Bachelor Thesis

Virtual usability test for digitally controlled hand prostheses

Johannes Schoisswohl, e1327384
Technical University of Vienna

July 25, 2017

Supervisors:
Cosima Prahm, MSc. BA., Medical University of Vienna
Eugenijus Kaniusas, Ao.Prof. Dipl.Ing. Dr.Techn., TU Vienna
Christian Fermüller, Ao.Prof. Dipl.Ing. Dr.Techn., TU Vienna

Contents

1 Abstract

Modern prosthetics makes use of sophisticated technologies for implementing control modules, but only rather simple techniques are used for assessing the usability of those modules. Since providing a prosthesis that is suitable for everyday use is the target of development, the controller's usability is the main asset for the end user. Therefore a virtual usability test for prostheses control systems was developed in the course of this thesis.

The program provides a default level, inspired by the Box and Blocks Test, a simple possibility for adding more levels, and one more level as a proof of concept. Furthermore the program involves a simple C# programming interface, to let developers add their own controllers.

An experiment with 14 participants of different ages and gender, was conducted to detect possible problems with the program and to find out whether users felt motivated by the game.

2 Introduction

Techniques for controlling prostheses made a lot of progress in the last decades. Especially myoelectric control systems evolved from rather simple devices, controlling only one degree of freedom, to more complex systems, offering simultaneous control of multiple degrees of freedom [1, 19]. For a natural means of control, two EMG[1]-electrodes, measuring excitation of two independent muscle groups, should be available per degree of freedom [1]. This control strategy is also called direct control. Unfortunately, in practice, the EMG is obtained by surface electrodes, which only provide a mixture of the electric signals of all muscles located close to them. Since neighbouring muscles are often responsible for very different movements[2] direct control in multiple degrees of freedom is mostly not possible in practice [1]. This might be one of the reasons why only rather simple systems, controlling only one degree of freedom at once, are found in commercial products [13]. Nevertheless, due to the progress in pattern recognition, academic research made use of machine learning techniques, classifying complex signals from arrays of EMG-electrodes for more sophisticated control modules [1].

Since machine learning systems are always trained using a ground truth data set G, their classification error rate can always be evaluated by training the algorithm using a random subset $T \subset G$ and testing the resulting classifier on the rest of the data $G - T$. Therefore machine learning systems are always evaluated well on a low level.

However a good classification rate is not necessarily related to good usability of the finished control module. Therefore the evaluation of the usability is an important part of the process of developing new control modules. This is the step of development is the one this thesis deals with.

The currently most common way to handle the task of evaluation is to let probands participate in occupational therapeutic tests. There are different tests like the Box and Blocks Test [8, 16, 2, 5], the Nine-Hole Peg Test [8], the Clothespin Relocation Test [5, 19], the Block Stacking Test [5], and the Southampton Hand Assessment Procedure [14], to measure different aspects and levels of manual dexterity.

To enable patients to participate in such an occupational therapeutic test, either a prosthesis needs to be fixed to an able-bodied's arm with a special construction [8] or a prosthesis needs to be personally adjusted for an amputee [2]. In both cases a socket needs to be adjusted to a participant's needs, which is time consuming as well as costly. Another problem with executing standardized occupational therapeutic tests is the little amount of data that is obtained from a single run. The performance of a patient in the Box and Blocks test, for example, is simply measured in the amount of boxes the patient manages to move within one minute [16]. Once the test is over there is no possibility to find out why a patient might have had a good or bad result.

[1] **e**lectro**myog**raphy

[2] e.g.: The muscles of the forearm are responsible for individual finger movements [12].

The target of this bachelor thesis is to develop a virtual test to evaluate the usability of prosthesis-controlling modules to overcome these issues.

3 Methodology

3.1 Target specification

- Standardized tests are repetitive tasks, which may cause patients' boredom. Therefore the test should include some simple gamification elements to enhance the users' motivation.

- The program should offer a default level, which should be as difficult as one of the established occupational therapeutic tests.

- Furthermore it should be easily extensible so that custom levels can be added without being forced to write a single line of code. The game should track enough data to make it possible for developers to reconstruct each game run. This should help developers to not only find out whether the controller is usable or not but also about which specific controller movements did possibly not work well for the user.

- The program aims to offer testing prostheses controllers with two degrees of freedom. Moreover the program should force the user to use both to complete the task.

- The game should offer a simple programming interface so that developers can easily add their own control modules into the game. Besides that a simple keyboard-driven controller should be implemented for the purpose of debugging custom levels. Additionally at least one EMG-driven controller should be implemented as a proof of concept.

Since the Box and Blocks Test is one of the most widely used and most simple occupational therapeutic tests used for evaluation of the upper limb function [8, 15, 2, 5], the default level should be as difficult as the Box and Blocks Test. It is important to point out that the target is not to implement a simulation of the Box and Blocks test[3] but to create a framework and use it for implementing a virtual test that offers testing the same level of fine motor function.
As described in [16], the Box and Blocks Test consists of a rectangular box with an open top. The box is separated into two halfs by a board, which is higher than the box' walls. The task for the participant is to move as many wooden cubes as possible from one half of the box to the other within one minute.

[3]This was already done by others as described in [15].

3.2 Technologies

The program is implemented using Unity® 5.6.1f1 Personal [25]. Unity® is a cross platform framework for creating computer games. All physics and graphics calculations of the game are handled by the game engine. Furthermore the framework provides a C# [18] interface and access to the .Net Framework [4] for implementing the game logic. Therefore the game was implemented using C#. Additionally Unity® offers an editor for 3D modelling, managing dependency injection, creating animations, and many other tasks related to developing a computer game.

The default EMG-driven controllers are implemented using the Thalimc Myo wristband [6] (short: Myo). The Myo offers an array of 8 EMG-electrodes and can be connected to a computer wirelessly using a USB bluetooth dongle. Besides the EMG it includes a three axis accelerometer, a three axis gyroscope, and a three axis magnetometer built into the Myo.

Since the program is planned to be used to evaluate the usability of Transfer Learning to retrain an Echo State Network-driven controller, one of the default EMG-driven controllers will be implemented[4] using an Echo State Network as introduced in [21].

The game outputs user data for analyzing game results as files in JSON-format [11].

The 3D models used within the Unity® were created using Blender[3].

3.3 Software modules

From the developer of a prosthesis' perspective the software can be seen as three modules.

The core module handling all logic calculations, the user interface of all menus, and the persistence of the game data.

The level module lets the developer add custom levels. A level can be created by just placing Unity® game objects in the scene using the Unity® editor.

The control module provides the possibility to connect the hardware prosthesis controller to the game. Therefore it provides an abstract C# class that must be implemented and attached to a Unity® game object to be used in the game.

[4]The actual implementation of the network and finding a workflow to get good results was not done by me but by Cosima Prahm, Benjamin Paaßen, and Alexander Schulz.

4 Implementation

The full implementation is accessible on the public GitHub repository `https://github.com/joeschman/BoxAndBeans`.

4.1 Gameplay

The game screen shows a brown box with an abstracted yellow hand representing the prosthesis and a some bean-like shaped objects. The target of the game is it to move as many beans as possible to a level specific target area within a fixed amount of time.

