  65
#define NOTE_CS2 69
#define NOTE_D2  73
#define NOTE_DS2 78
#define NOTE_E2  82
#define NOTE_F2  87
#define NOTE_FS2 93
#define NOTE_G2  98
#define NOTE_GS2 104
#define NOTE_A2  110
#define NOTE_AS2 117
#define NOTE_B2  123
#define NOTE_C3  131
#define NOTE_CS3 139
#define NOTE_D3  147
#define NOTE_DS3 156
#define NOTE_E3  165
#define NOTE_F3  175
#define NOTE_FS3 185
#define NOTE_G3  196
#define NOTE_GS3 208
#define NOTE_A3  220
#define NOTE_AS3 233
#define NOTE_B3  247
#define NOTE_C4  262
#define NOTE_CS4 277
#define NOTE_D4  294
#define NOTE_DS4 311
#define NOTE_E4  330
#define NOTE_F4  349
#define NOTE_FS4 370
#define NOTE_G4  392
#define NOTE_GS4 415
#define NOTE_A4  440
#define NOTE_AS4 466
#define NOTE_B4  494
```

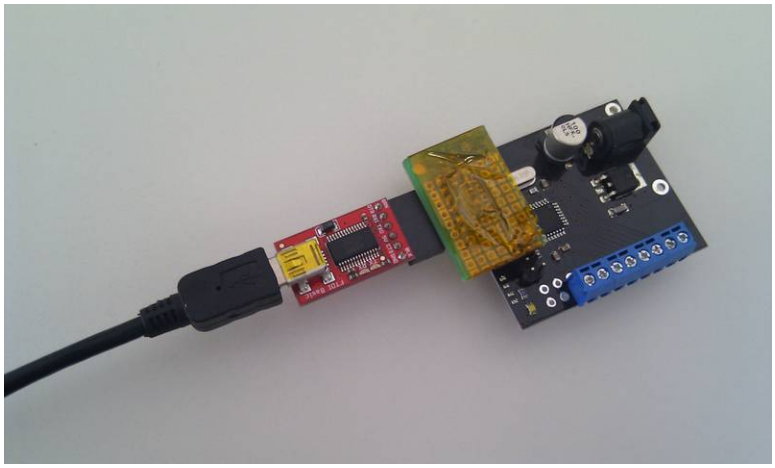
```
#define NOTE_C5  523
#define NOTE_CS5 554
#define NOTE_D5  587
#define NOTE_DS5 622
#define NOTE_E5  659
#define NOTE_F5  698
#define NOTE_FS5 740
#define NOTE_G5  784
#define NOTE_GS5 831
#define NOTE_A5  880
#define NOTE_AS5 932
#define NOTE_B5  988
#define NOTE_C6  1047
#define NOTE_CS6 1109
#define NOTE_D6  1175
#define NOTE_DS6 1245
#define NOTE_E6  1319
#define NOTE_F6  1397
#define NOTE_FS6 1480
#define NOTE_G6  1568
#define NOTE_GS6 1661
#define NOTE_A6  1760
#define NOTE_AS6 1865
#define NOTE_B6  1976
