University of Southern Queensland

Faculty of Engineering and Surveying

Design and Develop Virtual Reality Games Utilising the "Anti-gravity" Arm Support for Stroke Rehabilitation Therapy

A dissertation submitted by

Ryan Fitzhenry

In fulfilment of the requirements of

Courses ENG4111 and 4112 Research project

Towards the degree of

Bachelor of Engineering (Computer Systems)

Submitted: October, 2009

Abstract

Approximately 16,000 Australians each year are left with a disability as a direct consequence of stroke. The number of strokes that occur in Australia each year is increasing, putting a strong reliance on home and community based rehabilitation having an increasing role in the rehabilitation process.

Strokes are caused by a sudden disruption to the flow of blood to parts of the brain. If an artery is blocked, the brain cells (neurons) cannot make enough energy and will eventually stop working.

Stroke affects patients in a number of different ways depending on the severity of the stroke and the type of stroke in which the patient suffers from. There two main types of disabilities that are a result of stroke: hemiplegia and hemiparesis.

The project aims to develop a virtual reality application to assist in the rehabilitation of the upper extremities of stoke patients who suffer from hemiparesis. The project will endeavour to design a low cost, home based system that will motivate patients by creating intermediate goals that can be adopted into the rehabilitation process. The project will utilise the "anti-gravity" arm support system to lessen the affect of reduced muscle strength and control. The project objectives are to:

- Research relevant background information on the effects of stroke.
- Research traditional methods of stroke rehabilitation and assessment of rehabilitation progression.
- Implement hardware and program for position data acquisition.
- Develop virtual reality application for exercise and rehabilitation assessment.
-

The project is based around detecting the movement or the patients arm and creating a computer representation. This has been performed by monitoring the potentiometers on the “anti-gravity” arm support utilizing the PIC AXE microcontroller. The microcontroller to converts the signal into a digital integer and transfers them to the computer via a serial link.

The games were designed around conventional physiotherapy exercises allowing the user to complete the exercises in a self motivating environment. The games were developed in both 2D and 3D environments utilizing Microsoft’s XNA games studio.

The project has been successful in accurately representing a user’s movement within a virtual environment. This has been tested by use of advance 3D mapping techniques; however the project is still not a stage where it is practical to perform clinical trials.

Certification

I certify that the ideas, designs and experimental work, results, analyses and conclusions set out in this dissertation are entirely my own effort, except where otherwise indicated and acknowledged.

I further certify that the work is original and has not been previously submitted for assessment in any other course or institution, except where specifically stated.

Ryan Fitzhenry

0050025782

____

Signature

____

Date

Acknowledgements

I would like to thank the following people:

My supervisor Selvan Pather for all the time, effort and enthusiasm which he has generously donated to the project.

My family for all their love and support they have provided over the last year.

My girlfriend Ally for her all support and putting up with me talking continuously about the project.

Contents

Table of Figures

Chapter 1
Introduction

1.1 Introduction

Approximately 16,000 Australians each year are left with a disability as a direct consequence of stroke. The number of strokes that occur in Australia each year is increasing, placing a strong reliance on home and community based rehabilitation. Virtual reality games are having an increasing role in the rehabilitation process. They are a great source of exercise and provide intermediate goals for the user to achieve and in turn increases the motivation of patients by rewarding them with a sense of achievement. With the dramatic increasing number of strokes each year, there is certainly a need for inexpensive effective rehabilitation programs that can be used in the home. This project will endeavour to design a low cost, home based exercise system that will motivate patients and can be adopted into the rehabilitation process.

1.2 Necessity for the Project

Rehabilitation usually starts with the first 24 hours of the stroke, as the process is most effective if immediate rehabilitation is provided. After acute care is given at hospitals, patients are either discharged to a rehabilitation unit for further therapy or sent home and receive outpatient therapy that is typically 2 to 3 times per week.

Stroke is Australia's second single greatest killer after coronary heart disease and the number of strokes that occur in Australia increases each year. Stroke is exterminated to cost Australia $2.14 billion each year which is certainly putting an increased strain of the Heath Care System. Consequently this puts pressure on doctors to discharge patients earlier into home and community based rehabilitation programs. (Australian Institute of Health and Welfare 2009). As new and better stroke treatments have been developed and implemented and the number of surviving stroke patients living in the community has increased. According to Elliot J. Roth MD from the Rehabilitation Institute Research Corporation, there is not only is there a need for further research into promising new inventions that promote health and function, but also a growing need for inventions that can be delivered in home and community settings.

1.3 Aims of the Project

The project aims to develop a virtual reality application to assist in the rehabilitation of the upper-extremities of stoke patients who suffer from hemiparesis, or a muscular weakness of one half of the body (refer to 0). The application will enable patients to conduct exercises that will complement their rehabilitation program.

The project will endeavour to design a low cost, home based system that can be adopted into their rehabilitation program and that will motivate patients by creating intermediate goals. The project will utilise the "anti-gravity" arm support system to reduce the affect of decreased muscle strength and control. The design will abide by the ideals and goals of the development of the device but will not be confined by its design choices or implementation.

1.4 Project Objectives and Goals

The project objective is to design and develop a virtual reality application to assist in the s rehabilitation of motor function in the upper and lower arm, including limited shoulder movement. In order to achieve this there are a number of vital preliminary research and design objectives that must be meet:

1. Research relevant background information on the cause and effects of stroke.
2. Research traditional methods of stroke rehabilitation and assessment of rehabilitation progression.
3. Identify exercises and assessment methods compatible with the "anti-gravity" arm support.
4. Implement hardware and program for position data acquisition.
5. Evaluate development packages and programming languages suitable for interfacing and games development.
6. Develop virtual reality application for exercise and rehabilitation assessment.

1.5 Methodology

The methodology behind the development of the project is structured so that the objectives are broken down into set tasks and allocated a time frame in which they must be completed. The first task is to define the project scope and establish clear boundaries of what the project will and will not address. Following this a strategic plan is developed by deciding on a logical sequence of activities. The project was based predominantly on research and development so it is important that the plan can handle unexpected delays and is dynamic enough to allow redesign pending the results of testing.

The following tasks were assigned to achieve the objectives:

- Research the cause and effect of stroke.

The first step is having a clear understanding of what stroke does to the body and the way in which it effects the person’s movement. By understanding this it is possible to predict the functional limitation that a stroke patient may have and what abilities and inanities can be expected from a typical virtual reality user.

- Investigate the rehabilitation process.

A key focus of the project is taking traditional rehabilitation techniques and further developing them through the use of technology. Investigating the current rehabilitation process is vital in understanding how the device has to adapt to changes as the patient’s recovery progresses. The task also includes the investigation of exercises used in rehabilitation therapy.

- Examine the current “anti gravity” arm system

The project is centred on the further development of the “anti gravity” arm support system. It is essential to have a clear understanding of the design and current specification of the device. By understanding how the system works and what it was designed for, it can be evaluated to determine if it is appropriate for the specific application or whether modifications are required.

- Develop conceptual virtual reality game designs

After the current system has been examined and the rehabilitation exercises have been researched it is necessary to determine which of these exercises from traditional rehabilitation techniques are compatible with the “anti-gravity” arm support. The next step is developing the conceptual game designs for the patient to complete the exercise in an entertaining and motivating setting. This section my also include the conceptual ideas of measuring and assessing the patients rehabilitation progress.

- Select programming language and games development environment.

This is one of the most important steps as it had the most amount of dependence on other tasks. It involves creating a specification of the system with a focus on interfacing the "anti-gravity" arm support and meeting the requirements for an appropriate games development environment.

- Program position acquisition module

This step evolves construction of a self contained code module which can be incorporated into the games development environment without additional handles or maintaining functions required within the main application. This code will act as an interface between the "anti-gravity" arm support and the selected games development environment. the code may also include the calibration and initialisation of the arm support. At this point, a programming aid test bed can be constructed, so that graphics programming can be done without repositioning the arm support.

- Design and development virtual reality game

The first stage in the process is evaluating the conceptual game designs and finding a balance between as to what best assists the patient and what can practical to be developed into a game and then implemented. This component is dependent on the conceptual design stage and will utilise programming aid test bed. The last step in this process will be interfacing the "anti-gravity" arm using the position acquisition module.

- Evaluation and analysis of the final system

This process will involve the testing and analysis of the system. The system will then be evaluated against the project specification. This stage may involve the use of a third-party for an unbiased assessment.

1.6 Conclusion

Stroke affects an increasing number of people not only in Australia but worldwide. Thousands of people that are diagnosed with stroke each year, very few survive and most are left with a varying degree of disability. Surviving stroke patients are sometimes discharged from acute care hospitals prematurely and do not receive all appropriate therapy suitable for recovery, as rehabilitation is expensive.

The project aims to further develop an automated virtual reality system to assist in the rehabilitation of the upper-extremities of stroke patients who suffer from hemiparesis. Investigation into the causes and effects of stroke is required to obtain an understanding of the various functional limitations of stroke patients. These investigations will also include a review of the current devices and practices used in current rehabilitation therapy.

This chapter has investigated the need for the venture and established a clear methodology for conducting the project. This includes researching the various aspects of stroke and rehabilitation therapy; and a structured approach for system design and development. It also outlines are procedure for test and evaluation of the system and software.

Chapter 2

Background of Stroke and Rehabilitation

2.1 Introduction

Stroke affects approximately 48,000 Australians each year. A third of these people will die within the first three months, another third will make a complete recovery and one third will be left with a disability that causes some reliance on others for assistance with day to day activities (National Health and Medical Research Council 2005). This chapter reflects on the causes of stroke, the effects it has on patients and current rehabilitation therapies.

2.2 Cause of Stroke

There are two main types of stroke: ischemic and haemorrhagic. An ischemic stroke is the most common type of stroke (85%) and is caused when a blood clot is formed in an artery or blood vessel in the brain. The Blood clot causes a sudden change in the disruption of blood flow to parts of the brain. When blood cannot reach those areas of the brain, the oxygen supply is cut off causing infarct to occur. Infarct is the death of localised tissue due to a lack of oxygen. An ischemic stroke can occur in two ways: the first being an embolic stroke, which occurs when a blood clot is formed elsewhere in the body and travels through the body to the brain. This naturally restricts the flow of blood to the brain and results in almost immediate physical and neurological deficits. The other way is through a thrombotic stroke which occurs when blood vessels narrow as a result of blood fat, cholesterol or calcium which grows to completely block the blood vessel (Queensland Health 1999).

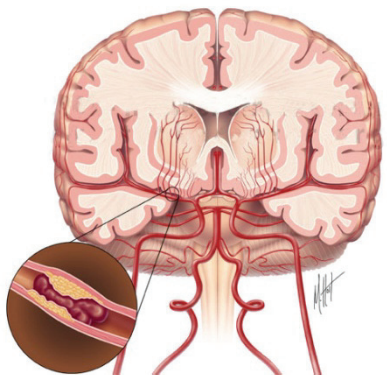


Figure 1 - Ischemic Stroke (Washington University School of Medicine 2008)

The second type of stroke is Haemorrhagic, which occurs less frequently. Haemorrhagic stroke is the result of a burst in the blood vessels in the brain causing blood to spread into parts of the brain. There are two main types of haemorrhaging that can occur. The most common type is an intra-cerebral haemorrhage which involves the bleeding of the brain tissue itself. The other type is a subarachnoid haemorrhage which is when bleeding occurs in the space around the brain (Washington University School of Medicine 2008).

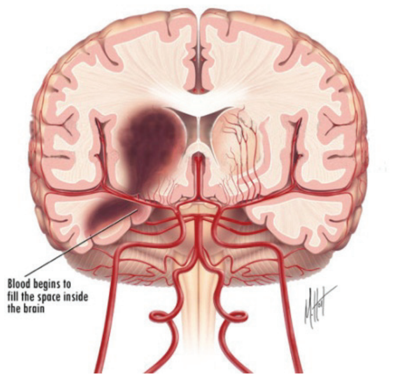


Figure 2 - Intra-cerebral Haemorrhage (Washington University School of Medicine 2008)

An aneurysm is an abnormal swelling or bulge in the wall of a blood vessel. They are caused by a weak or thin spot on a blood vessel wall that formes a balloon shape over time due the force of the pumping of blood. As the aneurysm gets larger the wall becomes thinner causing it to rupture. Aneurysms usually develop at a point where the blood vessel branch because the structure is more vulnerable (Queensland Health 1999).

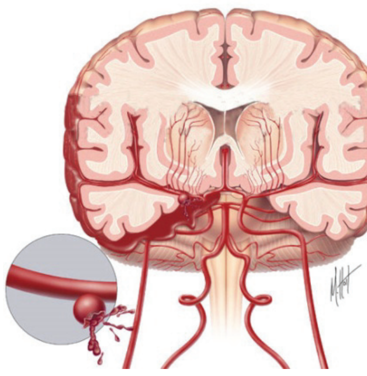


Figure 3 – Aneurysm (Washington University School of Medicine 2008)

2.3 Effects of Stroke

Stroke affects patients in a number of different ways depending on the severity and type of stroke in which the patient suffers from. There two main types of disabilities that are a result of stroke: hemiplegia and hemiparesis.

2.3.1 Effects associated with hemiplegia and hemiparesis

The most common type is hemiplegia which is the complete paralysis of one side of the body, which can include one side of the face. A related less severe disability is hemiparesis which is the weakening of muscles affecting one side of the body (Australian Goverment Intiative 2008). Both hemiplegia and hemiparesis are on a side of the body which is opposite to the side of the brain affected by the stroke.

People with hemiplegia or hemiparesis may experience difficulties with:

- speech, chewing and swallowing

- walking, balance, coordination and posture
- partial loss of vision
- loss of sensation

2.3.2 Physical and psychological effects of stroke

People that have had a stroke undergo physical and psychological changes. Physical changes include fatigue, neglect, movement and communication difficulties. Fatigue is a very common occurrence among patients who have had a stroke and this often can last for six months. Neglect is another common consequence of a stroke affecting the right side of the brain. This term is used when the patient does not take notice of the left side of the body or perform actions within the space left of their body. A loss of movement is another physical change and can be caused by the loss of muscle control and loss of sensory ability, which makes it difficult to move parts of the body. Communication difficulties mainly occur in patients that have damage to the left side of the brain which controls speech, reading and writing. Communication difficulties include aphasia, dysarthria and dysphasia. Aphasia is the most severe whereby the patients have unintelligible speech or no speech at all. The most common communication difficultly is dysarthria caused by weakness of the mouth whereby speech is slow or slurred. The dysphasia is also quite common which is a difficultly expressing or understanding speech. Suffers from dysphasia also have difficultly naming objects and expressing ideas. (Queensland Health 1999)

Aside from having visible physical affects, stroke suffers also can have a number of long term and short term psychological changes. The short psychological changes can lead to intense emotional changes such as (Queensland Health 1999):

- Grief
- Depression
- Frustration
- Fear

Short term psychological effects may be the result of a normal psychological reaction to a personal loss. Long term effects are the result of injury to the area of brain that control emotional function. Long term effects include:

- Memory Loss
- Logical Problem Solving
- Personality Change
- Loss of Motivation
- Depression

2.4 Stroke Rehabilitation

Stroke rehabilitation essentially evolves re-training of parts of the brain to take the role of the damaged areas due to the stroke. The rehabilitation process usually starts with the first 24 hours of the stroke as the process is most effective if acute rehabilitation is available. It is a goal-oriented process that aims to improve function and prevent deterioration of abilities to achieve the highest possible level of independence. Rehabilitation of stroke is targeted at all disabilities or impairments caused by stroke such as swallowing, mobility, speech and movement. (Washington University School of Medicine 2008) For the purpose of this report the main focus will be movement.

2.4.1 Physiotherapy

The rehabilitation of movement is performed by physiotherapists and is aimed at addressing the recovery of sensorimotor function in the upper and lower limbs. Sensorimotor function relates to the combination of effect muscle control and sensation feedback. They also assist in the treatment in musculoskeletal problems and respiratory problems. Rehabilitation of motor and sensory skills is preformed though a series of exercises. There are three main type of exercise used in the rehabilitation process (Medical Education Division 2007):

- Passive Range of Motion (PROM) Exercises

Passive range of motion exercises are perform with the assistance of a therapist where by the therapist supports and moves the body part through the full range of motion. PROM can assist with sensory difficulties; however it is more focused on maintaining good blood flow and keeping muscles and tensions flexible, preventing joints from tightening.

A typical passive range (Figure 4) exercises are performed from the sited position with patient's arm is positioned horizontal in line with the shoulder with the palm facing forward. The arm is then lifted straight up towards the ear. These exercises may be repeated 10 to 15 times in a typical therapy session.

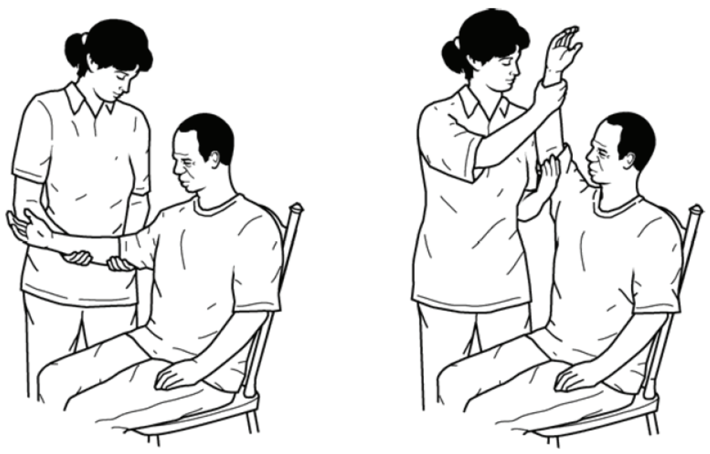


Figure 4 - Passive Range of Motion Exercises for the Arm (Ohio State University Medical Center 2007)

- Active Range of Motion (AROM) Exercises

Active range of motion exercises means the patient will lift or move the body part through the range of motion against gravity without help. AROM exercises can be performed with or without a therapist. They are good at improving motor and sensory skill however they are only able to be performed by a patient that has enough strength and control to support and move the body part.

There is a number of active range motion exercises that can be performed targeted at improving shoulder elbow and wrist rotation. A typical exercise aimed at the shoulder and elbow movement is where the patient is in the standing position with elbow bent. The patient then performs a rotation through the shoulder joint by lowering the hand towards the floor.

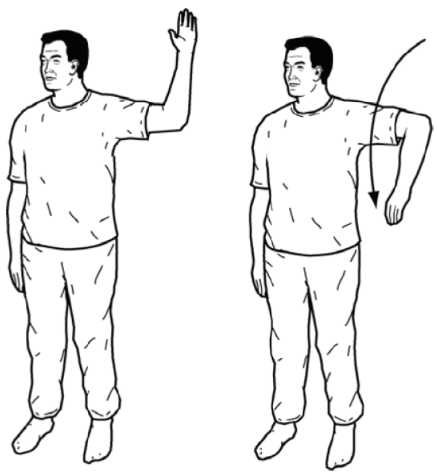


Figure 5 - Active Range of Motion Exercises for the Arm (Ohio State University Medical Center 2007)

- Active Assisted Range of Motion (AAROM) Exercises

Active assisted ranges of motion exercises are essentially anti-gravity exercises whereby the body part is supported and the patient is able to move freely. AAROM exercises are used when the patient has some sensorimotor function, however does not have enough strength to support the affected body part. This type exercises are able to assist with muscle control, sensory skills while also acting as a good diagnostic tool used by the therapist for determined the patient’s intermediate progress. As the patients movement is now visible because the affected body part is not prohibited by its weight.

A common active assisted range of motion exercises is whereby the patient is laid flat on their back while the therapist holds the arm vertical. This ensures that the weight of the arm is distributed between the shoulder joint and the therapist allowing the patient to move freely.

