# Controlling the Wiimote with Object-Oriented Programming

Patricia Gavelek, Anders Dahl
Gavelek.p@husky.neu.edu
dahl.a@husky.neu.edu

Submit date: Feb. 29, 2016
Due Date: Feb. 29, 2016

## Abstract

In this lab, the Wiimote will be used as an input device for the ZedBoard. The Wiimote gives values representing pressed buttons and device motion. These values will be used to display acceleration information on the LEDs of the ZedBoard, using an object-oriented approach. The position, acceleration, velocity, and time elapsed of the Wiimote were also recorded and displayed for the extra credit. Short shell scripts, allowing a user to connect to the Wiimote, will be written to aid in development.

# Introduction

In this lab, the Wiimote will be used as an input device for the ZedBoard. The Wiimote gives values representing pressed buttons and device motion. These values will be used to display acceleration information on the LEDs of the ZedBoard, using an object-oriented approach. The position, acceleration, velocity, and time elapsed of the Wiimote were also recorded and displayed for the extra credit. Short shell scripts, allowing a user to connect to the Wiimote, will be written to aid in development.

# Lab Discussion

In order to develop the programs, the C programming language, the gcc compiler, the gdb debugger, the ZedBoard, and Wiimote were all used. A variety of C libraries were needed – stdio, stdlib, fcntl, unistd, and iostream.

The pre-lab showed how to write simple bash scripts and how to lay out C++ programs. Here are the results:

a).

CreateDir.bash

```
0 # !/ bin / bash
1 # cli - args . bash
2
3 $(mkdir $1)
```

```
Linok-2 :: Desktop/Embedded Des Enabling Robotics/lab4 » ./CreateDir.bash aaaaaaaaaaaaaaaaaaaaa
Linok-2 :: Desktop/Embedded Des Enabling Robotics/lab4 » ls
CreateDir.bash          Makefile              ShowDate.bash          ZedBoard.h             aaaaaaaaaaaaaaaaaaaaa zedMain.o
Lab-4_Assignment.pdf    Pre-Lab4.odt          ZedBoard.cpp           ZedBoard.o             zedMain.cpp           zedboard
```

b).

ShowDate.bash

```
1 # !/ bin / bash
0
1 echo "The current date is $(date)"
~
```

c), d), e).

Will be shown as part of this lab report.

# Results and Analysis

**Assignment 1:**

| A | B |
| --- | --- |
| A | 48 |
| B | 49 |
| up | 103 |
| down | 108 |
| left | 105 |
| right | 106 |
| - | 156 |
| + | 151 |
| home | 60 |
| 1 | 1 |
| 2 | 2 |

**Assignment 2:**

```
GCC = g++
CFLAGS = -g -Wall
OBJS = WiimoteBtns.o main.o
EXE = WiimoteBtns

$(EXE): $(OBJS)
	$(GCC) $(OBJS) -o $(EXE)

WiimoteBtns.o: WiimoteBtns.cpp WiimoteBtns.h
	$(GCC) $(CFLAGS) -c WiimoteBtns.cpp

main.o: main.cpp WiimoteBtns.h
	$(GCC) $(CFLAGS) -c main.cpp

clean:
	rm $(OBJS) $(EXE)
```

```
#ifndef ZEDBOARD_H
#define ZEDBOARD_H

// Class Definition
class WiimoteBtns {
    private:
        int fd; // File descriptor
    public:
        WiimoteBtns();  // Default constructor
        ~WiimoteBtns(); // Destructor
        void Listen();
        void ButtonEvent(int code, int value);
};

#endif
```

```cpp
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <iostream>
#include "WiimoteBtns.h"

/**
 * Opens /dev/input/event2 and checks for errors.
 */
WiimoteBtns::WiimoteBtns() {

    // Open Wiimote event file
    fd = open("/dev/input/event2", O_RDONLY);

    if (fd == -1) // Error openining the file descriptor
    {
        std::cerr << "Error: Could not open event file - forgot sudo?\n";
        exit(1);
    }
}

/**
 * Closes the file descriptor
 */
WiimoteBtns::~WiimoteBtns() {
    close(fd);
}
```

```
/**
 * Enters an infinite loop where a new event is read from the virtual file.
 * When an event is ready, its associated code and value fields are passed
 * in an invocation to ButtonEvent().
 */
void WiimoteBtns::Listen() {
    while (true)
    {
        // Read a packet of 32 bytes from Wiimote
        char buffer[32];
        read(fd, buffer, 32);

        // Extract code (byte 10) and value (byte 12) from packet
        int code = buffer[10];
        int value = buffer[12];

        this->ButtonEvent(code, value);
    }
}

/**
 * Displays a message on the screen reporting two values.
 *
 * @param code, a number showing the code
 * @param value, a number showing the value
 */
void WiimoteBtns::ButtonEvent(int code, int value) {

    // Print them
    std::cout << "Code = " << code << ", value = " << value << '\n';

}
```

```cpp
#include "WiimoteBtns.h"

int main()
{
    WiimoteBtns wiimote;
    wiimote.Listen();

    // Close Wiimote event file
    return 0;
}
```