#define NOTE_C7  2093
#define NOTE_CS7 2217
#define NOTE_D7  2349
#define NOTE_DS7 2489
#define NOTE_E7  2637
#define NOTE_F7  2794
#define NOTE_FS7 2960
#define NOTE_G7  3136
#define NOTE_GS7 3322
#define NOTE_A7  3520
#define NOTE_AS7 3729
#define NOTE_B7  3951
#define NOTE_C8  4186
#define NOTE_CS8 4435
#define NOTE_D8  4699
#define NOTE_DS8 4978
```

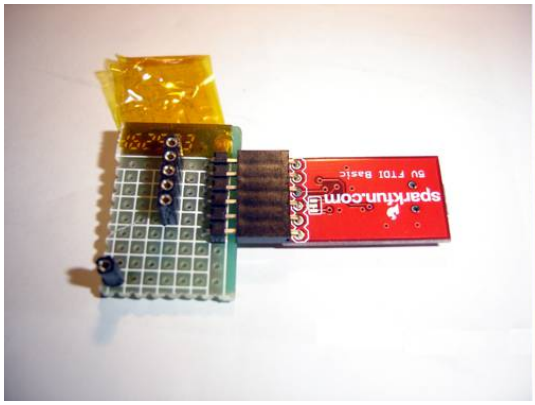
Instalación y operación

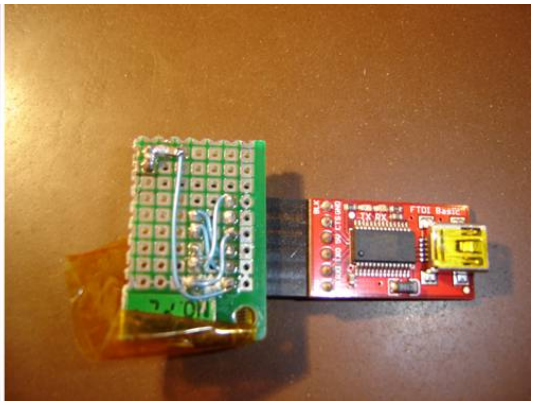
El dispositivo electrónico funciona de forma autónoma, necesitando únicamente conectar los diferentes elementos tal y como se indica en el esquema de conexiones anterior. Tanto el sensor como el altavoz deben ser conectados antes de conectar la alimentación. En caso de conectar el potenciómetro de ajuste del sonido, se debe conectar también antes de conectar la alimentación.
Cuando el dispositivo se enciende por primera vez se ejecuta el programa almacenado, seleccionando un sonido aleatorio que emitirá el altavoz. La frecuencia de dicho sonido puede ser alterada con el dispositivo encendido (en tiempo de ejecución) si previamente se ha conectado el potenciómetro a las clemas antes indicadas. Cuando se utilice el potenciómetro, el valor de éste quedará almacenado en la memoria no volátil del microcontrolador, de tal forma que