**An\_Najah National University  
Computer Engineering Department**

**Archery**  
Safer than Ever Before

**Senior Project 2**

**Siba Fadda Munia Touqan**

**Supervised By:**

**Dr.Luai Malhis Dr.Samer Arandi**



Table of Contents:

[Abstract 6](#_Toc324186685)

[Introduction 7](#_Toc324186686)

[Chapter 1: System Hardware 8](#_Toc324186687)

[1.1 Stimulus (IR Light Source) 8](#_Toc324186688)

[1.2 Force Sensor 8](#_Toc324186689)

[1.3 Camera 9](#_Toc324186690)

[1.3.1 Overview 9](#_Toc324186691)

[1.3.2 Initialization and Sensitivity Modes 10](#_Toc324186692)

[1.3 Microcontroller (Control and Communication Unit) 12](#_Toc324186693)

[1.4 Accelerometer 14](#_Toc324186694)

[1.5 XBEE Wireless Transmitter/Receiver 16](#_Toc324186695)

[1.6 Bow and Arrow 17](#_Toc324186696)

[Chapter 2: System Software 17](#_Toc324186697)

[2.1 PIC microcontroller software 17](#_Toc324186698)

[2.1.1 First PIC 17](#_Toc324186699)

[2.1.1.1 Force Sensor 17](#_Toc324186700)

[2.1.1.2 Camera Code 18](#_Toc324186701)

[2.1.1.3 Accelerometer code 19](#_Toc324186702)

[2.1.1.4 Wireless Communication 19](#_Toc324186703)

[2.1.2 Second PIC 19](#_Toc324186704)

[2.2 C# game 20](#_Toc324186705)

[Game Flow 20](#_Toc324186706)

[Let’s assume for now that the user chose the first option, Classical Mode. 21](#_Toc324186707)

[2.2.1 Classical Archery 21](#_Toc324186708)

[2.2.2 Freestyle 23](#_Toc324186709)

[Chapter 3: Work Flow 24](#_Toc324186710)

[Chapter 4: Problems and Solutions 25](#_Toc324186711)

[Camera 25](#_Toc324186712)

[XBEE Wireless Component 26](#_Toc324186713)

[Accelerometer 26](#_Toc324186714)

[Lab Hours 27](#_Toc324186715)

[Possible Improvements 28](#_Toc324186716)

[Conclusion 29](#_Toc324186717)

[References 30](#_Toc324186718)

[Appendix A 31](#_Toc324186719)

List of Figures:

[Figure 1.1 IR LED 8](#_Toc324190981)

[Figure1.2.1 Force Sensor 9](#_Toc324190982)

[Figure 1.2.2 How to Connect a FSR 9](#_Toc324190983)

[Figure1.3.2.1: The PixArt IC Pin Assignment 11](#_Toc324190984)

[Figure1.3.2. 2: What the PixArt Camera looks like 11](#_Toc324190985)

[Figure 1.3 PIC18F4620 12](#_Toc324190986)

[Figure 1.4.1 ADXL335 Accelerometer 14](#_Toc324190987)

[Figure 1.4.2Shows the Various Tilt Conditions for an Accelerometer. 15](#_Toc324190988)

[Figure 1.5.1 XBEE S1 16](#_Toc324190989)

[Figure 2.2.1Archery 21](#_Toc324190990)

[Figure 2.2.1.3Level Selection 22](#_Toc324190991)

[Figure 2.2.1.2 Targets Look and Feel in the First Game. 21](file:///D:\UNI\project\Documentation\Archery.docx#_Toc324190992)

[Figure 2.2.1.4 Classical Mode Levels 22](#_Toc324190993)

[Figure 2.2.1.5 Trophy 23](#_Toc324190994)

[Figure 2.2.1.6 Extra Levels 23](#_Toc324190995)

[Figure 0.7 Wiimote Broken Apart Figure 0.8 Wiimote IR Camera 32](#_Toc324190996)

[Figure 0.9 IR Camera oscillator Circuit 32](#_Toc324190997)

[Figure 0.10 Voltage Converter 33](#_Toc324190998)

List of Tables:

[Table 1: Wii IR Camera Pinout 9](#_Toc324107584)

[Table 2: Pin configuration of Xbee Series 1 14](#_Toc324107585)

Abstract

The game of Archery is a very fun game. It teaches people the art of patience and concentration. But parents are often hesitant to allow their children to play and practice from a young age, and that is because it can be dangerous due to the nature of the game equipment. In our senior project, we developed a way to make the game of Archery safe. We use non-pointy arrows. The equipment used is made from very smooth wood that does not have any sharp edges or splinters. The arrow does not have to be physically fired because the way we designed the bow and arrow, the aroow has a very wide base that cannot leave the bow, and thus ensures no flying objects that would harm the player or those around.

We used PIC microcontrollers to control our system. XBEE wireless components to communicate various data within the system. An IR camera and an IR LED to determine coordinates. A force sensor to detect arrow launching. An accelerometer for angles, and finally we interact with a game built in C#.

Introduction

This documentation will describe in details, all aspects of our Senior Project II: Hardware.

The method we will use to present this data will be through tackling each major part of this project, and explaining its implementation and logic behind it separately, then, at the end, we will demonstrate how all these parts come together in a highly structured way to give us the final outcome, safety.

