Physical Computing : Final Project

HyperRecorder

Initial Analysis

My original proposal was for some kind of musical instrument / performance controller with a recorder-like casing, and using breath and recorder fingering. The aim is a working controller and a performance using it at the presentation.

There are several obvious requirements

a) to capture the player's finger positions.
b) to capture the player's breath.
c) to mount the device in a playable form.
d) to convert the signal from the controller to music

which, as we start to think about them, unfold into more complex challenges. I will summarize the initial thinking on each of these before describing the detailed work on each part and how my solution evolved.

a) How to capture players' finger positions.

A recorder-like device needs small pads or holes, close together so that different combinations of fingers can define the note. Were it possible to produce actual hinged finger pads (such as a concert flute or saxophone) perhaps these could be turned into momentary buttons. However, I am unable to construct such an intricate machinery and have no existing instrument that I am prepared to hack in that way. A recorder is far simpler, but requires the player to cover holes.

My initial thought, therefore, is to use light-dependent resistors inside each hole, which will measure the degree of light entering. From this we will infer the degree of covering.

This decision expands into a number of issues :

A recorder has 8 holes (7 at the front and one thumb hole at the back). Plus I want to record the breath of the player. That adds up to 9 signals being sent from the controller to our Arduino. The Arduino, however, has only 5 analogue input pins. How, then, can we get all the inputs from the controller into the Arduino?

One possibility is to use the digital inputs to the Arduino which are more plentiful. But that raises another problem. You can build a voltage divider between a fixed resistor and the LDR and, given the right light levels and the right fixed resistance, it will act as a switch, sending a 0 or a 1 into the digital input pin. But it is inflexible and extremely sensitive to the ambient light. We could find ourselves in too dark an environment unable to distinguish between the hole closed or open.

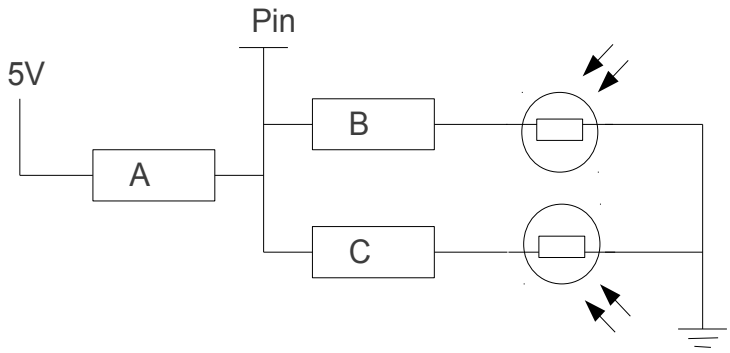
With an analogue input, as long as we can get some voltage difference between covered and uncovered holes, we can calibrate the software in the Arduino to distinguish the ranges for that particular environment.

So, we face the first difficult challenge : how to get around the limits of too few analogue inputs or to make better use of the digital inputs. Two solutions are worth exploring further :

Solution 1 :

The first solution which occurred to me is to try to feed data from two LDRs to each analogue input. I propose a cell designed as in Circuit #1. A voltage divider between a fixed resistor A and a sub-circuit consisting of two parallel tracks, each containing an LDR and a fixed resistor. (We'll call these resistors B and C). We take the analogue signal from between the fixed resistance A and the sub-circuit.

Circuit #1

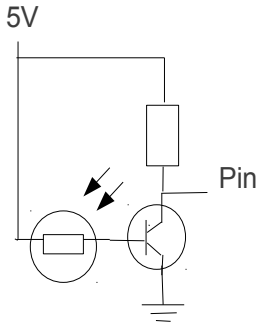


Because resistances B and C are different, the overall resistance of the sub-circuit will also change depending on which LDRs are occluded. With sensible choices of A, B and C and software calibration, the four possibilities (neither, B, C, both) can be distinguished. With this method, 4 analogue input pins can accommodate the 8 holes and still have one left for the breath input.

