Building a Physical Analogue for Feedback Rich Systems

Maria Foo
u5021276

Supervised by Dr Chris Browne

13 November 2016

A thesis submitted in partial fulfilment of the degree of Bachelor of Engineering

Department of Engineering
The Australian National University

Maria Foo
13 November 2016

Acknowledgements

I would like to thank my supervisor, Dr. Chris Browne, for providing a rewarding project, and for introducing me to the field of system dynamics education.

I would also like to acknowledge Roger Qiu for his significant contributions to the development of the software, especially in the area of network programming.

I would like to thank Mr. Jeremy Smith, and the Humanitarian Engineering & Educational Development Group for providing a supportive research environment for the past year.

I changed to engineering when I was in my third year of university thanks to my friend Jack Zhang who encouraged me to pursue my interests.

Finally, I would like to thank my mum Carmelita Ly, dad Foo Hau Kiun, brother Sen Foo, aunty Clara Nunes, and uncle Oscar Nunes for their ever-present support and guidance.

Systems dynamics has changed the way I understand problems. I hope that more educational tools may be created to improve our understanding of complex systems over time.

Abstract

Systems in the modern world are often complex and non-linear. The study of how a system behaves over time falls within the domain of system dynamics, which provides an endogenous model to solve problems through feedback loops, time delays, stocks, and flows. The latter – stocks and flows – refers respectively to a quantity accumulated over time and the rate of accumulation, and are the features of the model concerned in this project.

While simple at surface, the mathematical principles behind the operation of systems are not easy for the typical person to grasp. Literature shows that even people with technical backgrounds are unable to distinguish between a stock and flow in everyday examples. However, the benefits of system dynamics extend to policy design, which often cause unexpected side effects due to unforseen feedback behaviour. Therefore, there is a growing need for an accessible educational platform to explain its core concept, and allow participants to understand the interaction between stocks and flows through firsthand experience.

The research and design work embodied in this thesis produced an interactive Arduino gaming platform consisting of a remote control, game server, and game client. The game entails a hot air balloon that changes height (the stock) with respect to input volume of hot air (the flow). To increase the flow and observe its effect on the stock, the user would create a clockwise rotation gesture with their hand using the controller, likened to winding a pulley. The user's hand motion is analogous to flows, and this physical analogue allows the user to create mental models of system behaviour.

Agile development principles were demonstrated through the design of hardware and software. In its course, incremental releases defined by development, test, and deployment cycles resulted in rapid verification of the system's capabilities. Hence, an Agile model best compensated for the unfamiliar problem and solution domain. In particular, the initial and final prototypes proved the controller design feasible, and that design decisions are on the right path. This process was formalised into a decision tree. Future design work and enhancements to the system, and similar interactive educational tools, could well adopt this concept.

Table of Contents

List of Figures

List of Tables

List of Equations

List of Snippets

List of Decision Matrices

Glossary of Terms

Term	*Definition*
Accelerometer	Accelerometers measure linear acceleration (specified in mV/g) along one or several axis. These devices measure the proper acceleration of the rigid body they are attached to.
Analogue Signal	A continuous signal where the quantity that varies with time represents changes in the physical world.
Digital Signal	A signal where the quantity that varies with time takes discrete values.
Analogue-to-Digital Converter	A device that converts a continuous (Analogue) signal into equivalent discrete representation over defined time steps.
Angular Orientation	The direction of motion.
Angular Velocity	A measure of how rapidly the direction of motion is changing with respect to time. SI unit: rad/s.
Autocorrelation	Correlation of a signal sampled over a time interval with respect to the same signal at another time interval, typically used to find repetitions.
Bandwidth	The allowable range of frequencies within a signal band.
Centripetal Acceleration	The acceleration of an object toward the centre of rotation when undergoing circular motion.
Frequency	The number of complete waveforms over a fixed time domain, typically 1 second.
Fundamental Frequency	The lowest frequency of a multi-harmonic waveform.
g-force	Acceleration when compared with the Earth's gravity.
Gyroscope	Gyroscopes measure angular velocity on one or several axis (specified in mV/deg/s).
Inertial Measurement Unit	A device that measures the change in motion – both angular and linear motion.
Input Output Pin	An electrical connection pin on a circuit or chip.
Integrated Circuit	A multi-component electric circuit built on to a single semiconductor unit with connections embedded in the device.
Linear Acceleration	A measure of the rate of change of velocity in linear motion. SI unit: m/s^2.
Linear Motion	Motion along a straight line.
Microcontroller	A control device built with a microprocessor as its base hardware.
Period	
Proper Acceleration	Physical acceleration experienced by an object relative to a free fall.
Revolution	Rotational motion about an axis that is external to the rotating object.
Rotation	Motion about a circular path.
Simple Harmonic Motion	An oscillation motion under the presence of a restoration force. The restoration force is proportional to the displacement.
Spin	Rotational motion around an axis that passes through the centre of mass of the rotating object
Translation	Shift from one point in space to another.
Uniform Circular Motion	The object is in circular motion with constant speed and angular velocity.
Zero Crossing Rate	Rate in which a signal switches from positive to negative (or from negative to positive).

List of Abbreviations

Abbreviation	*Description*	*Abbreviation*	*Description*
3D	Three Dimensional	RPM	Revolutions Per Minute
ADC	Analogue-to-Digital Converter	RPS	Revolutions Per Second
ANU	The Australian National University	ZCR	Zero Crossing Rate
ENU	East North Up		
I/O	Input Output		
IC	Integrated Circuit		
IMU	Inertial Measurement Unit		

List of Symbols

Symbol	*Description*	*Units*
a_c	Centripetal Acceleration	m/s^2
g_n	g-force	g
a	Acceleration	m/s^2
f	Frequency	Hz
f_0	Fundamental Frequency	Hz
I	Current	A
r	Radius	m
S	Sensitivity	mV/g
T	Period	s
v	Velocity	m/s
V	Voltage	V
ω	Angular velocity	rad/s

Chapter 1

Introduction

1.1 Overview

System dynamics is the study of the behaviour of complex systems, based on principles of endogenous change. Since the 1950s, system dynamics has been applied to analyse policies and social systems. The relevance of system dynamics to business decision-making has become increasingly more pronounced, and there is a gap in the current level of understanding of system dynamics among industry professionals. This projects attempts to address challenges in system dynamics education through the development of a physical analogue for flows.

System Dynamics encompasses several branches: feedback loops, non-linearity, stocks and flows. The latter – stocks and flows – is concerned in this study. In more complex systems, there may be multiple sources of flow and multiple stocks, the interaction may non-linear and may not commence immediately i.e. a time delay. This is certainly the case in policies, management, sociology and economics.

There are numerous examples of management failures due to the inability to understand stock and flows, a phenomenon referred to as stock-flow failure. In the Urban Dynamics model, Forrester (1969) provided the case study of a city where low-housing accommodation was provided to address the issue the housing affordability, however the policy resulted in gradual downward spiral in income level and inadvertently reduced affordability. Forrester's model showed that by removing the low-income estates, a counterintuitive policy at first glance, with industries, the average income level was increased due to additional employment opportunities, which in turn improved housing affordability.

