15 May 08

E3390 Electronic Circuits Design Lab Final Project Report

THE ÆTHER: A MINOR REVOLUTION IN FUN

Submitted in partial fulfillment of the requirements of the Bachelor of Science degree

David Albert

Nicholas Bergson-Shilcock

Spencer Russell

Dwight Tejano

Department of Electrical Engineering Columbia University

TABLE OF CONTENTS

EXECUTIVE SUMMARY

This deliciously tantalizing project aims to create a tabletop gaming system. This gaming system will be controlled solely by an innovative touchless player interface (the Æthersense) and will be able to support four players.

The games, built in Python, will be designed in conjunction to utilize the unique control scheme in a creative way. With an eventual aim at giving users the ability to create their own games through standard open-source software, this system, both versatile and different, would be a welcome addition to the ever-expanding realm of gaming.

The form factor of the Æther will be similar to the tabletop arcade games (also known as cocktail arcade machine) with players seated around a horizontal screen. A pair of Æthersense controllers will be on each edge of the tabletop system, and they will allow all four players to experience the excitement of competitive or of cooperative play (dependent, of course, on the game.)

DESIGN DETAILS

Being an embedded system at its core, there are two primary components to this project: hardware and software.

The physical hardware will be comprised of:

- eight USB Æthersense controllers
- an LCD television display
- PC with no moving parts (aside from processor cooling)

The controllers will be placed at toward the edges of the system (two on each side) with the display in the middle. Each controller will interface as a separate USB device operating over the HID protocol. This allows extended flexibility and an independently marketable product.

The total software package contains both the game software code and additional utilities to aid others in the development of the Æther.

All of our hardware schematics and our entire source code (under the GNU General Public License) will be released freely into the public domain.

DESIGN DETAILS (cont'd)

Specifications

PC

- Integrated video graphics card with 3D hardware acceleration
- 1.6 GHz Intel processor (minimum)
- 1 GB RAM (minimum)
- Integrated sound card
- CompactFlash to IDE converter
- CompactFlash card (at least 4 GB)
- 8 USB ports (integrated or via hub)

Æthersense

- MaxBotix EZ4 Ultrasonic Distance sensor
- ATtiny45 USB Microcontroller

Software

- Languages: C (AVR-USB Firmware), Python (Game software)
- Video: OpenGL

Design Target

The ultimate target for the Æther is illustrated with the block diagram on the next page. Our final design reached our lofty goals and can be seen in our final project.

Block Diagram

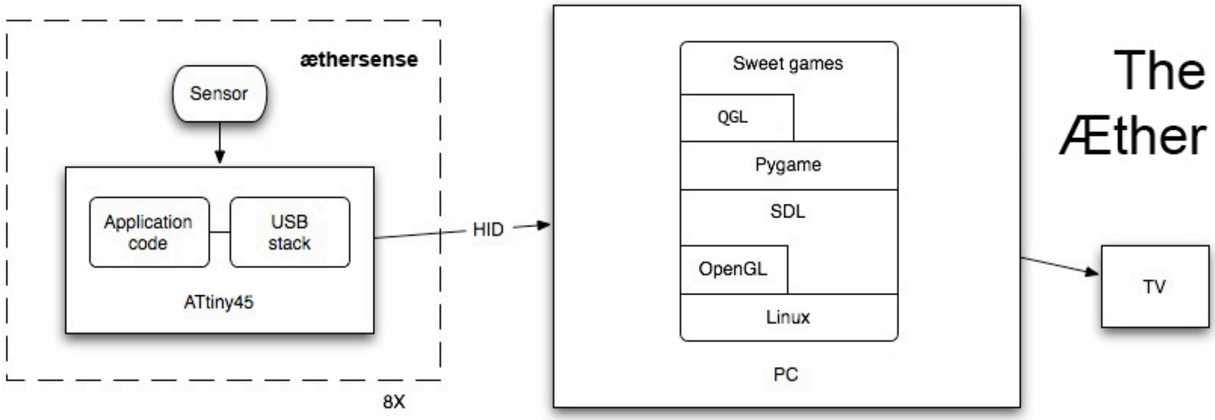


Figure 1. Block diagram of the Æther's components. The Æthersense connects via USB to the PC, and its visual output is sent to the TV.

Individual Block Descriptions

Sensor

The MaxBotix EZ4 Ultrasonic Distance Sensor (see datasheet in Appendix F.) Powered by the USB (+5V), the AVR-USB Microcontroller sends Rx high, which causes PW to go high and the distance sensor to send out a signal at 42 KHz. PW runs low when the echo returns to the sensor, and that time is used to calculate the distance. When PW is detected low, Rx runs high again, thereby repeating the process and retaking the distance.

Application Code

The Application Code (written in C) is the firmware for the AVR-USB Microcontroller. This code controls the pin that sends Rx high, which begins the entire distance measurement process. The code also calculates the distance by taking the echo time received by the sensor. Interfacing with the standard USB HID driver set, the application code sends the data via USB to the main board for use in gaming. (See Appendix A: Æthersense Firmware)

USB Stack

Entirely software-based stack, originally created by Christian Starkjohann, for use on the AVR Microcontrollers that allows full functionality of the USB HID driver set. (See Appendix A: AVR-USB Driver)

Linux

The PC is running Xubutu 8.04 (Hardy Heron), which uses an xfce-based desktop environment to facilitate faster speeds while still keeping a friendly UI.

Pygame

Pygame is Python wrapper for SDL designed for creating computer games in Python.

OpenGL

OpenGL is a cross-platform 2D and 3D drawing framework. A significant advantage of OpenGL is that it provides hardware acceleration. The PC has an Intel GMA 965 integrated graphics card, which has mature and reliable open-source drivers available for Linux.

SDL

The Simple DirectMedia Layer is cross-platform framework for creating multimedia applications such as games. It allows easy access to input and output devices and interfaces with OpenGL.

QGL

QGL is a Python module designed to be used with Pygame and OpenGL that provides basic scene graph (structure that provides a level of abstraction above the direct drawing commands of OpenGL) functionality. In the scene graph, each node represents an object in the scene.

Sweet games

You betcha.

Television

The Æther prototype uses a Dell W1900 19" LCD TV as its primary display, which runs at a native 1280 x 768 resolution. The Dell W1900 supports not only the standard composite and component connections, but it also supports VGA and DVI connections, making the switch over from a standard flat panel computer monitor to a television ultimately painless.

