Embedded Pong

Ellen Lo (*lowing*) and Gene Grinberg (*genegrin*)

Abstract

Our project is a single player pong game that runs on an LCD screen connected to the Gumstix board. The paddle in the game is controlled by the players hand, moving up and down in front of a Kinect.

Introduction

Embedded Pong aims to bring an alternative Pong game experience by using motion as control. We wanted to challenge traditional game controls, such as joystick or touchscreen, which typically encourage players to stay inactive and focused on TV or computer screens. Hence, we decided to experiment with motion as game control.

When brainstorming what to do for this project, we both knew we wanted to make something to do with gaming, since that would make the project very fun for us. The example project topics said we could implement a classic arcade game, but we wanted to add a twist to it. We had the idea of using a sensor to make it a motion controlled game, and therefore decided to make pong because it made the most sense to be a motion-controlled game.

Originally, we had many ideas for our implementation. We considered making the game 2-player, have a motion controlled menu interface, and be projected onto a bigger screen of some sort. However, we were advised to cut down the features, as we would not have a lot of time to implement this. That is why we decided to make the game single player, and have just the game in the UI without any menus, etc.To accomplish this, we had to come up with a detailed plan. Our implementation plan includes getting data from Kinect motion sensor, implementing gameplay on LCD screen through Gumstix, and facilitating the communication of motion data into hardware. In the end, we created a fully functioning 1-player Pong game that is entirely controlled by players' hands in front of a kinect.

Design Flow

Our project is divided into tasks handled by client and server respectively and the communication between the two.

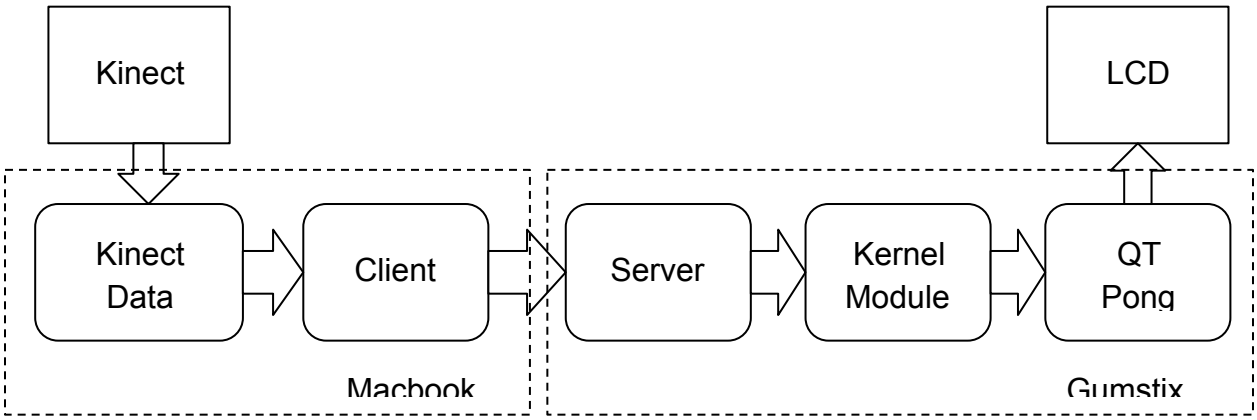
The client side, implemented by Ellen, processes data from Kinect.

The server side, executed by Gene, involves creating the interface on LCD screen with QT and handling the logic of the gameplay.

The communication between client and server, also implemented by Ellen, uses Ethernet connection to send data to Gumstix.

Overall contribution:

Ellen Lo 50%, Gene Grinberg 50%.



Project Details

a. Processing of Kinect data (see glview.c)

The code initializes the Kinect motion sensor, opens up a desktop application on the PC, and creates a pthread which continuously takes in data, processes the data that correspond to the closest object to sensor and send them over to the hardware (explanation in part e).

