MIRROR

DISCOVERING THE POSSIBILITIES OF UBIQUTIOUS COMPUTING FOR ONE-TO-ONE COMMUNICATION OF MOVEMENTS THROUGH TACTILE FEEDBACK.

A PROJECT BY
SÖREN TITZE & JUDITH GLASER

Abstract

Being able to transfer motion from person to person is a widespread problem. Whether you are a professional dancing teacher, a game designer or a physical therapist. Being able to convey the motion of another person is often at the very core of your problem statement. With Mirror we investigated the possibilities of combine motion tracking with tactile feedback in order to enhance the experience of shared motion. This document aims to provide a general overview of the nature and goals of project Mirror but focusses strongly on technical aspects.

Contents

1 Introduction **1**

2 How does it work? **1**
2.1 Variations on a theme 1
2.2 Technical Challenges 1
2.3 Progress 2
2.4 Final Implementation 3
2.4.1 Code 4
2.4.2 Flaws 6

3 Interesting Links **6**

1 Introduction

During the summer course “IXD S16 - On the Internet of things” the students of the computer science department of the “Freie Universität Berlin” and the product design students of the ”Kunsthochschule Weißensee” collaborated to create projects which envisioned the new possibilities the new generation of small ubiquitous computing devices provided. One of those projects is “Mirror” a device - or rather a set of devices - which aims to add a new layer of experience when two or more people move together. The name already indicates the basic idea of “Mirror”. One person moves and another receives tactile feedback in order to mirror the first person.

2 How does it work?

The basic principle is quite simple. The device contains multiple sensors to detect its current acceleration and rotation. This data is then transmitted to the other participant(s). Depending on which data shall be used (see section 2.1) it might be neccessary to do some processing beforehand. The other participant’s devices then instrumentalizes vibration motors to translate the received data into patterns that represent the transferred motion. After rather short discussion we agreed on concentrating our efforts to transfer the motion of the arms (or rather one arm). This helped us narrow our focus and we think that almost every movement of the bodies is reflected in the motions of the arm.

2.1 Variations on a theme

Since we used a chip that contained a three-axis accelerometer in order to detect motion we identified several modes of operation. It would be possible to either directly transmit the acceleration data and mainly convey a sense of tact. Another option would be to integrate the acceleration data and calculate the current velocity of the device to transmit a sense of direction and force/speed. The last possibility would be to integrate the velocity data again to compute and then transmit the actual position of the device which would allow to convey a precise feeling of the motion. Transferring only the acceleration data could for example mean to only pulse one vibration motor to indicate motion in a single direction. Modulating the pulse’s “intensity” or its overall duration could both be used to convey either duration or force of the original motion. More insights into our thoughts regarding motor placement and pulse modulation can be gained in chapter 2.3. While we explored those possibilites we discovered that trying to increase the precision of the transferred motion (e.g. transmit the velocity instead of the acceleration) always meant to overcome more challenges. The next chapter (2.2) deals with those challenges in more detail.

2.2 Technical Challenges

The biggest technical challenge was the evaluation of the acceleration data. Due to the very nature of how the acceleration is measured there will always be some error in the measurement. Most accelerometer chips don’t cancel out the gravitational pull which makes it even more difficult to obtain useful data. To illustrate this point please have a look at figure 2.2 and 2.2. On the first glance they are quite similar.

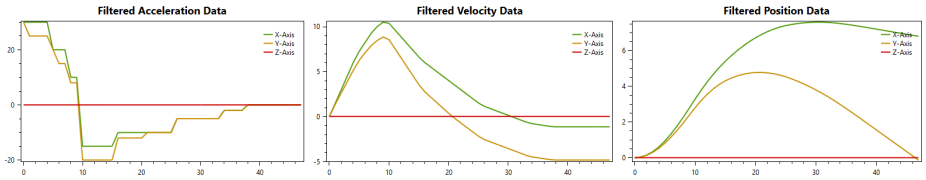

Figure 2.1: Graph created from fictional acceleration data.