Since the 60s, numerous simulation models have been created in attempt to predict the behaviour of complex systems, with varying degrees of success, to influence decision-making. A better approach to improve the outcomes of decision may instead be to establish an intuition among the decision makers, which requires an education platform that enables them to gain a first-hand experience in working with dynamic systems. The hypothesis statement is: *A physical analogue based an interactive education platform improves the understanding of stock and flows.*

A two-fold approach is needed to answer this question, first to design the platform and second proven its ability to education through a social experiment. This research and design project addresses the first, and prototypes the platform with an interactive learning game.

1.2 Scope

The following deliverables are included in the scope:

- Alpha prototype of controller and single-player game for demonstration. The controller interfaces the game and can be used via rotation gesture to create changes in the flow. The game allows the user to visualise the changes to the stock.
- Open source hardware design and software code developed from compatible modules, and using an Agile approach.
- Limited to linear, non-feedback and non-time delay stock and flow scenarios
- A design decision map, similar to a decision tree to inform and document the Agile development process to guide future work.
- A survey in the Australian National University (ANU) assess stock and flow failure among the educated population.

The following are excluded from the scope:

- The software and hardware design will be minimal functional
- The game does not allow for multiple input
- No User Experience (UX) testing will be conducted
- The abovementioned hypothesis statement can only be tested through a social experiment, which is exclude from this project

Chapter 2

Background

This chapter explains the principles of system dynamics, and applies concepts in the field to assess student understanding of stocks and flows at The Australian National University (ANU). System process models are also examined to determine the most appropriate methodology for designing the controller and game.

2.1 Systems Thinking

Systems thinking originates from the field of system dynamics, developed in 1956 by Jay Forrester. Forrester (1989) recognised the need to test the outcomes of social systems, with the same rigour that engineering systems are tested. Hence, system dynamics is fundamentally different from traditional methods of analysis, which focuses on the separation of components belonging to a larger structure (Aronson, 1996). In contrast, system dynamics is founded on the concept of endogenous change, enabled by an organising framework of feedback loops (Sterman, 2002).

2.1.1 Stocks, Flows, and Feedback Loops

There are two types of feedback loops that dictate the behaviour of a system (Meadows, 2009). First, balancing feedback loops are goal-seeking structures in systems that cause stability and resistance to change. Second, reinforcing feedback loops are self-enhancing, leading to exponential growth or decay. However, feedback loops alone are not sufficient to model the dynamic behaviour of a system. Stocks (levels) and flows (rates) demonstrate the effects of feedback loops, which may include time delays, and nonlinearities.

Stocks are state variables that accumulate or deplete over time, whereas flows are the operating elements of a system, which can either increase or decrease a stock. Figure 1 represents a stock and flow diagram for the level of water in a dam. The system has been simplified so that there is only inflow due to rainfall, and outflow due to evaporation.

In Figure 2, when the rate of rainfall exceeds the rate of evaporation, the dam storage level increases over time until it is full. However, when the rate of rainfall is less than the rate of evaporation, the dam storage level decreases over time until it is empty. Dynamic equilibrium occurs when the inflow equals the outflow. While it has not been shown in the diagram, stock behaviour is affected by feedback loops within the system. In this case, feedback loops ensure that flows are constant, as opposed to exponential or logarithmic.

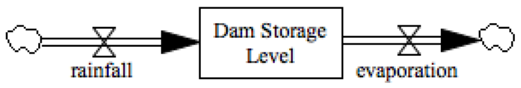


Figure 1. Dam storage level for stock flow diagram

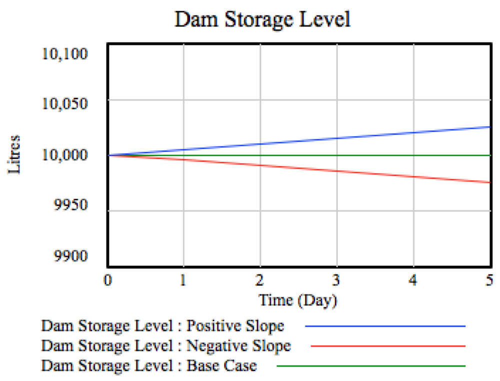


Figure 2. Simulation of dam storage level over 5 days

Systems with similar feedback structures can be categorised into archetypes. In this manner, system dynamics can be used as a common thread to understand concepts across disciplines (Forrester, 2007). Braun (2002) documents ten archetypes that may be applied to gain insight into a system. A well-known example is Limits to Growth by Meadows (1972), which introduces the concept of diminishing returns that emerges when the limit of a system is reached.

While system dynamics has been successfully applied in business to understand policy design, it has drawn many criticisms due to its inability to mimic reality, reliance on quantitative data, deterministic attributes, and consideration of hierarchy (Featherston and Doolan, 2012). These criticisms arise from two central problems: firstly, system dynamics is often applied incorrectly, and secondly, people are misinformed of its endogenous goals, outcomes, and limitations (Featherston and Doolan, 2012; Richardson, 2011). Forrester (2007) suggests that these factors have hindered the adoption of system dynamics from the public, resulting in the continuation of flaws in the way policies are established.

2.1.2 Stock-Flow Failure

Understanding stock flow systems is crucial for many areas of life, ranging from the accumulation of money in a bank account, to the accumulation of greenhouse gases in the atmosphere (Sterman, 2002). The solution to system dynamics adoption lies in education, based on the following ideas by Forrester (2007): 1) educating system dynamics experts, 2) creating a new form of management education around generic structures, and 3) establishing system dynamics as a foundation for concepts taught in primary and secondary schools.

The current difficulty in system dynamics education arises from stock-flow failure, which describes the fundamental misunderstanding of accumulation witnessed in even highly educated people with technical backgrounds (Cronin et al., 2009; Brunstein et al., 2010). Stock-flow failure comes from incorrect application of the correlation heuristic, a term to denote the assumption that a stock is positively correlated to its net flow rate (Cronin et al., 2009).

Cronin et al. (2009) surveyed 173 participants who were presented with a graph showing the number of people entering and leaving a department store over time. This system can be modelled by a single stock and two flows. The study tested the participants’ ability to correctly distinguish between inflows and outflows, and infer the behaviour of stocks from flows. While a majority of participants correctly identified inflows and outflows, less than half were able to answer stock flow questions correctly. Additionally, it was found that participants increasingly used the correlation heuristic to model change in stocks as the flows became more complex. These results cannot be attributed to an inability to interpret graphs, contextual knowledge, motivation, or cognitive capacity.

In a simulation environment, similar results have been observed (Dutt and Gonzalez, 2007; Gonzalez and Dutt, 2011). For example, in a study by Dutt and Gonzalez (2007), participants were required to keep the accumulation of a simulated stock within an acceptable range by controlling inflow and outflow rates. The simulation also had the condition of varying environmental inflow and constant environmental outflow. The study found that participants were quickly able to control a system with increasing environmental inflow. However, under the same net flow, they took significantly more time to control decreasing or non-linear environmental inflow. This study supports the observation that people instinctively use the correlation heuristic, but experience difficulty with negative correlation or non-linearity in stock flow problems.