The aim of our project is to make the game of Archery safer for everyone, especially children. To do that, the first thing we’ve done is to substitute the regular dangerous pointy arrows with a wooden arrow that has smooth ends. Instead of interacting with a real physical target board, we will be interacting with a C# game. The arrow will not be flying out of the bow. Instead, it will move as if it is being fired but it is designed in a way that inhibits it from flying out of the bow to ensure the safety of the users in addition to the aforementioned smooth arrow ends.

An IR camera is used to detect the position of the IR LED placed on the arrow to figure out the x,y coordinates and map them onto the game. An accelerometer is used to detect the angles of the arrow to get more accurate positions for the arrow. XBEE is used for wireless communication in the system so send coordinates and commands. A force sensor is used to detect the launching of the arrow, and two PIC microcontrollers are used to control the hardware, the C# game, and the wireless communication.

Chapter 1: System Hardware

# 1.1 Stimulus (IR Light Source)

An IR LED, also known as IR transmitter, is a special purpose LED that transmits infrared rays in the range of 760 nm wavelength. Such LEDs are usually made of gallium arsenide or aluminum gallium arsenide.

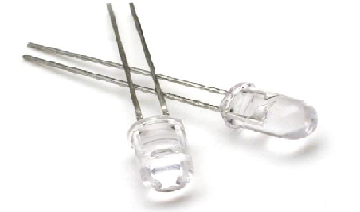


Figure 1. IR LED

An IR LED (as shown in Figure 1.1.1) is the simplest way for generating a reasonable IR light source; of course this LED is attached to the arrow to get accurate reading for the x and y coordinates of the arrow. Keep in mind that the longer connector is the Anode, and the shorter one is the Cathode. An IR LED needs to be connected to VCC through a 10K resistor to make sure it doesn’t get damaged.

# 1.2 Force Sensor

Force Sensing Resistors (FSR) are a polymer thick film (PTF) device which exhibits a decrease in resistance with an increase in the force applied to the active surface. Its force sensitivity is optimized for use in human touch control of electronic devices.

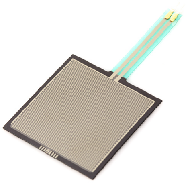


Figure.2.1 Force Sensor

A force sensor (as seen in Figure 1.2.1) is attached to the end of the arrow. Its main objective in our system is to detect the launching of the arrow. The FSR is connected as shown in Figure 1.2.2. Using trial and error, we were able to determine the appropriate threshold that ensures launching. When the output value fluctuates above and below the threshold value within a small period of time (critical time), launching event is detected. Afterwards, the PIC starts reading and transmitting x and y coordinates of the IR LED attached to the arrow. In addition to accelerometer values to determine the angles in all coordinates, and sends then back to the PC host to perform computation on the values. Then map them to their correct location onto the screen and perform actions accordingly.

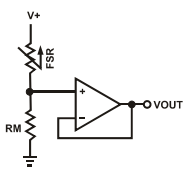


Figure 1..2 How to Connect a FSR

# 1.3 Camera

## 1.3.1 Overview

Instead of purchasing an expensive IR camera that only detects IR points. We bought a Wiimote and extracted the existing IR camera within. Refer to Appendix A for a step-by-step guide on how to extract the IR camera within the Wiimot and how to interface it with a microcontroller circuit and make use of its values.

PixArt\* CMOS sensor consists of two functional parts:

1. The actual sensing part, were up to four IR points can be detected at the same time.
2. The internal processor part, which is responsible of translating the location of the IR points to correct x,y coordinates and size value (which gives a parameter indicating the distance of the IR point from the cameras position).

Hence this camera chip has an integrated Multi-Object Tracking™ engine (MOT sensor™) [3],it is basically the hardware version of blob detection software’s but with a resolution of 1,024 horizontal pixels by 768 vertical pixels (it actually has a resolution of 128x96 but the built in processor uses 8x sub-pixel analysis to provide the described 1024X768 resolution), and a refresh rate of 100 samples per second.

Other Specifications:

* 8 pins (4x2)
* Operates with 3.3V.
* 25 MHz clock.
* I²C (Inter-Integrated Circuit) Bus communication. (400 KHz fast with slave address 0xB0)
* Field of View is about 33 degrees horizontally and 23 degrees vertically.
* 3 sensitivity modes.

Since this chip evolved from the incorporation of two companies (Nintendo and PixArt) there is no public datasheet, all the information gathered are from various engineering researches and individual contributions.

## 1.3.2 Initialization and Sensitivity Modes

The chip must be enabled by certain instruction formats, which will leave it in one of three states:

1. On, but no data is being taken.
2. On, and data is being taken at half sensitivity.
3. On, and data is being taken at full sensitivity.

The data output format will differ according to these states, but can still be categorized in three modes:

1. Basic. (10 bytes , 5 bytes for each pair, with a total of 4 dots “two pairs”)
2. Extended. (12 bytes, 3 bytes for each dot, with a total of 4 dots)
3. Full (36 bytes, 9 bytes for each dot, with a total of 4 dots)

The main difference between each mode is the number of bytes being transmitted for each IR dot, and whether the rough size value for each dot is required or not (We used the extended mode for our project).