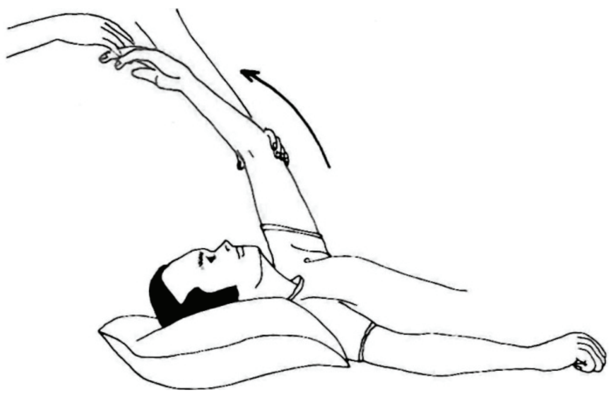


Figure 6 - Active assisted ranges of motion exercises for the Arm (Johnstone 1987)

2.5 Virtual Reality Game's Role in Rehabilitation

Virtual reality games are slowly being introduced into the rehabilitation process. The reluctance for the rehabilitation units to adopt the new technology is largely due to the expense of the current systems specifically designed for stroke rehabilitation. This is most evident in the amount of stroke rehabilitations units that have included the Nintendo Wii in the exercises for stroke patients (Cambourne 2008). The Nintendo Wii game console is relatively inexpensive and enables an active range of motion in a virtual reality environment. This game console requires the user to move a controller through a range of movement in order to achieve tasks provided on a television screen. Virtual reality games enable more interesting rehabilitation tasks to be created. Using a game environment also holds many advantages over more traditional methods (J.W.Burke et al. 2008):

- Varying level of difficulty

Having different level of difficulty is important to both game play and the rehabilitation process, as it keeps the user interested while allowing them to always be challenged at their level of progression. A varying level of difficulty can be implemented in a number of ways such as change the amount to precision, speed or logical problem solving required in the user.

- Increased motivation

Rehabilitation is a goal-oriented process, virtual reality games can provide more intermediate goals for the user to achieve and it therefore increases motivation by rewarding the user with a sense of achievement. This can be achieved as the game can provide instant feedback and hence the user can measure their success in real time. Virtual reality environments encourage greater levels of motivation, immersion and involvement leading to a greater sense of presence in the game which has a consistent positive correlation with the task preformed.

- Intensive repetition

Within traditional rehabilitation methods repetition of exercise and therapy is vital to the patient's success. Virtual reality offers enables individuals to repeat exercises in varying, interesting and relative realistic two and three dimensional environments at an almost subconscious level. Intensive repetition exercises performed in a virtual environment can provided a smooth transition from simple to more complex tasks in the real world.

2.6 Conclusion

Stroke affects patients in a number of different ways depending on the severity of the stroke and the type of stroke in which the patient suffers from. There are two main types of stroke: ischemic and haemorrhaging. The most common type of stroke is ischemic caused when a blood clot is formed in an artery or blood vessel in the brain. The less frequently occurring type of stroke is haemorrhaging which is the result of a burst in the blood vessels in the brain.

There are two main outcomes of stroke, hemiplegia and hemiparesis. Hemiplegia which is the complete paralysis of one side and in most cases is completely revisable. Hemiparesis is the main focus of this chapter which is the weakening of affected side of the body.

Stroke rehabilitation essentially evolves re-training of parts of the brain to take the role of the damaged areas due to the stroke. The rehabilitation of movement is mainly based on exercises assisted by a physiotherapists and is aimed at addressing the recovery of sensorimotor function in the upper and lower limbs.

Virtual reality games are slowly being introduced into the rehabilitation process and have a great potential to improve its effectiveness. The reluctance for the rehabilitation units to adopt the new technology is largely due to the expense of the current systems specifically designed for stroke rehabilitation.

This chapter has provided a background into the causes and effect of stroke. It has outlined the role of rehabilitation and basic concepts behind the exercises used by physiotherapists.

Chapter 3
The "Anti-Gravity" Arm

3.1 Introduction

The 'Anti Gravity' arm was a prototype developed by Scott Carden in 2008. The device was designed to facilitate a stroke patient's arm through a range of movements. The anti gravity arm neutralise the weight of the device itself and the changing weights of the different patient's arm. The design was quite successful in allowing a patient to move their arm is a range of movement with comparatively little force. This chapter will briefly discuss the structure of the anti gravity arm and method used for calculating it position. It will also introduce other devices which have been developed for use in the rehabilitation of upper extremities.

3.2 Structure

The structure of the 'Anti Gravity' arm is based on a modular design which allows certain aspects of the antigravity set up to be changed. This includes both force generation and the movement of the arm itself. Each module of the design is aimed at meeting different movements of the human arm. There are four main sections antigravity arm (Figure 7); the clavicle, shoulder and upper and lower elbow. These sessions are joined by pivots to create the six different degrees of freedom which the antigravity arm is able to achieve.

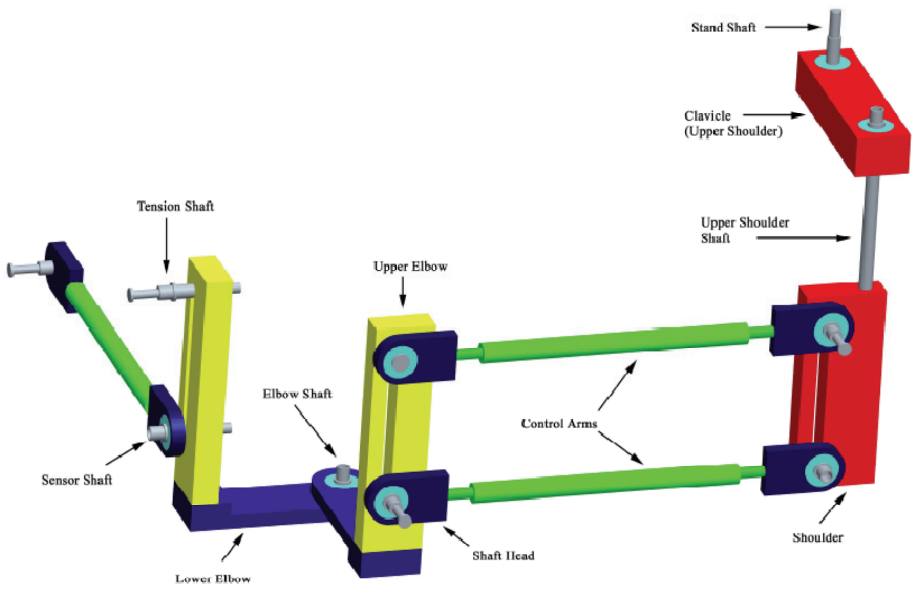


Figure 7 - Anti Gravity Arm Prototype (Carden 2008)

The anti gravity prototype is quite a heavy apparatus with an aluminium and steel construction. The design utilizes ball bearings mounted at each point of rotation help to eliminate much of the friction. The prototype uses rubber bands in order to support the weight of the device and the added weight of the patient's arm. The structure of device is very flexible in its design, in that it enables different amount of tension to be applied by changing the bands or changing the position of the tension shaft to achieve the desired force. Adjusting tension shafts also changes the position of the control arms and hence the changing characteristics of movement and lift (Carden 2008).

3.3 Position Acquisition

The anti gravity prototype has used Burnes 10 turn precision potentiometers to measure each to the five degrees of motion. This resistance is measured by converting the analogue voltage to a digital signal though the use of five ADC's on the MPC-MT-2113 ARM microcontroller (Figure 8). After the rotation has been measured, the three dimensional coordinates of the endpoint of the device (tip of the hand) are calculated using the Denavit-Hartenberg forward kinematics equation which is processed on the microcontroller as a series of matrix equations. The coordinates of the endpoint is then periodical sent to the computer via a serial RS232 connection (Carden 2008).

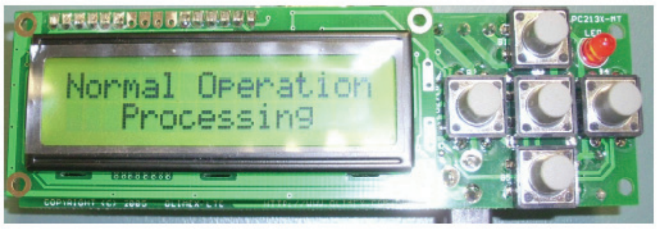


Figure 8 - MPC-MT-2113 ARM microcontroller (Carden 2008)

The endpoint coordinates are processed by the computer and display this point on the screen in which the user is able to move. The main disadvantage of the interface is this continually poling the microprocessor in order to receive the position data which is creating bottleneck in transferring data and causing delays in the applications. While this software does enable the user to interact with a virtual environment, the true position of the arm including the shoulder and forearm is not monitored.

Due to the nature and structure of the anti gravity arm, the centre position is arbitrary, as different users and even seating positions can affect how the device will be positioned relative to the users arm. The original design has overcome this issue by enabling the user to centre the device through a menu provided by the microcontroller.

3.4 Similar devices

The basic concept behind the anti gravity arm is that the arm will be supported through a range of movements without the user have to support the arms weight. There a number of apparatus designed at achieving this, including the IntelliArm7 and T-Wrex systems which are discussed below.

3.4.1 IntelliArm7

The IntelliArm 7 is a similar device to the anti gravity arm, in that they are based on a partial exoskeleton. The difference is that the IntelliArm is design to generate anatomic corrective movement and simulate passive range of motion exercises 'PROM' (see chapter one), While the anti gravity arm is design to assist with AAROM exercises and does not provide corrective movement. This is where the device guides the patients arm to a waypoint while prompting the user to a predetermined position.

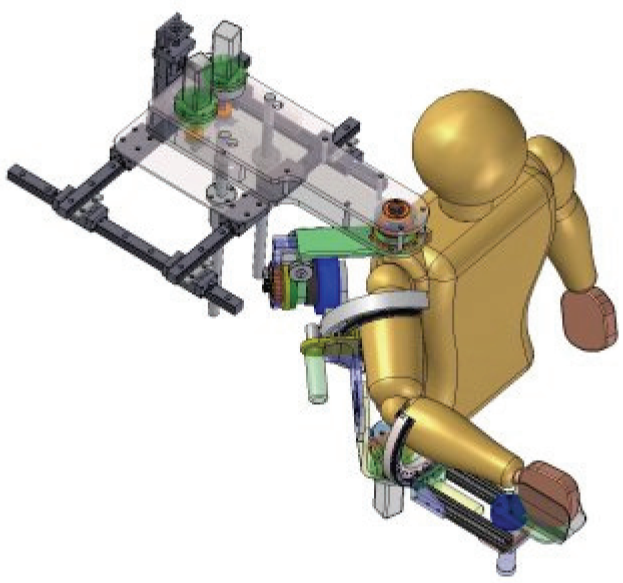


Figure 9 - IntelliArm 7 (Hyung-Soon Park 2008)

The IntelliArm 7 has three active degrees of freedom (DOF) and two passive DOFs. There are two active DOFs positioned at the elbow and one at the wrist. The two passive joints are in the shoulder joint to simulate the movement of the clavicle. The active support system for the IntelliArm 7 is achieved through the use of a combination of DC motors, cable and electronic clutches.

One of the design key features of the device is its ability to predict the user's movement for adding dynamic force based on the user's position and muscle strength. The software used with the device is very advanced as it monitors all joints and displays direct feedback to the user. Through the use of active assistance the user can interact with objects within a virtual environment which increases user's awareness and involvement in applications such as games and exercise routines.

The IntelliArm 7 is a very comprehensive therapy tool designed for use in a hospital environment in part of a rehabilitation program. One of its major disadvantages is that the device is very expense which prevents its use as part of home based therapy (Hyung-Soon Park 2008).

3.4.2 T-WREX

The T-WRWX is based upon the redesign of the Wilmington Robotic Exoskeleton (WREX) originally designed by Dr. Tariq in 2005. The original design was focused creating as an assistive device for children with neuromuscular weakness. The modified T-WREX system is adept at assisting with the functional limitations of stroke patients. It is a passive device with five degrees of freedom and uses rubber bands in order to balance the users arm (Housman et al. 2005).

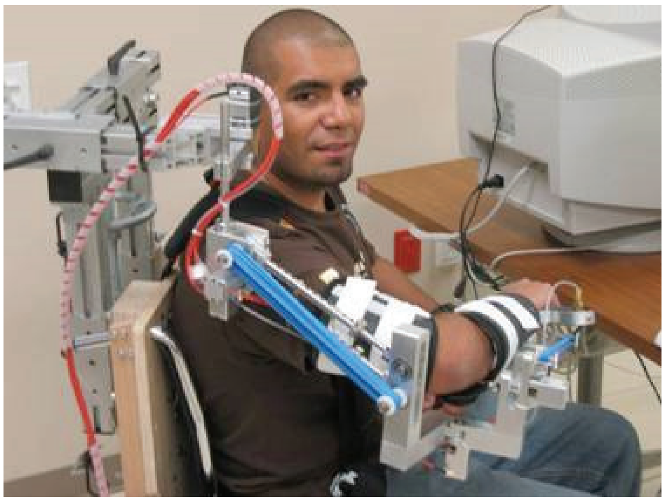


Figure 10 - T-WREX (Housman et al. 2005)

The T-WREX is used to track the patient's movements and display the arm position in two dimensions on screen. In addition to tracking the arms movement, the device also has a grip sensor for measuring hand strength which is also incorporated in the games and analysis software developed for the device. The system is developed using low cost components for inexpensive home use without the requirement of a supervising therapist.

The T-WREX system has been rigorously tested on a number stroke patient and has proven that the system can reduce motor impairment of stroke survivors who suffer from hemiparesis. The system has similar pitfalls as the anti gravity arm as they both require a physiotherapist to perform the initial setup of the device and do not provide the facility of remotely sending data about the user's progress and activity levels.

3.5 Conclusion

The antigravity arm is a passive device designed to facilitate a stroke patient's arm through a range of movements and is intended to assist with AROM exercises. The anti gravity arm neutralizes the weight of the device itself and the changing weights of the different patient's arms. It has a flexible modular design which enables certain components of the device to be adjusted in order to change balancing forces and movement behavior. The force required to balance the arm is achieved through rubber bands which also act to dampen the user's movement

The rotation of each joint is measured using potentiometers and then is converted into a digital signal using MPC-MT-2113 ARM microcontroller. The endpoint of the device is then calculated and transferred to the computer via a serial RS232 connection.

This chapter also reviewed two other devices design to assist in the rehabilitation of the arm; the IntelliArm 7 for PROM exercise and the T-WRWX for AROM exercises.. IntelliArm 7 is perhaps the more sophisticated of the two as devices, as it incorporates an automatic system for achieving active assistance and resistance. However the T-WRWX, like the antigravity arm is designed for inexpensive home based use.

Chapter 4

System Development

4.1 Introduction

The system development is an important process which starts revaluating the aims of the project and developing a general concept. After the concept has been formulated, it is subdivide into related designed sections which can be addressed individually. The two main designed section for this project as hardware and software.

The hardware design section is focused of determined the position of the users arm and transferring to the computer in a useful format so that the computer monitor the movement of the arm in real time. The software design section is comprised of three subsections; the microcontroller software, computer interface software and the games/application that utilize the data.

4.2 Concept

The basic function the interface module is to continuously read position data from the sensors connected the 'anti-gravity' arm's microprocessor. The micro-controller will then convert the rotation into an integer for each joint and transfer it to the computer. The interface will also enable the user to adjust setting and initialise the 'anti-gravity' arm's centre position. Each time the application starts it will enable the user to save the load user profiles. As shown in (Figure 11) the concept has been subdivided into the follow section which can be address individually:

- Hardware
  - Sensors
  - Micro-controller
- Software
  - Micro-controller software
  - Interface module
  - Game/Applications

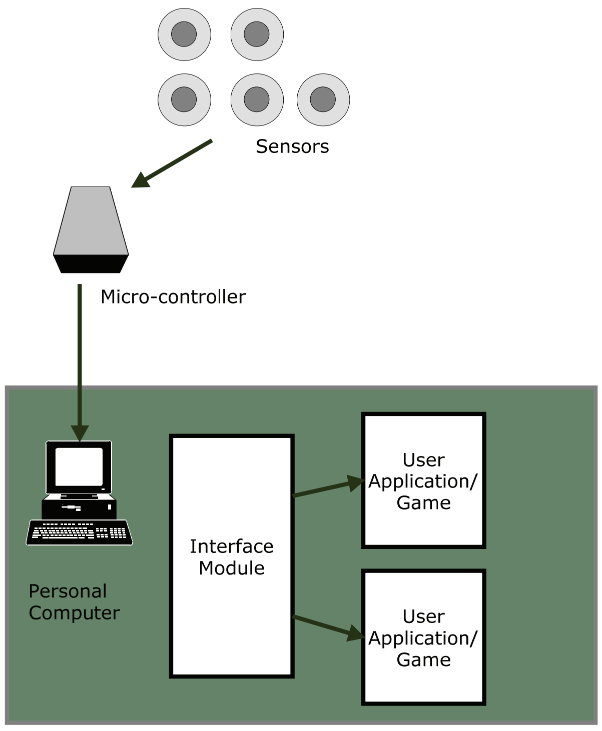


Figure 11 - Interface Concept

4.3 Hardware

In order to successfully measure the arm's position and display it in a meaningful format, there are a number of hardware issues that need to be resolved. From conceptual design developed, the main hardware components include sensors for measuring the arms joint rotation, processing these measurements and communications with the computer.

4.3.1 Sensors

The applications developed by the project will be used to directly evaluate the user's ability to perform certain exercises. For this reason it is important to accurately measure the user's position. The most common method used for measuring and recording a person's position is to use a motion capture studio.

Motion capture studios are mainly used by the entertainment industry for creating CGI animations for movies and games. They work essentially by recording video footage of the subject from multiple angles. The subject has uniquely coloured patches placed in key positions over the body part, which they wish to animate. After the video footage has been recorded it is then passed through image recognition software to identify the colour patches which are then mapped to a rigid model. Using the shape, intensity and position, vectors can be formed and translated into animation. Motion capture studios are very popular in measuring a human's movement as it provides the greatest degree of freedom, although is no simple way of measuring joint rotation. However for this application the antigravity arm acts as an exoskeleton whereby the rotation of the user's joints can be directly measured from the rotation of the joints of the antigravity arm.

There are a few different ways in which the joints of the antigravity arm can be measured. These are:

- The Hall Effect rotational sensors which work by using a transducer that varies its output voltage dependant the position of a rotating magnetic field. Hall Effect rotation (Figure 12) sensors are light weight, compact devices that offer a high degree of precision with a tight linearity tolerance typically around ±2%.

Figure 12 - Honeywell Hall Effect Rotational Sensor (Honeywell)

- Optical encoders are another possible sensor and work though the use of a photo interrupter with a thin metal plate that break the light beam. The metal plate has a large number of holes around the rim. As these holes passed through the photo interrupter a pulse is generated which can be read by a microprocessor to determine the angle. Optical encoders (Figure 13) offer a high degree of precision; however they have to be calibrated each time the device is used to register centre position.

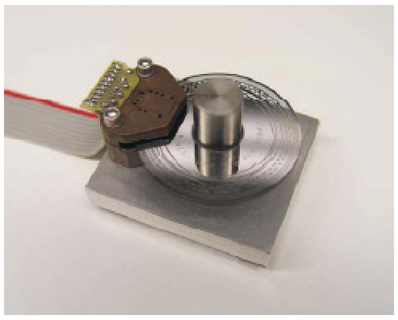


Figure 13 - Optical Encoder (Neamen 2007)

- Potentiometers are commonly called variable resistors. The most common type used for calculating rotation in robotics is the wire wound potentiometer (Figure 14) due to their degree of accuracy and tight linear tolerance. They consist of a conductive wiper and thin gauged wire wrapped around a spindle, as the wiper moves around the spindle the resistance is varied. By applying the rail voltage across the spindle, the wipers voltage will vary across the range indicating the rotation.

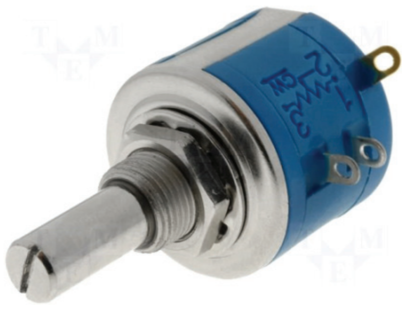


Figure 14 –Wire Wound Potentiometer

4.3.2 Data Acquisition

The output of the selected sensor will then have to be processed so that it can be transmitted to the computer. The simplest means of doing this is to use a microprocessor to convert the signal into an integer that can be sent to the computer.