A recent study by Fischer et al. (2015) suggests that previous research may have potentially used error-inducing formats. In simulations, the source of error may be due to lack of procedural knowledge when interfacing with the system (Mislevy, 2013). In paper-based tasks, errors may arise from the use of scientific notation including coordinate systems, graphs, and percentage values (Hoeffner and Shah, 2002). Therefore, participants may have basic understanding of stock flow systems, which have been concealed due to presentation format (Fischer et al., 2015). Fisher et al. (2015) found that participants arrive at the correct solution when stock flow problems are presented in text. The study suggests that both simulation-based learning and communication of stock flow problems could be improved with more emphasis on verbal information.

2.2 ANU Survey

This survey tests participants’ ability to distinguish between stocks and flows given two types of definitions: a purely verbal explanation, and a mathematical explanation involving the concept of differentiation and integration. The following null hypotheses have been tested:

1. $H_{0,1}$: Performance is the same for both verbal and mathematical explanations
2. $H_{0,2}$: Performance is the same for stock and flow questions.

2.2.1 Participants

Forty-four graduate and undergraduate students from ENGN3410 Engineering Sustainable Systems at ANU participated in the survey. Approximately 77% of students had not studied systems dynamics, and 63% had completed pre-reading. For a vast majority of students this was an elective course, and most had previously completed university-level calculus.

2.2.2 Procedure

Stock and flow equivalents were asked in a verbal and pictorial format. To reduce chances of copying, 10 survey variations were made. Each survey contained approximately six flows and six stocks from a pool of 24 possible questions. Additionally, either a mathematical or verbal explanation was assigned to each survey. For the 44 participants, a total of 24 surveys with mathematical explanations, and 20 surveys with verbal explanations were distributed. A survey for each explanation type is available in Appendix 2A – Survey Questions.

Participants were asked to identify if the verbal and pictorial examples are more likely to be a stock or flow by ticking a checkbox. Participants were then asked to complete six demographic questions. The questions have been summarised in Table 14 and Table 15 of Appendix 2A – Survey Questions.

2.2.3 Results

Each survey has been assigned a pass or fail grade; a pass requires at least 50% of the questions to be answered correctly. Since the analysis involves categorical data, Fisher's exact test was employed to evaluate the association between mathematical and verbal explanations. The calculated p-value is 0.48, as shown in Table 1. Therefore, the null hypothesis is not rejected at a significance level of 0.05. This implies there is no difference between mathematical and verbal explanations on the ability to solve stock flow problems.

Table 1. Contingency table for Fisher's exact test on $H_{0,1}$.

	Mathematical (M)	*Verbal (V)*	Total	Fisher's Exact Test	
Passed	18	17	35	Two-sided p-value	0.4771
Failed	6	3	9		
Total	24	20	44		

Errors in the results may be attributed to participants not reading the mathematical and verbal explanations. In the future, another survey that includes incentives for performance should be conducted on a larger sample size. Alternatively, mathematical or verbal explanations of stock flow questions can be incorporated into the primer, to determine if participants have read the explanations. These methods should reduce error, and provide more reliable results.

Given there is no difference between mathematical and verbal explanations, the number of correct responses for stock and flow questions have been analysed. The second null hypothesis is tested in Table 2 using Fisher's exact test.

Table 2. Contingency table for Fisher's exact test on $H_{0,2}$.

All Questions	*Stocks*	*Flows*	Total	Fisher's Exact Test	
Correct	207	150	357	Two-sided p-value	0.0000
Incorrect	58	113	171		
Total	265	263	528		

Table 2 shows that the p-value is 0, supporting the stock-flow failure phenomenon. The number of stocks that have been answered correctly is 207, whereas the number of flows that have been answered correctly is 150, as shown in Figure 3. Hence, stock bias can be observed in the results. This is expected since stocks are more tangible than flows. That is, there are more examples of stocks in nature, such as trees in a forest, than its related flow, growth.

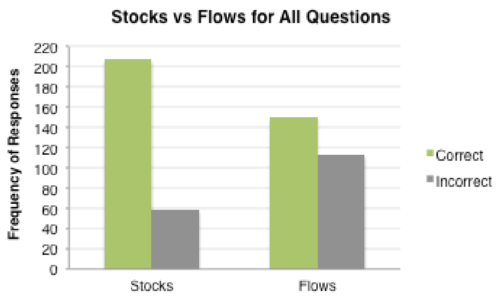


Figure 3. Frequency of correct and incorrect responses for all stock and flow questions.

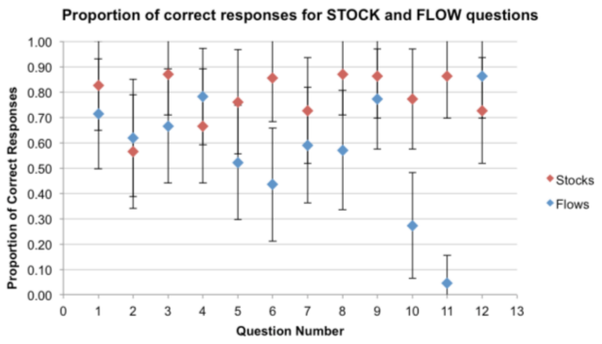


Figure 4. Proportion of correct responses for stock and flow questions for comparison.

The proportion of correct responses for each question has been graphed in Figure 4. Error bars have been included to show an error margin of 0.05. It can be observed that the error bars of questions 6, 10, and 11 do not overlap, indicating a possible two-sided p-value of less than 0.05. Fisher's exact test has been conducted on each question to confirm this result. It was found that questions 8, 10, 11, and 12 produced the most confusion, resulting in a p-value of less than 0.05. The contingency tables are shown in Table 16 to Table 19 of Appendix 2B – Significant Outcomes from Fisher's Exact Test.

Sources of error may arise from misleading pictures, since the same pictorial representation has been used for both the stock and flow variations of each question. However, overall outcomes from this survey aligns with previous studies, which demonstrates the presence of stock-flow failure even in basic tasks regardless of presentation format (Cronin et al., 2009; Dutt and Gonzalez, 2007).

2.3 Physical Analogues

Results from the survey show that stock-flow failure occurs regardless of verbal and mathematical definitions of both concepts. In particular, people seem to demonstrate stock bias to examples in everyday life. In this project, an interactive activity has been developed as an educational tool to promote understanding of flows in system dynamics. The activity builds on the idea of physical analogues to explore the behaviour of flows (Browne, 2013). Physical analogues help to create productive internal mental models for understanding of feedback rich systems (Newell, 2012; Browne, 2013).

Newell (2012) suggests that the human conceptual system works on metaphors that relate understanding of a concept in terms of another. This is demonstrated in children’s toys such as Cuisenaire rods, shown in Figure 5. Cuisenaire rods are plastic rods that come in ten lengths, each representing a number. Manipulation of Cuisenaire rods is analogous to arithmetic operations. Newell (2012) gives an example of multiplication, where three red and two green rods grouped above the box in Figure 5 correspond to 3 × 2 = 2 × 3.

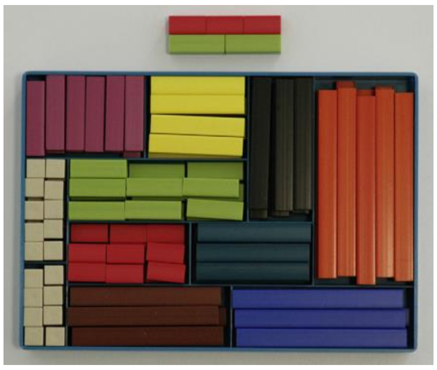


Figure 5. Cuisenaire rods (Newell, 2012).