en los sucesivos encendidos del dispositivo se usará el valor almacenado en lugar del valor aleatorio inicial. No obstante, se puede volver a conectar el potenciómetro nuevamente para variar la frecuencia del sonido a emitir y ésta quedará almacenada, reemplazando a la anterior.

Reprogramación del software

Con el fin de poder reprogramar el dispositivo se suministran dos programadores que se conectan por un lado a un puerto USB de un ordenador, y por el otro al dispositivo, a través de un conector Mini USB, tal y como muestra la siguiente figura:

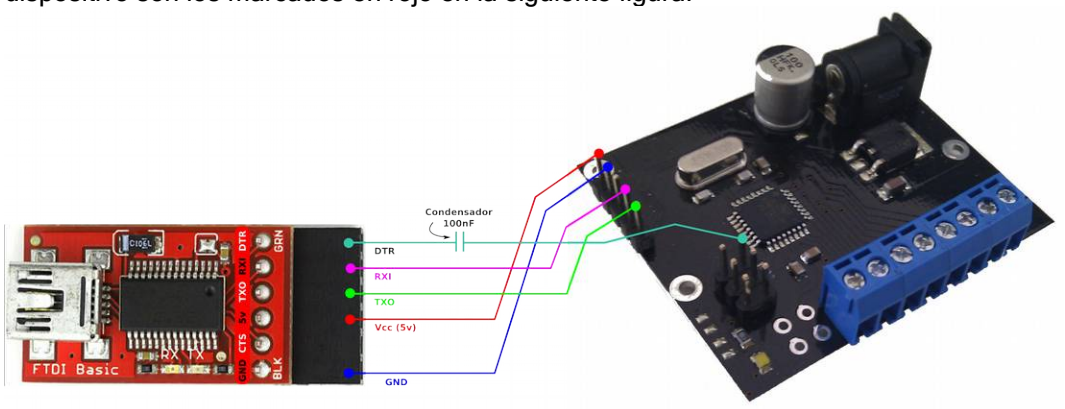


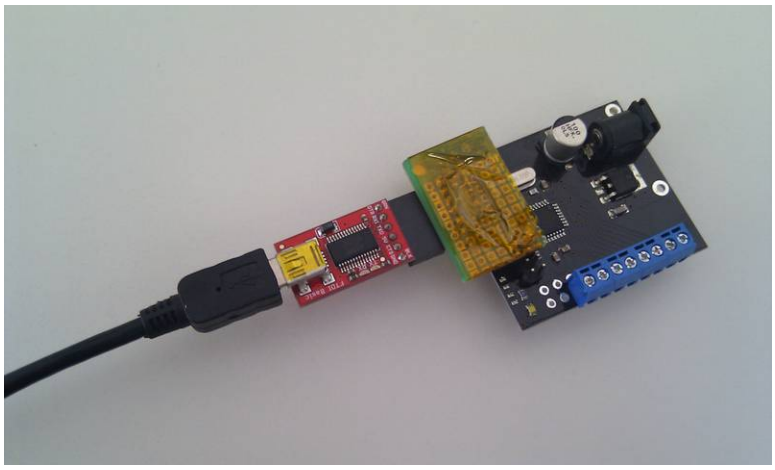




MUY IMPORTANTE: cuando se conecte el programador al dispositivo para programarlo desde el entorno de Arduino, no se debe conectar la fuente de alimentación, es decir, sólo se conecta el programador al dispositivo, pero no se conecta el dispositivo a la fuente de alimentación, tal y como muestra la imagen anterior. Esto es porque mientras se programa el dispositivo, obtiene su alimentación a través del programador, que a su vez toma la alimentación del puerto USB del ordenador.

Cabe destacar que para la conexión del programador al dispositivo se debe usar la placa electrónica intermedia proporcionada. Los puntos de conexión del programador al dispositivo son los marcados en rojo en la siguiente figura:





