Sarah Wood

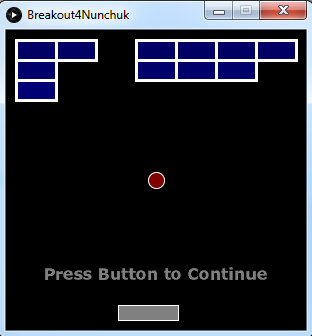
Embedded Programming

Chapter 9

Assignment 2

I will be using the Breakout code for Processing. See Ch7:Assg1 “Arduinoid: the Saga” if you have questions.

It took a little massaging of code to get the joystick to map properly.



<https://youtu.be/ABW4ghmdCOE> contains a 30 second video of the program working.

Arduino Code

WiiChuckDemo4Processing

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

#include "Wire.h"

//#include "WiiChuckClass.h" //most likely its WiiChuck.h for the rest of us.

#include "WiiChuck.h" //"WiiChuck.h" is the alternate -- ACCELEROMETER based

//#include "WiiChuck1.h" //joystick based

WiiChuck chuck = WiiChuck();

void setup() {

//nunchuck\_init();

Serial.begin(115200);

chuck.begin();

chuck.update();

//chuck.calibrateJoy();

}

void loop() {

delay(20);

chuck.update();

Serial.print(chuck.readJoyX());

Serial.print(" ");

Serial.print(chuck.readJoyY());

Serial.print(" ");

if (chuck.buttonZ) {

Serial.print("1");

} else {

Serial.print("0");

}

Serial.println();

}

WiiChuck.h

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

/\*

\* This is a version with declared POWER and GND pins.

\*

\* Nunchuck -- Use a Wii Nunchuck

\* Tim Hirzel http://www.growdown.com

\*

notes on Wii Nunchuck Behavior.

This library provides an improved derivation of rotation angles from the nunchuck accelerometer data.

The biggest different over existing libraries (that I know of ) is the full 360 degrees of Roll data

from teh combination of the x and z axis accelerometer data using the math library atan2.

It is accurate with 360 degrees of roll (rotation around axis coming out of the c button, the front of the wii),

and about 180 degrees of pitch (rotation about the axis coming out of the side of the wii). (read more below)

In terms of mapping the wii position to angles, its important to note that while the Nunchuck

sense Pitch, and Roll, it does not sense Yaw, or the compass direction. This creates an important

disparity where the nunchuck only works within one hemisphere. At a result, when the pitch values are

less than about 10, and greater than about 170, the Roll data gets very unstable. essentially, the roll

data flips over 180 degrees very quickly. To understand this property better, rotate the wii around the

axis of the joystick. You see the sensor data stays constant (with noise). Because of this, it cant know

the difference between arriving upside via 180 degree Roll, or 180 degree pitch. It just assumes its always

180 roll.

\*

\* This file is an adaptation of the code by these authors:

\* Tod E. Kurt, http://todbot.com/blog/

\*

\* The Wii Nunchuck reading code is taken from Windmeadow Labs

\* http://www.windmeadow.com/node/42

\*

\* Conversion to Arduino 1.0 by Danjovic

\* http://hotbit.blogspot.com

\*

\* Included the Fix from Martin Peris by Leopold Klimesch

\* http://blog.martinperis.com/2011/04/arduino-wiichuck.html

\*

\*

\*/

#ifndef WiiChuck\_h

#define WiiChuck\_h

#include "Arduino.h"

#include <Wire.h>

#include <math.h>

// these may need to be adjusted for each nunchuck for calibration

#define ZEROX 510

#define ZEROY 490

#define ZEROZ 460

#define RADIUS 210 // probably pretty universal

#define DEFAULT\_ZERO\_JOY\_X 124

#define DEFAULT\_ZERO\_JOY\_Y 132

//Set the power pins for the wiichuck, otherwise it will not be powered up

#define pwrpin PORTC3

#define gndpin PORTC2

class WiiChuck {

private:

uint8\_t cnt;

uint8\_t status[6]; // array to store wiichuck output

uint8\_t averageCounter;