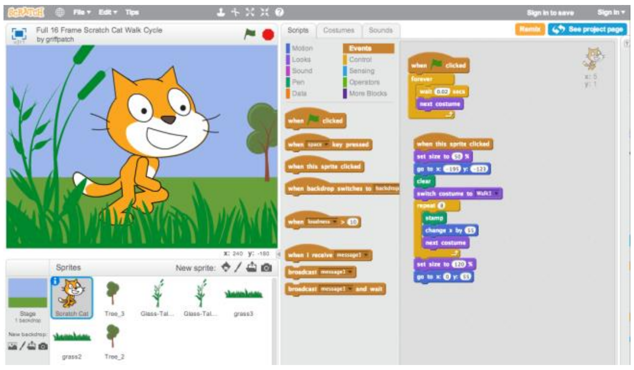
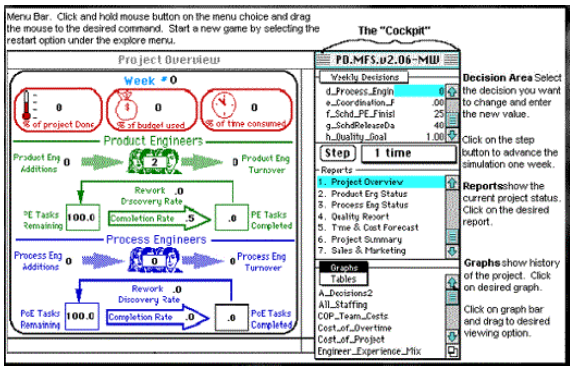


Figure 6. Scratch programming interface (MIT Media Lab, 2013)

Another application that uses analogues for education is Scratch from MIT. Scratch (Figure 6) is an interactive platform for young people to create computer programs. The basic concepts of programming such as variables, control structures, data structures, and syntax are mapped to colourful graphical blocks. Snapping together blocks enable programs such as animated stories, virtual tours, and science simulations to be created. Scratch’s success is represented by its online community of approximately 1.5 million people (MIT Media Lab, 2013). Similar analogical principles are applied in toys such as Lego to create an interactive platform for otherwise abstract tasks.

Physical analogues create interactive decision-making environments that are more accessible than current system dynamic programs such as management flight simulators. In management flight simulators, users manage a dynamic system via graphs, as shown in Figure 7. Decisions are made by knobs, dials, and input boxes on a screen to achieve an outcome.


Menu Bar. Click and hold mouse button on the menu choice and drag the mouse to the desired command. Start a new game by selecting the restart option under the explore menu.

Decision Area Select the decision you want to change and enter the new value.

Click on the step button to advance the simulation one week.

Reports show the current project status. Click on the desired report.

Graphs show history of the project. Click on desired graph.

Click on graph bar and drag to desired viewing option.

Figure 7. Interface of a management flight simulator (Sterman, 2006).

While there are debates about the validity of management flight simulators, they are viewed as one of the best methods available for understanding the complexity of large systems (Papageorgiou and Abrosimova, 2007). However, they exhibit poor user interface design according to Nielsen and Molich's (1990) nine usability heuristics, summarised in Table 3. The nine usability heuristics are used in human factors engineering to explain problems that can be observed in user interface design. In terms of management flight simulators, usability problems may arise due to presentation of dense information (criteria 3).

Table 3. Nine usability heuristics (Nielsen and Molich, 1990)

1	Simple and natural dialogue	6	Provide clearly marked exits
2	Speak the user's language	7	Provide shortcuts
3	Minimise user memory load	8	Good error messages
4	Be consistent	9	Prevent errors
5	Provide feedback		

The need for a usable and accessible system dynamics learning tool is the motivation for this project, which uses the concept of physical analogues to better map stocks and flows to the user's environment. The tool is intended for young people to make decisions about the flows affecting a stock in real-time, observe the result of the stock, then make flow decisions for the next time instant. Interactive environments have been recommended by Cronin et al. (2009) and Forrester (2007) to be beneficial in examining stock-flow failure in decision-making, and to improve learning capacity at an early age.

2.4 System Process Models

In systems engineering, system process models are methodologies that can be tailored to manage project lifecycles (Blanchard and Fabrycky, 1998). Four system process models have been considered in this project: the waterfall, spiral, 'vee', and agile development models. The chosen model must be appropriate for the following project characteristics: vague requirements, unfamiliar problem and solution domains, and a small project size.

2.4.1 Waterfall Process Model

The waterfall model is a sequential design process where each activity must be completed before moving on to the next activity (Ruparelia, 2010). While iteration between immediately connected phases is common, there is limited opportunity to change requirements once they have been defined. This is exemplified by feedback loops in Figure 8, which occur in adjacent phases, and at the design and validation phases only.

The waterfall model is appropriate when requirements are clearly understood, and the problem and solution domains are familiar (Waldron, 2015). While it is easy to implement, one aspect of the model is its requirements for thorough documentation, which includes the development of a requirements document, preliminary design specification, interface design specification, final design specification, test plan, and operations manual (Ruparelia, 2010). Given an unfamiliar problem and solution domain, it is not possible to clearly define future steps in this project. Thus, multiple iterations between non-adjacent phases may be necessary, which is not captured in this methodology.

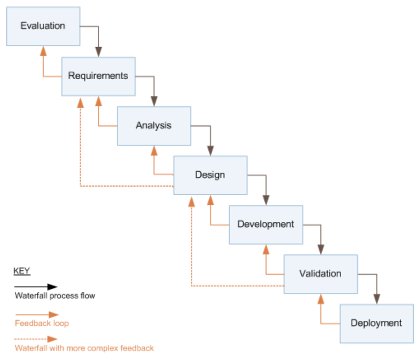


Figure 8. Waterfall model (Ruparelia, 2010).

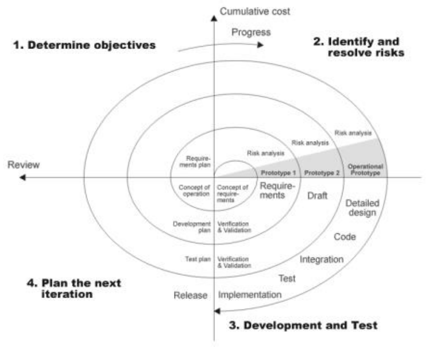


Figure 9. Spiral process model (Ruparelia, 2010).

2.4.2 Spiral Process Model

The spiral process model is a risk-driven approach for systems (Blanchard and Fabrycky, 1998). This model places an emphasis on iterative prototyping that follows four distinct phases, as shown in Figure 9 (Ruparelia, 2010). The development of each prototype starts with a requirements definition phase, and ends with an evaluation and risk analysis phase. The evaluation phase allows customers to evaluate the prototype and give feedback to developers. Risk is monitored before the development of each prototype to control schedule slippage and cost overrun.

The spiral model is typically used in projects with complex requirements that require evaluation for clarity (ISTQB, 2016a). The iterative nature of this methodology allows for the creation of many prototypes. While this satisfies the project characteristics in terms of vague requirements, and unfamiliar problem and solution domains, the spiral model does not add value for small projects that have a low risk profile.