Es importante conectar el programador al dispositivo con la orientación correcta (tal y como muestra la foto) ya que si se conecta al revés se pueden dañar los componentes tanto del programador como del dispositivo. Como se puede observar, el programador se conecta con los componentes hacia arriba (placa roja) y con el celofán hacia arriba (placa verde).

Una vez conectado el programador al ordenador se puede usar el entorno de programación Arduino para realizar cambios al software y reprogramar el dispositivo. Tanto el manejo del entorno Arduino como los fundamentos de programación en esta plataforma quedan fuera del alcance de este documento, pero se indican a continuación los pasos básicos para reprogramar el dispositivo:

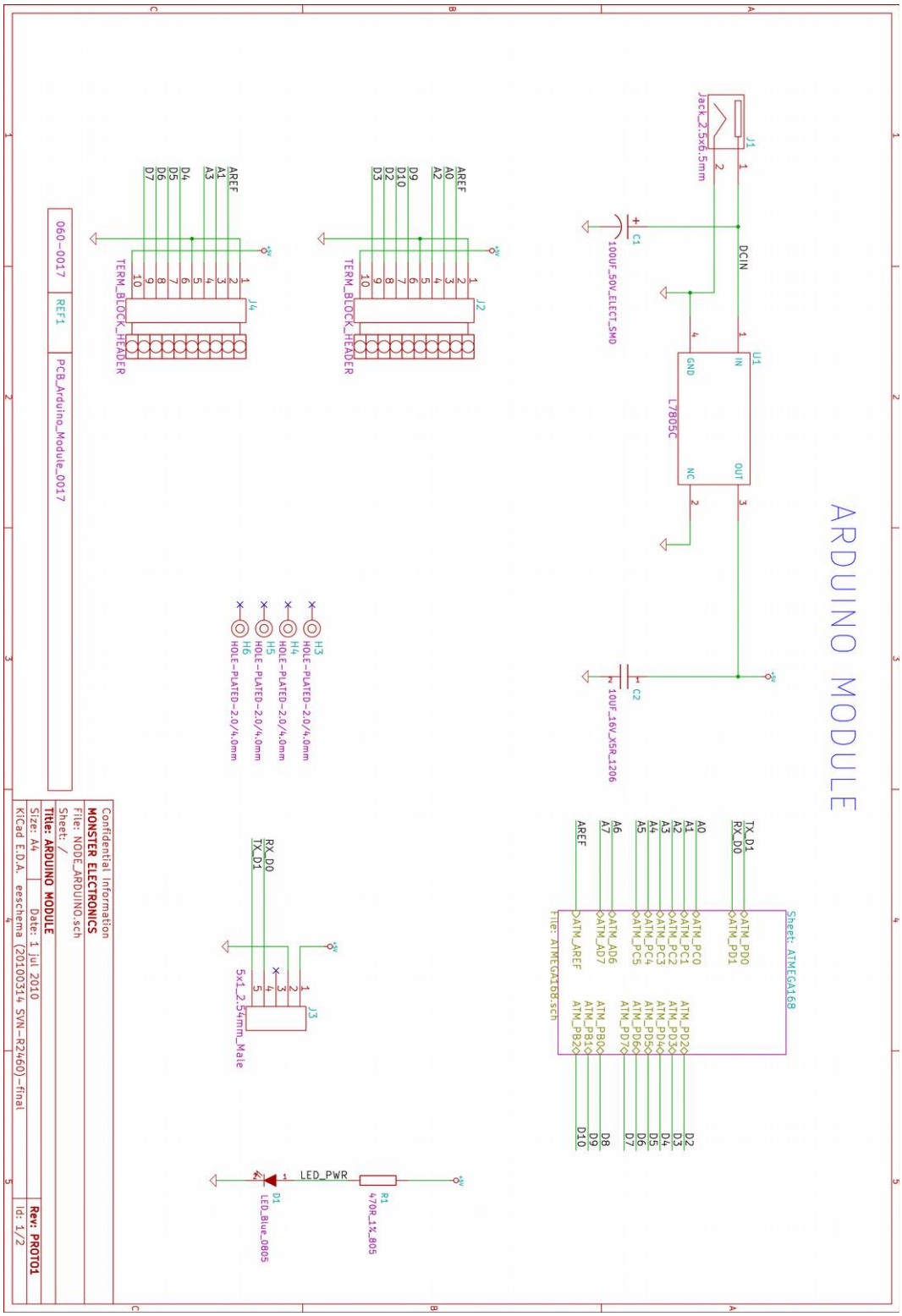
1. Se conecta el programador al ordenador. También se conecta el programador a la placa intermedia. Se conecta la placa intermedia al dispositivo microcontrolador.
2. Se inicia el entorno Arduino.
3. Se seleccionan el modelo de microcontrolador (Atmega328) y el puerto serie a utilizar. Nótese que el nombre del puerto serie es diferente para sistemas GNU/Linux, MacOS y Windows.
4. Se abre el fichero que contiene el código fuente y se realizan los cambios oportunos.