While the graphics is rendered in 3D, the gameplay takes place only in two dimensions. Therefore the player can freely move the hand around on the screen. Furthermore he can open, close, and rotate the prosthesis around the axis that is perpendicular to the plane the game takes place on. How the movements are controlled exactly depends on the selected controller module[5] [6].

When starting the game the beans are floating around in the game plane without rotating and without being influenced by gravity. The player has to open the hand and rotate it so the hand can be moved over the centre of a bean without pushing the bean away. When the hand is located above the bean's centre the player has to close it to grab the bean. When the bean is grabbed its rotation gets unfrozen and the player has to move it to one of the level-dependent target areas. A bean counts as collected once it is dragged and released there. When the bean is collected gravity starts getting applied to it. If a collected bean leaves the target area, it does not count as collected anymore and falls back into the initial state of no gravity being applied to it and the rotation being frozen. If all beans are collected a new bunch of beans is spawned. How many beans and where they are spawned at once depends on the current level[7].

The reason for freezing the beans' rotations is to force the user to rotate the hand to grab it. The reason why gravity is not applied to them at the beginning is, that it is sometimes very hard to grab the beans when they are stacked on top of each other due to Unity® 's physics engine.

Basically the box, the game takes place in, has a back wall, three visible side walls, plus a transparent wall on the top to prevent the beans from floating off the screen. Besides this fixed setup the box has some additional level dependent walls. There are visible as well as invisible walls. The visible walls behave as suspected just like walls do: Nothing can pass through them and they can be used by the creator of a level to structure the level and to mark and separate the level specific target and spawning areas. Invisible walls are used to keep the beans from floating around everywhere on the screen. Beans cannot pass through them before getting grabbed for the first time.

[5]See 4.1.2 for which control modules were implemented for this thesis.

[6]How to exactly the controller modules work and how to add them to the game can be found in 4.4.1.

[7]Details on how the beans are spawned can be found in 4.4.2

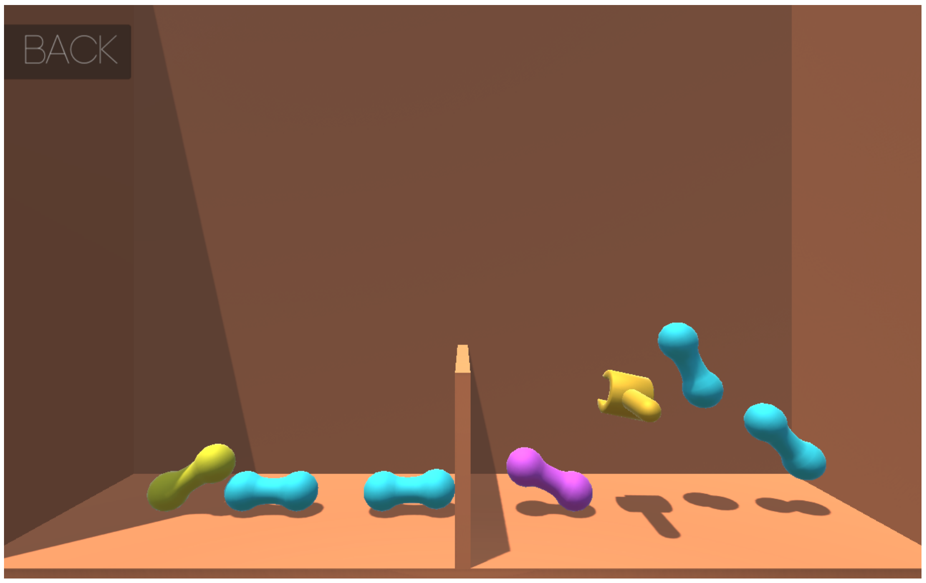


Figure 1: Level “Bean Moving”

Besides the normal mode, where the player has to collect as many beans as possible in a fixed amount of time, there is also a Try-Out mode. This mode is used to let the player get familiar with the game environment without being influenced by the pressure of a timer or a score counter.

4.1.1 Default levels

Two levels are implemented: The default level, called “Bean Moving”, inspired by the Box and Blocks test, and a second one, called “Bean Stacking”. The latter has been implemented as a proof of concept, that it’s possible to add more levels.

Bean Moving (Figure 1) Since the first level should be similar in difficulty to the Box and Blocks test, it is heavily inspired the test’s setup. There is one visible wall in the middle, to separate the target from the spawning area. Beans are spawned in the right half and have to be moved to the left half. An invisible wall separates the two areas to prevent the beans from floating to the target area before being grabbed.

Bean Stacking (Figure 2) In the second level there is one spawning area on the left and another on the right side of the screen. In the centre there is a horizontal wall. The target area covers the parts of the game plane that are are above the horizontal wall. Since the platform is quite small the player has to stack the beans on top of it. This level does not contain any invisible walls.

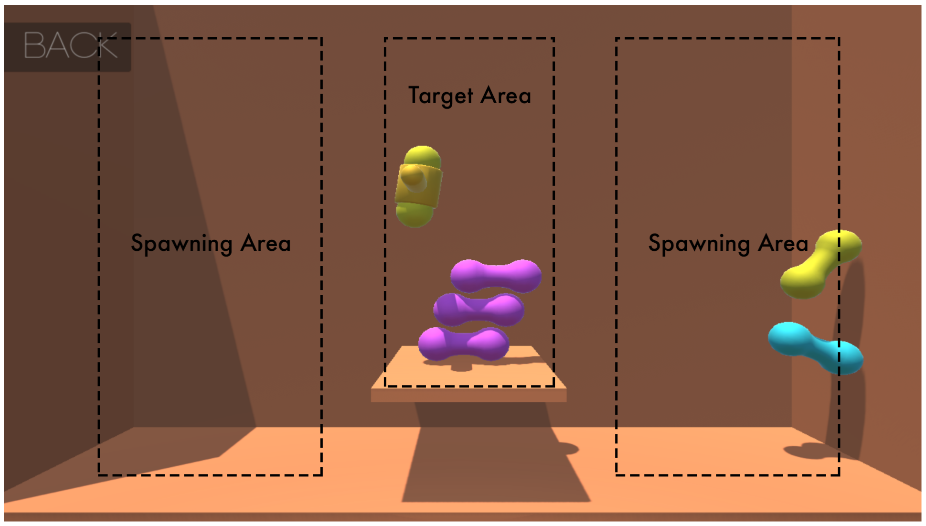


Figure 2: Level “Bean Stacking”

4.1.2 Default controllers

Currently, there are four controllers implemented: The Thalmic Pose Controller, the State Machine Controller, the ESN Controller, and the Keyboard Controller.

