Playing a Virtual Musical Instrument

Matthew Stangel
Daniel Torgerson
Mike Rowe, Ph.D.
Computer Science and Software Engineering Department
University of Wisconsin-Platteville
215 Ullrich Hall
Platteville, Wisconsin 53818
stangelma@uwplatt.edu
torgersond@uwplatt.edu
rowemi@uwplatt.edu

Abstract

This paper demonstrates a relatively new user interface, hand position. The interface is characterized by using a common webcam to interpret hand position. This interface is demonstrated by a system which implements a virtual percussive keyboard musical instrument. To simplify the playing the virtual keyboard and also the discrimination of position from the background, the user holds a drum stick with a ball of red tape wrapped around the end similar to a child's xylophone mallet. A user can simulate playing this instrument by moving this drum stick within the view of the webcam relative to the virtual keyboard. The webcam processes a drum stick's position and maps it to a virtual keyboard. The virtual keyboard is interfaced to a MIDI device which produces any of the various sounds supported by MIDI, such as a xylophone, organ, guitar, trombone, flute, etc.

Introduction

This project demonstrates a relatively new type of program interface. This interface is characterized by using a common web cam to interpret hand position. Specifically this project implements the playing of a virtual percussive keyboard musical instrument similar to a child's xylophone. To simplify the playing of the virtual keyboard and also help with discrimination of position from the background, the user holds a drum stick with a ball of red tape wrapped around the end similar to a xylophone mallet. A user can simulate the playing of the instrument by moving the xylophone mallet within the view of the web cam. The web cam processes the xylophone mallet's position and maps it to a virtual keyboard. The virtual keyboard is interfaced to a MIDI device and produces any of the various sounds supported by MIDI, such as a xylophone, organ, guitar, trombone, flute, etc.

The project exhibits how easy functionality similar, although much simpler, to that of the Xbox Kinect® and Playstation Move® can be implemented from scratch using a common webcam, Visual Studio C#, a standard MIDI device, and common computer soundcard and speakers. The functional parts of this system, see figure 1, include the following modules:

- Image Capture – this module captures raw bitmap images from the webcam that can be further processed by the Image Converter module.
- Image Converter – this module changes the image format to one that is more easily processed by the Image Interpreter module.
- Image Interpreter – this module locates the red tape ball on the end of the drum stick in an image frame and determines its relative positions with respect to a virtual keyboard.
- Note Converter – this module translates the red tape ball position relative to the virtual keyboard into a musical note.
- MIDI Interface – this module sends the musical note to the MIDI device which drives the computer speakers.
- Image Displayer – this module displays an image that helps the user determine position relative to a virtual keyboard. This image is a composite of the webcam image of the xylophone mallet and the virtual keyboard.
- GUI Control – this module allows the user to specify the MIDI instrument characteristics and to adjust video capture parameters, such as color to make the xylophone mallet easier to discriminate from the background.