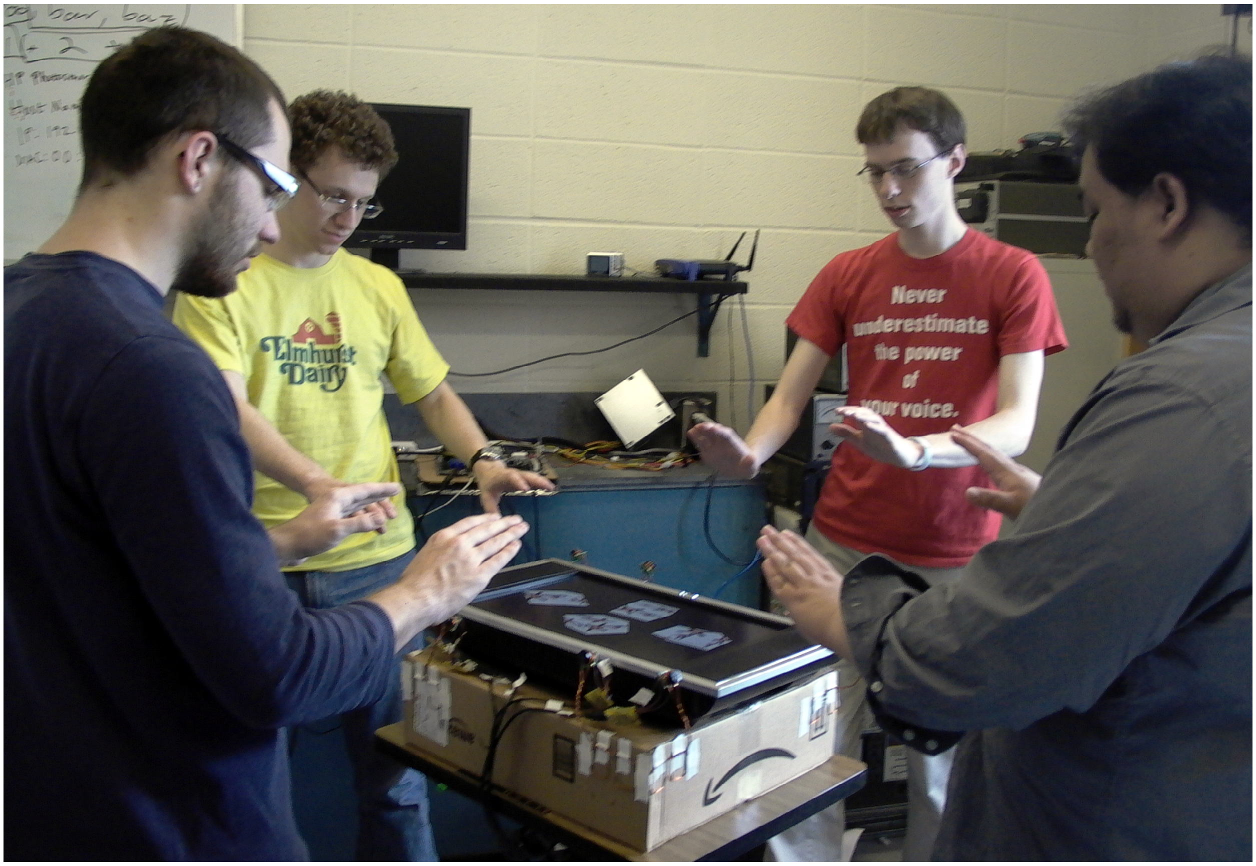


Figure 2. The luscious developers using the Æther.

BILL OF MATERIALS

Manufacturer	Item Description/Part Number	Quantity	Unit Price	Total
MaxBotix	LV-MaxSonar-EZ4 Sonar Range Finder	8	$19.95	$159.60
Atmel	ATtiny45 - 20PU	8	$1.83	$14.64
Amp	USB Type B Receptacle	8	$1.35	$10.80
Monoprice	USB A-to-B Cable	8	$0.79	$6.32
Intel	DG33TL Micro ATX Motherboard	1	$117.99	$117.99
Intel	1.6 GHz Celeron Conroe-L	1	$43.99	$43.99
Crucial	1 GB DDR2-667 RAM	1	$24.99	$24.99
Syba	SY-IDE2CF-DU CF-to-IDE UDMA Adapter	1	$13.09	$13.09
Transcend	8 GB CompactFlash Card	1	$87.99	$87.99
Adv. Circuits	Prototype Custom PCBs	8	$26.65	$213.20
Fairchild	1N747A 3.6 V Zener Diodes	16	$0.05	$0.80
	100 nF Capacitors	8		
	10 µF Capacitors	8		
	68 Ω Resistors	16		
	220 Ω Resistors	8		
	1.5K Ω Resistors	8		
	LEDs	8		
			TOTAL COST	$693.41

Table 1. Full table of materials used and their costs.

The final list of materials necessary to create the Æther are listed in Table 1 above. Note that all items without a specific manufacturer or unit cost were taken from the department's Student Projects Laboratory and would be negligible in cost to any person wishing to reproduce this project.

The display used in the prototype was a Dell W1900 LCD TV, and this was taken from the personal collection of one of the Æther's developers. Comparable televisions with VGA inputs cost approximately $400.

POTENTIAL HAZARDS

The following warnings are important to understand before the use of the Æther. If this product will be used by young children, this information should be read and explained to them by an adult. Failing to do so may cause injury or damage to property.

Product Dangers

Ultrasound Exposure

The Æther is based entirely on the use of an ultrasonic sensor as its controller interface. When operating nominally, the ultrasonic sensor emits approximately 125 pings per second at a frequency of 42 KHz. The Æther is designed to keep well within the guidelines of the maximum allowable sound power levels by law; however, this may cause problems with pets (with hearing sensitivity in the 40 KHz range) and people with hearing aids or other sound amplifiers.

Health Issues

Seizures

Some people (about 1 in 4000) may have seizures or blackouts triggered by light flashes or patterns, and this may occur while they are watching TV or playing video games, even if they have never had a seizure before.

Repetitive Motion Injuries and Eyestrain

Playing video games can make your muscles, joints, skin or eyes hurt. Due to the active nature of the Æther, physical issues such as tendinitis, carpal tunnel syndrome, skin irritation or eyestrain can occur.

Motion Sickness

Playing video games can cause motion sickness in some players. If any player feels dizzy or nauseous when playing video games, he or she should stop playing and rest.

Environmental Hazards

Electric Shock

Being an electrical device, there is the possibility of electrical shock that could cause serious injury or death. The Æther's internal components should not be modified, nor should the Æther or Æthersense be used immersed an electrically conductive environment, such as water.

APPENDICES

The follow pages are illustrate all of the code and all of the schemata that went into the development of this project.

A: ÆTHERSENSE FIRMWARE (REV. 25)

B: ÆTHERSENSE SCHEMATIC

C: "LANDERS" SOFTWARE CODE

D: TECH DEMO ("SHAPES") SOFTWARE CODE

E: ÆTHER UTILITY CODE

F: HISTORICAL NOTES AND DATASHEETS

For additional information regarding the Æther, the Æthersense, their development, and their developers, please consult http://projectiles.wikidot.com/ and http://www.lookmanohands.org/.