The rotation signal is fed into the micro-controller as an analogue voltage from the position sensor. This voltage must then be converted to a digital number so that it can be read by the computer and position of the arm can be calculated. The device used to perform this is called an analogue-to-digital converter (ADC). The device converts the continuous signal to a set discrete digital numbers through the use of a bank of logic circuitry which compares the signal to varying reference voltage. The performance is classified by two characteristics: the resolution and the sample rate. As the output from the sensor in this application is a slow varying DC voltage, the affect of the sample rate is minimal. The accuracy of the interface is completely dependent on the accuracy of the sensor and the resolution of the analogue-to-digital converter. The most common resolutions used by ADCs are 8 bit, 10 bit and 12 bit. The accuracy of the ADC measured in volts per step is described by the following equation.

$$Q = \frac{E_{FSR}}{2^M} = \frac{E_{FSR}}{N}$$

Where

$$\begin{aligned} Q &= resolution\ in\ volts\ per\ step \\ E_{FSR} &= Full\ scale\ voltage\ range \\ M &= bit\ resolution \\ N &= the\ number\ of\ intervels \end{aligned}$$

Given that all the considered miro-controllers require 5 volt supply, $E_{FSR} = 5\text{v}$

Table 1 –Analogue-to-digital conversion accuracy

ADC resolution(M)	Number of Increments(N)	Volts per step(Q)
8 bit	256	19.5 mV/Step
10 bit	1024	4.8 mV/Step
12 bit	4096	1.2 mV/Step

The accuracy of converting an analogue voltage to a digital signal is directly related to the number of increments used as shown in table 1. However increasing the resolution also has the effect of increasing the amount of data that has to be transmitted to the computer.

The accuracy of rotational sensors are described by the number of turns performed to achieve full scale deflection. The expected accuracy of the complete system can be determined by calculating the number of steps that the ADC can read per degree. (Neamen 2007)

$$A = \frac{360.T}{2^M} = \frac{360.T}{N}$$

Where

$$A = \text{Accuracy in degrees per step}$$
$$T = Sensor\ turns\ per\ full\ scale\ deflection$$
$$M = bit\ resolution$$
$$N = the\ number\ of\ intervels$$

Table 2 - System Accuracy

ADC resolution (M)	**Sensor turns per full-scale deflection (T)**	**Accuracy in degrees per step (A)**
8 bit	1	1.4^{o}deg/step
8 bit	2	2.81^{o}deg/step
8 bit	5	7.03^{o}deg/step
8 bit	10	14.06^{o}deg/step
10 bit	1	0.35^{o}deg/step
10 bit	2	0.70^{o}deg/step
10 bit	5	1.75^{o}deg/step
10 bit	10	3.51^{o}deg/step
12 bit	1	0.08^{o}deg/step
12 bit	2	0.18^{o}deg/step
12 bit	5	0.74^{o}deg/step
12 bit	10	0.88^{o}deg/step

From these results it is found that using a 12 bit analogue to digital converter with a single turn rotation sensor it is possible to of measuring the rotational joints with the highest degree of accuracy. However there are physical limitations that will need to be considered in order to select the right devices, one of these limitations is the data rate achievable between the computer and a microprocessor.

4.3.3 Device Communication

There are three main types of communication protocols used for interfacing a device with the computer:

- USB - *Universal Serial Bus*

USB (*Universal Serial Bus*) in recent years has become the most popular medium for setting up communication with peripheral devices. There are two versions of USB, USB 1.1 which can support speed of up 12Mbit/s and USB 2.0 which can support 480Mbit/s. Due to the success of USB a large number of micro-processor manufactures have included USB interface modules and code libraries into their products.

- RS232 –Standard Serial Protocol

RS232 or serial connection is perhaps one of the old communication protocols still in use. It was intended for speeds of up to 20,000 bits per second. The protocol also previsions for parity bit and start/stop bit checking.

- Ethernet – Local Area Network Standard

Ethernet is the most popular protocol for transfer data long distance. It can support 2 Gigabit/s speeds, however most micro-controllers cannot support this speed. The main reason that Ethernet was considered as an option is its simple integration with wireless technologies.

4.3.4 Interface Hardware Selection

In considering the different hardware options, it is easy to identify the option which achieves the highest accuracy and response. However there are number of non-technical aspects which have to be considered to ensure the best option is selected. These aspects include:

- Minimum requirements and physical limitations
- Quality vs. Expense

The minimum requirements of the interface can be determined by considering in the physical limitations of both the patients and the limitations of the antigravity arm device itself. Due to the configuration of the antigravity arm, the maximum rotation that any joint can achieve is 180°, therefore using sensors that offer more than a single turn has no advantage. The applications for the interface are predominantly used for displaying the user's movement. Although computer monitors are capable of displaying very high fresh rates, the human eye cannot distinguish between the refresh rates high than 25 frames per second. Therefore there is no benefit in enabling the interface to obtain and transfer more than 25 positions per second.

There were three rotational sensors considered for the project. The Hall Effect rotational sensors and optical encoders were suggested to have the highest degree of precision. The original design has used potentiometers is on each joint which is proven to be an inexpensive option that still maintains a high degree of precision.

From the system accuracy table (Table 2), it was found that a single turn sensor with a 12bit ADC will provide the most accurate solution. However micro-controllers that offer 12bit ADCs are typically less common and are significantly more expensive. Ideally for this application the accuracy should be able to achieve at least 2° per step. The resolution of the analogue to digital converter does not only affect the accuracy of measurement. It also affects the data rate back to the computer, as the higher resolution of the ADC increases the amount of data that needs to be sent back to the computer.

When considering these factors a single turn sensor with an 8 bit ADC would be an optimum solution which will offer 1.4 degrees per step with the lowest amount of data required to be transferred to the computer.

The original prototype design used Bournes 3590S, 10 turn precision potentiometers. These potentiometers have a tolerance of 5% and linearity of 1.25%. In the interest of constructing a working prototype as a proof of concept these potentiometers will not be replaced. However this will increase the requirements of the analogue to digital converter as a 10 turn potentiometer using an 8 bit ADC will only achieve 14° per step which is not accurate enough to measure the user's movement. For this reason the minimum requirement for the prototype will be a 10 bit ADC.

The communication interface can now be determined by calculating the minimum data rate. This can be calculated by knowing the analogue to digital converter’s resolution in the minimum refresh rate that has been selected above. Firstly the amount of data has to be calculated, using a 10 bit ADC means that each joint rotation will be represented by 10 bits. The antigravity arm has five monitored joints; therefore the interface has to send 50 bits each sample. The minimum samples per second was found to be 25, therefore the interface needs to send 1250 bits per second. This data rate is within the capability of the serial port. The serial port interface is the most simple to implement and is also most commonly available on micro-controllers.

4.3.5 Micro-controller Selection

A micro-controller is a low-cost integrated circuit that contains memory, processing units and input/out circuitry in a single unit. The purpose of microcontroller is to use information collected from a variety of sensors to make decisions about how to control the output. Micro-controllers are used in numerous devices around the home and in industrial applications. The main advantages of using micro-controllers over dedicated hardware are:

- Fast development.
- Small circuitry layout.
- Increased reliability through a smaller part count.
- Ability to upgrade or provide fixes after the product has been shipped.
- Low power consumption

The selection criteria for the micro-controller, is that it must be a low cost, low development time and reliable device that meets the minimum requirement established in section above. The minimum requirements of the micro-controller are:

- Five 10 bit analogue-to-digital converters
- Capable of achieving 25 samples per second
- Capable of sending 1250 bits per second via a communication link to the computer.

There were three micro-controller considered for the project, however given the relatively low requirement for this application many more could have considered. The micro-controllers consider included:

- ARM 7 LCP-MT-2138
- Motorola MC68HC12
- PICAXE microcontroller

4.3.5.1 ARM 7 LCP-MT-2138

The first micro-controller considered for data acquisition was the ARM 7. This is the microprocessor which was used in the original prototype with the LPC-MT-2138 development board. The ARM7 2138 series has a CPU frequency of 60MHz with 32kBytes of Random Access Memory (RAM) and 512kBytes of flash memory for storage. This microcontroller is very advanced, its main features are:

- Speed operating at 60Mhz
- LCD 16x2 display and buzzer
- RS232 serial port interface
- Inbuilt boot loader for rapid development
- Eight 10 bit analogue to digital channels
- Six pulse width modulated outputs
- Five built in buttons
- Controlled 10A 250VAC Relay

The main advantage of the micro-controller is its flash memory and inbuilt with boot loader which improves the development process. By using flash memory, programs can be downloaded, tested and erased quickly. The boot loader ensures board recovery in the event of software failure. Its main disadvantage is its expense and code complexity. The controller also requires a large amount of circuitry in order to run under minimal conditions.

4.3.5.2 Motorola MC68HC12

The second micro-controller considered was the Motorola MC68HC12. It is one of the most advanced single chip computers in the market place today. There are approximately twenty different processers in the HC12 family offering different input/output configuration. The HC12 was originally designed by Motorola’s Semiconductor Division in 2004 however this division has now separated from the parent company and is now known as Freescale Semiconductors. HC12 is capable of running in both 16MHz and 8MHz operation modes, its major features include:

- low power consumption
- power saving features including STOP and WAIT modes
    - large addressable memory 1024 Bytes of RAM

  - 4096 Bytes of EEPROM (Electrically Erasable Programmable Read-Only Memory)
  - On-Chip Memory Mapping Allows Expansion to over 5 Mbytes of Address Space
- 18 analogue to digital channels with 8bit, 10bit, 12bit resolution modes, note using different modes prohibit all channels from being useable
- Enhanced Synchronous Serial Peripheral Interface
- Pulse width modulation outputs
- Two Enhanced Asynchronous Non-Return to Zero (NRZ) Serial Communication Interfaces (SCI)

Motorola MC68HC12 is a very flexible micro-controller which can be further expanded by connecting interface modules and additional input/output devices to the chips 16 Multiplex address/data lines. One of the major disadvantages with this particular micro-controller is the amount of complexity in coding for simple applications. This is caused due to the vast amount of functionality which the board is able to achieve.

4.3.5.3 PICAXE microcontroller

A PICAXE micro-controller series was also considered. There are ten micro-controllers in the series each offer different out/input configurations. The PICAXE micro-controller is a standard PICmicro chip that has been pre-programmed with the PICAXE boot loader code similar to that of the ARM 7. After evaluating each chip in the series, it was find that the PICAXE 40X1 was the most appropriate, its major features are:

- Low power consumption
- 7 analogue to digital in both 8bit and 10 bit resolution modes
- Internal timer
- 20Mhz speed
- 28 bytes of memory
- 260 bytes of flash memory
- High speed serial interface (RS323)
- 2 pulse width modulated outputs

The main advantage of the PICAXE microcontroller is simple instruction set, which makes the controller less flexible. However it is well suited for this simple application of read analogue voltages and relaying them through the serial port. The micro-controller's flash memory and boot loader make the system well designed for rapid development. The PICAXE Company also provided an extensive development environment which also allows the developer to write code in C and compile the code directly in the PICAXE instruction set.

4.3.5.4 Selected micro-controller

After evaluating each of the considered micro-controllers, the PICAXE 40X1 (Figure 15) was found to be the optimum solution. The main reason for selection was the micro-controller's low cost and rapid development. PICAXE 40X1 microcontrollers are valued at $22 and the development board costs just $50. While this microcontroller was not as advanced as many of the others consider, it still maintains a number of features which will allow the process scope to extend as the project progresses. Another advantage of the PICAXE is the extensive free software and tutorials that is supplied with the chip, such as:

- PICAXE Program editor

  The PICAXE Program editor is the main application for program the PICAXE microcontroller. The software is windows compatible which supports textual programming in BASIC and graphical flowcharting methods for program development. The software also includes a basic simulator for the PICAXE microcontrollers.

- Logicator for PIC micros

  Logicator is a flowcharting application designed for developing program flowcharts which can then be converted to BASIC and downloaded to the PICAXE chips.

- PICAXE VSM.

  This software is a circuit simulator which can simulate the developed PICAXE program and how the interacting circuitries will behaviour to different inputs and outputs.

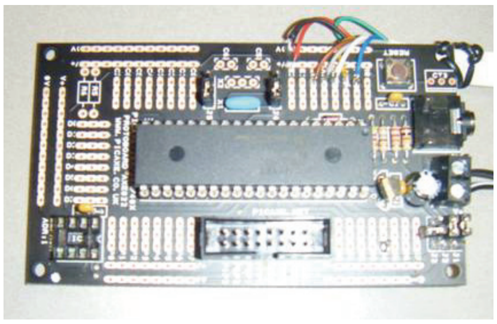


Figure 15 - PICAXE Development Board

4.4 Software

The software consists of three main parts, the interface module, GUI (graphical user interface) configuration menus and the exercise games. Selection of the development environment can be broken into four parts:

- Graphics/Game Engine
- Graphical User Interface Toolboxes
- Programming Language
- Compiler

Almost all the programming languages provide the ability to produce graphical user interface tools and serial interfacing libraries. However not all provide the ability to transform matrixes and vectors for the development of games. For this reason the selected games engine will be a major factor in the chosen development environment.

4.4.1 3D RAD

The first game engine considered was a free 3-D game maker called 3D RAD. The game engine was designed to run on both Windows Vista and Windows XP. It provided an intuitive set of menus developing game without providing any code. Its main features include:

- Ability to import 3-D models and 2D sprites
- Performing matrix and the vector calculations in real time
- Simple physics such as collision detection and bone structure
- High-quality rendering

- Artificial intelligence
- Network connectivity
- Sound effects and music
- Pixel Shading and partial movement

The major advantage with this game engine is its relatively easy implementation of games without coding. However this is also a major downside as it limits and restricts the developer to certain game styles. In order to interface the microprocessor with the game an interface module would have to be developed to send rotational data via series of key presses. This would have the effect of stalling the game each time data is received through the serial port though the use of the keyboard interrupt.

4.4.2 Source Game engine

The Source Game Engine is developed by the Valve Corporation in 2004. It has been used to develop games such as Counter Strike, Half-Life 2 and many others. In order to develop games for the Source Game Engine, the game modification called Garry's Mod is required. This game modification can be purchased at a price of $60. The source game engine utilises graphics in both OpenGL and DirectX. Its main features include:

- Advanced physics engine
- Ability to import models and sprites
- Utilisation of multi-core processors.
- Network connectivity
- Sound effects and music
- High-quality rendering.
- Support for high refresh rates.
- Pixel shading and special effects

The source game engine is designed for developing first person shooter games in C++. The Valve Corporation distributed a development kit called Source SDK. Due to the complexity of the game engine this development kit incorporates a large set of tools for modelling and texture. This game engine will not support 2-D games. While the game engine is very flexible, the development of simple games is quite a complicated and lengthy process.

4.4.3 Unreal Game Engine

The Unreal game engine was developed over several of years by Epic Games. The most recent version of the Unreal Game Engine is version 3. This version was developed for the release of Unreal Tournament 2004. The engine has been continually updated at its release and now supports 64-bit processing and DirectX 10. The game engine includes a vast number of features:

- Advanced NVIDIA physics
- Fast 3D texturing
- Multi platform development
- Advanced artificial intelligence
- Movie back play including DIVX video compression
- Network connectivity
- High-quality rendering
- Support for high refresh rates.
- Pixel shading and special effects

The development package for the Unreal game engine incorporates a large number of tools including speech development software, AI development and Autodesk's Kynapse for 3D modelling. The game engine itself is programmed in C++ and incorporates a large number mesh and vector transformation tools.

4.4.4 Away-3D Flash Game Engine

Adobe Flash was first developed by Macromedia as 2D animation tool. Throughout recent years of development it has provided more user interactivity and is now commonly used in its standalone format and as a tool for displaying animations on web pages. Away-3D is an open-source 3D engine for Adobe Flash. It is still under developed however it is available for download and offers a number of compressive features:

- Keyboard and mouse input
- Light weight application
- Incorporated web page support
- Sound effects and music
- Ability to import models and sprites

Away 3-D is a very small game engine only focused on Web distribution. It has a large user base and open source development team which provide excellent support for developers. The engine itself has a well-documented library and improvements are continually being made. It supports development under both C++ and Java.

4.4.5 Open-GL

OpenGL was first developed by Silicon Graphics Inc in 1992 and since then has become a widely adopted graphics application programming interface (API). By definition OpenGL is not a game engine but is more just a tool for displaying graphics on a computer. It comes with its own set of library functions for drawing both 2-D sprites and importing 3-D models. Its main features are:

- Cross platform support
- Ability to incorporate 2-D stripes and 3-D models
- Fast rendering
- Flexible low-level programming

As mentioned previously, OpenGL is simply a raw tool to displaying graphics. It does not provide any games functionality including manipulation of vectors and meshes. The OpenGL library is available in a number of languages including C++ and Java. Due to its low-level design is extremely flexible and enables the developer to incorporate virtually any application specific modules. However this means that all mathematical helper functions and manipulation functions have to be written in the development of the game. The user interaction with the game also has to be written including keyboard and mouse functions.

4.4.6 XNA Game Studio

XNA Game Studio is a game engine/complete development environment developed by Microsoft for the use of amateur game developers. Games designed in XNA can be run on both Windows systems and Xbox 360 consoles without modification. XNA requires visual studio C sharp as the integrated development environment, which incorporates windows APIs libraries and a C sharp compiler. It also includes software for integrating sound and music into the games call “Microsoft cross-platform sound creator”. The complete package is free to download from the XNA in a creator's website for non-commercial applications. Some of the key features include:

- Ability to import models and sprites
- Incorporated sound and music
- Multiprocessor support
- Easy to use vector and matrix calculations tools
- Asset library for managing game materials
- Collision detection for both 2-D and 3-D objects
- Fast 3D texturing
- Pixel shading and special effects
- Xbox 360 compatibility.

Animating objects in real time is a complicated task in most game engines. One of the major advantages of XNA game studio is the maths helper library. This library enables 2D sprites that have been mapped to vectors or 3D models that have been mapped to meshes to be manipulated in one line of code. Another key advantage of using XNA is the Cross-Platform Audio Creation Tool. Which enables WAV (waveform audio format) files to be allocated to cues that can be call throughout the game to play sounds.

4.5 Conclusion

There are many possible ways to create a system for accurately measuring the arms position and creating a game for use in stroke rehabilitation. This past chapter has discussed many technologies for measuring the arms position and incorporating this position into an interactive game. The topics cover both hardware devices and software environment selection.

The hardware design incorporated all components from accurately measuring the position to processing the data and transfer to the computer. From the different design options, the selected design involved measuring the rotation of each joint using 10 turn wire wound potentiometers, processing the analogue signal using the PICAXE X1 and transferring the data via a serial connection to the computer.

The software design concluded that XNA Game Studio would provide the best integrated development environment and that all coding will be performed in Microsoft Visual C sharp. This option was selected because of its advanced vector and mesh manipulation libraries and intuitive design. Programming under Microsoft Visual C sharp will also enable the creation of an interactive graphical user interfaces.

Chapter 5
Interface Software Development

5.1 Introduction

The first stage in the software development cycle is to design realistic concepts for the program to achieve miro-controller communication. After the concepts have been clearly defined the next stage is to form a specification. The specification will identify all operational and developmental details of the project. This phase will also choose the appropriate software development environment and programming language. The software design process will also include testing of individual modules and application goals.

5.2 Concept

The concept of the interface software is directly related to the conceptual design of the interface hardware developed in chapter 4. The concept is that the microcontroller will continuously stream four 10bit integers from the miro-controller indicating the rotation of each of the "Anti Gravity" arm's joints.

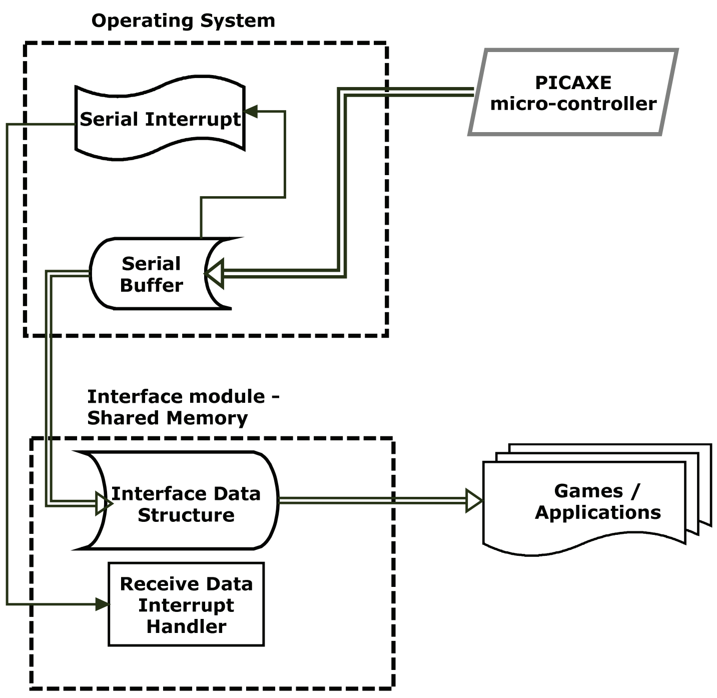


Figure 16 - Interface concept design

There will be two elements to the interfacing software, the micro-controller software and the computer interface. The micro-controller software will perform the analogue to digital conversion and send the data via RS232 standard serial protocol. This data will then fill the operating systems serial buffer triggering an interrupt.

