Long Distance Intimacies

Iris de Vries - Jan Dudek - Lal Avgen

Hardware & Physical Computing
Media Technology, Leiden University

Paul Jansen Klomp
TA: Daphne Wong A Foe

Table of Contents:

1. Introduction

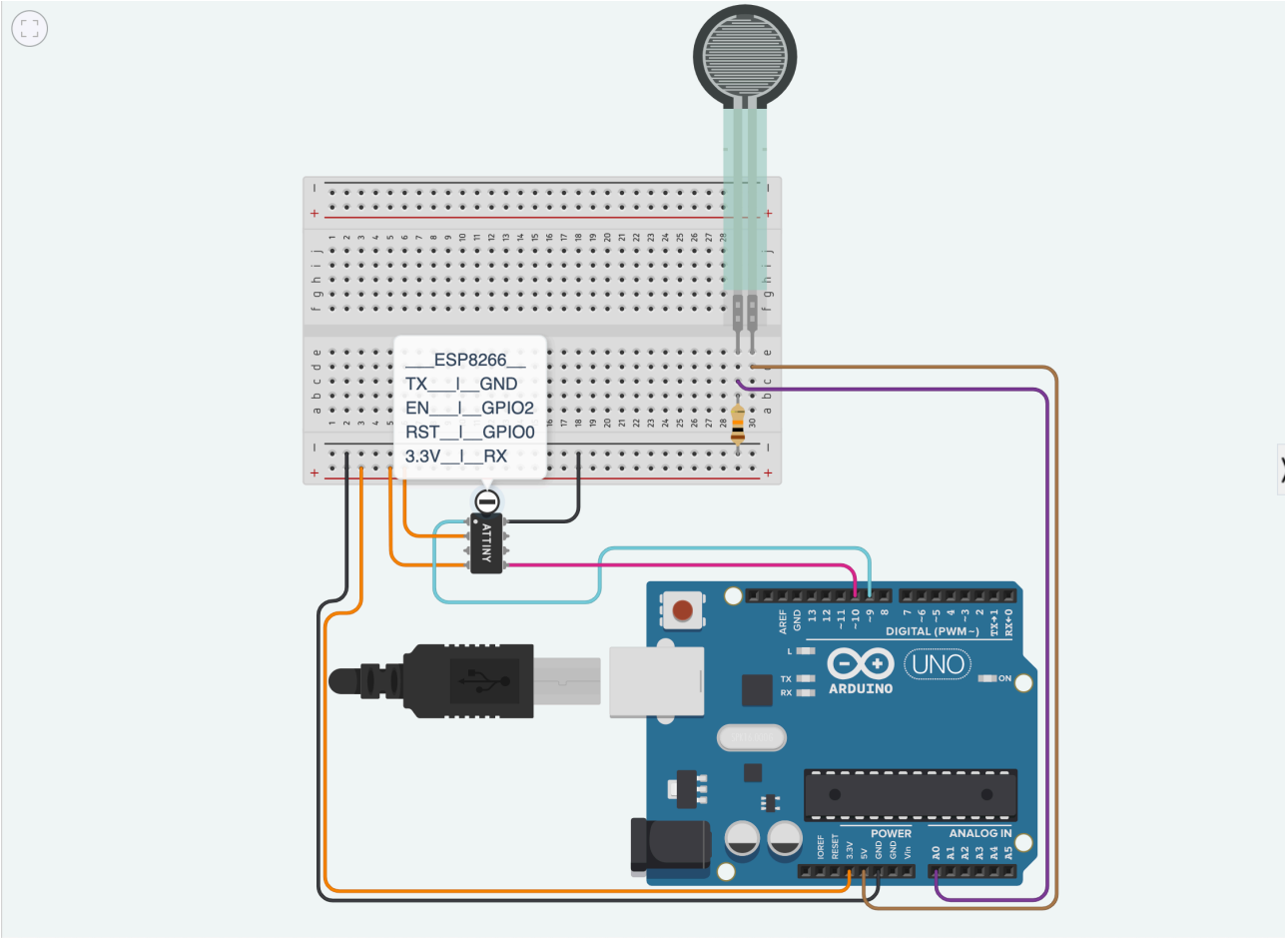
The goal of this project is to create a set of objects, input and output, allowing two users to interact with each other and to project the sense of intimacy over the long distance. Kinaesthetic interaction with an input device fitted with a sensor is expected to produce a haptic response on the receiver end.