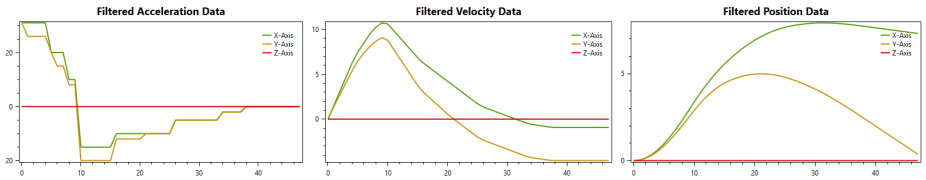

Figure 2.2: Slight variation in the data compared to the previous example.

But upon comparing the graphs it becomes quite clear that in the second one the objects would still be “in the air”. This is due to the fact that the (completely fabricated) data is slightly different in the first five data points. This would mean that especially for the third variant that we discussed in the previous section a small error would lead to huge variations in the transmitted data.
Another big challenge is posed by the power consumption of the complete system. This means that it is always neccessary to also incorporate a rather big battery, which leads

to compromises regarding the form factor. Considering the fact that we aimed at a light-weight and easily wearable system, this presented us with quite some difficulties.
The last noteworthy challenge was what we would like to describe as “shakiness” of the system. Especially the data transfer would often not work properly and lead to an overall break down of the system.

2.3 Progress

With the project progressing we mainly experimented with two aspects:

1. How to improve the recording and processing of the accelerometer data

2. How to setup the vibration motors in order to convey the motion

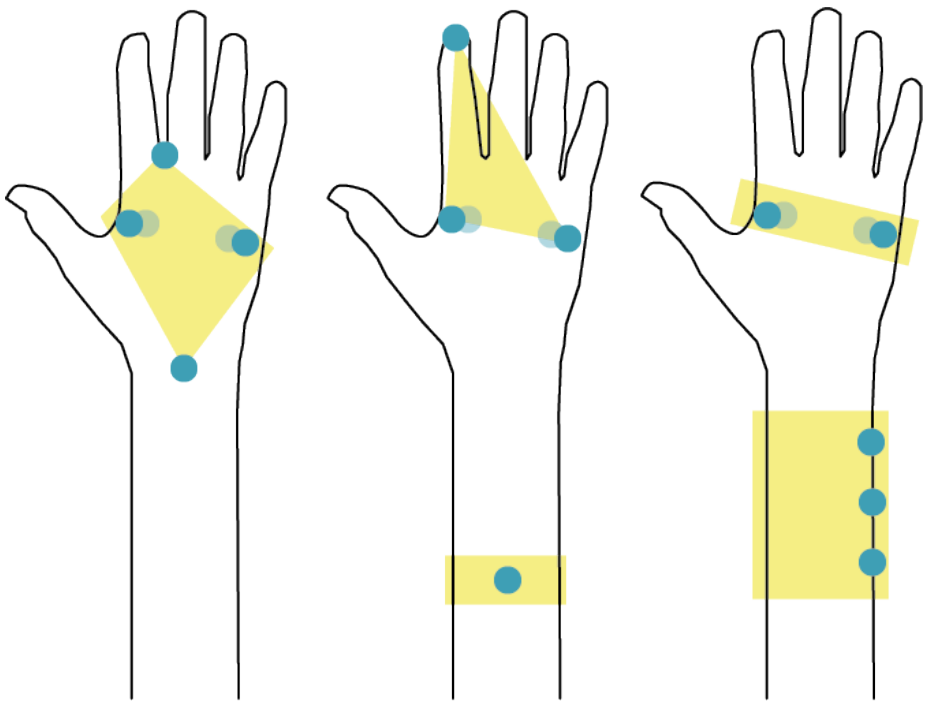

Figure 2.3: Different variations on how to position the vibration motors

The first part proved to be very difficult and in the final prototype even became obsolete when we switched the chips we used to the Bosch [BNO055]. This chip automatically accounts for the gravitational pull which helped us overcome the biggest obstacle mentioned in the previous section (2.2). We also looked into different filtering techniques but ended up implementing none of it due to time constraints.
The second main aspect we progressed in was the positioning of the vibration motors and the signal modulation in order to improve the accessability of the transfered motion signal. In figure 2.3 we want to present three main ideas of motor positioning. One of the points we tried to focus on was minimize the amount of motors while still retaining a good feeling for the transmitted motion. That’s why the setups in the figure only contain six to seven motors. The first setup is the one we actually ended up using. It is based on the idea that every motion of the arm is reflected in the hand.