2.3.3 “Vee” Process Model

The “Vee” process model (Figure 10) is an adaptation of the Waterfall model (Ruparelia, 2010). The model commences with user requirements, and gradually decomposes these requirements to the subsystems and components level. The bottom of the ‘V’ represents the development phase. Verification and validation is conducted in parallel to the design process to create a requirements-driven solution (Balaji and Sundararajan Murugaiyan, 2012). Parallel testing saves time and avoids the downward flow of defects (ISTQB, 2016b). While the model works well for small projects where requirements are easily understood, no prototypes are created until the development phase. Therefore, it is a very rigid model that will not work well in unfamiliar problem and solution domains.

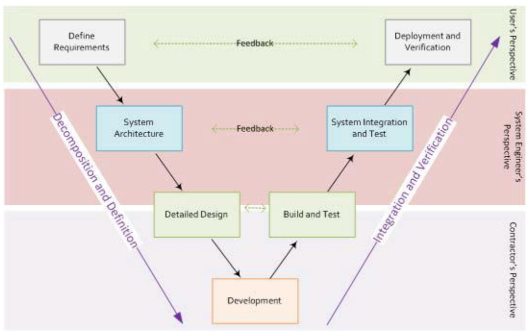

Define Requirements
Feedback
Deployment and Verification
User's Perspective
System Architecture
Feedback
System Integration and Test
System Engineer's Perspective
Detailed Design
Build and Test
Development
Contractor's Perspective
Decomposition and Definition
Integration and Verification

Figure 10. Vee model (Ruparelia, 2010).

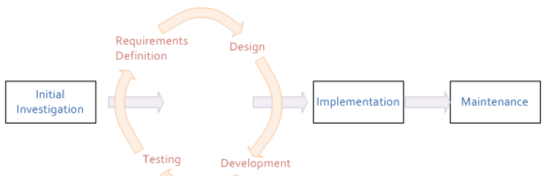


Figure 11. Agile development model (Ruparelia, 2010).

2.3.4 Agile Development Model

Technology and consulting companies are moving toward agile techniques for faster delivery of products. For example, by adopting agile models in 82% of projects, Cisco has realised a fivefold increase in the number of delivered solutions yearly due to rapid feedback processes (Deloitte University Press, 2016). This model relies on rapid prototyping for iterative development, as demonstrated in Figure 11 (Ruparelia, 2010). Development occurs in short intervals and frequent prototypes are created to capture small incremental changes. Agile development methods produce little documentation during development, requiring substantial post project documentation. In this manner, it avoids the formal process-driven steps observed in the previous models.

The advantage of agile development is rapid response to changing requirements (Balaji and Sundararajan Murugaiyan, 2012). This approach helps projects respond to unpredictability through incremental, iterative work that is responsive to feedback. Hence, agile development principles of frequent feedback from short iterations will be used to better predict next steps for the project.

Building a Physical Analogue

This chapter details key requirements for the system, and demonstrates the iterative development process followed to obtain the final product by way of a decision tree.

3.1 Key Requirements

There are six key requirements for the system to be developed in this project:

1. Detect rotation.
2. Detect anticlockwise and clockwise rotation directions.
3. Demonstrate the behaviour of system archetypes.
4. Incorporate real-time feedback relay.
5. Incorporate wireless communication.
6. Held by user.

The requirements are achieved through the creation of a game controller and server that detects rotation speed, measured in revolutions per second (RPS), and anticlockwise and clockwise rotation direction. A system dynamics game has been developed to demonstrate the operation of the controller in real time. The controller acts as a physical analogue to flows, where inflow is mapped as clockwise rotation, and outflow as anticlockwise rotation. The game utilises the RPS and rotation direction information to directly manipulate the behaviour of a stock. Metaphorical mapping of this system is shown in Figure 12 and Table 4. Gravity acts as a constant environmental outflow to create a feedback loop driven by height discrepancies in the system that is observable by the user.

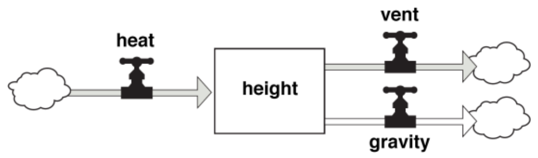


Figure 12. Stock and flow model.

Table 4. Metaphorical mapping of the stock flow model.

Hot Air Balloon	Game and Controller
Height	Game sprite
Gravity	Constant
Heat addition	Clockwise rotation
Heat release	Anticlockwise rotation

3.2 Decision Tree

A decision tree has been has been presented in Figure 12b as a framework for navigation in this project. Black boxes show configuration items, which are distinct system components. For each of these, a decision has been made to decide which hardware component or software feature would be utilised. Black lines relate two configuration item decisions made in consequent of each other. For example, in R5 Connection, by choosing Bluetooth for wireless communication, the choices of HC-06, HC-05, and TinySine 4.0 BLE were made available.

The vertical axis of the diagram relates to the complexity of the system. At the top level, the baseline configuration is chosen. Progressing down the diagram, more items are added to the decisions, which reflect an increase in system dependency, as well as capability. For example, after selecting an Arduino microcontroller as the base hardware, the sensors and platform is decided upon. The bottom level of the decision tree relates to more detailed items, such as data analysis methods.

Red and blue arrows in the tree indicate ‘footprints’ of decisions. Red arrows represent a trail of decisions that have culminated in the final prototype, whereas blue arrows represent decisions that have been tested, but discarded for better alternatives. Grey lines are used to indicate the influence of one decision on another that introduces constraints to the system. For example, the rotation detection algorithm requires the implementation of sampling methods so that acceleration readings can be analysed.

The initial and final prototypes have been represented in a triangle, and octagon respectively. Dashed lines surrounding the shapes specify the components and features included in the prototype. It is important to note that development and testing have been conducted in parallel, without the creation of design artefacts. This is due to an uncertain problem and solution domain, which required frequent testing to define the next path forward. The direction of the project was confined by the start and end goals of creating a controller and demonstrating its operational characteristics in a game.

Decision matrices have been incorporated in this thesis to explain the opportunities and constraints considered in each decision, and identify key requirements that have been fulfilled. In this manner, critical decisions that have been made during development can be identified for future prototyping decisions.