5. Se compila para verificar que el código fuente no contiene errores sintácticos.
6. Se reprograma el dispositivo mediante el botón “Upload”.

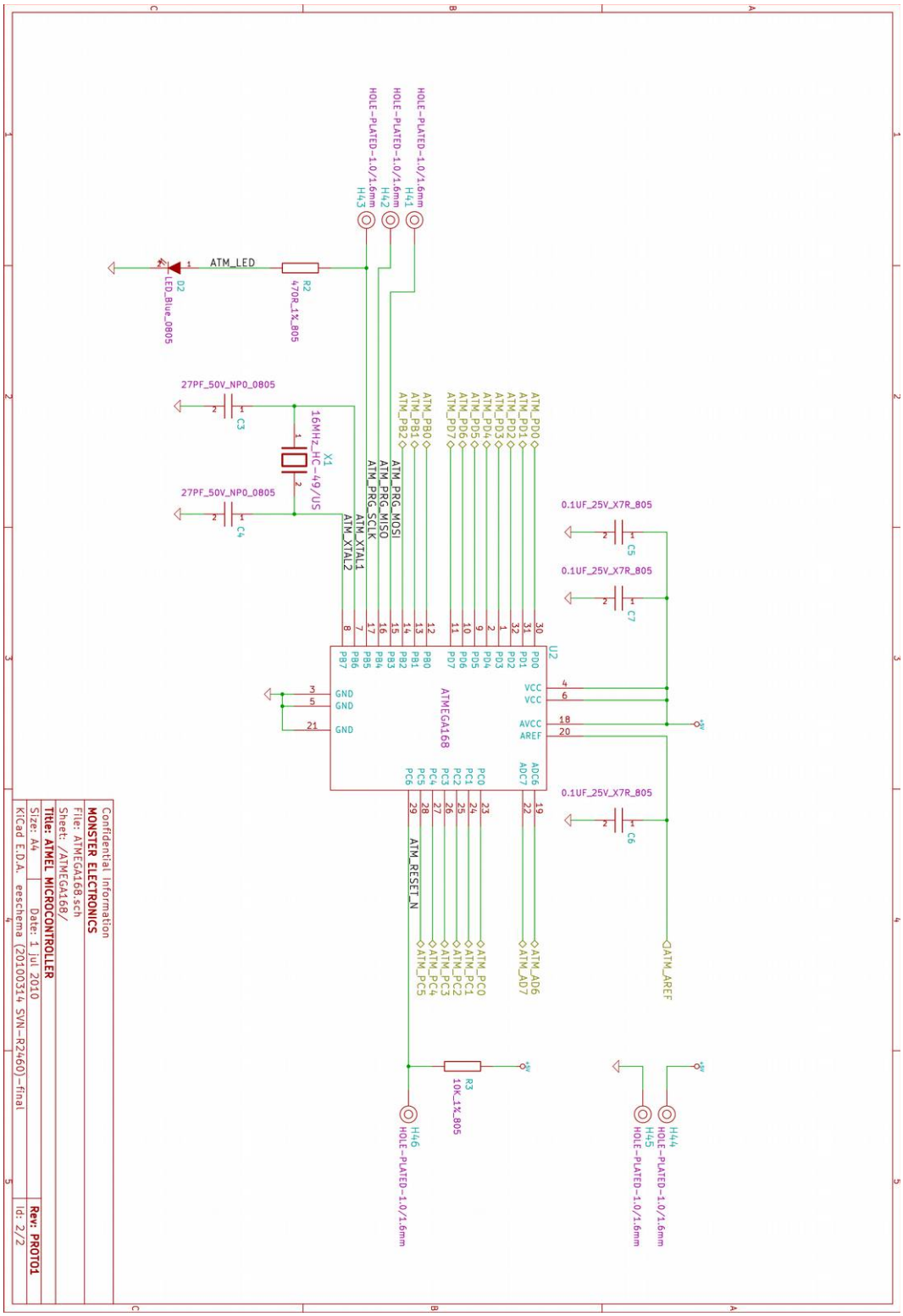
Consideraciones generales

No se debe utilizar una fuente de alimentación que provea menos de 9 Vdc ni más de 15 Vdc, ya que el dispositivo puede estropearse si se emplea con fuentes de alimentación cuyos valores de voltaje no se encuentren en este rango.
El conector para la fuente de alimentación lleva el voltaje (+) en el centro (*tip*) y la tierra (-) en el exterior (*ring*). Debajo de cada tarjeta Arduino aparece un gráfico que muestra esto mismo. Además el conector de alimentación necesario es un jack de 2.5mm de diámetro de punta (*tip*) y de 6.5mm de diámetro exterior (*ring*).
No se debe utilizar el dispositivo a temperaturas inferiores a -20ºC ni superiores a 50ºC.
No se debe utilizar el dispositivo en ambientes con una humedad relativa superior al 90%, así como conviene no colocarlo ni operarlo cerca de líquidos. El contacto con líquidos, además de estropear el dispositivo, puede causar descargas eléctricas perjudiciales para las personas.

Anexo técnico

Circuito microcontrolador (Arduino) A continuación se muestra el esquema electrónico de la placa que contiene el microcontrolador.