The Pin assignment and the actual packaging for the camera chip:

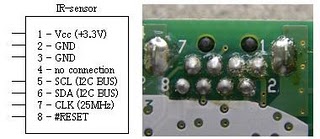


Figure1.3.2.: The PixArt IC Pin Assignment

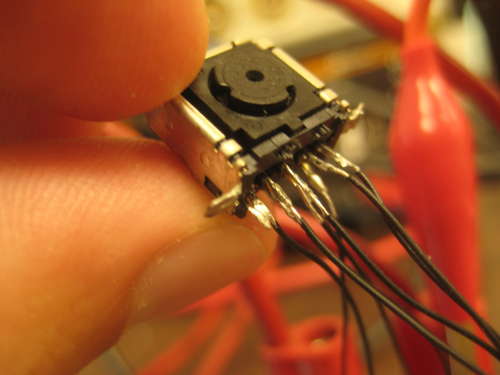


Figure1.3.2. : What the PixArt Camera looks like

**Pinouts:**

In the following table we show the functionality of each pin for the Wii IR camera and how it is connected.

Table : Wii IR Camera Pinout

|  |  |
| --- | --- |
| Pin # | Description |
| 1 | pulled up to +3.3 V |
| 2,3 | Grounded |
| 4 | Not Connected |
| 5 | Serial Clock (I2C) |
| 6 | Serial Data (I2C) |
| 7 | Components Clock |
| 8 | Active Low Reset |

One last note before jumping to the next component, is that the communication with the camera chip will be through I2C bus protocol, which is a master/slave architecture were all devices are connected together by two signals : serial clock (SCL) (Pin 5) and serial data (SDA) (Pin 6).

# 1.3 Microcontroller (Control and Communication Unit)

The Microcontroller needed for the system is a high performance one with I2C capabilities. This can be found in the Microchip PIC 18FXXX High-Performance Series, which includes (PIC18F2450, PIC18F4620, PIC18F2550, and PIC18F4550). As we mentioned in the camera chip section, the IC needs to be initialized with a certain instruction sequence, and here comes the role of the Microcontroller, which would send the right instructions for the chip to enable its operation after it is being plugged to the power supply (3.3V).

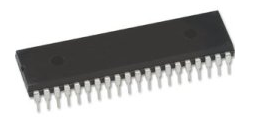


Figure .3 PIC18F4620

Of course this wouldn’t be the only task of the microcontroller. The data transmitted from the camera chip afterwards (X, Y coordinates of the wanted IR dots on I2C bus) must go through this controller and then sent through the wireless XBEE component to the PC side to control the game.

All the communication taking place at this stage is over the SDA line of the I2C bus, were the PIC Microcontroller acts as the master and the camera chip acts as the only slave connected to the bus lines (SCL,SDA).

Note that our project uses two PIC microcontrollers. The first PIC is attached to the IR camera. It communicates with the camera to get the coordinates of the position of the IR LED. It also captures the values of the accelerometer positioned on the arrow. It then sends both values to the second PIC microcontroller. The second PIC is attached to the host PC where the C# game is present. It sends the data serially to the computer which in turn performs all appropriate computations and then maps the values to their appropriate locations. The game software then performs further actions accordingly.

Theory of Operation:

* The PIC (master device) issues a start condition, hence the camera chip (slave) starts listening on the serial line to see if it is the one concerned (by checking the address sent by the master and comparing it with its own)
* A read/write operation is specified accordingly by the PIC (master)
* The PIC (master) starts transmitting and waits for an acknowledgement from the camera chip (slave). Usually the transmitter sends 8-bits of data and the receiver replies with a one bit ACK (I2C standards). An ACK with the value of “0” would mean the camera is responsive. An ACK of “1” would mean it is not operational.
* The PIC (master) issues a stop condition when the communication is complete.
* The first PIC communicates with the second one to send the data (both coordinates and tilt values).
* The second PIC receives this data wirelessly using XBEE S1. **Refer to 1.5 for details.**
* The second PIC communicates with the C# game that performs computation on the received data to acquire valid values to be used in the game.

This operation is valid when the PIC is issuing a write command (hence sending instructions to initialize the camera), or when it is issuing a read command (when it is reading the data coordinates form the camera).

After filling the data buffers with the x, y coordinates, they get sent from the first PIC microcontroller to the second PIC microcontroller. And from there, the second PIC microcontroller communicates with the C# game at a PC host and sends those values to it. At the PC host, the C# game prior to using the values obtained from the camera, it performs further processing to calibrate the values acquired and make sure it is within the range. As well as map it to its correct location on screen to have accurate projection of results.

Everything will be explained in details as you advance in the documentation. Each part has a separate section to describe the hardware used and the software employed to control it.

# 1.4 Accelerometer

We used a 3-axis accelerometer to acquire tilt values of the arrow upon launching. These values are incorporated in our calculations to provide more accurate mapping to the location of where the arrow is supposed to hit the target, and score the player accordingly.

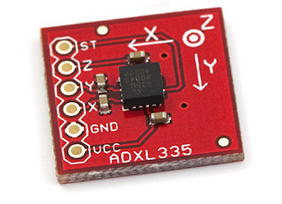


Figure 1.4. ADXL335 Accelerometer

The ADXL335 is a complete 3-axis acceleration measurement system. The ADXL335 has a measurement range of ±3 g minimum. It contains a polysilicon surface-micromachined sensor and signal conditioning circuitry to implement open-loop acceleration measurement architecture. The output signals are analog voltages that are proportional to acceleration. The ADXL335 is tested and specified at VS = 3 V; however, it can be powered with VS as low as 1.8 V or as high as 3.6 V.