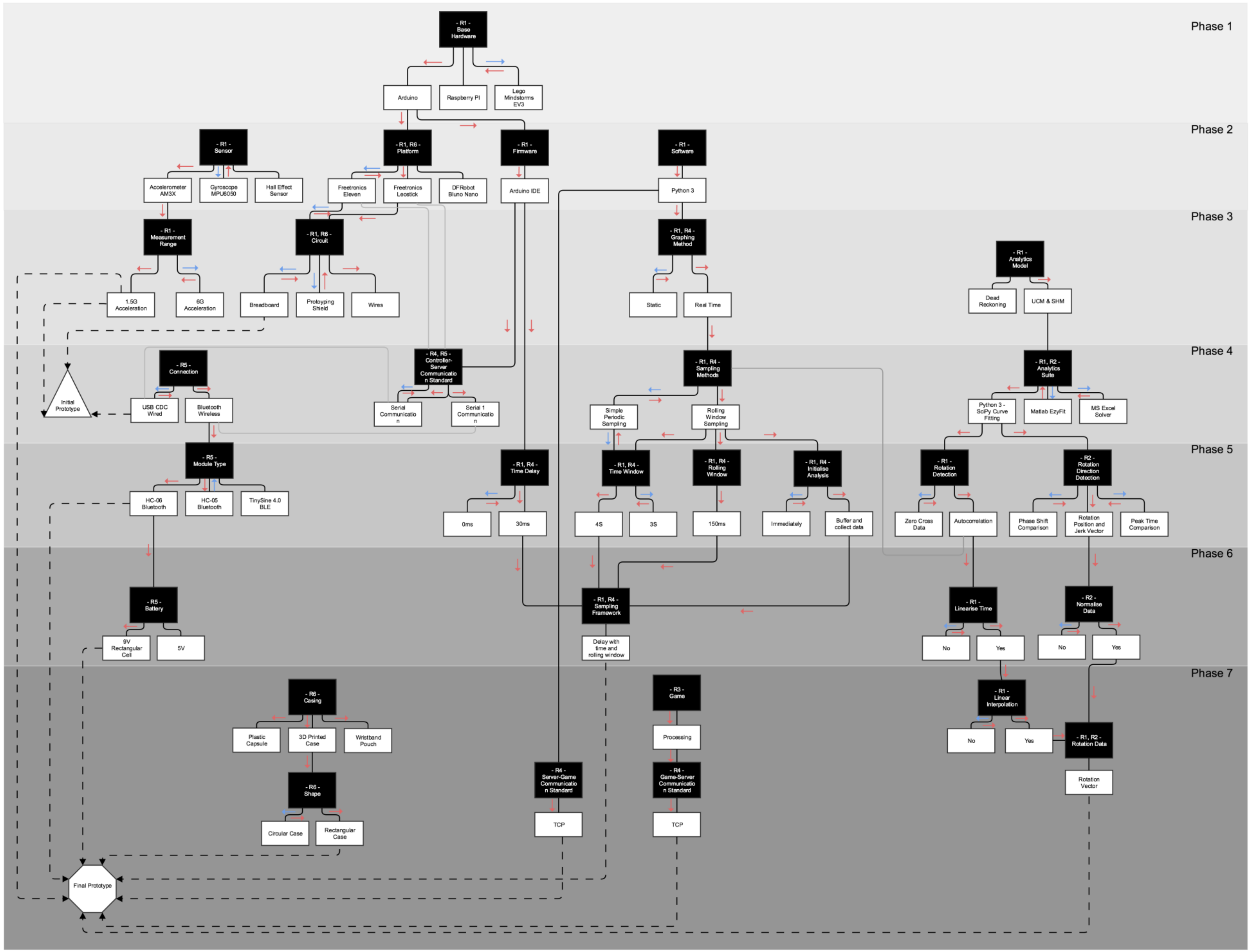


Figure 12b: Decision tree

Chapter 4

Hardware Development

This chapter will discuss the hardware used to create the controller, and the hardware miniaturisation process. The microcontroller and accelerometer that has been used in this project is created by Freetronics in Victoria, Australia.

4.1 Arduino Board

Arduino is an open source platform for prototyping using electronics. Its core is a microcontroller, which is a small computer on an integrated circuit (IC) (Mellis et al., 2007). Sensors can be connected to the microcontroller to create interactive devices. Arduino boards are programmable through the Arduino IDE, which is used to manage, edit, compile, and upload Arduino programs, also known as sketches.

In this project, the Freetronics Eleven and Freetronics LeoStick have been used to create the controller. The Freetronics Eleven is based on the Arduino UNO, whereas the Freetronics LeoStick is based on the Arduino Leonardo (Freetronics, 2016c; Freetronics, 2016d). An additional board that has been considered is the DFRobot Bluno Nano with integrated Bluetooth. While it has not been used due to limited local availability, the DFRobot Bluno Nano's compact size makes it a suitable option for future prototypes.

Rasberry Pi and Lego Mindstorms EV3 have additionally been considered as alternative prototyping platforms to the Arduino board. A Rasberry Pi is a general-purpose computer that runs multiple programs. However, it has not been used in this project since the controller will only be required to perform repetitive tasks, requiring minimal processing capacity. The Lego Mindstorms EV3 robotics kit has been used for its simple GUI programming software, but attempts to model rotation gestures using EV3s were not successful due to limitations placed by its software and components.

Decision Matrix 1. Base Hardware.

Decision	Opportunities	Constraints	Req.
Arduino	Open source.	-	1
Rasberry Pi	General-purpose computer.	-	
Lego Mindstorms EV3	GUI programming software.	Lego software, components.	

Decision Matrix 2. Firmware.

Decision	Opportunities	Constraints	Req.
Arduino IDE	Open source.	-	1

Decision Matrix 3. Platform.

Decision	Opportunities	Constraints	Req.
Freetronics Eleven	Arduino UNO compatible.	Size.	1, 6
Freetronics LeoStick	Arduino Leonardo compatible, size.	-	
DFRobot Bluno Nano	Arduino Nano compatible, integrated Bluetooth, size.	Limited local availability.	

4.2 Sensor

For inertial sensors, a gyroscope and an accelerometer have been considered as two alternatives to measure RPS. 3-axis accelerometers measure proper acceleration as it translates in the X, Y, and Z-axis (Dadafshar, 2015). In contrast, 3-axis gyroscopes measure angular velocity as it rotates about the X, Y, and Z-axis. Consequently, gyroscopes will not be able to detect translation in 3D space (Dadafshar, 2015).

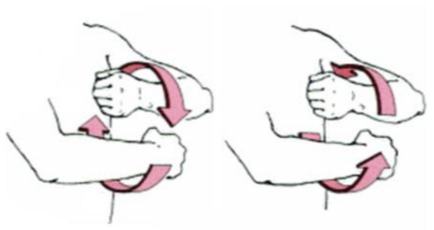


Figure 13. Clockwise and anticlockwise rotation gestures (Mumenthaler and Mattle, 2006).

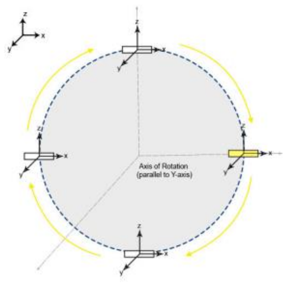


Figure 14. Clockwise revolution showing fixed angular orientation of the controller.

A key requirement for the controller is to recognise clockwise and anticlockwise revolution as demonstrated in Figure 13. Users are expected to hold the device in one hand while moving their forearm in circles. Clockwise revolution will resemble Figure 14 in the 3D coordinate system, where the axis of revolution is parallel to the Y-axis. Therefore, the motion is a translation on the X-Z plane, which can be completely captured by a 3-axis accelerometer. The data can then be analysed to return RPS, and the rotation direction. In this application a gyroscope will not be able to detect the gesture because the device will not be spinning.

A Hall effect sensor has also been considered to detect rotation by the presence of magnets. Applications of Hall effect sensors include the measurement of speed on wheels and shafts, such as motors (Jezny and Curilla, 2013). However, the sensor is based on proximity sensing rather than inertial motion sensing. Inertial motion sensing is the preferred option, since it is more commonly applied in gesture recognition applications (Analog Devices, 2009).

