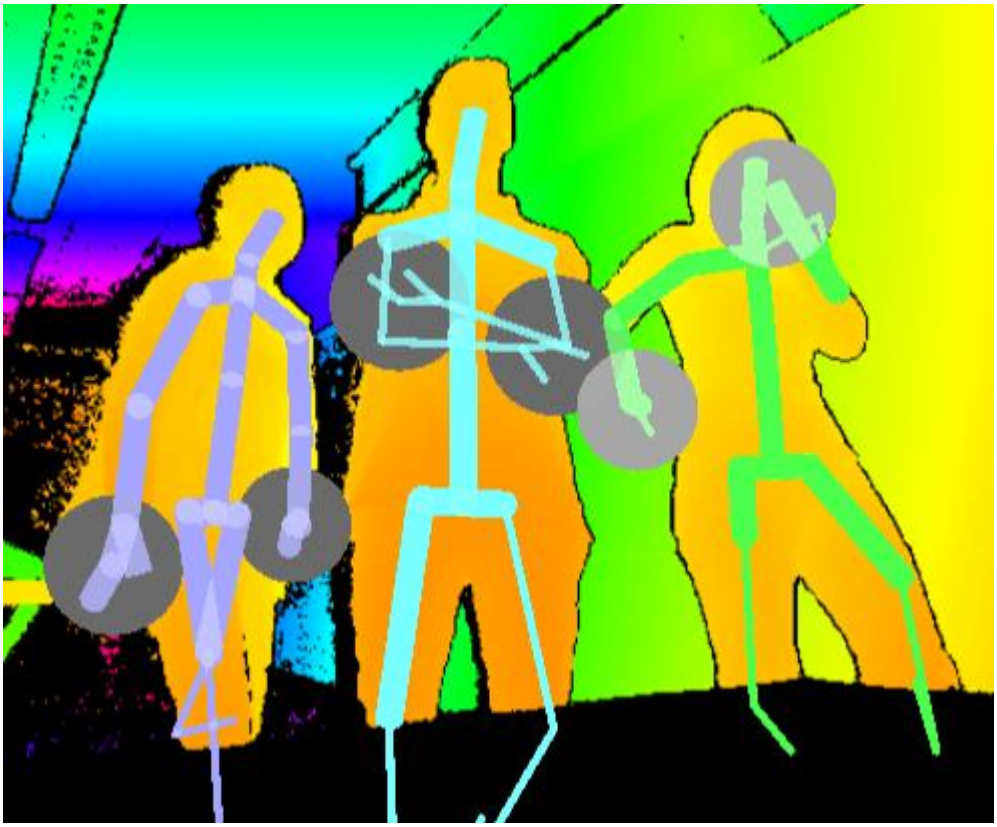
EC535 Introduction to Embedded Systems

Project Exer-Snake

BOSTON UNIVERSITY



Team DownToHack
Tess Gauthier, Hannah Hebert, Duncan Silver
tgauth@bu.edu hhebert1@bu.edu djsilver@bu.edu
Prof. Wenchao Li
May 4, 2017
Project Report, Spring 2017

Abstract:

ExerSnake is modern take on the classic arcade game. Combining exercise and fun, the interactive game that interfaces the Kinect with a Gumstix kernel module. Using four gestures, left, right, up, and down, the user can control the movement of a snake displayed on the LCD, as it attempts to reach the elusive food. The Kinect sends data to a Windows 8 Application running on a PC, which sends data via bluetooth to a bluetooth module connected to an Arduino which is connected the GPIO pins of the Gumstix. The kernel module responds to the inputs from the GPIO pins and updates an array which is then displayed on the LCD using a QT module.

Introduction:

Project Exer-Snake is an interactive adaptation of the classic snake game working to combine physical gestures to control the direction of the snake. The classic snake game originated as an arcade game named *Blockade* in 1976 [1]. Since then it has been released in hundreds of different versions. However, the primary concept has stayed the same where a player guides a line, or snake, around a screen which grows in length as the game progresses.