Appendix A: Æthersense Firmware (Rev. 25)

```
/****************************************************************************
 * main.c
 * aethersense firmware
 * the aethersense is a 1-axis distance sensor that acts
 * as an HID joystick
 * Author: Spencer Russell, based on work by Christian Starkjohann
 * Copyright: (c) 2006 by OBJECTIVE DEVELOPMENT Software GmbH
 * License: Proprietary, free under certain conditions. See Documentation.
 * ************************************************************************/

#include <avr/io.h>
#include <avr/wdt.h>
#include <avr/eeprom.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>
#include <util/delay.h>
#include <stdlib.h>

#include "usbdrv.h"
#include "oddebug.h"

/*
Pin assignment:
PB1 = measurement trigger
PB3 = pulse width input
PB4 = LED output (active high)

PB0, PB2 = USB data lines
*/

#define BIT_LED 4
#define BIT_TRIG 1
#define BIT_PW 3

#define FILTERLENGTH 2 /* length of the moving-average filter */
#define JUMP_THRESH 8000

#define UTIL_BIN4(x)          (uchar)((0##x & 01000)/64 + (0##x & 0100)/16 +
(0##x & 010)/4 + (0##x & 1))
#define UTIL_BIN8(hi, lo)    (uchar)(UTIL_BIN4(hi) * 16 + UTIL_BIN4(lo))

#ifndef NULL
#define NULL     ((void *)0)
#endif


static uchar     reportBuffer[2];     /* buffer for HID reports */
static uchar     idleRate;            /* in 4 ms units */

PROGMEM char
usbHidReportDescriptor[USB_CFG_HID_REPORT_DESCRIPTOR_LENGTH] = {
    0x05, 0x01,                    // USAGE_PAGE (Generic Desktop)
    0x15, 0x00,                    // LOGICAL_MINIMUM (0)
    0x09, 0x04,                    // USAGE (Joystick)
    0xa1, 0x01,                    // COLLECTION (Application)
    0x05, 0x01,                    //   USAGE_PAGE (Generic Desktop)
    0x09, 0x01,                    //   USAGE (Pointer)
    0xa1, 0x00,                    //   COLLECTION (Physical)
    0x09, 0x31,                    //     USAGE (Y)
    0x27, 0xff, 0xff, 0x00, 0x00,  //     LOGICAL_MAXIMUM (65535)
    0x15, 0x00,                    //     LOGICAL_MINIMUM (0)
    0x75, 0x10,                    //     REPORT_SIZE (16)
    0x95, 0x01,                    //     REPORT_COUNT (1)
    0x81, 0x02,                    //     INPUT (Data,Var,Abs)
    0xc0,                          //   END_COLLECTION
    0xc0                           // END_COLLECTION
};


/*
 * Report Format:
 *
 * BYTE0    BYTE1
 * YYYYYYYY-----BYY
 * 76543210          -----098
 * y - axis value 0-1023
 * B - button 0-1
 */
static void buildReport(unsigned int value)
{
    reportBuffer[0] = (uchar)(value & 0xFF);
    reportBuffer[1] = (uchar)(value >> 8);
}

/*
 * measures the time it takes for an echo to return.
 * conversion factor: 0.1728 m/ms
 * 7ms = 57750 clock tics (at 8.25 MHz)
 * we'll just wait 65536 clock tics (~ 8ms)
 */
unsigned int getdistance()
{
            unsigned long int distance = 0;
            unsigned int overflow_count = 0;
            static unsigned int measurements[FILTERLENGTH];
            static unsigned int current_index = 0;
            static unsigned int last_measurement = 0;
            unsigned int current_measurement;
            int i;

            PORTB |= 1 << BIT_TRIG; /* trigger the measurement */
            while(!(PINB & (1 << BIT_PW))) {} /* wait until the PW pin goes high
*/
            TCNT1 = 0; /* reset counter */
            TIFR = (1 << TOV1); /* clear overflow if set */
            PORTB &= ~(1 << BIT_TRIG); /* bring the trigger pin low again */
            do
            {
                        if(TIFR & (1 << TOV1))
                        {
                                    TIFR = (1 << TOV1); /* clear overflow */
                                    overflow_count++;
                        }
            } while(overflow_count <= 255 && (PINB & (1 << BIT_PW)));
            current_measurement = (256 * overflow_count + TCNT1);

            /* get rid of single-sample outliers or timed-out samples */
            if(abs(current_measurement - last_measurement) < JUMP_THRESH &&
overflow_count < 250)
            {
                        measurements[current_index] = current_measurement;
                        current_index = (current_index + 1) % FILTERLENGTH;

                        /* moving average filter */
                        for(i = 0; i < FILTERLENGTH; i++)
                                    distance += measurements[i];
                        distance /= FILTERLENGTH;
            }
            else
                        distance = 0;
            last_measurement = current_measurement;
            return distance;
}


static void timerInit(void)
{
    TCCR1 = UTIL_BIN8(0000, 0010);             /* timer clock = clock/2,
8.25MHz*/
}

/* ------------------------------------------------------------------------- */
/* ------------------------ interface to USB driver ------------------------ */
/* ------------------------------------------------------------------------- */

uchar        usbFunctionSetup(uchar data[8])
```

```
{
usbRequest_t     *rq = (void *)data;

    usbMsgPtr = reportBuffer;
    if((rq->bmRequestType & USBRQ_TYPE_MASK) == USBRQ_TYPE_CLASS)
              {    /* class request type */
        if(rq->bRequest == USBRQ_HID_GET_REPORT)
                              {  /* wValue: ReportType (highbyte), ReportID (lowbyte)
*/
            /* we only have one report type, so don't look at wValue */
            buildReport(0);
            return sizeof(reportBuffer);
        }
                              else if(rq->bRequest == USBRQ_HID_GET_IDLE)
                              {
            usbMsgPtr = &idleRate;
            return 1;
        }
                              else if(rq->bRequest == USBRQ_HID_SET_IDLE)
                              {
            idleRate = rq->wValue.bytes[1];
        }
    }
              else
              {
        /* no vendor specific requests implemented */
    }
              return 0;
}

/* ------------------------------------------------------------------------- */
/* --------------------------------- main ---------------------------------- */
/* ------------------------------------------------------------------------- */

int main(void)
{
int i;
int valPending = 0;
unsigned int distance = 0;

/* Calibrate the RC oscillator to 8.25 MHz. The core clock of 16.5 MHz is
 * derived from the 66 MHz peripheral clock by dividing. We assume that the
 * EEPROM contains a calibration value in location 0. If no calibration value
 * has been stored during programming, we offset Atmel's 8 MHz calibration
 * value according to the clock vs OSCCAL diagram in the data sheet. This
 * seems to be sufficiently precise (<= 1%).
 */
    uchar calibrationValue = eeprom_read_byte(0);
    if(calibrationValue != 0xff)
              {
        OSCCAL = calibrationValue;  /* a calibration value is supplied */
    }
              else
              {
        /* we have no calibration value, assume 8 MHz calibration and adjust from
there */
```