The accelerometer can be used for measuring both dynamic and static measurements of acceleration. Tilt is a static measurement where gravity is the acceleration being measured. Therefore, to achieve the highest degree resolution of a tilt measurement, a low-g, high sensitivity accelerometer is required.

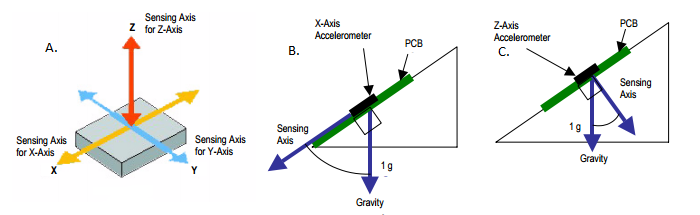
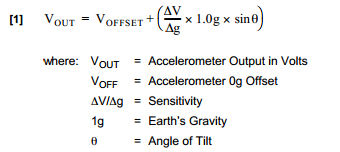


Figure 1.4.Shows the Various Tilt Conditions for an Accelerometer. In (A) We See X, Y, and Z-Axis for Sensing the Acceleration. (B) Shows the Gravity Component of a Tilted X-Axis Accelerometer. (C) Shows the Gravity Component of a Tilted Z-Axis Accelerometer

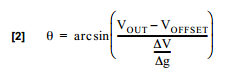
In order to determine the angle of tilt, θ, the A/D values from the accelerometer are sampled by the ADC channel on the microcontroller. The acceleration is compared to the zero g offset to determine if it is a positive or a negative acceleration. For instance, if the value is greater than the offset then the acceleration is seeing a positive acceleration, so the offset is subtracted from the value and the resulting value is then used with a lookup table to determine the corresponding degree of tilt, or the value is passed to a tilt algorithm. If the acceleration is negative, then the value is subtracted from the offset to determine the amount of negative acceleration and then passed to the lookup table or algorithm. One solution can measure 0° to 90° of tilt with a single axis accelerometer, or another solution can measure 360° of tilt with two axis configuration (XY, X and Z), or a single axis configuration (e.g. X or Z), where values in two directions are converted to degrees and compared to determine the quadrant that they are in. A tilt solution can be solved by either implementing an arccosine function, an arcsine function, or a look-up table depending on the power of the microcontroller and the accuracy required by the application. For simplicity, we will use the equation: θ = arcsin(x). The arcsin(y) can determine the range from 0° to 180°, but it cannot discriminate the angles in range from 0° to 360°, e.g. arcsin(45º) = arcsin(135º). However, the sign of x and y can be used to determine which quadrant the angle is in. By this means, we can calculate the angle β in one quadrant (0-90º) using arcsin(y) and then determine θ in the determined quadrant.

This first equation determines the value of Vout for certain tilt angles. It calculates it as follows:



The value of ∆V/∆g is determined from the accelerometers datasheet. For our accelerometer, the value of ∆V/∆g is 300 mV/g.

Solving the first equation for the angle yields the following:



The computed values will be used to determine the trajectory of the arrow to accurately map its location.

# 1.5 XBEE Wireless Transmitter/Receiver

We used XBEE S1 wireless component to send and receive data between our two PIC microcontrollers. XBEE is a wireless communication device from digi.com that uses ZigBee protocol. “ZigBee” is a specification for a suite of high level communication protocols using small, low-power digital radios based on the IEEE 802.15.4-2003 standard for wireless personal area networks (WPANs). The technology defined by the ZigBee specification is intended to be simpler and less expensive than other WPANs , such as Bluetooth . ZigBee is targeted at radio-frequency (RF) applications that require a low data rate, long battery life, and secure networking.

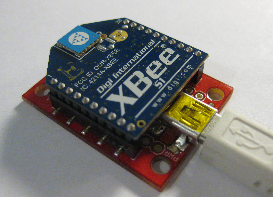
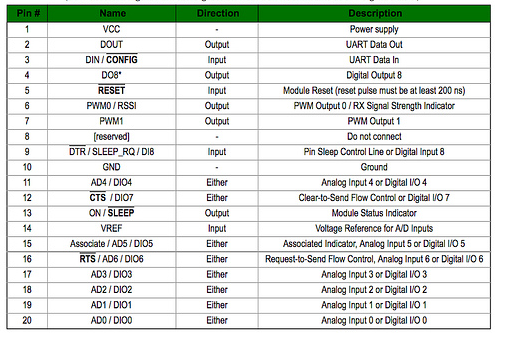


Figure 1.5. XBEE S1

It operates with 3.3V and uses 50mA. Some Xbee ‘s pins can be set as Analog Input, Digital Input, Digital Output, or Analog Output (PWM). The numbers of pins are up to seven channels of analog input, nine channels of digital I/O and two channels of PWM.

Table : Pin configuration of Xbee Series 1



# 1.6 Bow and Arrow

We constructed our bow and arrow from wood. We wanted to use a safe material that would not break easily. We wanted to use a material that would be smooth while used and would not have sharp edges to provide the level of safety we are aiming for. The arrow is designed in a way that would make it unable to leave the bow, which would ensure no flying objects that could harm the player or others around.