If a Kinect sensor connection is detected, a desktop application, shown in Figure 1, will open up on the PC so that the user can always make sure if they place their hands at a detectable distance from the sensor. The left shows the depth map of the video, with red being the closest object detected and blue being the furthest; the right behaves like a regular webcam. The desktop application is powered by OpenGL.[2]

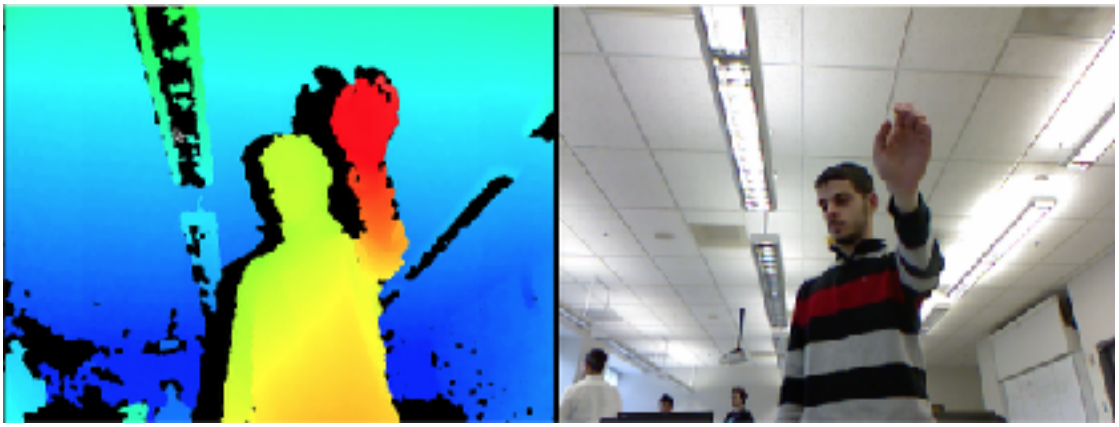


Fig 1. Desktop application that shows depth and rgb values of video

In the callback function of getting depth data of video, we sum up the red data points (closest detected) collected in each frame, and take the average of the sum in order to obtain the approximate position of hand in space. The averaged value, which represents the relative distance between user's hand and the highest possible area detected, is then mapped to a range from 0 to 280 and sent to the hardware /dev/mygpio file (explained in part e).

b. Reading from user space in kernel space (see mygpio.c)

This is a simple kernel module written in order to have the user control the pong paddle. The only functionality the module has is writing and reading a number from a dev file. This number is written to the file by the server module, and is read by the QT pong game.

c. QT Pong Game (see /Pong, specifically /Pong/gameplay.cpp)

The game is a QT implementation of the classic pong arcade game. It is a continuously running game, with a single number for scoring. Whenever the player scores this number goes up, and whenever the computer player scores this number goes down. The game UI itself has three main components: two paddles and a ball. The paddles move up and down and the ball bounces between them and the walls.

The pong game is based on a game tick update system. What this means is that there is a timer running that expires every 12 milliseconds, constituting a tick. Each time the timer expires, it calls the update function - called tick() - which handles most of the game logic. In this function, the program calculates what the direction of each of the three game objects should be and moves them in that direction. This involves several steps.
First of all, it detects collisions. If the ball hits the top wall, bottom wall, or paddles, its y-direction gets inverted. If it hits the left or right wall, its x-direction gets inverted. Next, it calculates the direction of the computer paddle. To make the paddle not perfect, it moves with a probability of one tenth each game tick. Most importantly, this function calculates the direction of the users paddle. To do this, it reads the number from the dev file. If this number is more than the value of the current position, it moves the paddle down, and if the the number is less it moves the paddle up. This is how the user controls the paddle.
After all of these calculations are made, all of the game objects' positions are incremented or decremented using the moveBy() function. [4][5]

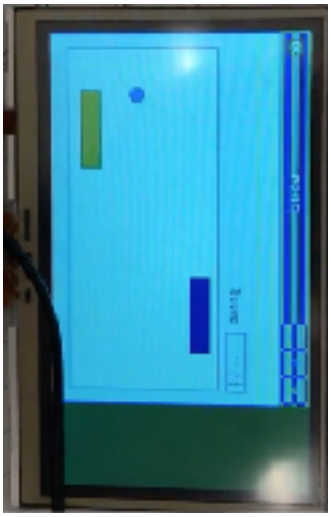


Fig 2. Pong Game UI

d. Socket programming (see glview.c and /server/server.c)

This part of the code handles the ethernet communication from PC to the gumstix. We chose ethernet instead of other communication methods such as Bluetooth or WiFi as the project requires almost instantaneous feedback to facilitate synchronized direction for the ball in the game. We set up a server on the gumstix and client on the PC. As long as the server is running, it takes input from the PC.