**Assignment 3:**

```
GCC = g++
CFLAGS = -g -Wall
OBJS = WiimoteAccel.o main.o
EXE = WiimoteAccel

$(EXE): $(OBJS)
	$(GCC) $(OBJS) -o $(EXE)

WiimoteBtns.o: WiimoteAccel.cpp WiimoteAccel.h
	$(GCC) $(CFLAGS) -c WiimoteBtns.cpp

main.o: main.cpp WiimoteAccel.h
	$(GCC) $(CFLAGS) -c main.cpp

clean:
	rm $(OBJS) $(EXE)
```

```h
#ifndef WIIMOTE_ACCEL_H
#define WIIMOTE_ACCEL_H

// Class Definition
class WiimoteAccel {
    private:
        int fd; // File descriptor
    public:
        WiimoteAccel(); // Default constructor
        ~WiimoteAccel(); // Destructor
        void Listen();
        virtual void AccelerationEvent(int code, int acceleration);
};

#endif
```

```cpp
#include "WiimoteAccel.h"
#include <iostream>
#include <fcntl.h>

/**
 * A class constructor opens /dev/input/event0 (instead of /dev/input/event2 as
 * before) and checks for errors.
 */
WiimoteAccel::WiimoteAccel()
{
    fd = open("/dev/input/event0", O_RDONLY);
    if (fd == -1) // Error opening the file descriptor
    {
        std::cerr << "Error: Could not open event file - forgot sudo?\n";
        exit(1);
    }
}

/**
 * A class destructor closes the file.
 */
WiimoteAccel::~WiimoteAccel()
{
    // Close Wiimote event file
    close(fd);
}
```

```
/**
 * A public function called Listen() enters an infinite loop where a new
 * acceleration event is read from the virtual file. When found, the associated
 * code and acceleration values are passed to a public function called
 * AccelerationEvent().
 */
void WiimoteAccel::Listen() {
    while (true)
    {
        // Read a packet of 16 bytes from Wiimote
        char buffer[16];
        read(fd, buffer, 16);

        // Extract code (byte 10) and value (byte 12) from packet
        int code = buffer[10];
        short acceleration = * (short *) (buffer + 12);

        this->AccelerationEvent(code, acceleration);

    }
}

/**
 * A public function called AccelerationEvent() takes a code and acceleration
 * as integer arguments, and displays a message on the screen reporting these
 * two values.
 *
 * @param code, a number representing the code
 * @param acceleration, a number representing the acceleration
 */
void WiimoteAccel::AccelerationEvent(int code, int acceleration)
{
    // Print them
    std::cout << "Code = " << code <<
        ", acceleration = " << acceleration << '\n';
}
```

```
#include <stdlib.h>
#include <unistd.h>
#include "WiimoteAccel.h"

int main()
{

    WiimoteAccel wiimoteAccel;
    wiimoteAccel.Listen();

    return 0;
}
```

