Remo - Kinect based remote control application for XBMC

Introduction

Kinect is a line of motion sensing input devices by Microsoft for Xbox 360 and Xbox One video game consoles and Windows PCs. Based around a webcam-style add-on peripheral, it enables users to control and interact with their console/computer without the need for a game controller, through a natural user interface using gestures and spoken commands. Kinect sensor is based on Prime-Sense technology and there are similar sensors on the market such as Asus Xtion and PrimeSense Carmine.

XBMC is a free and open source media player developed by the XBMC Foundation, a non-profit technology consortium. XBMC is available for multiple operating systems and hardware platforms, with a 10-foot user interface for use with televisions and remote controls. It allows users to play and view most videos, music, such as podcasts from the internet, and all common digital media files from local and network storage media.
in our project we were asked to create a remote control application for XBMC utilizing the Kinect sensor capabilities.

Project Objectives

A good remote control application must have the following characteristics:

- Intuitive
- Responsive
- Short learning curve
- Well defined gestures

We will later explain why our application meets the following requirements.
It is important to note that the main focus of this project was to create a full and working application. Because of a limited time frame when possible we used existing libraries (e.g. Fizbin) which saved a lot of development time and simplified the interaction with the sensor.

Project Infrastructure

The project is A WPF(Windows Presentation Foundation) application, based on Microsoft Kinect SDK 1.7 and written in C# under Visual Studio 2012, and relies on the following libraries:

Microsoft.Kinect.Toolkit - This is the Microsoft kinect toolkit library which provides means to connect to a Kinect sensor, and access to the 3 main streams a Kinect sensor provides:

- Color Stream
- Depth Stream
- Skeleton Stream

which will be explained in more detail in the implementation section.
Fizbin.Kinect.Gestures - The Fizbin Gesture Library, for Microsoft Kinect for Windows, provides a simple and straight forward means of recognizing both static and dynamic gestures. Based off code described in Writing a Gesture Service With the Kinect for Windows SDK, by Michael Tsikkos and James Glading, the library has been updated with support for SDK 1.5 and extended.
for more information about how fizbin works see:

http://blogs.msdn.com/b/mcsuksoldev/archive/2011/08/08/writing-a-gesture-service-with-the-kinect-for-windows-sdk.aspx

Microsoft.Kinect.Interaction - This is a part of the Microsoft Kinect SDK which provides access to the kinect sensor Interaction Stream, which enables hand tracking and hand open/close recognition.

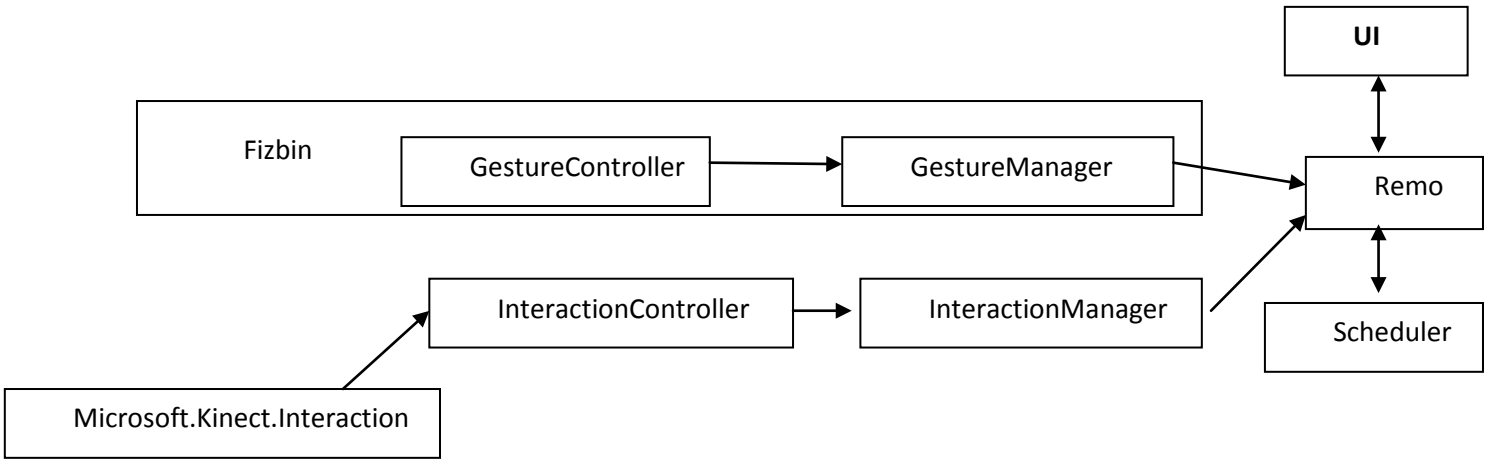
Design

a standalone application

Being a popular open source application, XBMC has a very developed add-on system, and developing the application as an XBMC add-on could have provided us with more control over XBMC, and a very convenient distribution system. Despite the above we have decided to develop an open source application due to the following reasons:

