Engineering Modular Playware

Henrik Hautop Lund, Morten Roed Frederiksen, Massimiliano Leggieri
Center for Playware, Technical University of Denmark, Building 326, 2800 Kgs. Lyngby, Denmark

E-mail: hhl@playware.dtu.dk
www.playware.elektro.dtu.dk

Abstract

In this paper, we describe the engineering of modular playware. Constructing playware systems out of individual modules allows the system to be easily transported to be used anywhere, to easily build different bodies and brains, and to allow people to construct and combine the modules in a creative fashion. Thus the modularity helps fulfilling the Playware ABC. We exemplify the engineering challenges of the hardware, software and communication design of modules through the development of handheld modules for playful rehabilitation. The handheld modules allow for a fun and motivational training of upper extremities, and can be viewed as an extension to the Moto Tiles for lower extremity training, which have proved to be a highly successful playware for prevention and rehabilitation among seniors. The paper outlines the engineering challenges and proposed solutions to make such modular playware for playful upper extremity training.

Keywords: Playful technology, Playware, Personal Health Technology, Adaptive Games.

1. Introduction

Playware is defined as intelligent hardware and software that creates play and playful experiences for users of all ages [1, 2], and such playware has been developed for a number of application areas including rehabilitation [3, 4, 5]. Within this area of application, the playware for rehabilitation can be viewed as games for health, though playware puts primary focus on play rather than on games. Playware objects are often designed to match the plug-and-play mentality of modular technology, since such design strategies help facilitate the realization of the concepts of Playware ABC. The Playware ABC entails developing technology for Anybody, Anywhere, Anytime by Building Bodies and Brains, which facilitates that users can Construct, Combine, and Create [6].

Approaching the constructing of playware systems with a focus on modularity, allows the systems to be easily distributed and put to use anywhere, to easily build different bodies and brains, and to allow people to construct and combine the modules to become creative. Hence, modularity often helps when aiming to fulfill the Playware ABC.

Modular playware units act as interactive nodes in a network. The control of the modules can be either fully contained in the embedded systems modules themselves, or the control can, for instance, be divided between on-board control on the embedded system modules and control from a network connected smart device, e.g. tablet or smartphone. In the latter case, communication from the smart device to a server backbone may provide the possibility of data collection and data analysis. In the work presented here, we investigate this latter case and thereby the engineering research examines the division of control between the individual modules and the smart device, and the best strategy for robust communication between them. The focus is working on low level

software for controlling the embedded systems; along with the investigation of the on-board sensors and actuators of the boards with the objective of providing playware experiences for the users.

One example of this for lower extremity interaction are the Moto Tiles [4, 7]. For upper extremity interaction, we develop and investigate hand-held modular playware devices. More specifically, for Moto Tiles using NordicRF microchips for developing ANT+ radio communication protocols, and for the hand-held devices using ESP microchips, which provide WiFi connectivity between devices as well as between devices and the Android device. The Internet of Things has opened a new world of possibilities by using protocols that have been around us for a while, such as TCP/IP. The abovementioned ESP chips currently have become the most inexpensive and most accessible choice for this kind of projects involving WiFi communications. Furthermore, the amount of energy consumed by the device, places it as an efficient utility for low-energy and low-cost working contexts. However, using the ESP modules poses a challenge for both the embedded software and the client systems that communicates with them, as they can be error prone and sometimes proves unreliable in fast connect-disconnect scenarios.

Hence the following questions arise:

- What are the software measures needed in order of working around the unpredictability of using inexpensive modules for swift real time communication?
- How can such communication between modular devices be used to create a playware experience by interacting with the users?
- Which sensory modalities and HW implementations can be used reliably to track simple upper body movements in a manner as robust as seen from the sensors of the Moto Tiles in order to provide an equally robust interactive playware e.g. for the elderly performing upper extremity play?

By answering these questions through our research and achieving robust, simple fault tolerant communication between hardware modules, we aim to develop a methodology for designing and adapting playware game activities to motivate people to perform the desired actions. The research methodology [8] is based on iterative prototyping, along with prototyping and testing in laboratory conditions with external probands. Hence, knowledge is built from iterations of synthesis and application. Here, Moto Tiles serves as an example at a much later iteration stage than the hand-held modules at an earlier iteration stage of rapid prototyping, illustrating the research methodology and implementation.

2. Rehab app with handheld modules

The goal of this application is to engage the elderly generation in pulse-increasing activities while having fun. It specifically targets the solar plexus area and focuses on accurate arm movements in a fun and motivating setting, utilizing modules that can be used by Anybody, Anywhere, Anytime according to the Playware ABC (see Fig. 1).