The computer interface will register a function with the operating system’s interrupt handler. This function will be called every time the buffer is filled, the function will then copy the data out of the buffer and indicate to the operating system that the buffer can now by re-written to. Games and applications will now directly access data via the shared interface data structure.

The computer interface will also now incorporate a series of menus for configuring RS232 interface; loading and saving user profiles; and centring the device upon the initial setup.

5.3 Software specification

The main of the interface module is required to have the ability to maintain and receive data from the connection with microcontroller.

Connection to the micro-controller is maintained through the computer’s serial port and hence the only requirement of interface module is that the selected language must have the tools for interfacing with the RS232 port. Almost all modern day programming languages contain tools and modules for directly addressing the operating system’s API calls for serial communication. Therefore the chosen game development environment will have a major influence in the programming language selection. The interface module is also required to provide a graphical user interface for troubleshooting the serial connection to the micro-controller and allowing the user to centre the arms position each time the application has started. Each time the interface has loaded, it will provide the user with the option to load and save settings.

The chosen games development environment must have the ability for the interface to continue streaming while the game is running. It must also assist the user in tracking their performance throughout their use of the game. In order of to track the user's movement in real time, the development package has to include efficient vector manipulation functions for transforming 2-D sprites and 3-D model meshes.

5.4 Software design and implementation

In the design phase, the system specification is transformed from a list of functions and requirements to a detailed statement of implementation. This stage will specify how the system requirements are to be met by partitioning the functionality into class consisting of data structures and functions.

5.4.1 Micro-controller Interface

The microcontroller interface is a relatively simple program which reads the potentiometers voltages through the analogue to digital integers and sends them via this serial port. The PICAXE has seven word variables labelled from W0 to W6 and these variables can also be addressed in bytes form B0 to B13. The first task is to read in each of the ADC values to word variables.

'Anti-gravity' arm Joint	Word Variable	Byte Variable
Joint 1	W0	B0-B1
Joint 2	W1	B2-B3
Joint 3	W2	B4-B5
Joint 4	W3	B6-B7
Joint 5	W4	B8-B9

Table 3 - Variable Usage

The serial port works by sending a single fame at a time. Frame sizes can vary from 5 to 9 bytes. The interface is required to send 5 ten bit integers however one restriction of the PICAXE is that the bits have to be sent as a single bytes or words. Therefore interface will be required to send five words which are 10 bytes. This means that 2 frames will be required and additional bytes will have to be sent as to determine which frame is being used for the particular potentiometers. These additional bytes will hold characters which will also assist in error checking.

First Frame

"S"	B0	B1	B2	B3	"E"	"E"	"E"

Second Frame

"T"	B4	B5	B6	B7	B8	B9	"F"

For the serial communication to transmit data successfully the microcontroller has to wait in between each frame sent. A more efficient way of doing this is to allow the microcontroller to read in values from the analogue to digital converter in between each frame.

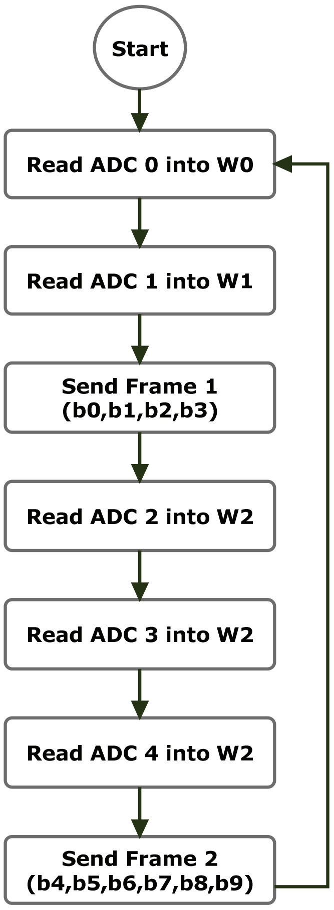


Table 4 - Microcontroller Flow Diagram

5.4.2 Computer Interface

The interface module is responsible for maintaining the connection to the antigravity arm and recording its position. This module is expected to run relatively slow compared to other applications that use the data, as the program will be continually reading the status of the I/O port. Modern computers utilise multi-core processors which enable applications to perform multiple processes and threads at the same time. Using this technology for the interface module will have significant performance advantages and improves CPU utilization. In order to achieve this, applications utilising the module can receive and manipulate data within the interface module will require the use of shared memory.

Shared memory across multiple processes causes a lot of problems. Such as when memory is accessed by while it is partially been written by another process. To overcome this problem, the interface module will be implemented as a shared library whereby Microsoft visual C# will provide its own protection to ensure this problem does not occur. Another feature that will be implemented to improve this module's performance is to directly utilise the system interrupts for serial communication. This will prevent the program being required to continually poll the serial port.

5.4.2.1 Data structures

Stage one in designing the interface module is to define all variables and functions that will be required. As this module is the core of the application and will be utilised by a number of different programs, it is important to ensure that data specific to the interface module cannot be directly addressed or modified. This is done in C# by defining variables as private. These variables can only be accessed by the interface module itself. Data required by the interface module can be classified into three sections:

1. Serial port information
    - baud rate
    - port name
    - received byte threshold
    - number of other bits
2. Device calibration information
    - device name
    - potentiometer turns
    - potentiometer values
3. Connection status information
    - number of failed readings per second
    - number of successful reads per second
    - status(string)

5.4.2.2 Functions

After defining the data structures for the interface class, the functions can be defined. There are six main functions of the interface module:

- Open Interface

One of the first function that has to be performed is the Connect function, however before this can be performed the interface has to be initialised by providing information about the connection to be made such as baud rate, port name etc. These variables are provided to the interface in two ways; by a series of public set calls or a load configuration calls.

- Connection synchronisation

Syntonisation is often required after the serial connection has been made. After experimentation with the serial connection it was found that using different operating systems and hardware devices such as USB to serial converters often caused fames to become corrupt. This is a well documented error particularly with the Windows Vista operating system. The simplest way to overcome this problem is to monitor the connection recording the number of successful reads per second. If the number of successful reads drops below a predetermined level than syntonisation is required. The process of performing syntonisation on the serial port is very rudimental; it is done by creating delays the connection by disabling the read interrupt for random periods of time until the connection is re-established. If the syntonisation fails on more than five attempts it is likely the interface is suffering from a critical error.

- Read Data Interrupt

The most important role of the interface module used is read data out of the serial buffer and interpret it so it can be accessed by other programs. The microcontroller transfer data has hexadecimal bytes, therefore in order to convert these into integers. The program must then identify which framers it has received. Then convert the hexadecimal bytes into hexadecimal characters which can then be converted into integers.

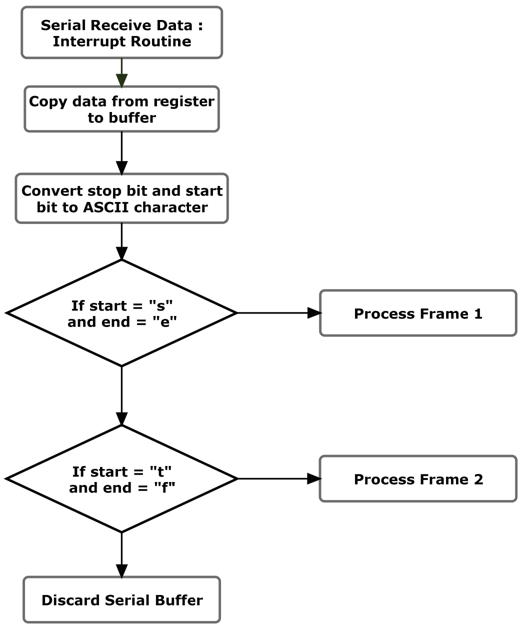

Serial Receive Data : Interrupt Routine
Copy data from register to buffer
Convert stop bit and start bit to ASCII character
If start = "s" and end = "e"
Process Frame 1
If start = "t" and end = "f"
Process Frame 2
Discard Serial Buffer

Figure 17 - Micro-controller software flow diagram

Each time a frame is read, the start and end bytes are checked to ensure the frame is vaild. If a framed does not containing the correct start and end byte then frame is corrupt due to a transmission error. In this case the data in the serial buffer is discarded and the count of unsuccessful reads is incremented.

- Load and Save Configuration

Each time an application starts it requires certain information about the interface and the device in order to make a successful connection to the control. This process is made simpler by incorporating load and save functions into the interface module. There are three files which are maintained by the interface module: config.txt, ArmCali.txt and TestbedCal.txt.

The file titled “config.txt” contains all information about the serial communication including:

- port name
- baud rate
- byte threshold
- current device

Files “ArmCali.txt” and “TestbedCal.txt” contain all information about the device that is connected on the interface. The current device field in the configuration file is used to make the decision which device file is loaded. Device files include the following information about the device:

- device name
- sensor rotations
- centre values for all sensors

- Set Centre Position

Each time the 'anti-gravity' arm is used the bands and height are adjusted, this causes the centre position to change. The centre position is the position when the arm is held horizontal and in line with the body. This position is used as a reference point to calculate the joints rotation.

- Connection status functions

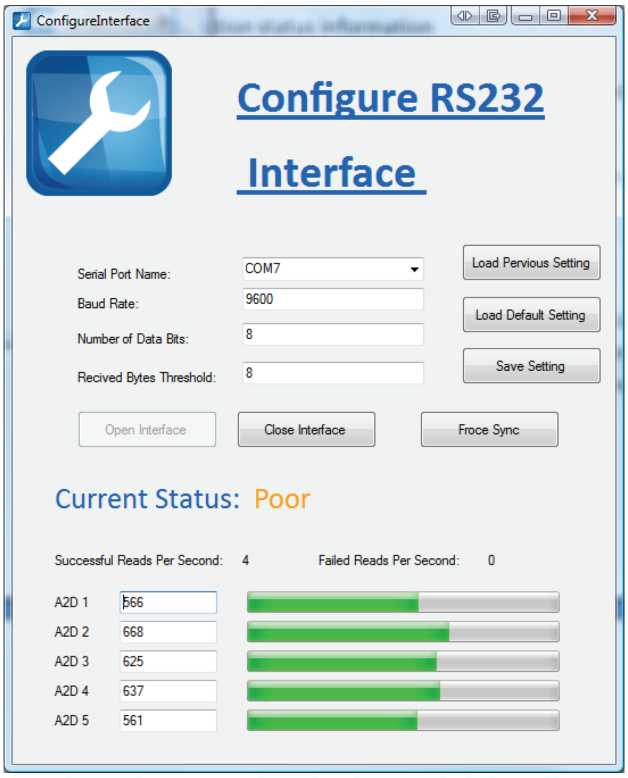
In addition to maintaining the connection, the interface module also monitors the status of the interface by recording the number of successful reads per second and failed reads per second. These variables are maintained by using a timer interrupt. The main reason is to ensure synchronisation however these values are all so used to display connection status to the user. After experimentation it was found that successful reads greater than 10 offers a more reliable interface. This information is displayed to the user by a generate string function which returns the following:

$$\text{Successful reads per second} > 10 \quad : \mathit{Healthy}$$

$$\text{Successful reads per second} =< 10 \quad : \mathit{Poor}$$

5.4.3 Graphical user interface

The graphical user interface (GUI) is now the most common way of allowing users to interact with programs. It allows the layer of abstraction which helps the user have a basic understanding of what the program is doing. The graphical user interface of this application will be a lot more comprehensive than the final product, as this version of the software will allow for testing and development of interface. There are two main GUIs that are designed to directly interpreted information from the interface module. The first one is RS232 configurations menu (figure 18); it allows the user to change various aspects of the connection. The other is device calibration menu which allow particulars of the device to be altered. All these GUIs interpret information from the interface module however they are not a part of the interface module; the interface module is stand-alone code which can be incorporated into any application.



Serial Port Name: COM7
Baud Rate: 9600
Number of Data Bits: 8
Recived Bytes Threshold: 8

Current Status: Poor

Successful Reads Per Second: 4   Failed Reads Per Second: 0

A2D 1 566
A2D 2 668
A2D 3 625
A2D 4 637
A2D 5 561

Figure 18 - Configuration menu

5.4.3.1 Configuration Menu

Computers often have more than one serial port that can be either mounted on the motherboard or can use I/O expansion boards and USB converters. The nature of the serial port is that the computer has no inherent information about what devices are connected to it. This causes problems with programs that assume a particular port name.

The first item on the configurations menu is a drop-down box which lists all serial ports active on the system. This is done through a system call to the operating system which returns an array of serial port names. The serial port name is mapped through the operating system the particular of hardware address and buffers. The other fields which the user is able to modify will rarely need to be changed however these were very useful in the testing and development stage. These fields include the baud rate, number of bytes per frame, and received byte threshold. There are also a number of buttons on the configuration menu:

- Load Pervious Setting

The function of this button is to load a connection calibration file. This function is also performed when the window is created. After this is loaded all fields update to the information contained within the file.

- Load Default Setting

This function is essentially the same as the load function however the program loads the information a default calibration file. The default calibration file is saved as read only file and cannot be changed from with the program.

- Save Setting

This button allows the user to save all settings related to the connection. The system only maintains a one configuration file, this is done so that games can load the file without requesting which file to load.

- Open / Close Interface

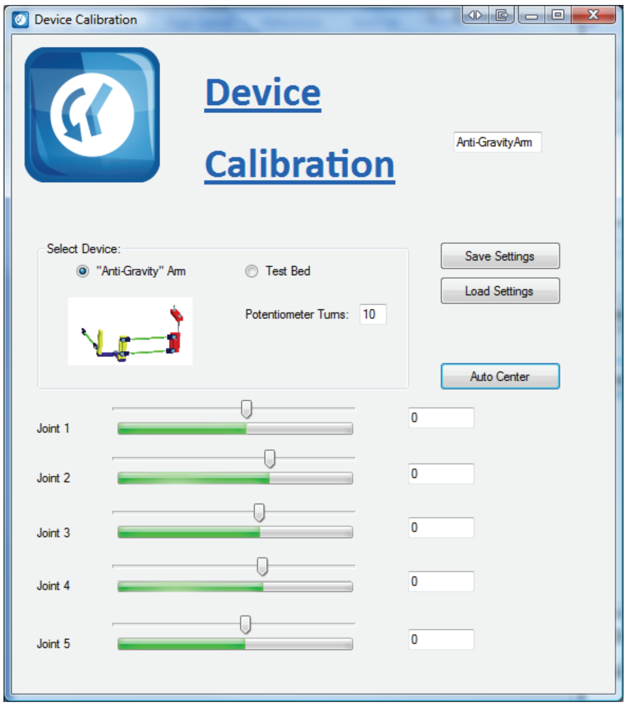
These buttons are used to signal the interface module to open and close the connection. It should be noted that once the interface module has established a connection, it will be maintained until it is manually closed or all processes utilizing the module are closed.

- Force synchronization

This button calls the synchronization in the interface module, it should very rarely be required, however it is helpful when using USB to serial converters.

5.4.3.2 Device Calibration menu

The second GUI designed for the interface module was the device calibration menu (Figure 19). This menu is used each time 'anti-gravity' arm is adjusted for a particular user. Due to the structure of the 'anti-gravity' arm, the centre position of each joint is dependent of the user's personal physical attributes and the seating position. For this reason the arm may need to be re-calibrated in between sessions to ensure that the arm position is measured accurately. In addition to marinating the centre of the arms position, it also enables users to save and load a calibration so that the information can automatically be recalled each time a game is started.



Anti-GravityArm

Select Device: [✓ "Anti-Gravity" Arm] Test Bed

Potentiometer Turns: 10

Joint 1 0

Joint 2 0

Joint 3 0

Joint 4 0

Joint 5 0

Figure 19 - Device Calibration Menu

5.5 Conclusion

In this chapter the design and implementation of the interface between the computer and microcontroller was designed and implemented. The development cycle was identified to include three main stages, concept, specification and design and implementation.

The concept involved commencing from the basic function to transfer data via the serial port. This also included some ideas as to interrupts and shared memory will be used in the program.

After the concept was selected, the specification was created to outline the requirements and design outcomes. This also included information about how the function will behave and how data will move between different processes. The specification was then divided into logical related segments for implementation.

The implementation was conducted over three sections; the miro-controller program, the computer interface program and the GUI for configuring the device.

Chapter 6
Game Software Development

6.1 Introduction

The aim of using virtual reality games in the rehabilitation process is to provide an environment in which patients are able to conduct range of movement exercises in a rewarding and intuitive environment. The success behind using these video games in conjunction with main stream rehabilitation is due to the games ability to provide motivation, and short term goal achievement with successful physiotherapy techniques.

6.2 Game Concept

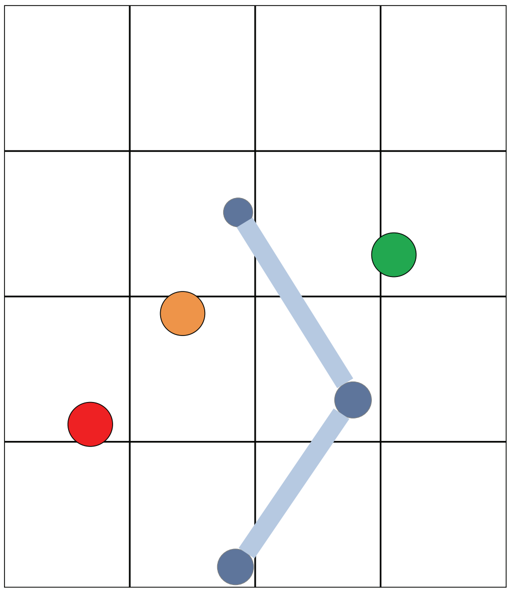


Figure 20 - Game concept

The concept behind the game is a simple movement program whereby the user will be required to reposition the arm so that the hand is in line with dots on this screen. At the first level, the user simply has too position the hand in correlation with a green dots, regardless of the movement at the shoulder which in turn influences arm height. This is often one of the preliminary exercises that physiotherapists may use, as the patient has little control over raising and lowering their shoulder. While the user is 'collecting' the dots, the time taken and number of successful dots collected will be recorded. The second level requires the user to collect orange dots below the mid height shoulder position. As the user’s ability progresses both green dots and orange dots are given. The game then encourages the user to move faster to target the different dots, which in turn requires that user to have more control over the coordination and speed of their arm.. Finally red dots are included which require the user to perform positions above the shoulder mid height position. These levels act as intermediate goals which can be saved and recorded to demonstrate the user's progress.

6.3 Programming Computer Graphics

Programming a game is entirely different from programming a sequential application. Programming a computer-aided game involves displaying meaningful graphics and allowing the user to interact. There are several ways in which graphics generated by a program can be displayed:

- Frame-by-Frame - A single frame is draw individually as the user waits
- Frame-by-Frame under user control – A sequence of frames are drawn and the user provides input to change frames (Power Point slides)
- Animation - A sequence of frames produced at a particular rate.

A computer game uses a combination of these options and the user controls the flow from one frame to another. The program itself is in a continuous loop modifying objects and redrawing the display. It is important that the program utilizes non-blocking I/O and is never caused to halt while waiting for calculations or functions to finish.

6.4 Game Objects

The game was developed in XNA Game Studio and explanation behind this particular games selection is the System Development Chapter 4. All programming is to be implemented in C sharp in Microsoft visual development environment. C sharp is an object orientated language whereby each object has a data structure or class with defined attributes and functions that relate to the particular object.

6.4.1 Arm Object

The Arm object is responsible for updating the position of the users arm and drawing it to the screen. It is also responsible for maintaining a hit box, which is used to detect collisions between the arm and the different balls.