Chapter 2: System Software

Our system is composed of two main parts. The first part consists of the actual wooden hardware of the bow and arrow with the first PIC microcontroller. The first PIC microcontroller controls the IR camera, accelerometer, force sensor and a XBEE wireless component. The second PIC microcontroller controls the other XBEE wireless component in addition to communicating with the host PC to send the values of the coordinates and acceleration in different axis’s.

# 2.1 PIC microcontroller software

We will be talking about each PIC separately. We will describe in details the software used to control and communicate with each component connected to the PIC.

## 2.1.1 First PIC

As mentioned earlier, the first PIC controls and communicates with the force sensor, the IR camera, an accelerometer, and the WBEE wireless component. We will discuss each individually.

### 2.1.1.1 Force Sensor

The fluctuation of the value of the force sensor is the trigger for the rest of the system. Because it serves as an announcement that the player has fired! As we’ve seen in 1.2, the FSR gives out a voltage that represents the pressure exerted on it. The PIC takes that voltage in as an ADC input and reads it continuously. Pin0 was assigned to the value of the force sensor. When the value fluctuates above and below the threshold the PIC reads the value of the accelerometer and sends it to the other PIC for further processing.

The way the triggering works around the threshold is the same concept as the schmitt trigger. When the output of the force sensor is above the high threshold (>2.5), then the system goes into “Ready” mode. It stays there as long as the sensors output value stays greater than the low threshold. The instant that the output value is less than the low threshold (<1) then the system goes into “Go” mode. After this fluctuation occurs, launching is assumed to have happened and further actions are taken accordingly.

The way the high threshold was determined was through trial-and-error. We had many volunteers test out our force sensor. Their readings were recorded and averaged. Volunteers belonged to various age groups to ensure adequate thresholds that would enable multiple players of different ages and sexes to enjoy the game.

### 2.1.1.2 Camera Code

The camera communicates with the PIC using I2C. Before any data can be obtained from the camera, it needs to be initialized. So first, the PIC must send the camera’s initialization sequence and wait for an ACKNOWLEDMENT from the camera.

“Send 0x04 to Output Report 0x13

Send 0x04 to Output Report 0x1a

Write 0x08 to register 0xb00030

Write Sensitivity Block 1 to registers at 0xb00000

Write Sensitivity Block 2 to registers at 0xb0001a

Write Mode Number to register 0xb00033 (Mode number: basic =1, extended =3, full =5)

Write 0x08 to register 0xb00030 (again)”

Now that we’ve received an ACK from the camera, we still have one more thing to do before we can start obtaining meaningful results. Coordinates at this point are still messy and all over the place. We still need to calibrate our camera. Meaning; we need to give it a reference area to give coordinates according to it. That was done through software we developed, where you would initially give the camera four points that would be regarded as the four “edges” of its “screen”. Then, every coordinate it reads from then on would be compared to those edges and mapped within those boundaries accordingly.

Now that the camera is initialized and ready to be used, we built a C-code by defining a READ-IR command on the PIC side that is triggered by the firing of the arrow, and send the result for further visual analysis to the PC side.

The results at this stage are three integers representing x,y coordinates and the blob size (s), in the following order:

• If no IR source is present: x = 0, y = 0, s = 0

• Else one of the three values would be not equal to zero

After this step, the C# game would have the coordinates of the location of the IR LED upon launching. Further analysis and mapping would be performed at the PC side.

### 2.1.1.3 Accelerometer code

The accelerometer provides values of acceleration for all three axes. When the PIC detects launching (through the varying values of the FSR), the PIC reads the values of the accelerometer through the ADC. The accelerometer outputs different voltage levels to each axis depending on the amount of tilt present. These voltage outputs for x, y and z are used as inputs to the PIC ADC Pin1, Pin2, and Pin3 respectively. So the PIC reads the ADC values and then we use the equations from section 1.4 to determine the angles in different directions.

After angles are calculated and are ready to be used. They get sent wirelessly.

### 2.1.1.4 Wireless Communication

Our XBEE wireless component is used to communicate with the other PIC microcontroller. It sends the values that the IR camera captures (the x and y coordinates of the IR LED which represents the position of the arrow) and the accelerometer values post processing (after the exact angles have been calculated from the tilt values obtained in various directions as explained earlier).

Since the wireless component will be sending different data that belong to different other components, we had to define a protocol so the other PIC microcontroller would be able to understand and interpret the data it receives and know to which component it belongs since we can only send and receive a character each time.

The way we went about doing that was by defining a way of communication, w protocol of some sort that the wireless components use to describe the data. Before the wireless component starts transmitting data, it first needs to send a letter that indicates the origin of the values to come. A stands for accelerometer, and C stands for Camera. After the initial identification has been made, it then needs to send a letter that refers to which directions are the values to come are in respect to. Naturally, x would mean X-Axis, letter y represents the Y-Axis and finally, letter Z represents the Z-Axis. Those letters are used to signify the different axes whether we are communicating IR LED coordinates or accelerometer angle values.

Each value (coordinate or angle) will be considered to be consisting of three digits, and so, the three digits following the initialization would be used as the value of the coordinate/angle in that respective axis.

## 2.1.2 Second PIC

The second PIC controls the other XBEE wireless component that receives the data from the first PIC microcontroller. It receives the coordinate data and the accelerometer data post processing after angles are obtained. It is connected serially to the host PC where the C# game is present. It sends the coordinate and accelerometer angles to the C# game. The game does further calculations to determine accurate projection.