Another aspect of this setup is the placement of the motors on nerve clusters. This helps the user to discern the pulse in each motor which is paramount to feel direction. The basic idea is to translate forward/backward motion by pulsing either the motor between the fingers (forward) or the one at the wrist (backward). To move left or right the respective two motors are activated. The same goes for up or down motion. Interesting are rotating movements which are translated by pulsing the four motors on the rotation axis sequentially. The second setup is quite similar to the first one but increases the distance between the motors in order to help the user distinguish between each signal location. This proved its biggest flaw and the reason why we abondoned for the first one. The mounting of the frontal motor would be to fragile to actually be practical. The motor back on the arm would surprise the user and rather than help him understand what motion is transferred. The third setup is too complex. Its idea is to pulse the three motors on the arm to indicate a

forward or backward motion. But this means that the motion in the other axis would be fundamentally different. Also rotation would be very hard to convey to the user.

2.4 Final Implementation

As can be seen in figure 2.4 the final setup is quite complex. There are some obvious flaws in the design that will be discussed in section 2.4.2. We aimed for a device that would be able to wirelessly transmit and receive its data.

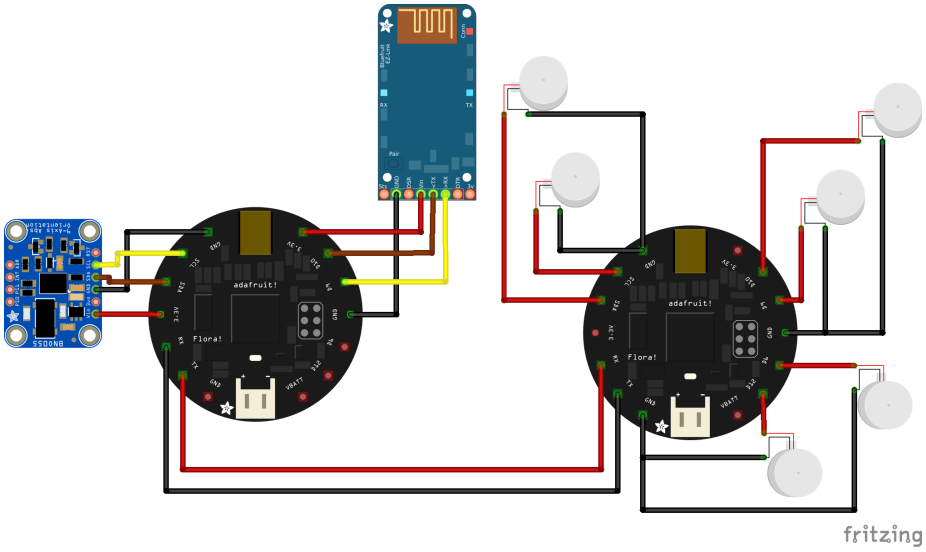

Figure 2.4: Schematics of the final prototype.

We split the final prototype into two parts. The first one measures acceleration and rotation data and transmits it via bluetooth while the second one controls the motors. The power supply is not shown in the figure. We actually powered it by using two [LiPo] batteries. For presentation purposes we later on connected the whole setup via USB to the PC. This helped with communication issues as well as the power supply.

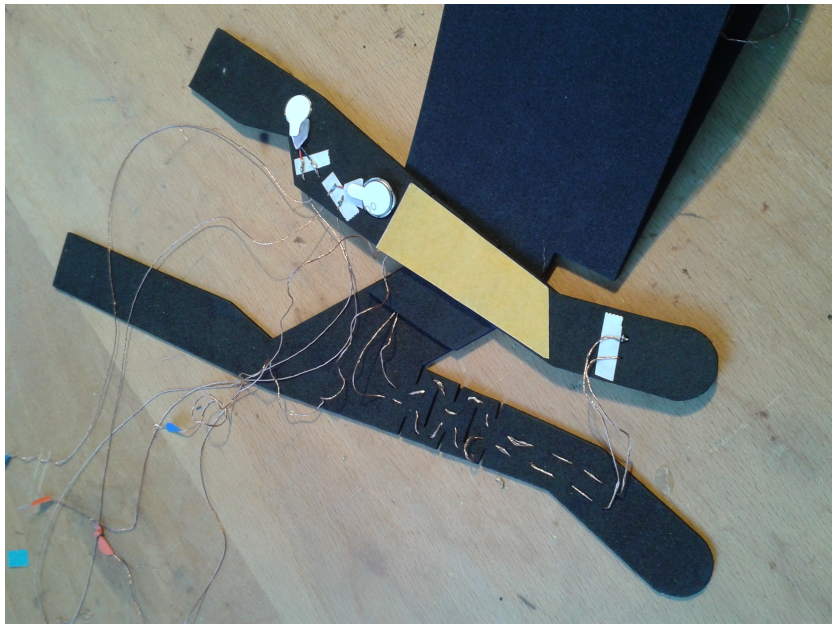

Figure 2.5: Look at the vibration motors and the connecting wires.

