

Gamified 3D Orthopaedic Rehabilitation Using Low Cost and Portable Inertial Sensors

Grigorios Kontadakis

A thesis submitted in fulfillment of the requirements for the degree of Master of Science in Electrical and Computer Engineering.

School of Electrical and Computer Engineering
Laboratory of Distributed Multimedia Information Systems and Applications – *TUC/MUSIC*

CHANIA 2018

Abstract

This work introduces an innovative gamified rehabilitation platform comprising of a mobile game and a custom sensor placed on the knee, intended for patients that have undergone Total Knee Replacement surgery, in collaboration with the General Hospital in Chania. The application uses a single custom-made, light, portable and low-cost sensor node consisting of an Inertial Measurement Unit (IMU) attached on a lower limb in order to capture its orientation in space in real-time, while the patient is completing a physiotherapy protocol.

An IMU measures and reports a body's specific force (accelerometer), angular rate (gyroscope), and sometimes the magnetic field surrounding the body (magnetometer). IMUs provide the leading technology used in wearable devices and are employed in this thesis as they meet the main design constraints of this framework, maximizing portability and minimizing cost. Previous work indicates that when multiple IMUs are employed, satisfactory rehabilitation exercise classification accuracy results are achieved based on three, two and one IMUs. Such results drive the further investigation of the challenging classification problem using just a single IMU in this thesis, enhancing maximum portability compared to multiple IMU systems, in conjunction with sufficiently high success rates of movement detection.

Our goal is to reduce the need for the physical presence of a physiotherapist by aiding the efficient performance of exercise sessions at any location and increase patient engagement during physiotherapy by motivating the user to participate in a game using the current ultra-portable framework of just a single IMU sensor and an Android device. The proposed sensor node attached on the lower limb provides input to the gamified experience displayed on an Android mobile device, offering feedback to the patient in relation to whether the performed exercises were accurately conducted. A classification algorithm is proposed that automatically classifies an exercise in real-time as correct or incorrect, according to physiotherapists' set criteria. Initial testing of the system is conducted in the Chania's General Hospital Orthopaedic Clinic, Greece, in collaboration with Orthopaedic Surgeons and Physiotherapists. This testing indicates that patient engagement is enhanced in most cases.

Declaration

The work in this thesis is original and no portion of the work referred to here has been submitted in support of an application for another degree or qualification of this or any other university or institution of learning.

Signed:

Date:

Kontadakis Grigorios

Acknowledgements

This study was performed at the Technical University of Crete in collaboration with the Orthopaedic Clinic Department of Chania General Hospital. The current project wouldn't be possible without the formal authorization of the hospital to work with physiotherapists, orthopaedics and Total Knee Replacement (TKR) patients and the voluntary consent of TKR patients themselves to contribute to this study.

Publications

- Kontadakis, G., Chasiouras, D., Proimaki, D., Halkiadakis, M., Fyntikaki, M. and Mania, K. (accepted 2018). Gamified Platform for Rehabilitation after Total Knee Replacement Surgery Employing Low Cost and Portable Inertial Measurement Sensor Node. *Multimedia Tools and Applications*, Springer.
- Kontadakis, G., Chasiouras, D., Proimaki, D. and Mania, K. (2017). Gamified 3D Orthopaedic Rehabilitation Using Low Cost and Portable Inertial Sensors. In Proc. *IEEE Virtual Worlds and Games for Serious Applications 2017 (VS-Games)*, pp. 165-168. **SELECTED AS ONE OF BEST PAPERS.**
- Kontadakis, G., Chasiouras, D., Proimaki, D. and Mania, K. (2006). Gamified 3D Orthopaedic Rehabilitation Using Low Cost and Portable Inertial Sensors. Poster, *CGI 2016*, in co-operation with ACM and Eurographics.
- Kontadakis, G., Chasiouras, D., Proimaki, D. (2015). Design of Augmented Reality 3D Rehabilitation Application using an Inertial Measurement Unit. In Proc. of *SFHMMY 8*.

Table of Contents

| | |
|---|-----------|
| Abstract | 2 |
| Declaration | 3 |
| Acknowledgements | 4 |
| Publications | 5 |
| Table of Contents | 6 |
| List of Figures | 9 |
| List of Tables | 12 |
| 1 Chapter 1 – Introduction..... | 13 |
| 1.1 Contribution..... | 14 |
| 1.2 Thesis Outline | 15 |
| 2 Chapter 2 – Existing Work & Background | 17 |
| 2.1 Medical Background..... | 17 |
| 2.1.1 Knee Anatomy & Reduced Functionality Causes | 17 |
| 2.1.2 TKR..... | 19 |
| 2.1.3 Rehabilitation Exercises for TKR | 20 |
| 2.2 Existing Technologies for Limb Motion Tracking | 21 |
| 2.2.1 Optical Systems | 21 |
| 2.2.2 Exo-skeletons..... | 21 |
| 2.2.3 Electrogoniometers | 22 |
| 2.2.4 Magnetic Systems..... | 22 |
| 2.2.5 Inertial Measurement Units | 23 |
| 2.3 Related Works & Applications..... | 24 |
| 2.3.1 Rehabilitation Projects | 24 |
| 2.3.2 IMU Example Applications..... | 26 |
| 2.4 IMU Functionality | 29 |
| 2.4.1 MPU-9150 Overview..... | 29 |
| 2.4.2 MPU-9150 Product Specification..... | 30 |
| 2.4.3 MPU-9150 Register Map | 31 |
| 2.5 IMU Filtering Methods | 33 |
| 2.5.1 Complementary Filter..... | 33 |
| 2.5.2 Kalman Filter..... | 34 |
| 2.5.3 Madgwick Filter | 34 |

Table of Contents

| | | |
|------------|--|-----------|
| 2.5.4 | Derivation of Kalman Filter Parameters | 35 |
| 2.6 | Embedding the IMU | 38 |
| 2.6.1 | Single-board Computer vs Microcontroller | 38 |
| 2.6.2 | Raspberry Pi Overview | 39 |
| 2.6.3 | Configuring Raspberry Pi as a Microcontroller | 40 |
| 3 | Chapter 3 – Overview of IDE & Development Tools | 43 |
| 3.1 | Game Development | 43 |
| 3.2 | Game Engine Systems | 45 |
| 3.2.1 | Unreal Engine Overview | 45 |
| 3.2.2 | Unity 3D 5.6 Overview | 46 |
| 3.2.3 | Comments | 47 |
| 3.3 | Unity Gameplay Creation | 47 |
| 3.3.1 | Scenes | 47 |
| 3.3.2 | GameObject | 48 |
| 3.3.3 | Components | 48 |
| 3.3.4 | Assets | 49 |
| 3.3.5 | Script Life Cycle | 49 |
| 3.3.6 | Unity Singleton | 52 |
| 3.3.7 | Unity Profiler | 53 |
| 3.4 | Combining Unity & Android | 53 |
| 3.4.1 | Unity & Android Limitations | 53 |
| 3.4.2 | Android Studio | 54 |
| 3.4.3 | Creating Sample Plugin | 55 |
| 4 | Chapter 4 – Design | 57 |
| 4.1 | Application Functionality | 57 |
| 4.1.1 | Application Procedure | 57 |
| 4.1.2 | Gamification Feedback | 58 |
| 4.1.3 | Binary Classification | 59 |
| 4.2 | Mobile Application UI Design | 60 |
| 4.2.1 | Main Menu Design | 60 |
| 4.2.2 | Gameplay HUD | 64 |
| 4.2.3 | Automatic Graph Generation | 67 |
| 5 | Chapter 5 – Implementation | 70 |
| 5.1 | Node Implementation | 70 |
| 5.1.1 | Hardware Setup | 70 |
| 5.1.2 | Software Setup | 71 |

Table of Contents

| | | |
|------------|---|------------|
| 5.2 | Mobile Application Core Implementation | 74 |
| 5.2.1 | Android Plugin for Bluetooth Data | 75 |
| 5.2.2 | Filtering Received Data | 77 |
| 5.2.3 | Automatic Exercise Classification Algorithm | 78 |
| 5.2.4 | Scene Hierarchy & Persistent Game Objects | 80 |
| 5.2.5 | Game Levels Functionality | 83 |
| 6 | Chapter 6 – Experiments | 89 |
| 6.1 | Materials..... | 89 |
| 6.1.1 | Participants..... | 89 |
| 6.1.2 | Apparatus | 89 |
| 6.2 | Methods | 89 |
| 6.2.1 | Qualitative and Quantitative Research..... | 89 |
| 6.2.2 | Experimental Procedure | 90 |
| 6.2.3 | Training – Healthy Subjects & Recovering TKR Patients | 90 |
| 6.2.4 | Testing – TKR Inpatients | 91 |
| 6.3 | Data Analysis | 92 |
| 6.3.1 | ROM Graphs | 92 |
| 6.3.2 | Classification Examples..... | 96 |
| 6.4 | Results | 97 |
| 6.4.1 | ROM Graphs | 97 |
| 6.4.2 | Classification Examples..... | 98 |
| 6.5 | Comments..... | 99 |
| 6.6 | Discussion | 100 |
| 7 | Chapter 7 – Conclusions & Future Work | 102 |
| 7.1 | Limitations | 102 |
| 7.2 | Implications for Future Work | 103 |
| 8 | References – Bibliography..... | 106 |

List of Figures

| | |
|--|----|
| Figure 1: End Application Example..... | 14 |
| Figure 2: Normal knee anatomy. In a healthy knee, these structures work together to ensure smooth, natural function and movement (https://orthoinfo.aaos.org/). | 17 |
| Figure 3: Osteoarthritis often results in bone rubbing on bone. Bone spurs are a common feature of this form of arthritis (https://orthoinfo.aaos.org/). | 18 |
| Figure 4: ROM Measurement Example (https://study.com/)..... | 18 |
| Figure 5: Goniometer used by orthopaedics and physiotherapists for measuring ROM (http://www.medigauge.com/). | 19 |
| Figure 6: Knee before TKR and after adding Implant (http://www.jaipurjointreplacement.com/)..... | 19 |
| Figure 7: Left, Hydrotherapy example (http://sportsmed.in/). Right, Electrotherapy (NMES) example (http://nmesquadrehabpostkneesurgery.blogspot.com/). | 20 |
| Figure 8: Vicon motion capture platform (https://www.vicon.com/). These technologies often use markers for successful motion. | 21 |
| Figure 9: HEXOSYS-II Exoskeleton Prototype, for stroke rehabilitation (Iqbal & Baizid 2015). .. | 22 |
| Figure 10: Electrogoniometer System (Wang et al. 2011). | 22 |
| Figure 11: Polhemus Patriot magnetic system (http://polhemus.com/)..... | 23 |
| Figure 12: Xsens motion capture suit employs 17 MIMUs (Roetenberg et al. 2009). | 23 |
| Figure 13: Hand movement rehabilitation using motion capture and data 5DT gloves (Shen et al. 2008)..... | 24 |
| Figure 14: 17 inertial sensor network (Field et al. 2013). | 25 |
| Figure 15: Use of Kinect in conjunction with p5 glove (Crocher et al. 2013)..... | 25 |
| Figure 16: A screenshot of the Dexteria App developed to train fine motor skills (Rand et al. 2013). | 26 |
| Figure 17: Estimation of 3D joint positions using Kinect (Bo et al. 2011). | 26 |
| Figure 18: Joint angle estimation through sensor network for mobile app (Lee et al. 2014)..... | 27 |
| Figure 19: Sensor network consist of 2 IMUs: (a) on arm, (b) on wrist (Lee et al. 2013). | 27 |
| Figure 20: Single IMU placed on shin (Giggins et al. 2013). | 28 |
| Figure 21: Activity coaching system, consisting of a smartphone and an accelerometer based activity sensor (Harmelink et al. 2017). | 28 |
| Figure 22: Left, relocation measurement error. Right, angle measurement error (Buonocunto & Marinoni 2014)..... | 30 |
| Figure 23: MPU-9150 Pins, Axes of Sensitivity for Accelerometer, Gyroscope, Magnetometer (https://www.invensense.com/)..... | 31 |
| Figure 24: MPU-9150 Register Map Contents. Short description of each register functionality (https://www.invensense.com/)..... | 32 |
| Figure 25: MPU-9150 Register 107 Schematic (https://www.invensense.com/)..... | 32 |
| Figure 26: MPU-9150 Registers 65, 66 Schematic (https://www.invensense.com/). | 33 |
| Figure 27: Quaternion-Based Complementary Orientation Filter (Mahony et al. 2008)..... | 34 |
| Figure 28: Simple block diagram of Kalman filter design (Grewal 2011). | 34 |
| Figure 29: Kalman filter process model (Grewal 2011)..... | 34 |

List of Figures

| | |
|--|----|
| Figure 30: Block diagram of Madgwick orientation estimation algorithm for IMU (Madgwick et al. 2011)..... | 35 |
| Figure 31: Kalman Algorithm Steps (https://www.wikipedia.org/)..... | 35 |
| Figure 32: Left, Arduino BT (https://www.arduino.cc/). Right, Raspberry Pi Zero W (https://www.raspberrypi.org/)..... | 39 |
| Figure 33: Raspberry Pi 40 GPIO pin header (https://www.raspberrypi.org/)..... | 40 |
| Figure 34: Setting Console Autologin through Boot Options in raspi-config..... | 41 |
| Figure 35: Main Screen in raspi-config. Localisation Options are selected. | 41 |
| Figure 36: Enable I ² C. | 42 |
| Figure 37: Game Engine Architecture (Gregory 2009)..... | 43 |
| Figure 38: Rendering Optimization Example. Left, No Culling. Right, Occlusion & Frustum Culling (https://unity3d.com/)..... | 44 |
| Figure 39: Urho 3D Simple Editor Interface (https://urho3d.github.io/)..... | 45 |
| Figure 40: Unreal Engine Sophisticated Editor Interface (https://www.unrealengine.com/).... | 46 |
| Figure 41: Scene in Unity Editor (https://unity3d.com/)..... | 47 |
| Figure 42: Four different types of GameObject: an animated character, a light, a tree, and an audio source (https://unity3d.com/)..... | 48 |
| Figure 43: Component browser (https://unity3d.com/)..... | 48 |
| Figure 44: Unity Asset store (https://unity3d.com/)..... | 49 |
| Figure 45: Execution Order of Event Functions (https://unity3d.com/)..... | 50 |
| Figure 46: Using Unity profiler to track heap allocations (https://unity3d.com/)..... | 53 |
| Figure 47: Android Studio Workspace (https://developer.android.com/)..... | 54 |
| Figure 48: Left, Android Studio SDK Manager Installed APIs (https://developer.android.com/). Right, Unity minimum API defined as 4.2 (Jelly Bean). | 55 |
| Figure 49: Airplane Game example. The user raises the airplane by moving the operated knee. | 58 |
| Figure 50: Airplane game with repetition classified as incorrect. Patient is advised to try again. | 60 |
| Figure 51: Application Main Menu..... | 61 |
| Figure 52: User Submenu..... | 63 |
| Figure 53: Exercise Submenu..... | 63 |
| Figure 54: Game Submenu..... | 64 |
| Figure 55: Stats Submenu. Iterate through graphs using arrows. | 64 |
| Figure 56: IMU HUD Elements. Rom bar, Acceleration Bar, Main and Side Angle Indicators. ... | 65 |
| Figure 57: Different Repetition state hints. | 66 |
| Figure 58: Correct and Wrong Repetition messages. | 66 |
| Figure 59: Pain Rating GUI..... | 66 |
| Figure 60: Final Gameplay HUD. | 66 |
| Figure 61: Coin Score panel and detailed score in pop-up panel. | 67 |
| Figure 62: Specific game hint pop-up panel..... | 67 |
| Figure 63: ROM Percentage Graph for correct repetitions..... | 68 |
| Figure 64: Correct / Incorrect repetitions pie graph..... | 68 |
| Figure 65: Node Components. Left, Raspberry Pi. Center, IMU. Right, Battery. | 70 |
| Figure 66: I ² C Connection implementation between Raspberry Pi Zero W and MPU-9150..... | 71 |
| Figure 67: Resulting sensor node & Elastic bandage for stabilization on patient's knee. | 71 |

List of Figures

| | |
|--|-----|
| Figure 68: i2cdetect show the address an I ² C device is connected. | 74 |
| Figure 69: Automatic Exercise Classification Algorithm Simplified Diagram. | 79 |
| Figure 70: Neutral Position when Ready for Move after Calibration phase. | 84 |
| Figure 71: Upwards Movement. | 85 |
| Figure 72: Downwards Movement..... | 85 |
| Figure 73: Multiple airplanes advanced game play feature..... | 86 |
| Figure 74: Wrong Repetition Implementation..... | 86 |
| Figure 75: Fish Game..... | 87 |
| Figure 76: Multiple fishes and larger fishes advanced game play features..... | 87 |
| Figure 77: Chania General Hospital premises (http://www.chaniahospital.gr/)..... | 91 |
| Figure 78: Photo for patient 1. | 93 |
| Figure 79: Generated ROM Graph for patient 1. | 93 |
| Figure 80: Photo for patient 2. | 94 |
| Figure 81: Generated ROM Graph for patient 2. | 94 |
| Figure 82: Photo for patient 4. | 95 |
| Figure 83: Generated ROM Graph for patient 4. | 95 |
| Figure 84: Generated ROM Graph for patient 8. | 96 |
| Figure 85: Classification Graph for patient 2..... | 97 |
| Figure 86: Classification ROM Graph for patient 8. | 97 |
| Figure 87: Formulae for error percentage calculation of designed algorithm. | 98 |
| Figure 88: Ticwatch E smartwatch (https://www.mobvoi.com/). | 105 |

List of Tables

Table 1: Common Rehabilitation Exercises for TKR. 20

Table 2: Implemented Games for exercises in Table 1 58

1 Chapter 1 – Introduction

The term ‘Serious’ applied to 3D games denotes the application of 3D gaming technologies and playful design strategies to domains of society and culture that are traditionally not associated with entertainment such as the medical domain. What sets Serious Games (SGs) apart from entertainment games is their focus on intentional learning outcomes which are measurable promoting sustained changes in the attitude and performance of individuals. SGs have been successfully employed to promote transformation of processes and protocols to education, as well as practical training and medical activities (Tseklevs et al. 2014, Paraskevopoulos et al. 2014). SG academic research, in combination with several successful practical applications, promoted the ‘serious’ use of 3D games to the forefront of the agendas of diverse fields.

Among diverse areas, research has shown that SGs are effectively enhancing motivation as well as efficiency of rehabilitation training (Wouters et al. 2013, Pirovano et al. 2016). Research challenges that are still prominent in this area include accuracy of training performance, accuracy of motion capture when sensors attached to patients’ body are involved, efficient feedback to the doctor in clinical settings as well as to the patient in home or in any location, therefore, portability of the training environment and implementation of medical and physiotherapy protocols in SG training environments which are equally if not more efficient than traditional rehabilitation methods (Levin et al. 2012).

Portability is most often restricted to either the home or clinical environment because of motion capture and other associated equipment, while the need of a rehabilitation patient who is often required to perform repetitive exercises for a long time, is to be able to perform them anywhere - at home, in the park, outdoors and indoors based on the use of a mobile phone or tablet and a small sensor. The above requirements as well as enhanced portability in any setting are met by the system proposed in this thesis.

The work presented in this thesis puts forward the design and implementation of a custom-made ultra-portable, mobile and low cost 3D rehabilitation application intended for patients that underwent Total Knee Replacement (TKR) (Convery & Beber 1973) using only an Android mobile device and a small sensor placed on the patient’s limb to track movement. The first weeks following knee surgery are crucial so that the Range of Motion (ROM) of the operated knee is deemed fully operational. If the patient fails to perform the exercises appointed by the physiotherapist during this recovery period, an, otherwise, technically accurate operation might result in poor functional outcome leading to reduced quality of life. The aim of our gamified application is to motivate the patient to exercise efficiently by providing feedback, while the physiotherapy exercises are performed in any setting, e.g. clinical, at home, indoors, outdoors or even in public areas.

Initially, a randomly selected control group performs the exercises under physiotherapist supervision who marks them as accurately performed or not. An Inertial Measurement Unit (IMU) node was utilized worn by the patient recognizing limb rotation and acceleration. It is challenging to identify whether the proposed application can classify the exercises as accurately performed or not reliably utilizing just a single sensor node and provide

classification feedback of comparable quality with the physiotherapist feedback. Providing gamified feedback to the patient at home or in other locations in relation to performance using widely available mobile devices, is also challenging, minimizing the need for expensive physiotherapy under supervision, resulting in more engaging and accessible rehabilitation.

This thesis focuses on the description of the rehabilitation system involving the hardware sensor, the implementation of the gamified 3D environment, as well as the initial testing of the software framework in the hospital, while patients are undergoing physiotherapy treatment. The scope of this project involves the development of an integrated ultra-portable gamified platform, including gamified tasks to be utilized for rehabilitation exercises commonly performed after TKR surgery (Figure 1). The main goal is to improve compliance to the physiotherapy protocol, increase patient engagement, monitor physiological conditions and provide feedback based on rewards via a gamified experience.

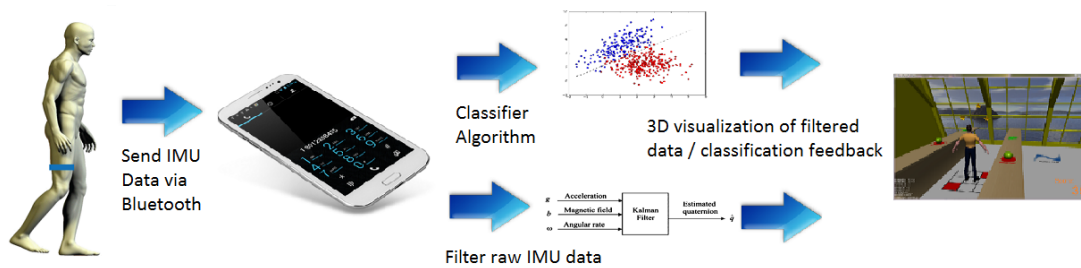


Figure 1: End Application Example.

1.1 Contribution

The current framework introduces an ultra-portable rehabilitation application comprising of just a single IMU sensor linked to a 3D gamified environment, to be adopted by patients that have undergone TKR surgery for their highly repetitive, but very significant post-operative physiotherapy. Although this framework uses a custom made sensor incorporating a Raspberry Pi Zero W single-board computer, the framework can be generalized to any wearable device that employs an accelerometer and gyroscope and complies with the specified raw data communication format. A first iteration of an automatic exercise classification algorithm is proposed, that uses angular and acceleration predefined thresholds to fine-tune the system. In future iterations, these thresholds can be inferred from variant Machine Learning techniques, e.g. RVMs (Tipping 2003), using the filtered sensor data collected. Such methods will maximize the success rate of the current algorithm, using just a single node as long as there is a capable number of samples collected by patients.

This application engages the patient to accurately perform the recovery exercises through a gamified 3D experience, ultimately minimizing physiotherapist supervision, at most locations. The goal is to understand in a qualitative manner the feedback collected from the patients. In this iteration, quantitative measures that include ROM and classification feedback were provided. The collected data provide qualitative feedback towards understanding the user's and physiotherapist expectations in relation to the proposed rehabilitation system. A relationship between quantitative measures (ROM) and quality of physiotherapy and knee recovery is investigated. Results from the experiments with patients in the hospital in this first

iteration effectively encourage this investigation. In 80% of the collected samples, the maximum ROM percentage measurements are following an ascending pattern. The same patients were still trying to improve their previous repetition while engaging in a serious game using the proposed framework. Short term measurements encourage the use of rehabilitation using serious games. Nevertheless a long term study will be necessary to provide accurate results on the quality of knee recovery using the proposed gamified approach in physiotherapy.

1.2 Thesis Outline

This thesis is divided into a number of chapters, which will be outlined below.

Chapter 2 – Existing Work & Background: This chapter introduces a set of fundamental terms regarding medical background concerning anatomy of the knee, causes that can lead to Knee Replacement surgery and recovering process through physiotherapy. The basic rehabilitation techniques after TKR and exercises are analyzed. Subsequently, a technical background of state of the art technologies for limb rehabilitation is provided. Furthermore, Inertial Measurement Unit (IMU) technology is analyzed which enables low-cost and portable limb motion tracking as it is the building block used in this thesis. The IMU MPU-9150 utilized is presented along with an overview of common filtering methods. Moreover, the Raspberry Pi single-board computer is described which along with an IMU sensor compose the motion tracking node of the proposed system.

Chapter 3 – Overview of IDE & Development tools: In this chapter, the development tools employed in this project are introduced. An overview of popular game engine choices is summarized that leads to the current project game engine, Unity 3D. Building blocks of Unity are described along with methods for scripting objects, design patterns used and profiling tools. These tools enable Game and Mobile development for android. Technical details are discussed that enable the combination of android native development with Unity. The resulting plugin is necessary for Bluetooth communication between the sensor node and the android mobile device.

Chapter 4 – Design: Chapter 4 describes in detail the design of the proposed gamified mobile rehabilitation system involving the hardware sensor, the gamified 3D environment, the patient role as a player of a rehabilitation game, the exercise repetition protocol and the received gamification feedback after a single repetition. This feedback can be either real-time visual feedback of events happening in the game or the binary classification feedback of the implemented algorithm that classifies a single repetition as correct / incorrect under the specified physiotherapist criteria. This design in turn leads to the fundamental User Interface implementation of the application. Main Menu Design, Gameplay Design and Statistical Graph Generation of the application are described here. Statistical Graph Generation is an additional feedback type of all the performed repetitions that provides classification error and ROM statistics for each patient. These graph statistics are used extensively in the results section.

Chapter 5 – Implementation: Chapter 5 describes in detail the implementation of the proposed gamified mobile application framework. Details about the custom made motion

tracking sensor node implementation are provided in a hardware and software level. These details include the I²C connection of the hardware node which is the way the IMU communicates with the Raspberry PI and the setup of the sensor as a Bluetooth server to accept incoming connections from android mobile devices that use the implemented application. Key elements of the application functionality in the mobile device are provided. These elements include Bluetooth communication between application and sensor node, 3D Scene Implementation and a first iteration of an automatic exercise classification algorithm. Here the game levels functionality implemented is described along with the whole rehabilitation exercise pipeline of a single repetition from the moment a patient starts the limb orientation till the patient returns to the neutral exercise pose. Snippets of code are provided that handle Bluetooth connectivity, game events, exercise serialization events.

Chapter 6 – Experiments: This chapter is concerned with the Experimental Methods employed when the actual experiments were conducted in the Orthopaedic Clinic of the Chania General Hospital. The experimental procedure included healthy subjects as well as TKR patients. These patients could be either recovering TKR patients, in physio follow up, 12-14 days post-op, or TKR patients 48h post op after the removal of the drain. The exercise studied was Knee Extension and the implemented game, the Airplane game, where the patient is instructed to raise the knee and observe the movement of the airplane along with the movement of the knee. At the same time a manual training / fine-tuning of the classification algorithm parameters was performed on healthy subjects and recovering TKR patients. This led to the testing of the framework on the 48h post op TKR patients and the extraction of qualitative and quantitative results. The quantitative results are based on the automatically generated application ROM and classification graphs. The qualitative results are based on patient behavior and engagement while using the proposed framework. ROM improvements (quantitative measure) over different testing sessions contribute to enhanced user engagement (qualitative measure), while ROM declination is observed on patients that failed to engage in the game experience.

Chapter 7 – Conclusions & Future Work: In the final chapter, the conclusions of this thesis are presented as well as limitations & hints about future work. Limitations include poor User Experience on the unsupervised positioning of the sensor node, few TKR measurements in Chania municipality, lack of long term observations, fixed algorithm thresholding. Future work should strive to address these issues by improving UX of the framework, using commercial smartwatch sensors, gathering long term observations and capable amount of data to apply machine learning technics thus improving classification algorithm success rates.

2 Chapter 2 – Existing Work & Background

2.1 Medical Background

2.1.1 Knee Anatomy & Reduced Functionality Causes

The knee is the largest joint in the body and having healthy knees is required to perform most everyday activities. The knee is made up of the lower end of the thighbone (femur), the upper end of the shinbone (tibia), and the kneecap (patella) (Figure 2). The ends of these three bones where they touch are covered with articular cartilage, a smooth substance that protects the bones and enables them to move easily. The menisci are located between the femur and tibia. These C-shaped wedges act as ‘shock absorbers’ that cushion the joint. Large ligaments hold the femur and tibia together and provide stability. The long thigh muscles give the knee strength. All remaining surfaces of the knee are covered by a thin lining called the synovial membrane. This membrane releases a fluid that lubricates the cartilage, reducing friction to nearly zero in a healthy knee.

Normally, all of these components work in harmony. But disease or injury can disrupt this harmony, resulting in pain, muscle weakness, and reduced function.

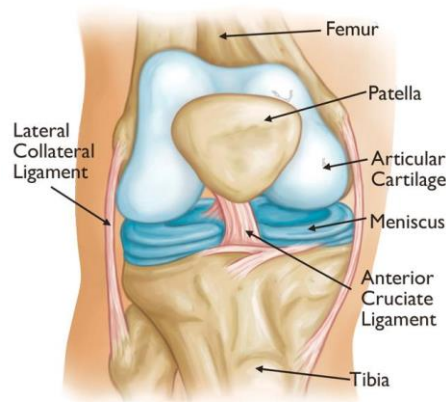


Figure 2: Normal knee anatomy. In a healthy knee, these structures work together to ensure smooth, natural function and movement (<https://orthoinfo.aaos.org/>).

The most common cause of chronic knee pain and disability is arthritis. Although there are many types of arthritis, most knee pain is caused by just three types: osteoarthritis, rheumatoid arthritis, and post-traumatic arthritis.

- **Osteoarthritis.** This is an age-related ‘wear and tear’ type of arthritis. It usually occurs in people 50 years of age and older, but may occur in younger people, too. The cartilage that cushions the bones of the knee softens and wears away. The bones then rub against one another, causing knee pain and stiffness.
- **Rheumatoid arthritis.** This is a disease in which the synovial membrane that surrounds the joint becomes inflamed and thickened. This chronic inflammation can damage the

cartilage and eventually cause cartilage loss, pain, and stiffness. Rheumatoid arthritis is the most common form of a group of disorders termed ‘inflammatory arthritis’.

- **Post-traumatic arthritis.** This can follow a serious knee injury. Fractures of the bones surrounding the knee or tears of the knee ligaments may damage the articular cartilage over time, causing knee pain and limiting knee function.

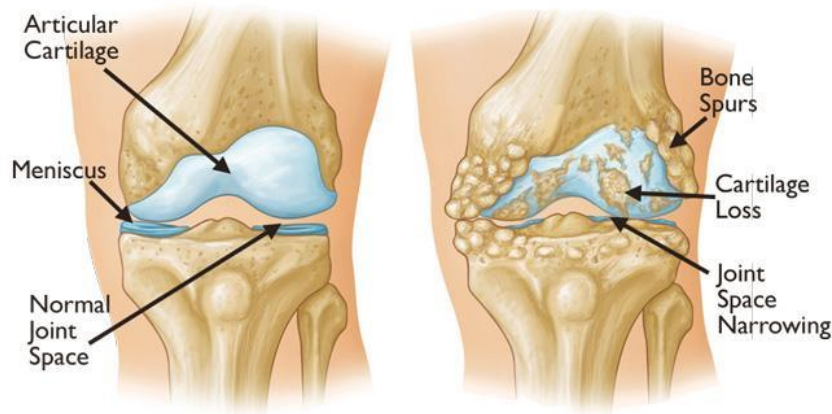


Figure 3: Osteoarthritis often results in bone rubbing on bone. Bone spurs are a common feature of this form of arthritis (<https://orthoinfo.aaos.org/>).

All the aforementioned conditions can effectively limit joint Range of Motion (ROM) (Figure 4). ROM is the measurement of movement around a specific joint or body part. It is measured in degrees from the center of the knee. ROM includes flexion (bending), extension (straightening), adduction (movement towards center of the body), abduction (movement away from center of the body), rotations (inward and outward).