All but the Keyboard Controller are driven by the Myo. These controllers use the Myo's gyroscope to control the player's hand in x and y direction and use the EMG to control opening, closing and rotation. While the interpretation of the EMG differs, the way the prosthesis moves on the plane is the same for all three controllers:
The Euler angles $\vec{r}$ of the Myo's rotation are determined relative to a calibration vector using the Myo's gyroscope and its accelerometer. The vertical and horizontal angles r_v and r_h are trimmed to $[-v_{max}; v_{max}]$ and $[-h_{max}; h_{max}]$. Finally the angles are normalized to fit $[-1; 1]$. In one expression the coordinates calculated like this:

$$x := \frac{max\{min\{r_v, v_{max}\}, -v_{max}\} + v_{max}}{2v_{max}}$$
$$y := \frac{max\{min\{r_h, h_{max}\}, -h_{max}\} + h_{max}}{2h_{max}}$$

v_{max} and h_{max} can be configured using the Unity® editor and are set to $v_{max} = 25°$ and $h_{max} = 30°$ per default . Since the gyroscope is not completely accurate, after some movements the rotation calculated by the controller may not match the Myo's actual rotation anymore[8]. To correct this error the controller can be recalibrated to the centre of the screen by pressing the space bar.

[8]Actually this divergence does only appear at the vertical rotation component due to the

Thalmic Pose Controller The Thalmic Pose controller simply uses the poses the Myo's default software can detect [7] to rotate, open, and close the player's hand. In exact the poses `Fist` and `FingerSpread` are mapped to closing and opening, while `WaveOut` and `WaveIn` are mapped to clockwise and counter-clockwise rotation.

State Machine Controller The state machine controller is designed to be as similar as possible to the control modules of the prostheses, the participants of the experiment described in section 5.2 are using. The reason for this is that their in-game performance using this game controller shall be compared to their performance in the Box and Blocks test using their physical prostheses.

Their controllers work as follows:
The controller has two EMG-electrodes for two muscles with independent electric activity. The EMG-signal of both muscles is sampled at 100 Hz and afterwards filtered using an root mean square filter with window size of 150 ms.

The controller has one state per degree of freedom of the prosthesis. If the filtered EMG-signal of one electrode becomes greater than a patient-specific threshold, the prosthesis will move the in one direction of the state-related degree of freedom. If the other electrode is triggered the prosthesis moves in the opposite direction. If both electrodes are triggered at once, the controller advances to the next state. Basically the game's State Machine Controller works in the same way. Which two electrodes of the Myo's eight electrodes are to be used for the two triggers can be configured via a JSON-file[9]. It must have the following structure:

```
{
   "electrode1" : 4,
   "electrode2" : 0,
   "threshold1" : 0.2,
   "threshold2" : 0.2,
   "electrode1_open" : true,
   "electrode1_rotate_clockwise" : true
}
```

The fields `"electrode1"` and `"electrode2"` hold the indices $i \in [0;8[$ of both electrodes used for thresholding. The threshold of each electrode is defined by the fields `"threshold1"` and `"threshold2"` and may be within $[0;1]$. If the controller is in the opening/closing state and `"electrode1_open"` is true the hand will open when electrode 1 and close when electro 2 is triggered. If `"electrode1_open"` is false the meaning of electrode 1 and 2 is swapped in that state. The field `"electrode1_rotate_clockwise"` has analogous semantics for the rotating-state.
To find out which of the electrodes are enabled for which hand movement and to determine the optimal threshold, there is an additional scene in the Unity®

fact that the horizontal rotation can always be corrected by measuring the direction of the acceleration of gravity.

[9]Details on where the JSON file must be located see 4.4.1.

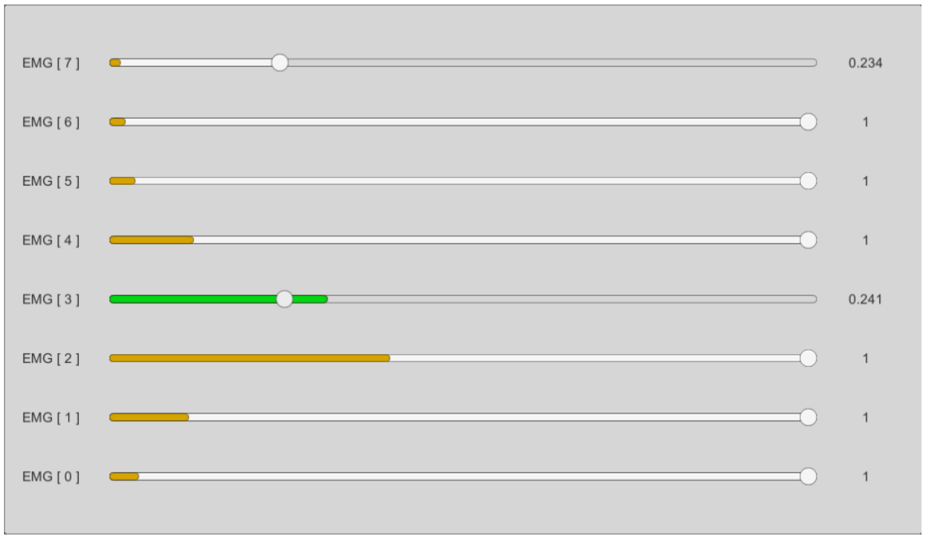


Figure 3: Scene `FindThresholdUi`

project. The scene is located in `<repo>/BoxAndBeans/Assets/Scenes` and is called `FindThresholdUi`. This scene visualizes the filtered EMG-signals of the Myo's electrodes. Additionally one can set a threshold per electrode. Whenever an electrode signal reaches the threshold it turns green instead of yellow. To use the configuration the thresholds have to be entered to a JSON file manually and the file's path has to be entered in the settings menu.

ESN Controller The last EMG-driven controller is the ESN-Controller. For EMG-processing the controller uses an Echo State Network (ESN). ESNs are Recurrent Neural Networks with a non-adaptable reservoir network for representing input dynamics and a memoryless output-layer. For training an ESN only the output weights are to be adapted, which reduces the computational complexity of the training procedure tremendously [24, 9, 21]. This reduction in computational complexity yields less time for training those networks. This implies an improvement concerning patient compliance and therefore makes Echo State Networks promising for practical use [21].
The ESN-controller loads the reservoir and the output weights from two individual JSON-files, from a user defined folder[10]. The files need to be named `current_output_weights.json` and `current_reservoir.json`. Both JSON files are obtained through a MATLAB[17]-driven training procedure. All steps of the procedure are started via the script `main` which can be found in the directory `<repo>/esn\_training`. It is assumed that this directory is the user's current working directory in MATLAB. Pittily the recording procedure depends on a platform dependent software module. Therefore recording can only be per-

[10]See 4.2.2 for how to set this folder.

formed on a Windows machine[11].

Recording In the first step the user has to call `main record <name> <left_or_right>` to record the EMG-data of several hand-movements: Each possible combination of pronation/supination/no movement and opening/closing/no movement of the hand is recorded for about 5 seconds. The recorded data is stored in a file. This step has to be repeated several times, to get enough data to get an ESN that is usable in practice. Ten recordings turned out to yield results with an average classification error of about 5 % [21].

Training After performing the recording steps several times, `main train <name> <left_or_right>` has to be called. This script loads the EMG-recordings and uses them to train an Echo State Network. The average classification error is printed to the screen so the user can check whether the trained ESN is usable or if more data is required to train the network. Furthermore the script stores the trained ESN's reservoir and output weights as JSON files and prints the directory where the files were written to.