//int accelArray[3][AVERAGE\_N]; // X,Y,Z

int i;

int total;

uint8\_t zeroJoyX; // these are about where mine are

uint8\_t zeroJoyY; // use calibrateJoy when the stick is at zero to correct

int lastJoyX;

int lastJoyY;

int angles[3];

bool lastZ, lastC;

public:

uint8\_t joyX;

uint8\_t joyY;

bool buttonZ;

bool buttonC;

void begin()

{

//Set power pinds

DDRC |= \_BV(pwrpin) | \_BV(gndpin);

PORTC &=~ \_BV(gndpin);

PORTC |= \_BV(pwrpin);

delay(100); // wait for things to stabilize

//send initialization handshake

Wire.begin();

cnt = 0;

averageCounter = 0;

// instead of the common 0x40 -> 0x00 initialization, we

// use 0xF0 -> 0x55 followed by 0xFB -> 0x00.

// this lets us use 3rd party nunchucks (like cheap $4 ebay ones)

// while still letting us use official oness.

// only side effect is that we no longer need to decode bytes in \_nunchuk\_decode\_byte

// seehttp://forum.arduino.cc/index.php?topic=45924#msg333160

//

Wire.beginTransmission(0x52); // device address

Wire.write(0xF0);

Wire.write(0x55);

Wire.endTransmission();

delay(1);

Wire.beginTransmission(0x52);

Wire.write(0xFB);

Wire.write((uint8\_t)0x00);

Wire.endTransmission();

update();

for (i = 0; i<3;i++) {

angles[i] = 0;

}

zeroJoyX = DEFAULT\_ZERO\_JOY\_X;

zeroJoyY = DEFAULT\_ZERO\_JOY\_Y;

}

void calibrateJoy() {

zeroJoyX = joyX;

zeroJoyY = joyY;

}

void update() {

Wire.requestFrom (0x52, 6); // request data from nunchuck

while (Wire.available ()) {

// receive byte as an integer

status[cnt] = \_nunchuk\_decode\_byte (Wire.read()); //

cnt++;

}

if (cnt > 5) {

lastZ = buttonZ;

lastC = buttonC;

lastJoyX = readJoyX();

lastJoyY = readJoyY();

//averageCounter ++;

//if (averageCounter >= AVERAGE\_N)

// averageCounter = 0;

cnt = 0;

joyX = (status[0]);

joyY = (status[1]);

for (i = 0; i < 3; i++)

//accelArray[i][averageCounter] = ((int)status[i+2] << 2) + ((status[5] & (B00000011 << ((i+1)\*2) ) >> ((i+1)\*2)));

angles[i] = (status[i+2] << 2) + ((status[5] & (B00000011 << ((i+1)\*2) ) >> ((i+1)\*2)));

//accelYArray[averageCounter] = ((int)status[3] << 2) + ((status[5] & B00110000) >> 4);

//accelZArray[averageCounter] = ((int)status[4] << 2) + ((status[5] & B11000000) >> 6);

buttonZ = !( status[5] & B00000001);

buttonC = !((status[5] & B00000010) >> 1);

\_send\_zero(); // send the request for next bytes

}

}

// UNCOMMENT FOR DEBUGGING

//byte \* getStatus() {

// return status;

//}

float readAccelX() {

// total = 0; // accelArray[xyz][averageCounter] \* FAST\_WEIGHT;

return (float)angles[0] - ZEROX;

}

float readAccelY() {

// total = 0; // accelArray[xyz][averageCounter] \* FAST\_WEIGHT;

return (float)angles[1] - ZEROY;

}

float readAccelZ() {

// total = 0; // accelArray[xyz][averageCounter] \* FAST\_WEIGHT;

return (float)angles[2] - ZEROZ;

}

bool zPressed() {

return (buttonZ && ! lastZ);

}

bool cPressed() {

return (buttonC && ! lastC);

}

// for using the joystick like a directional button

bool rightJoy(int thresh=60) {

return (readJoyX() > thresh and lastJoyX <= thresh);

}

// for using the joystick like a directional button

bool leftJoy(int thresh=60) {

return (readJoyX() < -thresh and lastJoyX >= -thresh);

}

int readJoyX() {

return (int) joyX - zeroJoyX;