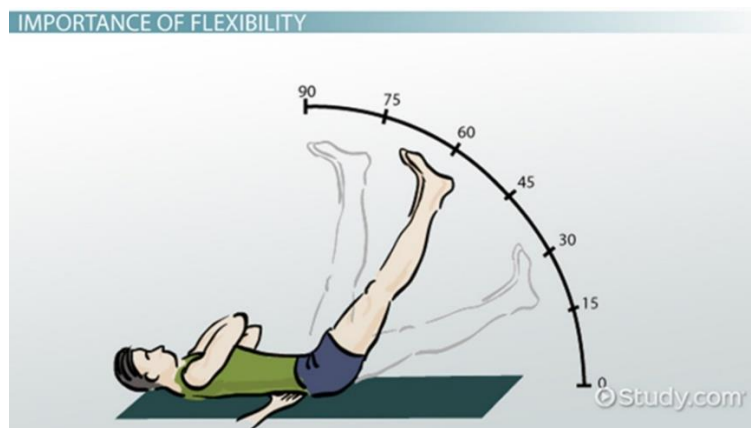


Figure 4: ROM Measurement Example (<https://study.com/>).

ROM is measured using an instrument called a ‘goniometer’ (Figure 5). For instance, a completely straight knee joint measure 0° while a fully bent knee clocks in at about 135° degrees of flexion.



Figure 5: Goniometer used by orthopaedics and physiotherapists for measuring ROM (<http://www.medigauge.com/>).

2.1.2 TKR

Total Knee Arthroplasty (TKA), also known as Total Knee Replacement (TKR), is one of the most commonly performed orthopedic procedures (Figure 6). As of 2010, over 600,000 total knee replacements were being performed annually in the United States and were increasingly common (Martin et al. 2014). Among older patients in the United States, the per capita number of primary total knee replacements doubled from 1991 to 2010 (from 31 to 62 per 10,000 Medicare enrollees annually). The number of total knee replacements performed annually in the United States is expected to grow by 673 percent to 3.48 million procedures by 2030. A variety of pathologic conditions affecting the knee can be treated with total knee replacement, leading to pain relief, to restoration of function, and to mobility.

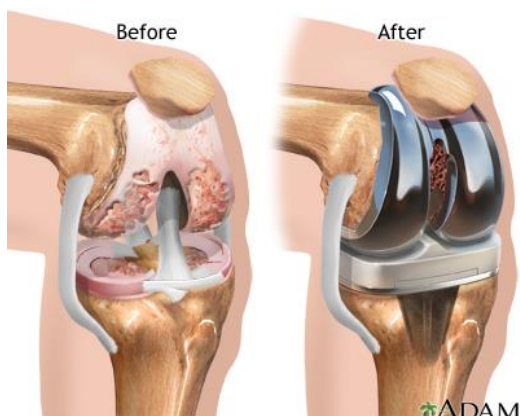


Figure 6: Knee before TKR and after adding Implant (<http://www.jaipurjointreplacement.com/>).

Total knee replacement is one option to relieve pain and to restore function to an arthritic knee. The most common reason for knee replacement is that other treatments (weight loss, exercise/physical therapy, medicines, and injections) have failed to relieve arthritis-associated knee pain. The goal of knee replacement is to relieve pain, improve quality of life, and maintain or improve knee function. The procedure is performed on people of all ages, with the exception of children, whose bones are still growing. It is important to have significant pain and/or disability prior to considering this procedure. Because the replacement parts can break down over time, and healthcare providers generally recommend delaying knee replacement until it is absolutely necessary.

2.1.3 Rehabilitation Exercises for TKR

Physical therapy is an essential part of rehabilitation after total knee replacement. The first weeks following knee surgery are crucial so that the Range of Motion (ROM) of the operated knee is deemed fully operational. Insufficient exercise at this phase predisposes patients to disability with increasing age (Stevens-Lapsley et al. 2012). Traditionally a physiotherapist instructs a number of exercises to the patient. The patient should repeat these exercises in regular basis in order to successfully recover motion of the knee. The exercises in question are selected by the physiotherapists based on the American Academy of Orthopaedic Surgeons TKR exercise guide (Parvizi et al. 2008). The physiotherapists supervise patients during each exercise repetition in order to determine whether it was performed in a compliant manner, involving the appropriate body posture speed rate. The physiotherapist evaluates the performance of the exercises. Common Exercises utilized are shown in Table 1. These exercises form the base of the current rehabilitation framework implementation. Moreover, the knee extension exercise is used extensively throughout our experimental procedure.

| Exercise | Sensor Placement | Description |
|--------------------|------------------|--|
| Knee Extension | Shin | From sitting position, the leg is extended, then lowered back to starting position. |
| Straight Leg Raise | Shin | From lying on back position, the leg is lifted and then slowly lowered back to starting position. |
| Heel Slide | Shin | From lying on back position, the heel is slowly moved up and then slowly moved down to starting position. |
| Lying Kicks | Shin | From lying on back position, an object is inserted under the knee, thus, raising it. Then, the leg is raised and lowered back again. |

Table 1: Common Rehabilitation Exercises for TKR.

There can be variant categories of post-operative physiotherapy interventions. Hydrotherapy, e.g. exercise in a warm water environment when recovering from knee surgery, was associated with comparable outcomes with land-based rehabilitation up to 26 weeks post-surgery (Harmer et al. 2009). Hydrotherapy, though, requires specific environmental set-up restricting rehabilitation portability. Electrotherapy through muscle neuromuscular electrical stimulation (NMES), initiated 48 hours after TKR, effectively improved functional performance following TKR (Stevens-Lapsley et al. 2012). Physiotherapist attaches patches to the patient that give a small electrical shock. This shock empowers the muscles of the knee and help in performing the required exercise. This method is suitable for recovering patients a few weeks after surgery. For inpatients while it can be done, it can cause undesirable pain to the patient.

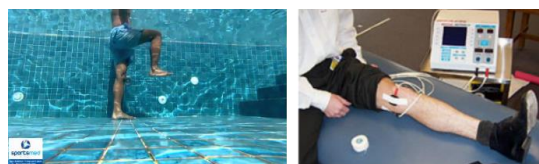


Figure 7: Left, Hydrotherapy example (<http://sportsmed.in/>). Right, Electrotherapy (NMES) example (<http://nmesquadrehabpostkneesurgery.blogspot.com/>).

The method proposed in this work motivates the patient to actively put effort in order to perform the exercises, in contrast with electrotherapy for which the motion is performed passively based on electrical stimulation of the muscle cells. The scope of this project is to examine if a visual interactive stimulation through gaming can help the patient to focus on the game instead of the pain or discomfort of the exercise.

2.2 Existing Technologies for Limb Motion Tracking

2.2.1 Optical Systems

They use visual data captured by one or more cameras to triangulate the 3D position of a set of points detected, with the aid of markers attached to the body (Figure 8). Some of them can reach a high precision (i.e, in the range of few millimeters), but also have very high costs, like for example the Vicon. A cheaper solution is represented by Microsoft Kinect, which uses only one RGB camera and an infrared depth sensor composed by an infrared laser grid scanner and a related infrared camera. It is not as precise as the more expensive systems: its precision is in the order of centimeters instead of millimeters, and has a limited maximum range (5m). It also suffers by all typical disadvantages of optical systems, the most important are the need of instrumenting the scene, preparing a setup of cameras and the possibility of operating properly only in the presence of line of sight between the cameras and the object being tracked. There are other kind of systems that exploit the high frequency interference patterns caused by lasers (Zizka et al. 2011) and sense direct or reflected light using a high framerate lensless 2D image sensor. They can be used to track fast (i.e, up to 1000Hz) motion using minimal hardware. Despite their lower cost and much simpler hardware requirement than camera-based sensor, these systems still need the instrumentation of the scene.

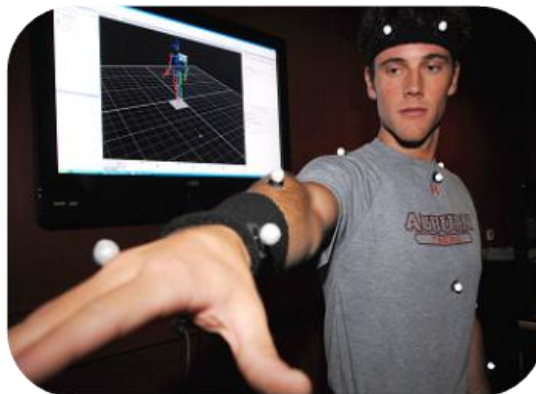


Figure 8: Vicon motion capture platform (<https://www.vicon.com/>). These technologies often use markers for successful motion.

2.2.2 Exo-skeletons

They are rigid structures of jointed, straight metal or plastic rods linked together with potentiometers or encoders that articulate at the joints of the body (Solazzi et al. 2010, Iqbal & Baizid 2015) (Figure 9). These systems offer real-time, high precision acquisition and have the

advantage of not being influenced by external factors (Menache 2000), such as visual occlusion, quality and the number of cameras, and they have no limit on maximum capture volume. Some systems also provide haptic limited force feedback. Their main disadvantage is the movement limitation imposed by the mechanical constraints of the exoskeleton structures (Harada et al. 2004).



Figure 9: HEXOSYS-II Exoskeleton Prototype, for stroke rehabilitation (Iqbal & Baizid 2015).

2.2.3 Electrogoniometers

Flexible electrogoniometers are widely used to measure human joint movements, with applications ranging from medicine (e.g. kinesiology, rehabilitation, diagnostics) to virtual reality interfaces and computer peripherals (Wang et al. 2011). Their advantage over conventional potentiometric goniometers is that they adapt better to body parts and are not sensitive to misalignments concomitant with the movement of polycentric joints. On the other hand, the major weakness of this technology is its relatively high cost (Figure 10).

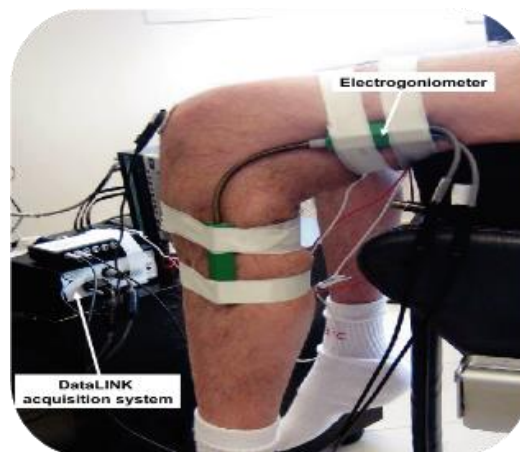


Figure 10: Electrogoniometer System (Wang et al. 2011).

2.2.4 Magnetic Systems

They calculate the position and orientation of a magnetic sensor probe moving in a magnetic field generated by 3 orthogonal coils (Figure 11). The main disadvantage of these systems is that they are susceptible to magnetic and electrical interference from metal objects

in the environment, which affect the magnetic field, or electromagnetic sources, such as monitors, lights, cables and computers.



Figure 11: Polhemus Patriot magnetic system (<http://polhemus.com/>).

2.2.5 Inertial Measurement Units

Inertial Motion Capture (Roetenberg et al. 2009) technology is based on miniature inertial sensors, biomechanical models, and sensor fusion algorithms (Figure 12). The motion data of the inertial sensors (inertial guidance system) is often transmitted via wireless to a computer, where the motion is recorded or viewed. Most inertial systems use gyroscopes to measure rotational speed. These rotations are translated to a skeleton by the software. Inertial motion systems capture the full 6 degrees of freedom body motion of a human in real-time and can also include a magnetic sensor, although these have a much lower resolution and are susceptible to electromagnetic noise. Benefits of using inertial systems include low cost, small dimensions, portability, and large capture areas. Disadvantages include lower positional accuracy and positional drift.



Figure 12: Xsens motion capture suit employs 17 MIMUs (Roetenberg et al. 2009).

On the current project, IMUs were preferred over the aforementioned technologies. IMUs respect the current framework's main design constraints of low cost and portability. By employing a single IMU the portability can be maximized while the accuracy remains in acceptable levels (Giggins et al. 2014, Huang et al. 2016). There are trade-offs to consider when selecting a desired technology concerning factors such as accuracy, portability, cost. In that aspect, exo-skeletons provide high accuracy in measurements but limited portability and

very high cost at the time of writing. Optical systems and specifically the Kinect system could also be an interesting choice and can be seen in rehabilitation systems at home due to its acceptable accuracy and low cost. On the other hand, a single IMU sensor offers enhanced portability since it doesn't limit rehabilitation only at home. Instead it allows rehabilitation anywhere in the park, outdoors and indoors based on the use of a mobile phone or tablet and a small sensor allows rehabilitation at practically any place. These trade-offs are to be considered in the multiple IMUs solution compared to a single IMU solution. Multiple IMUs provide higher accuracy (Huang et al. 2016) but raise the cost while lowering portability. The accuracy provided by a single IMU solution after applying Kalman filtering to the sensor data can be acceptable for the case of simple knee rehabilitation exercises provided that critical factors are handle properly. One such factor is the proper node placement of the sensor on the appropriate body part. IMUs are very sensitive to misalignments. Users of the application must be instructed to properly position the sensor, so they will be able to use it by themselves when performing rehabilitation without supervision.

2.3 Related Works & Applications

Several approaches listed below have been employed to track limb motion of diverse precision, cost and complexity. Although the focus is mainly on the low-cost solutions inertial sensor systems can provide, useful results are provided from existing frameworks in costly applications.

2.3.1 Rehabilitation Projects

An Augmented reality (AR) system is proposed for the rehabilitation of hand movements which have been impaired due to illness or accident (Shen et al. 2008). Through the proposed system, the patient can practice daily at home utilizing a standard computer, a webcam and two wireless 5DT data gloves (Figure 13). Using AR technology, a highly controllable environment including tasks of varied difficulty levels is provided to the patients for them to perform the exercise gradually and systematically. A similar technique was used and tested on 8 stroke patients (Cameirão et al. 2011). The results were positive compared to conventional therapy. The rehabilitation gaming system group displayed significantly improved performance in paretic arm speed. The use of the data gloves, however, raises the cost and the technical complexity of this system which may require specialized technical support and maintenance.

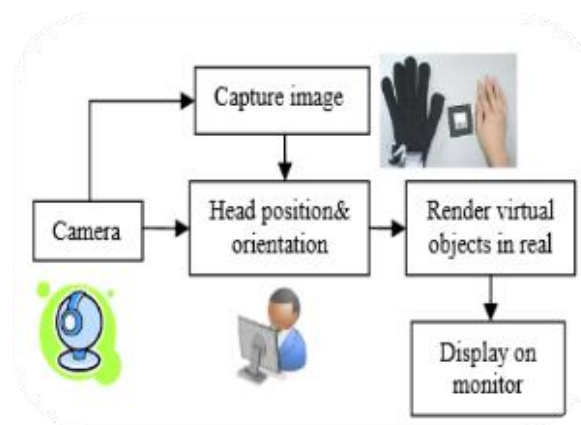


Figure 13: Hand movement rehabilitation using motion capture and data 5DT gloves (Shen et al. 2008).

Furthermore, 3 methods were proposed for representing changes in human motion symmetry during injury rehabilitation (Field et al. 2013). In this context, a motion capture suit was employed with 17 inertial sensors (each worth about 2000 \$) to measure body postures (Figure 14). The methods are tested on an injured athlete over 4 months of recovery from an ankle operation and validated by comparing the observed improvement to the variation among a group of uninjured subjects. The results indicate that gradual changes are detected in the motion symmetry, thus providing quantitative measures to aid clinical decisions. Methods such as this one that employ a sensor network provide higher measurement accuracy as mentioned in the previous section but greatly increase the application cost and decrease portability.

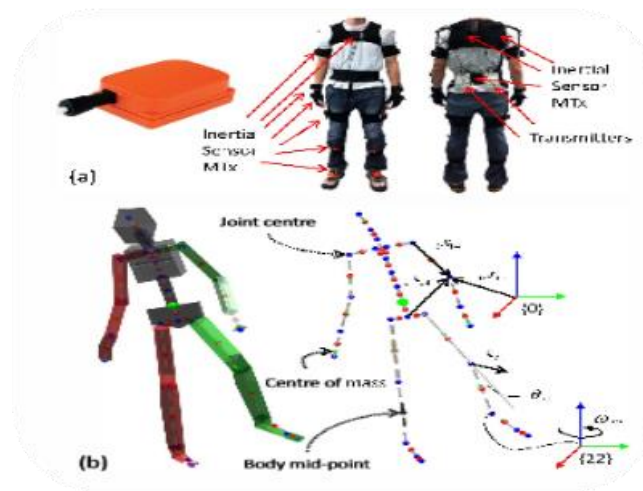


Figure 14: 17 inertial sensor network (Field et al. 2013).

Several rehabilitation studies using motion tracking employ the Kinect optical system, which, however, requires specialized set-up and technical support (Figure 15). Previous research has identified patients' general expectations for rehabilitation games and evaluated two newly developed low-cost puzzle and archery rehabilitation games through surveys (Crocher et al. 2013). Testing on 3 post-stroke patients identified the need to improve reliability and precision of the low-cost hardware as well as to demonstrate clinical benefits.



Figure 15: Use of Kinect in conjunction with p5 glove (Crocher et al. 2013).

There also exists iPad applications (Rand et al. 2013) that exploit sensory input (light, touch and accelerometer) in order to improve hand function post-stroke (Figure 16). Preliminary findings point to the potential of using apps in the process of post-stroke hand rehabilitation. However, the input for this method is limited at finger tapping, not suitable for more complex exercises and not based on motion tracking data providing feedback to the patient.



Figure 16: A screenshot of the Dexteria App developed to train fine motor skills (Rand et al. 2013).

2.3.2 IMU Example Applications

There are several AR experimental applications that employ IMUs, in virtual rehabilitation. There are frameworks that correct sensor measurements with the use of Kinect (Figure 17) and Kalman Filtering (Bo et al. 2011). In other projects combination of sensors were assessed in order to be implemented as a multi-sensor unit for a portable arm rehabilitation device (Ambar et al. 2012). These sensors include, besides an accelerometer, a flex sensor instead of a gyroscope for specific angle measurement during limb bending and a force sensitive sensor to measure limb pressure.

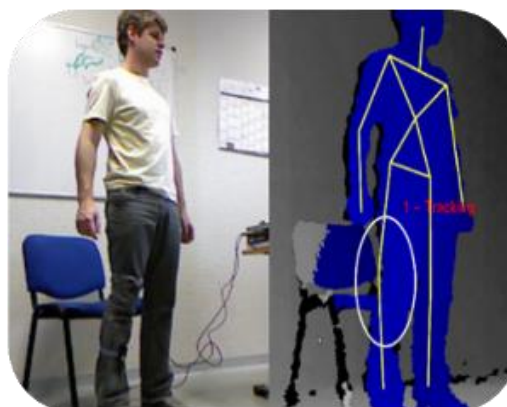


Figure 17: Estimation of 3D joint positions using Kinect (Bo et al. 2011).

Typically multiple IMUs are used to create a network that computes body limb orientation. One such sensor network system is used to automate a portion of the Upper-

Extremity Fugl-Meyer Assessment (FMA), which is widely used to quantify motor deficits in stroke survivors (Lee et al. 2014). The system has the ability to automatically identify the assessment item being conducted for the FMA test and calculate the maximum respective joint angle achieved. The system has shown comparable levels of accuracy in measuring a range of joint angles (Figure 18) when compared to the goniometer used by clinicians (compared measurements from 2 therapists).

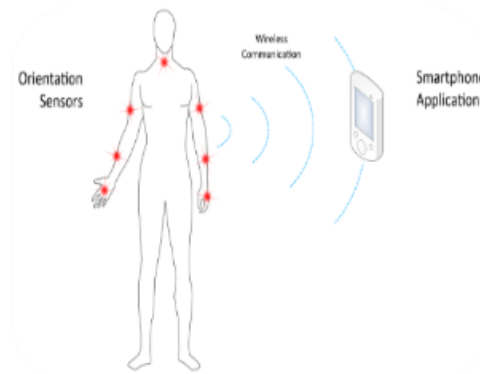


Figure 18: Joint angle estimation through sensor network for mobile app (Lee et al. 2014).

IMUs are widely used for gait cycle observation. An example framework on this topic describes an improved IMU-based gait analysis processing method that uses gyroscope angular rate reversal to identify the start of each gait cycle during walking (Hundza et al. 2014). In validation tests with 6 subjects with Parkinson disease (PD), including those with severe shuffling gait patterns, and 7 controls, the probability of True-Positive event detection and False-Positive event detection was 100% and 0%, respectively.

Having a large sensor network can increase the computational demands and cost of an application. Reducing number of sensors can provide portable results with some loss in accuracy (Giggins et al. 2014, Huang et al. 2016). An example of such a framework uses a network of 2 wireless sensor nodes (arm, wrist) in order to classify 6 rehabilitation exercises (Figure 19) for the frozen shoulder condition by means of an artificial neural network (ANN) algorithm (Lee et al. 2013). As results, 5 of 6 exercises are successfully recognized with 85-90% of accuracy rates but the complex one (i.e. the spiral rotation exercise) reached only around 60%.



Figure 19: Sensor network consist of 2 IMUs: (a) on arm, (b) on wrist (Lee et al. 2013).

There are also projects that examine the use of a single IMU allowing maximum portability. One such work uses a single IMU node on the shin (Figure 20) to examine if it is capable of identifying conditions of poor technique during 7 lower limb exercises (Giggins et al. 2013). The data presented has revealed that a single IMU can be used to identify the conditions of poor technique during 5 of the 7 exercises studied. Moreover, a single IMU system is also designed capable of measuring tibia angle using a shank mounted wireless inertial sensor. The system employs a simple setup with a single skin-mounted triaxial accelerometer and gyroscope module attached to the tibia segment, and an algorithm to estimate the tibia angle, achieving an RMSE of 1.6 ± 1.1 and 2.5 ± 1.6 degrees in tibia-flexion and tibia-adduction angles. Such results further encourage the use of a single inertial unit in Total Knee Replacement (TKR) patients additionally using the gamification strategy proposed in this thesis in order to extract quantitative results on the Range of Motion (ROM) of the patients.



Figure 20: Single IMU placed on shin (Giggins et al. 2013).

New interactive technologies have been recently applied to rehabilitation sessions with the aim to increase strength and balance while improving patient stimulation, compliance and satisfaction with treatment. The effectiveness of an activity coaching system including an accelerometer-based activity sensor, alongside a home-based exercise program has been examined (Harmelink et al. 2017). A hand-held electronic device was connected to a mobile application on a smartphone providing information and advice on exercise behavior during the day. There are no conclusive results yet, but the expectation is that using the system will result in an increase of physical functioning in the group receiving the activity coaching system. This coaching system can be a very useful addition to be used alongside with a gamification system since it can improve user experience and provide useful statistics during the day that later can be send to health professionals for assessment (e.g. doctors or physiotherapists).



Figure 21: Activity coaching system, consisting of a smartphone and an accelerometer based activity sensor (Harmelink et al. 2017).

The use of integrating the Wii-Fit game into a rehabilitation paradigm after TKA has also been investigated (Fung et al. 2012). In addition to standard therapy, users received 15 minutes of Wii-Fit gaming activity, while the control group received 15 minutes of additional lower extremity exercise. There were no differences between groups for ROM. These findings suggest that the addition of Wii-Fit as an alternative to lower extremity strengthening may be an appropriate rehabilitation tool further encouraging gamification strategies on rehabilitation. Wii-Fit provides a low cost solution to rehabilitation practices, although restricting the rehabilitation mostly at home, similarly to Kinect based approaches.

The use of new digital technologies encourages the further investigation of automatic exercise rehabilitation classification. There exists an increasing number of past research that employ IMU nodes for evaluation of limb rehabilitation exercises (Lee et al. 2013, Giggins et al. 2013, Huang et al. 2016). Previous work indicates that when multiple IMUs are employed, satisfactory exercise classification accuracy results are achieved based on three, two and one IMUs (Lee et al. 2013). Such results drive the further investigation of the challenging classification problem using just a single IMU in this thesis, enhancing maximum portability compared to multiple IMU systems, in conjunction with sufficiently high success rates of movement detection. Employing more than two sensor nodes can achieve higher accuracy in ROM measurement under certain conditions, e.g. when accurate node placement is ensured (Huang et al. 2016), however, such systems are difficult to operate and of limited portability.

2.4 IMU Functionality

The desired portability can be provided by the use of inertial sensors in the application design process. Inertial 9-axis or 6-axis sensors provide a low-cost solution for rehabilitation procedures. 6-axis sensors contain accelerometer and gyroscope, for relocation and angle measurement respectively. 9-axis sensors additionally include a magnetometer for more accurate and complete results. On the other hand magnetometer measurements are susceptible to noise caused by gravitational fields and electricity. One should consider the application requirements in order to choose which type of sensor to use.

2.4.1 MPU-9150 Overview

Inertial Measurement Units (IMUs) provide the leading technology used in smartphones and wearable devices in order to measure rotational and translational movements. In this project, the IMU MPU-9150 is used, which is small in size, cheap and portable. The utilized sensor node contains several sensors:

- *3-axis Accelerometer*. The accelerometer measures inertial force caused by gravity and acceleration. It can accurately measure rotation, however, it is susceptible to noise caused by rapid changes of acceleration. In order to be able to capture orientation along the z-axis, a magnetometer (compass) is used complementary to the accelerometer.
- *3-axis Gyroscope*. The gyroscope measures the rate of change of any angle at a specified frequency, e.g. 100 Hz. This makes it suitable for short-term observations and fast rotational signal changes. In relation to long-term

observations, it is susceptible to drift errors. Then, measurements should be sampled in exact intervals according to a specified frequency.

- *3-axis Magnetometer.* The magnetometer measures the earth's magnetic field. It is used in conjunction to the gyroscope sensor in order to capture rotation around z-axis.
- *Temperature sensor.* Measuring environmental temperature.

The suggested IMU to be used is the MPU-9150 9-Axis (Gyro + Accelerometer + Compass) MEMS MotionTracking™ Device. Accuracy error measurements for relocation and angle for this IMU are presented (Figure 22).

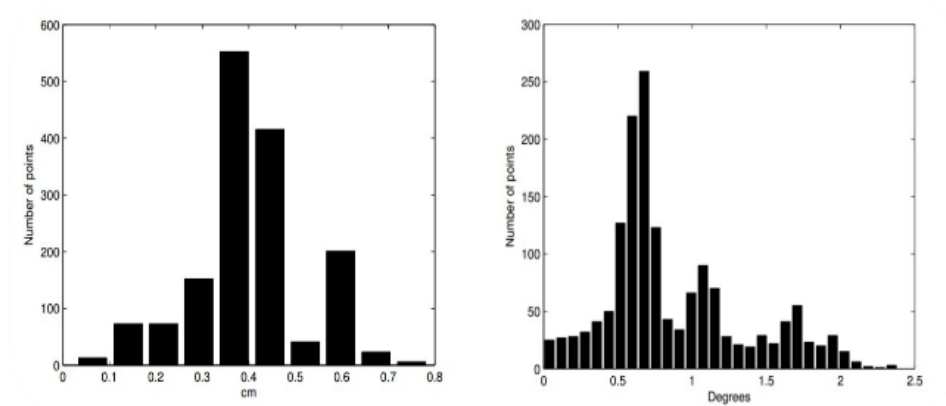


Figure 22: Left, relocation measurement error. Right, angle measurement error (Buonocunto & Marinoni 2014).

For more information on setting up and reading data from MPU 9150, there are 2 documents that must be studied:

- MPU-9150 Product Specification.
- MPU-9150 Register Map.

Below these 2 documents are briefly described.

2.4.2 MPU-9150 Product Specification

The product specification provides information regarding the electrical specification and design related information for the MPU-9150™ Motion Processing Unit™ or MPU™. Electrical characteristics are based upon design analysis and simulation results only.

The MPU-9150 is the world's first integrated 9-axis MotionTracking device that combines a 3-axis MEMS gyroscope, a 3-axis MEMS accelerometer, a 3-axis MEMS magnetometer and a Digital Motion Processor™ (DMP™) hardware accelerator engine. The MPU-9150 is an ideal solution for handset and tablet applications, game controllers, motion pointer remote controls, and other consumer devices. The MPU-9150's 9-axis MotionFusion combines acceleration and rotational motion plus heading information into a single data stream for the application. This MotionProcessing™ technology integration provides a smaller footprint and has inherent cost advantages compared to discrete gyroscope, accelerometer, plus magnetometer solutions. In smartphones, it finds use in applications such as gesture

commands for applications and phone control, enhanced gaming, augmented reality, panoramic photo capture and viewing, and pedestrian and vehicle navigation. With its ability to precisely and accurately track user motions, MotionTracking technology can convert handsets and tablets into powerful 3D intelligent devices that can be used in applications ranging from health and fitness monitoring to location-based services. Key design constraints are small package size, low power consumption, high accuracy and repeatability, high shock tolerance, and application specific performance programmability – all at a low consumer price point.

The MPU-9150 features three 16-bit analog-to-digital converters (ADCs) for digitizing the gyroscope outputs, three 16-bit ADCs for digitizing the accelerometer outputs and three 13-bit ADCs for digitizing the magnetometer outputs. For precision tracking of both fast and slow motions, the parts feature a user programmable gyroscope full-scale range of ± 250 , ± 500 , ± 1000 , and $\pm 2000^\circ/\text{sec}$ (dps), a user programmable accelerometer full-scale range of $\pm 2g$, $\pm 4g$, $\pm 8g$, and $\pm 16g$, and a magnetometer full-scale range of $\pm 1200\mu\text{T}$. The MPU-9150 is a multi-chip module (MCM) consisting of two dies integrated into a single LGA package. One die houses the 3-Axis gyroscope and the 3-Axis accelerometer. The other die houses the AK8975 3-Axis magnetometer from Asahi Kasei Microdevices Corporation. Additional features include an embedded temperature sensor and an on-chip oscillator with $\pm 1\%$ variation over the operating temperature range.

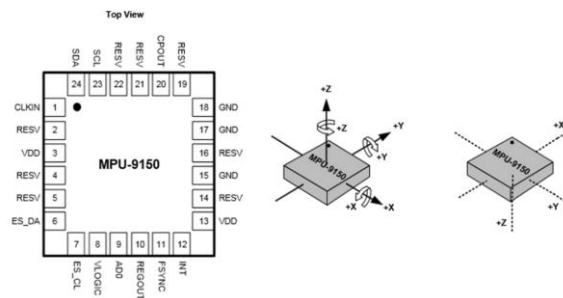


Figure 23: MPU-9150 Pins, Axes of Sensitivity for Accelerometer, Gyroscope, Magnetometer (<https://www.invensense.com/>).

2.4.3 MPU-9150 Register Map

This document provides preliminary information regarding the register map and descriptions for the MPU-9150™ Motion Processing Unit™ or MPU™. The outputs from MPU-9150 are not analog signals expressed in e.g. Volts. They are digitized signals that use ADCs as mentioned in the specification above. The output from MPU-9150 as well as the configuration input to MPU-9150 is stored in 8 bit registers. Communication with all registers of the device is performed using I²C at 400 kHz. I²C is a bus protocol that allows short-distance, serial data transfer. The addresses on the bus are 7 bits wide. So a register address can range between 0-2⁷. This follows that the device can have maximum 128 registers to read or write. The Register Map document contains information about the available Registers for read/write (Figure 24).

| | | |
|----------|--|----------|
| 1 | REVISION HISTORY | 4 |
| 2 | PURPOSE AND SCOPE | 5 |
| 3 | REGISTER MAP FOR GYROSCOPE AND ACCELEROMETER | 6 |
| 4 | REGISTER DESCRIPTIONS FOR GYROSCOPE AND ACCELEROMETER | 9 |
| 4.1 | REGISTERS 13 TO 16 – SELF TEST REGISTERS | 9 |
| 4.2 | REGISTER 25 – SAMPLE RATE DIVIDER | 9 |
| 4.3 | REGISTER 26 – CONFIGURATION | 11 |
| 4.4 | REGISTER 27 – GYROSCOPE CONFIGURATION | 12 |
| 4.5 | REGISTER 28 – ACCELEROMETER CONFIGURATION | 13 |
| 4.6 | REGISTER 35 – FIFO ENABLE | 13 |
| 4.7 | REGISTER 36 – I ² C MASTER CONTROL | 14 |
| 4.8 | REGISTERS 37 TO 39 – I ² C SLAVE 0 CONTROL | 17 |
| 4.9 | REGISTERS 40 TO 42 – I ² C SLAVE 1 CONTROL | 20 |
| 4.10 | REGISTERS 43 TO 45 – I ² C SLAVE 2 CONTROL | 20 |
| 4.11 | REGISTERS 46 TO 48 – I ² C SLAVE 3 CONTROL | 20 |
| 4.12 | REGISTERS 49 TO 53 – I ² C SLAVE 4 CONTROL | 21 |
| 4.13 | REGISTER 54 – I ² C MASTER STATUS | 23 |
| 4.14 | REGISTER 55 – INT PIN / BYPASS ENABLE CONFIGURATION | 24 |
| 4.15 | REGISTER 56 – INTERRUPT ENABLE | 26 |
| 4.16 | REGISTER 58 – INTERRUPT STATUS | 27 |
| 4.17 | REGISTERS 59 TO 64 – ACCELEROMETER MEASUREMENTS | 28 |
| 4.18 | REGISTERS 65 AND 66 – TEMPERATURE MEASUREMENT | 29 |
| 4.19 | REGISTERS 67 TO 72 – GYROSCOPE MEASUREMENTS | 30 |
| 4.20 | REGISTERS 73 TO 96 – EXTERNAL SENSOR DATA | 31 |
| 4.21 | REGISTER 99 – I ² C SLAVE 0 DATA OUT | 33 |
| 4.22 | REGISTER 100 – I ² C SLAVE 1 DATA OUT | 33 |
| 4.23 | REGISTER 101 – I ² C SLAVE 2 DATA OUT | 34 |
| 4.24 | REGISTER 102 – I ² C SLAVE 3 DATA OUT | 34 |
| 4.25 | REGISTER 103 – I ² C MASTER DELAY CONTROL | 35 |
| 4.26 | REGISTER 104 – SIGNAL PATH RESET | 36 |
| 4.27 | REGISTER 106 – USER CONTROL | 37 |
| 4.28 | REGISTER 107 – POWER MANAGEMENT 1 | 38 |
| 4.29 | REGISTER 108 – POWER MANAGEMENT 2 | 40 |

Figure 24: MPU-9150 Register Map Contents. Short description of each register functionality (<https://www.invensense.com/>).