2. Prototype

2.1 Transmitter

Input object consists of set of lips 3D printed in silicone filament, fitted with FSR sensor; further, sensor is connected to the Arduino UNO board which processes the force reading and writes the value in its serial monitor.

Arduino UNO board is connected to an esp8266-01 chip, preprogrammed for reading the incoming serial messages and, if able to connect to wifi, passes the value along to the receiver.



The design and development of the transmitter was possible thanks to **Iris de Vries**, who has been in charge of this section of the project.


17-11

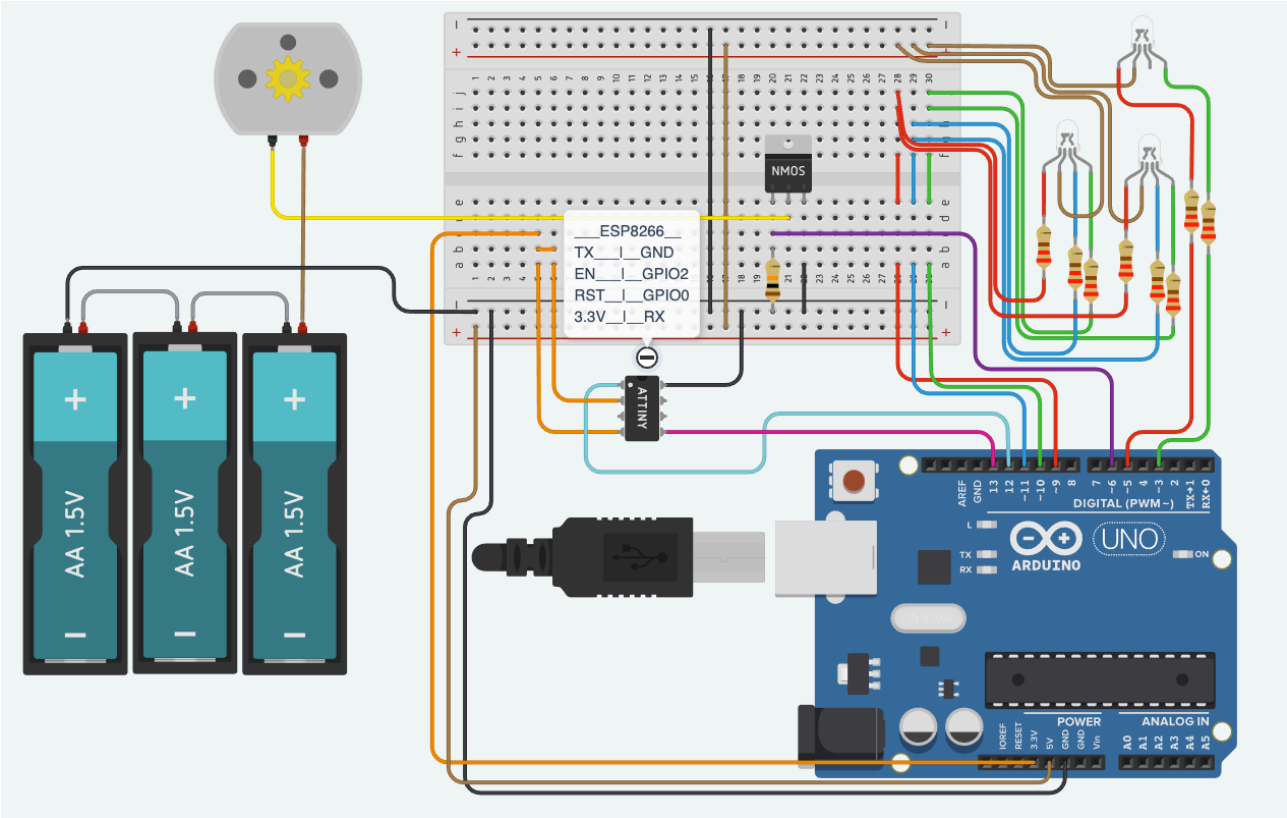
2.2 Receiver

On the end side of the system lays second esp8266-01s chip, which when connected to the wi-fi downloads the incoming data send by the transmitter and through the serial monitor passes it to the Arduino UNO board.