Decision Matrix 4. Sensor.

Decision	Opportunities	Constraints	Req.
Accelerometer AM3X	Detects revolution.	Does not detect angular velocity.	1
Gyroscope MPU6050	Detects spin.	Does not detect translation.	
Hall Effect Sensor	Detects revolution and spin.	Not an inertial sensor.	

4.2.1 AM3X Accelerometer Module

The accelerometer module that has been used in this project is the AM3X from Freetronics as shown in Figure 15. The AM3X is a 3-axis accelerometer with independent analogue X, Y, and Z-axis outputs (Freetronics, 2016b). It carries a MMA7361LC chip from Freescale, which is the accelerometer IC (Freetronics, 2016b). Acceleration is displayed in g-force (g) units, where 1g is approximately 9.8m/s^2. Typical applications of this accelerometer are motion sensing in 3D gaming, and laptop PC free fall detection (Freescale Semiconductor, 2010). The following subsections describe key operating characteristics of the accelerometer, summarised in Table 5, which will influence the implementation of the program.

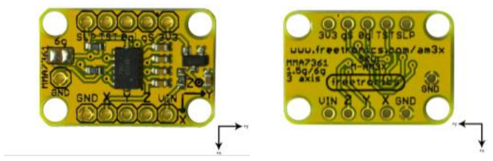


Figure 15. AM3X Accelerometer Module face up (left) and face down (right) (Freetronics, 2016a).

Table 5. Key operating characteristics of the MMA7361LC (Freescale Semiconductor, 2010).

Characteristic	Symbol	Value	Unit
Bandwidth	f_{-3dBXY}	400 for X and Y	Hz
	f_{-3dBZ}	300 for Z	
Range	g_{FS}	$\pm$1.5 or $\pm$6	g
Sensitivity	$S_{1.5g}$	800 at 1.5 g	mV/g
	S_{6g}	206 at 6 g	
Supply Current	I_{DD}	400 to 600	μA
Supply Voltage	V_{DD}	2.2 to 3.6	V
Zero-g Voltage	-	1.65	V

4.2.1.1 Power Supply

The accelerometer module runs at 3.3V, and has an on board 3.3V regulator to enable connection to a microcontroller from either the 5V or 3.3V pins. The accelerometer IC consumes approximately 400 to 600μA, which is under the maximum current of 200mA for all I/O pins on the microcontroller (Freetronics, 2016c; Freetronics, 2016d).

4.2.1.2 Analogue Outputs

The AM3X is an analogue style accelerometer that outputs a voltage directly proportional to the acceleration being experienced. Since each axis can display both positive and negative values, the outputs are offset to half the operational voltage of the accelerometer chip. The chip runs at 3.3V, so any axis that is experiencing 0g acceleration should read 1.65V (Freetronics, 2016b). This is known as the zero-g voltage in Table 5. Figure 53 in Appendix 4A – Summary of IC Datasheet (MMA7361LC) shows the voltage outputs for each axis when the accelerometer is static at various positions.

4.2.1.3 Sensitivity

Sensitivity for the AM3X is 800mV/g at 1.5g and 206mV/g at 6g. The device converts acceleration readings to voltage output using the following relationship.

Equation 1. AM3X acceleration readings to voltage output conversion (Freetronics, 2016b)

$$voltage\ at\ 0g\ (V) + \left(g\ force(g)\ \times\ sensitivity\ \left(\frac{mV}{g}\right)\right) = voltage\ output\ (V)$$

This means that in 1.5g mode, an axis experiencing +1g acceleration will show $1.65V + (1 \times 0.8V) = 2.45V$. Similarly, when experiencing -1g acceleration, it will show $1.65V + (-1 \times 0.8V) = 0.85V$.

In the microcontroller, the 10-bit analogue-to-digital converter (ADC) measures voltages between 0V and 5V on a scale of 0 to 1023 (Freetronics, 2016b). The microcontroller converts analogue to digital signals using the following relationship.

Equation 2. Microcontroller analogue to digital conversion (Freetronics, 2016b)

$$\frac{1023}{5V} \times\ voltage\ output\ (V) = digital\ output$$

Reading an axis at 0g will return a value of approximately $(1023/5V)\ \times 1.65V = 338$, whereas an axis at 1g will return a value of approximately $(1023/5V)\ \times\ 2.45V = 501$. In general, higher sensitivity means that for a given change in acceleration, there will be a larger change in signal (SparkFun Electronics, 2016b). The benefit is more accurate readings, although this presents a trade-off with range (Dimension Engineering LLC, 2016).

4.2.1.4 Range

Range is the maximum and minimum acceleration amplitude the accelerometer can accurately measure before distorting the output signal (Dadafshar, 2015). The AM3X can be configured to ±1.5g or ±6g ranges. This project will use ±1.5g to improve the sensitivity of data for the curve-fitting algorithm used in the Python application. A range of ±1.5g corresponds to a maximum and minimum acceleration of $1.5 \times 9.8 = 14.7\,m/s^2$ and $-1.5 \times 9.8 = -14.7\,m/s^2$ respectively.

Experimental data shows that the recorded maximum and minimum acceleration exceeds the $\pm$1.5g range. This is because the accelerometer runs at 3.3V, and 1.5g represents only $1.65V + (1.5 \times 0.8V) = 2.85V$. The following calculation shows that the full range for a voltage output of 3.3V is $\pm$2g, which is approximately $\pm$20m/s^2. While the accelerometer can output values up to $\pm$2g, any value beyond $\pm$1.5g will not be accurate. For example, since the accelerometer has maxed its voltage output, a reading of 2g may actually be 3g.

Equation 3. Voltage output to acceleration conversion

$$g\,force(g) = \frac{voltage\ output\ (V) - voltage\ at\ 0g\ (V)}{sensitivity\ \left(\frac{mV}{g}\right)} = \frac{3.3V - 1.65V}{0.8\ \left(\frac{V}{g}\right)} = 2g$$

There are various accelerometers available in the market with ranges from $\pm$1g to $\pm$250g (SparkFun Electronics, 2016a). While the highest acceleration amplitude for hand motion is about $\pm$5g, most low g inertial sensing applications are less than $\pm$2g (Graham, 2000; Ang et al., 2004). These applications include robotic navigation, motion tracking of handheld devices for microsurgery, and entertainment (Ang et al., 2004). Therefore, a range of $\pm$1.5g is assumed to be suitable for the performance of this device.

Decision Matrix 5. Measurement Range.

Decision	Opportunities	Constraints	Req.
1.5G	Higher sensitivity.	Lower measurement range.	1
6G	Higher measurement range.	Lower sensitivity.	

4.2.1.5 Bandwidth

Bandwidth indicates how many times a reliable reading can be taken by the accelerometer in one second (Dadafshar, 2015). Humans cannot create body motion beyond the range of 10Hz to 12Hz (Dadafshar, 2015; Mitcheson et al., 2008). For this reason, the AM3X bandwidth of 300Hz for Z-axis outputs, and 400Hz for X and Y-axis outputs are adequate for sensing human motion.