For each one of the device registers the Register Map shows detailed information on each individual register functionality and how to write or read data to or from the register. As a simple example we will examine the registers needed in order to read temperature data. For simplicity let's assume that the device is configured correctly except the temperature sensor. So in order to receive correct output, the temperature sensor must be enabled. This can be achieved by checking the Register Map documentation for Register 107 (Figure 25).

4.28 Register 107 – Power Management 1 PWR_MGMT_1

Type: Read/Write

| Register (Hex) | Register (Decimal) | Bit7 | Bit6 | Bit5 | Bit4 | Bit3 | Bit2 | Bit1 | Bit0 |
|----------------|--------------------|--------------|-------|-------|------|----------|-------------|------|------|
| 6B | 107 | DEVICE_RESET | SLEEP | CYCLE | - | TEMP_DIS | CLKSEL[2:0] | | |

Figure 25: MPU-9150 Register 107 Schematic (<https://www.invensense.com/>).

The bus address for this sensor is 6B in Hex or equivalently 107 in Decimal. This is a Read/Write sensor. The temperature sensor configuration is achieved using bit TEMP_DIS

(Bit3). The documentation for this bit states: When set to 1, this bit disables the temperature sensor. The temperature sensor can be enabled by simply setting this bit to 0. In order not to change the existing value of the register it is advised to first read the register existing value and then apply an 'AND mask' for Bit3 (11110111). The Register Map documentation points to Registers 65 and 66 for reading temperature measurement (Figure 26).

4.18 Registers 65 and 66 – Temperature Measurement TEMP_OUT_H and TEMP_OUT_L

Type: Read Only

| Register (Hex) | Register (Decimal) | Bit7 | Bit6 | Bit5 | Bit4 | Bit3 | Bit2 | Bit1 | Bit0 |
|----------------|--------------------|----------------|------|------|------|------|------|------|------|
| 41 | 65 | TEMP_OUT[15:8] | | | | | | | |
| 42 | 66 | TEMP_OUT[7:0] | | | | | | | |

Figure 26: MPU-9150 Registers 65, 66 Schematic (<https://www.invensense.com/>).

These registers store the most recent temperature sensor measurement. This measurement is a 16-bit value so it uses two 8-bit registers to compute TEMP_OUT. Documentation states that the temperature in degrees C for a given register value may be computed as:

$$\text{Temperature in degrees C} = (\text{TEMP_OUT Register Value as a signed quantity})/340 + 35$$

Before starting using the device all registers must be set to correct settings, registers such as Sample Rate, Accelerometer & Gyroscope configuration etc. The configuration for this project is discussed in the Implementation section of the sensor node.

2.5 IMU Filtering Methods

IMUs have the disadvantage of lower positional accuracy and positional drift as mentioned above. There are ways to minimize the error accumulation with the use of an appropriate filtering method (Buonocunto & Marinoni 2014). Such methods include Complementary filter, Kalman Filter, Madgwick Filter. Although the errors are minimized, it still remains a challenge to maintain accurate angle and relocation measurements of limb motion. Especially if the rate of these movements is relatively fast.

2.5.1 Complementary Filter

These filters follow a frequency-based approach, and this is one of the first methodologies used to address these issues. The key idea is to filter one signal through a low-pass filter, the other one through a high-pass filter, and combine them to obtain the final rate. In case of IMUs, it can be better to combine slow moving signals from the accelerometer and magnetometer, and fast moving signals from the gyroscope. The result is to favor accelerometer measurements of orientation at low angular velocities and the integrated gyroscope measurements at high angular velocities. Such an approach is simple but may only be effective under limited operating conditions. There are algorithms (Bachmann et al. 2001, Mahony et al. 2008) that employ a complementary filter process, using adaptive parameters. This algorithm structure has been shown to provide a good trade-off between effective performance and computational expense (Figure 27).

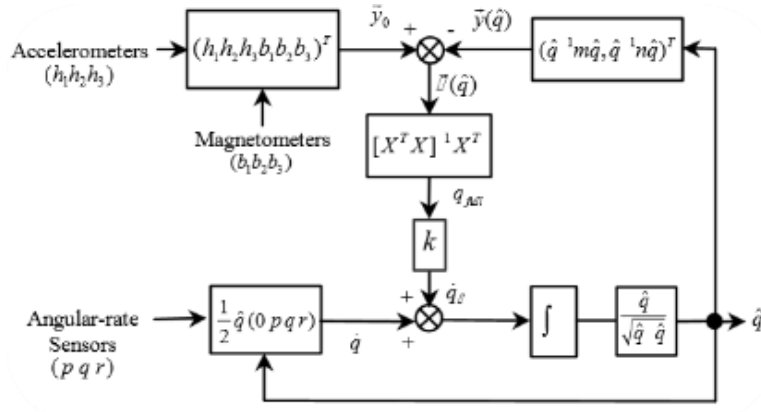


Figure 27: Quaternion-Based Complementary Orientation Filter (Mahony et al. 2008).

2.5.2 Kalman Filter

The Kalman filter (Kalman 1960, Grewal 2011) has become a basis for the majority of orientation algorithms and commercial inertial orientation sensors, like Xsens, Intersense, and many others. The widespread use of Kalman-based solutions is a guarantee of their accuracy and effectiveness. Nevertheless, the Kalman filter implementation imposes a high computational load due to a lot of recursive formulas that need to be calculated to minimize the least mean squared error. Kalman filter is the methodology used in the current implementation. Refer to *Derivation of Kalman filter parameters* section below for justification of this choice.

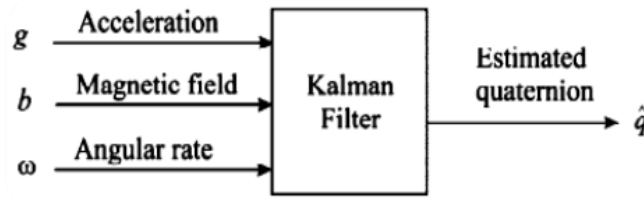


Figure 28: Simple block diagram of Kalman filter design (Grewal 2011).

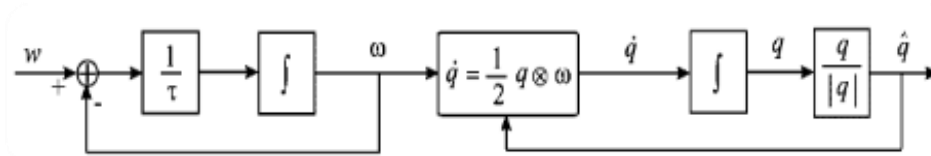


Figure 29: Kalman filter process model (Grewal 2011).

2.5.3 Madgwick Filter

The algorithm uses a quaternion representation, allowing accelerometer and magnetometer data to be used in an analytically derived and optimized gradient descent algorithm to compute the direction of the gyroscope measurement error as a quaternion derivative (Madgwick et al. 2011). The algorithm achieves levels of accuracy matching that of

Kalman-based methods, but with lower computational load, and the ability to operate at small sampling rates.

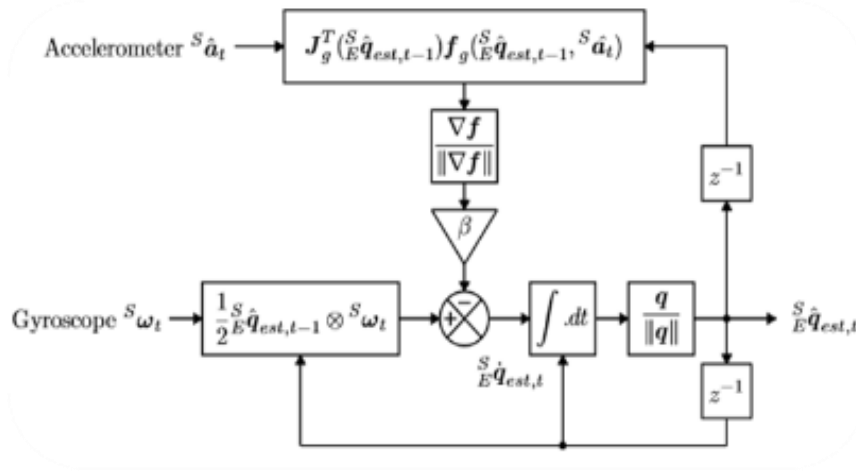


Figure 30: Block diagram of Madgwick orientation estimation algorithm for IMU (Madgwick et al. 2011).

2.5.4 Derivation of Kalman Filter Parameters

The filtering procedure in the present study is not handled in the sensor node containing the IMU. Instead it is handled by the application software. The application software runs on Android devices. These devices usually follow ARM architecture using new generation multicore CPUs that can handle higher filtering computational load. So Kalman filtering is preferred for its optimality over different methods discussed. Complementary filtering was also tested and provided very similar results to these of Kalman filtering. Below the Kalman filtering method is analyzed for Accelerometer and Gyroscope data. This analysis leads to the detailed implementation provided in the Implementation Section.

The algorithm works in a two-step process. In the prediction step, the Kalman filter produces estimates of the current state variables, along with their uncertainties. Once the outcome of the next measurement (necessarily corrupted with some amount of error, including random noise) is observed, these estimates are updated using a weighted average, with more weight being given to estimates with higher certainty (Figure 31).

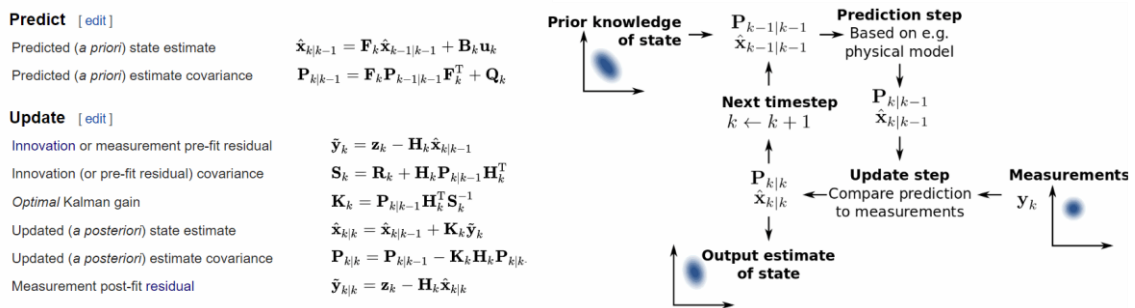


Figure 31: Kalman Algorithm Steps (<https://www.wikipedia.org/>).

The Kalman filter model assumes the true state at time k is evolved from the previous state (at $k-1$) according to:

$$\mathbf{x}_k = \mathbf{F} \mathbf{x}_{k-1} + \mathbf{B} u_k + w_k$$

Where \mathbf{x}_k is the state matrix which is given by:

$$\mathbf{x}_k = \begin{bmatrix} \theta \\ \dot{\theta}_b \end{bmatrix}_k$$

The output of the filter will be the angle θ but also the bias θ_b based upon the measurements from the accelerometer and gyroscope. The bias is the amount the gyro has drifted. This means that one can get the true rate by subtracting the bias from the gyro measurement. The next is the \mathbf{F} matrix, which is the state transition model which is applied to the previous state \mathbf{x}_{k-1} . In this case \mathbf{F} is defined as:

$$\mathbf{F} = \begin{bmatrix} 1 & -\Delta t \\ 0 & 1 \end{bmatrix}$$

Where Δt is the gyroscope constant rate which is configured at e.g. 100 Hz. This will result in a $\Delta t = 0.01 \text{ s} = 10 \text{ ms}$. Following there is the control input u_k , in this case it is the gyroscope measurement in rad (or degrees) per second ($^\circ/\text{s}$) at time k . This is denoted as the rate $\dot{\theta}$. The state equation then becomes:

$$\mathbf{x}_k = \mathbf{F} \mathbf{x}_{k-1} + \mathbf{B} \dot{\theta}_k + w_k$$

The matrix \mathbf{B} is independent of k , since it depends on the constant gyroscope rate Δt . So \mathbf{B} notation is used instead of \mathbf{B}_k . This is the control-input model, which is defined as:

$$\mathbf{B} = \begin{bmatrix} \Delta t \\ 0 \end{bmatrix}$$

The w_k vector is process noise which is Gaussian distributed with a zero mean and with covariance \mathbf{Q} to the time k :

$$w_k \sim N(0, \mathbf{Q}_k)$$

\mathbf{Q}_k is the process noise covariance matrix and in this case the covariance matrix of the state estimate of the accelerometer and bias. In this case we will consider the estimate of the bias and the accelerometer to be independent, so it's just equal to the variance of the estimate of the accelerometer and bias. The final matrix is defined as so:

$$\mathbf{Q}_k = \begin{bmatrix} Q_\theta & 0 \\ 0 & Q_{\dot{\theta}_b} \end{bmatrix} \Delta t$$

At time k an observation (or measurement) z_k of the true state \mathbf{x}_k is made according to:

$$z_k = \mathbf{H} \mathbf{x}_k + v_k$$

\mathbf{H} is the observation model and is used to map the true state space into the observed space. The true state cannot be observed. Since in this case the measurement is just the measurement from the accelerometer, \mathbf{H} is independent of k so the notion \mathbf{H}_k is dropped and is given simply by:

$$\mathbf{H} = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

The noise of the measurement have to be Gaussian distributed as well with a zero mean and \mathbf{R} as the covariance:

$$\mathbf{v}_k \sim N(0, \mathbf{R})$$

But as \mathbf{R} is not a matrix, the measurement noise is just equal to the variance of the measurement, since the covariance of the same variable is equal to the variance. Now \mathbf{R} can be defined as so:

$$\mathbf{R} = E \begin{bmatrix} v_k & v_k^T \end{bmatrix} = var(v_k)$$

Assuming that the measurement noise is the same and does not depend on the time k then the resulting \mathbf{R} becomes:

$$var(v_k) = var(v)$$

The Kalman algorithm requires the fine tuning of the noise variance parameters for \mathbf{Q}_k and \mathbf{R} . In this project an estimation of these parameters is achieved by observing the variance of the accelerometer and gyroscope signals on their resting state where their initial values are known.

Now that the accelerometer and gyroscope variables have been defined, the computation of the Kalman algorithm steps can commence. By replacing the current model parameters, the following results are derived:

- Step 1. Predicted apriori state estimate.

$$\begin{aligned} \hat{\mathbf{x}}_{k|k-1} &= \mathbf{F} \hat{\mathbf{x}}_{k-1|k-1} + \mathbf{B} \dot{\theta}_k \\ &= \begin{bmatrix} \theta + \Delta t(\dot{\theta} - \dot{\theta}_b) \\ \dot{\theta}_b \end{bmatrix} \end{aligned}$$

- Step 2. Predicted (a priori) estimate covariance.

$$\begin{aligned} \mathbf{P}_{k|k-1} &= \mathbf{F} \mathbf{P}_{k-1|k-1} \mathbf{F}^T + \mathbf{Q}_k \\ &= \begin{bmatrix} P_{00} + \Delta t(\Delta t P_{11} - P_{01} - P_{10} + Q_\theta) & P_{01} - \Delta t P_{11} \\ P_{10} - \Delta t P_{11} & P_{11} + Q_{\dot{\theta}_b} \Delta t \end{bmatrix} \end{aligned}$$

Where

$$\mathbf{P} = \begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix}$$

- Step 3. Innovation or measurement pre-fit residual.

$$\begin{aligned}\tilde{\mathbf{y}}_k &= \mathbf{z}_k - \mathbf{H}\hat{\mathbf{x}}_{k|k-1} \\ &= \mathbf{z}_k - \theta_{k|k-1}\end{aligned}$$

- Step 4. Innovation (or pre-fit residual) covariance.

$$\begin{aligned}\mathbf{S}_k &= \mathbf{H}\mathbf{P}_{k|k-1}\mathbf{H}^T + \mathbf{R} \\ &= P_{00k|k-1} + \text{var}(v)\end{aligned}$$

- Step 5. Optimal Kalman gain.

$$\begin{aligned}\mathbf{K}_k &= \mathbf{P}_{k|k-1}\mathbf{H}^T\mathbf{S}_k^{-1} \\ &= \frac{\begin{bmatrix} P_{00} \\ P_{10} \end{bmatrix}_{k|k-1}}{\mathbf{S}_k}\end{aligned}$$

Where,

$$\mathbf{K} = \begin{bmatrix} K_0 \\ K_1 \end{bmatrix}$$

- Step 6. Updated (a posteriori) state estimate.

$$\begin{aligned}\hat{\mathbf{x}}_{k|k} &= \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k \tilde{\mathbf{y}}_k \\ &= \begin{bmatrix} \theta \\ \dot{\theta}_b \end{bmatrix}_{k|k-1} + \begin{bmatrix} K_0 \tilde{\mathbf{y}} \\ K_1 \tilde{\mathbf{y}} \end{bmatrix}_k\end{aligned}$$

- Step 7. Updated (a posteriori) estimate covariance.

$$\begin{aligned}\mathbf{P}_{k|k} &= (\mathbf{I} - \mathbf{K}_k\mathbf{H})\mathbf{P}_{k|k-1} \\ &= \begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix}_{k|k-1} - \begin{bmatrix} K_0 P_{00} & K_0 P_{01} \\ K_1 P_{00} & K_1 P_{01} \end{bmatrix}\end{aligned}$$

2.6 Embedding the IMU

2.6.1 Single-board Computer vs Microcontroller

Controlling any sensor requires the connection of this sensor to either a microcontroller or single-board computer (SBC) configured as a microcontroller. A microcontroller (or MCU for microcontroller unit) is a small computer on a single integrated circuit. Popular MCU examples are Arduino products. On the other hand a single-board computer is a complete computer built on a single circuit board, with microprocessor(s),

memory, input/output (I/O) and other features required of a functional computer. Popular SBC examples are the Raspberry Pi series.

For reading/writing sensor data MCUs are preferred due to lower power consumption. On the other hand SBCs have the advantage of the Operating System drivers support in order to customize and configure functionalities like I²C and networking in a simple manner without the need of external Bluetooth shields like in Arduino. This project will not focus on the low level consumption hardware design. It will focus on simplicity of hardware and the software application framework that should be independent of the sensor hardware. It follows that the software application should work with any hardware implementation if data are received in a consistent manner. Simplicity and Bluetooth connectivity are the main reasons we have chosen Raspberry Pi for this project in order to create a prototype node. There is also the option of Andruino BT which is still costly at the time of writing, about 5 times more than the Bluetooth equivalent of Raspberry Pi, model Zero W.

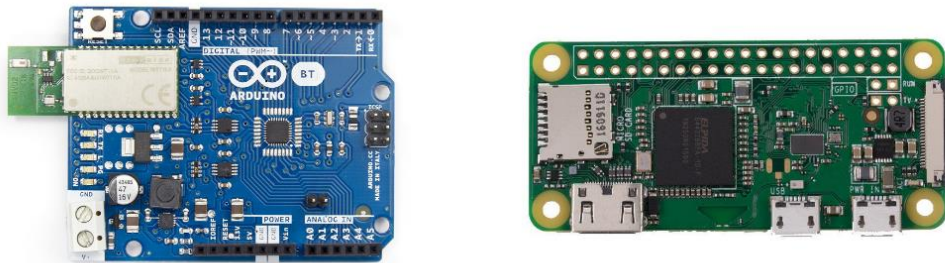


Figure 32: Left, Arduino BT (<https://www.arduino.cc/>). Right, Raspberry Pi Zero W (<https://www.raspberrypi.org/>).

2.6.2 Raspberry Pi Overview

The Raspberry Pi is a series of small single-board computers developed in the United Kingdom by the Raspberry Pi Foundation to promote the teaching of basic computer science in schools and in developing countries. Raspberry Pi series consist of several models with different specifications. For use in an embedded project the Raspberry Pi Zero family is ideal due to its smaller size and lower power consumption. On 28 February 2017, the Raspberry Pi Zero W was launched, which is like Raspberry Pi Zero with Wi-Fi and Bluetooth, for US\$10. This is the specific Raspberry Pi model used in the current thesis for the hardware node implementation. The Raspberry Pi Zero W extends the Pi Zero family. The Pi Zero W has all the functionality of the original Pi Zero, but comes with added connectivity, consisting of:

- 802.11 b/g/n wireless LAN.
- Bluetooth 4.1.
- Bluetooth Low Energy (BLE).

For the current project, this is the ideal candidate since the sensor data are to be transferred in the application through a Bluetooth Connection. Like the Pi Zero, it also has:

- 1GHz, single-core CPU.

- 512MB RAM.
- Mini HDMI and USB On-The-Go ports.
- Micro USB power.
- HAT-compatible 40-pin header.
- Composite video and reset headers.
- CSI camera connector.

An external sensor like an IMU can be attached on the 40-pin header provided by the Raspberry Pi for General Purpose Input and Output (GPIO) (Figure 33). This is the case in the current project as the IMU is connected to 4 pins of the GPIO using the I²C interface as described in detail in the Hardware Node Implementation section.

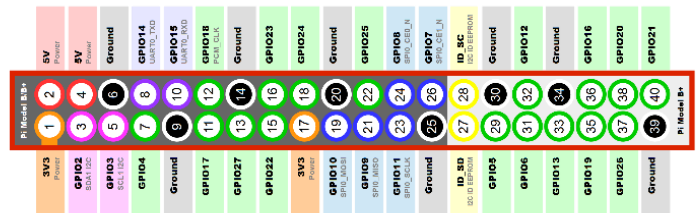


Figure 33: Raspberry Pi 40 GPIO pin header (<https://www.raspberrypi.org/>).

2.6.3 Configuring Raspberry Pi as a Microcontroller

By default Raspberry boots in the Linux OS installed in desktop mode. The default Linux distribution on Raspberry Pi is Raspbian. Running in desktop mode consumes unnecessary resources and is not suitable for embedded development. This behavior can change using the command `raspi-config`. This command starts the configuration utility of Raspberry Pi. There, under Boot Option there is the option of Automatic login in Console mode (Figure 34). This has 2 benefits. Using only the Console and allowing for the Raspberry Pi to boot automatically without using any credentials. This way a custom made script can run automatically on boot, as described further in this section.

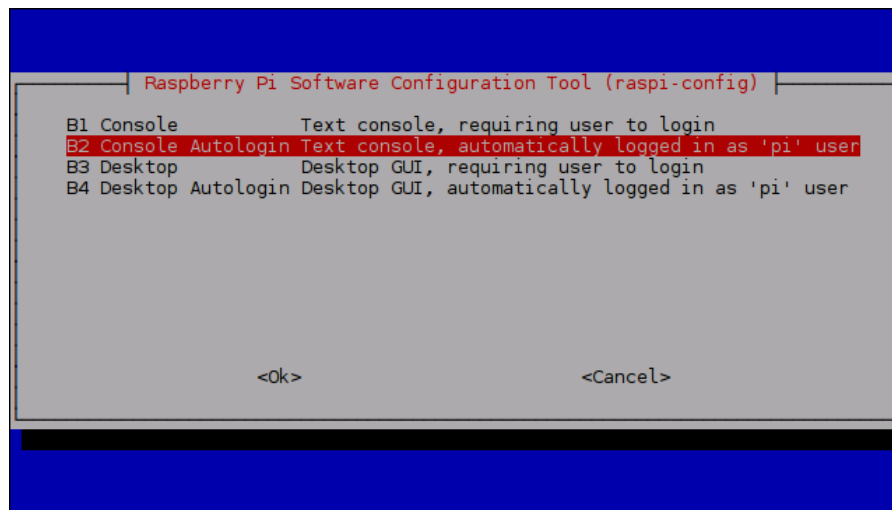


Figure 34: Setting Console Autologin through Boot Options in raspi-config.

A useful option for development is to also change the default layout for the keyboard to match the one used. Raspbian doesn't detect localization settings automatically like many desktop Operating Systems do. The user must change the Localisation Option using raspi-config to match the keyboard layout in order for the keyboard input to be correct (Figure 35).

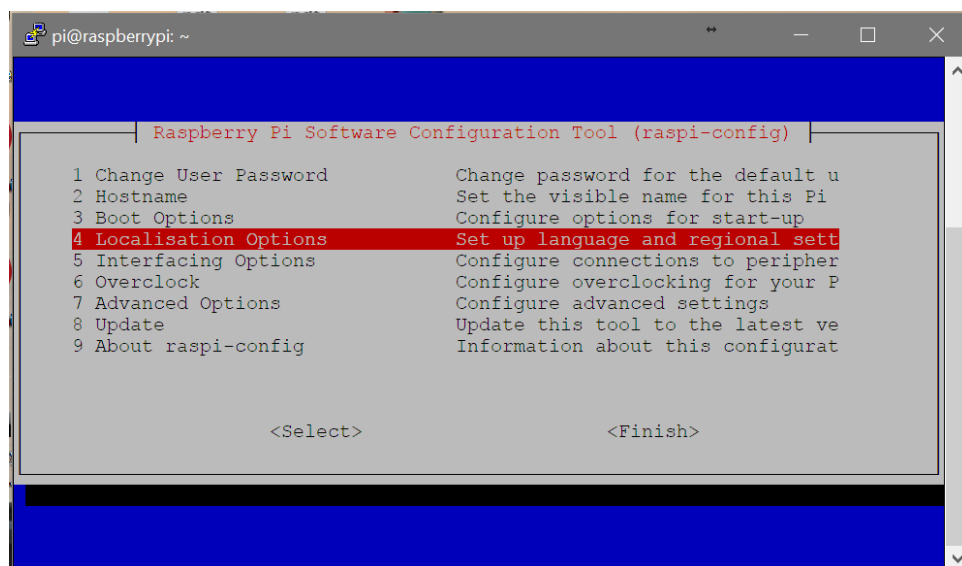


Figure 35: Main Screen in raspi-config. Localisation Options are selected.

The next important setting is to enable I²C in order to communicate with the sensor using this protocol. This can be achieved through Interfacing Options (Figure 36).

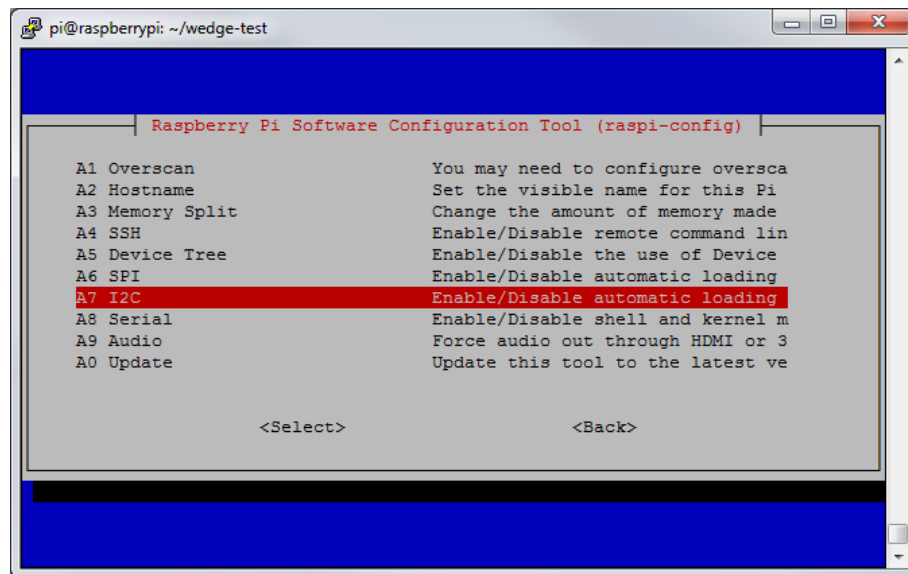


Figure 36: Enable I²C.

These are the necessary configuration through the `raspi-config` interface. The next important thing is to configure Bluetooth and Wi-Fi as needed. Both should be enabled by default. In the current application however, Wi-Fi is not needed. It consumes resources and depletes the battery faster. So it must be disabled. This can be done by editing `/etc/modprobe.d/raspi-blacklist.conf` and adding the following lines in order to blacklist the wifi.

```
#wifi
blacklist brcmfmac
blacklist brcmutil
```

This way should suffice although there are other techniques like editing `config.txt` or removing `wpa-conf` from `interfaces.d`.

Finally to fully automatize the process, the Raspberry Pi can run a custom bash script upon start up. There are again several ways to achieve this. One of them is to use the file `/home/pi/.bashrc` and append the script path in the end of the file. The script path can be of the following form: `/home/pi/MyCustomPath/MyCustomScript.sh`. In order for a custom script to automatically start it must be an executable. This can be done by changing its permission with the following command:

```
sudo chmod +x /home/pi/MyCustomPath/MyCustomScript.sh
```

At this point the basic configuration for using raspberry pi as a microcontroller is completed. This process will later be used in the implementation section in order to connect the MPU-9150 to the Raspberry Pi.

3 Chapter 3 – Overview of IDE & Development Tools

3.1 Game Development

With the advance of technology and increase in processing power, the development of more complex games became a reality. At first rendering engines emerged that focused on projecting 3D models on the screen. The use of a graphics processing unit (GPU) enabled very efficient manipulation of computer graphics. Modern GPUs highly parallel structure make them more effective than general-purpose CPUs for algorithms where processing of large blocks of data is done in parallel. Besides graphic engines a game nowadays needs more functional components such as sound engine, physics engine, etc. All these subsystems of game functionality (often called middleware) are the components of a system called game engine (Eberly 2006, Gregory 2009). Game engines are used primarily for developing next generation games by allowing the developers to focus on higher level aspects of a game (interaction between objects) instead of building a game from scratch as in earlier eras (the game had to be designed from the bottom up to make optimal use of the display hardware). As game engine technology matures and becomes more user-friendly, the application of game engines has broadened in scope. They are now being used for serious games: visualization, training, medical, and simulation applications. To facilitate this accessibility, new hardware platforms are now being targeted by game engines, including mobile phones and web browsers.

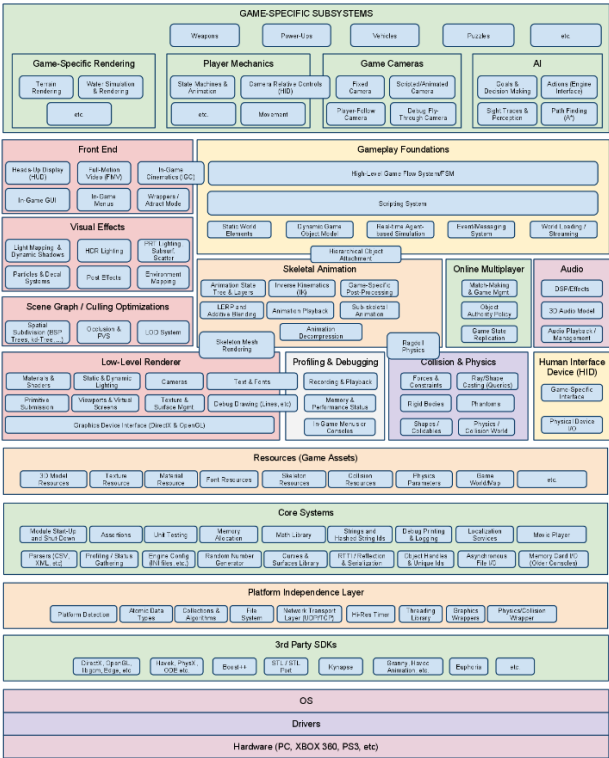


Figure 37: Game Engine Architecture (Gregory 2009).