The PIC communicates with the host PC serially. It sends the data through a serial connector and the C# game receives that data and analyzes it properly.

# 2.2 C# game

The game was developed using C#. The code also includes computations for the values received from the camera and accelerometer. It uses the IR LED coordinates, the accelerometer values and the distance between the player the target it is aiming at and computes the exact location of where the arrow would presumably hit the target. It then performs actions accordingly. Exact actions would be explained in details as we go through the game in steps.

## Game Flow

We have two different games implemented that employ the same hardware. Our games are designed to be highly interactive and to fully employ the hardware built. Instead of the usual approach of using a mouse to play and choose modes and other things, we use the bow and arrow to enable the player to interact with the game. The player gets to choose which one he likes to play by firing the arrow directly at the desired games symbol.

The game will be described in details. Snapshots will be utilized to clarify all aspects.

When the game first starts, the user needs to choose which game he would like to play.



Figure 2.2.1Archery

### Let’s assume for now that the user chose the first option, Classical Mode.

### 2.2.1 Classical Archery

In the first game, the player is exposed to the classical game of archery. In the classical mode, the player would be aiming at rounded target boards with a certain set of scores allocated to each different part. The nearer you get to the inner circle, the more points you get. The outermost circle is awarded a score of 10 points. The innermost circle is awarded 100 points. The targets the player would aim at have the same look and feel of real archery targets as shown in Figure 2.2.1.2. The scoring system is also the same.

Figure 2.2.1. Targets Look and Feel in the First Game.

In the classical mode, we have three different levels (shown in Figure 2.2.1.3). The difference between the three is in the distance between the player and the target. Of course the further away the player is from the target the more difficult that level is.



Figure 2.2.1.3Level Selection



Figure 2.2.1.4 Classical Mode Levels

The player cannot proceed from one level to the next unless they complete the previous one successfully. As you can see in Figure 2.2.1.4, the levels differ in virtual distance between the player and the targets. The player has 5 shots for each level. The player needs to score 200 points or higher to qualify to make it to the next round. Otherwise, they will have to play again.

When the player gets through all 3 levels he wins!



Figure 2.2.1.5 Trophy

There are “extra” advanced levels displayed in Figure 2.2.1.6 that can only be unlocked if the players accumulative scores from the previous 3 levels are above 900 points!

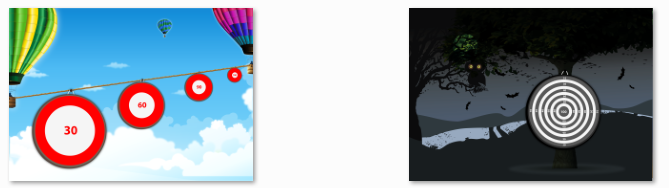


Figure 2.2.1.6 Extra Levels

Now, what if the player had chosen the freestyle option?

### 2.2.2 Freestyle

In Freestyle is where we adopt a more outside-of-the-box approach. In this game, instead of doing your regular archery game, we use the same hardware but add a little twist to it. The player wouldn’t aim at target boards, but instead on characters that appear for only a short time, then disappear again.

We use types of characters to make things a little bit more interesting. You have the “good guys” and you have the “bad guys”!. Naturally, you have to hit and wound the bad guys to get points, but if you by accident hit and wound an innocent bystander then points will be deducted away from you.

The bad guys are in the form of pirates, very mean-looking pirates. And the good guys are in the form of ordinary people, smiling and standing around.

The rules of this game, first of all as we’ve already mentioned, is that you should not hit any of the good people, and if you do, points will be taken away from you. Objects stay on screen for a certain amount of time, and if you fail to hit them within the allocated time, they will disappear.

The game ends when your points reach zero; you keep wounding good people and getting your points taken away from you. Or, when you point count reaches 100, then you have won.



Chapter 3: Work Flow

The game starts when the player turns on the system and decided to give our game a try! When the system is on, both PIC circuitries are up and running. The wireless devices are also working and are now waiting to start communicating with each other. The player starts the game, decides which mode would he like to play (classical or freestyle). And then, the user grabs the bow and arrow and starts pulling the arrow back and aim it towards the target. The force sensor is positioned in a strategic place; the player *has* to apply pressure upon the sensor while he’s aiming at the target. And then when the player is confident about their aim, they fire the arrow. Launching is detected; here is when the magic starts.

When launching is detected, the PIC microcontroller reads the values of the accelerometer and the IR camera to get x and y coordinates of the IR LED at the moment of launching. It then uses the equations described in the accelerometer section to obtain angles. When those values are ready, it starts transmitting the values of the IR LED coordinates and the angles calculated from the accelerometers tilt values.

When the second PIC receives the results through the XBEE wireless component at its side, of course first it needs to sort out the data it had received and separate the x and y IR LED coordinate values from the x, y and z accelerometer angle values.

When that data is ready, it sends it through the serial port to the PC host through a C# code that is part of our game that is also developed using C#.

When the C# game receives all of that data, it maps the IR LED coordinates to their appropriate location within the screen. It then incorporates the accelerometer angle values and the virtual distance between the player and the target to calculate accurately the projection of where the arrow is supposed to hit onto the target board (or people in case the player was in the freestyle mode of the game).

After the projection of the location of the arrow is calculated, the appropriate score is awarded to the player, and their accumulative score would be updated respectively.