The goal in this project was to make the game more interactive by requiring users to guide the snake through physical movements including jumping and squatting. As the trend of interactive gaming grows, adapting the classic-style arcade games is imperative to their continue success. In addition, this project allows user to get a workout in while playing their favorite childhood game.

In this project, we included the following modules and features to provide an updated version of the antiquated game:

1. Kinect Module
2. Bluetooth Module
3. Device Driver for Gumstix Controller
4. LCD Display

The Kinect module is used for the gesture reading. The bluetooth module allows for a wireless data transfer to the controller and display allowing for the screen to be located anywhere the user can see. The Gumstix controller is used as the primary source of processing the input data to play the game. The LCD display provides a GUI for the user to visualize the game processing.

With this project, we were able to achieve a functioning game utilizing the Kinect gesture recognition abilities as well as bluetooth functionality. Our module development and driver efficiency allows the snake to move smoothly with no noticeable delay of the movement to the LCD display.

Design Overview:

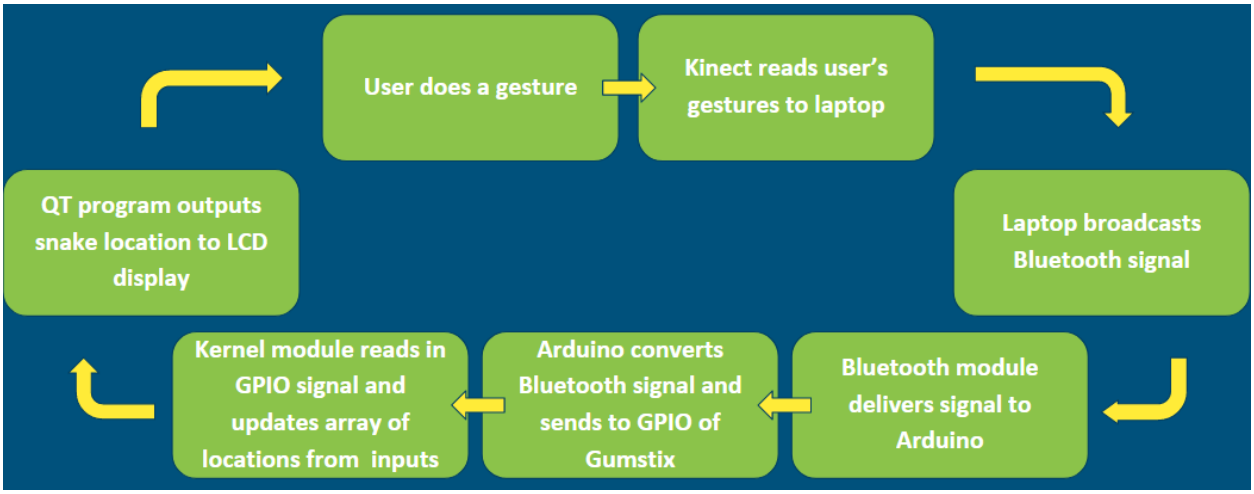


Figure 1: Flow Diagram for ExerSnake

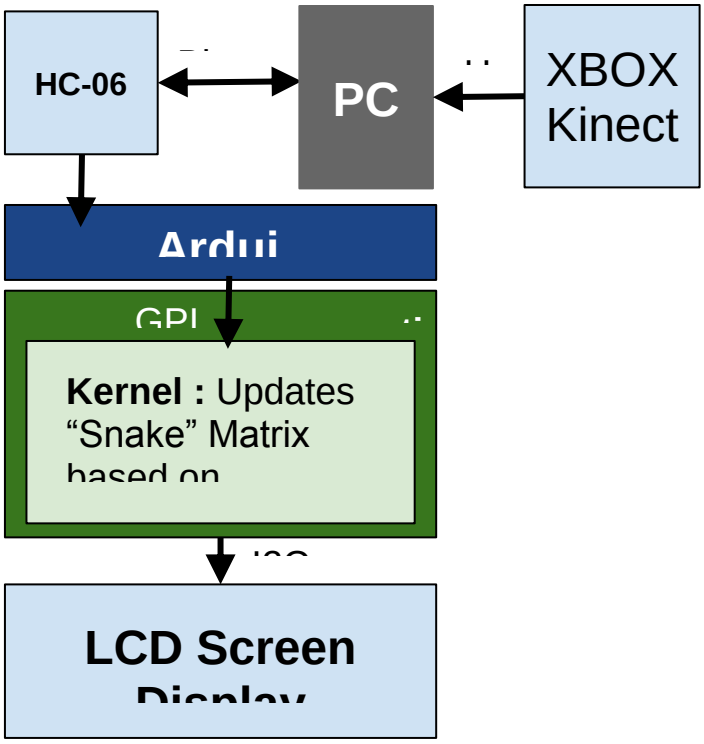


Figure 2: Functional Diagram for ExerSnake

Tess implemented the Windows 8 Application that reads in the data from the Kinect, determines which gesture, if any, was completed by the user, and sends the data via bluetooth to the HS-06. She also implemented the Arduino sketch that interprets the data sent from the bluetooth module and writes digital output pins high and low, depending on the gesture. Duncan focused on devloping the kernel module with the data processing. He worked on receiving the input from the GPIO signals, debugging the kernel module, and then sending the output array to the Qt program.

Project Details:

1. **Kinect Module** *(See /Windows8App for the source code)*

The kinect module is implemented as a standalone Windows 8 Application. It runs locally on the PC that the Kinect is connected to through a USB [2]. The Kinect for Xbox One was used because the Kinect Studio v2 SDK supports it. As the latest version of the SDK, it has a Visual Gesture Builder Preview built into it. This program allows gesture libraries to be built that can be integrated into applications [3]. The Gestures.vgbsoln is in the Database folder of the Bluetooth Communication Sample Controller folder of the Windows8App folder. To develop the gesture library, videos of a person doing each gesture were recorded on the Kinect. Each clip could be imported into the