Solution 2 :

This was suggested by an electronic engineer that I put the problem to. To use a transistor as an amplifier to guarantee a reliable digital output from an LDR. This would lead to something like Circuit #2 : Here, at least a little light on the LDR guarantees that the transistor is switched on, which produces a clear output signal to send to a digital input. The question here, though was whether we could be confident of sufficient light to switch the transistor.

Circuit #2



In the event, I mocked up a version of solution 1 on breadboard and found that it did seem to work (the four combinations could be distinguished on a single analogue input) and so I built a board around this idea. I was concerned that Solution 2 would still largely depend on an absolute value of the light

b) How to capture the player's breath

Two possibilities suggested themselves initially. A microphone or some kind of pressure sensor. I had few intuitions about this before starting so acquired three possible candidates : a Piezo contact mic, a microphone insert and a pressure sensor.

c) How to mount the device

Ideally I wanted to mount within a real flute / recorder-like body. I had several of these and eventually chose to start with a plastic treble recorder (treble because it is larger, and so easier to work with than a descant.)

I also experimented with some bamboo cane. Bamboo would be a more beautiful material to work with than the plastic of the recorder. Unfortunately, without the proper equipment I found it hard to drill into. Trying to start my making a hole with hammer and screwdriver (used as an awl) simply split the bamboo.

d) Software

The big decision with regards to software was how to divide the work between the Arduino and the main computer. The more processing took place on the Arduino, the more flexible the instrument could be. For example, if all the processing necessary to decode LDR readings to actual note values took place on the Arduino, it would be fairly simple to wire up an old-school Midi cable to go straight into a synthesiser. Weighed against this was the fact that I needed a fairly fiddly calibration phase. If this occurred on the computer, I could drive calibration from the computer's keyboard and give visual cues on the screen. If I wanted to do it all on the Arduino, I'd need to add extra input elements (eg a button to switch between calibration mode and performance mode) and extra outputs (ways to tell the user where we were in the calibration process and what to do next.)

Prototyping and Evaluation

Fingerings

Experimentally on breadboard I found that the following resistances seemed to work.

A = 10 K Ohms, B = 560 Ohms, C = 220 Ohms

In one example measurement I recorded the LDR values as being 72KOhms when open, 256KOhms when closed.

I wrote a quick program to check the expected resistance values that would be produced by this. (Listing #1)

Listing #1 : Python script to calculate resistances.

```
# three resistors a, b, c

# resistence of LDR component (b + lb) | (c + lc)
# 1/Rt = 1/(b + lb) + 1/ (c + lc) ...
#        Rt = 1 / (1/(b + lb) + 1/ (c + lc))

def resist(a,b,c,lb,lc) :
      bb = 1.0/(b+lb)
      cc = 1.0/(c+lc)

      inner = 1.0/(bb+cc)
      return a+inner


l_open = 3.6*20000
l_closed = 12.8*20000

a = 10000.0
b = 560.0
c = 220.0

def pp(a,b,c,l1,l2) :
      r = resist(a,b,c,l1,l2)
      print "Resist: %s, Ratio: %s, Volt: %s" % (r, a/r, 5*(a/r))

pp(a,b,c,l_open,l_open)
pp(a,b,c,l_closed,l_open)
pp(a,b,c,l_open,l_closed)
pp(a,b,c,l_closed,l_closed)
```

If you run these values the results are as follows :

```
Open, Open    : Resist: 46194.80, Ratio: 0.2164, Volt: 1.0823
Closed, Open  : Resist: 66356.11, Ratio: 0.1507, Volt: 0.7535
Open, Closed  : Resist: 66546.39, Ratio: 0.1502, Volt: 0.7513
Closed, Closed: Resist: 13819.94, Ratio: 0.072,  Volt: 0.3618
```

At this point, we might suspect that the recorder will have difficulty distinguishing the closed-open from the open-closed pattern. And I need to confess that I only wrote this code and did this more detailed analysis *after* I'd decided on my resistor values and soldered them into place.