Often, game engines are designed with a component-based architecture (Figure 37). Component-based software engineering (CBSE), also called as component-based development (CBD), is a branch of software engineering that emphasizes the separation of concerns with respect to the wide-ranging functionality available throughout a given software system. It is a reuse-based approach to defining, implementing and composing loosely coupled independent components into systems. At the game engine context, the core engine components include a rendering engine (renderer) for 2D or 3D graphics, a physics engine or collision detection (and collision response), sound, scripting, animation, artificial intelligence, networking, streaming, memory management, threading, localization support, scene graph, and may include video support for cinematics.

The end user of the game engine can focus on the topmost level components of this architecture which include game logic aspects such as interaction of 3D Entities and AI Intelligence. This fact simplifies the development process. Often though it is important to understand the functionality of lower level middleware components, such as the rendering procedure. An end user doesn't have to be concerned exactly how the Rendering process occurs. Often though it is important to understand underlying mechanisms in order to configure the respective subsystem in an optimized way. So in the rendering component, the user may have to choose between different rendering APIs (Direct3D or OpenGL), different render techniques (Forward rendering, Light pre-pass rendering or Deferred rendering), and even game logic decisions that help in rendering optimizations (Object Occlusion & Frustum Culling, Camera Clipping).

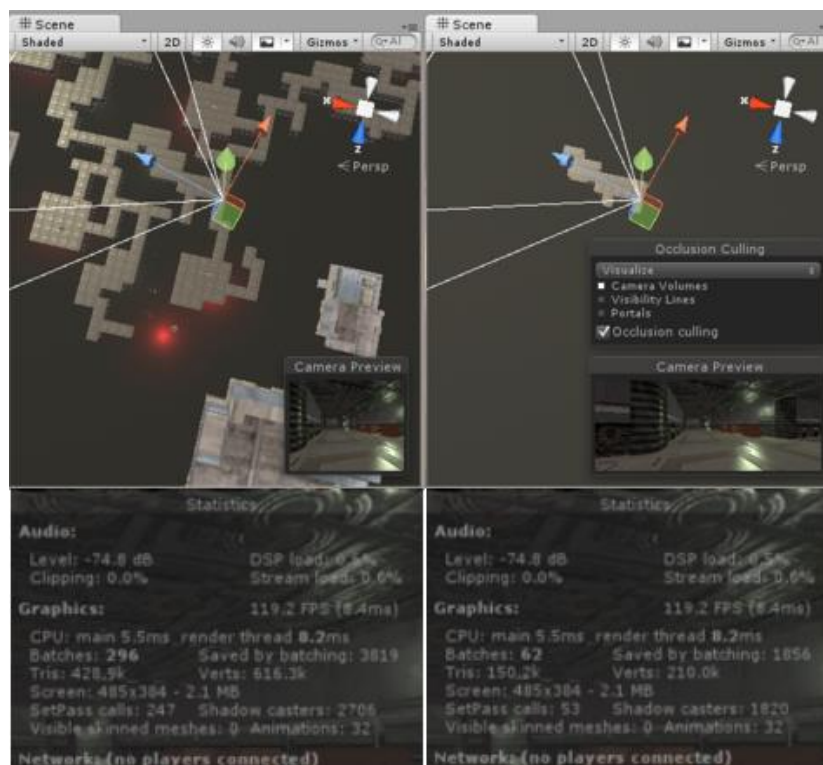


Figure 38: Rendering Optimization Example. Left, No Culling. Right, Occlusion & Frustum Culling (<https://unity3d.com/>).

3.2 Game Engine Systems

Many game engines have emerged over the past years giving the developer a wide range to choose from. Main criteria for the current project is ease of use and community support. This limits our search to popular commercial game engines mostly. These engines have large communities and can provide extended support in the development process. Also they usually include sophisticated Editors that simplify the Level Design process of a game. Two popular choices on this aspect are Unity and Unreal game engines. We will briefly summarize each one of them below. Although commercial engines provide ease of use, it is worth supporting and contributing to less popular middleware engine frameworks for custom projects. These engine frameworks may need a lot more work to be done to reach the quality of popular frameworks but can achieve similar results under a free software license instead of using a commercial license. One such engine is Urho 3D which is a free lightweight, cross-platform 2D and 3D game engine implemented in C++ and released under the MIT license and is greatly inspired by OGRE and Horde3D (Figure 39). For the current research project, a more restrictive license can also be used in order to account for the tradeoff between development time and license requirements.

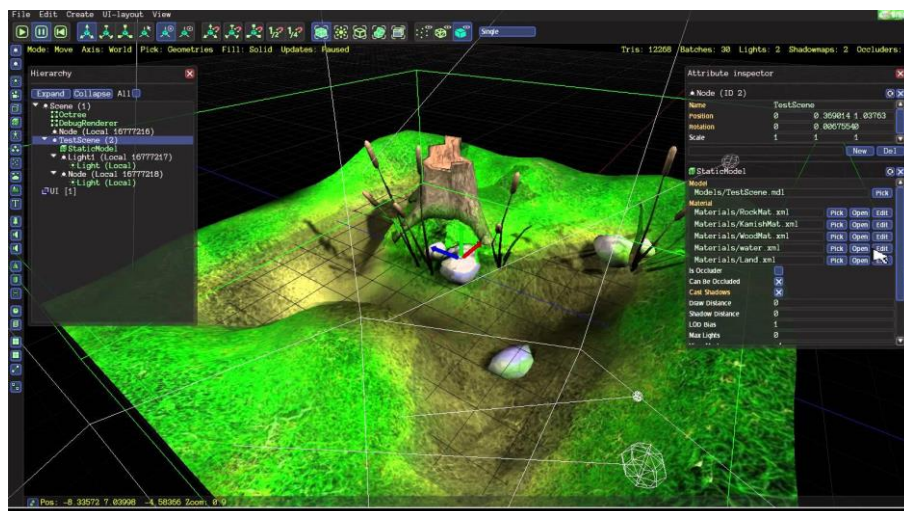


Figure 39: Urho 3D Simple Editor Interface (<https://urho3d.github.io/>).

3.2.1 Unreal Engine Overview

Unreal is one of the most popular game engines to develop high-end triple-A titles for years now. The Unreal Engine is developed by Epic Games and was first released in 1998. Although primarily developed for first-person shooters, it has been successfully used in a variety of other genres, including stealth, MMORPGS and other RPGs. With its code written in C++, the Unreal Engine features a high degree of portability and is a tool used by many game developers today. Its blueprint system in the newer version makes scripting almost non-existent as it gives the developers the ability to create game logic by simply connecting lines and blocks of commands. Finally, it has its own version of an asset store providing its users with content both premium and free to add into their projects.

Advantages:

- Built-in beginner solution with Blueprint.
- Multi-platform export including consoles.
- Outstanding next-gen graphics.
- Good online resources.
- Free to use until a game starts making a profit.

Disadvantages:

- Heavy and demanding on performance.
- Steep learning curve.
- Marketplace not as full as Unity's.

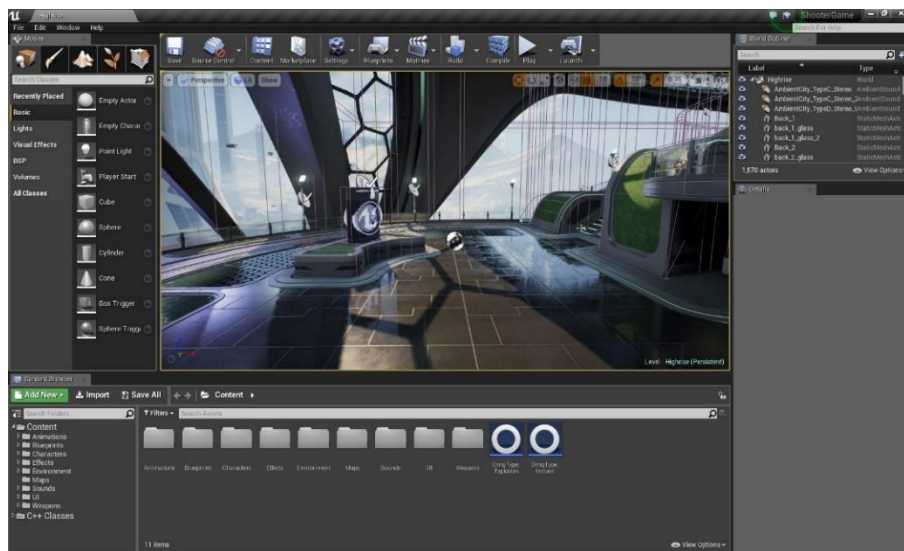


Figure 40: Unreal Engine Sophisticated Editor Interface (<https://www.unrealengine.com/>).

3.2.2 Unity 3D 5.6 Overview

Unity is a cross-platform game engine developed by Unity Technologies, which is primarily used to develop video games and simulations for computers, consoles and mobile devices. Unity is marketed to be a multiple purpose engine, and as a result supports both 2D and 3D graphics, drag and drop functionality and scripting through its 3 custom languages C#, Boo and UnityScript (basically JavaScript with type annotations). The Unity's development kit community has widely adopted C# and the majority of plugins and examples use it. It also provides cross platform publishing, millions of ready-made assets in the Asset Store and a growing online community. For individual developers and studios, Unity's environment reduces the time and cost allowing its users to develop advanced interactive games. It provides flexibility to deploy projects on multi-platforms like iOS, Windows and Android.

Advantages:

- Strong community of asset and plugin creators.
- Excellent resources and tutorials.

- Produces high quality results without any complex configuration.
- Free license provides the majority of features.

Disadvantages:

- Can be slower in terms of performance compared to Unreal Engine 4.

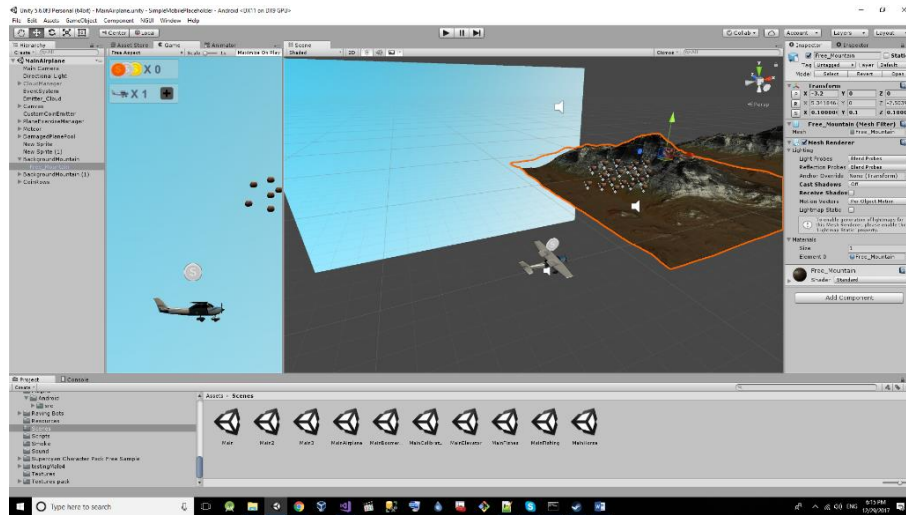


Figure 41: Scene in Unity Editor (<https://unity3d.com/>).

3.2.3 Comments

Both engines offer great support and quality results. Which one is going to be utilized is a matter of personal preference. The choice for this project will be Unity as it has less steep learning curve than Unreal Engine. It is also a choice of programming language preference. C++ can be faster in terms of performance, but the developer must handle memory management explicitly. C# on the other hand utilizes an automatic Garbage Collector that handles memory management implicitly. This may not be however the ideal situation. When an object is marked to be garbage collected there is not any guarantee if and when the garbage collector will be executed. Therefore it is not known when the reserved memory will be released in contrast with C++ where memory is immediately released when explicitly stated. The developer should employ best practices in order to improve performance and minimize created garbage and allocations in the memory heap. So by working with C# instead with C++ we sacrifice performance for ease of use. For the current project this is acceptable. The serious game created has minimal requirements and can work quite efficiently in current and outdated Android devices.

3.3 Unity Gameplay Creation

3.3.1 Scenes

Scenes contain the environments and menus of the game. Each unique Scene file corresponds to a unique gameplay level. A Scene consists of multiple GameObject, which create the environment of a virtual scene along with the behavior and interactions between them. Unity Editor provides a very friendly way of viewing and transforming a GameObject inside a scene (Figure 41). Note that Unity Editor is not an asset creation software. It imports

and utilizes assets that are created from within a 3D creation software, for example Blender 3D or 3D Studio Max.

3.3.2 GameObject

Every object in a Unity game is a GameObject, from characters and collectible items to lights, cameras and special effects. However, a GameObject can't do anything on its own. The user needs to specify its properties before it can become a character, an environment, or a special effect. GameObjects are the fundamental objects in Unity that represent characters, collectible items, lights, cameras, special effects and scenery. They do not accomplish much by themselves but they act as containers for Components, which implement the real functionality.

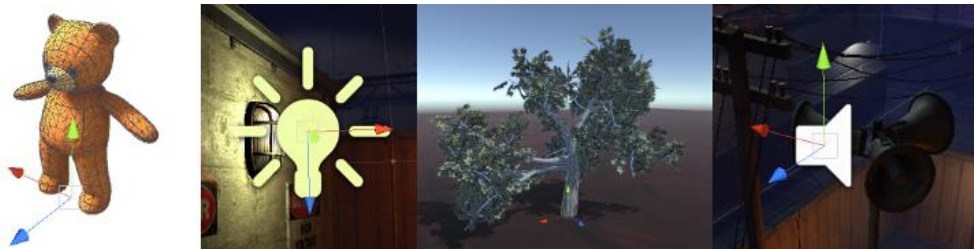


Figure 42: Four different types of GameObject: an animated character, a light, a tree, and an audio source (<https://unity3d.com/>).

3.3.3 Components

A GameObject contains components. Components control the behaviors of a GameObject in a game. They are the functional pieces of every GameObject. A component connects a GameObject with the functionality of each individual game engine subsystem. For example, if we want a 3D mesh to be rendered we need to add 2 components to our game object:

- A mesh filter component to hold the mesh data for the 3D model.
- A mesh renderer component to render the mesh data in the scene.

Similarly to the Rendering functionality a Component can be used to provide Physics, Audio, Video, Network functionality to a GameObject or even yet our custom game logic functionality through script components. Unity provides a variety of components to choose from (Figure 43). Each one of these components is related to the subsystems that are on the middle layers of the game architecture as described above.

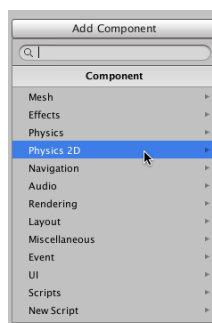


Figure 43: Component browser (<https://unity3d.com/>).

Each GameObject by default has the transform component which specifies the physical presence of an object in 3D space using translation, rotation and scale transformations. Even when an object is purely functional and doesn't have a model connected to it (so it is not visible in the scene), it has the transform component in 3D space.

3.3.4 Assets

An asset is a representation of any item that can be used in a game or project. An asset may come from a file created outside of Unity, such as a 3D model, 3D animation, an audio file, an image, or any of the other types of files that Unity supports. There are also some asset types that can be created within Unity, such as an Animator Controller, an Audio Mixer or a Render Texture.

The Unity Asset Store is home to a growing library of free and commercial assets created both by Unity Technologies and also members of the community (Figure 44). A wide variety of assets is available, covering everything from textures, models and animations to whole project examples, tutorials and editor extensions. The assets are accessed from a simple interface built into the Unity Editor and are downloaded and imported directly into a working project.

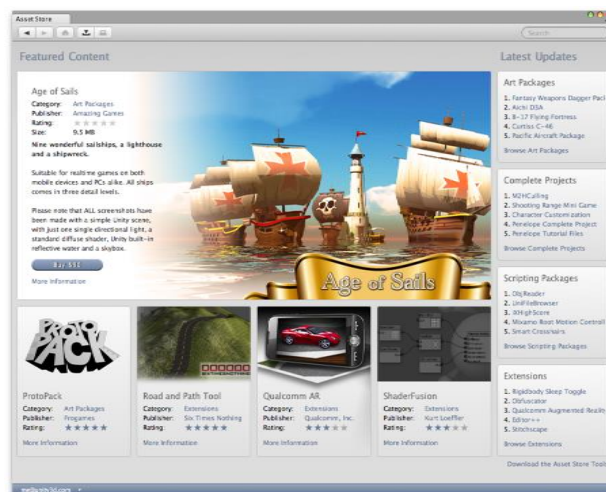


Figure 44: Unity Asset store (<https://unity3d.com/>).

3.3.5 Script Life Cycle

Unity allows custom Components creation using scripts. These allow to trigger game events, modify Component properties over time and respond to user input. The initial contents of a newly created script file will look something like this:

```
using UnityEngine;

namespace Assets.Scripts
{
    public class MainCamera : MonoBehaviour {

        // Use this for initialization
        void Start () {
```

```

}

// Update is called once per frame
void Update () {

}

}
}

```

It is important to note that a Unity Script extends the MonoBehaviour class. MonoBehaviour is the base class from which every Unity script derives. This class exposes very specific event functions. In Unity scripting, there are a number of event functions that get executed in a predetermined order as a script executes. In the example above there are 2 event functions declared with empty implementation. When the scene loads a GameObject which has the above Component, it will execute these functions in a specific order. This execution order can be seen in Figure 45.

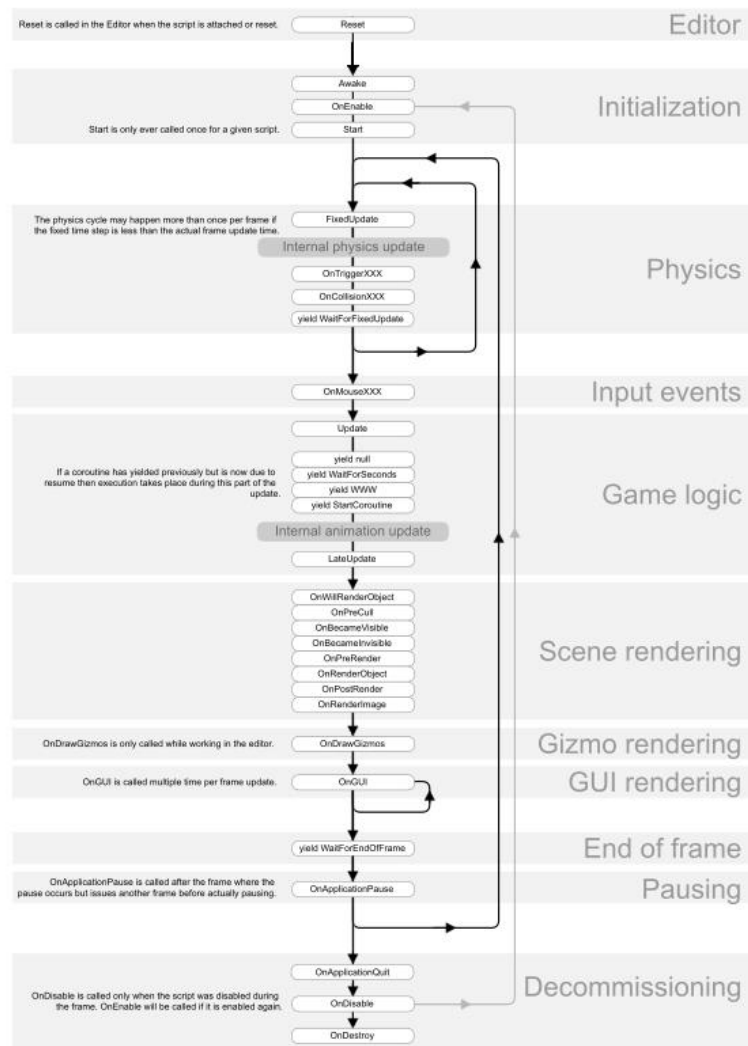


Figure 45: Execution Order of Event Functions (<https://unity3d.com/>).

Below common events are described which are used frequently during implementation of a game logic component:

- **First Scene Load.** These functions get called when a scene starts (once for each object in the scene).
 - *Awake:* This function is always called before any Start functions and also just after a prefab is instantiated. If a GameObject is inactive during start up Awake is not called until it is made active.
 - *OnEnable:* Only called if the Object is active. This function is called just after the object is enabled. This happens when a MonoBehaviour instance is created, such as when a level is loaded or a GameObject with the script component is instantiated.
 - *OnLevelWasLoaded:* This function is executed to inform the game that a new level has been loaded.
- **Before the first frame update.**
 - *Start:* Start is called before the first frame update only if the script instance is enabled.
- **Update Order.** When keeping track of game logic and interactions, animations, camera positions, etc., there are a few different events available. The common pattern is to perform most tasks inside the Update function, but there are also other functions that can be utilized.
 - *FixedUpdate:* FixedUpdate is often called more frequently than Update. It can be called multiple times per frame, if the frame rate is low and it may not be called between frames at all if the frame rate is high. All physics calculations and updates occur immediately after FixedUpdate. FixedUpdate is called on a reliable timer, independent of the frame rate.
 - *Update:* Update is called once per frame. It is the main workhorse function for frame updates.
 - *LateUpdate:* LateUpdate is called once per frame, after Update has finished. Any calculations that are performed in Update will have completed when LateUpdate begins.
 -
- **When the Object is destroyed.**
 - *OnDestroy:* This function is called after all frame updates for the last frame of the object's existence (the object might be destroyed in response to *Object.Destroy* or at the closure of a scene).
- **On Application Quit.** These functions get called on all the active objects in a scene.
 - *OnApplicationQuit:* This function is called on all game objects before the application is quit. In the editor it is called when the user stops play mode.
 - *OnDisable:* This function is called when the behavior becomes disabled or inactive.

These are the main events a developer must utilize in order to create custom behaviors or override existing ones.

3.3.6 Unity Singleton

In programming Singleton is a commonly used design pattern. A software design pattern is a general, reusable solution to a commonly occurring problem within a given context in software design. Design patterns are formalized best practices that the programmer can use to solve common problems when designing an application or system. The singleton pattern is a software design pattern that restricts the instantiation of a class to one object. This is useful when exactly one object is needed to coordinate actions across the system. In Unity there isn't a concept of global variables (global GameObject). A GameObject has Scene scope. When another Scene is loaded all GameObjects are destroyed. Unity however offer the DontDestroyOnLoad method which enables a GameObject to survive in consecutive scenes. Using this method a Unity Singleton can be implemented. Below is an example of a Singleton implementation:

```
using System.Collections.Generic;
using UnityEngine;

//Marks any GameObject it is attached to as a Singleton
//Game Objects with same name are considered the same
public class MySingleton : MonoBehaviour {

    //Use a custom hash of objects to check if object already exists
    //Making it static readonly, compiler ensures it will be initialized
    //in the beginning of the app in a non lazy manner
    private static readonly HashSet<string> Objs = new HashSet<string>();

    //Singleton should be the only method to use Awake
    private void Awake()
    {
        //If object is already created then Destroy new copy
        //If it is first time to create this add to hash
        //and instruct DontDestroyOnLoad to keep it alive
        //through different scenes
        if (Objs.Contains(gameObject.name))
        {
            Destroy(gameObject);
        }
        else
        {
            Objs.Add(gameObject.name);
            DontDestroyOnLoad(gameObject);
        }
    }

    //Helper method to check if object exists
    //It should only be used on beginning of Awake of a Singleton
    //GameObject in case that GameObject wants to use Awake
    public static bool IsCached(string name)
    {
        return Objs != null && Objs.Contains(name);
    }
}
```

3.3.7 Unity Profiler

The Unity Profiler Window helps in optimizing a Unity application (in general a game). It reports how much time is spent in the various areas of a game. For example, it can report the percentage of time spent rendering, animating or in game logic. It provides performance analysis of the GPU, CPU, memory, rendering, and audio. As mentioned previously C# uses automatic garbage collection for memory management.

The profiler is a very useful tool to find heap allocations (Figure 46). Then it is up to the programmer to apply best practices and strategies in order to reduce the impact of garbage collection in the game:

- Organize the code to in order to have fewer heap allocations and fewer object references. Fewer objects on the heap and fewer references to examine means that when garbage collection is triggered, it takes less time to run.
- Reduce the frequency of heap allocations and deallocations, particularly at performance-critical times. Fewer allocations and deallocations means fewer occasions that trigger garbage collection. This also reduces risk of heap fragmentation.
- Attempt to time garbage collection and heap expansion so that they happen at predictable and convenient times. This is a more difficult and less reliable approach, but when used as part of an overall memory management strategy can reduce the impact of garbage collection.

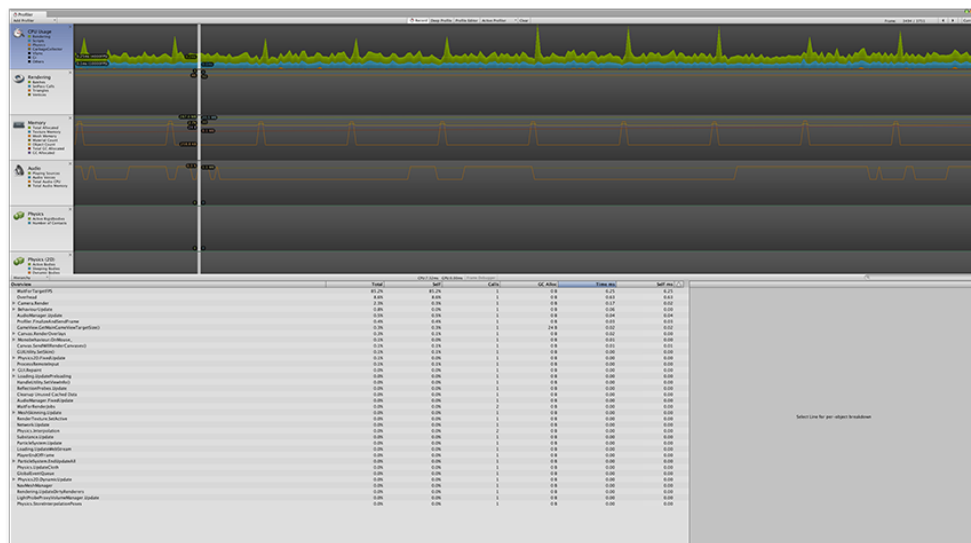


Figure 46: Using Unity profiler to track heap allocations (<https://unity3d.com/>).

3.4 Combining Unity & Android

3.4.1 Unity & Android Limitations

Unity is a cross platform game engine. It supports multiple environments including PC, Mac, Linux, Android, iOS, WebGL, PS4, Xbox One. Sometimes though there is a functionality that is platform specific. This is the case in this project for Android. Bluetooth communication with a custom sensor must be handled explicitly using a plugin specifically targeting the

Android platform. The incoming Bluetooth connection cannot be handled in a cross platform manner. Bluetooth data must first be received with native Android development using a Java Activity. So there are 2 requirements:

- Run a native java activity as a background process in the Android device with the use of a custom plugin.
- Pass the Bluetooth sensor data received by the Java activity to Unity application and handle it using C#.

Fortunately, Unity provides functionality for Building and using plugins for Android. In order to implement such a plugin the Android Studio is used.

3.4.2 Android Studio

Android Studio is the official integrated development environment (IDE) for Google's Android operating system, built on JetBrains' IntelliJ IDEA software and designed specifically for Android development. It is a replacement for the Eclipse Android Development Tools (ADT) as the primary IDE for native Android application development.

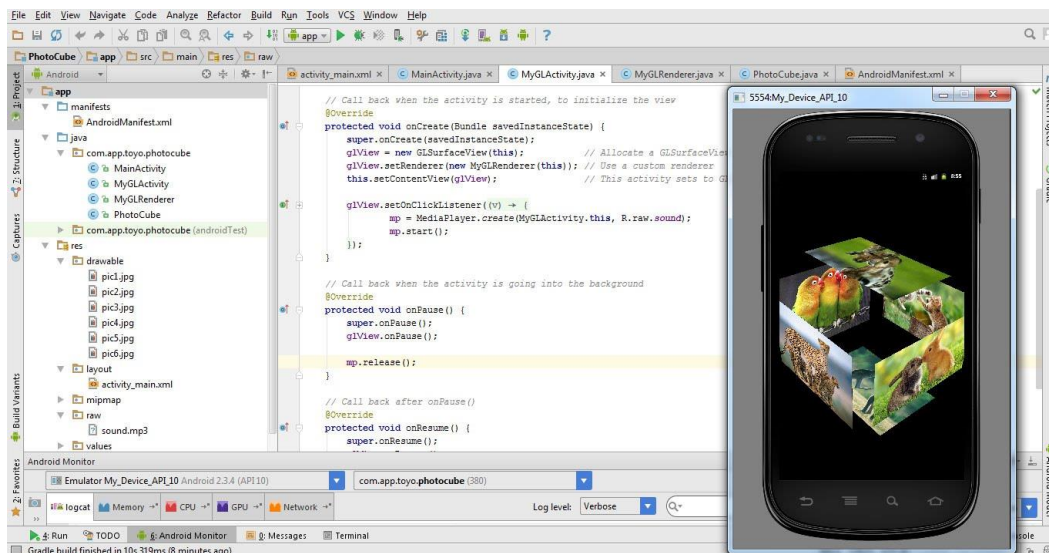


Figure 47: Android Studio Workspace (<https://developer.android.com/>).

Android Studio is not needed by Unity to build an application in Android and create the necessary plugins. The process only requires the installation of Android SDK which is included in Android Studio. However it is a powerful tool for debugging and monitoring the data collected from the sensor. Another requirement is the specification of the minimum Android API used. This guarantees backward compatibility with Android devices that use the minimum version. Here the tradeoff for a commercial application is number of devices that can install the application versus number of features supported. The latest version of Android may not be used in the majority of the population but contains many new features. In our case an older minimum API version of 4.2 (JellyBean) is used that provides a good balance between number of devices and features supported (Figure 48).

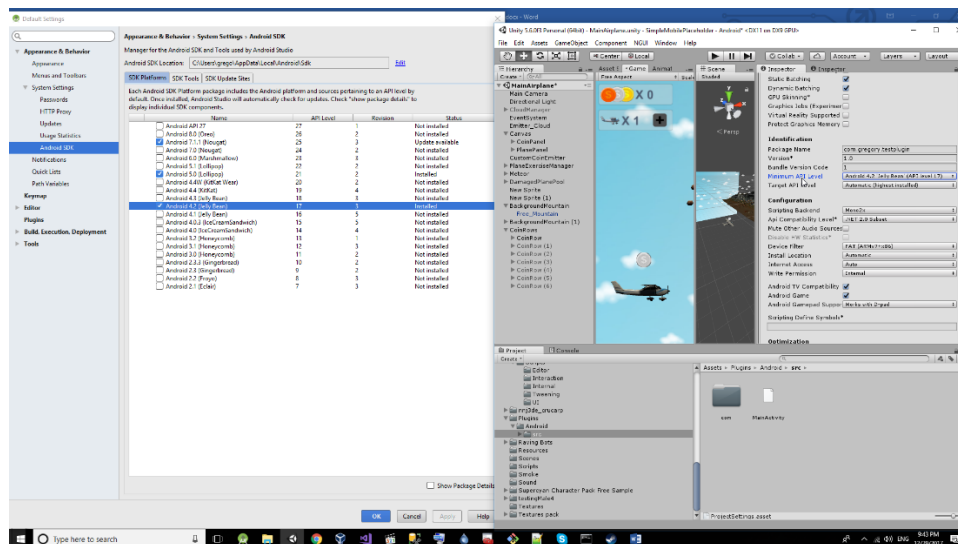


Figure 48: Left, Android Studio SDK Manager Installed APIs (<https://developer.android.com/>). Right, Unity minimum API defined as 4.2 (Jelly Bean).

This basic setup allows building Unity projects in Android. The next step would be the creation of a sample plugin with Android Studio.