library after providing some basic information about the type of gesture it would be, such as whether it involved the user's hands, upper body, or lower body. For the Gestures.vgbsoln library, AdaBoostTrigger was used for the gesture detection. This detection algorithm returns a boolean; either the gesture is occurring or it is not. The clips imported into the library had property fields that could be updated to "true" when the intended gesture was occurring, at different times throughout the clip. This is what the algorithm bases its detection on. After all of the gestures have been imported into the visual gesture builder project, it can be built to create the Gestures.vgbsoln. Using Live Preview, it can be shown how the library handles input gestures from the Kinect in real time, before the library gets imported into a project. Some difficulties occurred with gesture recognition, as some of the gestures were too similar, initially, so they had to be modified. For example, the up gesture was originally raising both hands above the person's head. However, the raising of arms was similar to the left and right gestures, so this gesture was replaced with a person simply jumping up and down once, keeping her arms down. The final library was copied to the folder of the Windows 8 Application. After adding it to that folder and importing it to the Visual Studio Solution, its settings needs to be updated in the file properties section. The Build Action field was set to Content and the Copy to Output Directory field was set to Copy Always [4].

The Windows 8 Application that interprets the input data from the Kinect, determines if it matches one of the gestures in the custom gesture library, and if so, sends the corresponding character to the bluetooth module is Bluetooth Communication Sample Controller [5]. The license for the original project is in the Windows8App folder [6]. The project was found as an example for Bluetooth Communication between Arduino and Windows 8.1, after research was conducted about the best way to transmit information from the Windows application to the Gumstix. Establishing a connection with the Gumstix directly from the Windows Application proved to be a difficult task. The Windows Application was needed to be Windows 8 to support the Kinect, as Windows 10 and UWP applications do not yet have Kinect support, but the bluetooth functionality

and documentation for Windows 8 as opposed to UWP applications proved to be more difficult to implement. The team opted to go use a bluetooth module connected to an Arduino as a more suitable implementation. Using the sample project as a starting point, the parts not applicable to the project, like most of the UI, were removed, and the code to read in data from the Kinect was added.