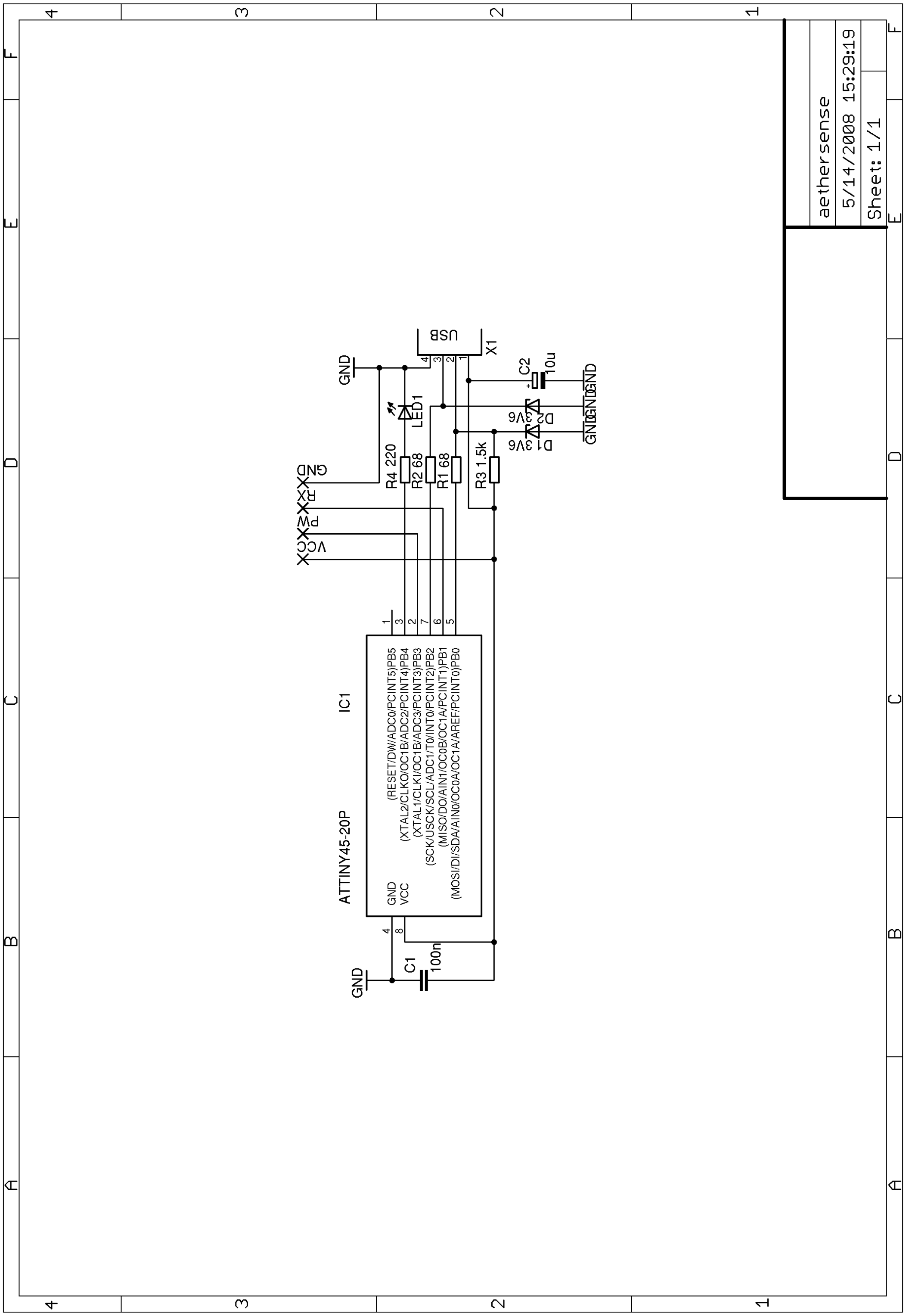
```
        if(OSCCAL < 125)
                              {
            OSCCAL += 3;     /* should be 3.5 */
        }
                              else if(OSCCAL >= 128)
                              {
            OSCCAL += 7;     /* should be 7 */
        }
                              else
                              {  /* must be between 125 and 128 */
            OSCCAL = 127;    /* maximum possible avoiding discontinuity */
        }
    }
    odDebugInit();
    DDRB = (1 << USB_CFG_DMINUS_BIT) | (1 << USB_CFG_DPLUS_BIT);
    PORTB = 0;            /* indicate USB disconnect to host */
    for(i=0;i<20;i++)
              {  /* 300 ms disconnect, also allows our oscillator to stabilize */
        _delay_ms(15);
    }
    DDRB = 1 << BIT_LED | 1 << BIT_TRIG;     /* output for LED and measurement
trigger */
    wdt_enable(WDTO_1S);
    timerInit();
    usbInit();
    sei();
    while(1)
              {     /* main event loop */
        wdt_reset();
        usbPoll();
                              /* if a new value is ready and the last value was sent */
        if(valPending && usbInterruptIsReady())
                              {
                                          buildReport(distance);
            usbSetInterrupt(reportBuffer, sizeof(reportBuffer));
                                          valPending = 0;
                                          PORTB &= ~(1 << BIT_LED);    /* turn off
LED */
        }
                              /* if the last measurement has been handed to the USB
driver */
                              if(!valPending)
                              {
                                          distance = getdistance();
                                          if(distance) /* distance returns 0 for outliers
*/
                                          {
                                                      PORTB |= 1 << BIT_LED;    /*
turn on LED */
                                                      valPending = 1;
                                          }
                              }
    }
    return 0;
}
```