3.4.3 Creating Sample Plugin

The goal is the development of a java activity that will communicate with Unity. When developing a Unity Android application, it is possible to extend the standard `UnityPlayerActivity` class (the primary Java class for the Unity Player on Android, similar to `AppController.mm` on Unity iOS) by using plugins. An application can override any or all of the basic interaction between the Android OS and the Unity Android application. Two steps are required to override the default activity:

- Create the new Activity which derives from `UnityPlayerActivity`.
- Modify the Android Manifest to have the new activity as the application's entry point.

The following is an example `UnityPlayerActivity` file:

```
package com.company.product;
import android.content.Context;
import com.unity3d.player.UnityPlayer;
import com.unity3d.player.UnityPlayerActivity;
import android.os.Bundle;
import android.util.Log;

public class OverrideExample extends UnityPlayerActivity {
    protected void onCreate(Bundle savedInstanceState) {
        // call UnityPlayerActivity.onCreate()
        super.onCreate(savedInstanceState);
        // print debug message to logcat
        Log.d("OverrideActivity", "onCreate called!");
        //Send actual data from Java to Unity.
        //Send them to the script MyUnityScript
        //which will call the function MyUnityFunction
    }
}
```

```
//with String arguments "Hello World!"  
//MyUnityFunction should be defined as follows  
//public void MyUnityFunction(string someStr) { };  
UnityPlayer.UnitySendMessage("MyUnityScript", "MyUnityFunction", "Hello World!");  
}  
}
```

The data are sent to Unity with the `UnitySendMessage` function. This is necessary for implementing the Bluetooth plugin later on. More info is provided in the Implementation Section. The following is an example AndroidManifest.xml file to initialize the implemented activity in the background:

```
<?xml version="1.0" encoding="utf-8"?>  
<manifest xmlns:android="http://schemas.android.com/apk/res/android"  
package="com.company.product">  
  <application android:icon="@drawable/app_icon" android:label="@string/app_name">  
    <activity android:name=".OverrideExample" android:label="@string/app_name"  
android:configChanges="fontScale|keyboard|keyboardHidden|locale|mnc|mcc|navigation|orientation  
|screenLayout|screenSize|smallestScreenSize|uiMode|touchscreen">  
      <intent-filter>  
        <action android:name="android.intent.action.MAIN" />  
        <category android:name="android.intent.category.LAUNCHER" />  
      </intent-filter>  
    </activity>  
  </application>  
</manifest>
```


4 Chapter 4 – Design

The work presented in this thesis puts forward the design and implementation of a custom-made ultra-portable, mobile and low cost 3D rehabilitation application intended for patients that underwent Total Knee Replacement (TKR) using only an Android mobile device and a small sensor placed on the patient's limb to track movement. The first weeks following knee surgery are crucial so that the Range of Motion (ROM) of the operated knee is deemed fully operational. If the patient fails to perform the exercises appointed by the physiotherapist during this recovery period, an, otherwise, technically accurate operation might result in poor functional outcome leading to reduced quality of life. The aim of our gamified application is to motivate the patient to exercise efficiently by providing feedback, while the physiotherapy exercises are performed in any setting, e.g. clinical, at home, indoors, outdoors or even in public areas.

This thesis focuses on the description of the rehabilitation system involving the hardware sensor, the implementation of the gamified 3D environment, as well as the initial testing of the software framework in the hospital, while patients are undergoing physiotherapy treatment. The scope of this project involves the development of an integrated ultra-portable gamified platform, including gamified tasks to be utilized for rehabilitation exercises commonly performed after TKR surgery (Figure 1). The main goal is to improve compliance to the physiotherapy protocol, increase patient engagement, monitor physiological conditions and provide feedback based on rewards via a gamified experience.

4.1 Application Functionality

An Inertial Measurement Unit (IMU) node was utilized worn by the patient recognizing limb rotation and acceleration. It is challenging to identify whether the proposed application classifies the exercises reliably utilizing just a single sensor node. Providing gamified feedback to the patient at home or in other locations in relation to performance using widely available mobile devices, is also challenging, minimizing the need for expensive physiotherapy under supervision, resulting in more engaging and accessible rehabilitation.

4.1.1 Application Procedure

The IMU is fitted at a specified limb location depending on the exercise performed. The session starts with the person in a neutral pose ready to perform one of the predetermined exercises. The raw data collected from the IMU is sent via Bluetooth to a mobile device. The application computes an orientation measurement with the use of Kalman filtering. Real-time visualization on a mobile device offers feedback in the form of a 3D game presented to the patient. The filtered data received are then provided as input to an automatic exercise classification algorithm. The algorithm decides if the exercise was accurately performed. This classification feedback is displayed on the mobile device in a readable form translated to a 3D visualization, after the end of the motion. The procedure is then concluded and the participant can perform another repetition of the same exercise or can select a different exercise.

4.1.2 Gamification Feedback

The main goal of the application is to improve compliance to the physiotherapy protocol, increase patient engagement, monitor physiological conditions and provide feedback using a reward process via a gamified experience, using the following methods:

Real-time IMU feedback. Raw data from the IMU are filtered and limb orientation is determined. By collecting this data, the proposed framework visualizes in real-time an approximation of the user's motion in a 3D scene. In this context, a user engages in a serious game of a specific objective, for instance, the patient is instructed to try and fly an airplane while moving the knee (Figure 49) in the vertical axis in order to collect coins. In this specific context the user engages to the Knee Extension exercise that requires the user to perform extension of the knee while moving it up / down along with the movement of the airplane.



Figure 49: Airplane Game example. The user raises the airplane by moving the operated knee.

The game is designed so as to motivate the user. A number of mini-games are designed (Table 2) for the four TKR exercises specified by the physiotherapists. The exercises in question are selected by the physiotherapists based on the American Academy of Orthopaedic Surgeons TKR exercise guide (Convery & Beber 1973). For testing purposes, the user can select any combination of gamified TKR exercises listed in Table 1 for every game implemented. The application's main menu provides a simple way of selecting exercises, games, as well as adding users as can be seen in later section of the Main Menu Design (Figure 51).



| Game | Screen Shot | Description |
|---------------|---|--|
| Airplane Game |  | When the user moves the operated knee, the airplane also moves vertically following the movement of the knee in order to gather coins. |
| Fish Game |  | When the user moves the operated knee, the fish also moves horizontally following the movement of the knee in order to gather coins. |

Table 2: Implemented Games for exercises in Table 1

Classification feedback. The raw data of the IMU are inserted in the classification algorithm. The algorithm decides whether the exercise has been accurately performed. If by the end of a single repetition, the exercise was classified as accurately performed, the player is rewarded, e.g. by increasing a coin score attribute and by providing animated information related to the success of movement. If the exercise was classified as inaccurately performed, in which case the movement violates angle or acceleration constraints, the application informs the user of the correct movement, e.g. by providing a limb movement animation and encourages the patient to try again. Along with the classification result, the algorithm also outputs the maximum ROM percentage of the achieved movement for this repetition. A 100% percentage means that the patient achieved maximum rotation of the knee. The maximum rotation of the knee differs in each designated exercise. For instance, in relation to the Knee Extension, it is 90° degrees. When a patient achieves a maximum ROM percentage of 50% in a single repetition then this corresponds to 45° degrees orientation. The generated repetition data are serialized in a local mobile database and are used through the Automatic Graph Generation procedure of this application.

4.1.3 Binary Classification

Having briefly discussed the layout of the application, it is worth summarizing the training procedure. The exercises in question are selected by the physiotherapists based on the American Academy of Orthopaedic Surgeons TKR exercise guide (Convery & Beber 1973) as mentioned in the Medical Background section. The physiotherapists also supervise each exercise in order to determine whether it was performed in a compliant manner with the correct body posture and speed rate. The sensor is worn by the subject on the appropriate limb location. The physiotherapist evaluates the exercise result.

A binary evaluation would indicate if an exercise is successful or unsuccessful under the specified criteria. The raw data of the IMU for each exercise are saved on a local mobile database. The participant proceeds to the next exercise in the same manner. When sufficient data are collected, the same procedure is repeated for the next participant. When all participants have completed the procedure, the data gathered can be used to train and test the classification algorithm. The training and testing procedures take place at the Orthopaedic Clinic of Chania General Hospital under the supervision of a physiotherapist. This procedure gave sufficient feedback in order to improve and impose custom constraints on the procedure offering feedback to the user. The classification algorithm input is the sensor filtered data using Kalman filtering along with complementary filtering for testing (Bachmann et al. 2001, Mahony et al. 2008, Kalman 1960, Grewal 2011). The first iteration of this algorithm checks the filtered accelerometer measurements. If these measurements exceed a predefined threshold, inferred from the training procedure, the user is advised to lower the rate of limb motion. Along with the filtered gyroscope measurements an estimation for the current user ROM is evaluated. The ROM is evaluated only if the filtered angle measurements are constantly within the specified by the training process acceptable thresholds. In any other case the measurement is labeled incorrect.



Figure 50: Airplane game with repetition classified as incorrect. Patient is advised to try again.

The angle of rotation of the limb must be in a specified range during motion, as well as of certain speed. If each movement is performed too fast, then the participant is informed indicating inaccurately performed movement and is advised to try again (Figure 50). This, alone, represents the initial classification feedback. The next iteration of the classification algorithm should automatically fine-tune angular and speed constraints inferred from training data with the help of machine learning methods.

4.2 Mobile Application UI Design

The proposed gamification framework is designed to be ultra-portable utilizing just a single Inertial Measurement Unit (IMU) sensor for tracking limb orientation and an android mobile device. The users of the application are TKR patients that have undergone the respective surgery and are in need of rehabilitation to restore their knee functionality. Most people who undergo a knee replacement are between the ages of 50 and 80. It is important for the User Interface of the application to be friendly to use to these people maximizing their experience. This is not always the case since elderly people may not always share the familiarity with technology that younger age groups probably have. Still the Main UI of the described application should be as simplistic as possible in order to increase usage probabilities even from people less acquainted with technology. Also the UI should provide more advanced features for those who are familiar with mobile use. Example of basic features are the *Start Game*, *Select Game*, *Select Exercise* functionalities. In each most basic form the patients must just press the Start Game button. Then the game guides them to fulfill a series of repetitions. Advanced features include the *Statistics Menu (Stats)* along with advanced gameplay features of each specific game described in the following sections.

4.2.1 Main Menu Design

The discussed implementation leads to a very specific design of the Main Menu and sub menus. The Main Menu should implement the following button functionality:

- *Select User*: Displays the Users sub menu. There an already existing user can be selected or a new user can be inserted.

- *Select Exercise*: Displays the Exercises sub menu. There the user has a choice between the 4 basic physiotherapy exercises implemented in the game.
- *Select Game*: Displays the Games sub menu. There users can select a game of their liking from the implemented games.
- *Start Game / Resume Game*: Starts the selected game for the selected exercise type and for the selected user. If the user is already in game then this option unpauses the game.
- *Stats*: Displays the Graphs sub GUI. There automated graph statistics are displayed for the current user and exercise type.
- *Exit / Exit to Menu*: Exits the application. If the user is already in game then this option exits to Main Menu screen.

Main Menu was designed in Unity Editor with the help of the UI System. It resides in the MenuCanvas Singleton. Its button events are handled by the MyMenuManager Singleton. The current values of user, exercise, game and stats are handled and accessed with the help of MyGameManager. The resulting GUI can be seen in Figure 51:



Figure 51: Application Main Menu.

A menu button is also added in the top right corner of the screen which also displays the menu when the user is in game. This is done for consistency with Android devices menu type. Furthermore, the application is always built in Landscape mode to maintain the aspect ratio of GUI elements. Portrait version limits the game usable area a lot and its use is discouraged in this kind of application. Finally the app logo can be seen in the upper left corner of the main screen.

The first of the submenus is the User Menu. It is triggered on Select User Button. It shows the existing users to choose from. It also allows the insertion of a new user with custom name. When the event is triggered the MyMenuManager passes the event to MyGameManager:

```
//Triggered on AddUser Button event  
public void AddUser()  
{
```

```

//Send handling of new user to game manager
var newUsername = _addUserInput.GetComponent<InputField>().text;
_myGameManager.AddUser(newUsername);

//Update all text values with the new ones provided by _myGameManager
_currentExerciseLabel.GetComponent<Text>().text = _myGameManager.GetExerciseName();
_currentGameLabel.GetComponent<Text>().text = _myGameManager.CurrentScene;
_userLabel.GetComponent<Text>().text = _myGameManager.CurrentUsername;
//Deactive all submenus
DeactivateAllSubMenus();
}

//Triggered on ChangeUser event. List button index corresponds to user id
public void ChangeUser(int index)
{
    //Send handling of change user to game manager
    _myGameManager.ChangeUser(index);

    //Update all text values with the new ones provided by _myGameManager
    _currentExerciseLabel.GetComponent<Text>().text = _myGameManager.GetExerciseName();
    _currentGameLabel.GetComponent<Text>().text = _myGameManager.CurrentScene;
    _userLabel.GetComponent<Text>().text = _myGameManager.CurrentUsername;
    //Deactive all submenus
    DeactivateAllSubMenus();
}

```

The User Menu buttons are created dynamically by the application by passing them the appropriate text and event delegates. This way scrolling is handled by Unity. This is done in a lazy manner when the Select User button is pressed and the `ShowUserMenu` method is called:

```

//ShowUserMenu event is triggered when Select User Option of Menu is pressed
//This method dynamically creates the buttons if they don't exist and handles scrolling
public void ShowUserMenu()
{
    //Request ordered usernames from game manager
    var names = _myGameManager.GetAllUsernamesOrdered();
    for (var i = 0; i < names.Length; i++)
    {
        //If index greater than current buttons then add new button
        //Happens in the beginning or when new user is added
        if (i >= _userContent.childCount)
        {
            //Instatiate new button
            var button = Instantiate(_userPrefabButton, _userContent) as GameObject;
            //Add the change user event as a button action. Encapsulate the ChangeUser
            // event to a delegate and pass as parameter to button click listener
            button.GetComponent<Button>().onClick.AddListener(
                delegate { ChangeUser(button.transform.GetSiblingIndex()); });

            //set button text to username
            button.transform.Find("Text").gameObject.GetComponent<Text>().text = names[i];
        }
        //Differently just change name of existing button
        //No users are added so no need to check for removing buttons
    }
}

```

```
{
    _userContent.GetChild(i).Find("Text").gameObject.GetComponent<Text>().text = names[i];
}
}
_addUserInput.GetComponent<InputField>().text = "User " + names.Length;
//Deactivate all submenus first and then enable this user submenu
DeactivateAllSubMenus();
_userPanel.SetActive(true);
}
```

In MyGameManager the insertion or update of the current user is handled properly and all the user button text labels are updated in order to get the appropriate values from MyGameManager. The submenu design can be seen in the following figure (Figure 52):

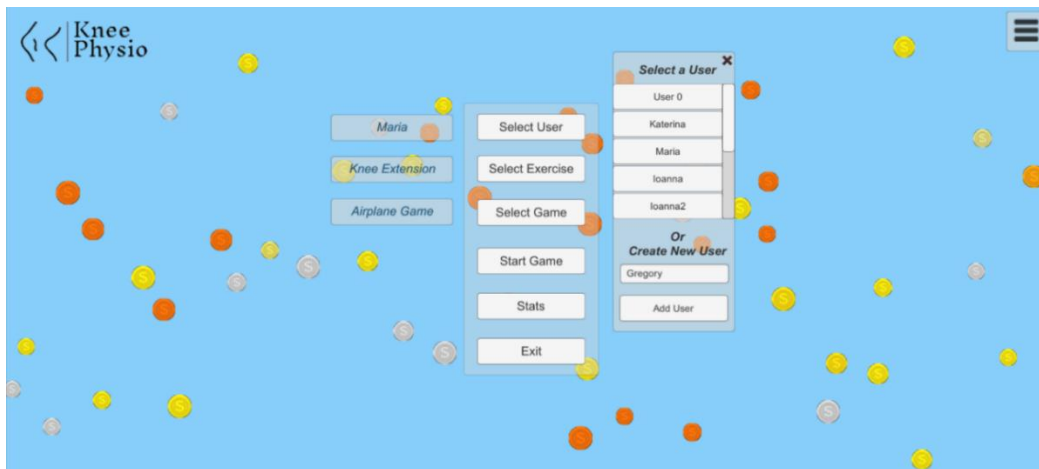


Figure 52: User Submenu.

Following is the design of the Exercise Submenu. The user can choose from the 4 available exercises implemented here (Figure 53):



Figure 53: Exercise Submenu.

The same principle applies for the Select Game submenu. The user can play any game for any exercise. So the implementation allows the user to play the same game for different exercises as long as the appropriate option through the main menu is selected (Figure 54).



Figure 54: Game Submenu.

Finally the Stats submenu shows automatically generated graphs for the selected user. It also allows to iterate through them through the arrow buttons. Once again MyMenuManager sends events for handling current graph selection to MyGameManager. When users want to exit this menu they can press either the *Back to Main Menu* button or the shortcut menu button in the top right corner of the screen (Figure 55).

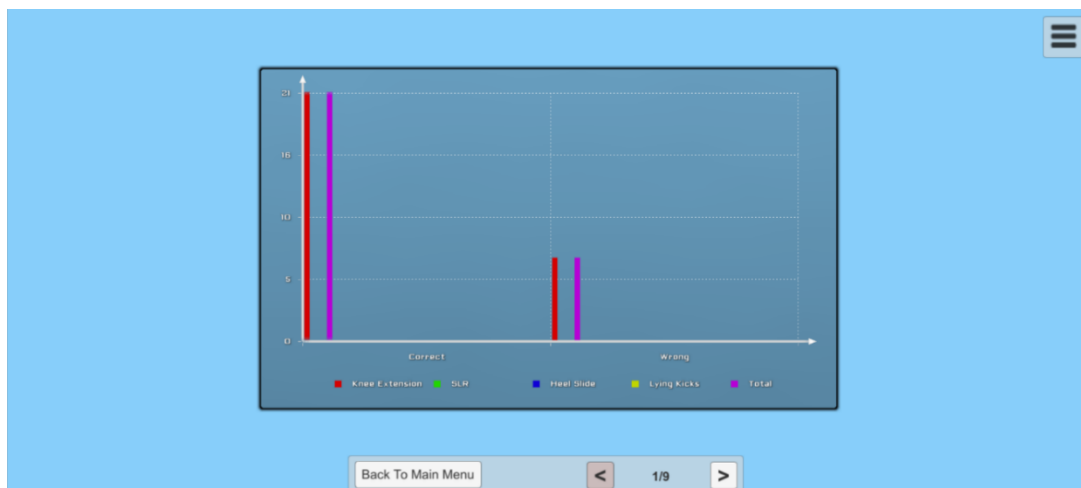


Figure 55: Stats Submenu. Iterate through graphs using arrows.

4.2.2 Gameplay HUD

MyGuiManager Singleton uses Immediate Mode GUI (IMGUI) in order to create common HUD functionality in between game scenes. Immediate Mode GUI is an entirely separate feature to Unity's main GameObject-based UI System. IMGUI is a code-driven GUI system, and is mainly intended as a tool for programmers. It is driven by calls to the OnGUI

function on any script which implements it. The IMGUI provides a useful tool for debugging the sensor data received and for creating the game HUD. The MyGuiManager HUD Singleton has the following custom functionality for displaying IMU information (Figure 56):

- *ROM bar*: Provides real-time feedback on the Range of Motion (ROM) percentage.
- *Acceleration bar*: Displays information about the Acceleration of the movement. It also points the *maxAcceleration* threshold below which the motion must be during the calibration phase.
- *Main Angle*: The MainAngle value computed by the filtering process. It is used to determine the ROM of motion. It also indicates the limits of the main angle (*minimumMainAngle*, *maximumMainAngle*) within which the motion must be during the calibration phase.
- *Side Angle*: The SideAngle value computed by the filtering process. It indicates the limits of the side angle (*minimumSideAngle*, *maximumSideAngle*) within which the motion must be during the calibration phase.



Figure 56: IMU HUD Elements. Rom bar, Acceleration Bar, Main and Side Angle Indicators.

In MyMenuManager there is also a GUI element specific to the current Exercise state of the detection algorithm. This gives the user feedback in form of text and texture hints in order to prepare for the exercise accordingly. The hints are described below (Figure 57).

- *Calibrating*: Algorithm is still in Calibration phase. User must make sure that angles are within limits as indicated by the IMU HUD Elements (Figure 56).
- *Ready for move*: The limb is within the appropriate constraints and the movement can commence.
- *Move in progress*: There is an ongoing repetition in progress right now.
- *Completed Please Rate*: At this point the exercise is completed and users are advised to rate the amount of pain they felt during the repetition.
- *Place Sensor Correctly*: The user performed wrong repetition and is advised to check sensor placement and comfort to neutral limb position.



Figure 57: Different Repetition state hints.

Additionally to this hint messages, users are notified with a success or failure message when they finish a repetition (Figure 58).



Figure 58: Correct and Wrong Repetition messages.

As mentioned above when the user completes a repetition, the user is asked to rate it according to how painful the whole procedure was. In that aspect 1 of 5 stars means no pain at all, and 5 of 5 stars means intense pain (Figure 59). This is optional and users can choose to ignore the rating of the repetition. If users however rate the repetition, then their feedback is recorded with the help of MyGameManager.



Figure 59: Pain Rating GUI.

Putting it all together, a snapshot of the result can be seen in Figure 60 during gameplay:

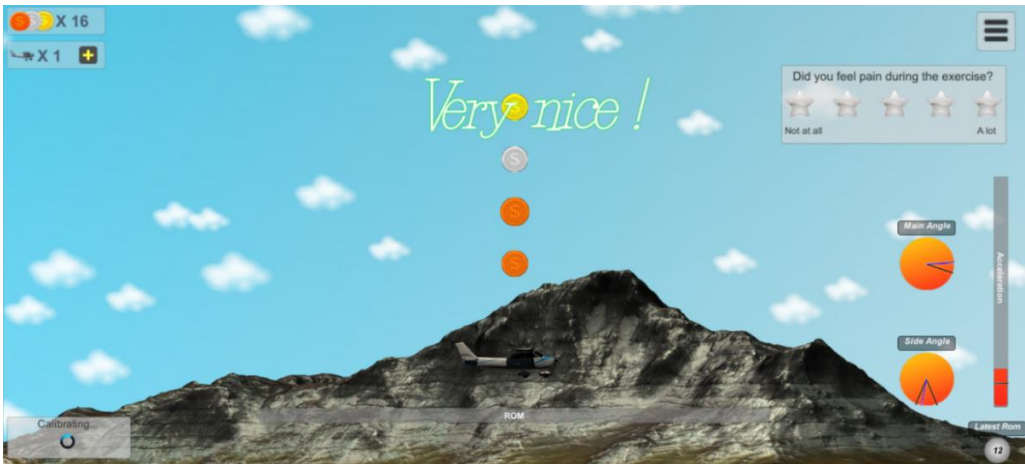


Figure 60: Final Gameplay HUD.

In each game level there are is also game logic GUI specific to a scene. In the example above, the coin score is visible, along with the number of airplanes currently available on the scene. The coin score is also a gameplay element that exists in all the implementations of mini-games in the current framework and is also used to the repetition description serialization as explained in detail on the Implementation section (Scene Hierarchy & Persistent Game Objects). Expanding the Coin Score, shows exactly how many coins a user has gathered in a pop-up panel. This screen is visible while the user is touching the Coin Score panel in the Android device (Figure 61).

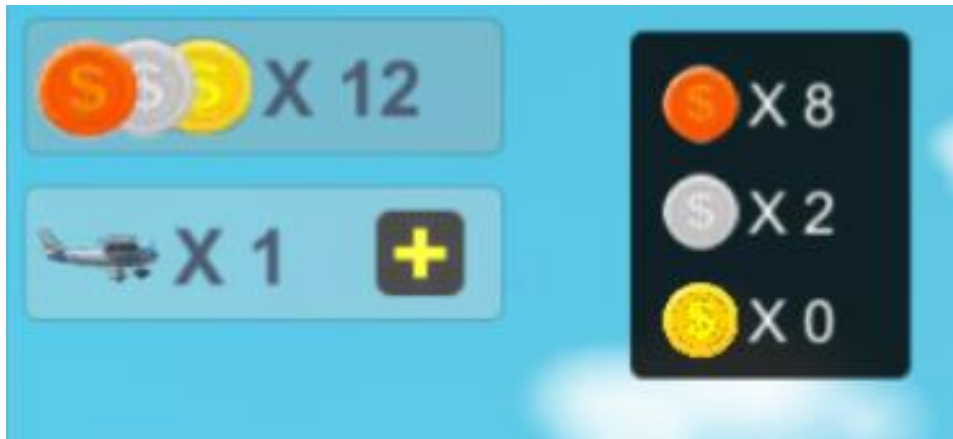


Figure 61: Coin Score panel and detailed score in pop-up panel.

The next GUI element visible is game level specific. In this mini-game it allows the user to buy more planes. In the same manner as before, by touching this element a hint pop-up panel appears that informs the user of its action (Figure 62).

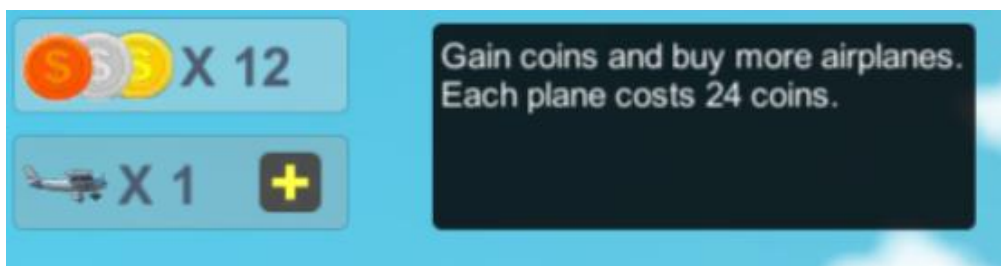


Figure 62: Specific game hint pop-up panel.

4.2.3 Automatic Graph Generation

An important feature of this framework is its graph generation capability. It is used during the testing of the application on inpatients in order to track their progress during physiotherapy sessions and analyze collected data. This data serves as an initial assessment of the physiotherapy framework capabilities and improvements. It also helps the users monitor their daily exercise progress through easily readable graphs. Moreover, it can be a useful tool for physiotherapists and orthopaedic surgeons in relation to tracking ROM improvements of patients over time. The generated graphs track the progress of users for each repetition in respect to the maximum achieved Range of Motion (ROM) and record which physiotherapy

exercises were accurately conducted and which were not. Two graph types are extensively utilized as indicated in the Results section, in this thesis. The percentage in relation to ROM per correct repetition plot graphs and the Correct/Wrong pie graphs (Figure 63, Figure 64).

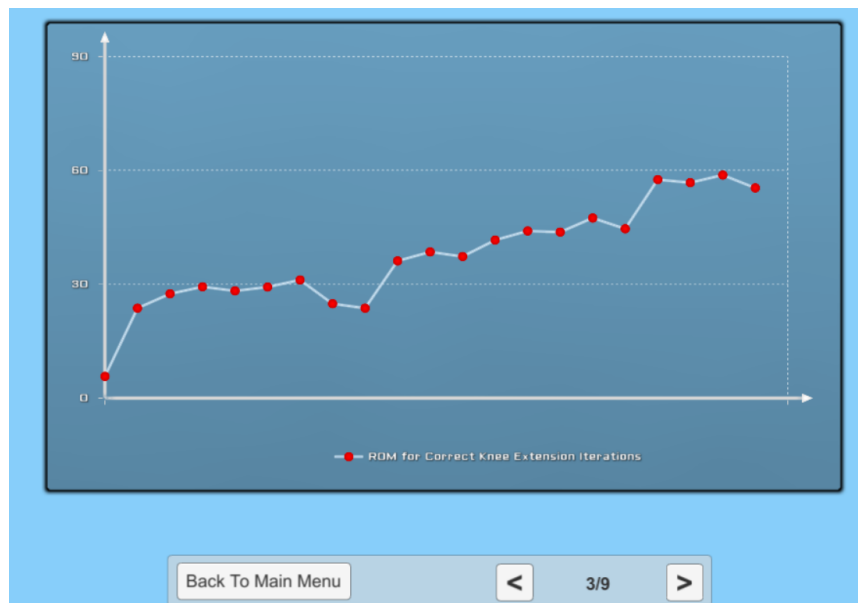


Figure 63: ROM Percentage Graph for correct repetitions.

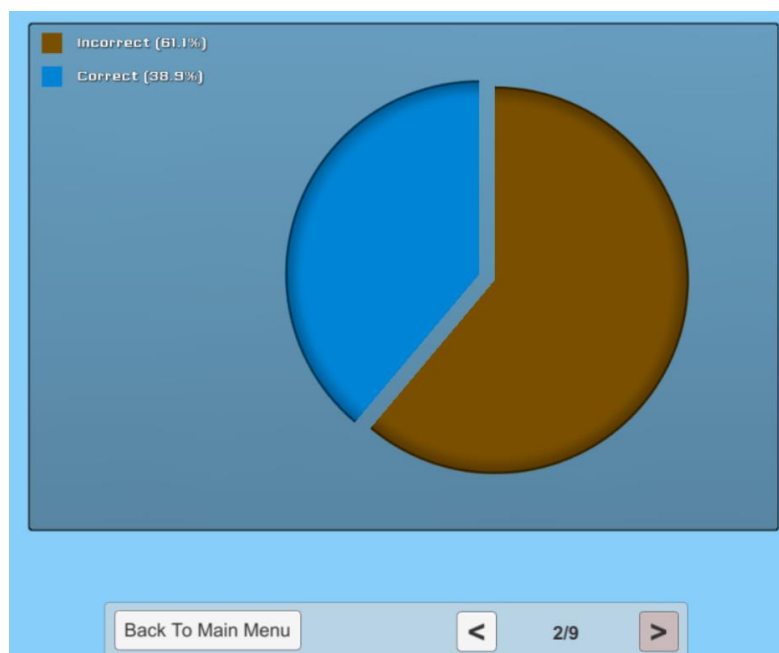


Figure 64: Correct / Incorrect repetitions pie graph.

The tool to produce this graph is a Unity package named Graph Maker. This package isn't free, but it can produce quality graphs in a very simplistic manner. Graph Maker in its turn uses another free Unity plugin named NGUI. NGUI stands for Next-Gen UI and is a package for creating custom User Interfaces. It is used though only to make easier the graph generation. In

all other aspects the Unity UI system is preferred. The graph templates reside in the GraphCanvas Singleton. When the MenuManager is instructed to display a graph it first requests the data from MyGameManager. Then it updates the graph template with the updated data. Finally the resulting graph can be displayed on screen. An example of the ROM Percentage Plot implementation is the following:

```
//Generate ROM Graph for current Exercise and User
public void RomPercentageGraphForExercise()
{
    //First get the ROM percentages for successful repetition from _myGameManager
    IEnumerable<float> correctPercentages = _myGameManager.GetUserSuccessRomPercentage();
    //Then initialize plot data and title
    WMG_Series series1 = _graphAnchor.Find("LineGraphRomPercentageCorrect").Find("Series")
                                     .Find("Series1").gameObject.GetComponent<WMG_Series>();
    series1.pointValues = new List<Vector2>();
    series1.seriesName = "ROM percentage for Correct " +
                        _myGameManager.GetExerciseName() + " Iterations";
    //Then add _myGameManager data to the plot
    foreach (var percentage in correctPercentages)
    {
        series1.pointValues.Add(new Vector2(0, percentage));
    }
}
```

MyGameManager code then returns the successful repetition ROM data using a Linq query. LINQ stands for Language Integrated Query. It is a Microsoft .NET Framework component that adds native data querying capabilities to .NET languages such as C# used here:

```
//Get ROM percentage data for correct repetitions
public IEnumerable<float> GetUserSuccessRomPercentage()
{
    //First request the data of current User and exercise from the cache
    var infos = GetInfoForUserAndExerciseType(CurrentUser, CurrentUserExercise);
    //Then use Linq query to select the RomPercentage only from the correct repetitions.
    //The percentage are 0 - 1. Convert to 0 - 100 for displaying on graphs
    return from info in infos where info.Correct select 100 * info.RomPercentage;
}
```

5 Chapter 5 – Implementation

5.1 Node Implementation

5.1.1 Hardware Setup

The accelerometer and gyroscope readings received by the Inertial Measurement Unit (IMU) are independent of the hardware setup. The application is responsible for filtering the received data and not the sensor node. This configuration provides two axes of rotation which is adequate for the majority of simple rehabilitation exercises and applications. These axes correspond to the pitch and roll angle measurements (x and y axes respectively as seen on Figure 23). The yaw measurement (z axis) additionally needs magnetometer data to be computed. This however adds extra complexity to the current project with small gain since two axes of movement are adequate for the scope of a physiotherapy application. The current project setup can be used by any wearable device that employs an accelerometer and gyroscope and complies with the raw data communication format. The sensor node consists of:

- Raspberry Pi model Zero W.
- IMU model MPU-9150.
- Rechargeable Li-Ion battery 1200 mAh.