The ESN Controller has three main advantages in comparison to the other EMG-driven controllers:

1. It maps real world movements to the same in-game hand movements which enables the user to control the game more naturally compared to the other controllers.
2. Two degrees of freedom can be controlled simultaneously which yields even more natural control.
3. The ESN Controller does not only classify the EMG-signals, but outputs the probability of the detected movement. Therefore the user can not only decide which degree of freedom to move in which direction, but also control the strength of this movement. This control strategy is also called proportional control [1].

Keyboard Controller The last controller is the most simple one. It does not need any configuration and can be controlled via the keyboard. Its main purpose is to be used for testing user created levels without being forced to configure an EMG-controller. The Keyboard Controller maps `w` to opening, `s` to closing, `d` to clockwise, and `a` to counter-clockwise rotation of the hand. It can be moved in x and y direction using the arrow keys.

4.2 Menus

4.2.1 Main menu

When the game is launched the screen shows the Main Menu. It has four buttons and one text input field. The player has to insert a name and press the "start"

[11]Recording and training is not only possible on a physical but also on a virtual Windows machine

button to launch the game in the normal mode (described in 4.1). Since the results of each game run need to be associated to a player inserting a name is obligatory to start the game.

The "try" button launches the game in a Try-Out mode also described in 4.1, and the "quit" button terminates the game. Additionally there is a button labeled "settings" which navigates to the Settings menu.

4.2.2 Settings menu

The settings menu lets the player configure different game properties:

Score saves This property can be used to set the directory where the data of each game run is to be stored. For details on the stored data see section 4.3. The default value is platform dependent.

Level This drop-down list lets the user choose the level to play.

Controller This drop-down list is used to choose the controller to be used.

Ctrl-config The string entered here will be passed to the chosen controller to let it load its settings. In most cases the string will be the path to a configuration file. The text field may be non-editable in case the controller does not need to be configured. For details on the controller configuration see 4.4.1.

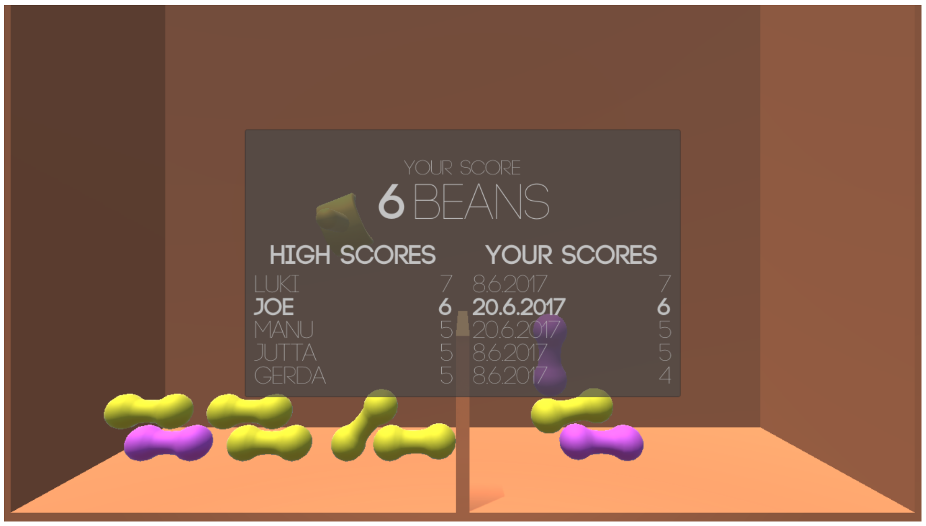
The user can save the settings or discard them using the buttons on the bottom of the menu. Before saving the controller is supplied with the "Ctrl-config" setting to load its configuration. If this loading fails an error message is provided to the user and the settings are not saved. When the settings are saved successfully or discarded the UI switches back to the main menu.

4.2.3 Game & Try-Out UI

During a normal game run there is a timer and a score counter in the top left corner of the screen. In Try-Out mode there is a button labelled "back" instead of the counter and the timer which makes the game switch back to the main menu.

4.2.4 Highscore Lists

When the user finishes a game run the UI shows the Highscore View. This view shows the number of beans the player scored in the last run and additionally two different leader boards. Both lists present different sets of scores. The left one, called the highscore list, processes the players last result and the best results of all players for the current level. The right one, called the player score list, processes all of the players results for the current level. The player score list labels the scores with the date of the score's game run while the highscore list labels the scores with the player names. In both lists the result of the last run is highlighted using a bold font.


YOUR SCORE

6 BEANS

HIGH SCORES		YOUR SCORES	
LUKI	7	8.6.2017	7
JOE	6	20.6.2017	6
MANU	5	20.6.2017	5
JUTTA	5	8.6.2017	5
GERDA	5	8.6.2017	4

Figure 4: Highscore View

As found in [10] listing people on top of leader boards increases players' enjoyment as well as their motivation. Therefore both leader boards are designed to show the current score always as the second best on the list. In exact the current result will be on the first position of the leader board if and only if the player scored the best result of the list. If he was not the best he will be ranked as second and the next better score will be ranked first. All other ranks will be filled with the successors' scores. The reason that the participant is not ranked on top of the score list, is that he should always see a score to compete against, to motivate the user. [22]

4.3 Persistence

The game stores two kinds of data: User-defined settings and the results of game runs.

4.3.1 Settings

As any other persistent data in the game, the settings are stored as JSON files. The location of the file is platform-dependent and corresponds to Unity® 's function `Application.persistentDataPath`[12]. This file should never be altered manually. The file's structure looks like:

```
{
   "currentLevel"      : "Bean Moving",
   "currentController" : "Keyboard Controller",
   "persistenceDir"    : "/directory/to/store/scores/in",
   "controllerConfs" : [
      {
         "controllerName" : "Keyboard Controller",
         "configuration" : ""
      },
      ...
   ]
}
```

There is one entry for each field in the settings menu, plus an additional array of fields, in which each entry defines the configuration string for a controller.

4.3.2 Results

The second, more important, kind of persistent data is the data that stores the results of each game run. It is stored in a configurable directory corresponding to the field "score saves" in the settings menu. The result of each run is stored as an individual file. The file name is a combination of the player name and the time the score was saved. In exact the format is `name_YYYY-MM-DD_hh-mm-ss.json`. Each files structure looks like this:

```
{
   "participant" : "joe",
   "level"       : "Bean Moving",
   "score"       :  4,
   "beans"       : [
      {
         "states" : [
            {
               "time"          :   0.06,
               "x"             :  20.90,
               "y"             : -10.81,
               "rotation"      : 110.69,
```

[12]https://docs.unity3d.com/ScriptReference/Application-persistentDataPath.html

```
                "wasCollected" :  false,
                "isGrabbed"    :  false
             },
             ...
          ]
       },
       ...
    ],
    "playground" : {
       "xMin" : -30.0,
       "xMax" :  30.0,
       "yMin" : -19.5,
       "yMax" :  19.5
    },
    "worldTime"      : "2017-06-08T19:01:45.1160620+02:00",
    "controllerPath" : [
       {
          "x"                : 1.17,
          "y"                : 1.89,
          "rotation"         : 57.60,
          "opened"           : 0.92,
          "openingVelocity"  : 0.0,
          "rotationVelocity" : 0.0,
          "time"             : 0.06
       },
       ...
    ]
}
```

The fields contain the following information:

participant The player's name.

level The name of the level of the stored run.

score The score the player achieved.

beans An array of JSON objects where each object represents one bean. A bean is described by an array of states. Each state has the following fields:

- **time** The time[13] when the state was captured.
- **x** The bean's horizontal position.
- **y** The bean's vertical position.
- **rotation** The bean's rotation in degrees. The rotation is measured counter clockwise relative to the horizontal axis.
- **isGrabbed** A boolean describing if the bean is currently grabbed by the hand or not.

