Project Goal
The aim of this project is to create a mobile VJ system, incorporating an external video control source, e.g. a remote accelerometer. The final result will be a mobile video player whose video effects are controlled live via a wired communication protocol.

Background
To start with a definition, a VJ is “a performance artist who creates moving visual art (often video) on large displays or screens, often at events such as concerts, nightclubs and music festivals, and usually in conjunction with other performance art” [1]. Thus, the VJ is a sort of video complement to a live music DJ. The VJ’s performance is often in sync with an audio track, with the artist applying live video effects in time using an instrument or controller.

Current VJ’s have many options for software support, most of which require a PC to run. ArKaos [2], a commercial product, offers live mixing and management of video clips, as do open source efforts like Pd [3] with GEM [4]. However, these applications are very versatile, and so require more processing power than is commonly offered in mobile devices. Should a VJ want to use a mobile device for simple video effects, these programs would not meet the artist’s needs.

Thus, there is an opportunity here for a mobile implementation of simple, straightforward video effects and control. Moreover, if this implementation can be abstracted to allow for different types of controllers, then a VJ could choose a controller that best fits his/her tastes.

Description
The PXA-VJ system can be broken down into the a block diagram, shown in Figure 1.
Controller Hardware

Rather than assemble a controller from scratch, I have chosen to use a Cypress CY3209 PSoC Development Kit for the controller hardware. The CY3209 has a 2D accelerometer connected to a Cypress CY8C27643 PSoC [5]. The PSoC has outlets to a radio socket, which I am repurposing as a simple RS-232 serial interface. The CY8C27643 supports up to two UARTs [10]; only one will be needed to transmit packetized control data based on accelerometer output.

Controller Firmware

The controller will have fairly simple firmware: It polls the accelerometer output, and does minimal processing of that data. It has its own UI, consisting of an LED array to indicate inclination angle, and an LCD screen to show the raw values read from the accelerometer. It also places the raw accelerometer data in packets to send to the host. The design aim is to have very little processing on the controller, and the majority of processing on the host.

Hardware Cable Design

The Intel PXA27x has DB-9 connectors, but the Cypress CY3209 does not. In order to connect the UARTs from the two systems, I needed to create a custom cable.

I repurposed the 3209’s radio socket for the serial cable, using a DMM to find which pins would work for Tx and GND. Then I cut the end of a DB-9, and used the DMM to identify Tx, Rx, and GND. Figure 2 shows how I connected the 3209 Tx to the PXA Rx line.

![Custom cable design](image.png)

Figure 2. Custom cable design.
**Host-Side Control Listener**

The host-side control listener application (vjcontrol) receives packetized data via /dev/ttyS1, processes this data, and produces media player control statements. These statements are written to a FIFO file in /tmp. See Appendix B for the vjcontrol source code.

**Host-Side Media Player**

The media player (vjplayer) is a wrapper around MPlayer. MPlayer has a feature called “slave mode,” which allows it to receive commands like “pause”, “quit”, and “brightness 100 1”. This command is most important: it allows a controlling application to set absolute values for brightness and contrast. Also noteworthy is that MPlayer can be configured to receive these text commands from a FIFO. The vjplayer then, starts a pre-made video and configures MPlayer for slave mode, receiving control statements from the aforementioned FIFO file. See Appendix A for the vjplayer source code.

**VJ Source Media**

In order to make the best use of the PXA27x’s 240x320 display, I created a demo video that fills the screen and contains slowly moving images. This is best suited to show the rapid effects of the VJ controls. I used Boinx FotoMagico to create the custom-size video, and then used QuickTime Pro with the Divx video codec to convert the video for PXA playback.

**Design Challenges and Adjustments**

**PX A UARTs**

The PXA system has two DB-9 ports. One resides on the daughter card, and all data received there is received by a root shell. Rather than risk creating a bug like Google Android’s recent much-publicized root-access flaw, I chose to examine the system schematics and found a second UART which is commonly used for Bluetooth support. With proper switch settings and knowledge of /dev entries, I was able to successfully read from this serial port.

Then I discovered that this second port also has a root shell attached to it. I needed to find a way to keep my semi-random communication traffic from executing in a shell, and found a solution while I set to work on communication protocols.

**Protocols**

I wanted to have a simple communication protocol for the accelerometer data: I knew I could calculate checksums or CRC’s to ensure data integrity, and use character escaping mechanisms to isolate my Start-of-Packet bytes, but all of this would place unnecessary computational burdens on the controller.

The first iteration of my simple protocol started each packet with 0xFF, followed by the big-endian representations of the two 16-bit signed integer readings from the
accelerometer. This was problematic, as the value 0xFF could occur anywhere in the packet.

The second iteration started each packet with 0xF0FF. Then, to prevent this two-byte sequence from occurring anywhere else, the two signed ints from the accelerometer were incremented by 1000. These values originally ranged from -1000 to 1000, so incrementing them ensured that they would stay positive, thus ensuring that no two bytes in a row would have their most significant bits set, making the start-of-packet sequence unique.

However, in considering the Android bug and my serial port predicament, I realized that I could take a different approach: use ASCII text commands, and let the root shell interpret them for me. I designed the third iteration of the communication protocol, with the following command formats:

- \texttt{vjcontrol accelx <value>}
- \texttt{vjcontrol accely <value>}

This is the extensible command format that the final project used.