In order to securely mount the device on the user's arm we found inspiration with dancing and boxing equipment. The basic idea that can also be seen in figure 2.4 is that of "boxing tape" i.e. tape that is used to protect the joints of boxers or dancers during exercise. We designed and cut a wrapper from foam rubber into which we integrated all the tech. To interconnect the electronics we used copper lined thread. This provided us with a rather solid connection of the several while integrating the whole setup into a single device.

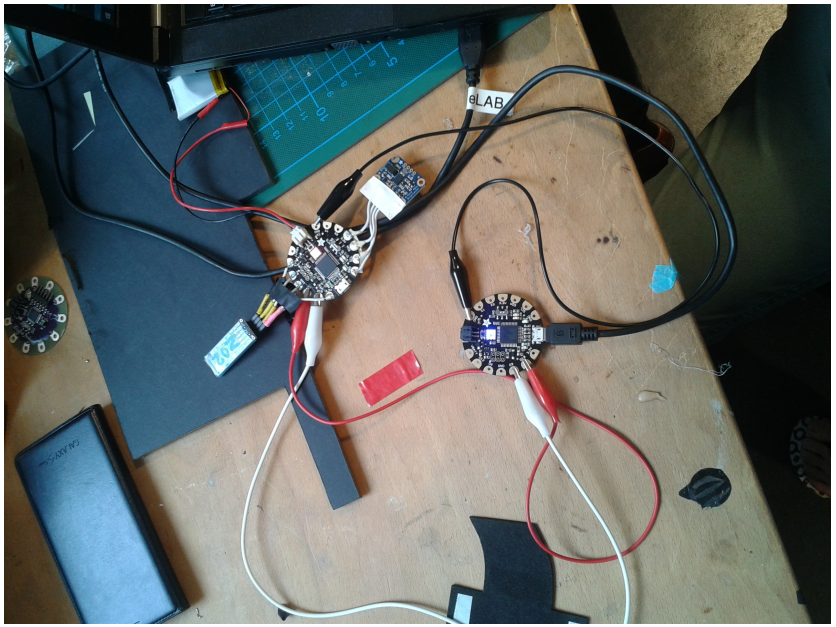

Figure 2.6: Interconnected Floras.

2.4.1 Code

Throughout the creation of "Mirror" we used several programming languages like [Processing], C++, C# and Python. The complete code can be found on our Github ([GitHub]) page. For the sake of brevity we will concentrate on the most interesting parts used in our final prototype.
The first snippet (2.4.1) shows how to connect to a running Arduino as well as sending and receiving data.

```
// Opens a serial port on the given port
// using the given baudrate. Sets the byte
// size to eight. Arduino uses no parity bit so
// set it to none, it does use a stop bit though.
// Set the read time out to one second.
ser = serial.Serial(
        self.port,
        self.baudrate,
        bytesize=serial.EIGHTBITS,
        parity=serial.PARITY_NONE,
        stopbits=serial.STOPBITS_ONE,
        timeout=1)

// To send a string simply write to the serial
// Don't forget to encode the command or else it will
// be interpreted as integer
ser.write(str(command_string).encode())

// Call readline to read until a \n. Make sure
// to use writeline on the Arduino site
line = ser.readline()
```