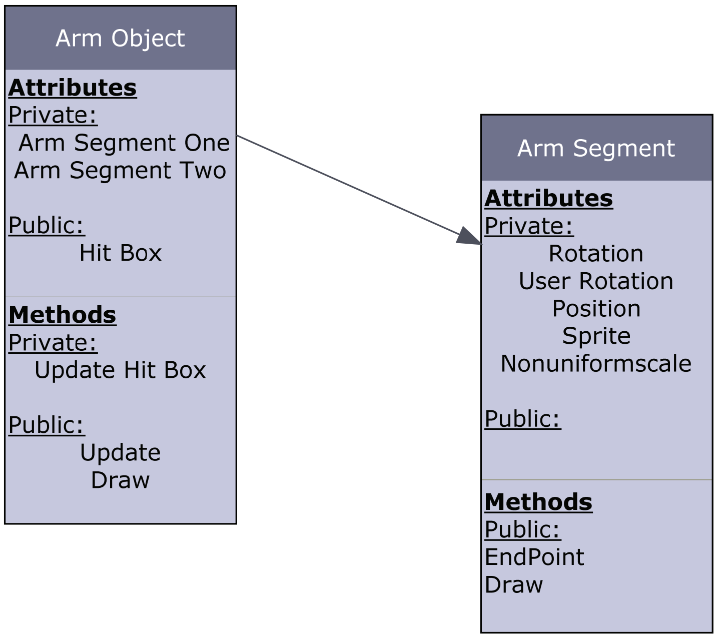


Figure 21 - Arm Object Classes

The arm object (Figure 21) has one public member called the "Hit Box". This is a rectangle class provided by the XNA games engine for storing the coordinates of a rectangle. The arm object class provides this rectangle for use when detecting if the user's hand as intersected a target. The private function updating the global position of the hit box position is called "Update Hit Box". This function is only called from within the class by the public function "Update". The Update function is also responsible for fetching data from the Interface Module and manipulating the scale and rotation of the arm segments (Figure 22).

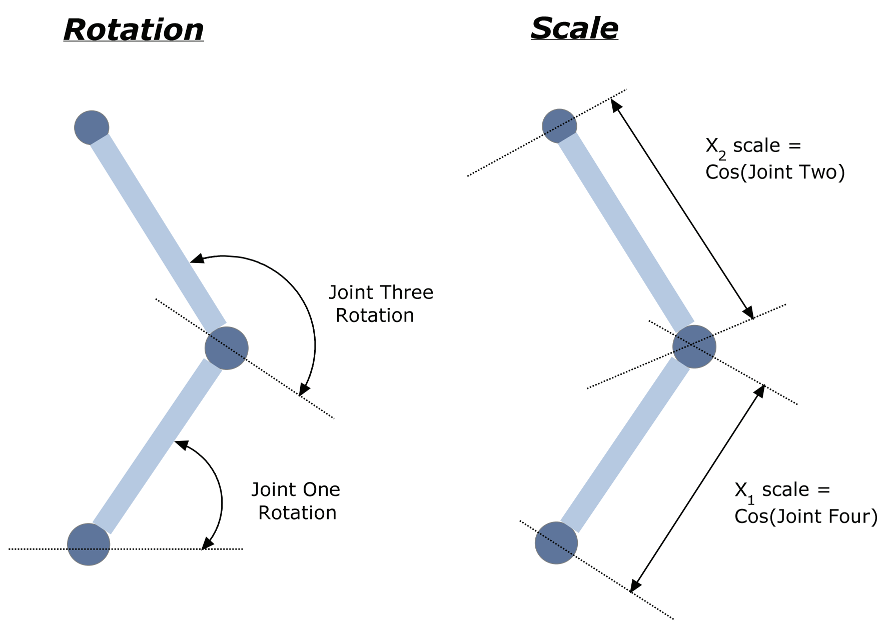


Figure 22 - Arm Object Scale and Rotation

The Arm Segments are individual object contain the following information:

- Rotation – this is the rotation of the object from the angle that the sprite is drawn to initial angle that the arm starts at.
- User Rotation – this is a data field that maintain by the update function and is used to draw the sprite.
- Position – this is the two dimensional origin of the arm segment that has translated to the centre of the pivot.
- Non-uniform scale – this is a 2 demotion vector containing the x and y scale of the sprite. When scaling a sprite, it scaled by both height and width from the original orientation.
- Sprite – A sprite is term that refers to a two dimensional image in a game. XNA can import and process sprites in a number of different formats including JEPG, BMP and PNG. Sprites have to a rectangular shape; however this is not always convenient. Transparency is required to display a non-rectangular image, whereby the sprite is manipulated as a rectangular but only parts of the image are shown on screen. The sprites used for the arm are show in Figure 23. This image has then been split into the two segments and the background of the image has been set to transparent.

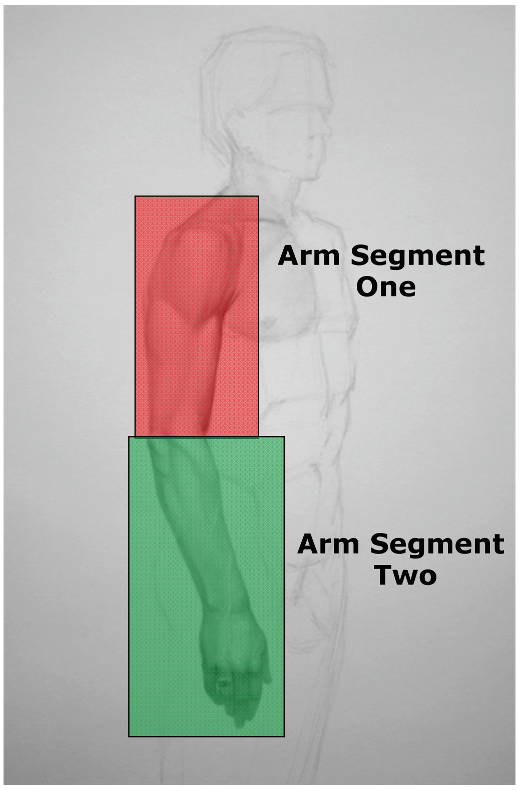


Figure 23 - Arm sprites

- Each arm segment has its own draw function within the class, this function is called only by the arm object so that drawing the complete arm can be perform from a single line of code.

6.4.2 Targets Objects

There are three different types of targets and the game uses 15 targets each level. By using a general class this can be expanded or modified by simply suppling more textures and rules to decide when and where the targets will be placed. When an image is drawn in XNA game studio it is drawn from the bottom left hand corner. This is why the target class (Figure 24) maintains both the centre and position variables. The centre position is used for positioning the target and maintaining hit box's position

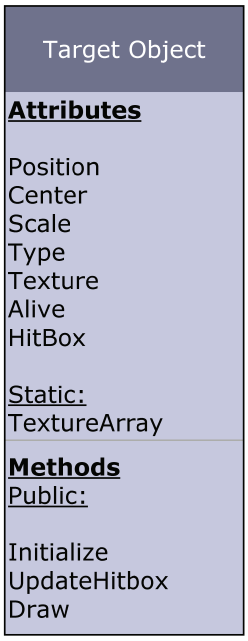


Figure 24 – Target Class

The Target class also contains a variable called *type*; this variable is used to distinguish between the different types of balls used in game. The type of ball determines which behavioural set is used when the ball is loaded and intersected by the arm. The target class also includes a static member called *texture array* which is common for all target members and contains the target sprites (Figure 25). When a target class member is instantiated the particular sprite from the texture array is assigned to the member and the height and width of the sprite are reflected in the hit box dimensions.

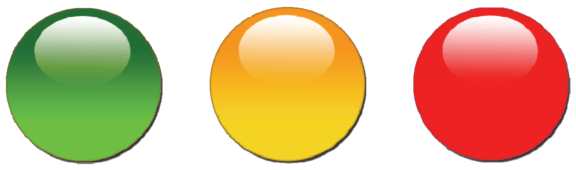


Figure 25 - Target Sprites

The target class also include two method calls: UpdateHitbox and Draw. The UpdateHitbox function is very similar to the one used in the arm class. It is called after a target has been placed to create a rectangle with the centre aligned with the object with the dimensions of the sprite. The second function is the draw function. By storing the sprite name and draw method within the class, an array of targets can be drawn from a single instruction.

6.4.3 Scoreboard Object

The Scoreboard Object (Figure 26) is responsible for keeping the score and level variables. However these variables cannot be modified within the class. Its main function is to draw the scoreboard sprite and displaying the user's current score and level.

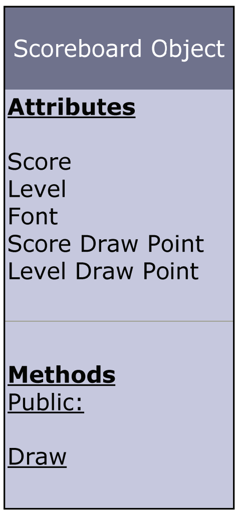


Figure 26 - Scoreboard Class

The score and level are continually changing throughout the game, therefore these sprits will have to be generated in real-time. This is essentially handled by the XNA Sprite Batch class which makes use of a XML font file. The fonts are then converted in a scalable vectors image which is finally converted into 2D sprites. The scoreboard class then uses two positional vectors (Figure 27) to located and draw the images.

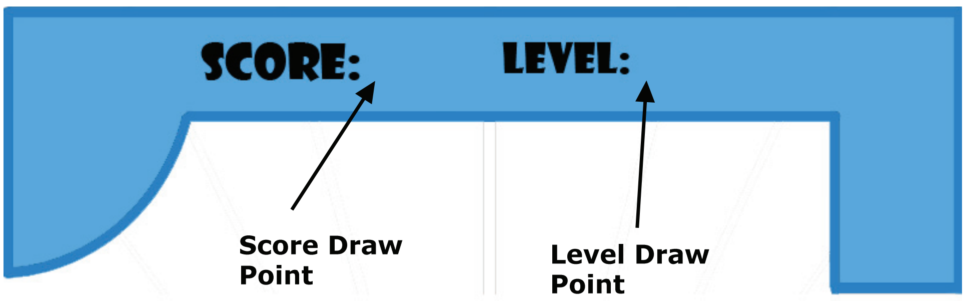
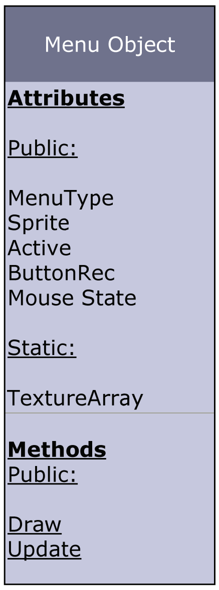


Figure 27 - Scoreboard sprite

6.4.4 Menu Overlay Object

The menu screen is used in the game to explain the rules of each level. The Menu Overlay (Figure 28) class is very similar to a target class, in that is make use of a texture array where the menu type determines which sprite is used. The menu uses a Boolean variable called *active* which is used to determine if a new menu is to be drawn.



Menu Object

Attributes

Public:

MenuType
Sprite
Active
ButtonRec
Mouse State

Static:

TextureArray

Methods
Public:

Draw
Update

Figure 28 - Menu Overlay Class

The menu also allows the user to click the “ok” button so that the game can be reset and the new level can be loaded. The menu class does this in the update function, which fetches the mouse state from the XNA library. The mouse state includes the mouse position and button state. After the state is stored, it is tested to find out if the mouse pointer is within the button rectangle (Figure 29). A mouse click is when the mouse button has been pressed and released. Each time this function is performed the previous state is recorded and compared to the current state in order to test for a mouse click.

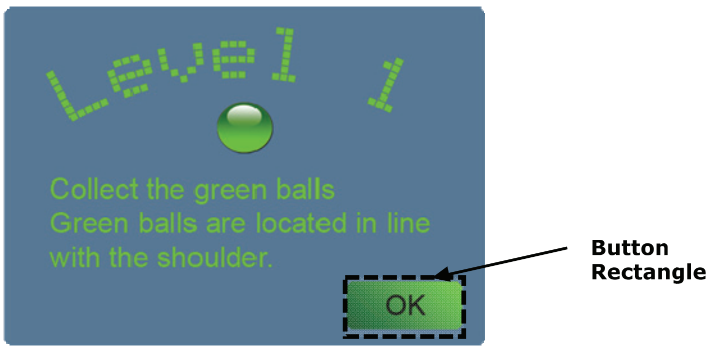


Figure 29 - Level One Menu

6.4.5 Level Object

The level class contains the rules for each level and methods for progressing through the games. Each level is stored in a static array within the class, whereby the level number is used as an index. The level class is also responsible for detecting to collisions between hit boxes and showing the menu at the start of each new level.

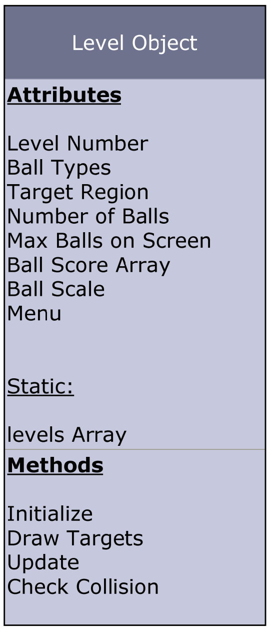


Figure 30 - Level Class

The first function to be called in the level class is the initialise function; this function is used to position the ball on screen and displaying the level menu. The variable ball type is used to indicate which types of balls are permitted in certain levels, the permitted types include:

- Type 1 - Green Balls Only
- Type 2 - Orange Balls Only
- Type 3 - Red Balls Only
- Type 4 - Green and Orange Balls Only
- Type 5 - Green and Red Balls Only
- Type 6 - Orange and Red Balls Only
- Type 7 - All Balls

A random set of balls are selected based upon the permitted type and are then stored in a target array. The initialise function then places these balls in random positions within the target region. After a ball has been placed it must be checked to ensure that it is not in a position where the ball does not intersect any objects that have already been placed. It does this by calling its own function called "check collision" which takes a rectangle and compares it to any active targets already within the level. This function is also used to detect collisions between the hand and the target objects. The update function within the class is then used to keep track of the number of balls still active and keep track of users score.

6.5 Conclusion

The development of the games started by review the project aim which was to develop a game that could be used in conjunction with standard rehabilitation programs and that will motivate patients by creating intermediate achievable goals. The next phase in the development process was to form a practical concept idea which suited the physical limitation of the patients and was also appropriate with the time considerations of the project.

The development process then considered how the game would be displayed by the computer and how the user's input will affect the graphics being displayed. The graphics of the game was based around three concepts; Frame-by-Frame, User Controlled Sequencing and Animation.

The particulars of the development environment selected in Chapter 4 were then reviewed. It was found that XNA Game Studio has an object orientated structure where each object in the game becomes a class in the program, holding attributes and methods explicitly related to the object. Each object of the game in the conceptual design was then allocated a class structure. The fundamental attributes and methods of each class were then defined; however these classes were often revised during the implementation stage.

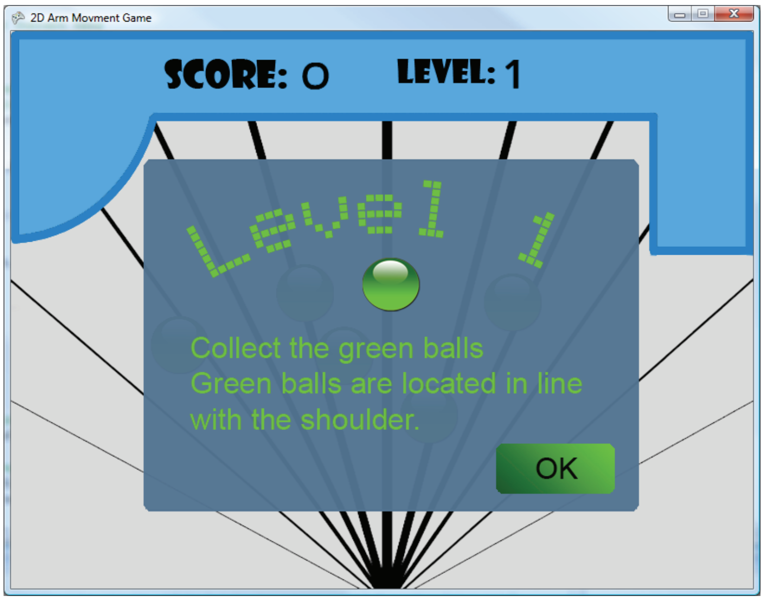
Chapter 7
Testing and Evaluation

7.1 Introduction

Testing and evaluation will be conducted over three main sections of the project which are the interface; the accuracy of measurement and the game it's self. The elements of the project will be tested to ensure they work as designed in the project specification. It is important to understand that the project still a lot more development required before it is suitable as a commercial device and at this stage the device is simply a prototype. Meaning that any software developed or hardware used is simply a proof of concept.

7.2 Game

The aim was to developing a game that it could adopt into the rehabilitation program and that will motivate patients by creating intermediate goals. The game design has been successfully in incorporating a series of changing exercises for the user to complete. Through the uses of different levels the game is able to adapt as the varying degrees of inability of the stoke patients.



Collect the green balls
Green balls are located in line with the shoulder.

Figure 31 - Games Level 1 screen capture

The game graphically represents the user's arm movement in two dimensions and makes use of scaling to achieve the appearance of three dimensions as the arm is moved in the vertical plane. While this is a reasonable way of representing rising and lowering of the shoulder and elbow, a threshold system is used to detect the arms height relative to the individual targets. This may prove to be an inappropriate technique.

The original software designed for the 'Anti-Gravity' arm preformed endpoint calculation on the micro-controller and transferred the endpoint coordinates to the computer. The new game software designed by this project calculates three dimensional coordinates of all points on the arm in real time. This holds sufficient advantages over the original software created for the device. One of the major advantages is that each joint of the arm can be monitored and visual feedback can be provided. This means that the software can distinguish between different positions or exercises that have the same hand position.

The game software has achieved all the design objectives and has been developed as a great proof of concept. However more testing and refinement of the programming needed to confirm the concept. This could include a combination of clinical trials and seeking the advice of a health care professional which is very likely result in a revision of the original specification.

7.3 Interface

A large portion of the project is focused on designing software and hardware to accurately measuring use's arm position in displaying it movements on a computer screen.

The hardware designed consisted of the selection of components for measuring joint rotation and send the information to the computer. The measurement of the arm's rotation has been achieved through the use of potentiometers which have proven to be an inexpensive option which still maintains a high degree of precision.

The PICAXE microcontroller was selected for processing and sending the information to the computer. Using the speciation of these devices has been math calculated accurately measure the rotation of each join within 3.51 degreases.

The testing of the interface performed over two types of tests shown below:

- Short Term Testing – the first test was to check the recoded joint rotation was a turn representation the joint of the device. This was performed by comparing the recoded joint rotation to the measured device rotation using a protractor. It was found the recorded rotation was within 3 degreases tolerance as expected.

  The next test was to start the game and configuration menus along with a 15 other application to ensure shared memory did not become corrupt as time slicing become non-periodic. This test was all successful as no discrepancies were found.

- Long term Testing – this test was also successful conducted by set the program to output the number of successful read every five minutes then allow the program to for 48 hours, no discrepancies were found.

The interface and position measurement techniques developed in this project holds as sufficient advantage over the original design. As the software is completely interrupt driven and there are no apparent delays cause by I/O with the device.

7.4 Accuracy

After the antigravity arm had been successfully interfaced with computer it was important to ensure that the graphical movement of the arm was a true representation of the user's movement. The structure of the antigravity is difficult to manually measure each individual DOF accurately. As the device is difficult hold in a set position for any length of time.

In order to accurately measure the position of the arm in correlation to data collected by the computer Leica ScanStation was used. The Leica ScanStation is a high-definition survey (HDS) system which moves a laser across the target area recording the x, y, and z coordinate of each point. In addition of collect the coordinates, it is also take a photograph at the same time which can used to identify the coordinate which are applicable to different parts of the target. HDS systems hold sufficient advances of the traditional measurement techniques in that they are able measure all dimensions and location of the subject within a small time frame. This held great advances for measuring the Anti-Gravity' arm as it is difficult to hold the device in a set position.

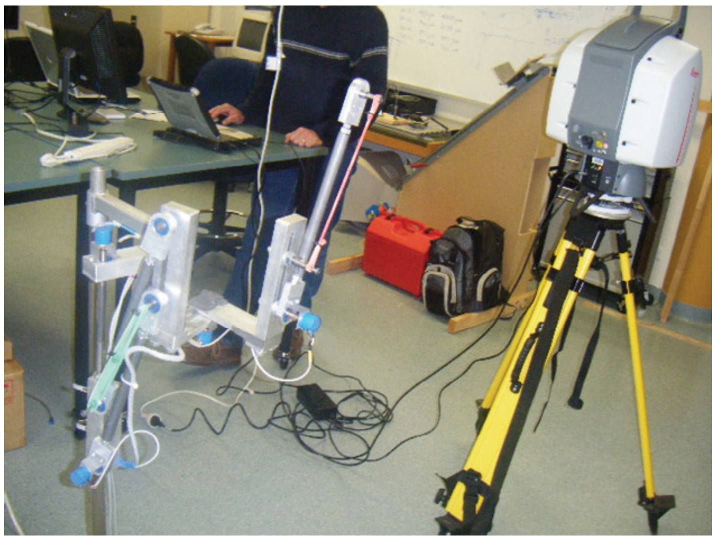


Figure 32 - Leica ScanStation

The software that is provided with the scan station is called “HDS cyclone”. There are three module provided in the package:

- Cyclone-SCAN – This is the software used interface and controlling the Leica ScanStation. This software also includes tools for extracting target area from the scan region.