My actual analysis consisted of spending some time doing informal testing on the breadboard, looking at the numbers being sent up the Serial cable from the Arduino and trying different combinations of the resistors I had available. The resistors chosen for A, B and C did look like the best bet for distinguishing even open-closed from closed-open. But in retrospect it would have been a lot smarter to have written this program before to discover optimal ratios between A,B and C. And then chosen the resistors accordingly.

Although these numbers don't look at all good, it should still be stressed that the fingering part of the recorder, does work ... often. And does distinguish between different notes more than it fails.

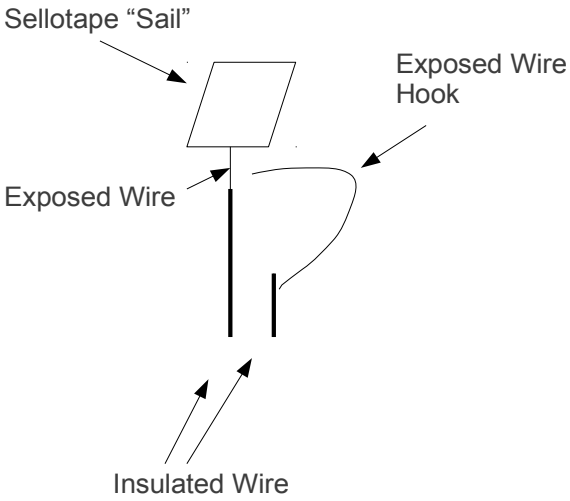
Breath Control

In early investigations I found that neither my pressure sensor nor the microphones (piezo ro normal insert) responded noticeably to being blown on.

I decided to change tack.

I wondered if I could make a simple momentary switch that could be “blown closed”. Some experiments with a couple of pieces of wire, a flat piece of Sellotape acting as a "sail" proved that the concept “could” work. But extremely unreliably. (Circuit #3)

Circuit #3 : Breath controlled momentary switch



The wire carrying the sale was a single strand of wire I cannibalised from an old pair of (cheap) personal-stereo earphones. I was surprised to find that the phones used so little wire. And also that it was so bouncy.

Although I was quite intrigued by the possibility of this kind of input device (the electronics are very simple as it is just a momentary switch + pull-down resistor), my implementation was too fragile to be usable.

Mounting the Device

After the failure with the bamboo, I returned to using the treble recorder body. I had two requirements while working on this prototype : a) not to destroy the recorder (I wanted the option of reverting to using it as a real recorder); and b) that it would be possible to remove any of the LDRs for maintenance. (If, for example, one of the soldered connections broke.)

I took the wires from inside an old CAT5 (ethernet) cable. This handily gave me a number of twisted pairs and I soldered one LDR to each pair. One LDR + wire was then threaded through each finger hole in the recorder body.

The LDR was just small enough to slip through so I needed to find a (reversible) way to hold it more or less at the position of the hole. I eventually settled on the idea of using a short piece of plastic “tag” threaded between the two legs of the LDR, and then taped across the hole. The plastic tag therefore separated the two legs of the LDR, (ensuring they didn't short) and prevented the LDR sliding inside the body of the recorder.

In fact, it stood maybe five or six mm off the surface of the recorder. It is a little bit uncomfortable compared to a covering holes, but the LDRs are close enough to the right place that a recorder or flute player can easily adapt to them.

Software

I opted for doing the majority of the work on the computer, using Processing and putting only a minimal piece of code on the Arduino. I think, this was the better option, despite losing flexibility. It meant that I could write (and continue to modify) the calibration and decoding routines for the computer, using the screen to feed instructions to the player during calibration - and to give an indication of what values were actually being read from the Arduino. This was useful because, as a player, I started adapting myself to problems of disambiguating fingerings by changing the angle of the recorder based on the screen showing me the levels it was actually reading.