Interestingly it doesn't matter whether or not we used bluetooth. The code stayed the same. It just takes an extra step in Windows to set up the bluetooth chip as a serial channel. In iOS no extra step was needed though. When the Arduino is connected via USB or via the default digital IOs it is possible to use the predefined **Serial** constant. It is possible though to create a software serial port. How this is done is shown in the next listing.

```
// Define IOs for serial connection.
const int RX_PIN_SER = 9;
const int TX_PIN_SER = 10;

// Define input size
const int INPUT_SIZE = 8;
// Define input buffer
char input[INPUT_SIZE + 1];

// Create serial connection object.
SoftwareSerial conn(RX_PIN_SER, TX_PIN_SER);

// This function is called once when the
// Arduino is started.
void setup(void)
```

```
{
  // Initialize software serial
  conn.begin(9600);
  delay(50);

  // Initialize default serial
  Serial.begin(9600);
}

// This function is called in a
// loop as soon as the previous
// run through has finished.
void loop(void){
  // Send a string on the custom
  // serial connection.
  conn.print("String");

  // Send a string on the default
  // serial connection followd by \n.
  Serial.println("Line")

  // Read the amount of bytes into the buffer.
  byte size = Serial.readBytes(input, INPUT_SIZE);
  // Write the null terminator to finish the string.
  input[size] = 0;
}
```

In order to realize the more complex vibration patterns we defined some helper functions. To find them please have a look at the [GitHub] repository. In the next listing we just present the basics of manipulating digital IOs.

```
// Define digital IOs.
const int INPUT_IO = 11;
const int OUTPUT_IO = 12;

int input = 0;

// This function is called once when the
// Arduino is started.
void setup(void)
{
  // Initialize the digital IOs
  pinMode(INPUT_IO, INPUT);
  pinMode(OUTPUT_IO, OUTPUT);

  // Pull output pin to low
  digitalWrite(OUTPUT_IO, LOW);
}

// This function is called in a
// loop as soon as the previous
// run through has finished.
void loop(void){
  // Set the output to HIGH which
  // translates to 3V.
  digitalWrite(OUTPUT_IO, HIGH);

  // Read the digital Pin. Iff it is higher
  // than 3V (for 5V systems) or 2V (for 3V systems)
  // this function returns HIGH which is equivalent
  // to a boolean true; LOW else which is equivalent
  // to a boolean false
  input = digitalRead(INPUT_IO);
}
```

Another interesting algorithm is the trapezoidal rule which we implemented in order to integrate the acceleration data.

```
// Define the weights to use.
// How those weights are distributed is defined
// by the Newton-Cotes formulars.
// More weights mean a better interpolation of
// the integral.
private float[] weights = new float[] { .5f,.5f };
```

```
public List<Vector3> Integrate(List<Vector3> input)
{
  var output = new List<Vector3>() {new Vector3(0,0,0)};

  for(int i = weights.Length - 1; i < input.Count; i++){
    output.Add((weights[0] * input[i - 1] + weights[1] *
        input[i]) * SamplePeriod + output[i-1]);
  }
  return output;
}
```

2.4.2 Flaws

One of the obvious flaws is the fact that we had to use two [Flora]s. This was neccessary because we did not plan to use that many digital IOs. Simply using a different Arduino board that retains a small formfactor but sports more digital IOs would solve this issue. This would also half the amount of batteries needed. Another big flaw were the motors used for our prototype. More precisely the robustness of the contacts. After a few hours of usage we had to resolder three of them and during the exhibition even more failed rendering our prototype useless. Sadly also the foam rubber we used to integrate our motors into were to soft and the vibrations were not reliably felt by the users. Last but not least we have to admit that we were not able to record and translate the motions life. We had to manually trigger the correct pulses in order to craft an enjoyable experience for test users.

All in all a lot of work has to be done to create the product we had envisioned at the outset of the project.

3 Interesting Links

We want to give a quick overview over the parts we actually ended up using and also provide you with the link to our GitHub repository where we uploaded all the code that was used for the final prototype and some of the visualisation in section 2.2. Please note that, although we do link to a certain online shop, we do not mean to endorse it. It is merely meant as a starting point to help you in your research if you would be interested procurring the parts to actually recreate our work.

References

Bosch BNO055. Bno055. https://cdn-shop.adafruit.com/datasheets/BST_BNO055_DS000_12.pdf. Accessed: 2016-10-14.

Adafruit Flora. Flora. https://www.adafruit.com/products/659. Accessed: 2016-10-14.

GitHub. Mirror. https://github.com/NemoNessuno/Mirror. Accessed: 2016-10-14.

Adafruit LiPo. Lipo. https://www.adafruit.com/categories/916. Accessed: 2016-10-14.

ProcessingFoundation Processing. Processing. https://processing.org/. Accessed: 2016-10-14.