HOLE-PLATED-1.0/1.6mm
H41
H42
H43
ATM_LED
D2
LED_Blue_0805
R2
470R_1%_805
27PF_50V_NP0_0805
C3
C4
X1
16MHZ_HC-49/US
ATM_PRG_MOSI
ATM_PRG_MISO
ATM_PRG_SCLK
ATM_XTAL1
ATM_XTAL2
ATM_PB0
ATM_PB1
ATM_PB2
ATM_PD0
ATM_PD1
ATM_PD2
ATM_PD3
ATM_PD4
ATM_PD5
ATM_PD6
ATM_PD7
0.1UF_25V_X7R_805
C5
C7
C6
U2
ATMEGA168
GND
VCC
AVCC
AREF
ADC6
ADC7
ATM_PC0
ATM_PC1
ATM_PC2
ATM_PC3
ATM_PC4
ATM_PC5
ATM_RESET_N
ATM_AD6
ATM_AD7
ATM_AREF
R3
10K_1%_805
H46
H44
H45
Confidential Information
MONSTER ELECTRONICS
File: ATMEGA168.sch
Sheet: /ATMEGA168/
Title: ATMEL MICROCONTROLLER
Size: A4
Date: 1 jul 2010
Rev: PROTO1
KiCad E.D.A. eeschema (20100314 SVN-R2460)-final
Id: 2/2

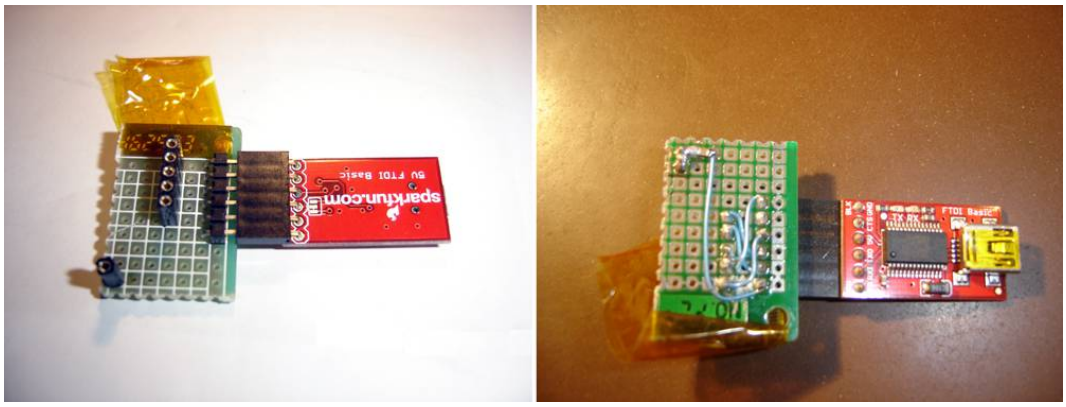
El listado de los materiales empleados para hacer la tarjeta es el siguiente:

Designator	Package	Quantity	Description
J4,J2	10-POS,TERM_BLOCK_HEADER	2	CON_TERMINAL_BLOCK_HEADER_10P_3.5MM-PITCH_SIDE_ENTRY_TH
X1	HC-49/US	1	XTAL_16MHZ_+/-50PPM_20PF_2-POS_HC-49/US_TH
R1,R2	SM0805	2	RES_FILM_470R_1/8W_1%_0805
C3,C4	SM0805	2	CAP_CER_27PF_50V_5%_NP0_0805
D2,D1	SM0805	2	LED_BLUE_CLEAR_70MCD_0805
R3	SM0805	1	RES_FILM_10K_1/8W_1%_0805
C1	EEE-FK-F	1	CAP_ELECT_100UF_50V_20%_SMD