[13]All timings are measured in seconds from the start of the stored run.

wasCollected A boolean describing whether the bean was dropped in the target area or not.

playground Describing the bounds of the area the game takes place on.

worldTime The time when the result of the file was stored. The time is represented as date time string with time zone, compatible to the ISO 8601 standard. The exact format is `YYYY-MM-DDThh:mm:ss.nnnnnnn(+|-)hh:mm`. For example 6th of June 2017 at 11:07:26 in Vienna would be stored as `2017-06-08T11:07:26.1212300+02:00`.

controllerPath This field holds an array of JSON-objects representing the state of the controller on different points of time. Each state has the following fields:

x The hand's horizontal position.

y The hand's vertical position.

time The time[13] the state of the controller was captured.

rotation The hand's counter clockwise rotation in degrees relative to the horizontal axis.

opened A number within $[0; 1]$ that describes how strong the hand is opened. 0 means that the hand is closed and 1 means the hand is totally opened.

openingVelocity A number within $[-1; 1]$ that describes how strong the prosthesis is currently opening or closing where -1 means closing at full speed +1 means opening at full speed.

rotationVelocity A number within $[-1; 1]$ that describes how fast the hand is currently rotating where -1 means full velocity counter clockwise and 1 means full velocity clockwise.

All states are sampled with a frequency depending primarily Unity® 's update rate, which depends on the machine's CPU load. To limit the amount of data the sampling time is set to 50 ms. This number can be configured using the Unity® editor, by setting the property `MinDocumentationFrameLength` of the `GameTime` game object.

4.4 Extending the game

The game can be extended in two ways: By implementing a controller or by creating a level.

4.4.1 Adding a controller

For creating a new controller the user has to implement the abstract C# class `ProsthesisController`[14]. The interface looks as follows:

```
public abstract class ProsthesisController : MonoBehaviour {
   /// a value within [-1; 1] where
   ///       1 means the prostheses should open at full velocity,
   ///      -1 means the prostheses should close at full velocity and
   ///       0 means the prostheses should neither open nor close
   public abstract float OpeningVelocity();

   /// a value within [-1; 1] where
   ///      -1 means turn counter clock wise at full velocity,
   ///       1 means turn clock wise at full velocity and
   ///       0 means stay still
   public abstract float RotationVelocity();

   /// the position of the prosthesis within 2 dimensional game area.
   ///   both coordinates must be within [ -1; 1 ]
   public abstract Vector2 Position();


   /// a unique user-readable identifier for the controller class
   public abstract string controllerName { get; }
   /// path to the default the configuration file
   public abstract string defaultConfigFile { get; }

   /// returns whether the controller needs to be configured or not
   public abstract bool needsConfiguration { get; }

   /// lets the controller load the given configuration file
   public abstract ConfigResult SetConfiguration (string configPath);
}
```

To determine how to move the in-game hand Unity® will call the methods `RotationVelocity`, `OpeningVelocity` and `Position` of the active controller in each frame. The other methods are used for configuring the controller. `controllerName` should return the controller class' unique name. It will be shown in the settings menu. Whenever the user presses “save” in the settings menu, the game will check whether the controller currently selected in the “controller” drop-down list needs to be configured by calling `needsConfiguration`. In case it needs to be configured the controller will call `SetConfiguration` with

[14]The class is defined in `<repo>/BoxAndBeans/Assets/Scripts/ProsthesisControllers/ProsthesisController.cs`.

the string currently entered in the "ctrl-config" textfield as argument. The controller should attempt to load the given configuration file and return whether it was successfull or not by returning a `ConfigResult`. `ConfigResult.error(String)` can be called to construct an error instance or `ConfigResult.OK` can be returned to signal success. On success the controller which was active before will be disabled and the new one will be enabled[15]. On failure the user will be provided with the error message of the `ConfigResult`. If the controller needs to perform any special tasks (like stopping threads or deallocating resources) when being disabled, one can add a function with the signature `void OnDisable()` which will be called by Unity® when disabling the script.

To use the contoller script in the game it has to be attached to any Unity® game object in the `GameView` scene using the Unity® editor. For the purpose of keeping the project clean I recommend attaching the controller to a new child object of the `Controllers` game object. To do this without any knowledge of the Unity® editor one has to execute the follwing steps[16]:

1. Place your controller implementation in `<repo>/BoxAndBeans/Assets/Scripts/ProsthesisControllers/`.
2. Open the project.
3. Unfold the item "GameView" in the hierachy-panel on the left hand side.
4. Right-click "Controllers" and select "Create empty".
5. The new object called "GameObject" should be selected automatically.
6. Rename the object to something useful in the inspector panel on the right hand side.
7. Click "Add Component" on the inspector panel and search for your script.
8. Save the game.

4.4.2 Adding a level

To add a level some basic knowledge about the Unity® editor is needed[17]. To create a level the `Level` prefab can be used as a starting point. It is located in `<repo>/BoxAndBeans/Assets/Prefabs`. Again to keep the project clean, the prefab should be added as a child object of the game object `Levels` when instancing it. I will refer to the instantiated prefab as "Level Game Object" in the remainder of this subsection. It has several child objects that define the level's behaviour:

[15]Enabled in this context means on the one hand that they will be used to control the game and on the other hand that Unity® will call `void Update()` on the object once per frame if such a method exists.

[16]Assuming running Unity® 5.6.1f1 Personal with the default layout.

[17]The beginner's tutorial on `https://unity3d.com/learn/tutorials` should be sufficient for this purpose.

`SpawningAreas` manages where the beans spawn. Whenever all beans on the screen are collected, the game spawns a fixed number of new beans. This number can be configured using the properties of the Level Game Object. For each bean `SpawningAreas` selects a random point within one of the `BoxCollider2D`s that are attached to its child objects. The `SpawningAreas` object can be configured by adding or removing `SpawningArea` prefabs and by translating and scaling them.

`TargetAreas` represents the areas where the beans need to be moved to. Any number of `TargetArea` prefabs may be attached or removed as child object. The `TargetArea` instances may be transformed arbitrarily. A bean counts as collected if it has been moved to any of the `TargetArea`s.

`Walls` Neither this game object nor its children have any special logic attached to them. It's children can be transformed arbitrarily and any game objects can be added as child object.