In MainPage.xaml.cs, the kinect sensors and data are all initialized. In the “Get Data from Kinect” region, the gestures from the imported gesture library are added to the frame source. Then, as frames come in from the Kinect, they are compared to all of these gestures. If a gesture matches, its corresponding character is set as the bluetooth character. If this character is different from the previous bluetooth character, then a function is called so it is transmitted. The BluetoothConnectionManager.cs handles the the transmission. It establishes a socket and establishes a connection when a “HS-06” device is recognized, and the connect button has been pressed on the UI.

2. **Bluetooth Module** *(See /ArduinoCode/ArduinoCode.ino for the source code)*

The HS-06 receives the data from the Windows 8 Application and uses direct connections to the RX/TX on the Arduino Mega 2560 to transmit the data. The ArduinoCode folder contains the sketch that is uploaded to the Arduino. Using Serial to read in the data, it then sets a digital output pin high if it is one of the recognized characters. The characters are sent as a string so their ascii values are used in the the Arduino code. The digital output pins, one for each gesture, connect the Gumstix GPIO pins. Initially, all the pins are set to low because the user has not done any gesture yet. At any given time, only one of the four pins will be set high, and the rest are set to low. The pin that is set high alerts the Gumstix that the corresponding gesture is being completed by the user. The pins are set when the incoming character differs from the previous character. The arduino receives a 1 after every transmitted character so this was factored in, when keeping track of the last character sent via bluetooth.

3. **Circuitry**

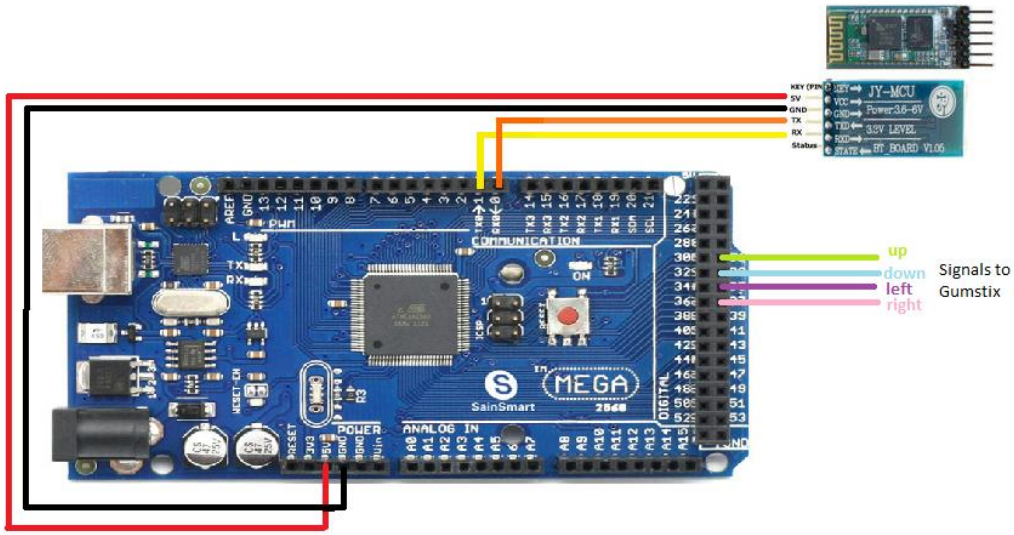


Figure 3: Bluetooth Module Inputs and Outputs

The bluetooth module receives a character sent from the Windows 8 App, regarding the gesture detected by the Kinect. The 5V (red) and GND (black) are connected to the Arduino, as well as the TX-RX (orange) and RX-TX (yellow) from the bluetooth module. The digital pins 31, 33, 35, and 37 on the Arduino are set as outputs. A high signal indicates to the Gumstix Kernel Module that the corresponding gesture is

being performed.