- Cyclone-RESGISTER - This is provides of a complete set of tools for aligning point clouds captured from different scanning positions, quickly and accurately.

- Cyclone-MODEL – the modelling software is used for viewing, manipulating and measuring parts of the point cloud.

Using these tools the data is then represented in what is called a point cloud (Figure 33), from which global vectors can be calculated for each segment of the arm and the rotation of the joints, be compared to the results recorded by the software.

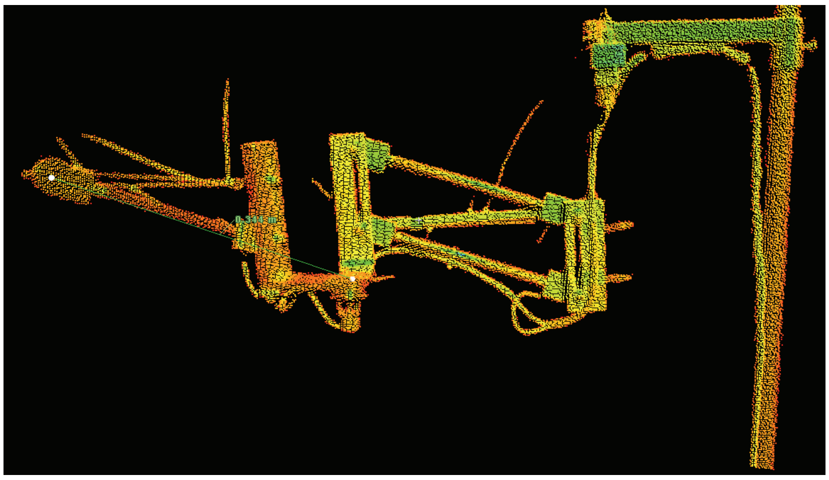


Figure 33 - Point Cloud

In order to stream line this process a testing program was developed which incorporated the interface module and c# excel macro functions so that individual coordinates could be calculated and recorded in real time. After the collected data was analysed it was found that the system works within a 2.5% tolerance.

7.5 Conclusion

Testing and evaluation has taken place over the three main sections of the project which are the interface; the accuracy of measurement and the game it's self.

The accuracy of measuring 'Anti-Gravity' arm joint rotation and three dimensional coordinates the position of the arm was found to be quite reasonable. The individual joint rotation was measured manual and compare to the rotation recorded by the computer. It was found that each joint rotation is measured with a 3 degree tolerance. The position of the arm is captured a high-definition survey (HDS) system and compared to the position calculated the game software. It was found that three dimensional coordinates of the arm position are accurately displayed with a 2.5 percent tolerance.

The two major sections of software produced by the project are the interface module and the game. The interface module was tested under various conditions and found to be a reliable system which successful prevents delays occurring during read operations.

The game was reasonably successful in that incorporated exercises that could adopted into the rehabilitation process. The game has achieved all the design objectives of provide, however it was suggested the game requires further testing and a possible revision of the specification.

Chapter 8

Investigation of an Active Support System

8.1 Introduction

In addition to successfully measuring the position of the user's arm and creating an exercise game, the project also conducted an investigation into a suitable design of an active system for the 'Anti-Gravity' arm. This active system is used to apply a dynamic force to assist or restrict the user's movement and can be used to add an addition level of complexity to the game

In keeping with one the original key focuses of the 'Anti-Gravity' arm, the expense of the active system needs to be kept to a minimum, and this became an important consideration.

8.2 Active and Passive Support Systems

The current "Anti-Gravity" arm support is a passive device; it is quite successful in supporting the majority of the weight of the patient's arm and at removing some of the efforts involved in movement. Passive devices support the weight of the affected limb by applying constant force and typically this force is generated using springs or rubber bands.

An active support system is one where the device can apply a dynamic force to assist or restrict the user's movement. The active system is comprised to two main elements, generating the applied force and predetermining where and how much force is required. One example of a active support system is the IntelliArm 7 which was described in Chapter 3.

8.3 Current Passive System

The "Anti-Gravity" currently uses rubber bands to support the combined weight of the device and patient's arm. In chapter 2 an evaluation of the design was conducted and found the major inefficacies are that:

- The device requires manual setup to adjust the device setup and change the rubber bands in order to achieve the required balance force.
- The device does not provided force-feedback to the users' movement for a virtual reality use.

- The device is not capable of providing varying degrees of difficulty as the patient progresses.

A further investigation into the passive forces present in the device is required in order to implement an active system. As stated above, a passive system works by neutralizing the weight of the arm by applying a constant force over a range of movement. The first step was to evaluate the devices ability to perform this task. There are two main elements which affected the force provided; the rubber bands and the mechanical structure.

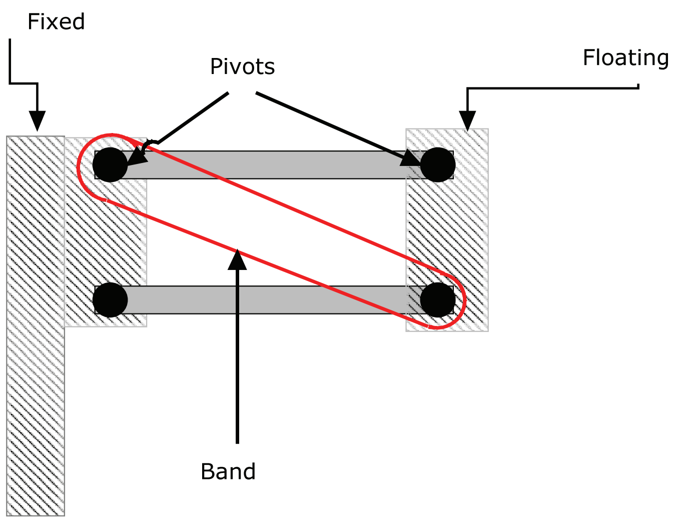


Figure 34 - Mechanical Structure

The mechanical structure reasonable for providing the supporting force is shown in Figure 34. It works by having two struts supported by pivots in between the fixed and floating sections of the arm. The band is positioned diagonally between the two pivots. As the floating section of the arm is lower due to weight of the patient's arm, opposite force is created to balance the arm. These structures are used by the 'Anti-Gravity' arm to support both the lower and upper sections of the arm.

It was important to understand how this structure performed at balancing the weight of the arm. This was done by testing the force provided by the rubber band as the device changed position. The first stage in testing was to measure the distance between the two pivots at the minimum, mid and maximum positions.

Table 5 – Pivot distance at different positions of the arm

Minimum position (above the shoulder joint)	220mm
Mid Position(parallel to the shoulder joint)	325mm
Maximum Position(bellow the shoulder joint)	370mm

The forces provide by the rubber bands then had to be measured to the different extension lengths found above. This measurement was completed by using a material strain testing machine called a tensometer made by Houndsfield. The machine is predominantly used for testing the breaking point to solid materials however it was very suited for this application. It works by stretching the material over different extension lengths, recording the force through a strain gauge.

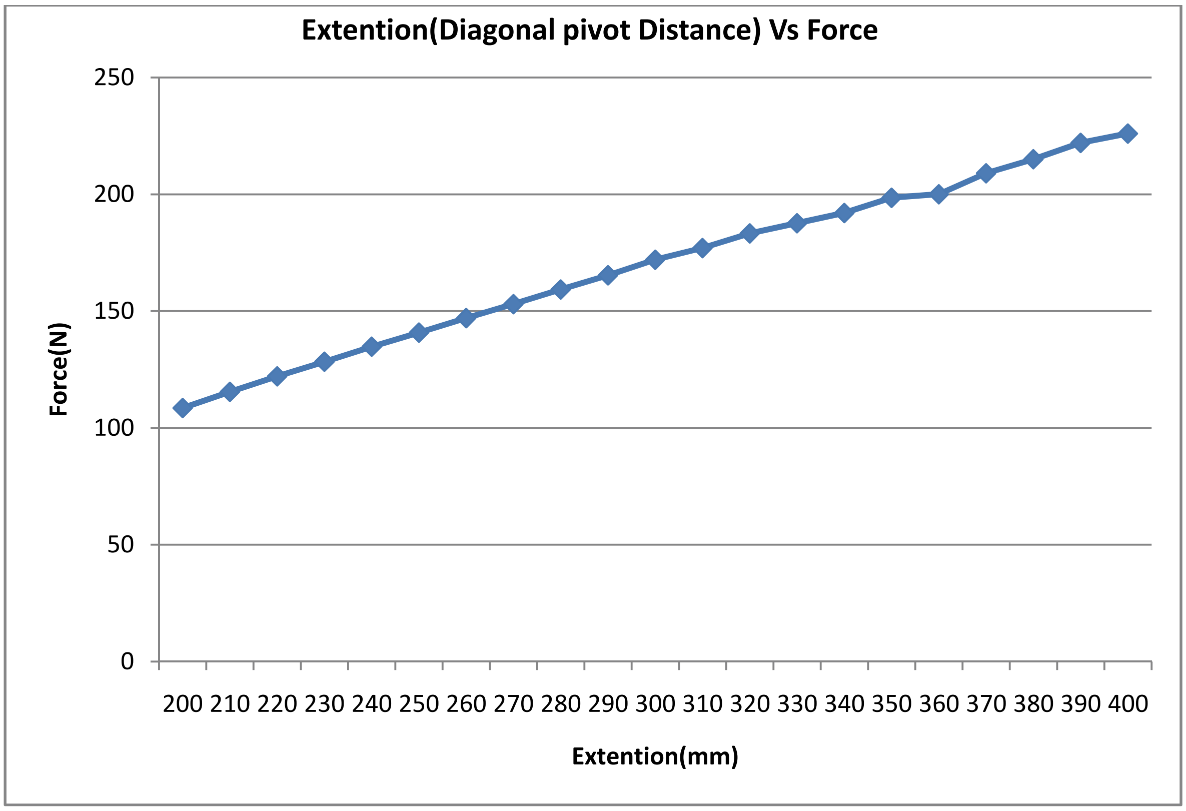


Figure 35 - Extension Vs Force of Rubber Bands

Table 6 - Position Vs Applied Force

Position of the arm	**Pivot Displacement**	**Applied Force**
Minimum position (above the shoulder joint)	220mm	183.25 N
Mid Position(parallel to the shoulder joint)	325mm	184.64 N
Maximum Position(bellow the shoulder joint)	370mm	209 N

These results concluded that the force provided by the bands is a linear relationship to the extension. It was also found that an active system is required to provide a dynamic force between 180 to 209 newtons of force.

The next stage of testing involved finding the force required to balance the device at different positions. Unfortunately I was unable to obtain an appropriate measure apparatus for performing this test. However a basic relationship was found using a cable and spring strain gauge.

The first position that was tested was the mid position (Figure 36). At this position the arm support behaves quite well and the force in the bands creates an applied force which matches the downward force cause by the weight of the arm.

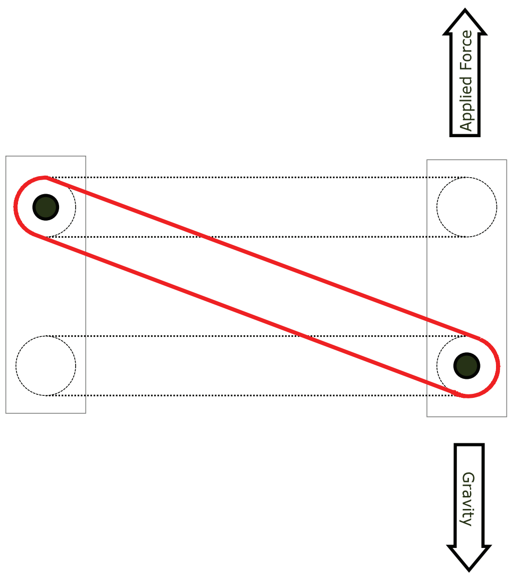


Figure 36 - Mid Position Force

At the minimum position (Figure 37) whereby the arm is below the shoulder, the distance between is pivots are increased and the band is extended. This creates a larger force in the band and hence a larger applied force. This causes the arm to become unbalanced.

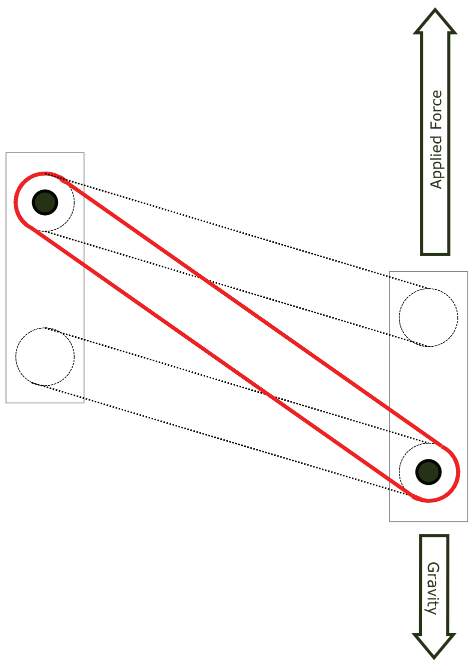


Figure 37 - Minimum Position Force

A similar relationship can be found when the arm is at its maximum position (Figure 38) and the length of the band is shortened. This also causes the arm to become unbalanced as the applied force is now less than the force supplied by gravity.

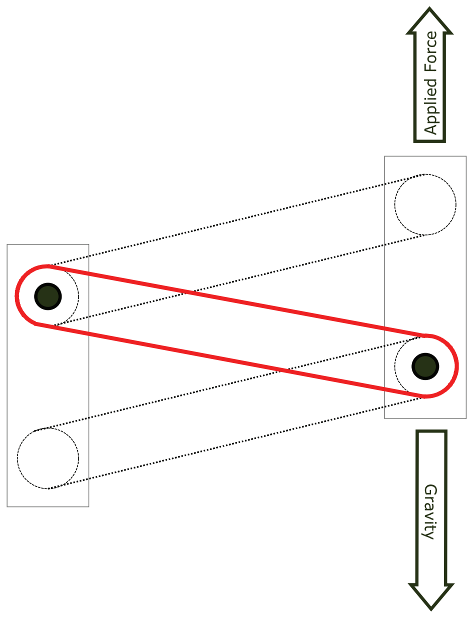


Figure 38 - Maximum Position Force

By sketching the force over the different position it was found that passive force supplied the rubber bands is very linear, however the required force required maintain a constant load is not linear (Figure 39). As this would require the maximum force when the arm is in the mid position, horizontal to the shoulder. This means that while the anti-gravity arm is able to significantly reduce that the amount of effort required by the user, it will never be able to completely neutralize the weight of the arm for all the different position using the current mechanical structure.

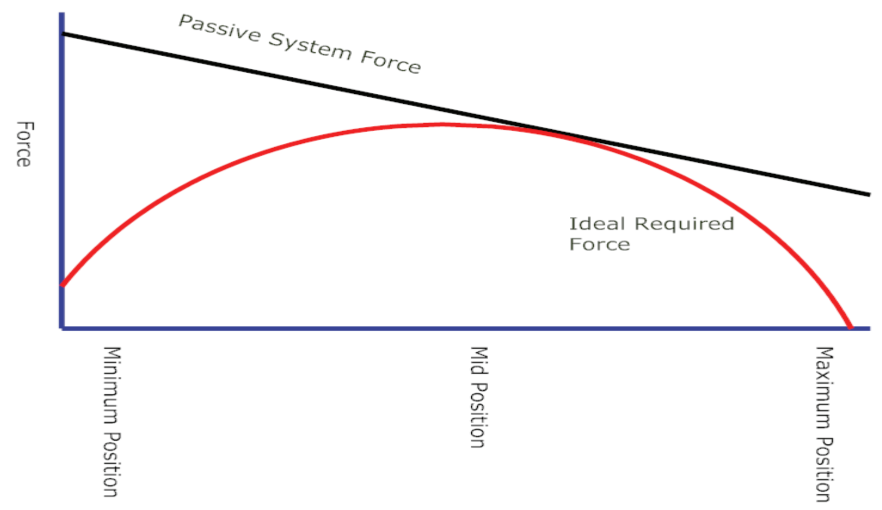


Figure 39 – Sketch of the Required Force Vs Passive Force

This is where an active control could be used to ensure the correct dynamic force is applied for different positions of the arm. Active control will also enable the apparatus to provide assistance or add resistance based on the patients progress and automate the setup procedure for different patients.

8.4 Conceptual Design

The concept deign for implementing active support system is to replace the current passive rubber bands with a electronically controlled mechanical device, which apply a dynamic force to assist or restrict the user's movement. There are a number of different ways active control could be implemented which, so of which include motors or linear pneumatics.

- DC Motors

DC Motors (Figure 40) could be used to create active control by providing the motor with the appropriate stall current to apply the desired force. The downside of using motors is that they would add considerable weight to the arm and require a very large current to maintain a constant force.

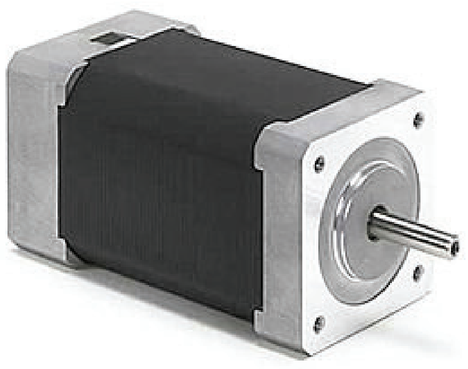


Figure 40 - DC Motor

- Linear Pneumatics

Linear Pneumatics wok by applying pressure to the inlet forcing the piston to move out. Then applying another pressure to the other inlet to force the piston to return. Linear pneumatics are capable of achieving large forces even at low pressures. The downside of linear Pneumatics is that due to the dual control it is very hard to produce a set desired force. The dual control also trends to lead to position control as the force provided by gravity is not enough to return actuator.

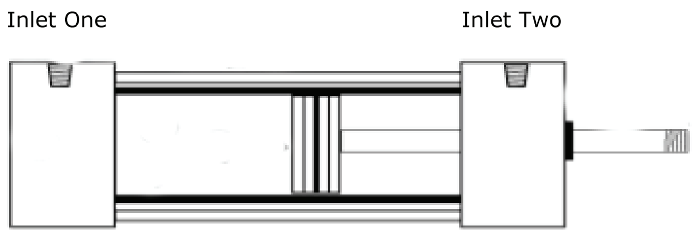


Figure 41 – Linear Pneumatic cylinder

An alternative to linear pneumatics are Air Muscles. Air muscles are light weight and very inexpensive typically costing around 10 dollars each depending of the size. They can provide 400 times more force than their weight.

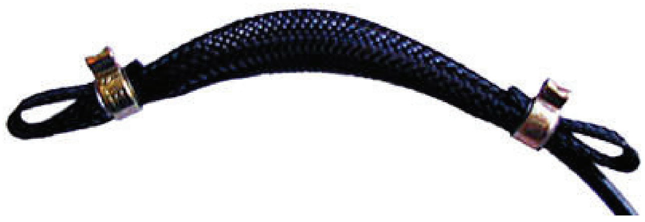


Figure 42 - Air Muscle (Shadow Robot Company 2009)

8.5 Air Muscles

The Air Muscle (Figure 43) consists of a rubber tube covered in tough plastic netting which shortens in a scissor action when inflated with compressed air.

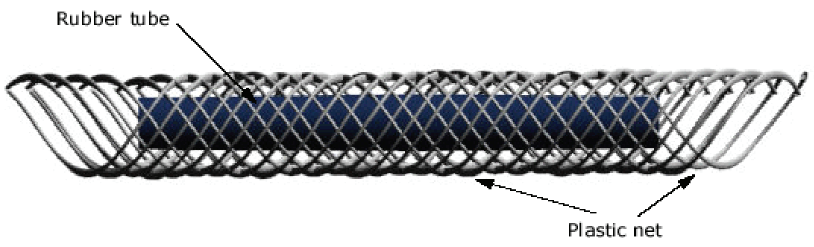


Figure 43 - Air Muscle Structure (Shadow Robot Company 2009)

The disadvantage with air muscles is that there is not a linear relationship between air pressure and force provided. This makes them somewhat difficult to control. Another disadvantage with the air muscle is that they have a very short stroke length around 40 percent of their original size.