C2	SM1206	1	CAP_CER_10UF_16V_20%_X5R_1206
C5,C6,C7	SM0805	3	CAP_CER_0.1UF_25V_10%_X7R_0805
U1	DPAK-3	1	REG_LDO_L7805C_35VIN_5VO_1.5A_DPAK-3
U2	TQFP-32	1	IC_MCU_ATMEGA168_16K_20MHZ_TQFP-32
H46,H45,H44,H43,H42,H41	HOLE-PLATED-1.0MM-1.6MM	6	HOLE_PLATED_1.0MM_DRILL_1.6MM_PAD
H3,H4,H5,H6	HOLE-PLATED-2.0MM-4.0MM	4	HOLE_PLATED_2.0MM_DRILL_4.0MM_PAD
J3	CON-HEADER-MALE-5X1-2.55MM	1	CON_HEADER_MALE_5X1P_2.54MM-PITCH_TH
J1	CON-JACK-MALE-2.5MMx6.5MM-TH	1	CON_MALE_JACK_2.5MM-IDX6.5MM-OD_TH
REF1	PCB_ARDUINO_MODULE_0017	1	PCB_ARDUINO_MODULE_0017

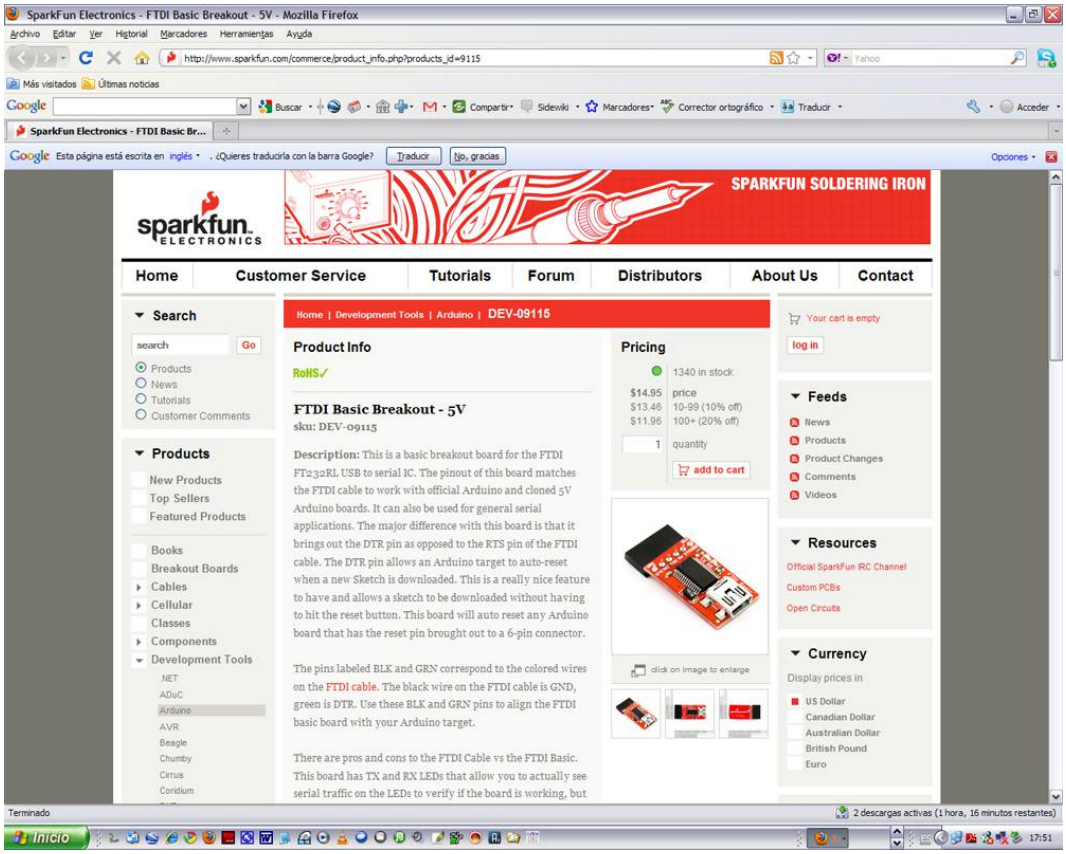
La columna "*Designator*" muestra los símbolos que aparecen en los esquemas anteriores para cada una de las partes electrónicas. La columna "*Package*" indica el tipo de paquete o encapsulado para cada parte. La columna "*Quantity*" indica la cantidad de cada parte que contiene cada tarjeta. Por último, la columna "*Description*" contiene una descripción breve para cada parte. Esta columna ofrece mucha información, ya que por ejemplo para una resistencia dice su valor, la potencia de funcionamiento, la tolerancia y el tipo de encapsulado. Información similar se muestra para el resto de las partes.