Figure 4: Complete Circuitry Set-Up for ExerSnake

4. **Kernel Module for Gumstix** *(See /km/mysnake.c and /km/Makefile for the source code)*

The device driver is written using a c-program kernel module. For our project, we used the kernel module program to complete our data processing as it was not an intensive program and no delays were noticed in processing the data similar to the fifth lab in this course. We did not utilize a user-level program since there was no input needed from the user in the command window. If our data processing was more intensive we would have analyzed the data in the user-level program or in the Qt application itself.

The code begins with including all necessary header files and declaring the kernel module functions. Our kernel module was split into four functionalities: initialize, read, write, and exit. In addition, we declared our timer structures, function handles, and global variables here. We utilized a single timer named “snaketimer” to set the period at which the snake moves. It was set to approximately 700ms allowing the user time to make physical gestures easier. For our user-level functions, we created functions to generate random points within the user-space for food and to reset the screen array of values. We built an array to store the location of the snake, the position of the snake’s head and tail, the gesture direction. We also utilized our GPIO functionality from lab 5 to receive the signal from the Arduino. We used the interrupt handlers to improve the functionality of the program [7]. Utilizing these GPIO I2C pins, we were able to interchange the Arduino with the button module for testing purposes. For this reason, there is a debounce timer to prevent pressing the button multiple times.

In the initialize function, we allocate memory for the buffer space and our timers. In addition, this is where the initialization of many of the global variables occurs. For example, the array for the snake location is set up so that it initially contains all zeroes. The snake location is designated with a one and in this case placed in the center of the screen. The food is randomly located and signified by a two. Finally, the border is represented by a three. This section is where the GPIO pins are set as inputs and their interrupt functions are called. The snaketimer and debounce timer also setup here to begin counting.

The write function was originally going to be used to have the signal input for the gesture be written to the command space. The kernel module would have then obtained this data and sent it to the Qt program. However, after many issues with our bluetooth module for the Gumstix, we switched to using the Arduino to receive in the data from the bluetooth module. This allowed us to more simply use the GPIO pins instead.

The read function is used to print the points of the snake and the food to the user space. Each time the snake timer expires, the system updates and prints these points to the user space which allows the Qt space to only update these points.

In the exit function, we free the memory that was allocated for the buffer, timers, and interrupt handlers. Doing so allowed us to successfully remove the module without having a kernel panic.

The snaketimer callback function is where the majority of signal processing occurs. Figure 5 shows the logic behind the callback function to update the array.