}

int readJoyY() {

return (int)joyY - zeroJoyY;

}

// R, the radius, generally hovers around 210 (at least it does with mine)

// int R() {

// return sqrt(readAccelX() \* readAccelX() +readAccelY() \* readAccelY() + readAccelZ() \* readAccelZ());

// }

// returns roll degrees

int readRoll() {

return (int)(atan2(readAccelX(),readAccelZ())/ M\_PI \* 180.0);

}

// returns pitch in degrees

int readPitch() {

return (int) (acos(readAccelY()/RADIUS)/ M\_PI \* 180.0); // optionally swap 'RADIUS' for 'R()'

}

private:

uint8\_t \_nunchuk\_decode\_byte (uint8\_t x)

{

//decode is only necessary with certain initializations

//x = (x ^ 0x17) + 0x17;

return x;

}

void \_send\_zero()

{

Wire.beginTransmission (0x52); // transmit to device 0x52

Wire.write ((uint8\_t)0x00); // sends one byte

Wire.endTransmission (); // stop transmitting

}

};

#endif

WiiChuck1.h

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

/\*

\* This is a version converted to Arduino 1.0

\*

\*

\* Nunchuck -- Use a Wii Nunchuck

\* Tim Hirzel http://www.growdown.com

\*

notes on Wii Nunchuck Behavior.

This library provides an improved derivation of rotation angles from the Nunchuck accelerometer data.

The biggest different over existing libraries (that I know of ) is the full 360 degrees of Roll data

from teh combination of the x and z axis accelerometer data using the math library atan2.

It is accurate with 360 degrees of roll (rotation around axis coming out of the c button, the front of the wii),

and about 180 degrees of pitch (rotation about the axis coming out of the side of the wii). (read more below)

In terms of mapping the wii position to angles, its important to note that while the Nunchuck

sense Pitch, and Roll, it does not sense Yaw, or the compass direction. This creates an important

disparity where the Nunchuck only works within one hemisphere. At a result, when the pitch values are

less than about 10, and greater than about 170, the Roll data gets very unstable. essentially, the roll

data flips over 180 degrees very quickly. To understand this property better, rotate the wii around the

axis of the joystick. You see the sensor data stays constant (with noise). Because of this, it cant know

the difference between arriving upside via 180 degree Roll, or 180 degree pitch. It just assumes its always

180 roll.

\*

\* This file is an adaptation of the code by these authors:

\* Tod E. Kurt, http://todbot.com/blog/

\*

\* The Wii Nunchuck reading code is taken from Windmeadow Labs

\* http://www.windmeadow.com/node/42

\*

\* Conversion to Arduino 1.0 by Danjovic

\* http://hotbit.blogspot.com

\*

\*/

#ifndef WiiChuck1\_h

#define WiiChuck1\_h

#include "Arduino.h"

#include <Wire.h>

#include <math.h>

// these may need to be adjusted for each nunchuck for calibration

#define ZEROX 510

#define ZEROY 490

#define ZEROZ 460

#define RADIUS 210 // probably pretty universal

#define DEFAULT\_ZERO\_JOY\_X 124

#define DEFAULT\_ZERO\_JOY\_Y 132

class WiiChuck {

private:

uint8\_t cnt;

uint8\_t status[6]; // array to store wiichuck output

uint8\_t averageCounter;

//int accelArray[3][AVERAGE\_N]; // X,Y,Z

int i;

int total;

uint8\_t zeroJoyX; // these are about where mine are

uint8\_t zeroJoyY; // use calibrateJoy when the stick is at zero to correct

int lastJoyX;

int lastJoyY;

int angles[3];

bool lastZ, lastC;

public:

uint8\_t joyX;

uint8\_t joyY;

bool buttonZ;

bool buttonC;

void begin()

{

Wire.begin();

cnt = 0;

averageCounter = 0;

// instead of the common 0x40 -> 0x00 initialization, we

// use 0xF0 -> 0x55 followed by 0xFB -> 0x00.

// this lets us use 3rd party nunchucks (like cheap $4 ebay ones)

// while still letting us use official oness.