**Assignment 4:**

```
GCC = g++
CFLAGS = -g -Wall
OBJS = zedMain.o ZedBoard.o WiimoteAccel.o
EXE = zedboard

$(EXE): $(OBJS)
    $(GCC) $(OBJS) -o $(EXE)

zedMain.o: zedMain.cpp ZedBoard.h
    $(GCC) $(CFLAGS) -c zedMain.cpp

WiimoteAccel.o: WiimoteAccel.cpp WiimoteAccel.h
    $(GCC) $(CFLAGS) -c WiimoteAccel.cpp

ZedBoard.o: ZedBoard.cpp ZedBoard.h
    $(GCC) $(CFLAGS) -c ZedBoard.cpp

clean:
    rm $(OBJS)
```

```
#ifndef ZEDBOARD_H
#define ZEDBOARD_H

// Class Definition
class ZedBoard {
    private:
        char *pBase;    // virtual address where I/O was mapped
        int fd;         // file descriptor for dev memory
    public:
        ZedBoard();     // Default Constructor
        ~ZedBoard();    // Destructor
        void RegisterWrite(int offset, int value);
        int RegisterRead(int offset);
        void Set1Led(int ledNum, int state);
        void SetLedNumber(int value);
};

#endif
```

```
#ifndef WIIMOTE_ACCEL_H
#define WIIMOTE_ACCEL_H

// Class Definition
class WiimoteAccel {
    private:
        int fd; // File descriptor
    public:
        WiimoteAccel(); // Default constructor
        ~WiimoteAccel(); // Destructor
        void Listen();
        virtual int AccelerationEvent(int code, int acceleration);
};

#endif
```

```cpp
#include "WiimoteAccel.h"
#include <iostream>
#include <fcntl.h>

/**
 * A class constructor opens /dev/input/event0 (instead of /dev/input/event2 as
 * before) and checks for errors.
 */
WiimoteAccel::WiimoteAccel()
{
    fd = open("/dev/input/event0", O_RDONLY);
    if (fd == -1) // Error opening the file descriptor
    {
        std::cerr << "Error: Could not open event file - forgot sudo?\n";
        exit(1);
    }
}

/**
 * A class destructor closes the file.
 */
WiimoteAccel::~WiimoteAccel()
{
    // Close Wiimote event file
    close(fd);
}
```

```
/**
 * A public function called Listen() enters an infinite loop where a new
 * acceleration event is read from the virtual file. When found, the associated
 * code and acceleration values are passed to a public function called
 * AccelerationEvent().
 */
void WiimoteAccel::Listen() {
    while (true)
    {
        // Read a packet of 16 bytes from Wiimote
        char buffer[16];
        read(fd, buffer, 16);

        // Extract code (byte 10) and value (byte 12) from packet
        int code = buffer[10];
        short acceleration = * (short *) (buffer + 12);

        this->AccelerationEvent(code, acceleration);

    }
}

/**
 * A public function called AccelerationEvent() takes a code and acceleration
 * as integer arguments, and displays a message on the screen reporting these
 * two values.
 *
 * @param code, a number representing the code
 * @param acceleration, a number representing the acceleration
 */
int WiimoteAccel::AccelerationEvent(int code, int acceleration)
{
    // Print them
    std::cout << "Code = " << code <<
        ", acceleration = " << acceleration << '\n';

    return 0;
}
```

```c
#include <sys/mman.h>
#include <iostream>
#include <fcntl.h>
#include "ZedBoard.h"

// Physical base address of GPIO
const unsigned gpio_address = 0x400d0000;

// Length of memory-mapped IO window
const unsigned gpio_size = 0xff;

const int gpio_led1_offset = 0x12C;  // Offset for LED1
const int gpio_led2_offset = 0x130;  // Offset for LED2
const int gpio_led3_offset = 0x134;  // Offset for LED3
const int gpio_led4_offset = 0x138;  // Offset for LED4
const int gpio_led5_offset = 0x13C;  // Offset for LED5
const int gpio_led6_offset = 0x140;  // Offset for LED6
const int gpio_led7_offset = 0x144;  // Offset for LED7
const int gpio_led8_offset = 0x148;  // Offset for LED8

const int gpio_sw1_offset = 0x14C;  // Offset for Switch 1
const int gpio_sw2_offset = 0x150;  // Offset for Switch 2
const int gpio_sw3_offset = 0x154;  // Offset for Switch 3
const int gpio_sw4_offset = 0x158;  // Offset for Switch 4
const int gpio_sw5_offset = 0x15C;  // Offset for Switch 5
const int gpio_sw6_offset = 0x160;  // Offset for Switch 6
const int gpio_sw7_offset = 0x164;  // Offset for Switch 7
const int gpio_sw8_offset = 0x168;  // Offset for Switch 8