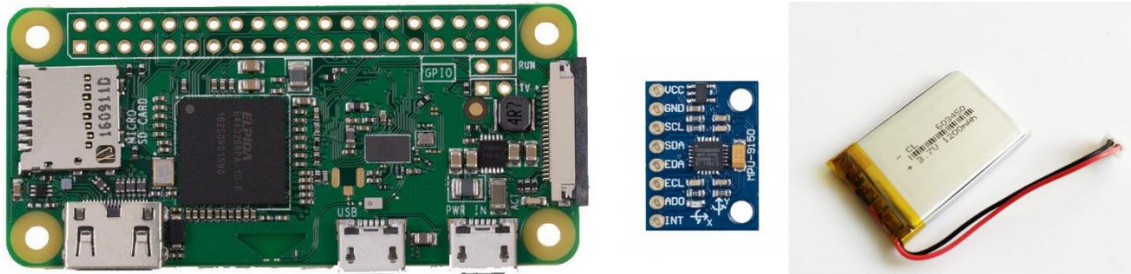


Figure 65: Node Components. Left, Raspberry Pi. Center, IMU. Right, Battery.

A summary of hardware component functionality used for the custom-made sensor node creation in this project follows. Raspberry Pi model Zero W is a credit card sized computer that runs Raspbian with Pixel (Linux Based kernel). It includes a pin header of 40 pins used for connections with all sorts of hardware peripherals, in our case an IMU. The IMU model used in the current work is the MPU-9150. It contains an accelerometer, a gyroscope, a magnetometer and a temperature sensor. It is connected to Raspberry Pi and sends the raw data captured signifying rotational motion. The connection to Raspberry Pi is achieved via the I²C protocol. This protocol employs the use of 4 pin connections: VCC, GND, SDA and SCL. The IMU pins are simply connected to the appropriate Raspberry Pi pins that implement the I²C protocol. This results to the following schematics (Figure 66):

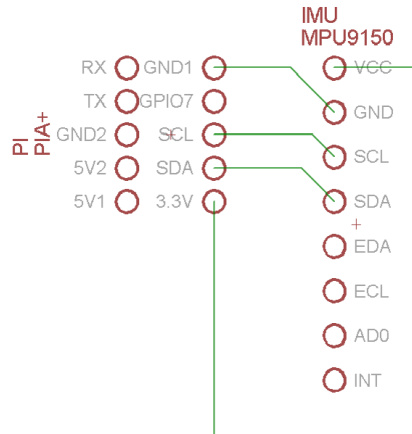


Figure 66: I²C Connection implementation between Raspberry Pi Zero W and MPU-9150.

The resulting node also has a Raspberry Pi Zero W case to protect it along with the IMU and the I²C connection in between them. An important issue that arises is the stabilization of the implemented sensor to the patient's knee. The node will be attached on the patient's limb during the training / testing phases. IMU is sensitive to misalignments and this can affect the collected data from the two axes of rotation and furthermore the observed results. To address this issue an elastic bandage is used that keeps the sensor node in place at the patient shin of the knee (Figure 67).



Figure 67: Resulting sensor node & Elastic bandage for stabilization on patient's knee.

Raspberry Pi Zero W integrates Bluetooth 4.1 and LE which makes it possible to send the raw data received from the node to our designed application that runs on an Android mobile device. By employing a rechargeable 1200 mAh Li-ion battery, the node can send data for 7 hours on full capacity. This is more than enough in order to perform the initial testing of the designed framework to Total Knee Replacement (TKR) patients.

5.1.2 Software Setup

The data are collected by Raspberry Pi using a custom script implemented in the Python programming language. Then the sensor node acts as a Bluetooth server waiting for incoming connections from Android devices. When a connection is established, the sensor

begins sending the raw IMU data to the client (Android device). When a connection is received by the client, it sets up the IMU, e.g. the gyroscope is sampled at 100 Hz. A received connection means the application is running on the end-user. The accelerometer, gyroscope and temperature data are collected and sent via Bluetooth to the application at a rate of 100 Hz. This continues until the user exits the application, or the battery depletes.

In order for the custom python script to work we must setup Raspberry Pi as a microcontroller. This procedure is discussed in the Embedding the IMU section above. The custom python script that implements the Bluetooth server is named `bserver.py`. This script in order to run automatically needs a bash script to run at system startup. As previously described in the *Embedding the IMU* section this script should be of the following form: `/home/pi/MyCustomPath/MyCustomScript.sh`. In the current implementation this script is: `/home/pi/bser/runBTServer.sh`. This script sets up the raspberry pi Bluetooth in order to be discoverable by nearby Android devices. It then goes to the Bluetooth server folder and runs the `bserver.py`. Its contents are the following:

```
#!/bin/bash
sudo hciconfig hci0 piscan
cd pythonScripts
sudo python bserver.py
```

The `bserver.py` script waits for incoming connections from nearby devices. If one is found it configures the sensor using the appropriate I²C registers and continuously sends data till the connection is interrupted. This happens if the Android device gets out of range and the connection is reset, or if the battery on the sensor depletes and script stops executing. If connection is reset the process is repeated, sensor device is reset and data are sent continuously in a frequency of 100 Hz per second. Bluetooth socket creation and service advertisement is implemented below:

```
#Create Bluetooth socket
#initialize to listen at any port
server_sock=BluetoothSocket( RFCOMM )
server_sock.bind(("",PORT_ANY))
server_sock.listen(1)
port = server_sock.getsockname()[1]

#Advertise Bluetooth service to be visible to android devices
advertise_service( server_sock, "MyBTServer",
    service_id = UUID,
    service_classes = [ UUID, SERIAL_PORT_CLASS ],
    profiles = [ SERIAL_PORT_PROFILE ] )
```

Then follows code responsible for resetting the IMU and configuring sensor parameters. After the configuration of the sensor, the thread blocks until data are ready. Then data are read from the sensor in the designated rate. Consequently a checksum is computed and the resulting bytes are send through the Bluetooth socket:

```
while True:
```



```

try:
    #Wait till data are ready
    #Data will be ready with a frequency of 100 Hz
    while bus.read_byte_data(address, REG_INT_STATUS)!=1:
        pass

    #Read consecutive data from registers containing
    #Accelerometer, Temperature & Gyroscope measurements
    result = bus.read_i2c_block_data(address, REG_START, DATA_LENGTH)
    #make it into an array and compute checksum
    resultArray = bytearray(result)
    checksum = sum(resultArray)
    #Store checksum in 2 bytes high and low
    #Shift high byte, mask low byte
    checksumHigh = checksum >> 8
    checksumLow = checksum & 255
    #append bytes to the result to send
    resultArray.append(checksumHigh)
    resultArray.append(checksumLow)
    #Convert and send bytes
    #The received result is 16 bytes
    #14 data bytes described above and 2 bytes checksum
    result = bytes(resultArray)
    client_sock.send(result)

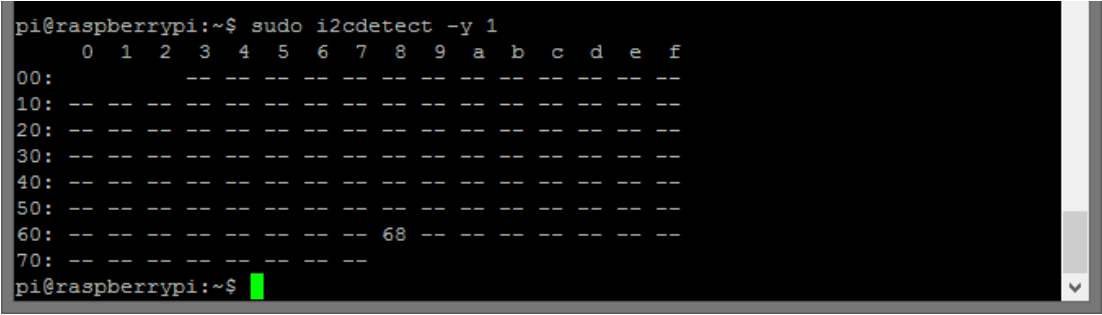
```

The register configuration reading and writing is achieved using the Register Map documentation for MPU-9150 as described in the IMU Functionality section. The data sent to the application are 16 bytes in a frequency of 100 Hz. This results to a constant throughput in the Bluetooth connection of $16 * 8 \text{ bits} * 100 \text{ s}^{-1} = 12.8 \text{ Kbit/s}$. It follows, that every hour of full sensor activity 5.76 Mb of data are created. The 16 bytes correspond to 6 byte measurements for accelerometer, 2 byte measurements for temperature sensor, 6 byte measurements for gyroscope and 2 bytes which are the checksum. Note that Bluetooth uses a reliable connection for sending data and checksum could be omitted. Nevertheless it is implemented as a good practice to check if the data are received correctly on the client. The additional data being sent are only a few Kbit overhead that the Bluetooth connection can handle. Note, that the bottleneck is not the Bluetooth connection but the sensor sampling rate which is configured to 100 samples/s. Although there is the possibility, there is no need to configure faster sampling frequency as this frequency is adequate in order to track human motion of a limb for rehabilitation.

One last implementation detail is the choice of the `BUS_ADDRESS = 0x68` for the IMU sensor device. If the I²C connection is correctly setup, then the bus address for the connected I²C device can be obtained using the `i2cdetect` command in the bash of the raspberry pi OS environment:

```
sudo i2cdetect -y 1
```

The `-y` argument points to the specific i2c device. For this Raspberry Pi model it is 1 by default. The following ascii schematic will appear that specifies the desired address (Figure 68).



```
pi@raspberrypi:~$ sudo i2cdetect -y 1
    0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:  -- -- -- -- -- -- -- -- -- -- -- -- -- --
10:  -- -- -- -- -- -- -- -- -- -- -- -- -- --
20:  -- -- -- -- -- -- -- -- -- -- -- -- -- --
30:  -- -- -- -- -- -- -- -- -- -- -- -- -- --
40:  -- -- -- -- -- -- -- -- -- -- -- -- -- --
50:  -- -- -- -- -- -- -- -- -- -- -- -- -- --
60:  -- -- -- -- -- -- -- 68 -- -- -- -- -- --
70:  -- -- -- -- -- -- -- -- -- -- -- -- -- --
pi@raspberrypi:~$
```

Figure 68: i2cdetect show the address an I²C device is connected.

If there are more devices connected with I²C protocol they will also appear in this schematic in a different address. So in the same Raspberry PI multiple IMUs could be connected if the application required to do so, but this is not the case in the current project.

5.2 Mobile Application Core Implementation

The client receives the data using an Android activity implemented in Java. The Unity Game Engine is utilized for visualization. The Java Native Interface (JNI) is used in order to transfer data from Java to C#, because C# is the programming language used by Unity. Unity provides leading architecture for game development along with a number of assets available for free. The application guides the users in order to become familiar with the application gameplay and furthermore to try and improve their ROM through guided exercises.

The user selects the exercise to be performed, e.g. Knee Extension. The system then guides the user in order to perform a few initial training exercises. The system classifies the exercise performed as correct or incorrect and provides visual feedback to the user. When the users feel comfortable performing the selected exercise, they can select to engage in a series of mini-games. By performing the limb motion, e.g. moving the knee, users interacts with objects in 3D space.

Indicative games are the Airplane Game and Fishes Game (Table 2). When selecting the Airplane game, the user tries to move an airplane in the vertical axis and gather as many coins as possible while performing the physiotherapy exercises. High ROM percentage means greater coin score, so greater reward. Lower ROM represents lower reward. Both cases are rewarded depending on their ROM percentage. When a movement is incorrect, the system will not reward the users but will give them hints in order to continue by correcting limb motion (Figure 58). The reward is greater for larger ROMs in order to engage users to improve their movement.

Similarly, when engaging with the Fishes game, the user moves a fish horizontally this time with the movement of the knee and tries to collect as many coins as possible depending on the ROM achieved. Along the whole process, the system prompts the user by offering additional visual hints according to detection of motion as captured by the sensor (Figure 57).

5.2.1 Android Plugin for Bluetooth Data

In the *Embedding the IMU* section the process for creating a sample plugin extending the `UnityPlayerActivity` was described. Here this description is used along with specific implementation details in order to create a plugin using Java that will receive Bluetooth data from the sensor and will transfer them to Unity. There C# language is used for manipulation and filtering of the received data.

What the Java activity does, is that it tries to establish a connection with the Bluetooth server implemented in the sensor node as described in the Software Node Implementation section. For this purpose client and server use the same UUID. UUID stands for Universally Unique Identifier. It is a unique Bluetooth service identifier used by Bluetooth sockets to establish a connection. The `bserver.py` advertises this service and the java client finds the device and connects to the service. When a connection is established the Java Activity initializes a background thread that constantly receives byte packages from the sensor server. There the data are checked for correctness using their byte length and checksum. If not correct a package is discarded. If everything is ok then the actual data are sent to Unity through C# for further processing. The background thread method that receives the data and then forwards them to Unity is shown below:

```
//Inner class that handles background receiving of data from the sensor
//And asynchronous sending of the data to Unity Application and C#
public void run()
{
    while(!Thread.currentThread().isInterrupted())
    {
        //Check available bytes at the stream
        //If it is not the expected data length discard them and try again
        int bytesAvailable = mmInputStream.available();

        if(bytesAvailable != DATA_LENGTH)
        {
            mmInputStream.skip(bytesAvailable);
            continue;
        }

        byte[] packetBytes = new byte[bytesAvailable];
        mmInputStream.read(packetBytes);

        //Get received checksum from last 2 bytes of data sent
        int checksumReceived = (packetBytes[packetBytes.length - 2] << 8) +
                                (packetBytes[packetBytes.length - 1] & 0xFF);

        //Compute checksum for the actual data (excluding the checksum bytes)
        //NOTE: Server sends unsigned bytes. Java receives signed bytes.
        //Bytes are received correctly but must be masked in order to
        //handle the received bytes as unsigned int
        int checksumComputed = 0;
        for(int i = 0; i < packetBytes.length - 2; i++)
        {
            checksumComputed += (packetBytes[i] & 0xFF);
        }

        //If the 2 checksum are not the same discard package
    }
}
```

```

        if(checksumComputed != checksumReceived)
        {
            mmInputStream.skip(bytesAvailable);
            continue;
        }

        //Keep only the data bytes excluding checksum bytes
        byte[] receivedBytes = new byte[packetBytes.length - 2];
        System.arraycopy(packetBytes, 0, receivedBytes, 0, DATA_LENGTH - 2);
        //Convert bytes to string to be compatible with UnitySendMessage
        final String data = Arrays.toString(receivedBytes);
        //And finally send asynchronous message to Unity with sensor data!
        //Data are sent to MyController script that is part of an active GameObject
        //MyController must have a method BluetoothData with a string argument(data)
        //This argument is our actual byte data
        UnityPlayer.UnitySendMessage("MyController", "BluetoothData", data);
    }
}

```

When this activity is configured properly and starts running in the application background it will send asynchronous messages every update with any newly received data. Then the application method is responsible for filtering the received data. The functionality is kept simple at this point. No filtering is performed. The end application in Unity is responsible for processing the received byte data. Finally for this configuration to work, the `AndroidManifest.xml` file must be specified:

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.tuc.kneePhysio">
    <uses-permission android:name="android.permission.BLUETOOTH" />
    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <application android:icon="@drawable/app_icon" android:label="@string/app_name"
android:theme="@style/UnityThemeSelector">
        <activity android:name=".MainActivity"
            android:label="@string/app_name"

android:configChanges="fontScale|keyboard|keyboardHidden|locale|mnc|mcc|navigation|orientation
|screenLayout|screenSize|smallestScreenSize|uiMode|touchscreen">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>

```

It is important to have correct package naming that should be the same in MainActivity and in Unity Android Player Settings. The last step towards building the plugin is compiling java code and build the jar file for the project location Plugins folder:

```

ProjectPath\Assets\Plugins\Android\src>javac MainActivity.java -source 1.7 -
target 1.7 -classpath "C:\Program

```

```
Files\Unity\Editor\Data\PlaybackEngines\AndroidPlayer\Variations\mono\Development\Classes\classes.jar" -bootclasspath
C:\Users\grego\AppData\Local\Android\sdk\platforms\android-21\android.jar -d .

ProjectPath\Assets\Plugins\Android\src>javap -s
com.tuc.kneePhysio.MainActivity

ProjectPath\Assets\Plugins\Android\src>jar cvfM ../bluetoothPlugin.jar com/
```

5.2.2 Filtering Received Data

The data from the Java Activity are received by Unity in MyController Singleton using the `BluetoothData` method. The data are received in a frequency of 100 Hz as configured by the gyroscope which is the bottleneck of the communication. This method is responsible for parsing the received Bluetooth data as byte values from the string and computing the 2's complement of the sensor data according to the MPU-9150 specifications. The gyroscope data are converted from degrees/s to rad/s for ease of computations in filtering methods. Consequently the received data are filtered. Although Kalman filter is used in the visualization of data, there is also an implementation of complementary filtering with and without gyroscope data. This is done mainly for debugging reasons and for sanity check that the aforementioned filtering methods behave in a similar manner. Actually the results received from both filters are very similar if the filter parameters are properly fine-tuned. So complementary filter can also be used to provide results on the data collected.

When filtering methods using gyroscope measurements are applied, it is important to change the sign of the rate of gyroscope, when appropriate, before feeding it as filter input. This is the case for Side Angle gyroscope rate. Side Angle is restricted to $[-\pi, \pi]$. If the rotation vector resides below the horizontal 3D plane then a change in gyroscope rate must occur. This happens when Main Angle of rotation (pitch angle) resides in either $[\pi, 2\pi]$ or $[-2\pi, -\pi]$. Regarding the filtering implementation, there is a base `MyFilter` abstract class that all implemented filters use. This base class simply defines a filter storing the computed angle in each step. It exposes the `ComputeAngle` method to client code and allows the overridden classes which are the implemented filters to override `ComputeAngleInternal` in order to provide the respective functionality for each implemented filter. Kalman filtering is implemented according to the derivation of Kalman filter parameters for IMU discussed in the IMU Filtering Methods section:

```
//Implementation of Kalman filter
protected override void ComputeAngleInternal(float newAngle, float newRate)
{
    // Discrete Kalman filter time update equations - Time Update ("Predict")
    // Update xhat - Project the state ahead
    /* Step 1 */
    _rate = newRate - _bias;
    Angle += Dt * _rate;

    // Update estimation error covariance - Project the error covariance ahead
    /* Step 2 */
    _p[0][0] += Dt * (Dt * _p[1][1] - _p[0][1] - _p[1][0] + QAngle);
    _p[0][1] -= Dt * _p[1][1];
    _p[1][0] -= Dt * _p[1][1];
    _p[1][1] += QBias * Dt;
```

```

// Discrete Kalman filter measurement update equations - Measurement Update ("Correct")
// Calculate Kalman gain - Compute the Kalman gain
/* Step 4 */
float s = _p[0][0] + RMeasure; // Estimate error
/* Step 5 */
float[] K = { _p[0][0] / s , _p[1][0] / s }; // Kalman gain - This is a 2x1 vector

// Calculate angle and bias - Update estimate with measurement zk (newAngle)
/* Step 3 */
float y = newAngle - Angle; // Angle difference

/* Step 6 */
Angle += K[0] * y;
_bias += K[1] * y;

// Calculate estimation error covariance - Update the error covariance
/* Step 7 */
float p00Temp = _p[0][0];
float p01Temp = _p[0][1];

_p[0][0] -= K[0] * p00Temp;
_p[0][1] -= K[0] * p01Temp;
_p[1][0] -= K[1] * p00Temp;
_p[1][1] -= K[1] * p01Temp;
}
}

```

The noise parameters of the filter are initially configured by checking the variance of the sensor signals in resting state. Then through the testing procedure in actual limbs of Total Knee Replacement (TKR) patients some further corrections in these thresholds occurred to represent more accurately the desired movement pattern. Complementary filter is implemented on a similar manner but with simpler representation due to its nature.

5.2.3 Automatic Exercise Classification Algorithm

A first iteration of the automatic exercise classification algorithm (Figure 8) uses angular and acceleration predefined thresholds to fine-tune the system. In future iterations, these thresholds can be inferred from variant Machine Learning techniques, e.g. RVMs (Tipping 2003), using the filtered sensor data collected. Such methods will maximize the success rate of the current algorithm, using just a single node. A significant requirement is the correct and stable sensor placement on the *Shin* of the patient's limb. Correct placement can be illustrated by a doctor or physiotherapist. Additionally if the wearable device isn't placed with the correct orientation, when patients try to perform an exercise they will be notified that the sensor is not positioned correctly. The input data of the algorithm are the filtered smoothed data of the sensor listed below and are acquired using Kalman filtering (Kalman 1960, Grewal 2011):

- *mainAngle*. The angle in the direction of the exercise movement (corresponds to pitch angle in popular notation). It determines the maximum achieved ROM of the patient.

- *sideAngle*. The angle that detects sideways limb motion (corresponds to roll angle in popular notation). It is used along with *mainAngle* to determine when user movement deviates from what is perceived to be an accurately performed exercise.
- *acceleration*. Limb acceleration is used to detect motion activity and probable wrong movement, e.g. if the limb movement is too fast.

The output of the algorithm is a computational decision whether the physiotherapy exercise is performed correctly or incorrectly by the patient. The training parameters are the following angle and acceleration thresholds:

- *minimumMainAngle*, *maximumMainAngle*. The main angle's minimum and maximum thresholds in the direction of motion. If *mainAngle* exceeds these thresholds, the exercise is deemed incorrectly performed.
- *minimumSideAngle*, *maximumSideAngle*. The side angle's minimum and maximum thresholds in the direction of motion. If *sideAngle* exceeds these thresholds, the exercise is deemed incorrectly performed.
- *maxAcceleration*. The maximum allowed acceleration. If acceleration exceeds this threshold, the exercise is deemed incorrectly performed.

The implemented algorithm is designed to work for each one of the specified rehabilitation exercises (Table 1) and can be generalized to additional exercises of similar format. Each exercise will just require different training parameters. Currently, these parameters are manually defined, always taking into account the training samples collected. When the first iteration of the designed application is completed and more samples are collected from patients with TKR, these parameters can be automatically adjusted using machine learning techniques in order to reliably measure success rate. This is not a trivial task as the data collected from training must be of a significant amount and variance in order to provide reliable and generalized results.

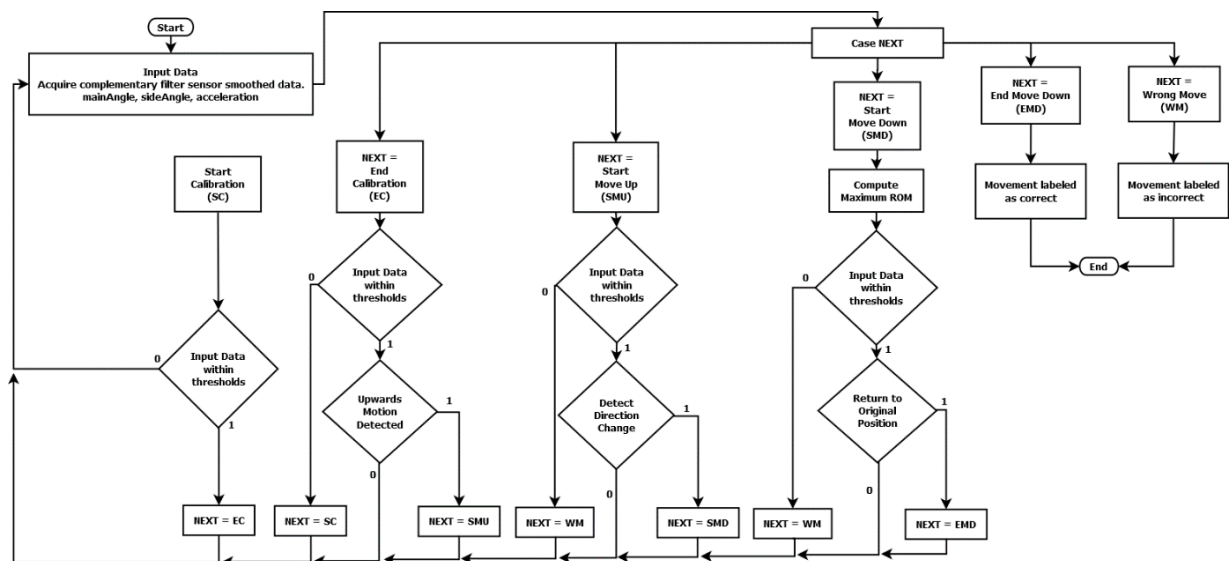


Figure 69: Automatic Exercise Classification Algorithm Simplified Diagram.

The Algorithm is implemented by MyExercise abstract class. The states are represented by the DetectionState enumeration:

```
//Automatic Algorithm Detection States
public enum DetectionState
{
    StartCalibration,
    Calibration,
    EndCalibration,
    StartMoveUp,
    StartMoveDown,
    EndMoveDown,
    WrongMove
}
//Current Algorithm State
private DetectionState _currentState = DetectionState.StartCalibration;
```

MyExercise implements the first iteration of the algorithm depicted in the figure above. It then exposes custom events to the programmer that can use these events to add game logic functionality by overriding MyExercise class.

5.2.4 Scene Hierarchy & Persistent Game Objects

In earlier section the Singleton pattern was defined. This is used extensively through the application design. The singletons can communicate safely between each other and between specific multiple GameObject in a scene, since their functionality is unaffected by Scene change. In the current game there are 6 objects acting as MySingleton.

- *MyController*: Receives Bluetooth data, performs filtering on these data and helps in recording and debugging.
- *MyGameManager*: Manages serialization and deserialization of MyExercise repetition data. It is the placeholder for storing current data. Finally it helps with Scene transitions and parallel task management.
- *MyMenuManager*: Uses Unity UI System to Handle Button Events from MenuCanvas Singleton and displays GraphCanvas Singleton.
- *MyGuiManager*: Uses Immediate Mode GUI to implement Custom GUI logic for the Game HUD. Also helps for custom debugging.
- *MenuCanvas*: GameObject that just holds the Menu GUI designed in Unity.
- *GraphCanvas*: GameObject that holds the automatically generated plots of the application. It uses the Graph Maker and NGUI packages for displaying the generated graphs.

MyController as a Singleton is responsible for filtering the data and providing values for MainAngle and SideAngle as described in the *Filtering Received Data* section. The application then is responsible for managing and visualizing the data provided. This data management is performed by MyExercise base class along with the help of MyGameManager Singleton for managing the application data.

MyExercise is responsible for providing the rules under which one Exercise Repetition is recognized and visualized on the screen. As described in the *Gamification Feedback* section

there are 4 exercises implemented (Table 1). Each Exercise handles the data using the same algorithm. The core differences are that they use different angle thresholds such as MainAngle limits (minimum, maximum), starting repetition MainAngle, and ending repetition MainAngle. Using the computed angles and thresholds, the ROM Percentage is computed every frame. For each exercise repetition a Maximum Achieved ROM Percentage is also computed.

The frames are drawn in a slower rate than the actual data are received and filtered. The data are filtered by MyController in a rate of 100 Hz. The visualization frame rate varies on the platform. For the current target platform (Android device) a typical frame rate is 30 Hz. This however doesn't impose a restriction. Since the data are filtered asynchronously in a rate of 100 Hz, the visualization can be slower. Even if an Android device could run to a rate of 100 Hz the difference to the human eye would not have been easily perceived compared to a rate of 30 Hz.

In order to gather the required results the exercise repetition data must be stored and handled appropriately. An exercise repetition is characterized by the following attributes:

- *User*: The user that performed the exercise.
- *Type*: The type of this Exercise.
- *Correct*: If the implemented algorithm detected the movement as Correct or not.
- *StartDate*: The beginning of the repetition.
- *StartActiveDate*: When the repetition became active. This happens when a user stops being idle and moves the limb for the first time.
- *StopDate*: The ending of the repetition, either correct or wrong.
- *Duration*: The total duration of repetition including idle and active phases.
- *ActiveDuration*: The total duration of repetition only in active phase.
- *Rom*: Maximum Rom in degrees.
- *RomPercentage*: Maximum Rom in percentage.
- *Score attributes*: The implemented exercise games use Coin Score as award feedback to the user. That can be either Gold, Silver, Bronze Coins depending on the maximum ROM achieved.
- *Raw Data*: The raw byte data recorded through the repetition for future analysis.

Each repetition information is serialized for further analysis. A custom serialization scheme is employed instead of a commercial database scheme. There are 3 reasons that lead to this implementation choice.

- The data stored locally are just a few with simple format.
- Simpler to test in both PC and Android platforms.
- No extra initialization overhead and technical requirements is added to the application.

For each repetition the information created is serialized using 2 files. These file paths use the following format:

```
{PersistentDataPath}\users\{userId}\{ExerciseType}\{RepetitionId}.info  
{PersistentDataPath}\users\{userId}\{ExerciseType}\{RepetitionId}.data
```

{PersistentDatapath} is the OS specific data path that Unity uses. All the data are stored in the custom users folder. Inside it there is the *{userNum}* folders. These folders contain data for each user with the specific unique auto increment id. Then each user has exercises performed. The *{ExerciseType}* folder specifies the type of exercise currently stored. Finally inside this folder are the actual data. The *{RepetitionId}* is unique auto increment id for the current repetition. The .info file has all the information of the repetition except the raw data. An example repetition info file is the following:

```
Correct|True
StartDate|2018-01-05 00:31:55.332
StartActiveDate|2018-01-05 00:31:57.915
StopDate|2018-01-05 00:31:58.746
duration|3411
activeDuration|831
Rom|45.94853
RomPercentage|0.6054431
CoinScore|6
GoldScore|0
SilverScore|1
BronzeScore|4
```

The Raw data are separately stored to a different .data file (due to larger size compared to the .info repetition summary data) with the same id *{RepetitionId}*. An example repetition data file is the following:

```
[1, 96, 64, -56, -4, 116, -7, 96, 0, -118, 0, 107, 0, -120]
[1, 112, 64, 120, -5, -4, -7, 112, -1, 16, 0, 110, 0, -113]
[1, 36, 64, -116, -4, 48, -7, -128, -2, 67, 0, 36, 0, -127]
.
.
.
[2, -64, 66, 88, -14, 88, -7, -128, 16, 28, 0, 74, -2, -86]
[3, 96, 63, -56, -14, 80, -7, -112, 17, 36, -2, -74, -3, -20]
```

This serialization is implemented in a separate thread. The serialization process is CPU demanding task and if not handled asynchronously the game execution will freeze for a few frames till serialization is completed. Until the asynchronous serialization is completed the exercise remains in the Calibration phase. The MyGameManager singleton handles the custom implemented serialization of users and exercises.

The MyGameManager singleton is also responsible for deserializing repetition data. This is useful for the automated graph generation performed by the application and for keeping track of the user repetitions. The implementation of the deserialization uses a Dictionary Cache to avoid extra computations when data are not changed. So data in memory are updated only when changes are detected. This way stale data are avoided and the access to the data remains fast using $O(1)$ complexity when data are not changed and $O(n)$ when data do change, where n is the number of repetitions performed by the user for this Exercise type.

In order to use an exercise in a game it must be inherited from MyExercise base class to be given custom functionality. As an example, consider the Airplane Game. This game overrides MyExercise base class. This base class is a custom component that should be attached in a GameObject in the current scene. In this example this GameObject is

PlaneExerciseManager and its children GameObjects are the airplane models imported to Unity. The base class handles the detection of the repetition. The derived class handles the game logic by using the events and data provided by the base class. This way the algorithm implementation is hidden from the derived class. In the Airplane Game the implemented methods handle the moving in the vertical direction (up and down) along with the movement of the user's limb. They also specify the correct or wrong repetition events. On a correct repetition the score is updated and the appropriate success message is appeared on the UI. On a wrong repetition a meteor shower is activated that can destroy additional airplanes and the user is prompted to try this exercise again.