AVR-USB DRIVER

```
/* Name: usbconfig.h
 * Project: AVR USB driver
 * Author: Christian Starkjohann
 * Creation Date: 2007-06-23
 * Tabsize: 4
 * Copyright: (c) 2007 by OBJECTIVE DEVELOPMENT Software GmbH
 * License: GNU GPL v2 (see License.txt) or proprietary (CommercialLicense.txt)
 * This Revision: $Id: usbconfig.h 362 2007-06-25 14:38:21Z cs $
 */

#ifndef __usbconfig_h_included__
#define __usbconfig_h_included__

/* ---------------------------- Hardware Config ---------------------------- */

#define USB_CFG_IOPORTNAME        B
/* This is the port where the USB bus is connected. When you configure it to
 * "B", the registers PORTB, PINB and DDRB will be used.
 */
#define USB_CFG_DMINUS_BIT        0
/* This is the bit number in USB_CFG_IOPORT where the USB D- line is connected.
 * This may be any bit in the port.
 */
#define USB_CFG_DPLUS_BIT         2
/* This is the bit number in USB_CFG_IOPORT where the USB D+ line is connected.
 * This may be any bit in the port. Please note that D+ must also be connected
 * to interrupt pin INT0!
 */
#define USB_CFG_CLOCK_KHZ         (F_CPU/1000)
/* Clock rate of the AVR in MHz. Legal values are 12000, 16000 or 16500.
 * The 16.5 MHz version of the code requires no crystal, it tolerates +/- 1%
 * deviation from the nominal frequency. All other rates require a precision
 * of 2000 ppm and thus a crystal!
 * Default if not specified: 12 MHz
 */

/* ----------------------- Optional Hardware Config ------------------------ */

/* #define USB_CFG_PULLUP_IOPORTNAME    D */
/* If you connect the 1.5k pullup resistor from D- to a port pin instead of
 * V+, you can connect and disconnect the device from firmware by calling
 * the macros usbDeviceConnect() and usbDeviceDisconnect() (see usbdrv.h).
 * This constant defines the port on which the pullup resistor is connected.
 */
/* #define USB_CFG_PULLUP_BIT           4 */
/* This constant defines the bit number in USB_CFG_PULLUP_IOPORT (defined
 * above) where the 1.5k pullup resistor is connected. See description
 * above for details.
 */

/* --------------------------- Functional Range ---------------------------- */

#define USB_CFG_HAVE_INTRIN_ENDPOINT     1
/* Define this to 1 if you want to compile a version with two endpoints: The
 * default control endpoint 0 and an interrupt-in endpoint 1.
 */
#define USB_CFG_HAVE_INTRIN_ENDPOINT3    0
/* Define this to 1 if you want to compile a version with three endpoints: The
 * default control endpoint 0, an interrupt-in endpoint 1 and an interrupt-in
 * endpoint 3. You must also enable endpoint 1 above.
 */
#define USB_CFG_IMPLEMENT_HALT           0
/* Define this to 1 if you also want to implement the ENDPOINT_HALT feature
 * for endpoint 1 (interrupt endpoint). Although you may not need this feature,
 * it is required by the standard. We have made it a config option because it
 * bloats the code considerably.
 */
#define USB_CFG_INTR_POLL_INTERVAL       10
/* If you compile a version with endpoint 1 (interrupt-in), this is the poll
 * interval. The value is in milliseconds and must not be less than 10 ms for
 * low speed devices.
 */
#define USB_CFG_IS_SELF_POWERED          0
/* Define this to 1 if the device has its own power supply. Set it to 0 if the
 * device is powered from the USB bus.
 */
#define USB_CFG_MAX_BUS_POWER            50
/* Set this variable to the maximum USB bus power consumption of your device.
 * The value is in milliamperes. [It will be divided by two since USB
 * communicates power requirements in units of 2 mA.]
 */
#define USB_CFG_IMPLEMENT_FN_WRITE       0
/* Set this to 1 if you want usbFunctionWrite() to be called for control-out
 * transfers. Set it to 0 if you don't need it and want to save a couple of
 * bytes.
 */
#define USB_CFG_IMPLEMENT_FN_READ        0
/* Set this to 1 if you need to send control replies which are generated
 * "on the fly" when usbFunctionRead() is called. If you only want to send
 * data from a static buffer, set it to 0 and return the data from
 * usbFunctionSetup(). This saves a couple of bytes.
 */
#define USB_CFG_IMPLEMENT_FN_WRITEOUT    0
/* Define this to 1 if you want to use interrupt-out (or bulk out) endpoint 1.
 * You must implement the function usbFunctionWriteOut() which receives all
 * interrupt/bulk data sent to endpoint 1.
 */
#define USB_CFG_HAVE_FLOWCONTROL         0
/* Define this to 1 if you want flowcontrol over USB data. See the definition
 * of the macros usbDisableAllRequests() and usbEnableAllRequests() in
 * usbdrv.h.
 */

/* -------------------------- Device Description --------------------------- */

#define  USB_CFG_VENDOR_ID       0x42, 0x42
/* USB vendor ID for the device, low byte first. If you have registered your
 * own Vendor ID, define it here. Otherwise you use obdev's free shared
 * VID/PID pair. Be sure to read USBID-License.txt for rules!
 * This template uses obdev's shared VID/PID pair for HIDs: 0x16c0/0x5df.
 * Use this VID/PID pair ONLY if you understand the implications!
 */
#define  USB_CFG_DEVICE_ID       0x31, 0xe1
/* This is the ID of the product, low byte first. It is interpreted in the
 * scope of the vendor ID. If you have registered your own VID with usb.org
 * or if you have licensed a PID from somebody else, define it here. Otherwise
 * you use obdev's free shared VID/PID pair. Be sure to read the rules in
 * USBID-License.txt!
 * This template uses obdev's shared VID/PID pair for HIDs: 0x16c0/0x5df.
 * Use this VID/PID pair ONLY if you understand the implications!
 */
#define USB_CFG_DEVICE_VERSION  0x00, 0x01
/* Version number of the device: Minor number first, then major number.
 */
#define USB_CFG_VENDOR_NAME     'P','r','o','j','e','c','t','i','l','e','s'
#define USB_CFG_VENDOR_NAME_LEN 11
/* These two values define the vendor name returned by the USB device. The
name
 * must be given as a list of characters under single quotes. The characters
 * are interpreted as Unicode (UTF-16) entities.
 * If you don't want a vendor name string, undefine these macros.
 * ALWAYS define a vendor name containing your Internet domain name if you use
 * obdev's free shared VID/PID pair. See the file USBID-License.txt for
 * details.
 */
#define USB_CFG_DEVICE_NAME     'A','e','t','h','e','r','s','e','n','s','e'
#define USB_CFG_DEVICE_NAME_LEN 11
/* Same as above for the device name. If you don't want a device name, undefine
 * the macros. See the file USBID-License.txt before you assign a name if you
 * use a shared VID/PID.
 */
#define USB_CFG_SERIAL_NUMBER   '0', '0', '0'
#define USB_CFG_SERIAL_NUMBER_LEN   3
/* Same as above for the serial number. If you don't want a serial number,
 * undefine the macros.
 * It may be useful to provide the serial number through other means than at
 * compile time. See the section about descriptor properties below for how
 * to fine tune control over USB descriptors such as the string descriptor
 * for the serial number.
 */
```

```
#define USB_CFG_DEVICE_CLASS          0
#define USB_CFG_DEVICE_SUBCLASS       0
/* See USB specification if you want to conform to an existing device class.
 */
#define USB_CFG_INTERFACE_CLASS       3    /* HID */
#define USB_CFG_INTERFACE_SUBCLASS   0    /* no boot interface */
#define USB_CFG_INTERFACE_PROTOCOL   0    /* no protocol */
/* See USB specification if you want to conform to an existing device class or
 * protocol.
 */
#define USB_CFG_HID_REPORT_DESCRIPTOR_LENGTH      31 /* total length of
report descriptor */
/* Define this to the length of the HID report descriptor, if you implement
 * an HID device. Otherwise don't define it or define it to 0.
 * Since this template defines a HID device, it must also specify a HID
 * report descriptor length. You must add a PROGMEM character array named
 * "usbHidReportDescriptor" to your code which contains the report descriptor.
 * Don't forget to keep the array and this define in sync!
 */

/* #define USB_PUBLIC static */
/* Use the define above if you #include usbdrv.c instead of linking against it.
 * This technique saves a couple of bytes in flash memory.
 */

/* ------------------- Fine Control over USB Descriptors ------------------- */
/* If you don't want to use the driver's default USB descriptors, you can
 * provide our own. These can be provided as (1) fixed length static data in
 * flash memory, (2) fixed length static data in RAM or (3) dynamically at
 * runtime in the function usbFunctionDescriptor(). See usbdrv.h for more
 * information about this function.
 * Descriptor handling is configured through the descriptor's properties. If
 * no properties are defined or if they are 0, the default descriptor is used.
 * Possible properties are:
 *   + USB_PROP_IS_DYNAMIC: The data for the descriptor should be fetched
 *     at runtime via usbFunctionDescriptor().
 *   + USB_PROP_IS_RAM: The data returned by usbFunctionDescriptor() or found
 *     in static memory is in RAM, not in flash memory.
 *   + USB_PROP_LENGTH(len): If the data is in static memory (RAM or flash),
 *     the driver must know the descriptor's length. The descriptor itself is
 *     found at the address of a well known identifier (see below).
 * List of static descriptor names (must be declared PROGMEM if in flash):
 *   char usbDescriptorDevice[];
 *   char usbDescriptorConfiguration[];
 *   char usbDescriptorHidReport[];
 *   char usbDescriptorString0[];
 *   int usbDescriptorStringVendor[];
 *   int usbDescriptorStringDevice[];
 *   int usbDescriptorStringSerialNumber[];
 * Other descriptors can't be provided statically, they must be provided
 * dynamically at runtime.
 *
 * Descriptor properties are or-ed or added together, e.g.:
 * #define USB_CFG_DESCR_PROPS_DEVICE   (USB_PROP_IS_RAM |
USB_PROP_LENGTH(18))
 *
 * The following descriptors are defined:
 *   USB_CFG_DESCR_PROPS_DEVICE
 *   USB_CFG_DESCR_PROPS_CONFIGURATION
 *   USB_CFG_DESCR_PROPS_STRINGS
 *   USB_CFG_DESCR_PROPS_STRING_0
 *   USB_CFG_DESCR_PROPS_STRING_VENDOR
 *   USB_CFG_DESCR_PROPS_STRING_PRODUCT
 *   USB_CFG_DESCR_PROPS_STRING_SERIAL_NUMBER
 *   USB_CFG_DESCR_PROPS_HID
 *   USB_CFG_DESCR_PROPS_HID_REPORT
 *   USB_CFG_DESCR_PROPS_UNKNOWN (for all descriptors not handled by the
driver)
 *
 */

#define USB_CFG_DESCR_PROPS_DEVICE                  0
#define USB_CFG_DESCR_PROPS_CONFIGURATION           0
#define USB_CFG_DESCR_PROPS_STRINGS                 0
#define USB_CFG_DESCR_PROPS_STRING_0                0
#define USB_CFG_DESCR_PROPS_STRING_VENDOR           0
#define USB_CFG_DESCR_PROPS_STRING_PRODUCT          0
#define USB_CFG_DESCR_PROPS_STRING_SERIAL_NUMBER    0
#define USB_CFG_DESCR_PROPS_HID                     0
#define USB_CFG_DESCR_PROPS_HID_REPORT              0
#define USB_CFG_DESCR_PROPS_UNKNOWN                 0

/* ----------------------- Optional MCU Description ------------------------ */

/* The following configurations have working defaults in usbdrv.h. You
 * usually don't need to set them explicitly. Only if you want to run
 * the driver on a device which is not yet supported or with a compiler
 * which is not fully supported (such as IAR C) or if you use a differnt
 * interrupt than INT0, you may have to define some of these.
 */
/* #define USB_INTR_CFG            MCUCR */
/* #define USB_INTR_CFG_SET        ((1 << ISC00) | (1 << ISC01)) */
/* #define USB_INTR_CFG_CLR        0 */
/* #define USB_INTR_ENABLE         GIMSK */
/* #define USB_INTR_ENABLE_BIT     INT0 */
/* #define USB_INTR_PENDING        GIFR */
/* #define USB_INTR_PENDING_BIT    INTF0 */

#endif /* __usbconfig_h_included__ */
```