Ethernet connection must first be configured on both PC and gumstix before data communication. On the gumstix, /etc/network/interfaces needs to be edited to enable eth0 and declare IP address and subnet mask. On the PC, network's IP

address and subnet mask have to be manually set as well to establish connection.[3]

In /server/server.c, a socket is created and it is bound to the address configured on the hardware. Once the server is set up, it waits for input from the PC (client). Everytime an input is received, the server opens up /dev/mygpio and reads and copies the buffer into the file.

In glview.c, where the setup of the client is embedded into the Kinect data processing component, socket is created and connected to the IP address of the hardware during initialization. The processed value is then passed into sendData() function to send to server.[1]

e. Hardware setup

Figure 3 describes the hardware setup of the game.

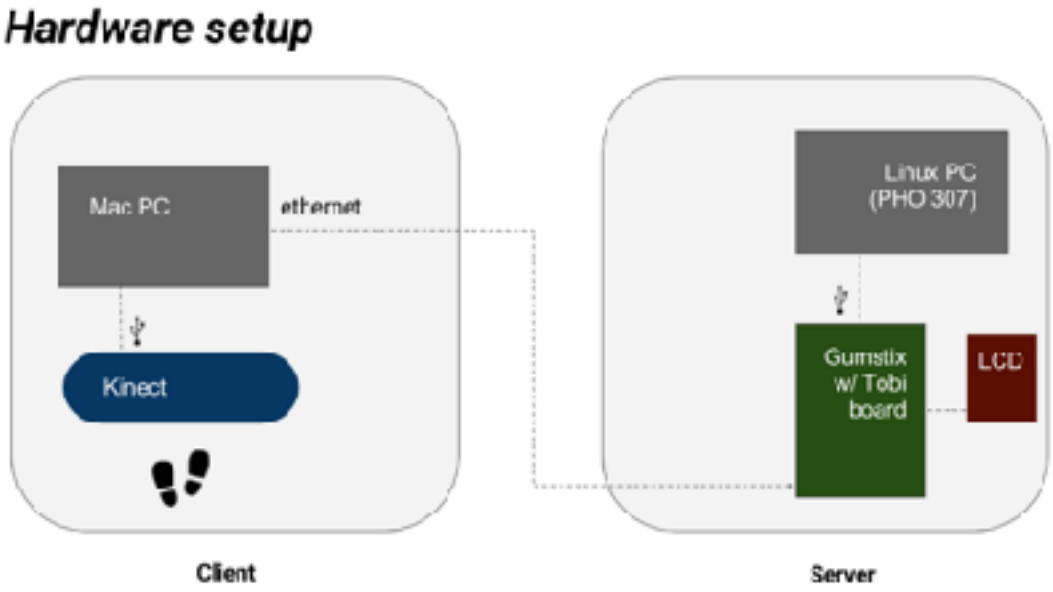


Fig 3. Diagram describing hardware setup

Summary

In conclusion, we both accomplished a lot, and left a lot of space for further development. We were successful in implementing a kinect motion controlled Pong game on the Gumstix board. To do this, we had to write programs that would process data from the kinect, send it over ethernet to the gumstix, and pipe it to the QT game through a dev file using a kernel module.

There are some improvements that can be made to our project. For example, one thing that can be done is smoothing out the movement of the paddle, making it easier to control. Also, there are a lot more features that can be added to improve this project, such as a menu GUI for the game and two player functionality. That being said, we are proud of our accomplishments working on this project.

Reference

1. Socket Programming in C/C++. http://www.geeksforgeeks.org/socket-programming-cc/
2. Documentation of libfreenect, open source, cross platform library built for Microsoft Kinect sensor. http://docs.ros.org/kinetic/api/libfreenect/html/
3. Linux ifconfig command. https://www.computerhope.com/unix/uifconfi.htm
4. Example of Pong game used as template https://github.com/ynonp/Pong
5. QT Documentation http://doc.qt.io/qt-5/reference-overview.html