```
//Event that is triggered when move has finished successfully
public override void OnMoveCompleted(float tpf)
{
    //Compute Scores for serialization before changing in base
    coinEmitter.ComputeScore();
    base.OnMoveCompleted(tpf);
}

//Event that is triggered when move has finished unsuccessfully
public override void OnWrongMoveDetected(float tpf)
{
    base.OnWrongMoveDetected(tpf);
    //On detection of wrong move return plane to starting position
    transform.position = startPos;
    //And do custom event. Here Activate Meteor Shower
    _myMeteor.ActivateMeteor(true);
    //Reset the coin score
    coinEmitter.RestoreScore();
}
```

5.2.5 Game Levels Functionality

Once the basic architecture is setup then the specific game scene creation can take place. Games Scenes are characterized by the following features and constraints:

- There is common HUD functionality concerning IMU feedback (angles, ROM, acceleration), Exercise feedback (Exercise state feedback, success / failure feedback) and award feedback (Coin Score). These elements are described in the UI Implementation Section.
- There is different HUD functionality depending on the specific scene gameplay. An example description of HUD is provided in the UI Implementation Section.
- Each Game Level is available for each user and each exercise type. If a user would like to change game level or exercise type this should be easily achievable from the main menu. More info on this functionality is provided in the UI Implementation Section.
- There is one GameObject that has a Custom Component derived from MyExercise class. This component handles the real-time visualization feedback of the IMU. This GameObject is attached to one or more 3D model that interacts / moves analogically to the ROM achieved. This is done by appropriately handling MyExercise events as described in the *Scene Hierarchy & Persistent Game* section.
- There is a common award system for all game scenes. This consists of Coin Score. Users gather coins if a repetition is successful. Depending on the ROM achieved they

gain highest value coins. Detailed description of the award system is provided furthermore.

- There is gameplay specific functionality and game assets in every scene. Some examples of these specific scenes are described below.

The award system of each game is kept the same for consistency of observed results. This shouldn't be an absolute constraint in future development, as different kind of awards can provide a sense of personalization to the framework. On the current implementation though, each case is handled in a similar manner. This simplifies the experimental and result generation processes.

An example game level description can clarify the award process. The example game level used is the Airplane Game Level. This level is chosen because it is the game level used for testing the framework to patients at the Orthopaedic Clinic of the Chania General Hospital. The specific exercise that was used was Knee Extension. But the same game works appropriately with any exercise that might be chosen. In this game there is *basic gameplay functionality* and *advanced functionality*.

The *basic gameplay functionality* is the core of the game process. When players raise their knee in order to perform Knee Extension then they can see an airplane going upwards along with their Knee movement. In the same manner the airplane goes downwards when patients lower their knee to the neutral position. Above the plane there are some ordered coins that the plane gathers when collision is detected. Collision is detected when the airplane model bounding box touches the coin bounding box. In that event the coin score is increased depending on the Coin value (Bronze, Gold, Silver coins give 1, 2, 3 points respectively). The goal here is to maximize award on higher ROM but also provide award for lower ROM achieved. This happens in order to engage even the patients with weaker knee recovery to put more effort to gain higher reward. So when in neutral position and after calibration phase the user sees the following (Figure 70):

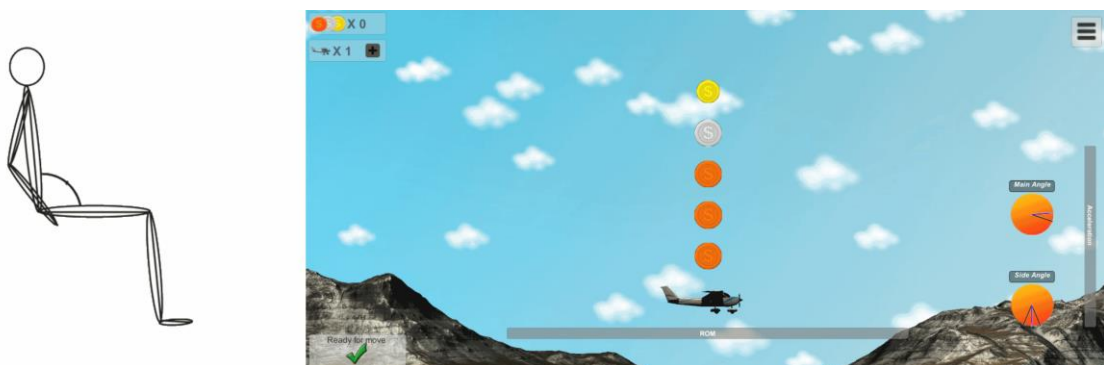


Figure 70: Neutral Position when Ready for Move after Calibration phase.

When users perform Knee Extension then they will observe that the airplane starts gathering coins and their score will increase (Figure 71):

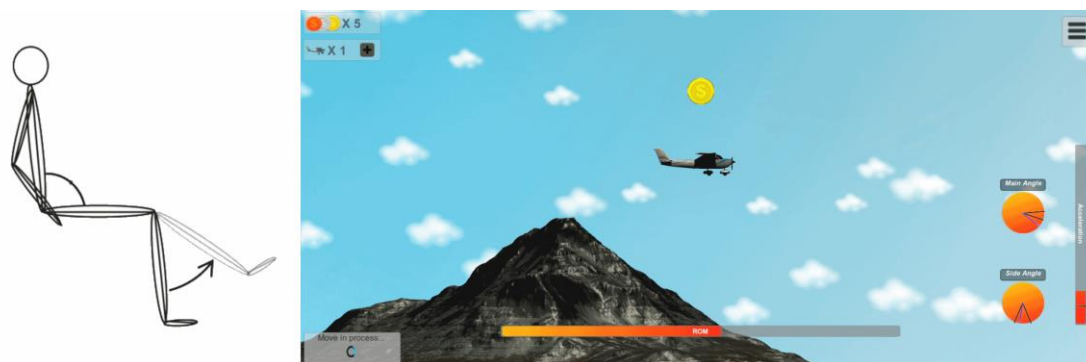


Figure 71: Upwards Movement.

The user maximum ROM Percentage during this movement was at a 65%. This user performed well but didn't manage to gather all the coins available. The current achieved score is 3 Bronze Coins x 1 point + 1 Silver coin x 2 points = 5 points. If however the user could achieve higher percentage, the Gold Coin will additionally be collected thus achieving maximum possible reward of 5 points + 1 Gold Coin x 3 points = 8 points. At this point however the movement has not yet been completed. The repetition can still be labeled incorrect. If the movement is labeled incorrect then the user doesn't gain any additional score. The user must return into neutral position. In order to provide additional motivation in this direction, when the downward movement is detected a last coin appears further down to the original airplane position that corresponds to the neutral position of the knee in reality (Figure 72):

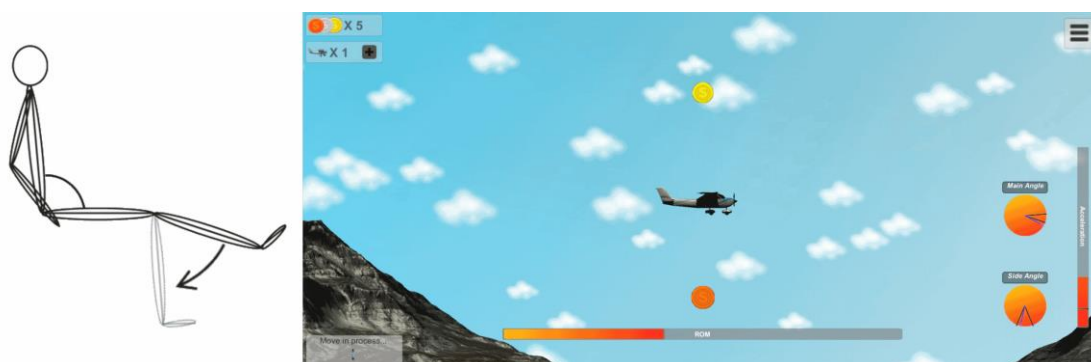


Figure 72: Downwards Movement.

The award system described was implemented with the help of physiotherapists and postoperative patients to make sure it can engage the patients to perform an exercise. As discussed further this is achieved mostly with the visual Coin motivation than the Coin Score itself.

The *advanced gameplay functionality* is an extension of the basic functionality. It provides additional gameplay features. The advanced feature in this game constitutes of the ability of a player to buy more airplanes and gather more coins faster (Figure 73). The player can control multiple airplanes with the same knee movement and collect coins and points faster. The user cannot buy more planes if there are no more coins available.

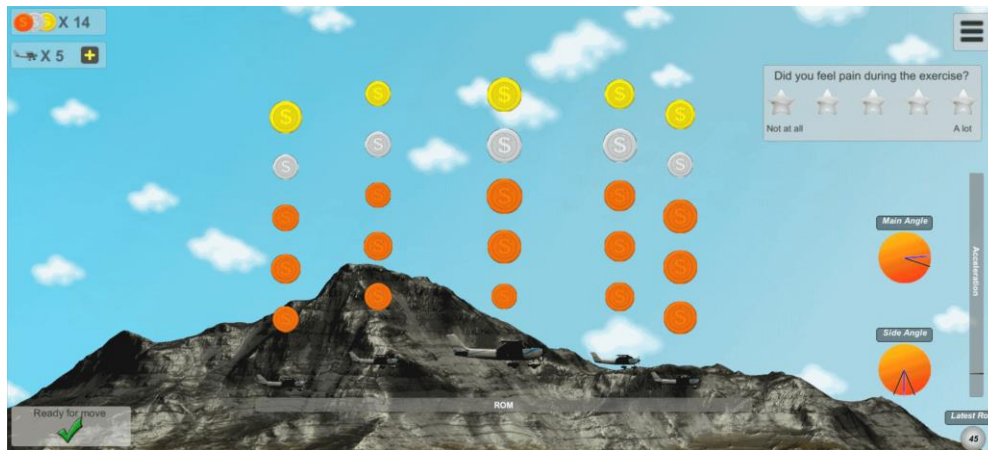


Figure 73: Multiple airplanes advanced game play feature.

When a user performs some incorrect movement additional effects can take place. In this case for example a meteor shower appears and destroys the smaller plains if it hits them (Figure 74). This way a player needs to add more effort in order to perform correct Knee Extension Repetitions, gather more coins and get the airplanes back.



Figure 74: Wrong Repetition Implementation.

A similarly implemented game is the Coin Fishes game. In this game there are fishes that gather coins in the horizontal direction instead of vertical as in the airplane game (Figure 75).

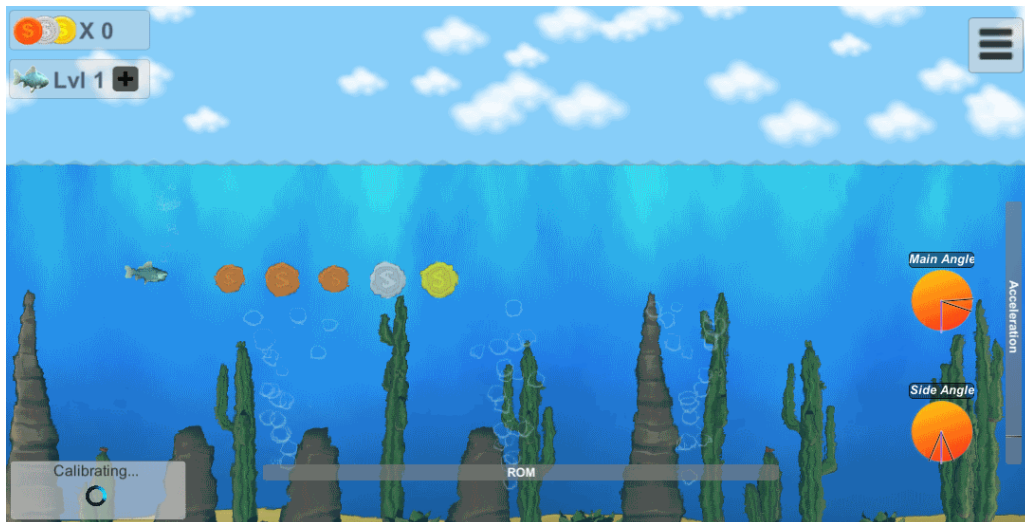


Figure 75: Fish Game.

There are 2 advanced features in this game. One is similar to the airplane game. There can be multiple fishes gathering coins. This however is triggered automatically if the user gathers enough coins. The next feature is that the user can increase the size of the fish if there are enough coins to spend (Figure 76).



Figure 76: Multiple fishes and larger fishes advanced game play features.

All the implemented mini-games employ several game assets. These assets are either visual content or game logic content. Visual content consists mainly of 3D mesh models (airplane, fish, mountains, plants), particle systems (cloud system, bubbles), Sprite Textures (Coins). Visual content is not created from scratch. It is imported through Unity packages. Game logic Content on the other hand is created from scratch. Game logic content constitutes of implemented scripts that are added to the 3D models as custom components attached to a GameObject, thus providing the desired functionality to the game. Such custom components are the Singletons described above. There are also lesser script components that add some

functionality that constitutes mostly on moving, rotating, enabling, disabling a specific GameObject. They do this by simply handling the transform component of a GameObject or by changing a flag value of a GameObject Component.

6 Chapter 6 – Experiments

6.1 Materials

This experiment was designed to track Total Knee Replacement (TKR) patients recovery after surgery using the proposed rehabilitation application. The goal was to motivate the patient to exercise correctly with minimum physiotherapist supervision. From a computer graphics point of view this experiment investigated whether a visual stimulus with elements of serious gaming could trigger the patient in order to perform the exercises appointed by the physiotherapists willingly and ignore the factor of pain up to a certain degree. Two groups of experiments were conducted. The first stage was essentially a training & parameter fine-tuning stage of the application and algorithm. It was conducted on healthy individuals as well as walking TKR patients on the path to recovery. The second stage took place at the Orthopaedic Clinic of the General Hospital of Chania. It was conducted on TKR patients after surgery.

6.1.1 Participants

The first training round was performed on 6 individuals (3 healthy ages 30 – 50, 3 walking TKR patients, ages 64 – 80). 10 patients (eight female, two male, range 64 - 80) underwent TKR surgery at General Hospital of Chania and agreed to participate in the second testing round of this framework that drove the results presented in this study. 1 patient refused to participate in this process. The patients had no background with Android devices or gaming.

6.1.2 Apparatus

Given the portability and low-cost design constraints of the application, the setup for the experiment was minimalistic. The designed sensor node was first placed in the shin of the patient's knee using an elastic bandage with Velcro (Figure 67). The goal was to stabilize the sensor node since Inertial Measurement Unit (IMUs) are very sensitive to placement misalignments. Then the patient was presented an Android device that run the designed application and was asked to follow the experimental procedure as described below. The selected device can be anything that meets the minimum hardware requirements for Android that Unity 3D specifies. These requirements are:

- OS 4.1 or later.
- ARMv7 CPU with NEON support or Atom CPU.
- OpenGL ES 2.0 or later.

6.2 Methods

6.2.1 Qualitative and Quantitative Research

Quantitative and qualitative research are commonly considered to differ fundamentally. Yet, their objectives as well as their applications overlap in numerous ways.

Quantitative Research is considered to have as its main purpose the quantification of data. This allows generalizations of results from a sample to an entire population of interest

and the measurement of the incidence of various views and opinions in a given sample. Yet, quantitative research is frequently followed by qualitative research which aims to explore the observed findings further.

Qualitative research is considered to be particularly suitable for gaining an in-depth understanding of underlying reasons and motivations. It provides insights into the setting of a problem. At the same time, it frequently generates ideas and hypotheses for later quantitative research.

Qualitative analysis involves a continual interplay between theory and analysis. In analyzing qualitative data, we seek to discover patterns such as changes over time or possible causal links between variables. Combining of qualitative and quantitative research is becoming more and more common. It is important to keep in mind that these are two different philosophies, not necessarily polar opposites. In fact, elements of both designs can be used together in mixed-methods studies.

Here we will primary employ qualitative research to try and understand patient engagement in the current framework. We will combine this qualitative research with quantitative findings such as ROM and classification rates. Although the quantity of these data is small they can provide a useful feedback in understanding patient engagement and recovery progress.

6.2.2 Experimental Procedure

The proposed framework implements 4 common rehabilitation exercises for the Total Knee Replacement (TKR) condition (Table 1). In order to simplify the testing procedure and reduce experiments duration, only one of these was used. The same procedure would apply to every exercise and the qualitative results acquired can be expected to share similar attributes with the ones acquired by a single exercise testing. Time constraints along with patient compliance could not allow testing all exercises. The selected exercise was Knee Extension. The patient positioned himself in a neutral sitting pose. He was presented with the Airplane Game as described in the Implementation Section. The goal was to raise the knee as high as possible and at the same time perform the exercise correctly and observe the visual stimulus. This stimulus was the movement of the airplane along with the movement of the knee as described in the *Game Levels Functionality* section. Patients were instructed to perform a minimum of 4 repetitions of this exercise. There was of course the possibility to perform more and stop when they feel tired or in pain. This procedure was common in both training and testing phases. Below the differentiations of each case are discussed.

6.2.3 Training – Healthy Subjects & Recovering TKR Patients

Firstly the application was applied to healthy subjects. This contributed in a first iteration of fine-tuning neutral position angle constraints for each exercise. A main issue when trying the application in healthy subjects is that these angle constraints are not representative for the case of TKR patients. Healthy subjects can have full ROM while TKR patients even at the stage of recovery have limited ROM of their knees. So a second iteration was deemed necessary in order to fine-tune angle constraints with respect to patient limited ROM.

Recovering TKR patient, is an outpatient in physio follow up, 12-14 days post-op after suture removal. This patient still has a limited ROM but can operate the knee normally and walk. We applied the same procedure to the recovering patient. An operated knee with limited ROM when bend has a neutral position angle smaller than a healthy subject angle. This fact allowed us to record new neutral pose angle constraints and automatic classification algorithm thresholds and make the appropriate changes in the respective implementations.

In both cases a single session was performed with each subject. Each subject only used the application one day compared to hospital inpatients where multiple data readings could be collected in a span of several days as described below. The duration of this training phase was 15 days.

Physiotherapist feedback during this training phase was invaluable as it helped finalize the experimental procedure and the reward system the game should provide, along with the gameplay mechanics functionality itself. They insisted that the coin reward system used should be as clear and simple as possible in order to be understandable by as many patients as possible independently of age criteria or other comorbidities. Also they noted that the movement of the knee for a specific exercise, should correspond to the visual stimulus in the virtual world. So for example, the upwards movement of the airplane corresponds to the upwards movement of the Knee Extension exercise. At this point the application was modified to tolerate limited ROM and the testing on the hospital orthopaedic clinic could commence.

6.2.4 Testing – TKR Inpatients

The initial testing of the application on patients after TKR surgery has been conducted in a similar manner at the Chania General Hospital under the supervision of a physiotherapist using the same postoperative procedure (Figure 77). All patients were operated by the same surgical team using the medial parà patellar approach and started their physiotherapy protocol 48h post op after the removal of the drain. Exclusion criteria were neurological deficit, previous operation on the ipsilateral or contralateral hip or knee, or functional deficit.



Figure 77: Chania General Hospital premises (<http://www.chaniahospital.gr/>).

10 TKR patients consented to try the application in order to gather motion data. TKR surgery is a common operation. The frequency of this operation in the public general hospital of Chania is approximately 5 patients per month. The data were collected in approximately 2 months of testing. A patient that underwent TKR surgery can stay in the hospital orthopaedic

clinic for a period of 4 days up to 12 days depending on recovery progress. A patient's recovery progress depends on the patient's age, physical condition and variant comorbidities. Younger patients tend to recover faster and people with obesity, respiratory problems or other comorbidities tend to recover slower.

We performed one to three sessions with each patient. A session is the testing of the application for a single day on a patient. In each session the patient is advised to perform a number of repetitions for the Knee Extension exercise. The first session for each patient was performed two to five days after surgery depending on recovery. Second session for the same patient, was performed four to five days after the first. Lastly the third session if any, was performed eight to nine days after the first. This could happen if a patient's recovery progress was slow and the patient had to stay longer in the hospital orthopaedic clinic. The number of the repetitions in a session can be four to thirteen depending on the patient's physical condition. Through these repetitions, motion data was collected by the application which in turn produces graphs in real-time that show statistics about the ROM of the patient. These statistics are used for data analysis and are presented further on.

6.3 Data Analysis

The focus is on the qualitative analysis of the data collected. At the current stage of the application, the focus is mainly on the tracking of recovery progress of each patient using ROM information extracted by the repetitions performed. Secondly an initial intuition along with preliminary results for the first iteration of the automated classification algorithm are provided.

6.3.1 ROM Graphs

In this section, the Range of Motion (ROM) percentage graphs generated by the application are presented. From the total of 10 patients, 4 were selected illustrating the variant cases that can occur. The data are presented in the following form:

- **Patient Id:** A unique number indicating the current patient.
 - **Gender:** Male or Female.
 - **Number of sessions:** Can be 1 – 3.
 - **Number of days after surgery:** Indicates when first session occurred.
 - **Comments:** Comments on health or reactions of the patient.
 - **Photo:** A photo of the patient when available.
 - **ROM graph:** The automatically generated graph of the application. The vertical axis is achieved ROM percentage (0% - 100%). The horizontal axis represents the number of repetitions performed grouped by specific dates. *Note:* Only the correct repetitions are used for ROM measurement, incorrect are discarded.

Below these results are presented:

- **Patient Id:** 1.
 - **Gender:** Female.
 - **Number of sessions:** 3.
 - **Number of days after surgery:** 2.

- **Comments:** Patient was immersed in the game. She tried to improve the score constantly. There is increasing progress during each session as seen in the graph. In the end of some sessions there are some decreasing ROM values. This is the outcome of the patient becoming tired while exercising.
- **Photo:**



Figure 78: Photo for patient 1.

- **ROM graph:**

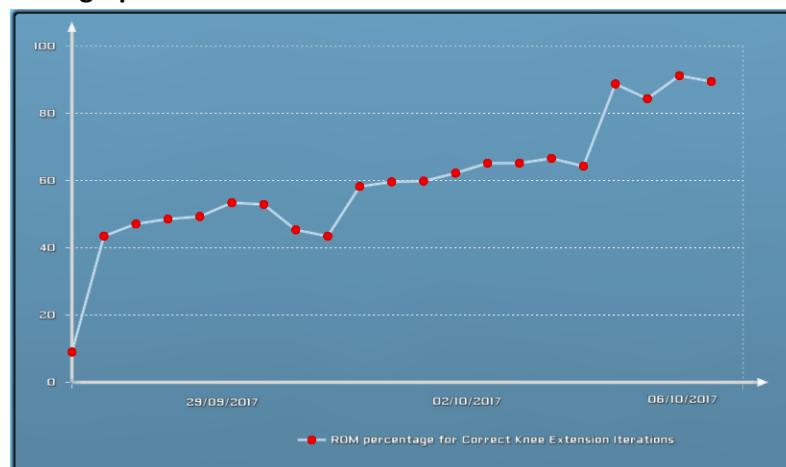


Figure 79: Generated ROM Graph for patient 1.

- **Patient Id: 2.**
 - **Gender:** Female.
 - **Number of sessions:** 3.
 - **Number of days after surgery:** 2.
 - **Comments:** Indicate declining progress. Patient was not immersed by the game and couldn't understand the goal. She performed the repetitions only because she was asked to do so. A minimum peak is an exercise with lower

ROM. Still it is a correct movement, but the patient performed some limited ROM repetitions in the process.

- **Photo:**



Figure 80: Photo for patient 2.

- **ROM graph:**

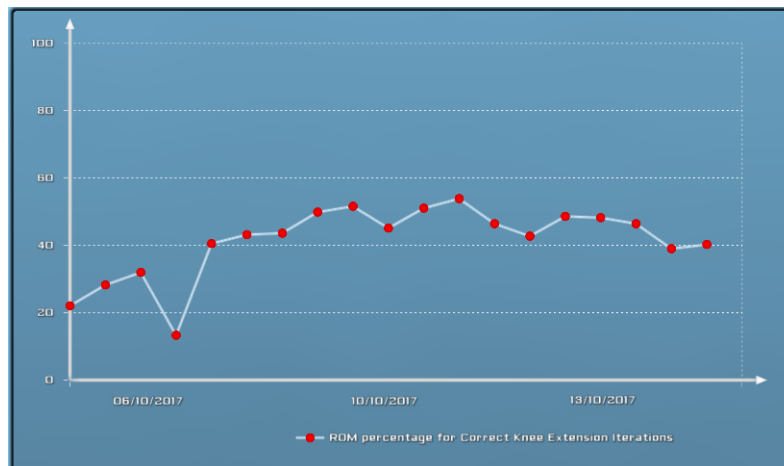


Figure 81: Generated ROM Graph for patient 2.

- **Patient Id: 4.**
 - **Gender:** Female.
 - **Number of sessions:** 1.
 - **Number of days after surgery:** 4.

- **Comments:** Stayed in the hospital orthopaedic clinic only a few days. Only a single session performed. Shows signs of engagement and constant effort to performed higher ROM repetitions. Last declining repetition indicates that the patient starts to become tired.
- **Photo:**



Figure 82: Photo for patient 4.

- **ROM graph:**

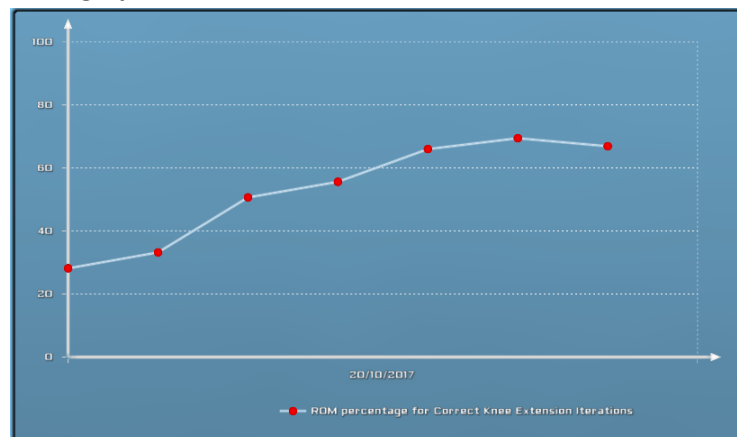


Figure 83: Generated ROM Graph for patient 4.

- **Patient Id: 8.**
 - **Gender:** Male
 - **Number of sessions:** 1.
 - **Number of days after surgery:** 5.
 - **Comments:** Younger patient with good physical condition. The achievable ROM was still limited but he actually scored high ROM by '*cheating*'. He

performed the repetitions by driving the whole body backwards along with the movement of the knee which was still bent. The sensor captures the angle in the axis of movement resulting in classifying the repetitions as correct while in fact they were incorrect.

- **Photo:** Not available.
- **ROM graph:**

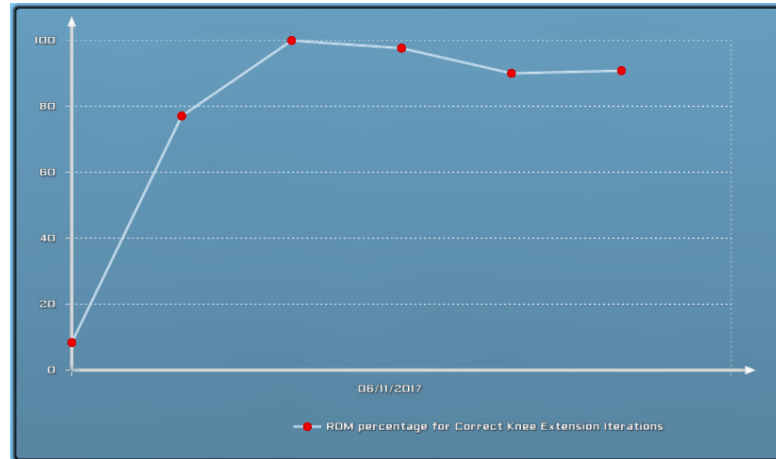


Figure 84: Generated ROM Graph for patient 8.

6.3.2 Classification Examples

In this section 2 graphs are presented that indicate the percentage of correct / incorrect repetitions for 2 patients respectively. These graphs are given in conjunction with the actual percentages for these cases. The actual percentage is essentially the physiotherapist estimation on how the movement should be performed. This way an intuition is provided on the first iteration of the classification algorithm and future improvements that are needed. The data presented here are discussed in the Results section and conform to the following format:

- **Patient Id:** A unique number indicating the current patient.
 - **Algorithm Classification Percentage:** 0 - 100 %.
 - **Physiotherapist Classification Percentage:** 0 - 100 %.
 - **Algorithm Classification graph:** The generated graph that indicates the Algorithm classification percentage.

Below these results are presented:

- **Patient Id: 2.**
 - **Algorithm Classification Percentage:** 73.1 %.
 - **Physiotherapist Classification Percentage:** 69.2 %.
 - **Algorithm Classification graph:**

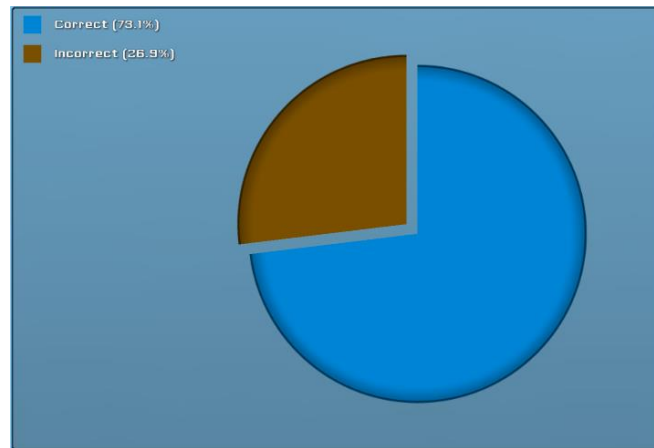


Figure 85: Classification Graph for patient 2.

- **Patient Id: 8.**
 - **Algorithm Classification Percentage:** 85.7 %.
 - **Physiotherapist Classification Percentage:** 16.6 %.
 - **Algorithm Classification graph:**

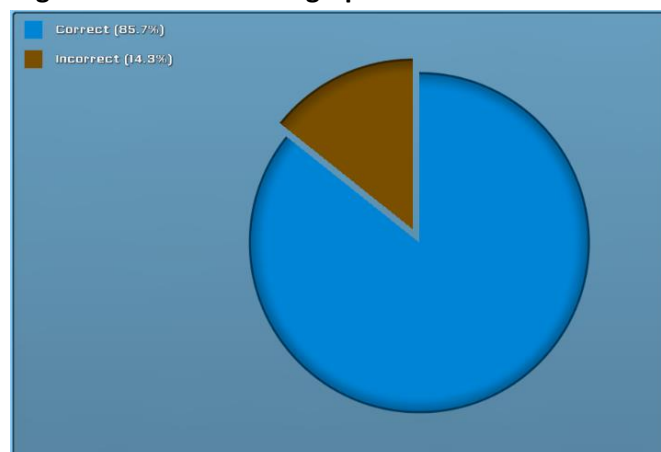


Figure 86: Classification ROM Graph for patient 8.

6.4 Results

In this chapter we will discuss the results that are provided in the Data Analysis section.

6.4.1 ROM Graphs

ROM graphs that are generated in the experimental procedure along with patient behavior provided feedback about the strengths and weaknesses of the proposed rehabilitation system. The majority of the patients understood the goal of the game and were immersed in the simple airplane scene. In 80% of the collected samples, the maximum ROM percentage measurements are increasing and follow a similar pattern as that of patient with id = 1 shown in Figure 79. This indicates that patients were constantly trying to improve their previous repetition performance by raising the airplane even higher and gather more coins. At

the end of certain sessions, the ROM measurements declined, in some cases, indicating that the patient was tired by the performed repetitions. This was mostly the case for patients demonstrating slower recovery due to other comorbidities. The slower recovery of these patients didn't necessarily result in lower engagement during the airplane game. Patients were still trying to improve their previous repetition 80% of the examined cases.

A minority of patients didn't improve their ROM (Figure 81). On these cases the patients didn't understand well the goal of the game and were not engaged with it. Therefore, patients only performed the repetitions because they were instructed to do so.

There was also a single case of a patient who performed a higher ROM repetition as recorded while the knee had actually a lower functional ROM. The patient achieved that by slowly positioning the entire body backwards in the direction of movement while sitting on a bed. The acceleration or angle constraints were not violated, so the maximum ROM percentage was measured and was close to 100% (Figure 84).

6.4.2 Classification Examples

While this testing was focused mainly on measuring the ROM of the patient, gathering feedback and recording their reactions, an initial evaluation is also provided regarding the implementation of the exercise classification algorithm in this first iteration. Automatically generated graphs show the detected classification percentage of the algorithm. The classification percentage indicates how many of the total repetitions performed for an exercise are classified as correct and therefore also indicates the repetitions classified as incorrect.

This percentage is compared with the actual classification percentage that is provided by the supervising physiotherapist. Then the error percentage of the algorithm can be measured by comparing the actual physiotherapist percentage with the application's generated percentage using the relative difference of the values. One way to define the relative difference of two numbers is to take absolute difference divided by the maximum absolute value of the two numbers. So the error percentage is described by the following formulae (Figure 87):

$$\%Error = \frac{|AlgorithmCorrect - PhysiotherapistCorrect|}{\max(|AlgorithmCorrect|, |PhysiotherapistCorrect|)} \times 100$$

Figure 87: Formulae for error percentage calculation of designed algorithm.

For instance, a patient achieving a correct classification percentage of 73.1% according to the automated algorithm (Figure 85), that has an actual correct percentage of 69.2% according to the physiotherapist, provides an algorithm error percentage of 5.6% for this patient. For a different patient who employed a strategy of demonstrating higher ROM than possible based on body positioning as explained above, the respective percentages were an algorithm correct percentage of 85.7% (Figure 86), a physiotherapist's actual percentage of 14.3%, resulting in an algorithmic error percentage of 83.3%. The observed data indicates that there can exist large variation in resulting accuracy of the algorithm. It is challenging to implement a generalized algorithm that avoids overfitting based on having only a few samples

during these two months of testing. Future iterations should optimally rely on a larger amount of data including data from different hospitals.

Detecting ‘cheating’ patient strategies when employing a single IMU sensor is significant. For such cases, there should be a minimum of two sensors, one placed on the shin and one on the thigh of the patient. However, this increases the cost of the application and reduces portability which were the main design constraints of the proposed system. There are two supplementary ways to resolve this issue. Firstly, the physiotherapist should train the patient to accurately perform the exercise such as the knee extension exercise so that, later, the patient can repeat the exercises alone with minimum supervision. Secondly, the user interface of the application should instruct the patient to maintain a neutral pose. This should be preferably implemented to include animations of humanoid postures rather than only text and image hints. Therefore, it is essential to provide a solid setup for the application. For example, while a knee extension exercise can be performed in bed, using a chair to sit on instead can avoid this form of ‘cheating’ by helping the patient to support the back and accurately exercise.

Throughout the experimental procedure it became apparent that the classification error is greatly influenced by the starting position of the sensor. It is essential that the sensor is accurately oriented so that side angle is close to 0° degrees in order to avoid inaccurate exercise repetitions. The IMU is very sensitive to misalignments. If the sensor is not placed correctly, the data collected from the sensor can lead to inaccurate results.

6.5 Comments

Insight gained throughout the experimental procedure while working with TKR patients, is note-worthy. When the patients were asked to perform an exercise based on the proposed rehabilitation system, they were reluctant or hesitant at best. One of them even refused to perform the experiment. Once the orthopedic surgeon or physiotherapist explained the procedure, they cooperated well and contributed to this research project although still hesitant in some cases. It was common for a patient to focus on the experienced pain before embarking on the experimental procedure. This fact negatively influenced the psychological state of the patient. After performing a single repetition, though, the patient was usually more tolerant of pain of the knee, ignoring it to a certain degree. There were many cases when after only a few repetitions, patients were heavily engaged in the game as indicated by their behavior and facial expressions. Examples of such expressions were smiling facial expressions, laughing or eagerness to start the next repetition. On the other hand, this eagerness could result in hasty movement and, thus, incorrect repetitions. In such cases, the patient was advised to perform slower and steadier movements of the knee. After a certain amount of repetitions, the pain was again more dominant than user engagement and the ROM percentage started to decline. By the end of the session, 80% of the participants were eager to perform another session.

Physiotherapist feedback was invaluable in both testing and training phases. As discussed in this section, during the testing phase in recovering patients, physiotherapist requirements helped in simplifying the design of the gameplay and reward procedure of the game. During this testing phase in inpatients, physiotherapists informed the patients on the experimental procedure which greatly helped in gaining their consent to proceed with our

measurements. Additionally they helped in sensor placement at the shin of the knee. They instructed the patients how to perform correct exercise repetitions using correct body posture and movement of the knee. When a patient needed help to perform a repetition they provided manual aid in the movement of the knee. Despite the physiotherapist guidance, there were cases of patients performing wrong repetitions or ‘cheating’. Still these cases give valuable feedback about the designed system response and provide some insight about future improvements needed such as more user friendly visual hints and improved UX.