`InvisibleWalls` is a container for all instances of the `InvisibleWall` prefab. `InvisibleWall`s may also be transformed as needed. They may also be removed and added using a prefab.

Additionally the name of the instantiated prefab should be configured properly, since this name should be a unique user-readable identifier for the level, which will be presented in the settings section. Furthermore the duration of each run as well as the number of beans spawned at once can be configured using the `Level` script's properties which are attached to the Level Game Object.

4.4.3 Building the game

For building the game without any knowledge about Unity® the following steps need to be executed[16]:

1. Click "File" > "Build Settings..." on the top bar.
2. Select the wanted target platform and architecture.
3. Click "Build" to select the destination path to write the build to.

5 Evaluation

The evaluation of the program is split into two phases. There is a pre-evaluation phase, to detect probable problems with the gameplay conducting the actual evaluation phase. The first phase will have been conducted when this thesis is finished and will therefore be discussed in section 5.4. The second phase will be conducted later.

5.1 Pre-Evaluation

In the first phase 14 able-bodied (7 male, 7 female; age ranging from 13 to 72) took part in the experiment. The game was set up with the State Machine controller and the Bean Moving level.

In the first step each of the participants put on the Myo on their dominant arm and the game was started in the Try-Out mode without configuring the controller. They were requested to practice moving around the hand on the screen to get a feeling for the gyroscope-based feeling without being distracted by playing around with the EMG[18]. Additionally they were instructed to recalibrate the position of the prosthesis whenever they felt it was beneficial.

Next the controller was configured using the `FindThresholdUi` described in section 4.1.2. After configuring the thresholds the gameplay and how to open, close, and rotate the hand was explained to the participants. They were told to try opening and closing the prosthesis for 1 to 2 minutes. If the controller seemed to be too much or too less sensitive the procedure was restarted from configuring the thresholds.

In the third step the users were given about 15 minutes time to play the game in Try-Out mode to get familiar with the controller and the game itself. Afterwards they were asked to rate statements about the game with a number on a scale ranging from 1 to 5, where 1 means "I don't agree at all.", and 5 means "I totally agree.".

Q1. The game is boring.

Q2. I have control about what is happening in the game.

Q3. The game is gripping me. I want continue playing.

After answering those questions the participants had to play five game runs. Afterwards they were asked to rate the same statements again. Additionally they were asked to rate the following statements.

Q4. I did felt time pressure when playing.

Q5. The highscore list put pressure on me.

Furthermore, if there was pressure, they were asked whether they experienced it as motivating or not. In addition the participants were asked if they experienced any problems with the game.

[18]This part was especially necessary for older participants.

5.2 Actual Evaluation

The actual evaluation of the program is part of a research project on using Echo State Networks and Transfer Learning for EMG-driven prostheses controllers [20, 21]. Therefore the study targets to evaluate both, the program, as well as whether a Transfer Learning based retraining of an Echo State Network can make up for a displacement of surface EMG-electrodes or not.

The study will be conducted on six to eight amputees. All of them will have a Michael Angelo prosthesis controlled similarly to the State Machine Controller.

First of all the participants will conduct the Box and Blocks test three times with their physical prostheses. Afterwards the patients will be asked to rate the statements 1 to 4 from Section 5.1.

In the next step they will play the game multiple times using different controllers. The game will be played three times with each controller. As in the previous phase the participants will get some time for getting familiar with the controller before starting the game runs. As in the pre-evaluation after each try out phase the participants will have to rate the statements 1 to 3 and after the three actual runs they will have to rate all 5 statements.

The controller setups for the patients will be the following:

- First they will use the State Machine controller. Since this controller is very similar to the control module of their physical prostheses, the outcomes of this run will be compared to the results of the Box and Blocks test to evaluate the programs difficulty.
- Afterwards the participants will train an Echo State Network and play the game using the ESN Controller.
- Next their Thalimc Myo wristband will be rotated around their arm so that all electrodes are shifted by about 2 centimetres. The participants will again do three game runs with this setup.
- For the last three runs the Echo State Network will be retrained using a Transfer Learning algorithm.

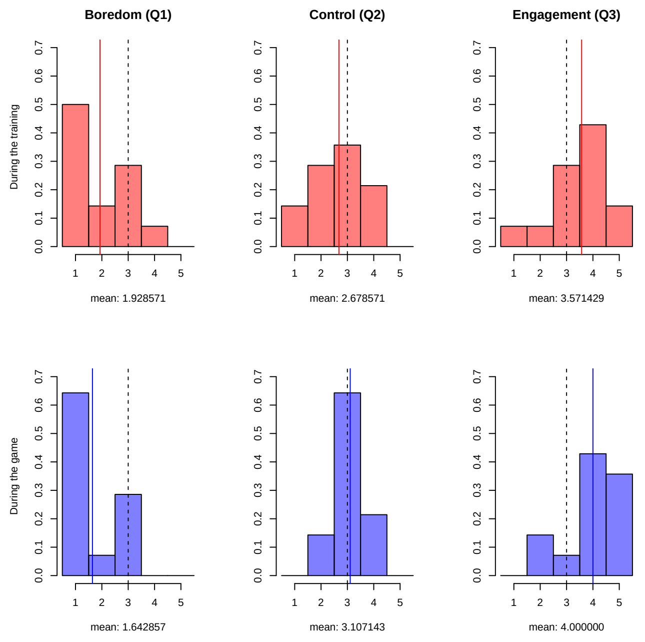


Figure 5: Statements 1 - 3

5.3 Results

All visualisation was done using R [23]. The script creating the plots, as well as all data obtained from the study can be found in `<repo>/evaluation`.

The histograms in figure 5 show the participants' ratings of the statements 1 to 3 after the training and after the game. In each histogram the mean is represented as a vertical line.

The upper half of figure 6 shows how strong the patients rated the pressure they were exposed to during the game. The lower half combines the pressure with the question whether they experienced the pressure as motivating or as stressful. Therefore the values where subtracted by 1 to be within $[0; 4]$ and the values of the people who experienced the pressure as stressful was made negative. In both cases the means were plotted as a horizontal line. To find possible connections between patients answers, there performance and their age the correlation matrix of the patients' ratings of the statements, the patients' ages, the patients' feelings of motivation gained by the time and by the highscore list, and the increment in control, boredom and grip between the training and the actual game was calculated. Figure 7 visualizes this correlation matrix. To clarify the figure all correlation within $[-0.5, 0.5]$ were set to zero.