// only side effect is that we no longer need to decode bytes in \_nunchuk\_decode\_byte

// see http://forum.arduino.cc/index.php?topic=45924#msg333160

//

Wire.beginTransmission(0x52); // device address

Wire.write(0xF0);

Wire.write(0x55);

Wire.endTransmission();

delay(1);

Wire.beginTransmission(0x52);

Wire.write(0xFB);

Wire.write((uint8\_t)0x00);

Wire.endTransmission();

update();

for (i = 0; i<3;i++) {

angles[i] = 0;

}

zeroJoyX = DEFAULT\_ZERO\_JOY\_X;

zeroJoyY = DEFAULT\_ZERO\_JOY\_Y;

}

void calibrateJoy() {

zeroJoyX = joyX;

zeroJoyY = joyY;

}

void update() {

Wire.requestFrom (0x52, 6); // request data from nunchuck

while (Wire.available ()) {

// receive byte as an integer

status[cnt] = \_nunchuk\_decode\_byte (Wire.read()); //

cnt++;

}

if (cnt > 5) {

lastZ = buttonZ;

lastC = buttonC;

lastJoyX = readJoyX();

lastJoyY = readJoyY();

//averageCounter ++;

//if (averageCounter >= AVERAGE\_N)

// averageCounter = 0;

cnt = 0;

joyX = (status[0]);

joyY = (status[1]);

for (i = 0; i < 3; i++)

//accelArray[i][averageCounter] = ((int)status[i+2] << 2) + ((status[5] & (B00000011 << ((i+1)\*2) ) >> ((i+1)\*2)));

angles[i] = (status[i+2] << 2) + ((status[5] & (B00000011 << ((i+1)\*2) ) >> ((i+1)\*2)));

//accelYArray[averageCounter] = ((int)status[3] << 2) + ((status[5] & B00110000) >> 4);

//accelZArray[averageCounter] = ((int)status[4] << 2) + ((status[5] & B11000000) >> 6);

buttonZ = !( status[5] & B00000001);

buttonC = !((status[5] & B00000010) >> 1);

\_send\_zero(); // send the request for next bytes

}

}

// UNCOMMENT FOR DEBUGGING

//byte \* getStatus() {

// return status;

//}

float readAccelX() {

// total = 0; // accelArray[xyz][averageCounter] \* FAST\_WEIGHT;

return (float)angles[0] - ZEROX;

}

float readAccelY() {

// total = 0; // accelArray[xyz][averageCounter] \* FAST\_WEIGHT;

return (float)angles[1] - ZEROY;

}

float readAccelZ() {

// total = 0; // accelArray[xyz][averageCounter] \* FAST\_WEIGHT;

return (float)angles[2] - ZEROZ;

}

bool zPressed() {

return (buttonZ && ! lastZ);

}

bool cPressed() {

return (buttonC && ! lastC);

}

// for using the joystick like a directional button

bool rightJoy(int thresh=60) {

return (readJoyX() > thresh and lastJoyX <= thresh);

}

// for using the joystick like a directional button

bool leftJoy(int thresh=60) {

return (readJoyX() < -thresh and lastJoyX >= -thresh);

}

int readJoyX() {

return (int) joyX - zeroJoyX;

}

int readJoyY() {

return (int)joyY - zeroJoyY;

}

// R, the radius, generally hovers around 210 (at least it does with mine)

// int R() {

// return sqrt(readAccelX() \* readAccelX() +readAccelY() \* readAccelY() + readAccelZ() \* readAccelZ());

// }

// returns roll degrees

int readRoll() {

return (int)(atan2(readAccelX(),readAccelZ())/ M\_PI \* 180.0);

}

// returns pitch in degrees

int readPitch() {

return (int) (acos(readAccelY()/RADIUS)/ M\_PI \* 180.0); // optionally swap 'RADIUS' for 'R()'

}

private:

uint8\_t \_nunchuk\_decode\_byte (uint8\_t x)

{

//decode is only necessary with certain initializations

//x = (x ^ 0x17) + 0x17;

return x;

}

void \_send\_zero()

{

Wire.beginTransmission (0x52); // transmit to device 0x52

Wire.write ((uint8\_t)0x00); // sends one byte

Wire.endTransmission (); // stop transmitting

}

};

#endif