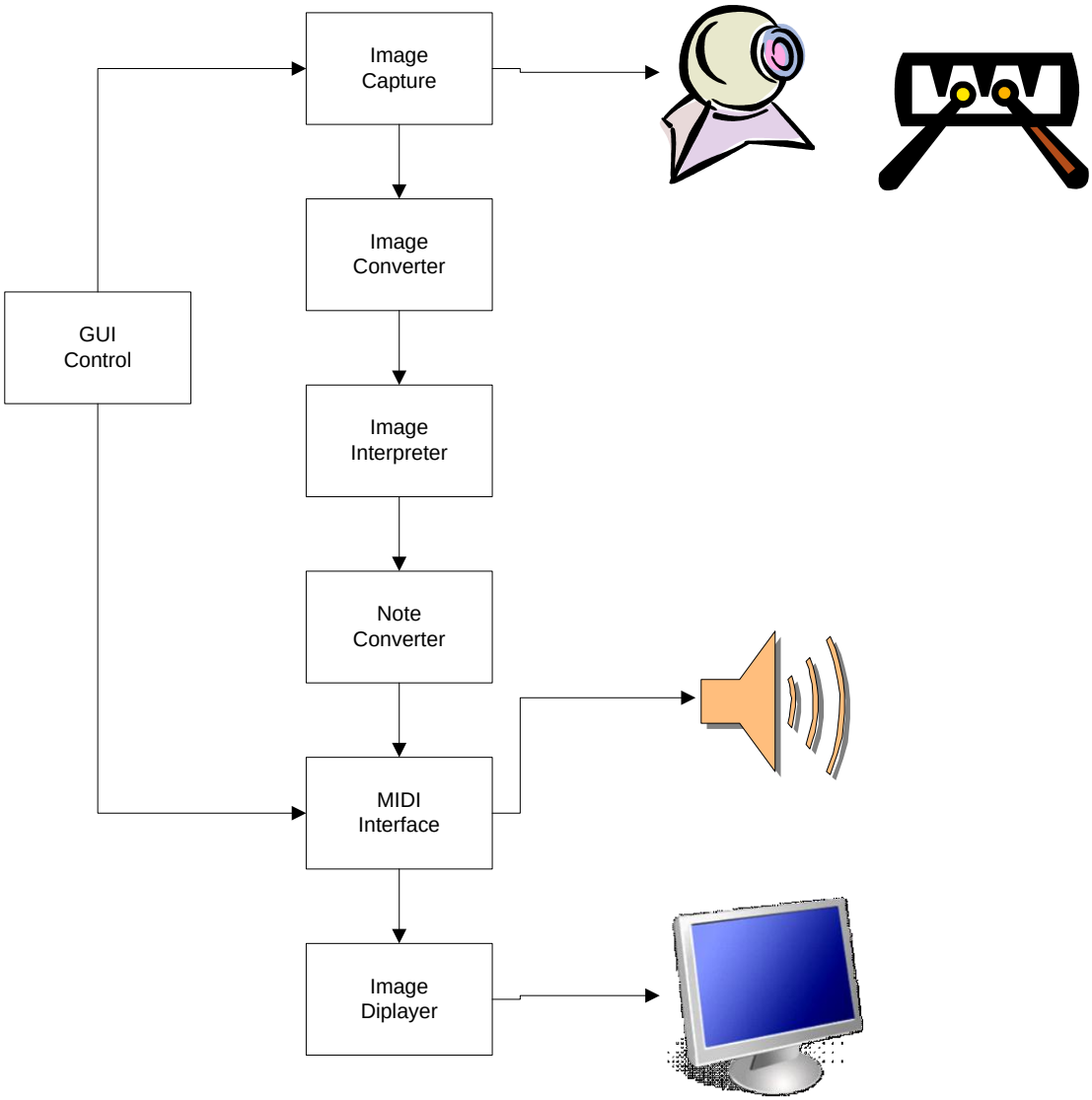

Figure 1: Block diagram of the system

The next section contains a discussion of the system architecture, describing the technology used to implement each of the functional parts of the system. The last section contains a discussion of future enhancements planned for this system.

Functional Description

The program runs in a single form without menu controls. See figure 2. Connection to a web camera is done automatically at startup. Output from the web camera is displayed on the largest part of the form, taking up about five-sixths of the total area of the form. The image captured from the web camera is scaled to fill the area. A virtual, color-coded keyboard child's xylophone is superimposed onto the image to provide user visual clues

as to where play. It is only within the region where these keys are shown that the program will interpret movements that will be converted to played notes.

User Interface Controls

The bottom sixth of the form contains several controls with which a user can modify the behavior of the program. These include:

- A trackbar near the bottom of the window labeled “Color Filter” allows the user to change the RGB(255, 0, 0) color threshold to improve detection of the red ball at the end of the drum stick from the background.
- A second trackbar labeled “Noise Filter” allows the user to adjust the threshold of minimum the grouping of pixels that the program uses to classify a moving object as the xylophone mallet.
- A drop-down menu allows the user to select the desired MIDI instrument. In Figure 2 the instrument selected in Marimba.
- A check box “Filtered Image”, when checked, stops displaying all pixels from the webcam other than those detected as “red”.
- A checkbox labeled “Emphasize” displays all detected pixels as the color magenta (RGB(255, 0, 255)) when checked.

Figure 2: GUI Window

Program Structure

Aside from the Sanford Multimedia package, the entirety of the program exists within the main form. Upon form load, the program initializes its components including the draw timer and the keyboard bitmap, populates the instrument list, initializes the MIDI device, and connects to the camera. Once the program completes its initialization, with every timer tick, the following process is performed to process input from the camera:

- The program copies the latest camera image to a bitmap and flips it horizontally to create a mirror image.
- The program converts the bitmap to an array of bytes. This conversion allows for the faster reading and writing to and from the individual pixels of the image.
- The program loops through the array 4 bytes at a time, starting with the first byte after the image format header (byte 54). From that byte until the last byte of the array, each group of 4 bytes describes one pixel of the image. The first of these groups of four bytes is the blue value of the pixel (from 0 to 255), the second is the green value, the third is the red value, and the fourth is the alpha value (which is ignored by this program).
- For each pixel, the program checks the color of that pixel against a group of constants to determine if the pixel is within the desired color range. If it is, the program calculates what key (if any) the pixel is above, and increases the detected pixel count for that key. Then, if the "Emphasize" option is selected on the main form, the program turns the pixel pink. If, however, the pixel is not in the specified color range, the pixel is replaced with the keyboard image's corresponding pixel (if one exists). If there is no keyboard pixel corresponding to this location, the pixel is colored black if the "Filtered Image" selection is selected on the main form. If neither of these is true, the pixel is ignored.
- The program checks how many pixels of the correct color were detected on each key. If that number is greater than the number required to play a note, the note corresponding with the key is added to a list.
- The program checks the notes currently playing against the list of notes just detected. If there are any notes playing that were not just detected, the program stops those notes from playing. Any notes currently playing that were detected again continue to play.
- The program checks the list of notes just detected against the list of notes currently playing. If any of the notes detected are not currently playing, the program begins playing those notes.
- Finally, the program draws the image from the camera onto the screen, modified by any alterations created in the pixel detection loop.

Planned Improvements

We focused our design and development efforts on making the program portable and usable with the most basic equipment without regard to performance. The next version will be performance tuned to provide more real-time "playing" performance. In addition a faster multi-core processor could also reduce lag and make the system more playable.

The initial project uses a rather old camera with low resolution and slow frame rate. The low camera resolution and slow frame rate sometimes results in the wrong note being detected and a slow frame rate (and the skill of the musician) limits the number of notes that can be

played in a time interval. The slow frame rate also contributes to a significant screen update delay. A better camera with a higher resolution picture would increase viewing space.