- writing an XBMC add-on would have restricted us to writing in a specific language and using openNI due to Linux compatibility.
- writing an XBMC add-on would have made our application dependent on XBMC updates and might not function well on next versions of XBMC.
- A standalone application is easy to adjust to other media players in the future.

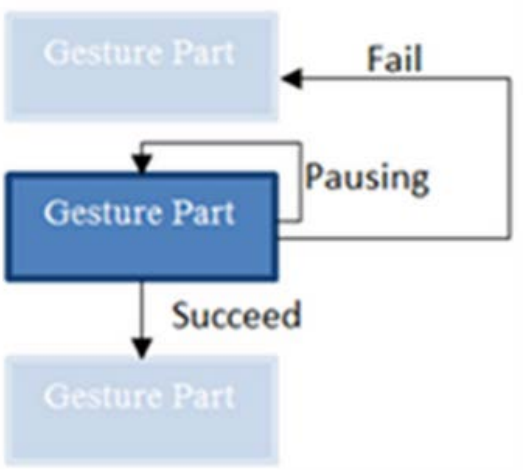
Application Structure



for every data stream we implemented a controller which gets raw data (interaction and skeleton) from the SDK's and translates it to events which are sent to a manager who handles these Events with respect to the application state. Remo will get data from the managers and translate it to user output using our UI.

GestureController

Gesture controller registers to Kinect.SkeletonStream and gets raw skeleton data from it This class actually holds a list of all gestures to be recognized by Remo and runs the following state machine for each one:



As you can see each gesture is divided to parts, every part consists of a set of relative conditions which will either move us to the next part or fail the gestures.

GestureManager

This class is in charge of deciding what to do when a gesture is recognized, and executing non UI related code(e.g sending keys to XBMC), also it has to send Remo relevant information so Remo would handle the UI and be aware of the application state.

InteractionController

Interaction Controller does pretty much the same as gestures controller only it registers to Kinect.InteractionStream and turns hand open/close/location data into hand moving related events (e.g. when we want to scroll fast it will calculate the hands speed when we open it. this will allow us later to know how much to scroll).

InteractionManager

Interaction Manager will get events from InteractionController and will act accordingly, it also keeps data on application state such as left/right hand open/close.

RemoScheduler

A good example that can explain RemoScheduler job is a mechanism implemented in order to prevent unwanted scrolling. when we move our hand left to scroll a small movement upwards or downwards can accidently trigger unwanted up/down scrolling. in order to prevent it RemoScheduler starts a timer every time a handmoved event is raised. and will only allow scrolling in other direction if 5 seconds pass. because scheduling may require coordination between all class it is implemented as a separate entity.

Implementation -

microsoft kinect sdk VS. openni

There are two main SDKs used today with prime-sense based depth sensors -
OpenNI - The OpenNI framework provides a set of open source APIs. These APIs are intended to become a standard for applications to access natural interaction devices.
The APIs provide support for:

- Voice and voice command recognition
- Hand gestures
- Body Motion Tracking

OpenNI is open source and linux compatible, it also supports most of the sensors on the market.

Microsoft Kinect SDK - The Kinect for Windows SDK enables to use C++, C#, or Visual Basic to create applications and experiences that support gesture and voice recognition by using the Kinect for Windows sensor and a computer or embedded device. The Developer Toolkit contains additional resources, sample applications with full source code Kinect Studio, and other resources to simplify and speed up application development.
Kinect SDK is only compatible for Windows Computers and Microsoft Kinect sensors.
We tried both SDKs and came to realize that for our purposes Microsoft Kinect SDK is more stable and easy to develop on. So we chose to implement Remo based on this SDK.

Communicating with XBMC

In order for Remo to be used for other applications its way of communicating with other applications (default XBMC) is emulating keyboard strokes. In order to implement it we used microsoft's System.Windows.Forms.SendKeys library.

On Screen Display

Because XBMC (and generally movie watching) is operating under full screen mode we needed a way to interact with the user while he is watching a movie, and a regular window can't be seen. Our way of solving this problem was using WPF's transparent window option which enabled us to show OSD like data to the user while he is watching a movie in fullscreen mode.