const int gpio_pbtnl_offset = 0x16C;  // Offset for left push button
const int gpio_pbtnr_offset = 0x170;  // Offset for right push button
const int gpio_pbtnu_offset = 0x174;  // Offset for up push button
const int gpio_pbtnd_offset = 0x178;  // Offset for down push button
const int gpio_pbtnc_offset = 0x17C;  // Offset for center push button
```

```
/**
 * Constructor Initialize general-purpose I/O
 *  - Opens access to physical memory /dev/mem
 *  - Maps memory at offset 'gpio_address' into virtual address space
 *
 * @param  None»Default constructor does not need arguments.
 * @return» None Default constructor does not return anything.
 */
ZedBoard::ZedBoard(){
    std::cout << "\nStarting...." << std::endl;
    fd = open( "/dev/mem", O_RDWR);
    pBase = (char *) mmap(NULL,gpio_size,PROT_READ | PROT_WRITE,
            MAP_SHARED,fd,gpio_address);
    /* Check error */
    if (pBase == MAP_FAILED)
    {
        std::cerr << "Mapping I/O memory failed - Did you run with 'sudo'?\n";
        exit(1); // Returns 1 to the operating system;
    }
}

/**
 * Destructor to close general-purpose I/O.
 * - Uses virtual address where I/O was mapped.
 * - Uses file descriptor previously returned by 'open'.
 *
 * @param  None»Destructor does not need arguments.
 * @return» None Destructor does not return anything.
 */
ZedBoard::~ZedBoard(){
    munmap(pBase, gpio_size);
    close(fd);
    std::cout << "\nTerminating...." << std::endl;
}
```

```
/**
 * Write a 4-byte value at the specified general-purpose I/O location.
 *
 * - Uses base address returned by 'mmap'.
 * @parem offset»   Offset where device is mapped.
 * @param value»»   Value to be written.
 */
void ZedBoard::RegisterWrite(int offset, int value)
{
    * (int *) (pBase + offset) = value;
}

/**
 * Read a 4-byte value from the specified general-purpose I/O location.
 *
 * - Uses base address returned by 'mmap'.
 * @param offset»   Offset where device is mapped.
 * @return» »   Value read.
 */
int ZedBoard::RegisterRead(int offset)
{
    return * (int *) (pBase + offset);
}
```

```
/**
 * Changes the state of an LED (ON or OFF)
 *
 * - Uses base address of I/O
 * @param ledNum    LED number (0 to 7)
 * @param state State to change to (ON or OFF)
 */
void ZedBoard::Set1Led(int ledNum, int state)
{
    std::cout << "\nWriting to LED " << ledNum << std::endl;
    RegisterWrite(gpio_led1_offset + (ledNum * 4), state);
    std::cout << "LED offset: " << std::hex << gpio_led1_offset+(ledNum*4) << "    ";
    std::cout << "LED state: " << state << std::endl;
}

/**
 * Show lower 8 bits of integer value on LEDs
 *
 * - Calls Set1Led() to set all LEDs
 * @param value Value to show on LEDs
 */
void ZedBoard::SetLedNumber(int value)
{
    std::cout << "\nWriting to LEDs...." << std::endl;
    for(int i = 0; i < 8; i++) {// write to all LEDs
        Set1Led(i, (value / (1<<i)) % 2);
    }
}
```

```
#include <iostream>
#include "ZedBoard.h"
#include "WiimoteAccel.h"

class WiimoteToLed : public WiimoteAccel {
        private:
                ZedBoard * zed_p;
        public:
                WiimoteToLed(ZedBoard * zed);
                int AccelerationEvent(int code, int acceleration);
};

WiimoteToLed::WiimoteToLed(ZedBoard * zed)
{
    zed_p = zed;
}
```

```
/**
 * An acceleration event taking in an acceleration from the Wiimote
 * and handling it accordingly.
 *
 * @param  code, an integer representing the code.
 * @param acceleration, an integer representing the acceleration.
 * @return an integer representing errors. 0 for no errors.
 */
int WiimoteToLed::AccelerationEvent(int code, int acceleration)
{
    if (code != 3)
        return 0;

    if (acceleration < -100)
        acceleration = -100;

    if (acceleration > 100)
        acceleration = 100;

    // How many LEDs to light up.
    acceleration = (double) abs(acceleration) / 100 * 8;

    // Sets the LEDs based on the acceleration.
    int i;
    for(i = 0; i < 8; i++) {
        if(i < acceleration) {
            zed_p -> Set1Led(i, 1);
        } else {
            zed_p -> Set1Led(i, 0);
        }
    }

    return 0;
}
```