Hardware setup

The current setup consists of custom built ESP modules running NODEMCU/Arduino firmware with custom tailored components to facilitate communication over WiFi. An external barometric pressure-meter has been added to the board to approximate the pressure at different altitude levels. The hardware sensors employed in this project consists of: raw accelerometer data from an onboard accelerometer, and barometric pressure data. In details, we use:

- The ESP8266 Tetra module, which is a low-cost Wi-Fi microchip at 80MHz with 32 kB memory for instruction and 80 kB for user data.
- The Adafruit BME280 I2C Temperature Humidity Pressure Sensor, which measures humidity with ±3% accuracy, barometric pressure with ±1 hPa absolute accuracy (used for altimeter), and temperature with ±1.0°C accuracy. It has a low altitude noise of 0.25m and a fast conversion time.
- The Adafruit 9-DOF Absolute Orientation IMU Fusion Breakout - BNO055, which provides a MEMS accelerometer, magnetometer and gyroscope on a single die with a high speed ARM Cortex-M0 based processor to digest all the sensor data, abstract the sensor fusion and

real time requirements away, and e.g. provide sensor data such as Absolute Orientation (Euler Vector, 100Hz), Three axis orientation data based on a 360° sphere, Absolute Orientation (Quaterion, 100Hz), Angular Velocity Vector (100Hz), Three axis of gravitational acceleration (minus any movement) in m/s^2.

Software setup

ESP firmware

The current iteration of the ESP firmware is utilized towards sending as much sensor data each second as possible. Original versions of the firmware were designed to be more versatile with incoming socket connections to accept incoming messages. However, to gain most of our sensors, the later versions skipped this part and focuses solely on sending out as much information as possible. The cycle is as follows:

1) Once turned on, the ESP module attempts to connect to previously stored WIFI access point.
2) If it fails it will become a hotspot itself and can be accessed at http:://192,168.4.1. Here it attempts to discover any nearby WiFi access points and lets the user select them from a given list. Any selected point is stored along with a user-entered password.
3) When it successfully connects to an access point, it broadcasts its services via MDNS/Bonjour protocols and can be discovered by a client.
4) When a client connects via the MDNS address to the outgoing TCP socket at port 8885, it sends out all given sensor data until the socket connection is broken.
5) The current format for the data transfer is JSON.

Client software

The device(s) is discovered via MDNS browsing software that basically scans the dans after ".local" addresses that match the service profile. Any found device is passed off to a handler thread that establishes the connection. The thread connects to the socket address at port 8886 and starts reading data. Once a complete JSON string is received it is parsed and added to a buffer-queue with sensor readings. The client software was developed both in JAVA for android devices as well as in SWIFT for IOS devices. Both platforms performed equally well in similar fashion.

Improving the level of fault tolerance

During the testing phase of the development of the playware modules, various errors were encountered in different phases of the process flow. Randomly the ESP modules they would fail to detect various onboard sensors during the initialization startup phase. To counter that, a measure was implemented to automatically restart the device, for the state where not all hardware checks had been passed. This could result in a never ending reset loop, so a visual cue was added to indicate the device startup phase (blinking led lights). This meant that a user could manually switch the device off and on again if the automatic restart did not solve the problem.

At the connection phase the modules would arbitrarily fail to connect multiple times. The source of this error might have been interference from other WiFi signals, however this was not verified. As a simple solution, the ESP firmware would allow two attempts at reconnecting before initiating the state WiFi access point. Defining the amount of reconnect attempts that should be allowed, is a tradeoff between how quickly the user should be able to reconfigure the device for a new working context, and how robust the connection phase should seem for an already configured device. Sometimes the ESP devices would disconnect the socket connections without any indication in the embedded software. To resolve that problem, an implementation of idle counters was added to both client and embedded software parts to see whether any data was transmitted or not. If no data was transmitted above a specified threshold amount of times, the ESP modules would enter idle mode and start re-accepting incoming socket connections. The client software would in such case attempt to ping the device to investigate whether the device was still connected to the same network. If the device answered, the client would reconnect. If the device was unresponsive the client notified the user that a module had disconnected.

3. Motion interpretation

The data gathered from the specific sensors are translated into specific movement patterns in two isolated steps. Some parts of the data received from the sensors are

rather not precise and needs to be averaged before they can be used. The two steps are detailed in the subsequent sections.