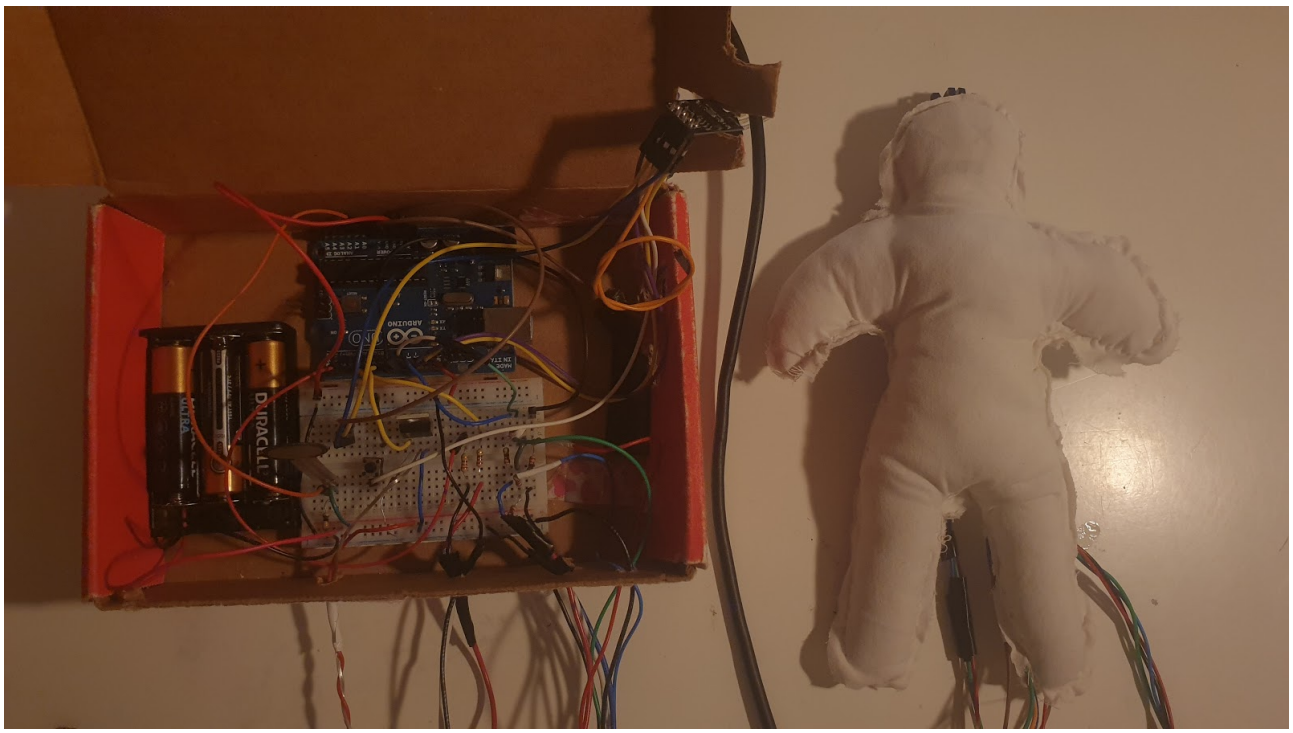
The board processes the data and based on it's value projects controls the actuators, which are:

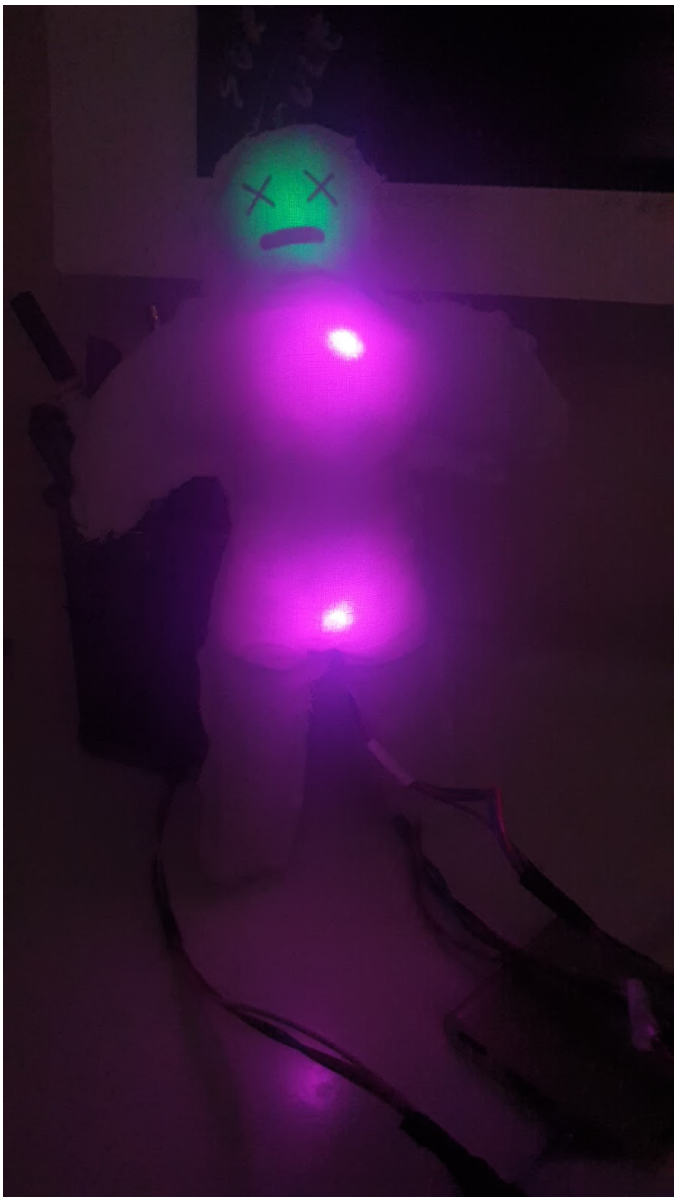
- status RGB LED - informs whether there is any data incoming from the transmitter (boolean, red / green).
- intensity RGB LEDs - inform about the intensity of interaction with a transmitter, as well as the duration of the interaction; two of these, grouped together for an enhanced visual effect.
- DC vibration motor - projects the intensity of interaction with a transmitter as a haptic sensation.

All of the above has been encased inside a cotton doll, while the Arduino UNO board has been fitted into a box. The esp chip and all of the LEDs are receiving power through the Arduino board, which itself is powered by 5V serial connection, while the motor receives the 4.5V, directly from the external battery pack; the current flow is controlled by the board through IRLZ44N MOSFET.



The output system has been a contribution of **Lal Avgen,** who has been responsible for creation and execution of the receiver object.