8.5.1 Electronic Control of Air Muscles

In order to implement active system utilizing air muscles to generate the dynamic force, an electronic system needs to be designed so the computer can control how much force is applied. There are many different ways of electronically controlling in the pressure in air muscles and these include:

- Electronic regulator

  The best method of electronically controlling in the pressure is the electronic regulator produced by SMC . This device is a self contained precise instrument that is very simple control with a computer or microcontroller. However they are very expensive costing well over $4,000

Figure 44 - electronic regulator (SMC)

- Proportional valve with pressure sensor feedback

  Proportional valves (Figure 45) are essentially a flow rate control valve that uses a servo to open and close the valve. This option would provide a smooth transition between different pressures, however are quite expensive.

Figure 45 - Proportional valve (SMC)

- Solenoid valve with pressure sensor feedback

Another option is to use an on/off solenoid valve (Figure 46) to pulse the air in order achieve the correct pressure. This option may result is a rough transition, however given the device is designed for inexpensive home use, this option is most appropriate.

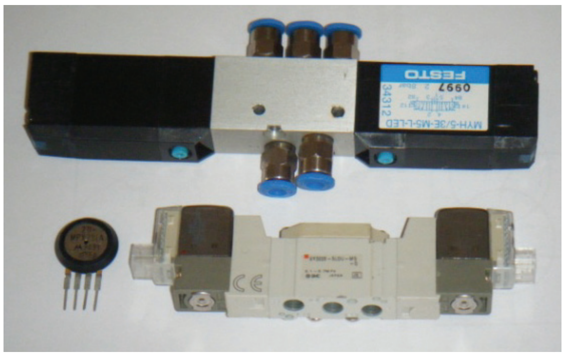


Figure 46 - Solenoid valve (SMC & FESTO)

8.5.2 Computer Control of Air Muscles

In order to accurately control the air muscles with a computer, the program needs to be able to determine how pressure is required to achieve a force. From the analysis of the 'Anti-Gravity' arm structure it was found that the required force to maintain a constant load at different positions is not a linear relationship. A mathematical model is required which relates:

- Extension of the Air Muscle
- Pressure in the Air Muscle
- Applied Force by the Structure

The extension of the air muscle can be determined by the computer as it has a simple trigonometric relationship to the rotation of the joints. The pressure in the air muscle will have to be monitored separately by amplifying the output pressure transducer (Figure 46) and using an additional ADC input of the microcontroller. The applied force by the structure is a variable which has been measured and incorporated into the model. In order to find this model and have a better understanding of how the devices work a series of air muscles were constructed (Figure 47).

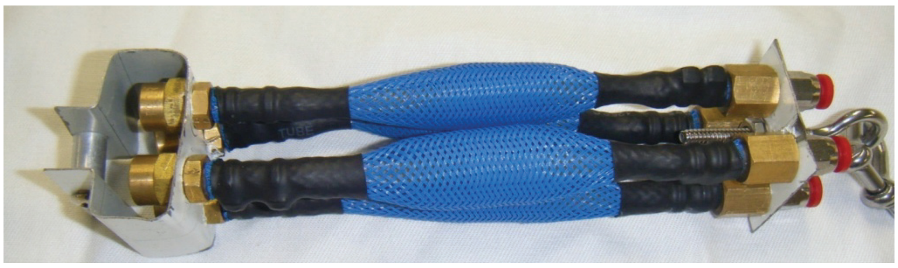


Figure 47 –Constructed Air Muscles

Preliminary testing on the on the air muscles found that each small muscle show in figure 47 is able to produce 80 newtons of force over an extension of 5 cm. However due to the time restrictions of the project further testing was not able to be conducted and the model could not be formulated.

8.6 Conclusion

An active support system of the 'Anti-Gravity' arm hold great advantages over the current passive system in that it will enable the device to automatically adjust for different users and will provide force-feed back to the users movement. It will also enable the 'Anti-Gravity' to completely neutralize the weight of the arm for all position.

From the investigation it was found that air muscles are an inexpensive means of generating the driving force for an active support system and hold many advantages to more traditional methods.

Chapter 9

Conclusion

9.1 Introduction

The aim of this project was to design and develop a virtual reality game to assist in the rehabilitation of the motor function in the upper and lower arm of people that have been affected by stroke. An underlining factor in the design was to limit the expense of the device, making it more accessible for community rehabilitation clinics and individual home use.

This chapter will provide an overview of the project, discuss the research conducted and consider aspects of the system design. Finally, the evaluation of the project will be reviewed and suggestions for the further development of this project will be discussed.

9.2 Discussion

Before commencing the project it was important to develop an understanding of the need for such a system and how it would complement the current rehabilitation process. In the first chapter it was found that there is a dramatic fluctuation in the number of strokes occurring in Australia each year, which is in turn putting additional strain on an already challenged Health Care System. As a direct consequence, there is escalating motivation from both the Federal Government and the private sector to improve and create more established home and community base rehabilitation programs.

Health care professionals are slowing recognising and embracing the great potential for virtual reality games to be included in the rehabilitation process as a method of complementing tradition therapy. This has become evident in that new technologies based on computer games, such as the Nintendo WII, have already been incorporated into the current rehabilitation programs for patients. The successful incorporation of these new potential treatment modalities exists because of the similarities between the aspects of rehabilitation and the aspect of the game theory. These include:

- Varying levels of difficulty

- Goal orientation
- Intensive repetition

The next chapter provided research into the background behind stroke, which was useful in establishing an understanding of why strokes occur and what inabilities and functional limitations that stroke victims suffer from. This research also included a study into the traditional rehabilitation techniques. It was found that stroke rehabilitation is focused on the recovery of muscle control and sensory ability, in an effort to aid a return to normal function and to make normal day activities as effortless as possible. Physiotherapists are a vital part of the rehabilitation process by creating ‘tailor-made’ exercises programs that are best suited to the inabilities of each individual patient. The exercises performed with the assistance of the physiotherapists fall under three categories:

- Passive Range of Motion
- Active Range of Motion
- Active Assisted Range of Motion

Using this new obtained understanding of the research a review of the ‘Anti Gravity’ arm was commenced. It was found that the movement supported by the device was best suited to Active Assisted Range of Motion (AAROM) exercise. Whereby the body part is supported by the device and the patient is able to move freely using their own control. This review also included the technical aspects of the device which would then assist with the system development.

After the research was conducted, the general conceptual design was formulated and broken in manageable sections, which could then be individually addressed This section also addressed both hardware and software design.

The hardware design incorporated the selection of all components from equipment responsible for accurately measuring the position of the arm to processing the data and transferring it to the computer. The final design involved measuring the rotation of each joint using 10 turn wire wound potentiometers, processing the analogue signal using the PICAXE X1 and transferring the data via a serial connection to the computer.

The software design concluded that XNA Game Studio would provide the best integrated development environment and that all coding would be performed in Microsoft Visual C sharp. This option was selected because of its advanced vector and mesh manipulation libraries and intuitive design. Great work

The game consisted of a simple movement program, whereby the user is required to reposition the arm so that the hand is in line with dots on this screen. The game was reasonably successful in that multiple joint exercises that could adopted into the rehabilitation process. The game achieved all the design objectives; however it is suggested the game requires further testing and a possible revision of the specification before commercial application.

The system has been successful in accurately representing a user's movement within a virtual environment. This has been tested by use of advance 3D mapping techniques. It was found that the computer software can accurately measure the each joint rotation with in 3 degrees and the final scaled position of the arm can be calculated a 2.5 percent tolerance.

9.3 Future Work

The testing and evaluation of the project has concluded that the system has achieved the majority of the project goals. It has also revealed that the developed system is able to accurately measure the 'Anti Gravity" arm's position. However the project still requires more development and testing before it is ready for clinical trials.

Future work for the project could include:

- Further Research

  Further research is required into how exercises are conducted and the methods used to evaluate a stroke patients recovery. This would also involve the development of a mathematical or heuristic model for determining patient's recovery that could be implemented as part of a computer system for the 'Anti Gravity'.

- Extending the Gaming Environment

The game design is easily extended by creating new levels and target objects. The design also made allowances of the use of 3D objects. This would require more research into exercise programs and would benefit from input from a rehabilitation health care professional.

- Implementation of an Active Support System

This project also conducted an investigation into the = active support for the 'Anti Gravity' and found that air muscles have a great potential for improving the device. However due to time restriction the project did not successfully design a mechanism to provide an active support system.

- Mechanical Re-design for Clinical Trails

This would involve researching the procedure and legislation related to conducting clinical trials. It would also include the design of a more satisfactory of method mounting the anti-gravity arm and addressing various safety issues.

- Mechanical Re-design for Commercial Product

This would involve a re-design of the anti-gravity arm for the release as a commercial product. This would include the structure itself and the addition of plastic aesthetics such as hand grips and guides.

9.4 Conclusion

The project has successfully designed and developed a virtual reality game which can assist in the stroke rehabilitation process. Testing of the system found that it was able to accurately measure the position of the user's arm and provide visual feedback in real time. The project has also indentified further venues in which the project could take in the future.

The 'Anti-Gravity' arm and virtual reality games have both shown a great potential to assist in the rehabilitation of people who suffer from hemiparesis.

This is interesting and captivating field of study which will hopefully one day help to improve the lives of people that suffer as a result of stroke.

References

Alamri, A, Eid, M, Iglesias, R & Shirmohammadi, S 2008, 'Haptic Virtual Rehanilitation Exercises for Poststroke Diagnosis', *IEEE transactions on instrumentation and measurement*, vol 57, no. 9, p. 9.

Australian Goverment Intiative 2008, *Hemiplegia - Job Access*, viewed 24 May 2009, <http://www.jobaccess.gov.au/JOAC/Advice/DisabilityOne/Hemiplegia.htm>.

Australian Institute of Health and Welfare 2009, *Facts, figures and statistics - Stroke Foundation*, viewed 24 May 2009, <http://www.strokefoundation.com.au/facts-figures-and-stats>.

Cambourne, K 2008, 'Wiihabilitation', *The Sydney Morning Herald*, p. 1.

Carden, SB 2008, 'The development of an antigravity arm to assist in the rehabilitation of stroke patients', *USQ*, vol I, no. 1, p. 127.

Elliot J. Roth 2008, *Rehab Wire - Nation Rehabilitation Center News* , viewed 24 May 2009, <http://www.naric.com/public/rehabwire/rw0811.pdf>.

Goverment, Queensland 1999, *Information Kit*, Stroke Association of Queensland, Brisbane QLD.

Housman, S, Le, V, Rahman, T, Sanchez, R & Reinkensmeyer, D 2005, 'Arm-Training with T-WREX After Chronic Stroke: Preliminary Results of a Randomized Controlled Trial', *IEE*, vol I, no. 1, p. 6.

Hyung-Soon Park, YRAL-QZSM 2008, 'IntelliArm: An Exoskeleton for Diagnosis and Treatment of Patients', *IEEE* , vol I, no. 1, p. 6.

J.W.Burke, P.J.Morrow, McNeill, M, S.M.McDonough & D.K.Charles 2008, 'Vision Based Games for Upper-Limb Stroke Rehabilitation', *IEEE Compter Society*, vol 1, no. 1, p. 5.

Johnson, LM & Winters, JM 2004, 'Enhanced TheraJoy Technology for the use in Upper-Extremity Stroke Rehabilitation', *IEEE EMBS*, vol 1, no. 1, p. 7.

Johnstone, M 1987, *Restoration of Motor Function in the Stroke Patient*, 3rd edn, Churchill Livingstone, Hong Kong.

Lars.I.E, Konrad, J, Williams, S., Karlsson, R & Ince, S 2009, 'A Rehabilitation Tool for Functional Balance using Altered Gravity', *IEEE*, vol 1, no. 1, p. 4.

Medical Education Division 2007, *Nursing Fundamentals - Lesson 5: Active and Passive Range of Motion Exercises*, viewed 24 May 2009, <http://www.brooksidepress.org/Products/Nursing_Fundamentals_1/lesson_5_Section_1.htm>.

National Health and Medical Research Council 2005, *Clinical Guidelines for Stroke - Nation Stroke Foundation*, viewed 24 MAY 2009, <http://www.nhmrc.gov.au/PUBLICATIONS/synopses/_files/cp105.pdf>.

National Institute of Neurological Disorders and Stroke 2009, *Post-Stroke Rehabilitation Fact Sheet*, viewed 24 May 2009, <http://www.ninds.nih.gov/disorders/stroke/poststrokerehab.htm>.

Neamen, DA 2007, *Microlelectronics*, McGraw-Hill, New York.

Ohio State University Medical Center 2007, *Health Information Transltions - Range Exercises*, viewed 24 May 2009, <http://www.healthinfotranslations.com/map.php>.

Queensland Health 1999, *Information Kit*, Stroke Association of Queensland, Brisbane QLD.

Shadow Robot Company 2009, viewed 29 October 2009, <http://www.shadowrobot.com/>.

Stroke Recovery Assciation NSW 2005, *After a Stroke - Physical Effects* , viewed 24 may 2009, <http://www.strokensw.org.au/physical.html>.

Washington University School of Medicine 2008, *The Internet Stroke Center*, viewed 24 May 2009, <http://www.strokecenter.org/patients/about.htm>.

Willmann, RD, Lanfermann, G, Saini, P, Timmermans, A, Vrugt, JT & Winter, S 2007, 'Home Stroke Reabilitation for Upper Limbs', *IEEE EMBS*, vol 1, no. 1, p. 5.

Appendices

10.1 Appendix A, Project Specification

University of Southern Queensland

FACULTY OF ENGINEERING AND SURVEYING

ENG4111/4112 Research Project
Project Specification

FOR:	**Ryan Fitzhenry**
TOPIC:	Design and develop virtual reality games utilising the "anti-gravity" arm support rehabilitation therapy.
SUPERVISOR:	Selvan Pather
PROJECT AIM:	The project aims to develop a virtual reality environment in which patients are able to conduct exercises and help to assess the rehabilitation progression. The project will utilise the "anti-gravity" arm support to lessen the affect of reduced muscle strength.

PROGRAMME: *Issue A, 19th March 2009*

1. Research relevant background information on the effects of stroke.
2. Research traditional methods of stroke rehabilitation and assessment of rehabilitation progression.
3. Identify exercises and assessment methods compatible with the "anti-gravity" arm support.
4. Implement hardware and program for position data acquisition.
5. Evaluate development packages and programming languages suitable for interfacing and games development.
6. Develop virtual reality application for exercise and rehabilitation assessment.

As time permits

7. Develop means to dynamically add resistance or assistance to the patient’s arm.
8. Conceptual design of using resistance or assistance to simulate the virtual environment.
9. Adapt interfacing hardware and software to allow inexpensive use at home utilising a games console.

AGREED:

_______________________(student) _______________________(supervisor)

Date: / / /2009 Date: / / /2009

Examiner/Co-examiner: ____

10.2 Appendix B, Rubber Band Testing

Experiment 1: Rubber Band Displacement Vs Strain

Purpose: To measure the strain which the bands are exposed to under a normal load.(Note the position of the parallelogram has been changed to the height is 88mm)

Step 1: Find the number of bands required to lift an average arm weight of 10N.

Step 2: Measure the displacement of the pins at the minimum, mid and maximum positions.

Minimum position (above the shoulder joint)	220mm
Mid Position(parallel to the shoulder joint)	325mm
Maximum Position(bellow the shoulder joint)	370mm

Step 3: Measure the strain vs. Displacement over a range.

Displacement(mm)	**Strain(N)**
200	108.5
210	115.42
220	122.08
230	128.33
240	134.75
250	140
260	153.0
270	159.25
280	165
290	172
300	177
310	183
320	187
330	192
340	198
350	200
360	209
370	215
380	222
390	225
400	230

Minimum position (above the shoulder joint)	220mm	183.25
Mid Position(parallel to the shoulder joint)	325mm	184.67
Maximum Position(bellow the shoulder joint)	370mm	209

10.3 Appendix C, Code Listing

10.3.1 Interface Module

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
using System.IO.Ports;
using System.Windows.Forms;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace ArmInterface
{

  class Interface
  {
    static public Interface MyPort = new Interface();
    //Struture Variables
    private System.IO.Ports.SerialPort port;          //Serial Port Object
    private string[] AvailPortNames;                  //Availables Ports
    private System.Timers.Timer StatusTimer;          //Status Timer
    private int TFRPS;                                //Temp Number of Failed Reads Per Second
    private int TSRPS;                                //Temp Number of Susscfull Reads Per Second
    private int FRPS;                                 //Number of Failed Reads Per Second
    private int SRPS;                                 //Number of Susscfull Reads Per Second
    private byte[] RecivedData;                       //Storage of current data bytes Recive
    private int[] A2DValue = new int[5];              //Stores the analog 2 Digitial Value(0-255)
    private string[] StatusString =                   //String Used For Displaying Statuss
    { "Healthy", "Poor", "Interface Not Open" };
    static public string[] DevicesAvail =             //string Used For Device options
    { "Anti-GravityArm", "TestBed" };
    private string CurrentDevice;
    private int PotNumTurns;
    private int[] PotCentres = new int[5];





    public Interface()     //Interface constructor
    {
      port = new System.IO.Ports.SerialPort();       //Call port Constructor
      StatusTimer = new System.Timers.Timer();       //Call Timer Constructor
      //Initialize status Timer
      StatusTimer.Interval = 1000;                   //Set Interval to Seconds
      StatusTimer.Enabled = true;                    //Enable Timer
      StatusTimer.Start();

      //Event Initialize
      StatusTimer.Elapsed +=
        new System.Timers.ElapsedEventHandler(StatusTimer_Elapsed);  //Define Status Timer Event
      port.DataReceived +=
        new System.IO.Ports.SerialDataReceivedEventHandler(port_DataRecived);

      LoadPreviousSettings();
    }
    public int GetPotTurns()
    {
        return PotNumTurns;
    }

    public void SetPotTurns(int Turns)
    {
        PotNumTurns = Turns;
    }
    public int[] GetPotCentres()
    {
        return PotCentres;
    }

    public void SetPotCentres(int[] centres)
    {
        PotCentres = centres;
    }

    //Gets the Current Device is Use
    public string GetCurrentDevice()
    {
        return CurrentDevice;

    }
    //Set the Current Device
    public void SetCurrentDevice(string Device)
    {
        CurrentDevice = Device;

    }


    //Force Sync
    public void ForceSync()
    {
        while (!TestSync())
        {
            if (port.IsOpen)
            {
                port.Close();
            }
            else
            {
                try
                {
                    port.Open();
                }
                catch
                { }
            }

        }

    }

    //Test Sync
    public bool TestSync()
    {
        TSRPS = 0;
        int NumberRetry = 0;

        while (NumberRetry < 5)
        {
            System.Threading.Thread.Sleep(100);
            NumberRetry++;
            if (TSRPS > 2)
                return true;
        }
        return false;
    }
```

```
//Return A string repesenting the Status
public string GetStatusString()
{
   if (!IsInterfaceOpen())
   {
      return StatusString[2];
   }
   else if (SRPS <= 10)
   {
      return StatusString[1];
   }
   else
   {
      return StatusString[0];

   }

}




//Return the Number of Susscfull Reads Per Second
public int GetSRPS()
{
   return SRPS;
}

//Return the Number of Failed Reads Per Second
public int GetFRPS()
{
   return FRPS;
}


//Return a List of the A2D Values
public int[] GetListA2DValues()
{
   return A2DValue;
}

//Return a Single A2D Value
public int GetSingleA2DValue(int PotNum)
{
   return A2DValue[PotNum];
}


//Returns the BaudRate
public int GetBaudRate()
{
   return port.BaudRate;
}
//Set BaudRate
public void SetBaudRate(int Rate)
{
   port.BaudRate = Rate;
}

//Returns if the Interface is open
public bool IsOpen()
{
   return port.IsOpen;
}

//Set Comunication Bit Number
public void SetDataBits(int NumDataBits)
{
   port.DataBits = NumDataBits;
```

```
    }
    //Get Number of Comnication Bits
    public int GetDataBits()
    {
      return port.DataBits;
    }
    //Set Number of Bytes to be recived
    public void SetReciveByteThreshold(int RxByteThreshold)
    {
      port.ReceivedBytesThreshold = RxByteThreshold;
    }
    //Get Received Bytes Thresholold
    public int GetReciveByteThreshold()
    {
      return port.ReceivedBytesThreshold;
    }

    //Open Interface
    public void OpenInterface()
    {
      //Try to Open Port
      try
      {
        port.Open();
      }
      catch
      {
        //Show Warning Message if port Cannot be opened
        System.Windows.Forms.MessageBox.Show("Serial Port: " + port.PortName + " failed to open", "Interface Error",
          System.Windows.Forms.MessageBoxButtons.OK, System.Windows.Forms.MessageBoxIcon.Warning);
      }
    }
    //Close Interface
    public void CloseInterface()
    {
      if (port.IsOpen)
      {
        port.Close();
      }