After that, the player advances in the game according to his scores and how well he can play the game.

The freestyle game brings a new and fresh content to the player and provides a more interactive approach and feel to the game. It is engaging in a sense that the player feels that he’s shooting the “bad guys” and saving “good people”. It would be a nice change from the classical mode of archery and target boards.

Chapter 4: Problems and Solutions

# Camera

The camera was our number one challenge. Because we were extracting the IR camera from a Wiimote and not buying a regular IR camera, we were not able to find sufficient information on the matter. We had to purchase 3 different Wiimotes until we found the exact model that worked with our system as other Wiimote models (although original) didn’t function properly and never replied with an ACK =0 (represents a correctly functioning camera).

The Wiimote was really challenging to deal with in terms of disassembly. Taking the Wiimote apart and desoldering the camera was a real challenge! The camera was firmly attached to the Wiimote board. And soldering points were really intricate and hard to get to and desolder. Plus, we had to keep in mind not to expose the camera to too much heat to make sure not to damage it as it is very sensitive and delicate.

IR-Camera’s pins are hard to plug in the holes of a regular breadboard (pins are small and too close); since the IC itself doesn’t follow the typical DIP pins that are usually used for breadboard prototyping. So we had to be extra careful when we were soldering the camera pins to longer socket pins for ease of use not to short-circuit any two pins together.

As mentioned earlier, since the IR camera is extracted from a device, there wasn’t sufficient documentation on how to use it, how to calibrate, and how to map the obtained values to the screen to show results. We had to do extensive research to find some relevant data that would guide us and give us some insight on how to go about doing that. The person that did some very remarkable work in determining the camera pin out and how to do the preliminary initialization was Kako, a Japanese guy passionate about technology. His work focused on the early steps of using the camera, he explained how to initialize the camera to the different modes it can operate in (the extended mode we are using is one of the available modes). His work was extremely helpful in obtaining raw camera data for the IR coordinates within the range of view of the camera.

Making sense of the data obtained from the camera was another challenge. The best way that enabled us to bring meaning to our data was by setting our own reference points. We would set 4 points that would resemble the edges of the screen we would map our data onto. And then, all the obtained data would be processed in a way to be within the range of view of those four points and would be relevant to them.

Another problem with the camera is that it is very sensitive to the light. Since it can detect up to 4 IR points, it often times detects noise in the room and regards them as points and sends their coordinates. The way we solved this problem was through extracting the glass filter within the Wiimote and attaching it to the camera. That filter ensured that only the IR LED would be detected and by employing the filter we minimized the noise considerably as it approached zero.

The SDA lines kept reading 0xFF (“255” a permanent logic high) from the pins of the IR-Camera, which would indicate that the values are not valid. Our assumption for the cause of the problem was the voltage level difference between the PIC and the IR-Camera, since the first operates correctly at 5V and the camera at 3.3V. Both the SDA and SCL lines from both sides should be pulled up to Vdd, but since we have two levels (5V and 3.3V) we decided to pull them up to 3.3V; since the IR-Camera’s pins are not tolerant to 5 volts, we went with the lower voltage as not to burn the camera. So at first, we changed the standard I2C pull-up resistors from 4.7K to 1.5 K. Unfortunately contrary to what we were expecting, SDA lines keeps reading 0xFF. Our next approach (which fortunately proved successful), we built a bi-directional level shifter N-Channel MOSFET circuit between the PIC (5V device) and the Camera (3.3V device) as recommended by PHILLIPS (which by the way established the protocols of I2C communication).See Appendix A for the shifter schematic circuit described in details.

# XBEE Wireless Component

The first problem we encountered was that the only shop in Nablus that sells electronic components didn’t have any more wireless components in stock. After asking and searching for quite a time, we were finally able to find a shop that had a pair in Ramallah. After going to Ramallah and purchasing the components, we went to the electronics shop in Nablus to buy boards to operate the wireless components and test them out. They were faulty! They were non-functional. We tried to “fix” it. Our first approach was to try and copy the framework from a functioning component and write that data to our faulty component in hopes that it is a software issue. At first, it didn’t work. After a few days, we tried a newer version of the same software we used to copy and write the data, and it works on one of the components (but was slower than usual). The other component was completely dysfunctional and didn’t work.

Naturally we needed two components, a sender and a receiver and up until this point, we only had one of the two. Fortunately, a group in our department had an extra component and they offered that we use it. Of course we did, and now we had the two components we needed to ensure proper functionality.

# Accelerometer

We are using the accelerometer to get angles, and not just detect acceleration in various directions. Most of the applications and documentation online do not address that issue. In addition that previous work done at our university didn’t involve extracting angle values from acceleration data. We had to look at various work and papers and see different formulas that people used to get angles. Another thing was that they were using a type of accelerometer that is not available here. They use a different one because it is more sensitive to change and gives more accurate results. At the end, we were able to finally have an equation that catered to our needs and enabled us to acquire the angle results accurately and are fitting to our requirements.

# Lab Hours

It was probably one of our biggest challenges for the hardware project this semester. Most of our work needs to be done at the lab, because we don’t have power supplies or other equipments that are only available at the lab. The TA’s officially finish work at 3:30 pm every day, sometimes they would stay 30 minutes longer because we ask them to, but eventually, they have to leave (and rightfully so). The way we tried to minimize the effect of this was to use batteries and regulators to get the voltages we needed, but batteries drained super fast. And there are equipments that we simply couldn’t substitute at all.