Circuito programador: placa Mini USB

Se trata del circuito comercial *5v FTDI Basic breakout*, fabricado y distribuido por Sparkfun Electronics. La documentación técnica para el mismo puede encontrarse en:



http://www.sparkfun.com/commerce/product_info.php?products_id=9115



Más información:

http://www.antonioalvarado.net/obras/2011/2011_001/ftdi_Basic.pdf

http://www.antonioalvarado.net/obras/2011/2011_001/ftdi_basica_breakout.pdf

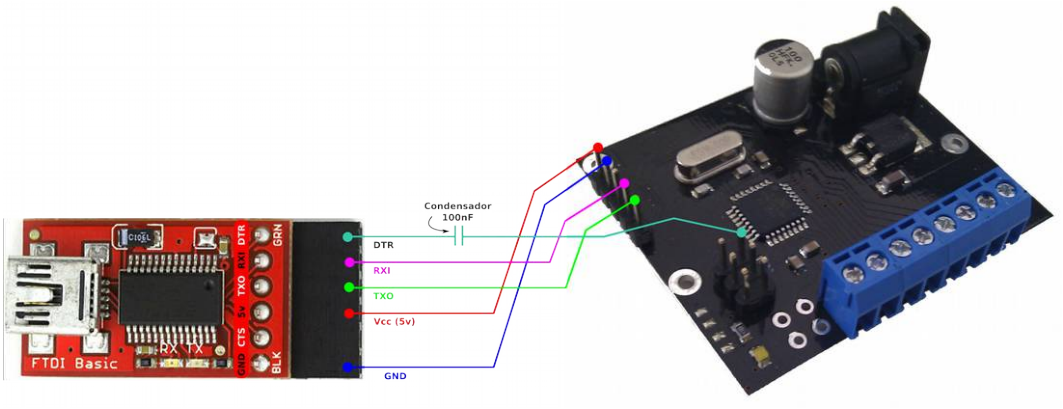
Como este circuito se basa en el chip FT232 de Ftdi Technologies, abrir el *driver* para su funcionamiento, en caso de que sea necesario. Este *driver* ya viene de serie con sistemas GNU/Linux, por lo que sólo se ofrecen las versiones para Windows (excepto Windows 98 y anteriores) y MacOS X. Para más información sobre el *driver* de Ftdi se puede consultar la página:

http://www.ftdichip.com/Drivers/VCP.htm

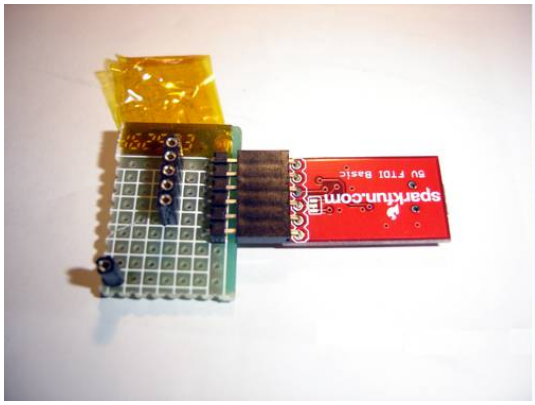
NOTA: Si se tiene Arduino instalado en un ordenador y ya se ha trabajado en alguna ocasión con él, no será necesario instalar este *driver*.

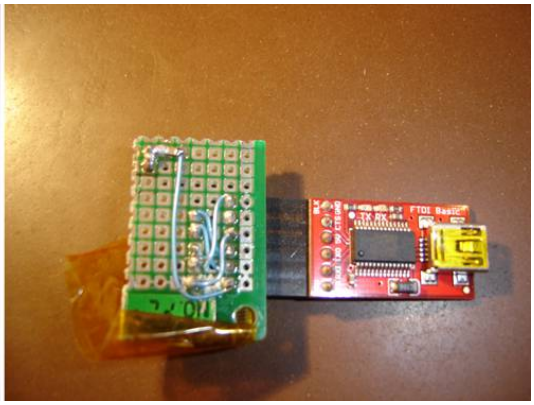
Circuito programador: placa intermedia

Esta placa está hecha a mano, y sirve para interconectar la placa de programación USB con el dispositivo microcontrolador. El diagrama de conexiones para esta placa es el siguiente:



Nótese que el pin DTR requiere de un condensador de 100 nF.





PIR Motion Detector Module

Para mas información sobre el modulo PIR:

http://www.antonioalvarado.net/obras/2011/2011_001/pir_Module.pdf