APPENDIX B: ÆTHERSENSE SCHEMATIC


aethersense
5/14/2008 15:29:19
Sheet: 1/1
ATTINY45-20P
IC1
GND
VCC
(RESET/DW/ADC0/PCINT5)PB5
(XTAL2/CLKO/OC1B/ADC2/PCINT4)PB4
(XTAL1/CLKI/OC1B/ADC3/PCINT3)PB3
(SCK/USCK/SCL/ADC1/T0/INT0/PCINT2)PB2
(MISO/DO/AIN1/OC0B/OC1A/PCINT1)PB1
(MOSI/DI/SDA/AIN0/OC0A/OC1A/AREF/PCINT0)PB0
C1
100n
R4 220
R2 68
R1 68
R3 1,5k
LED1
D1 3V6
D2 3V6
C2
10u
X1
USB
GND
RX
PW
VCC
1
2
3
4
A
B
C
D
E
F

Appendix C: “Lander” Software Code

```
"""" Landers for the Aether
Designed by David Albert, Nicholas Bergson-Shilcock, Spencer Russell,
and Dwight Tejano.

Landers is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see <http://www.gnu.org/licenses/>.
"""

import qgl, pygame, sys, qgle, math, random
from pygame.locals import *
import game


GRAVITY = 0.42
CRASH_VEL = 0.6
active_players = 0
winner = None


class Player:
    def __init__(self, player_num, left_js, right_js,
                 mesh="newbox.obj", texture="prglogo-sign.jpg"):
        self.player_num = player_num
        self.team = None
        self.h_acc = 0
        self.h_vel = 0
        self.h_pos = 0
        self.v_acc = 0
        self.v_vel = 0
        self.v_pos = 0

        self.ground = -118
        self.crashed = False
        self.target_left = -80
        self.target_right = 80
        self.left_wall = -150
        self.right_wall = 150

        self.group = qgl.scene.Group()
        self.group.axis = (1,1,0)
        self.group.translate = (0,0,-300)
        self.model = qgl.scene.state.Mesh(mesh)
        self.texture = qgl.scene.state.Texture(texture)

        self.light = qgl.scene.state.Light(position=(0,0,80))
        self.left_js = pygame.joystick.Joystick(left_js)
        self.left_js.init()
        self.right_js = pygame.joystick.Joystick(right_js)
        self.right_js.init()

    def update(self):
        global active_players, winner, CRASH_VEL, GRAVITY
        if not self.crashed:
            if self.v_pos - self.ground > 236:
                coef = 0
            else:
                coef = math.sqrt(1 - (self.v_pos-self.ground)/236)

            self.v_acc = (2 - self.right_js.get_axis(0) -
self.left_js.get_axis(0))/7*coef
            self.h_acc = (self.right_js.get_axis(0) -
self.left_js.get_axis(0))/20
            self.axis = (0,0,1)
            self.group.angle = (self.right_js.get_axis(0)-
self.left_js.get_axis(0))*5
            self.v_vel += self.v_acc - GRAVITY
            self.h_vel += self.h_acc
            self.v_pos += self.v_vel
            self.h_pos += self.h_vel

            if self.h_pos < self.left_wall or self.h_pos > self.right_wall:
                self.h_vel = self.h_vel*-.8

            self.group.translate = (self.h_pos,self.v_pos,-300)

        if self.v_pos <= self.ground:
            if self.v_vel <= - CRASH_VEL:
                self.crashed = True
                active_players -= 1
            elif self.h_pos < self.target_right and self.h_pos >
self.target_left:
                winner = self
            self.v_vel = self.v_vel*-.9
            self.h_vel = self.h_vel*.7


def convert_to_deg(n):
    """ Converts -1 <= n <= 1 to degree (0 <= deg <= 360) """
    return n*180 + 180

def convert_to_0_1(n):
    """ Converts -1 <= n <= 1 to 0 <= n <= 1 """
    return (n+1)/2

def main():
    global active_players, winner

    g = game.Game()

    active_players = 1

    players = []
    players.append(Player(0, 0, 1))
    #players.append(Player(1, 2, 3))
    #players.append(Player(2, 4, 5))
    #players.append(Player(3, 6, 7))

    fixed_light = qgl.scene.state.Light(position=(0,0,0))
    fixed_light.diffuse = (0.6,0.6,0.6,1.0)

    for p in players:
        g.group.add(p.group)
        p.group.add(p.light, p.texture, p.model)

    g.root_node.accept(g.compiler)

    clock = pygame.time.Clock()

    max_vel = 0.12
    acc_coef = 0.003

    while winner == None and active_players > 0:
        for event in pygame.event.get():
            if event.type == pygame.QUIT: sys.exit(0)
            if hasattr(event, 'key'):
                if event.key == K_ESCAPE: sys.exit(0)

        for p in players:
            p.update()

        g.draw()
        clock.tick(60)

    if winner:
        print "Player %s won!" % (winner.player_num+1)
    else:
        print "Game over"
    print "Final vertical velocity: %s" % players[0].v_vel

if __name__ == '__main__': main()
```