ARDUINO
DURACELL

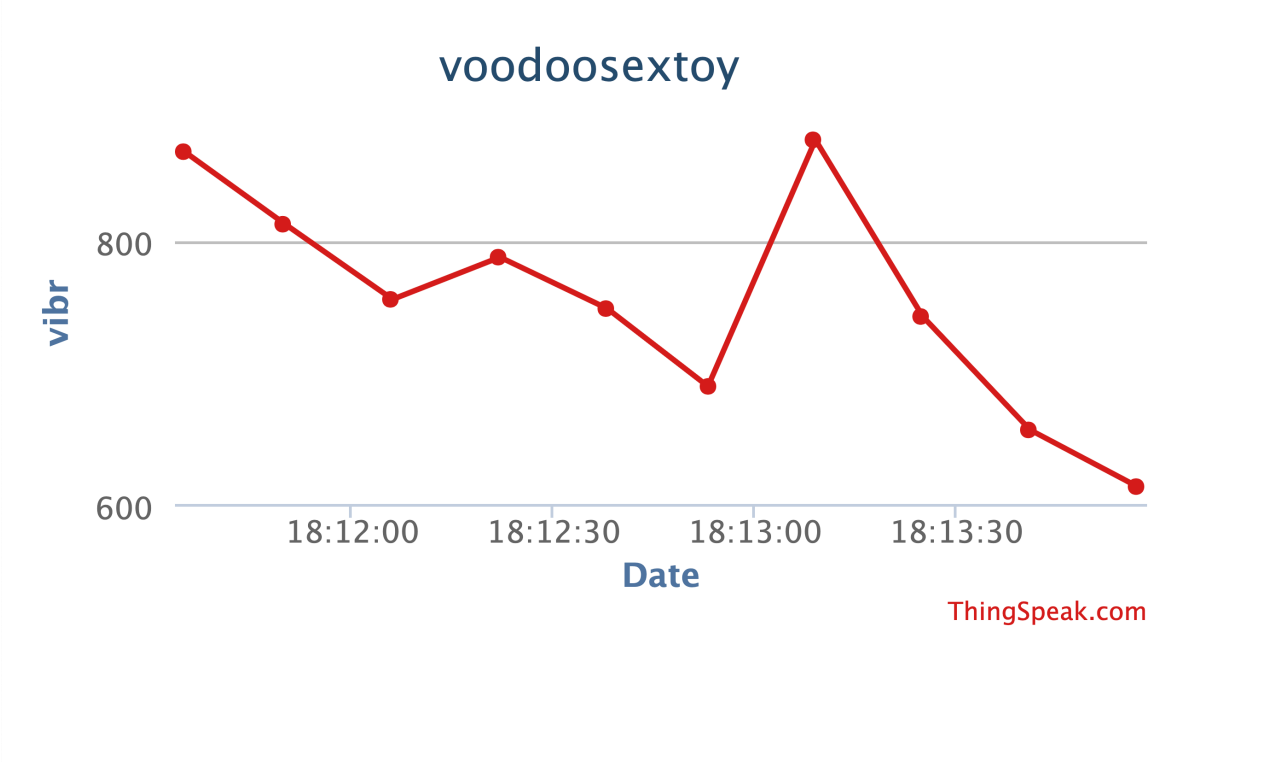


2.3 Connection

The transmitter and the receiver communicate with each other thanks to build-in esp8266-01s chips on both sides. Each of the chips has been preprogrammed with the credentials allowing it to connect to a WLAN. If successful connection is established, the transmitter chip reads the serial data from the Arduino (at 115200 baud rate, contrary to 9600 at which Arduino communicates with IDE) and writes the data to the server.

For this project we settled on using ThingSpeak platform developed by MathWorks. This approach allows us to simplify the project by removing the need for the receiver to actually "receive" the data. The platform provides an API which stores the data whenever an http request (containing a write key and the variable) is executed, and returns the data upon execution of an http request containing a read key. By doing so we've removed the problem of exact timing; the devices don't necessarily need to speak to each other in perfectly real time, as the receiver can just read the last uploaded variable whenever it executes the read command. Another benefit of using the platform is that MathWorks provides a dedicated library for Arduino and ESP allowing for an easy extraction of the integer from an HTML string.

As mentioned above, the receiver ESP chip (provided it successfully established a connection) makes an http request and strips the response to a simple integer. Said integer is then being written in its serial monitor (again, at a separate baud rate than Arduino - IDE serial communication).



Member of our team responsible for establishing a successful connection between the chips was **Jan Dudek**.

3. Development

3.1 Stability of power

One of the problems we have encountered was that esp chips require very stable power at 3.3V to operate correctly. If there is any disruption they will lose the WLAN connection which in result renders the whole system inoperable. Example of said disruption would be a DC motor pulling high amperage from the board.

To solve aforementioned problem we've decided to power the motor separately, through a pack of batteries; this allowed us to prevent the motor from influencing the voltage delivered to the chip.

3.2 EMI

While connecting the motor through external battery solved the voltage drop issue mentioned in 3.1, allowing motor to run for a short while, after few seconds the esp still drops the connection. Suspected problem here is connected with Faraday / Lenz law, i.e. that motion of the motor creates an electromagnetic field which in result may interfere with the WLAN waves, preventing the esp from successfully receive the data.