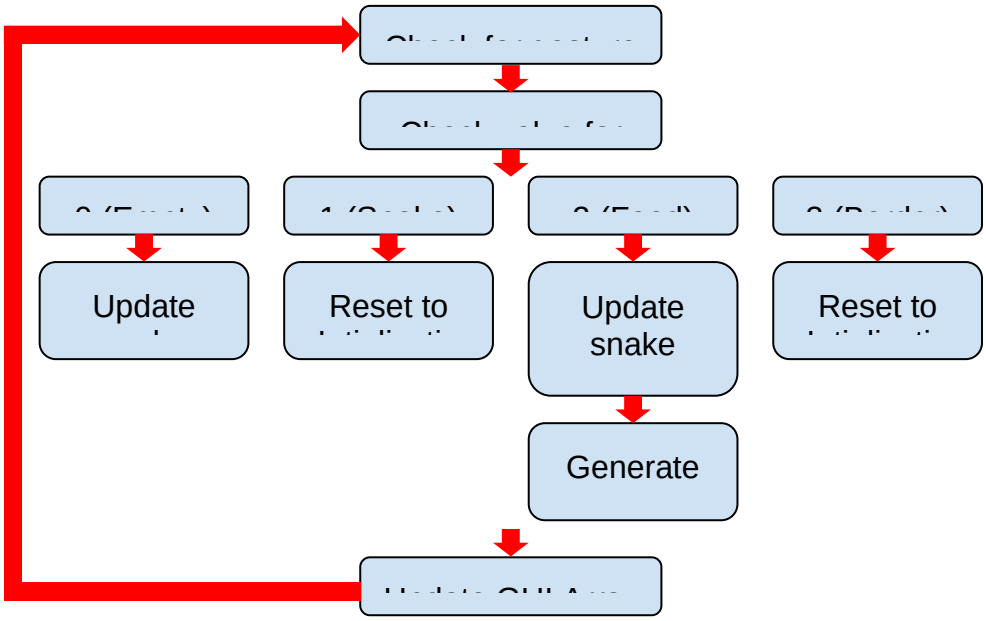


Figure 5: Flow diagram of snaketimer callback function

5. LCD Display

The LCD display works by continuously reading from the kernel module. The kernel module's read function transmits a vector of points that contain relevant information (i.e. the snake coordinates and the food coordinates). As the kernel modules runs, the qt function uses the cat /dev/mysnake command and reads the output into a lcd.log file. The function then parses this file, storing each point being sent from the kernel module into a vector of stored "QRectangles". The first item in this list is reserved for the snake "food" and is painted a different color than the remaining snake points. These rectangles are printed to the screen each time the list is updated. The program creates a new paint event for each new data list, effectively refreshing the screen. The rate of update is determined by the kernel module, refreshing based on the period set for the snake movement.

Summary:

This project allowed us to develop a simple yet fun adaptation to the classic snake game. We were able to interconnect the Kinect gesture recognition, Bluetooth module, the Gumstix controller, and an LCD Display. We were able to improve efficiency of the code in order to limit delay in signal processing and have more control of the period of the system. We developed a simple user interface and stable gesture recognition.

One of the major challenges for this project was Qt application interaction with the kernel module. In order to improve functionality, we could probably have moved this signal processing to the Qt application and eliminated some errors due to an overflow of information having to be passed to the application. However, in its current state the application is stable and meets our speed efficiency standards.

In addition, we gave ourselves the ability to add more functionality to the game if wanted such as a changing period as the game progresses and more GUI display types (i.e. different shapes, colors, user prompts…). Our current design was setup to primarily display functionality and successful implementation of the different modules and signal processing.

“We’re gonna need a bigger boat” - Noah
“We’re gonna need a bigger LCD” - Team DTH

References:

[1] "Blockade [Model 807-0001]." Blockade Arcade Pcb by Gremlin Ind, Inc. (1976). N.p., n.d. Web.

[2] "Kinect 2 for Windows Demo App." *Kinect 2 for Windows - Hands On Lab 1*. N.p., n.d. Web.

[3] Sugiura, Tsukasa. "How to Build Gesture Database (Discrete)." *YouTube*. YouTube, 10 Sept. 2014. Web.

[4] "Kinect 2 for Windows Demo App." *Kinect 2 for Windows - Hands On Lab 12*. N.p., n.d. Web.

[5] "TechTalk 13 - Kinect Visual Gesture Builder." *YouTube*. YouTube, 12 Oct. 2014. Web.

[6] Osthege, Michael. "Bluetooth Communication between Arduino and Windows 8.1." *Bluetooth Communication between Arduino and Windows 8.1 Sample in C#, XML for Visual Studio 2013*. Microsoft, 13 Nov. 2014. Web.

[7] "Chapter 10: Interrupt Handling." Computer Communications and Networks The Quintessential PIC® Microcontroller (n.d.): 185-212. Web.