Step 1 Averaging the data

We experimented with different buffer sizes to find a suitable match between speed and accuracy. We measured single runs of each step of the algorithm and found that each iteration takes about 20ms to complete letting us read complete samples at 50hz. However, some fitness arm movements are a lot swifter than a second so we opted for a buffer of 40 samples with a moving frame which seemed to fit our needs. All samples are averaged and we calculate the following data points:

• Average acceleration across x,y,z axis.
• Average rotation across roll, pitch, and yaw.
• Average altitude. Note that this number is already filtered on the ESP device as well using an average of 22 samples.
• A boolean value that determines whether a user defined acceleration threshold has been reached.
• A direction of altitude movement UP, DOWN or LEVEL based on the last 25 calculated average altitudes. If all but three values fall in one category it is determined that a movement in that direction happened.

For the altimeter data, we employed low pass filtering on all raw samples as well as the averages to hinder highly fluctuating output. Note that the sensors are placed inside a ball, why it makes sense to discard x,y, and z as isolated data.

Step 2 Using the information in a health setting.

The system obtains information on roughly what altitude the module is held (with occasional spiking errors), and if it is rotated or not. We employ all these features in a setting that captures the user's imagination (They believe they're tracked 1:1) and at the same time allows a certain movement freedom.

A separate thread handles all movement verification to allow the fetcher thread to gather data as fast as possible. With each looping round it fetches the calculated movement status. A target arm positional state is randomly selected for each controller: UP, DOWN or SIDE and different movement patterns are expected for each of them.

Example - target state UP:

For this state to trigger we expect the user to lift his or hers arm. This entails moving the module up a bit, rotating it a bit and keeping it still at a certain height. These are the signs we look for. With each iteration/measure, we see if the module has rotated. If it has, we increment the rotation counter. Once this value surpasses a fitting threshold, we set a HasRotated value to true. The same goes for correct altitude changes (we make sure the ball moves upwards). Once moved and rotated we expect the user to hold the ball still at the same level.

When that has occurred we accept the target as reached and resets the cycle with a new target state. Note that we could employ pitch, yaw and roll values for each position to assist the algorithm but we found that people shift their wrists around too much when they hold objects, making it impossible to determine if a target position has be reached using that specific data. It simply shifts around too much. The algorithm works well when the values and threshold have been fine tuned so the user don't have to hold still in outer positions too long. In such cases where the algorithm reacts slowly it gets hard to complete longer workouts.

We acknowledge that such slowness holds a certain fitness value, that might suit another perhaps younger audience. All in all the algorithm seems to fit well for the target audience, in that it is very forgiving for people that might not be able to reach outer arm positions or lack strength to move swiftly.

Step 3 game implementation

To make the game fun we have added a very simple scoring system that increments a counter for each correct position reached individually for each arm. As the game is started a progress counter starts crawling across the screen. In 30 seconds the player has to complete as many arm positions as possible. The final score depicts an ability to perform movements and stay coordinated in a stressing situation. It gets increasingly harder when two controllers are used with different target positions for

each arm. To facilitate training also the disabled, the game works just as well with a single controller.

Fig 1. Playing the rehab game for upper extremity training with two playware modules in the form of handheld balls.

Fig 2. The interface on the tablet when playing the rehab game for upper extremity training with two playware modules.

4. Future iteration and extension modules

With the calculated sensor data, there are no limits to the type of fitness games that we can make using the relative simple building stones of subsequently employed movements. For this game a high score of multiplayer component would be ideal but the real strength lies in how easy it is to pickup and start using. A developer kit has been made available for computer science classes to develop their own applications with the module and client software.

Fig 3. The tablet interface showing score.

References

1. H. H. Lund, and C. Jessen, "*Playware - Intelligent technology for children's play,*" Technical Report TR-2005-1, June, Maersk Institute, University of Southern Denmark, 2005.
2. H.H. Lund, T. Klitbo, and C. Jessen, "Playware Technology for Physically Activating Play", *Artificial Life and Robotics Journal*, 9:4, 165-174, 2005
3. H. H. Lund, "*Play for the Elderly - Effect Studies of Playful Technology," in Human Aspects of IT for the Aged Population. Design for Everyday Life*. (LNCS Vol. 9194, pp 500-511, Springer-Verlag, 2015)
4. H. H. Lund, and J. D. Jessen, "Effects of short-term training of community-dwelling elderly with modular interactive tiles," *GAMES FOR HEALTH: Research, Development, and Clinical Applications*, 3(5), 277-283, 2014.
5. H.H. Lund, M.D. Pedersen, R. Beck, "Modular robotic tiles: Experiments for children with autism", *Artificial Life and Robotics, 13*, pp. 393-400, 2009.
6. H. H. Lund, H.H. "Playware ABC: Engineering Play for Everybody." *Journal of Robotics Networks and Artificial Life, 3*(4), 2017
7. http://www.moto-tiles.com (checked: 6/12/2018)
8. H. H. Lund. "Playware Research–Methodological Considerations". *Journal of Robotics, Networking and Artificial Life, 1*, pp.23-27, 2014