The Arduino code is minimal. See listing #2. The Processing code, on the other hand, is correspondingly longer. Note that this prototype used 6 LDRs consolidated via a board on which I'd soldered three copies of Circuit #1, into three analogue signals going into analogue pins 0, 1 and 2.

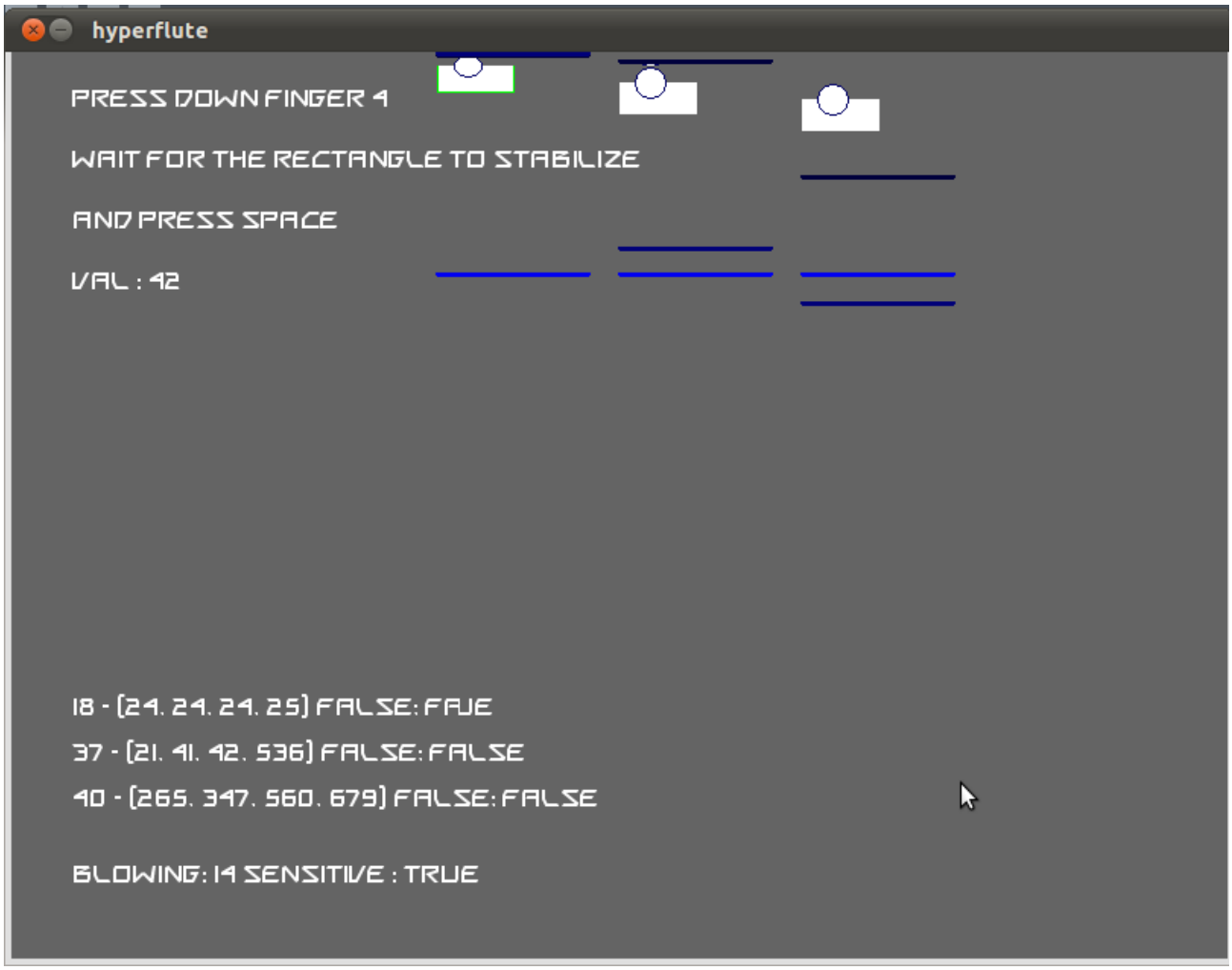
Analogue pin 5 was reserved for the breath controller.