**Assignment 5:**

```
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <math.h>
#include "ZedBoard.h"
#include "WiimoteAccel.h"

struct timeval t0_x;
struct timeval t0_y;
struct timeval t0_z;

int x_acceleration = 0;
int y_acceleration = 0;
int z_acceleration = 0;

int x_velocity = 0;
int y_velocity = 0;
int z_velocity = 0;

int x_position = 0;
int y_position = 0;
int z_position = 0;

// Class definition
class WiimoteToLed : public WiimoteAccel {
    private:
        ZedBoard * zed_p; // Pointer to a ZedBoard
    public:
        WiimoteToLed(ZedBoard * zed);
        int AccelerationEvent(int code, int acceleration);
};

WiimoteToLed::WiimoteToLed(ZedBoard * zed)
{
    zed_p = zed; // Set zedboard
}
```

```
/**
 * An acceleration event taking in an acceleration from the Wiimote
 * and handling it accordingly. It prints the acceleration,
 * velocity, position, and time elapsed.
 *
 * @param  code, an integer representing the code.
 * @param acceleration, an integer representing the acceleration.
 * @return an integer representing errors. 0 for no errors.
 */
int WiimoteToLed::AccelerationEvent(int code, int acceleration)
{
    int x_acceleration_new = 0;
    int y_acceleration_new = 0;
    int z_acceleration_new = 0;

    int x_velocity_new = 0;
    int y_velocity_new = 0;
    int z_velocity_new = 0;

    int x_position_new = 0;
    int y_position_new = 0;
    int z_position_new = 0;

    struct timeval t1_x;
    struct timeval t1_y;
    struct timeval t1_z;

    long time = 0;

    sleep(1);

    if(code == 0)
        return 0;
```

```
    // X coordinate
    if (code == 3) {

        gettimeofday(&t1_x, 0);
        time = ((t1_x.tv_sec-t0_x.tv_sec)*1000000 + t1_x.tv_usec-t0_x.tv_usec) / 1000000;

        x_acceleration_new = acceleration;
        x_velocity_new = x_velocity + (acceleration * time);
        x_position_new = (pow(x_velocity_new, 2) - pow(x_velocity, 2))
        / (2 * x_acceleration_new);

        x_acceleration = x_acceleration_new;
        x_velocity = x_velocity_new;
        x_position = x_position_new;
        t0_x = t1_x;

    // Y coordinate
    } else if (code == 4) {

            gettimeofday(&t1_y, 0);
        time = ((t1_y.tv_sec-t0_y.tv_sec)*1000000 + t1_y.tv_usec-t0_y.tv_usec) / 1000000;

        y_acceleration_new = acceleration;
        y_velocity_new = y_velocity + (acceleration * time);
        y_position_new = (pow(y_velocity_new, 2) - pow(y_velocity, 2))
        / (2 * y_acceleration_new);

        y_acceleration = y_acceleration_new;
        y_velocity = y_velocity_new;
        y_position = y_position_new;
        t0_y = t1_y;
```

```
    // Z coordinate
    } else if (code == 5) {

            gettimeofday(&t1_z, 0);
        time = ((t1_z.tv_sec-t0_z.tv_sec)*1000000 + t1_z.tv_usec-t0_z.tv_usec) / 1000000;

        z_acceleration_new = acceleration;
        z_velocity_new = z_velocity + (acceleration * time);
        z_position_new = (pow(z_velocity_new, 2) - pow(z_velocity, 2))
        / (2 * z_acceleration_new);

        z_acceleration = z_acceleration_new;
        z_velocity = z_velocity_new;
        z_position = z_position_new;
        t0_z = t1_z;
    }

    printf("Acceleration: (%d, %d, %d)\n", x_acceleration, y_acceleration, z_acceleration);
    printf("Velocity: (%d, %d, %d)\n", x_velocity, y_velocity, z_velocity);
    printf("Position: (%d, %d, %d)\n", x_position, y_position, z_position);
    printf("Time: (%ld)\n", time);

    return 0;
}
```

```
int main()
{
    // Instantiate ZedBoard object statically
    ZedBoard zed_board;

    gettimeofday(&t0_x, 0);
    gettimeofday(&t0_y, 0);
    gettimeofday(&t0_z, 0);

    // Instantiate WiimoteToLed object statically, passing a pointer to the
    // recently created ZedBoard object.
    WiimoteToLed wiimote_to_led(&zed_board);
    // Enter infinite loop listening to events. The overridden function
    // WiimoteToLed::AccelerationEvent() will be invoked when the user moves
    // the Wiimote.
    wiimote_to_led.Listen();
    // Unreachable code, previous function has an infinite loop
    return 0;
}
```

# Conclusion

In this lab, values were used to display acceleration information on the LEDs of the ZedBoard, using an object-oriented approach. The position, acceleration, velocity, and time elapsed of the Wiimote were also recorded and displayed for the extra credit. In general, knowing how to read and manipulate states and values critical to embedded design. In the future, for more work, more programs could be written to control the LEDs in different ways. The programs in the lab could also be rewritten, tested, and debugged. We ran into a few issues on the extra credit with number arithmetic. It would be helpful to write tests for such code. This is something that could be accomplished in future work.

# References

N/A