The problems the patients reported to have been experienced with the game can be summarized in 4 points:

- Most probands reported that the controller changed state when they did

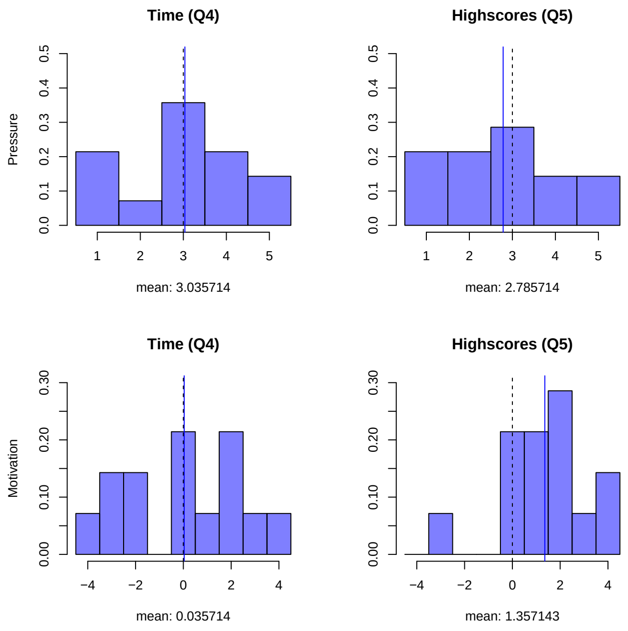


Figure 6: Statements 4 & 5

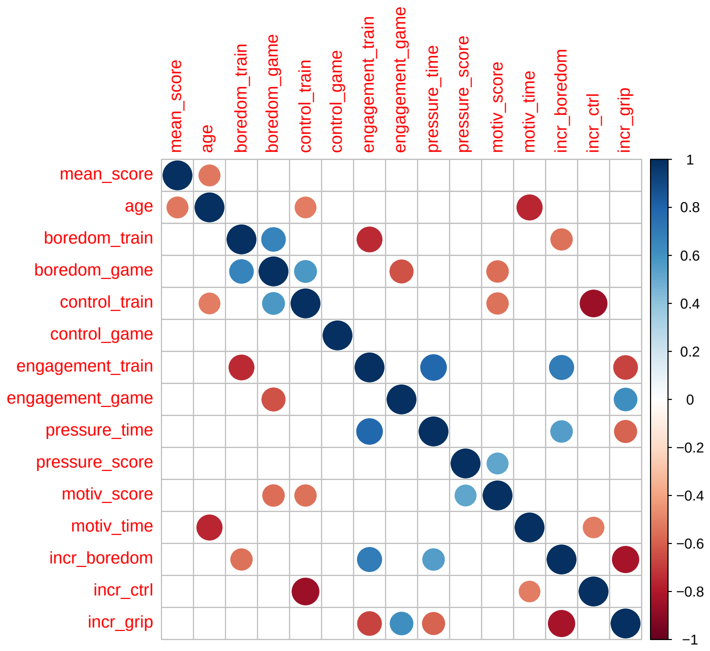


Figure 7: preprocessed correlation matrix

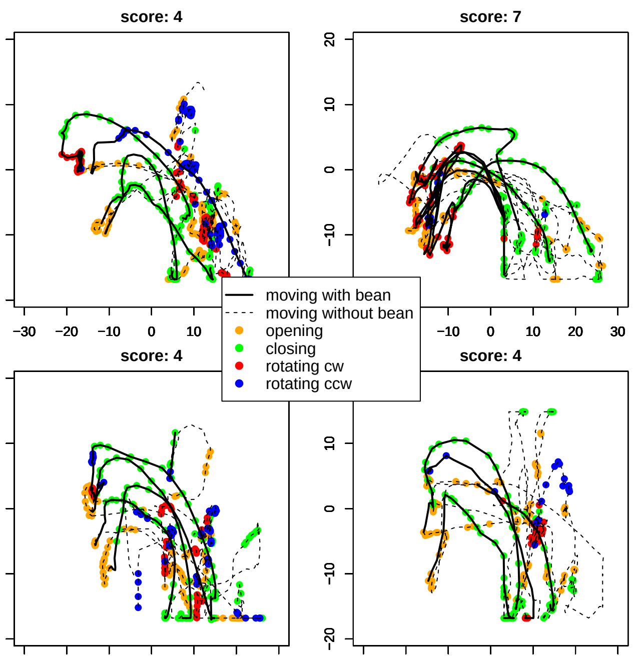


Figure 8: Visualization of game runs

not intend to do so. Especially more often during the game, than during the training.

- Many participants claimed to have had difficulties with the controller unnatural type of control. Very frequently probands tried to grab beans by closing their hand, which lead to a state change instead of the intended in-game hand closing. Nevertheless the all of them claimed that this problem may only be a matter of practise.
- In some cases the position of the in-game hand shifted to the right continuously.
- Some participants pointed out that it is difficult to grab the beans when they are located in a corner and rotated so that the corners' angle's bisecting is perpendicular to the beans longer axis.

Figure 8 demonstrates how the data tracked during a game run can be visualized. Therefore the path of the in-game hand was drawn using a dotted line, when it was moving without a bean, and using a bold continuous line when carrying a bean. Whenever the prosthesis did rotated, opened or closed, a coloured point, representing the movement, was drawn. All four plots use the data of four different runs of one participant.

5.4 Discussion

As one could expect boredom and engagement of the game as well as in the training are negatively correlated, since they measure kind of an opposite emotion. On average people experienced more boredom during the training than during the game. The opposite applies for being engaged by the game. Getting motivated by the highscore list might be a reason for this gain in engagement, since most participants claimed that the leaderboard did motivate them.

Many participants felt pressure being conducted to them by the limited time. Due to the fact that there was about the same number of people experiencing the pressure as motivating as the number of people experiencing it as negative, the time pressure can be neither interpreted as motivating nor as stressful.

The average feeling of control over the game was a bit higher after playing than after the training, which might be caused by the fact that people where more used to the program after the game runs. It was suspected that people with a higher score in the game would rate their feeling of control higher than the ones with worse scores. Surprisingly this was not the case since the correlation between the feeling of control control during the training and the game and the participants' scores was even only 0.01146677 and -0.14203033. The increment in the feeling of control was even correlated negatively with the score.

Many participants claimed that the prosthesis did switch state very often during the game, especially in the last runs. One reason for that may be that the patients need to learn to properly contract only one muscle group at once. Furthermore the loss of concentration during a session may also be a reason why participants triggered both electrodes unintentionally. Additionally sweat and warmth of the probands arm might be the reason for an increased conductivity and therefore an easier triggering of the electrodes. [22]

The correlation matrix shows that elder participants tend to be more stressed by the limited time, and that they scored less beans than younger. The reason for that might be that younger people are used to computer game, while elder people might experience the virtual environment more as a test than as a game.

The drift of the in-game hand, probands experienced, is caused by the inaccuracy of the Myo's gyroscope, which was already experienced by other users of the device[19]. The problem could be resolved by either using another device for measuring the absolute rotation or by using a different means of control for the hand's position. Therefore technologies like motion capture sensors could be applicable.

The problem with beans that are located in the corner of the game area could be resolved by adding an invisible sphere around each bean which does collide with the walls but not with the player's hand. This makes it impossible for the beans to reach such an unfortunate position.

Probands claimed that unwanted state changes were one of their main problems with the controller. This problem can also be seen when looking at figure 8 closely: Before opening the hand on the left side, to release a bean, in many

[19]Discussed in this forum post `https://developer.thalmic.com/forums/topic/813/?page=2` (accessed: 14th of June 2017) for example.

cases there was a clockwise rotation performed before. Since clockwise rotation and hand opening are triggered by the same electrode but in different states, one can conclude that this rotation was performed because the controller was not in the expected state.