Listing #2

```
void setup() {
   Serial.begin(9600);
}

void loop() {

  Serial.print("[");
  for (int i=0;i<3;i++) {
    Serial.print( analogRead(i));
    Serial.print(",");
  }

  Serial.print(analogRead(5));
  Serial.println("]");
}
```

On the computer, the software runs in one of two modes : calibration or play.

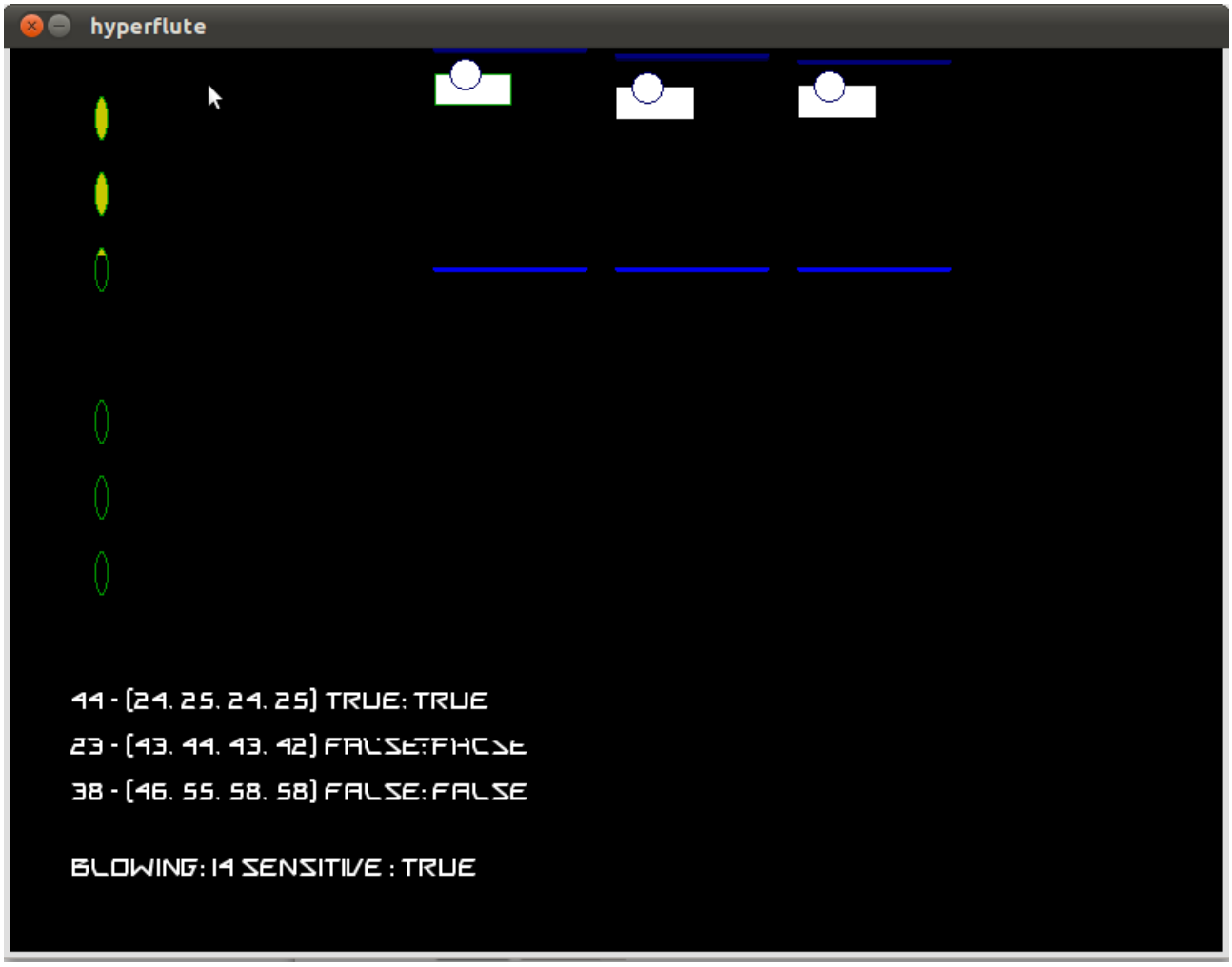
When in calibration mode the computer automatically steps through a sequence of notes. Displaying instructions to the player to hold down particular fingers or combinations.