Appendix D: "Shapes" Software Code

```
"""" Shapes for the Aether
Designed by David Albert, Nicholas Bergson-Shilcock, Spencer Russell,
and Dwight Tejano.

Shapes is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see <http://www.gnu.org/licenses/>.
"""

import qgl, pygame, sys, qgle, math
from pygame.locals import *
import random
import game

max_vel = 0.12 # maximum velocity
acc_coef = 0.005
winning_score = 500 # first player to get 500 point wins

class Team:
    def __init__(self, team_num):
        self.team_num = team_num
        self.score = 0
        self.members = []
        self.goal_color = (random(), random())

    def add_member(self, member):
        member.team = self
        self.members.append(member)

class Player:
    def __init__(self, player_num, left_js, right_js, parent,
                 translate=(0,0,0), axis=(0,0,0)):
        self.player_num = player_num
        self.team = None
        self.score = 0
        self.disturbed = False
        self.acc = 0
        self.vel = 0
        self.angle = 0

        self.left_js = pygame.joystick.Joystick(left_js)
        self.left_js.init()
        self.right_js = pygame.joystick.Joystick(right_js)
        self.right_js.init()

        self.model_group = qgl.scene.Group()
        self.model = qgl.scene.state.Mesh("newbox.obj")
        self.texture = qgl.scene.state.Texture("prglogo-sign.jpg")
        self.model_group.axis = axis
        self.light = qgl.scene.state.Light(position=(0,10,80))
        self.model_group.add(self.light, self.texture, self.model)
        self.model_group.translate = translate

        self.score_group = qgl.scene.Group()
        self.score_text = qgl.scene.state.Text("Score", "mono.ttf")
        self.score_group.add(self.score_text)
        self.score_group.axis = (0,0,1)
        if player_num == 0:
            self.score_group.translate = (-2.68, -2.05, -5)
            self.score_group.scale = (0.13, 0.15, 0.425)
        if player_num == 1:
            self.score_group.angle = 90
            self.score_group.translate = (2.75, -2, -5)
            self.score_group.scale = (0.15, 0.15, 0.425)
        if player_num == 2:
            self.score_group.angle = 180
            self.score_group.translate = (2.68, 2.05, -5)
            self.score_group.scale = (0.13, 0.15, 0.425)
        if player_num == 3:
            self.score_group.angle = 270
            self.score_group.translate = (-2.75, 2, -5)
            self.score_group.scale = (0.15, 0.15, 0.425)

        parent.add(self.model_group, self.score_group)

    def update(self):
        global acc_coef, max_vel
        self.acc = (self.left_js.get_axis(0)+1)*acc_coef -
(self.right_js.get_axis(0)+1)*acc_coef
        self.vel += self.acc
        if self.vel < -max_vel: self.vel = -max_vel
        elif self.vel > max_vel: self.vel = max_vel
        self.angle += self.vel
        self.model_group.angle = convert_to_deg(self.angle)
        self.score_text.text = str(int(math.floor(self.score/100)))

def convert_to_deg(n):
    """ Converts -1 <= n <= 1 to degree (0 <= deg <= 360) """
    return n*180 + 180

def main():
    g = game.Game()

    distance = -80

    players = []
    players.append(Player(0, 0, 1, g.group, translate=(0,-13.5,distance),
axis=(0,1,0)))
    players.append(Player(1, 3, 2, g.group, translate=(25.5,0,distance),
axis=(1,0,0)))
    players.append(Player(2, 5, 4, g.group, translate=(0,13.5,distance),
axis=(0,1,0)))
    players.append(Player(3, 6, 7, g.group, translate=(-25.5,0,distance),
axis=(1,0,0)))

    fixed_light = qgl.scene.state.Light(position=(0,10,80))
    fixed_light.diffuse = (0.6,0.6,0.6,1.0)

    g.group.add(fixed_light)
    g.root_node.accept(g.compiler)

    clock = pygame.time.Clock()

    max = 0
    winner = None
    new_win = False

    while True:
        for event in pygame.event.get():
            if event.type == pygame.QUIT: sys.exit(0)
            if hasattr(event, 'key'):
                if event.key == K_ESCAPE: sys.exit(0)

        if winner != None:
            if new_win:
                new_win = False
        else:
            for p in players:
                p.update()
                p.score += 1/(abs(p.vel)*2+.05)
                p.score_text.foreground = (1,1,1)
                if p.score/100 >= winning_score+1:
                    winner = p
                    new_win = True
                if p.score > max:
                    max = p.score
                    p.score_text.foreground = (0,1,0)

        g.draw()

        clock.tick(60)

if __name__ == '__main__': main()
```

APPENDIX E: HISTORICAL NOTES AND DATASHEETS

Historical Notes

- The Æther originally started as a small, handheld-sized device, with an ARM processor as its main strength. This idea was abandoned when we didn't believe ARM processors had enough speed to process 3D and OpenGL.
- The Æthersense started with capacitive sensing as its distance detection basis, but this idea was abandoned when capacitive sensing had too much drop off at farther distances. Infrared sensing was considered as a replacement, but the detection beam was found to be too narrow and the sensors would be disturbed too much by ambient light.
- The CF card is being used, essentially, as a cheap solid-state drive. However, we discovered (to the dismay of the Electrical Engineering department's student project budget) that Toshiba Compact Flash cards only support PIO4 data transfer (allowing only 1 MB/s read/write), the standard SYBA CF-to-IDE converters do not support UDMA (forcing only slow PIO4 or PIO6 transfers), and that RiData Compact Flash cards do not support OS booting. Eventually, we found the SYBA UDMA CF-to-IDE adapter and got the 8 GB Transcend Compact Flash card that supports all data transfer types and OS boot.
- The original Æthersense design used an ADC that converted the raw, analog MaxBotix sensor data for use in the rest of the system. The ADC, however, proved to be too noisy and added an inherent delay to a system where timing is critical.

- The Æthersense, it was posited, would work better with a 5 V tolerant, 3.3 V buffer in order to account for the difference in power coming out of the USB port and being required by the Atmel Microcontroller. However, after numerous redesigns, the Æthersense eventually used 3.6V Zener diodes to ground to control the voltages of the system.
- The display evolved from a simple (and gigantic) LED matrix to a full LCD display. After some investigation, we discovered that standard LCD monitors not only have too narrow a viewing angle for proper vertical alignment, but they also have a a native resolution that is simply overkill for the purposes of the project. During this investigation, we happened upon an LCD TV, which, of course, is optimized for wide viewing angles and relatively low resolution (720/1080.)
- The motherboard was originally planned on being a small, fanless, embedded VIA board. However, the VIA boards were on the edge of being cost prohibitive, and our research showed that the boards' integrated video chips were poor performers. In addition, we were not sure if there would be proper airflow for the fanless board in an enclosed space where the display would emit heat toward the board.

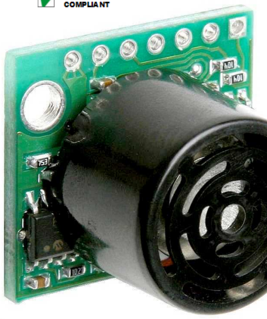
Datasheets Attached

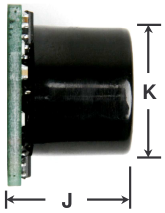
- The MaxBotix LV-MaxSonar-EZ4 High Performance Sonar Range Finder

LV-MaxSonar®-EZ4™ High Performance Sonar Range Finder

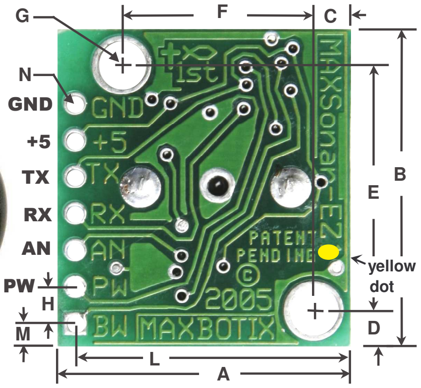
RoHS COMPLIANT

With 2.5V - 5.5V power the LV-MaxSonar®-EZ4™ provides very short to long-range detection and ranging, in an incredibly small package. The LV-MaxSonar®-EZ4™ detects objects from 0-inches to 254-inches (6.45-meters) and provides sonar range information from 6-inches out to 254-inches with 1-inch resolution. Objects from 0-inches to 6-inches range as 6-inches. The interface output formats included are pulse width output, analog voltage output, and serial digital output.





approximately actual size



A	0.785"	19.9 mm
B	0.870"	22.1 mm
C	0.100"	2.54 mm
D	0.100"	2.54 mm
E	0.670"	17.0 mm
F	0.510"	12.6 mm
G	0.124" dia.	3.1 mm dia.