This problem has not yet been solved as of the current prototype.

3.3 Resolution

As with any digital system, actual, real life transfer of sensation is never perfectly possible. Human touch is a continuous phenomenon, while any protocol will be only able to transfer distinct, quantised values. In case of our project, this limitation applies on several levels; the heaviest one present in the prototype is that with our MathWorks plan ThingSpeak platform only allows to write the channel once each 15 seconds. While the purchase of more able yearly plan would lift this limitation, another bottleneck comes from the esp chip itself; as each time it needs to receive and process response to from a server, it takes it around 400-500ms to execute the loop.

We have decided not to address this issue further; as actual perfection is never achievable and the chip used in a prototype cannot achieve better resolution than around 2Hz, we assume the prototype works as intended. Each 15 seconds RX chip does read the value 30 times, and TX chip does try to write the value 30 times. Anyone willing to increase the resolution can just simply purchase the upmarket plan.

3.4 Serial Communication

Another problem difficult to tackle was retrieving the integer from the serial monitor during the esp8266 - Arduino UNO communication. We've encountered numerous errors while trying to read the data, which in some cases gave the correct value in some cases not. The problem was rooted in what writing serial monitor actually does, i.e. sends a stream of bytes instead of distinct variables, like strings or integers. Hard-coding the variable to be an integer did not solve the problem; while it did started receiving only integer values, where to divide them remained unclear for the chip and it kept producing unexpected values.

To solution came with replacing the simple read function to .parseInt(); method from SoftwareSerial.h library; it cuts the stream whenever it receives non numerical value and allows to correctly extract the integer.

3.5 Void Data

Due to uncertain stability of the esp chip an issue prone to occurrence was an algorithm (on the receiver end) trying to update the variable responsible for controlling the motor and LEDs during a disruption in connection. In such case, the variable would be often updated to zero or (for some reason) 488, causing the actuators to misbehave. Another similar problem was, in case of multiple requests by the Arduino to fetch the data from serial monitor, made at different points of the loop's cycle. In such case the algorithm would pull the different value for motor and different for LEDs.

To tackle this problem we've decided to declare the discussed variables globally, and perform the update in a if(){} function, under the condition of a successful connection to the esp chip. By doing so, we ensure that no data gets updated in case of a disruption (in such case the motor and LEDs will "lag", i.e. keep performing at the last successfully updated value) and whenever it is updated, the data is fetched at a one, single point of the loop and stored in the Arduino board for further computations.

3.6 Testing

Given the format of the project, i.e. two devices that need to communicate with each other in real time, it proved to be greatly impractical to test the project as a whole from the very beginning. The receiver cannot be tested for receiving of there is no data being transmitted, and vice versa in case of the transmitter.

For this very reason we have run the tests for each element at a time, subsidising the transmitter by a python script. Given the nature of ThingSpeak environment, i.e. requesting an http in order to write data, creating such framework was rather straightforward, only thing necessary is the requests library, which eponymous purpose is, handling http requests.

Testing the performance of a transmitter proved to be even simpler, as ThingSpeak allows to follow the data written to the channel from their web app.

Once the performance of both ends was confirmed with the aforementioned methods, we started performing actual tests with combined setup of TX and RX.

4. Result

The result of our project works as intended, with aforementioned limitations. Thus:

- Both the transmitter and the receiver connect to WLAN and transfer the sensation;
- The WLAN credentials are within the esp code, hence the chip needs to reprogrammed in order to work with a specific network;
- Due to EMI generated by it there comes a disruption in a work of the motor;
- Resolution currently is one message updated each 15 seconds;
- Highest achievable resolution with upmarket ThingSpeak plan is one message each 500ms;

5. Licensing

In a current state due to reliance on the TeamSpeak platform, prototype is licence for only small, non commercial projects, with limited annual usage. MathWorks offer separate plans for commercial, governmental, academic-institutional use, which can be purchased with regards to the needs; further information regarding specific plans available here.