We also see the actual numbers that are coming up from the Arduino while the white blocks move up and down to represent the same information graphically.

In the Play mode, the player is shown a stylized image of the recorder holes. Filled yellow where the program interprets them as being covered, and unfilled where it interprets them as uncovered.

Again the player is also shown the rectangles representing the actual numbers that are being received. And this gives her a feel for how the recorder is behaving. It may be a quirk of how I hold it during calibration, or of the hardware, but I often find the second finger is most accurately recognised when the recorder is held almost horizontally. But the sixth usually requires the player to lean forward so that the recorder is almost vertical before it is recognised.



When the software has identified a note from the recorder it plays it on either a square or saw-tooth wave using the Minim library.

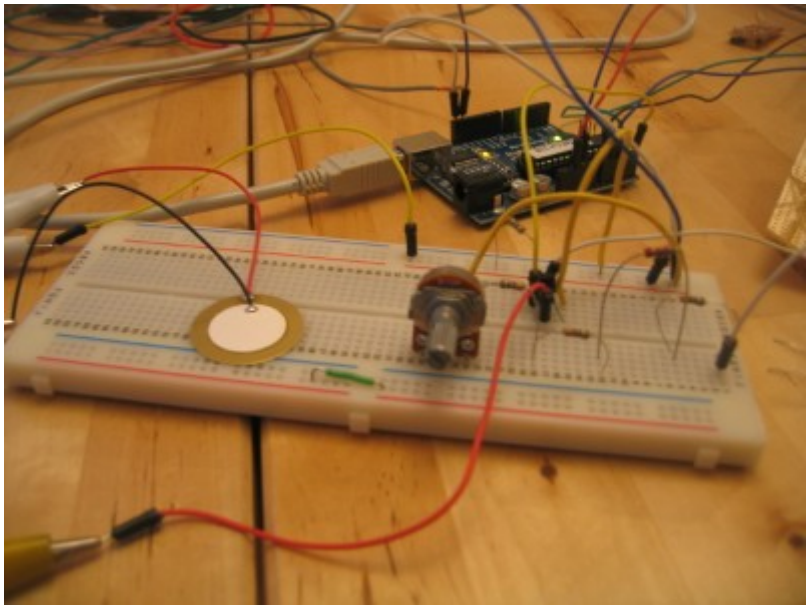
A video of me playing with the recorder can be seen, and the full Processing code can be downloaded, from this web-page.

http://www.doc.gold.ac.uk/~ma001pj/phys_comp/flute/index.html

Further Work

The hyper-recorder is a prototype and not suitable for any real performance. But it is “playable” by a musician and I feel that it gives a sufficiently plausible experience to continue working with. (See the aforementioned video.)

After showing the hyper-recorder briefly in the class last week, I was advised to try again with the Piezo microphone, using a transistor based amplifier to try to make a signal visible to the Arduino. I have experimented further on breadboard, but have so far failed to get this working.



It was also suggested that I go back to trying to find a way of using the digital inputs (which would eliminate the ambiguity of trying to squeeze two LDRs through the same analog input). I intend to do some further experiments in this direction, using a single potentiometer to try to calibrate all inputs to adjust for ambient light levels.