We tried to minimize the effect of this, but there was no true solution to this one particular problem.

Possible Improvements

* We could have a bigger game that is more wide-spectrum. It can contain many different levels.
* The game can also be developed to support multiple players as the IR camera already being used supports detection for up to 4 different IR points (the brightest four).
* In order for players to have a more fun and engaging experience, we can play the game on a projector instead of a PC. The only difference would be the calibration and IR coordinate mapping because naturally the screen would be bigger. But that would provide a better playing experience.
* At the moment, we are only using wireless communication to send the data of the IR LED and the accelerometer to be processed at the PC side. We could, as a future improvement have all communications done wirelessly as at this moment the IR camera is connected through a long wire to the PIC microcontroller.
* More games could be developed that use the same hardware as it is very versatile.

Conclusion

The main aim of this project was to make the game of Archery safer for children and beginners. By using a bow and arrow made of wood with smooth edges, we made sure that players would not harm themselves using the hardware (bow and arrow). Also, we designed the bow and arrow in a way that the arrows base is too wide to go through the base, and thus ensuring the inability of the arrow to fly away, which in turn would make sure of the safety of the player and those around him.

We were able to develop the game using relatively available components. We were also able to find effective replacements of parts that were not available in the local market.

The game is designed in a highly interactive way, it pushes the player to move around and it is very much different from the more classic sedentary video games that children play all the time. This provides a healthier alternative, and a very engaging way for children to play a game that teaches them concentration and patience whilst ensuring their safety.

As a conclusion, I believe we have achieved the goals we were aiming at. We have developed a functioning game of Archery that is both safe, and fun.

References

* <http://www.digi.com/products/wireless-wired-embedded-solutions/zigbee-rf-modules/point-multipoint-rfmodules/xbee-series1-module#overview>
* <http://www.ladyada.net/learn/sensors/fsr.html>
* <http://kako.com/>
* <http://wiibrew.org/>
* <http://www.freescale.com/files/sensors/doc/app_note/AN3107.pdf>
* <http://en.wikipedia.org/wiki/Componentbased_software_engineering#Definition_and_characteristics_of_components>
* I2C Bus Specifications, V 2.1, PHILIPS Documentation.

Appendix A

In this appendix, we will provide a detailed description on how to physically extract the IR camera in the Wiimote and then use it in other circuits with a microcontroller.

**Step 1: Extract the Wii IR Camera**

You'll need a Wii remote or "Wiimote" to start with. Normally they are about 400NIS. Make sure to buy the original one with the extended functionality (not the mini). Because a knock off would be a waste of your money as the camera would not operate according to the pinout and circuitry specified.

You'll need to rip apart the case. Now you need to desolder the 8 pins and the 2 struts holding the camera on the board. You can use some desoldering braid. It wicks the solder away and makes it easy to get the camera out. Be careful, don't break the camera, it is fragile!

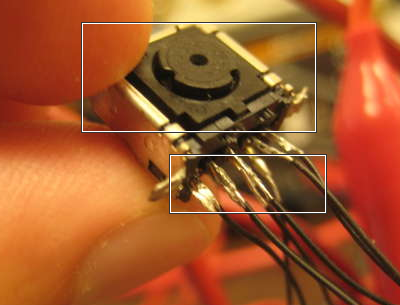


Figure . Wiimote Broken Apart Figure . Wiimote IR Camera

**Step 2: Interface Circuitry**

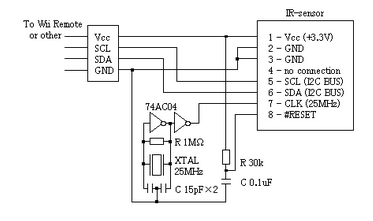
After the camera is extracted and longer pins (we cut a socket in half and used it) are soldered for ease of use, now we need to build the appropriate oscillator circuit to operate the camera (it needs 25Mhz). We connect the components as shown in the figure below.

Figure . IR Camera oscillator Circuit

We'll need a small circuit to interface the IR camera to the PIC microcontroller. The PIc microcontroller uses 5v to operate, whereas the IR camera requires only 3.3v. And thus we need a bus voltage converter so both SDA (Data) and SCL (Clock) can communicate with each other despite the different voltages. The recommended hot swabable voltage converter IC was not available in the local market, so we had to substitute that for the following voltage converter built using MOSFET transistors and other basic components.

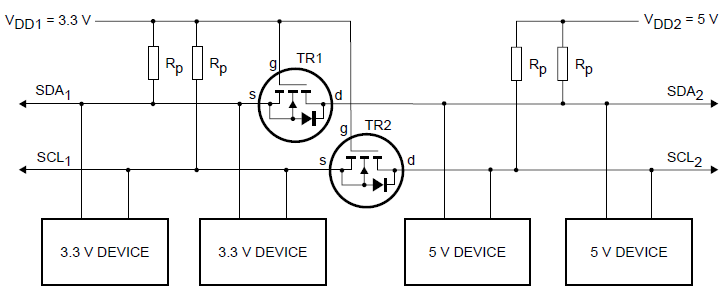


Figure . Voltage Converter

And now, we are ready to write a program to initialize and read the IR camera, obtain the coordinates, and use them to control our system.