    }
    public bool IsInterfaceOpen()
    {
      return port.IsOpen;
    }

    //Returned the Names of the Serial port in use
    public string[] GetAvailablePorts()
    {
      AvailPortNames = SerialPort.GetPortNames();
      return AvailPortNames;
    }

    //Sets the PortName
    public void SetPortName(string PortName)
    {
      //Test if Port is open and close it
      if (port.IsOpen) port.Close();
      port.PortName = PortName;
    }

    //Return the Name of the port used by the interface
    public string GetInterfacePortName()
    {
      return port.PortName.ToString();
    }

    public void SaveSettings()
    {
      string txtext;                              //Create String to Write Data
      txtext = "PortName: ";                          //Add Varables to string
```

```
        txtext += GetInterfacePortName();
        txtext += "\nBaudRate: ";
        txtext += GetBaudRate();
        txtext += "\nByteThreshold: ";
        txtext += GetReciveByteThreshold();
        txtext += "\nNumberOfBits: ";
        txtext += GetDataBits();
        txtext += "\nCurrentDevice: ";
        txtext += GetCurrentDevice();

        TextWriter tw = new StreamWriter("config.txt");    //Creat Writer and Open File
        tw.Write(txtext);                                  //write the String
        tw.Close();                                        //Close the writer

    }

    public void SaveCalibration()
    {
        string filename;

        string txtext;
        txtext = "DeviceName: ";
        txtext += GetCurrentDevice();
        txtext += "\nPotTurns: ";
        txtext += GetPotTurns();
        txtext += "\nPot1Centre: ";
        txtext += PotCentres[0];
        txtext += "\nPot2Centre: ";
        txtext += PotCentres[1];
        txtext += "\nPot3Centre: ";
        txtext += PotCentres[2];
        txtext += "\nPot4Centre: ";
        txtext += PotCentres[3];
        txtext += "\nPot5Centre: ";
        txtext += PotCentres[4];

        if (CurrentDevice == DevicesAvail[0])
            filename = "ArmCali.txt";
        else filename = "testbedCali.txt";

        TextWriter tw = new StreamWriter(filename);    //Creat Writer and Open File
        tw.Write(txtext);                              //write the String
        tw.Close();                                    //Close the writer

    }

    public void LoadCalibration()
    {
        string FileName;
        if (GetCurrentDevice() == DevicesAvail[1])
            FileName = "testbedCali.txt";
        else
        {
            SetCurrentDevice(DevicesAvail[0]);
            FileName = "ArmCali.txt";
        }
        char[] delimit = new char[] { ' ' };           //Delimter for spliting string
        string rxtext;                                 //Create string For reading data
        string[] subrxtext = { "name", "Data" };       //Delimted String
        string[] DataTextArray = { "name", "turns",
                            "c1","c2"
                            ,"c3","c4","c5"}; //Array of Data
        TextReader tr = new StreamReader(FileName);        //Creatw Reader and Open File
        for (int i = 0; i <= 6; i++)
        {
            rxtext = tr.ReadLine();                    //Read Data to string
            subrxtext = rxtext.Split(delimit);
            DataTextArray[i] = subrxtext[1];
        }
        SetPotTurns(Convert.ToInt32(DataTextArray[1]));
        for (int i = 2; i <= 6; i++)
        {
            PotCentres[i - 2] = Convert.ToInt32(DataTextArray[i]);

        }

    }
    public float GetAngle(int PotNum)
    {
        float DegPerUnit = ((float)PotNumTurns * 360.0f) / 1024.0f;
        return MathHelper.ToRadians((float)((float)GetSingleA2DValue(PotNum) - (float)PotCentres[PotNum]) * DegPerUnit);

    }



    //Restore Interface Setting From previous
    public void LoadPreviousSettings()
    {
        LoadSettings("config.txt");
    }
    //Load default Interface Setting
    public void LoadDefaultSettings()
    {
       LoadSettings("defaultconfig.txt");
    }

    private void LoadSettings(String FileName)
    {
        char[] delimit = new char[] { ' ' };            //Delimter for spliting string
        string rxtext;                                  //Create string For reading data
        string[] subrxtext= {"name", "Data"};           //Delimted String
        string[] DataTextArray = { "comps", "rate",
                               "Threshold","NumberofBits"
                               ,"CurrenDevice"}; //Array of Data
        TextReader tr = new StreamReader(FileName);    //Creat Reader and Open File

        for (int i = 0; i <=4; i++)
        {
            rxtext = tr.ReadLine();                         //Read Data to string
            subrxtext = rxtext.Split(delimit);
            DataTextArray[i] = subrxtext[1];
        }
        //Set Loaded Data
        SetPortName(DataTextArray[0]);
        SetBaudRate(Convert.ToInt32(DataTextArray[1]));
        SetReciveByteThreshold(Convert.ToInt32(DataTextArray[2]));
        SetDataBits(Convert.ToInt32(DataTextArray[3]));
        SetCurrentDevice(DataTextArray[4]);


        //rxtext.Substring(
        tr.Close();                                 //Close text reader

    }

    //Event Functions
    //Status Timer Event
    private void StatusTimer_Elapsed(object sender, EventArgs e)
    {
        //Reset Satus Variables
        SRPS = TSRPS;
        FRPS = TFRPS;
        TFRPS = 0;                              //Int Susscfull Reads Per Second
        TSRPS = 0;                              //Int Failed Reads Per Second

    }
    //Convert Byte Array to Hex String
    private static string BytesToHex(byte[] bytes)
    {
        StringBuilder hexString = new StringBuilder(bytes.Length);
        for (int i = 0; i < bytes.Length; i++)
```

```
            {
                hexString.Append(bytes[i].ToString("X2"));
            }
            return hexString.ToString();
        }

        //Data Recived Interput
        private void port_DataRecived(object sender, EventArgs e)
        {
            int NumBytes = port.BytesToRead;                //Calulate Number of bytes to Read
            RecivedData = new byte[NumBytes];               //Int Data Storage
            if (NumBytes == 8)
            {
                if (port.Read(RecivedData, 0, 8) != 0)
                {
                    int start = Convert.ToInt32(RecivedData[0]);
                    int end = Convert.ToInt32(RecivedData[7]);

                    if (start == 83 && end == 69)
                    {
                        byte[] bytes0 = { RecivedData[2], RecivedData[1] };
                        byte[] bytes1 = { RecivedData[4], RecivedData[3] };
                        A2DValue[0] = int.Parse(BytesToHex(bytes0), System.Globalization.NumberStyles.HexNumber, null);
                        A2DValue[1] = int.Parse(BytesToHex(bytes1), System.Globalization.NumberStyles.HexNumber, null);

                        TSRPS++;
                    }
                    else if (start == 84 && end == 70)
                    {

                        byte[] bytes2 = { RecivedData[2], RecivedData[1] };
                        byte[] bytes3 = { RecivedData[4], RecivedData[3] };
                        byte[] bytes4 = { RecivedData[6], RecivedData[5] };
                        A2DValue[2] = int.Parse(BytesToHex(bytes2), System.Globalization.NumberStyles.HexNumber, null);
                        A2DValue[3] = int.Parse(BytesToHex(bytes3), System.Globalization.NumberStyles.HexNumber, null);
                        A2DValue[4] = int.Parse(BytesToHex(bytes4), System.Globalization.NumberStyles.HexNumber, null);

                    }

                    else
                    {
                        port.DiscardInBuffer();
                        port.Close();
                        port.Open();
                        TFRPS++;
                    }

                }
            }
        }
        }



}
```

10.3.2 Arm Object

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;



namespace _2D_Arm_Movment_Game
{
   class ArmObject
   {
      public ArmSegment ArmSegment1;
      public ArmSegment ArmSegment2;
      public Rectangle HitBox;

      public ArmObject(Texture2D Texture1, Texture2D Texture2)
      {
         ArmSegment1 = new ArmSegment(Texture1);
         ArmSegment2 = new ArmSegment(Texture2);
         ArmSegment1.position = new Vector2(405, Target.viewport.Height);
         ArmSegment2.position = ArmSegment1.EndPoint();
         UpdateHitBox();
      }
      private void UpdateHitBox()
      {
         HitBox = new Rectangle((int)(ArmSegment2.EndPoint().X + ArmSegment2.sprite.Height),
                     (int)(ArmSegment2.EndPoint().Y + ArmSegment2.sprite.Height),
                     ArmSegment2.sprite.Height,
                     ArmSegment2.sprite.Height);

      }
      public void Update()
      {
         KeyboardState keyBoardState = Keyboard.GetState();
         if (keyBoardState.IsKeyDown(Keys.Left))
         {
            ArmSegment2.userRotation -= 0.1f;
         }
         if (keyBoardState.IsKeyDown(Keys.Right))
         {
            ArmSegment2.userRotation -= 0.1f;
         }
         if (keyBoardState.IsKeyDown(Keys.Up))
         {
            ArmSegment1.userRotation -= 0.1f;
         }
         if (keyBoardState.IsKeyDown(Keys.Down))
         {
            ArmSegment1.userRotation += 0.1f;
         }

         ArmSegment1.userRotation = (ArmInterface.Interface.MyPort.GetAngle(1) * -1);
         ArmSegment2.userRotation = (ArmInterface.Interface.MyPort.GetAngle(3));
         ArmSegment1.nonuniformscale.X = (float)Math.Cos(ArmInterface.Interface.MyPort.GetAngle(2));
         ArmSegment2.nonuniformscale.X = (float)Math.Cos(ArmInterface.Interface.MyPort.GetAngle(4));

         ArmSegment1.rotation = ArmSegment1.userRotation + MathHelper.ToRadians(-90.0f);
         ArmSegment2.rotation = ArmSegment2.userRotation + ArmSegment1.rotation;
         ArmSegment2.position = ArmSegment1.EndPoint();
```

```
            UpdateHitBox();
        }

        public void Draw(SpriteBatch spriteBatch, Rectangle viewportrec)
        {
            ArmSegment1.Draw(spriteBatch, viewportrec);
            ArmSegment2.Draw(spriteBatch, viewportrec);

        }

    }

}
```

10.3.3 ArmSegment Object

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace _2D_Arm_Movment_Game
{
    class ArmSegment
    {
        public Texture2D sprite;
        public Vector2 position;
        public float rotation;
        public float userRotation;
        public Vector2 center;
        public Vector2 nonuniformscale;


        public ArmSegment(Texture2D loadedTexture)
        {
            rotation = 0.0f;
            position = Vector2.Zero;
            sprite = loadedTexture;
            center = new Vector2(sprite.Width / 2, sprite.Height / 2);
            rotation = 0.0f;
            userRotation = 0.0f;
            nonuniformscale = new Vector2(1.0f, 1.0f);

        }

        public Vector2 EndPoint()
        {
            return new Vector2(
                (float)(Math.Cos(rotation) * (sprite.Width * nonuniformscale.X / 2)) + position.X,
                (float)(Math.Sin(rotation) * (sprite.Width * nonuniformscale.X) / 2) + position.Y);

        }

        public void Draw(SpriteBatch spriteBatch, Rectangle viewportrec)
        {
            spriteBatch.Draw(sprite,
                position,
                null,
                Color.White,
                rotation, center,
                nonuniformscale,
                SpriteEffects.None,
```

10.3.4 Level Object

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;
using Inter;

namespace _2D_Arm_Movment_Game
{
  class Level
  {
    public static Level[] Levels;
    public int LevelNumber;
    //public static Level* CurrentLevel;
    public int TypeOfBallAllowed;      //This is the type of Balls allowed on screen for the level
                                       //Type: 1 - Green Balls Only
                                       //Type: 2 - Orange Balls Only
                                       //Type: 3 - Red Balls Only
                                       //Type: 4 - Green and Orange Balls Only
                                       //Type: 5 - Green and Red Balls Only
                                       //Type: 6 - Orange and Red Balls Only
                                       //Type: 7 - All Balls
    public Vector2 MaxBallPosHeight;
    public Vector2 MaxBallPosWidth;
    public bool HasTimeLimit;
    public int TimeLimit;
    public int MaxBallsNumberOnScreen;
    public int NumberOfBalls;
    public Target[] balls;
    public float ballScale;
    private Random rand = new Random();
    public Rectangle TargetRegio;
    public List<Target> UnusedTargets = new List<Target>();
    public MenuOverlay InfoMenu;
    public int[] BallScore = new int[3]{1,1,1};




    public Level()
    {
      LevelNumber = new int();


    }
    public void CheckCollision(Rectangle TargetBox, SoundBank soundBank)
    {
      for(int i =0; i < NumberOfBalls;i++)
      {
        if((balls[i].Alive)&&(balls[i].HitBox.Intersects(TargetBox)))
        {

          if (balls[i].Type == 1)
          {
            balls[i].Alive = false;
            soundBank.PlayCue("mario_03");
            ScoreBoard.score += BallScore[balls[i].Type - 1];
          }
          if ((balls[i].Type == 2)&&((Interface.MyPort.GetAngle(2) > 0.4f)|| Interface.MyPort.GetAngle(4) > 0.4f))
          {
            balls[i].Alive = false;
            soundBank.PlayCue("mario_03");
```

```
                ScoreBoard.score += BallScore[balls[i].Type - 1];

              }

              if ((balls[i].Type == 3) && ((Interface.MyPort.GetAngle(2) < 0.4f) || Interface.MyPort.GetAngle(4) < 0.4f))
              {
                balls[i].Alive = false;
                soundBank.PlayCue("mario_03");
                ScoreBoard.score += BallScore[balls[i].Type - 1];

              }




            }
          }

        }



        public int GetRandomType()
        {

          switch(TypeOfBallAllowed)
          {
            //Green Balls Only
            case 1:
              return 1;

            //Orange Balls Only
            case 2:
              return 2;
            //Red Balls Only
            case 3:
              return 3;
            //Green and Orange Balls Only
            case 4:
              //return rand.Next(1,2);
              if (rand.NextDouble() < 0.5)
                return 1;
              else
                return 2;
            //Green and Red Balls Only
            case 5:
              if(rand.NextDouble() < 0.5)
                return 1;
              else
                return 3;
            //Orange and Red Balls Only
            case 6:
              if (rand.NextDouble() < 0.5)
                return 2;
              else
                return 3;
            //All types
            default:
              double d = rand.NextDouble();
              if (d < 0.3)
                return 1;
              else if(d< 0.6)
                return 2;
              else
                return 3;
          }
        }

        public void Initialize()
        {
```

```
        InfoMenu.Active = true;


        for (int i = 0; i < NumberOfBalls; i++)
        {

          int type = GetRandomType();

          balls[i] = new Target(type, ballScale, new Vector2(MaxBallPosWidth.X + (float)(rand.NextDouble() * (MaxBallPosWidth.Y -
MaxBallPosWidth.X)),
            MaxBallPosHeight.X + (float)(rand.NextDouble() * (MaxBallPosHeight.Y - MaxBallPosHeight.X))));


          balls[i].Initialize();
          UnusedTargets.Add(balls[i]);

          for (int j = 0; j <= i; j++)
          {
            if (i != j)
            {
              while (balls[i].HitBox.Intersects(balls[j].HitBox))
              {
                balls[i].Position = new Vector2(MaxBallPosWidth.X + (float)(rand.NextDouble() * (MaxBallPosWidth.Y -
MaxBallPosWidth.X)),
                        MaxBallPosHeight.X + (float)(rand.NextDouble() * (MaxBallPosHeight.Y - MaxBallPosHeight.X)));
                balls[i].UpdateHitBox();
              }
            }
          }
          balls[i].Initialize();

        }


      }
      public void DrawTargets(SpriteBatch spriteBatch, Rectangle viewportrec)
      {
        for (int i = 0; i < NumberOfBalls; i++)
        {
          balls[i].Draw(spriteBatch, viewportrec);
        }
        InfoMenu.Draw(spriteBatch, viewportrec);


      }
      public int Update(SoundBank soundBank)
      {
        ScoreBoard.level = LevelNumber;
        int NumberOfBallAlive = 0; ;
        for (int i = 0; i < NumberOfBalls; i++)
          if(balls[i].Alive)
            NumberOfBallAlive++;

        while ((NumberOfBallAlive < MaxBallsNumberOnScreen) && (UnusedTargets.Count() > 0))
        {
          UnusedTargets.Last().Alive = true;
          UnusedTargets.Remove(UnusedTargets.Last());
          NumberOfBallAlive++;


        }
        InfoMenu.Update();
        if ((UnusedTargets.Count == 0)&&(NumberOfBallAlive ==0))
        {
          System.Threading.Thread.Sleep(100);
          soundBank.PlayCue("mario_14");

          return LevelNumber + 1;
        }
        else
          return LevelNumber;
      }



   }
}
```

10.3.5 Scoreboard Object

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;


namespace _2D_Arm_Movment_Game
{
   class ScoreBoard
   {

      static public int score = 0;                              //Players Score
      static public int level = 1;                              //Players Level
      public SpriteFont ScoreFont;                              //Font Used in scoreboard
      private Vector2 scroreDrawPoint = new Vector2(0.39f, 0.028f);   //Score Location as percentage
      private Vector2 levelDrawPoint = new Vector2(0.666f, 0.028f);


      public ScoreBoard()
      {

      }

      public void Draw(SpriteBatch spriteBatch, Rectangle viewportrec)
      {
         //Draw Score
         spriteBatch.DrawString(ScoreFont,
           score.ToString(),
            new Vector2(scroreDrawPoint.X * viewportrec.Width,
               scroreDrawPoint.Y * viewportrec.Height),
               Color.Black);
         //Draw Level
         spriteBatch.DrawString(ScoreFont,
            level.ToString(),
            new Vector2(levelDrawPoint.X * viewportrec.Width,
               levelDrawPoint.Y * viewportrec.Height),
               Color.Black);



      }


   }
}
```

10.3.6 Menu Object

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;


namespace _2D_Arm_Movment_Game
{
   class MenuOverlay
   {
      public static Texture2D[] TextureArray;
      public int MenuType;
      public Texture2D Texture;
      public Rectangle ButtonRec;
      public bool Active;
      public MouseState mouseStateCurrent, mouseStatePrevious;


      public MenuOverlay(int Type)
      {
         Texture = TextureArray[Type-1];
      }
      public void Draw(SpriteBatch spriteBatch, Rectangle viewportrec)
      {
         if(Active)
         spriteBatch.Draw(Texture, viewportrec, Color.White);

      }

      public void Update()
      {
         mouseStateCurrent = Mouse.GetState();
         if ((mouseStateCurrent.LeftButton == ButtonState.Pressed) &&(mouseStatePrevious.LeftButton == ButtonState.Released))
         {
            //mouseStateCurrent.X
            if ((mouseStateCurrent.X >= 522) && (mouseStateCurrent.X <= 650) &&
               (mouseStateCurrent.Y >= 444) && (mouseStateCurrent.Y <= 497))
               Active = false;

         }

         mouseStatePrevious = mouseStateCurrent;



      }


   }
}
```

10.3.7 Target Object

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace _2D_Arm_Movment_Game
{
  class Target
  {
    public static Texture2D[] TextureArray;
    public static Rectangle viewport;
    public Texture2D Texture;
    public Vector2 Position;
    private float Scale;
    public int Type;
    private Vector2 Centre;
    public Rectangle HitBox;
    public bool Alive;


    public Target(int type, float scale, Vector2 position)
    {
      Type = type;
      Scale = scale;
      Position = position;
      //Alive = true;
    }

    public void Initialize()
    {
      Texture = TextureArray[Type - 1];
      Centre = new Vector2(Texture.Width / 2, Texture.Height / 2);
      HitBox = new Rectangle((int)(Position.X * viewport.Width), (int)(Position.Y * viewport.Height), (int)(Texture.Width * Scale),
(int)(Texture.Height * Scale));
    }

    public void UpdateHitBox()
    {
      HitBox = new Rectangle((int)(Position.X*viewport.Width), (int)(Position.Y*viewport.Height), (int)(Texture.Width * Scale),
(int)(Texture.Height * Scale));
    }

    public void Draw(SpriteBatch spriteBatch, Rectangle viewportrec)
    {
      if (Alive)
      {
        spriteBatch.Draw(Texture,
          new Vector2(Position.X * viewportrec.Width, Position.Y * viewportrec.Height),
          null,
          Color.White,
          0.0f,
          Centre,
          Scale,
          SpriteEffects.None,
          0);
      }
    }




  }
}
```