\textbf{Cypress PSoC Designer 5.0}

In my mid-course report, I noted that the Cypress PSoC Express software does not expose all of the PSoC’s capabilities: it is notably missing UART support. I planned to migrate my Express project to PSoC Designer 5.0, and easily add a UART to my existing design (a functioning accelerometer and on-device UI), as Cypress advertises. [12]

Unfortunately, PSoC Designer 5.0 is very new software (only several weeks old when I set to work), and it has significant, show-stopping bugs. I found the first when including accelerometer support in a hybrid project: the project will build successfully one time, and then subsequent builds (with no changes) will fail.

I hoped to work around this issue by using many different copies of my project—a time-consuming workaround, but doable. Then I found the second and fatal bug when adding UART support. Using Cypress’ example code and instructions, I found that the UART component produced many errors and unreliable output.

These unfortunate discoveries meant that I had chosen the wrong hardware platform for my controller, and would not have time to assemble a new accelerometer-based device.

\textbf{Keyboard Controller}

Fortunately, my design goal of interoperability meant that I could still prototype a different controller using a PC. I wrote an application to run on a Linux machine, which would use the keyboard as input and send the same vjcontrol commands out over the main serial port. See Appendix C for the source to vjremote.
Conclusion

It would have been ideal to successfully assemble the entire chain shown in Figure 1. It is unfortunate that the failure of one link in the chain (the outgoing UART from the PSoC) prevented the original accelerometer-based design from working.

Nonetheless, this has been an extraordinary cross-disciplinary learning experience. From user interface design all the way down to bit-level protocol design, the project required great knowledge and understanding of both Computer Science and Electrical Engineering. I also learned first hand the danger of finding that your early hardware design decisions were misinformed.

The final result of an extensible VJ platform with a working (if overpowered) controller prototype is still very useful, and can provide a helpful starting point for future work in this area.
References


Appendix A — vjplayer

#!/bin/sh
#
#  vjplayer.sh
#  CSE 237A Final Project
#
#  Copyright 2008 John Kooker. All rights reserved.
#
FILENAME="$1"

# check if /tmp/vjfifo exists, create if necessary
if [ ! -e /tmp/vjfifo ] ; then
    mkfifo /tmp/vjfifo
fi

# play a file while taking input from the fifo for control
mplayer -slave -input file=/tmp/vjfifo $FILENAME

# removed params "-zoom -x 240 -y 320 -loop 0"
Appendix B — vjcontrol

#!/bin/sh
#
#  vjcontrol.sh
#  CSE237A Final Project
#
#  Copyright 2008 John Kooker. All rights reserved.
#
PARAM="$1"
VALUE="$2"
PLAYERCONTROL=""

# check if /tmp/vjfifo exists, create if necessary
if [ ! -e /tmp/vjfifo ]; then
    mkfifo /tmp/vjfifo
fi

# bail if there's no player
# process param, scale value, and send command to fifo
case $PARAM in
    "accelx")
        PLAYERCONTROL="brightness"
        ;;
    "accely")
        PLAYERCONTROL="contrast"
        ;;
    *)
        # bail here
        echo "$PARAM - not a match"
        exit
        ;;
esac

if [ $VALUE = "" ]; then
    echo "missing value"
    exit
fi

# scale value from [-1000, 1000] to [-100,100]
VALUE=$(expr $VALUE / '10')

# mplayer control looks like "brightness <value> <abs>"
echo $PLAYERCONTROL $VALUE 1 > /tmp/vjfifo
echo $PLAYERCONTROL $VALUE 1
Appendix C — vjremote

#!/bin/sh
#
# vjremote.sh
# CSE 237A Final Project
#
# Created by John Kooker on 2008-12-03.
# Copyright 2008 John Kooker. All rights reserved.
#

# initialize ttyS0 to 9600
stty -F /dev/ttyS0 9600

CMD=""

while [ true ]; do
  # on keypress, send randomized accelerometer data
  read -n1 KEYPRESS
  echo

  case $KEYPRESS in
    1 )
      CMD="vjcontrol accelx -1000"
      ;;
    2 )
      CMD="vjcontrol accelx -750"
      ;;
    3 )
      CMD="vjcontrol accelx -500"
      ;;
    4 )
      CMD="vjcontrol accelx -250"
      ;;
    5 )
      CMD="vjcontrol accelx 0"
      ;;
    6 )
      CMD="vjcontrol accelx 200"
      ;;
    7 )
      CMD="vjcontrol accelx 400"
      ;;
    8 )
      CMD="vjcontrol accelx 600"
      ;;
  esac
end

9 )
   CMD="vjcontrol accelx 800"
   ;;
0 )
   CMD="vjcontrol accelx 1000"
   ;;
* )
   # random contrast
   # get one random number
   RAND1=$(dd if=/dev/urandom count=1 2> /dev/null | cksum | cut -f1 -d " ")
   RAND1=$(expr $RAND1 % 2001 - 1000)
   CMD="vjcontrol accely $RAND1"
   ;;
esac

   echo $CMD
   echo $CMD > /dev/ttyS0

done