Unfortunately it was not possible to evaluate whether the game is about as difficult as the Box and Blocks Test. To be able to do so, participants would need to perform the virtual as well as the physical test with a similar controller. Nevertheless this part of evaluation will be covered in another study as described in section 5.2. Furthermore the second study will allow to analyse the participants feelings about the game in comparison to the Box and Blocks Test.

6 Conclusion

Pattern recognition offers great opportunities for developing EMG-controlled prostheses. Nevertheless technologies using sophisticated algorithms need to well-evaluated. State of the art is equipping a physical prosthesis with a new control module and conducting an occupational therapeutic test. Since therefore a physical device needs to be adjusted, and since physical tests produce only little amount of data to be analysed by developers a serious computer game was developed for this thesis.

The game was implemented to test controllers with two degrees of freedom. It provides two levels testing different degrees of manual dexterity and three different EMG-driven controllers. Furthermore it provides a simple C# programming interface to let developers integrate their own controllers. Additionally developers can create custom levels without being forced to write a single line of code. The game tracks enough data in each game run so that developers can reconstruct the whole run afterwards. Since occupational therapeutic test are repetitive task, the program includes a leader board to enhance users' motivation, which was confirmed by a study with 14 participants.

Basically the game tests how many objects the player can move within a fixed amount of time. A future improvement of the program could involve a mode testing how long the player needs to move a fixed amount of objects. The gameplay basically takes place in two dimensions. Therefore a third dimension could be added so that the program offers testing controllers with three degrees of freedom. Furthermore the programming interface for adding controllers could be extended so that the game provides haptic feedback when the virtual prosthesis collides with game objects.

References

[1] Dario Farina, Ning Jiang, Hubertus Rehbaum, Ales Holobar, Bernhard Graimann, Hans Dietl, and Oskar C. Aszmann. The extraction of neural information from the surface emg for the control of upper-limb prostheses: Emerging avenues and challenges. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 22(4):797–809, 7 2014.

[2] T. R. Farrell and R. F. Weir. The optimal controller delay for myoelectric prostheses. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 15(1):111–118, March 2007.

[3] Blender Foundation. Blender (2.78), 2017. `https://www.blender.org/`.

[4] .NET Foundation. Mono (2.6.5), 2017. `http://www.mono-project.com`.

[5] L. J. Hargrove, B. A. Lock, and A. M. Simon. Pattern recognition control outperforms conventional myoelectric control in upper limb patients with targeted muscle reinnervation. In *2013 35th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pages 1599–1602, July 2013.

[6] Thalmic Labs Inc. Myo, 2017. `https://www.myo.com/`.

[7] Thalmic Labs Inc. Myo sdk (0.9.0), 2017. `https://developer.thalmic.com/docs/api_reference/platform/index.html`.

[8] FRCPC Jacqueline S. Hebert, MD, BSc Justin Lewicke, MBA, PEng Thomas R. Williams, PhD, and PhD Albert H. Vette. Normative data for modified box and blocks test measuring upper-limb function via motion capture. *Journal of Rehabilitation Research & Development (JRRD)*, page 919 — 932, 2014.

[9] Herbert Jaeger. Adaptive nonlinear system identification with echo state networks. *networks*, 8(9):17, 2003.

[10] Yuan Jia, Yikun Liu, Xing Yu, and Stephen Voida. Designing leaderboards for gamification: Perceived differences based on user ranking, application domain, and personality traits. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI '17, pages 1949–1960, New York, NY, USA, 2017. ACM.

[11] The JSON Data Interchange Format. Technical Report Standard ECMA-404 1st Edition / October 2013, ECMA, October 2013.

[12] Rami N. Khushaba, Sarath Kodagoda, Maen Takruri, and Gamini Dissanayake. Toward improved control of prosthetic fingers using surface electromyogram (emg) signals. *Expert Syst. Appl.*, 39(12):10731–10738, September 2012.

[13] Todd A. Kuiken, Guanglin Li, Blair A. Lock, Robert D. Lipschutz, Laura A. Miller, Kathy A. Stubblefield, and Kevin Englehart. Targeted muscle reinnervation for real-time myoelectric control of multifunction artificial arms. *JAMA : the journal of the American Medical Association*, 301(6):619–628", Februar 2009.

[14] Peter J. Kyberd, Alessio Murgia, Mark Gasson, Tristan Tjerks, Cheryl Metcalf, Paul H. Chappell, Kevin Warwick, Sian E. M. Lawson, and Tom Barnhill. Case studies to demonstrate the range of applications of the southampton hand assessment procedure. *British Journal of Occupational Therapy*, 72(5):212–218, 5 2009.

[15] Joris M. Lambrecht, Christopher L. Pulliam, and Robert F. Kirsch. Virtual reality environment for simulating tasks with a myoelectric prosthesis: an assessment and training tool. *Journal of prosthetics and orthotics*, 2011.

[16] Virgil Mathiowetz, Gloria Volland, Nancy Kashman, and Karen Weber. Adult norms for the box and block test of manual dexterity. *American Journal of Occupational Therapy*, 39(6):386–391, 1985.

[17] The Mathworks, Inc., Natick, Massachusetts. *MATLAB version 8.1.0.604 (R2013a)*, 2013.

[18] Microsoft. C# (6.0), 2017. `https://docs.microsoft.com/en-us/dotnet/csharp/csharp`.

[19] Laura A. Miller, Robert D. Lipschutz, Kathy A. Stubblefield, Blair A. Lock, He Huang, T. Walley Williams III, Richard F. Weir, and Todd A. Kuiken. Control of a six degree-of-freedom prosthetic arm after targeted muscle reinnervation surgery. pages 2057—-2065, 11 2008.

[20] Cosima Prahm, Benjamin Paassen, Alexander Schulz, Barbara Hammer, and Oskar Aszmann. Transfer learning for rapid re-calibration of a myoelectric prosthesis after electrode shift. In *Converging Clinical and Engineering Research on Neurorehabilitation II*, volume 15, pages pp 153–157. Springer International Publishing, 2017.

[21] Cosima Prahm, Alexander Schulz, Benjamin Paaßen, Oskar Aszmann, Barbara Hammer, and Georg Dorffner. Echo State Networks as Novel Approach for Low-Cost Myoelectric Control. In Annette ten Telje, John H. Holmes, Lucia Sacchi, and Christian Popow, editors, *Proceedings of the 16th Conference on Artificial Intelligence in Medicine (AIME 2017)*, volume 10259. Springer, in press.

[22] Cosima Prahm, Ivan Vujaklija, Fares Kayali, Peter Purgathofer, and C. Oskar Aszmann. Game-based rehabilitation for myoelectric prosthesis control. *JMIR Serious Games*, 5(1):e3, Feb 2017.

[23] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2008. ISBN 3-900051-07-0.

[24] A. Rodan and P. Tino. Minimum complexity echo state network. *IEEE Transactions on Neural Networks*, 22(1):131–144, Jan 2011.

[25] Unity Technologies. Unity® (5.6.1f1), 2017. `https://unity3d.com/`.