H	0.100"	2.54 mm
J	0.645"	16.4 mm
K	0.610"	15.5 mm
L	0.735"	18.7 mm
M	0.065"	1.7 mm
N	0.038" dia.	1.0 mm dia.
weight, 4.3 grams		

values are nominal

Features

- Continuously variable gain for beam control and side lobe suppression
- Object detection includes zero range objects
- 2.5V to 5.5V supply with 2mA typical current draw
- Readings can occur up to every 50mS, (20-Hz rate)
- Free run operation can continually measure and output range information
- Triggered operation provides the range reading as desired
- All interfaces are active simultaneously
  - Serial, 0 to Vcc
    - 9600Baud, 81N
  - Analog, (Vcc/512) / inch
  - Pulse width, (147uS/inch)
- Learns ringdown pattern when commanded to start ranging
- Designed for protected indoor environments
- Sensor operates at 42KHz
- High output square wave sensor drive (double Vcc)

Benefits

- Very low cost sonar ranger
- Reliable and stable range data
- Sensor dead zone virtually gone
- Lowest power ranger
- Quality beam characteristics
- Mounting holes provided on the circuit board
- Very low power ranger, excellent for multiple sensor or battery based systems
- Can be triggered externally or internally
- Sensor reports the range reading directly, frees up user processor
- Fast measurement cycle
- User can choose any of the three sensor outputs

Beam Characteristics

Many applications require a narrower beam or lower sensitivity than the LV-MaxSonar®-EZ1™. Consequently, MaxBotix® Inc., is offering, the EZ2™, EZ3™, & EZ4™ with progressively narrower beam angles allowing the sensor to match the application. Sample results for the LV-MaxSonar®-EZ4™ measured beam patterns are shown below on a 12-inch grid. The detection pattern is shown for;

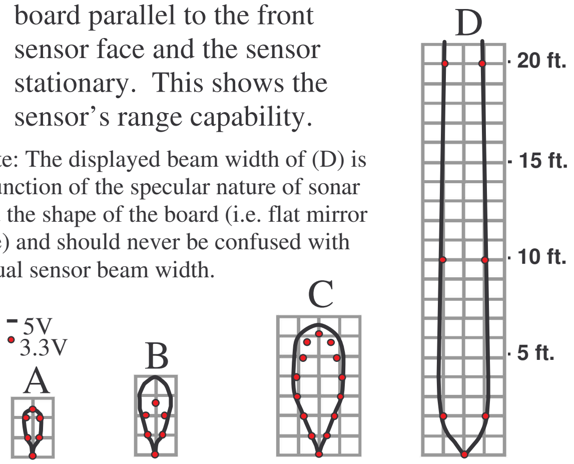
(A) 0.25-inch diameter dowel, note the narrow beam for close small objects,

(B) 1-inch diameter dowel, note the long narrow detection pattern,

(C) 3.25-inch diameter rod, note the long controlled detection pattern,

(D) 11-inch wide board moved left to right with the board parallel to the front sensor face and the sensor stationary. This shows the sensor's range capability.

Note: The displayed beam width of (D) is a function of the specular nature of sonar and the shape of the board (i.e. flat mirror like) and should never be confused with actual sensor beam width.



beam characteristics are approximate

LV-MaxSonar®-EZ4™ Pin Out

GND – Return for the DC power supply. GND (& Vcc) must be ripple and noise free for best operation.

+5V – Vcc – Operates on 2.5V - 5.5V. Recommended current capability of 3mA for 5V, and 2mA for 3V.

TX – When the *BW is open or held low, the TX output delivers asynchronous serial with an RS232 format, except voltages are 0-Vcc. The output is an ASCII capital “R”, followed by three ASCII character digits representing the range in inches up to a maximum of 255, followed by a carriage return (ASCII 13). The baud rate is 9600, 8 bits, no parity, with one stop bit. Although the voltage of 0-Vcc is outside the RS232 standard, most RS232 devices have sufficient margin to read 0-Vcc serial data. If standard voltage level RS232 is desired, invert, and connect an RS232 converter such as a MAX232. When BW pin is held high the TX output sends a single pulse, suitable for low noise chaining. (no serial data).

RX – This pin is internally pulled high. The EZ4™ will continually measure range and output if RX data is left unconnected or held high. If held low the EZ4™ will stop ranging. Bring high for 20uS or more to command a range reading.

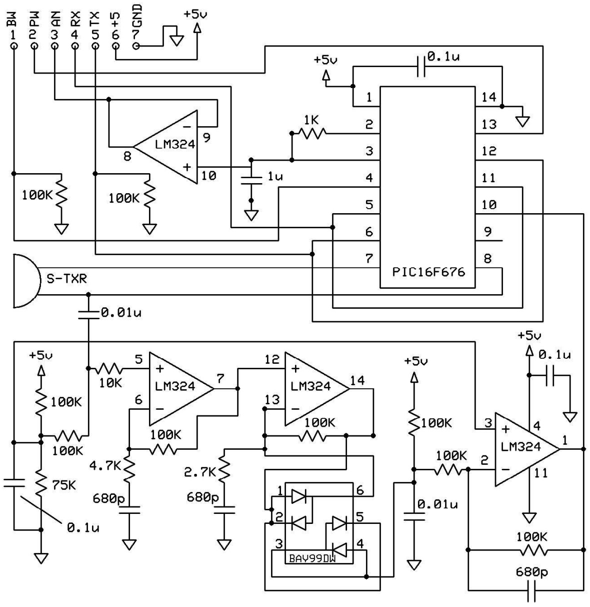
AN – Outputs analog voltage with a scaling factor of (Vcc/512) per inch. A supply of 5V yields ~9.8mV/in. and 3.3V yields ~6.4mV/in. The output is buffered and corresponds to the most recent range data.

PW – This pin outputs a pulse width representation of range. The distance can be calculated using the scale factor of 147uS per inch.

BW – *Leave open or hold low for serial output on the TX output. When BW pin is held high the TX output sends a pulse (instead of serial data), suitable for low noise chaining.

LV-MaxSonar®-EZ4™ Circuit

The LV-MaxSonar®-EZ4™ sensor functions using active components consisting of an LM324, a diode array, a PIC16F676, together with a variety of passive components.



LV-MaxSonar®-EZ4™ Timing Description

250mS after power-up, the LV-MaxSonar®-EZ4™ is ready to accept the RX command. If the RX pin is left open or held high, the sensor will first run a calibration cycle (49mS), and then it will take a range reading (49mS). Therefore, the first reading will take ~100mS. Subsequent readings will take 49mS. The LV-MaxSonar®-EZ4™ checks the RX pin at the end of every cycle. Range data can be acquired once every 49mS.

Each 49mS period starts by the RX being high or open, after which the LV-MaxSonar®-EZ4™ sends thirteen 42KHz waves, after which the pulse width pin (PW) is set high. When a target is detected the PW pin is pulled low. The PW pin is high for up to 37.5mS if no target is detected. The remainder of the 49mS time (less 4.7mS) is spent adjusting the analog voltage to the correct level. When a long distance is measured immediately after a short distance reading, the analog voltage may not reach the exact level within one read cycle. During the last 4.7mS, the serial data is sent. The LV-MaxSonar®-EZ4™ timing is factory calibrated to one percent at five volts, and in use is better than two percent. In addition, operation at 3.3V typically causes the objects range, to be reported, one to two percent further than actual.

LV-MaxSonar®-EZ4™ General Power-Up Instruction

Each time after the LV-MaxSonar®-EZ4™ is powered up, it will calibrate during its first read cycle. The sensor uses this stored information to range a close object. It is important that objects not be close to the sensor during this calibration cycle. The best sensitivity is obtained when it is clear for fourteen inches, but good results are common when clear for at least seven inches. If an object is too close during the calibration cycle, the sensor may then ignore objects at that distance.

The LV-MaxSonar®-EZ4™ does not use the calibration data to temperature compensate for range, but instead to compensate for the sensor ringdown pattern. If the temperature, humidity, or applied voltage changes during operation, the sensor may require recalibration to reacquire the ringdown pattern. Unless recalibrated, if the temperature increases, the sensor is more likely to have false close readings. If the temperature decreases, the sensor is more likely to have reduced up close sensitivity. To recalibrate the LV-MaxSonar®-EZ4™, cycle power, then command a read cycle.

Product / specifications subject to change without notice. For more info visit www.maxbotix.com/MaxSonar-EZ1_FAQ