When patients were asked if they had found the games helpful, only one patient doubted their effectiveness for training, although physiotherapists recorded noticeable improvement of the patient’s limb motion. Other patients realized they were applying enhanced personalized effort into their physiotherapy protocol when utilizing the gamified application. Another thing to note is that none of the participants used advanced gameplay features, e.g. buy more airplanes. All the participants were not acquainted with mobile technology, android devices and gaming. They participated however in the game in its basic form.

6.6 Discussion

We analyzed the data collected and interpreted the results. The goal is to understand in a qualitative manner the feedback collected from the patients. In this iteration quantitative measures that include ROM and classification feedback were provided. The collected data have a small quantity and cannot provide generalized results in terms of numbers, but nonetheless provide qualitative feedback for understanding user and physiotherapist expectations for the application.

Physiotherapists expect the patient to be able to understand the designated exercises and be able to perform them correctly with minimal guidance. Physiotherapists are also interested in the generated application graphs that can be a useful tool for monitoring patient recovering progress. Future improvement of the designed system, can include a mechanism for real-time measurement monitoring. The data gathered from Bluetooth can be sent via Wifi to a remote database. Then a designed web service can access this database and produce the same graphs for different users. So when a user is exercising from home or effectively from anywhere, the physiotherapist will be able to view a user’s progress remotely, provided that the user has access to the internet.

The user of the application expects fast recovery of the knee with minimum achievable pain in the process and minimum physiotherapist supervision. The goal of this application however is not to replace the physiotherapist. The monitoring of the physiotherapist is required in any case. Assuming that the aforementioned online monitoring is implemented, an interesting feature of this framework can be that a user can request the physiotherapist feedback on demand through the implemented application. The WiFi restriction still applies. This however is an advanced feature that should be designed carefully because as observed in the current experimental procedure, the user can have little involvement with mobile technology. So features like this should be designed in a way that both experienced and inexperienced mobile users can understand: Experienced users by employing Wi-Fi technology

and inexperienced users possibly through direct mobile network communication by enabling direct call to or from a supervising physiotherapist or orthopaedic.

The testing sessions were focused in providing feedback on the user engagement of this framework. As mentioned above, user expressions and notions through the exercise indicate a positive attitude towards this gamified approach in physiotherapy. A percentage of the users though, roughly 20% of them, failed to get immersed in the designed environment. This was indicated by their lack of attention or interest during the experimental procedure as well as from the resulting ROM graphs that were declining. These users performed the repetitions only because they were told to do so in contrast with immersed users that performed the repetitions willingly until they were instructed to stop or they were getting tired from the exercise.

A solution to this situation can be provided using the notion of personalization. So instead of testing only one game for a single exercise, try multiple games and observe if there are changes in user engagement. In preliminary testing of the proposed framework, some prototype games were tested. An example of such a game was something visually simple such as scaling a 3D Sphere using the movement of the knee. This kind of game was actually more helpful in some patients than more complex games. A more complex game introduced an animated character that was climbing a wall using the movement of the knee. This was a highly personalized example that a certain user might enjoy a lot, but another user cannot understand it that easily.

The airplane game was chosen for the experimental procedure as a more generalized and neutral game. The games designed should be simple and understandable. It is important to keep in mind that the users of this application are unrelated with technology and gaming. Additionally the majority of the users belong to an older age group (above 60 years old). It is not a trivial task to design games that target these age groups. One important constraint is that a user should be willing to undergo this kind of gaming procedure. Without the help of the majority of the patients these results would completely fail in assessing user engagement.

7 Chapter 7 – Conclusions & Future Work

Wearable technologies over the past years have been emerging for wide use in everyday activities. The need of utilizing wearable sensors is receiving wide attention, mostly for applications that exploit wearable sensors in order to improve quality of life. The current framework introduces an ultra-portable rehabilitation application comprising of just a single Inertial Measurement Unit (IMU) sensor linked to a 3D gamified environment, to be adopted by patients that have undergone Total Knee Replacement (TKR) surgery for their highly repetitive, but very significant post-operative physiotherapy. This application engages the patient to accurately perform the recovery exercises through a gamified 3D experience, ultimately minimizing physiotherapist supervision, at most locations. A 3D mini-game connected to each limb exercise appointed by the physiotherapist is implemented. This framework can run on Android smartphones with the use of a single sensor node maximizing portability and ease of use. Accurate patient exercise and compliance can be achieved by succeeding at each mini-game objective. Feedback offered based on angle measurement constraints enhances patient engagement. Future development should overcome limitations of few testing samples and derive a reliable accuracy result from the classification process. Furthermore a long term observation of TKR outpatients is required in order to provide meaningful and reliable results.

7.1 Limitations

Throughout this project there were several limitations during the development phase as well as the experimental phase. Recording these limitations can be really helpful to provide some insight on future improvements of this framework as well as improvements on the experimental procedure. These improvements will also lead to additional test phases design that can produce generalized results regarding physiotherapy of TKR patients using gamified technologies.

To begin with, the design constraints criteria selected, already impose very specific limitations. Note that the main design constraints of this framework are portability and low cost. Using 2 sensor nodes instead of one could achieve better motion tracking, but would violate our main design constraints. Future development will also respect these constraints. While employing a second sensor node can be more accurate (Huang et al. 2016), the goal is to obtain the optimal achievable result using 1 sensor node thus maximizing portability and minimizing cost.

A very significant factor and at the same time limitation of the experimental procedure is the positioning of the sensor node. IMUs are very sensitive to misalignments. The starting position of the sensor node is critical when measuring ROM and classifying a repetition as correct / incorrect. During the experimental procedure the placement was performed in a supervised manner by a physiotherapist. Eventually when a patient uses the application at home or anyplace unsupervised, that patient should be able to position the sensor node correctly to the shin of the knee and have the correct pose at the same time. Although the designed UI indicates the main and side angle rotation and the allowed margins, people with limited knowledge of technology may have trouble understanding what to do. This fact

indicates the need for a more visually descriptive UI rich in animations that will be understandable by the majority of the users if not all of them.

Another point to consider is the experimental process itself. It took place at the Chania General Hospital Orthopaedic Clinic. Although the results gained through the procedure would not be possible without the consent of the Hospital, it is very time consuming to gather just a few measurements on a number of patients. The TKR procedure is a common procedure, but in smaller communities like the city of Chania there are just 1-2 TKR patients per week compared to larger city hospitals that can have multiplied rate of operations per week. Gathering measurements from a second hospital would allow us to compute concrete quantitative results and have more diverse samples. Another difficulty is that although we tried to gather measurements at fixed time intervals in order to be more consistent with the results, this was not always possible. This is due to each patient having variable recovering times. So while one patient was available for measurement in the 2nd day after surgery, another patient would be available in the 3rd day after surgery.

The testing in the hospital as mentioned in the Methods section focused mainly in the ROM measurement and initial reactions of patients. Ideally though we want a testing procedure that will compare traditional physiotherapy with the proposed gamified one. To achieve this, there should be testing done that will track patients for a period of time even after they have exited the hospital. It will also require patient dedication on performing either therapy (traditional or gamified) assigned to them. Again this cannot easily be done in a smaller municipality like this of Chania and not every patient will be willing to agree to more long term tracking. To achieve such comparisons, measurements from larger municipalities will be required.

Furthermore, the testing in the hospital provided some insight on the first iteration of the automatic classification algorithm of physiotherapy exercises. Throughout the testing phase the algorithm parameters were fine-tuned manually to provide more accurate results. Still this process should be generalized. This generalization can be provided by gathering more measurements from more patients and applying machine learning techniques to compute algorithm parameters.

7.2 Implications for Future Work

In the section above, the major limitations of the current work were discussed. Following this discussion some ways to overcome these limitations are required. Here we will briefly overview these ways and how they can be integrated to the current project.

More effort should be put to improve User Experience (UX) of the application. The application should be used with the same ease by experienced and inexperienced mobile users and by users of any Age Group. Implementing visual hints, animations and strict user instructions can make the proposed framework more user friendly even for inexperienced users. It is not possible to satisfy all the users, but it is achievable to satisfy the majority of the TKR patients.

A difficult problem to address is to increase the user engagement for diverse populations. These can be achieved by the notion of personalization as mentioned in the

experiment section above. So the experimental procedure should change in a way that different users have more than one game to choose for each TKR exercise. This way there is increased probability that a person will be immersed to at least one game. The experimental procedure then can provide more meaningful results. This project already implements variant TKR games. More mini-games on various topics can be designed and implemented that can increase personalization and incorporated in the experimental procedure in future testing.

The online monitoring feature is already described in the results section. This feature along with more graph generations and quantitative data, can provide the supervising physiotherapist valuable data on the patient recovery progress and ROM improvement. Additionally the patient can have the option to consult the physiotherapist through the application and request feedback when there are questions concerning the recovery period and performed exercises.

Finally the focus should be on testing the application and comparing it with traditional physiotherapy. This, as mentioned in the limitations section, requires tracking patients for longer periods of time after TKR rehabilitation, for example up to 5 weeks. When we ensure that this is feasible and patients are available to cooperate for that long, the second round of clinical trials can commence. Clinical trial will include 20 randomly selected subjects that have undergone TKR performed by the orthopaedic surgical team. Patients should be randomly selected to receive a course of 10 treatments over 5 weeks, half of them will use a traditional set of exercises for post op rehabilitation and the other half will use the gamification process. By the end of the rehabilitation we will measure and compare parameters such as Range of Motion (ROM), gait speed and overall patient satisfaction in order to establish the potential benefits of the interactive approach.

The ultimate goal of this project would be to form the base of a commercial low-cost, portable mobile application. A very important factor towards this goal would be the replacement of the custom-made sensor with a commercial sensor. The cost of the custom sensor implemented in this project is 30 \$. It would be an interesting project to introduce such a sensor in the market, although at the time of writing there isn't a device incorporating only IMUs. Usually IMUs are embedded in a wearable device along with additional functionalities. This is the case of the smartwatches. A smartwatch is a touchscreen wearable computer in the form of a wristwatch. Smartwatches incorporate IMUs additionally with more systems such as heart rate, proximity sensor, GPS, speakers etc. This raises the cost of the application since the cheapest smartwatches that incorporate IMUs come at a cost of 130\$ at the time of writing. In the near future though this cost can be even smaller. Already at this cost the wearable watches are the strongest candidate for replacing the custom made sensor.

A low-cost smartwatch on this category is Ticwatch E. Ticwatch at the time of writing is the cheapest smartwatch that uses Wear OS (130\$). Wear OS, formally known as Wear OS by Google, and previously known as Android Wear, is a version of Google's Android operating system designed for smartwatches and other wearables. By pairing with mobile phones running Android version 4.3 or newer, or iOS version 8.2 or newer with limited support from Google's pairing application, Wear OS integrates Google Assistant technology and mobile notifications into a smartwatch form factor. Wear OS supports Bluetooth, Wi-Fi, 3G and LTE

connectivity, as well as a range of features and applications. In the first six months of availability, it is estimated that over 720,000 Android Wear smartwatches were shipped. As of 15 March 2018, Wear OS had between 10 and 50 million application installations. Wear OS was estimated to account for 10% of the smart watch market in 2015.



Figure 88: Ticwatch E smartwatch (<https://www.mobvoi.com/>).

Ticwatch E features include, Heart rate Monitor, Proximity Sensor, Accelerometer, Gyroscope, E-Compass, GLONASS, GPS. Google Assistant, Google Play, Sport and Fitness app, calorie counter, distance counter, pedometer, social app notifications, call and message notifications. Although the current work is implemented to be independent of the hardware, it is not a straight forward task to replace the custom made sensor with Ticwatch E. The main technical issue that needs to be addressed regards the integration of Wear OS with Unity Game Engine. As is the case in the current implementation a platform specific unity plugin must be implemented that handles Bluetooth connection between the smartwatch sensor with both Android and iOS devices through Wear OS functionality. On the other hand, as soon as this connection is established the data received from the smartwatch IMU will be already filtered or in worst case will require minimal post processing. Then the exercise classification algorithm can be applied.

Recent works (Bevilacqua et al. 2018) study the automatic classification of knee rehabilitation exercises using a single inertial sensor as a continuation of previous studies in the same topic (Giggins et al. 2014, Huang et al. 2016). The proposed methodology employs machine learning techniques to automatically analyze inertial measurement unit data collected during these exercises, and then assess whether each repetition of the exercise was executed correctly or not. It would be a useful addition for this methodology to be incorporated to the current framework and validated from the collected dataset. Still the dataset remains small in size. Incorporating a smartwatch can provide a solution to this issue mainly for two reasons: a) The Smartwatch can allow data to be gathered from a lot of healthy users and patients providing variant samples among population and b) Greater number of identical smartwatches can be configured for each user / patient, in order to enable long term monitoring of users providing solid results for the gamified rehabilitation approach.

8 References – Bibliography

- Buonocunto & Marinoni 2014 Buonocunto, P., & Marinoni, M. (2014). Tracking limbs motion using a wireless network of inertial measurement units. In *Industrial Embedded Systems (SIES), 2014 9th IEEE International Symposium on* (pp. 66-76). IEEE.
- Wu et al. 2013 Wu, H. K., Lee, S. W. Y., Chang, H. Y., & Liang, J. C. (2013). Current status, opportunities and challenges of augmented reality in education. *Computers & education*, 62, 41-49.
- Milgram et al. 1995 Milgram, P., Takemura, H., Utsumi, A., & Kishino, F. (1995). Augmented reality: A class of displays on the reality-virtuality continuum. In *Telemanipulator and telepresence technologies* (Vol. 2351, pp. 282-293). International Society for Optics and Photonics.
- Azuma 1997 Azuma, R. T. (1997). A survey of augmented reality. *Presence: Teleoperators & Virtual Environments*, 6(4), 355-385.
- Klopfer 2008 Klopfer, E. (2008). *Augmented learning: Research and design of mobile educational games*. MIT press.
- Klopfer & Squire 2008 Klopfer, E., & Squire, K. (2008). Environmental Detectives—the development of an augmented reality platform for environmental simulations. *Educational Technology Research and Development*, 56(2), 203-228.
- Klopfer & Sheldon 2010 Klopfer, E., & Sheldon, J. (2010). Augmenting your own reality: Student authoring of science-based augmented reality games. *New directions for youth development*, 2010(128), 85-94.
- Dunleavy et al. 2009 Dunleavy, M., Dede, C., & Mitchell, R. (2009). Affordances and limitations of immersive participatory augmented reality simulations for teaching and learning. *Journal of science Education and Technology*, 18(1), 7-22.
- Shen et al. 2008 Shen, Y., Ong, S. K., & Nee, A. Y. C. (2008). An augmented reality system for hand movement rehabilitation. In *Proceedings of the 2nd International Convention on Rehabilitation Engineering & Assistive Technology* (pp. 189-192). Singapore Therapeutic, Assistive & Rehabilitative Technologies (START) Centre.
- Zizka et al. 2011 Zizka, J., Olwal, A., & Raskar, R. (2011). SpeckleSense: fast, precise, low-cost and compact motion sensing using laser speckle. In *Proceedings of the 24th annual ACM symposium on User interface software and technology* (pp. 489-498). ACM.
- Solazzi et al. 2010 Solazzi, M., Frisoli, A., & Bergamasco, M. (2010). Design of a novel finger haptic interface for contact and orientation display. In *Haptics Symposium, 2010 IEEE* (pp. 129-132). IEEE.
- Iqbal & Baizid 2015 Iqbal, J., & Baizid, K. (2015). Stroke rehabilitation using exoskeleton-based robotic exercisers: Mini Review. *Biomedical Research*, 26(1).

- Menache 2000
Menache, A. (2000). *Understanding motion capture for computer animation and video games*. Morgan kaufmann.
- Harada et al. 2004
Harada, T., Mori, T., & Sato, T. (2004). Human posture probability density estimation based on actual motion measurement and eigenpostures. In *Systems, Man and Cybernetics, 2004 IEEE International Conference on* (Vol. 2, pp. 1595-1600). IEEE.
- Roetenberg et al. 2009
Roetenberg, D., Luinge, H., & Slycke, P. (2009). Xsens MVN: full 6DOF human motion tracking using miniature inertial sensors. *Xsens Motion Technologies BV, Tech. Rep, 1*.
- Shen et al. 2008
Shen, Y., Ong, S. K., & Nee, A. Y. C. (2008). An augmented reality system for hand movement rehabilitation. In *Proceedings of the 2nd International Convention on Rehabilitation Engineering & Assistive Technology* (pp. 189-192). Singapore Therapeutic, Assistive & Rehabilitative Technologies (START) Centre.
- Cameirão et al. 2011
da Silva Cameirão, M., Bermúdez i Badia, S., Duarte, E., & Verschure, P. F. (2011). Virtual reality based rehabilitation speeds up functional recovery of the upper extremities after stroke: a randomized controlled pilot study in the acute phase of stroke using the rehabilitation gaming system. *Restorative neurology and neuroscience*, 29(5), 287-298.
- Field et al. 2013
Field, M., Stirling, D., Ros, M., Pan, Z., & Naghdy, F. (2013). Inertial sensing for human motor control symmetry in injury rehabilitation. In *Advanced Intelligent Mechatronics (AIM), 2013 IEEE/ASME International Conference on* (pp. 1470-1475). IEEE.
- Crocher et al. 2013
Crocher, V., Hur, P., & Seo, N. J. (2013). Low-cost virtual rehabilitation games: House of quality to meet patient expectations. In *Virtual Rehabilitation (ICVR), 2013 International Conference on* (pp. 94-100). IEEE.
- Rand et al. 2013
Rand, D., Schejter-Margalit, T., Dudkiewicz, I., Kizony, R., & Zeilig, G. (2013). The use of the iPad for poststroke hand rehabilitation; a pilot study. In *Virtual Rehabilitation (ICVR), 2013 International Conference on* (pp. 109-113). IEEE.
- Lee et al. 2013
Lee, K., Lin, H. C., Kan, Y. C., & Chiang, S. Y. (2013). A pilot study of activity recognition on rehabilitation exercise of frozen shoulder using wireless inertial sensor node. In *Medical Information and Communication Technology (ISMICT), 2013 7th International Symposium on* (pp. 117-120). IEEE.
- Giggins et al. 2013
Giggins, O., Kelly, D., & Caulfield, B. (2013). Evaluating rehabilitation exercise performance using a single inertial measurement unit. In *Proceedings of the 7th International Conference on Pervasive Computing Technologies for Healthcare* (pp. 49-56). ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).
- Bo et al. 2011
Bo, A., Hayashibe, M., & Poinet, P. (2011). Joint angle estimation in rehabilitation with inertial sensors and its integration with Kinect. In *EMBC: Engineering in Medicine*

- Ambar et al. 2012
Ambar, R. B., Poada, H. B. M., Ali, A. M. B. M., Ahmad, M. S. B., & Jamil, M. M. B. A. (2012). Multi-sensor arm rehabilitation monitoring device. In *Biomedical Engineering (ICoBE), 2012 International Conference on* (pp. 424-429). IEEE.
- Lee et al. 2014
Lee, W. W., Yen, S. C., Tay, E. B. A., Zhao, Z., Xu, T. M., Ling, K. K. M., ... & Huat, G. K. C. (2014). A smartphone-centric system for the range of motion assessment in stroke patients. *IEEE journal of biomedical and health informatics*, 18(6), 1839-1847.
- Hundza et al. 2014
Hundza, S. R., Hook, W. R., Harris, C. R., Mahajan, S. V., Leslie, P. A., Spani, C. A., ... & Livingston, N. J. (2014). Accurate and reliable gait cycle detection in Parkinson's disease. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 22(1), 127-137.
- Hu et al. 2014
Hu, W., Charry, E., Umer, M., Ronchi, A., & Taylor, S. (2014). An inertial sensor system for measurements of tibia angle with applications to knee valgus/varus detection. In *Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2014 IEEE Ninth International Conference on* (pp. 1-6). IEEE.
- Wang et al. 2011
Wang, P. T., King, C. E., Do, A. H., & Nenadic, Z. (2011). A durable, low-cost electrogoniometer for dynamic measurement of joint trajectories. *Medical engineering & physics*, 33(5), 546-552.
- Bachmann et al. 2001
Bachmann, E. R., McGhee, R. B., Yun, X., & Zyda, M. J. (2001). Inertial and magnetic posture tracking for inserting humans into networked virtual environments. In *Proceedings of the ACM symposium on Virtual reality software and technology* (pp. 9-16). ACM.
- Mahony et al. 2008
Mahony, R., Hamel, T., & Pflimlin, J. M. (2008). Nonlinear complementary filters on the special orthogonal group. *IEEE Transactions on automatic control*, 53(5), 1203-1218.
- Madgwick et al. 2011
Madgwick, S. O., Harrison, A. J., & Vaidyanathan, R. (2011). Estimation of IMU and MARG orientation using a gradient descent algorithm. In *Rehabilitation Robotics (ICORR), 2011 IEEE International Conference on* (pp. 1-7). IEEE.
- Giggins et al. 2014
Giggins, O. M., Sweeney, K. T., & Caulfield, B. (2014). Rehabilitation exercise assessment using inertial sensors: a cross-sectional analytical study. *Journal of neuroengineering and rehabilitation*, 11(1), 158.
- Khalil et al. 2016
Khalil, Y. A., Hu, B., & Mishra, S. Attitude Estimation with Inertia/Magnetic Sensors.
- Convery & Beber 1973
Convery, F. R., & Beber, C. A. (1973). Total knee arthroplasty: indications, evaluation and postoperative management. *Clinical Orthopaedics and Related Research*®, 94, 42-49.
- Martin et al. 2014
Martin, G. M., Thornhill, T. S., & Katz, J. N. (2014). Total knee arthroplasty. *UpToDate*.
- Tipping 2003
Tipping, M. (2003). *U.S. Patent No. 6,633,857*. Washington, DC: U.S. Patent and Trademark Office.

- Huang et al. 2016 Huang, B., Giggins, O., Kechadi, T., & Caulfield, B. (2016). The limb movement analysis of rehabilitation exercises using wearable inertial sensors. In *Engineering in Medicine and Biology Society (EMBC), 2016 IEEE 38th Annual International Conference of the* (pp. 4686-4689). IEEE.
- Pirovano et al. 2016 Pirovano, M., Surer, E., Mainetti, R., Lanzi, P. L., & Borghese, N. A. (2016). Exergaming and rehabilitation: A methodology for the design of effective and safe therapeutic exergames. *Entertainment Computing*, 14, 55-65.
- Wouters et al. 2013 Wouters, P., Van Nimwegen, C., Van Oostendorp, H., & Van Der Spek, E. D. (2013). A meta-analysis of the cognitive and motivational effects of serious games. *Journal of educational psychology*, 105(2), 249.
- Tsekleves et al. 2014 Tsekleves, E., Warland, A., Kilbride, C., Paraskevopoulos, I., & Skordoulis, D. (2014). The use of the Nintendo Wii in motor rehabilitation for virtual reality interventions: a literature review. In *Virtual, Augmented Reality and Serious Games for Healthcare 1* (pp. 321-344). Springer, Berlin, Heidelberg.
- Levin et al. 2012 Levin, M. F., Snir, O., Liebermann, D. G., Weingarden, H., & Weiss, P. L. (2012). Virtual reality versus conventional treatment of reaching ability in chronic stroke: clinical feasibility study. *Neurology and therapy*, 1(1), 3.
- Paraskevopoulos et al. 2014 Paraskevopoulos, I. T., Tsekleves, E., Craig, C., Whyatt, C., & Cosmas, J. (2014). Design guidelines for developing customised serious games for Parkinson's Disease rehabilitation using bespoke game sensors. *Entertainment Computing*, 5(4), 413-424.
- Varela et al. 2012 Varela, D. G., Carneiro, J. A. O., & Colafêmina, J. F. (2012). Static postural balance study in patients with vestibular disorders using a three dimensional eletromagnetic sensor system. *Brazilian journal of otorhinolaryngology*, 78(3), 7-13.
- Chang et al. 2011 Chang, Y. J., Chen, S. F., & Huang, J. D. (2011). A Kinect-based system for physical rehabilitation: A pilot study for young adults with motor disabilities. *Research in developmental disabilities*, 32(6), 2566-2570.
- Kalman 1960 Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 82(1), 35-45.
- Grewal 2011 Grewal, M. S. (2011). Kalman filtering. In *International Encyclopedia of Statistical Science* (pp. 705-708). Springer, Berlin, Heidelberg.
- Eberly 2006 Eberly, D. H. (2006). *3D game engine design: a practical approach to real-time computer graphics*. CRC Press.
- Gregory 2009 Gregory, J. (2009). *Game engine architecture*. CRC Press.
- Parvizi et al. 2008 Parvizi, J., Azzam, K., & Rothman, R. H. (2008). Deep venous thrombosis prophylaxis for total joint arthroplasty: American Academy of Orthopaedic Surgeons guidelines. *The Journal of arthroplasty*, 23(7), 2-5.
- Stevens-Lapsley et al. 2012 Stevens-Lapsley, J. E., Balter, J. E., Wolfe, P., Eckhoff, D. G., & Kohrt, W. M. (2012). Early neuromuscular electrical stimulation to improve quadriceps muscle strength after total knee arthroplasty: a randomized controlled

- Harmer et al. 2009 trial. *Physical Therapy*, 92(2), 210-226.
 Harmer, A. R., Naylor, J. M., Crosbie, J., & Russell, T. (2009). Land-based versus water-based rehabilitation following total knee replacement: A randomized, single-blind trial. *Arthritis Care & Research*, 61(2), 184-191.
- Harmelink et al. 2017 Harmelink, K. E., Zeegers, A. V. C. M., Tönis, T. M., Hullegie, W., & Staal, J. B. (2017). The effectiveness of the use of a digital activity coaching system in addition to a two-week home-based exercise program in patients after total knee arthroplasty: study protocol for a randomized controlled trial. *BMC musculoskeletal disorders*, 18(1), 290.
- Fung et al. 2012 Fung, V., Ho, A., Shaffer, J., Chung, E., & Gomez, M. (2012). Use of Nintendo Wii Fit™ in the rehabilitation of outpatients following total knee replacement: a preliminary randomised controlled trial. *Physiotherapy*, 98(3), 183-188.
- Bevilacqua et al. 2018 Bevilacqua, A., Huang, B., Argent, R., Caulfield, B., & Kechadi, T. (2018). Automatic Classification of Knee Rehabilitation Exercises Using a Single Inertial Sensor: a Case Study.
- Wikipedia <https://www.wikipedia.org/>
 Chania Hospital <http://www.chaniahospital.gr/>
 Vicon <https://www.vicon.com/>
 Kinect <https://www.xbox.com/en-US/xbox-one/accessories/kinect/>
 Polhemus <http://polhemus.com/>
 Raspberry Pi <https://www.raspberrypi.org/>
 Arduino <https://www.arduino.cc/>
 MPU-9150 <https://www.invensense.com/products/motion-tracking/9-axis/mpu-9150/>
- TKJ Electronics <http://blog.tkjelectronics.dk/2012/09/a-practical-approach-to-kalman-filter-and-how-to-implement-it/>
- Unity 3D <https://unity3d.com/>
<https://docs.unity3d.com/>
<https://www.assetstore.unity3d.com/>
- Unreal Engine <https://www.unrealengine.com/>
 Urho 3D <https://urho3d.github.io/>
 Android Studio <https://developer.android.com/>
 OrthoInfo <https://orthoinfo.aaos.org/>
 Study <https://study.com/>
 Medigauge <http://www.medigauge.com/>
 Jaipur Joint Replacement <http://www.jaipurjointreplacement.com/>
 SportsMed <http://sportsmed.in/>
 Nmes Quad Rehab Post Knee Surgery <http://nmesquadrehabpostkneesurgery.blogspot.com/>
 Mobvoi <https://www.mobvoi.com/>