4.2.1.6 Pin Connections

The AM3X has 11 pins, although only five will be used in this project: the X, Y, Z, VIN, and GRD pins. Table 6 lists how these pins connect to the microcontroller. A description of all pins on the accelerometer IC is presented in Table 20 of Appendix 4A – Summary of IC Datasheet (MMA7361LC).

Table 6. AM3X Pin Connections to Microcontroller

Accelerometer		Microcontroller	
Pin	*Description*	*Pin*	*Description*
VIN	Input voltage to the device.	VIN	Input voltage to the board.
X	Analogue output for the X-axis reading.	A0	Converts analogue inputs to 10-bit ADC.
Y	Analogue output for the Y-axis reading.	A1	Converts analogue inputs to 10-bit ADC.
Z	Analogue output for the Z-axis reading.	A2	Converts analogue inputs to 10-bit ADC.
GND	Ground.	GND	Ground.

4.3 Wireless Components

USB has been used to communicate between the controller and server for the initial prototype. However, USB length places restrictions on hand movement and may interfere with gameplay. A Bluetooth module has been used for wireless communication, which requires the integration of a portable power source such as a battery.

Decision Matrix 6. Connection.

Decision	Opportunities	Constraints	Req.
USB	Plug and play.	Restricts hand movement.	5
Bluetooth	Plug and play, wireless.	Limited to range of Bluetooth module.	

4.3.1 HC-06 Bluetooth Module

The HC-06 Bluetooth module is a ‘slave’ device, which means that it can only connect to a single ‘master’ computer (Xin et al., 2011). The HC-06 is a power class 2 module, giving a maximum range of 10m (Xin et al., 2011; Sparkfun Electronics, 2016c). The pin connections from the module to the microcontroller are shown in Table 7.

Table 7. HC-06 Pin Connections to Microcontroller

HC-06 Bluetooth		Microcontroller	
Pin	*Description*	*Pin*	*Description*
VCC	Input voltage.	5V	Input voltage.
TX	Transmit terminal.	D0	RX equivalent terminal.
RX	Receiving terminal.	D1	TX equivalent terminal.
GND	Ground.	GND	Ground.

Alternatives to the HC-06 are the HC-05, and TinySine 4.0 Bluetooth Low Energy (BLE) modules. The HC-05 is a master and slave device. However, it is not required in this project because the controller will only need to connect to a single computer. The TinySine 4.0 BLE has also been considered for lower power consumption. It is a more expensive module that can connect to iOS devices such as iPhones. The drawback of all modules is the size. In the future, an Arduino board with integrated Bluetooth should be considered.

Decision Matrix 7. Bluetooth Module.

Decision	Opportunities	Constraints	Req.
HC-06	Slave device.	Size.	5
HC-05	Slave and master device.	Size.	
TinySine 4.0 BLE	Low power consumption, iOS compatible.	Size, cost.	

4.3.2 9V Battery, Connector, and 5V Regulator

The portable power source is a 9V battery connected to a 5V regulator via a battery snap, as shown in Figure 16. Connecting a 9V battery directly to V_{in} on the Freetronics LeoStick will damage the board. Therefore, the connection summarisd on Table 8 must be adhered to.

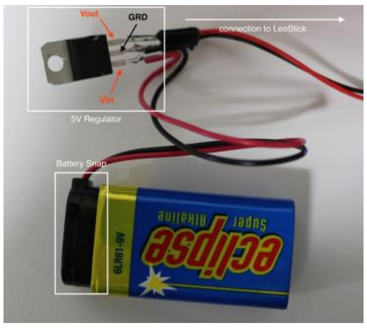


Figure 16. Power system.

Table 8. 9V Battery connections to 5V regulator and microcontroller

9V Battery		5V Battery Regulator		Microcontroller	
+	Positive	V_{in}	Input voltage.		
-	Negative	GRD	Ground.	GRD	Ground.
		V_{out}	Output voltage.	V_{in}	Input voltage.

Using a 5V regulator is inefficient since nearly half of the available power from the 9V battery is converted to heat. A more efficient form of power supply is to use a small LiPo StepUp or Boost converter board. The board takes small LiPo batteries and steps up the voltage to 5V. An additional benefit of this setup is reduction in prototype size.

Decision Matrix 8. Battery.

Decision	Opportunities	Constraints	Req.
9V Battery	Simple setup.	Power inefficient, large size.	5
5V LiPo Battery	Size reduction, power efficient.	Complex setup.	

4.4 Miniaturisation

Several prototypes have been made in the process of miniaturising the controller hardware. The initial and final prototypes use the same components to achieve similar functionality. However, the Freetronics LeoStick has been preferred in the final prototype for its smaller size. In the future, different components must be used to further miniaturise the hardware. In particular, the Bluetooth module and current form of power supply are limiting factors to the size of the controller. A development board with integrated Bluetooth such as the DFRobot Bluno Nano, and a smaller power supply such as LiPo batteries may be used as replacements.

4.4.1 Initial Prototype

The initial prototype features the AM3X accelerometer, breadboard, and the Freetronics Eleven. The physical prototype and circuit diagram is displayed in Figure 17 and
Figure 18. The breadboard setup has been preferred over a prototyping shield and wire-only setup because it does not require soldering. This means that components can be reused for other prototypes in the future.

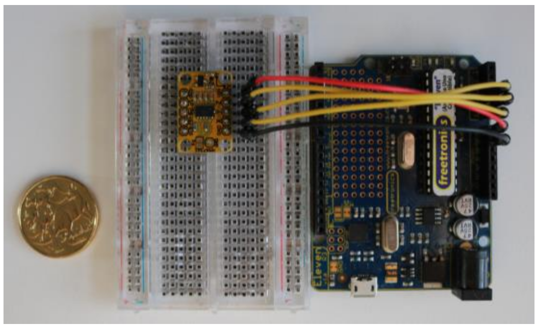


Figure 17. Initial prototype.

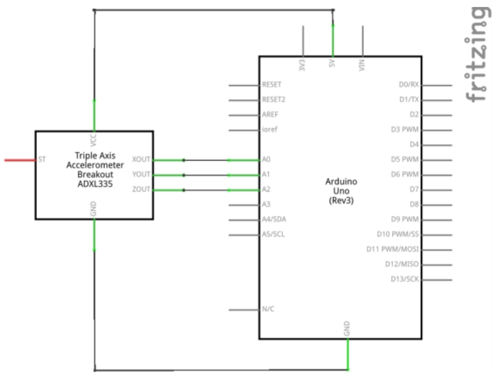


Figure 18. Circuit diagram for initial prototype.

4.4.2 Final Prototype

The final prototype can wirelessly connect to a computer and fits in the palm of the hand. It combines an AM3X accelerometer with a HC-06 Bluetooth module, 9V battery, and 5V regulator on the Freetronics LeoStick. The physical prototype and circuit diagram is displayed in Figure 19 and Figure 20.

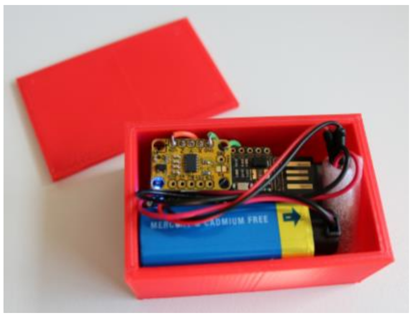


Figure 19. Final prototype.

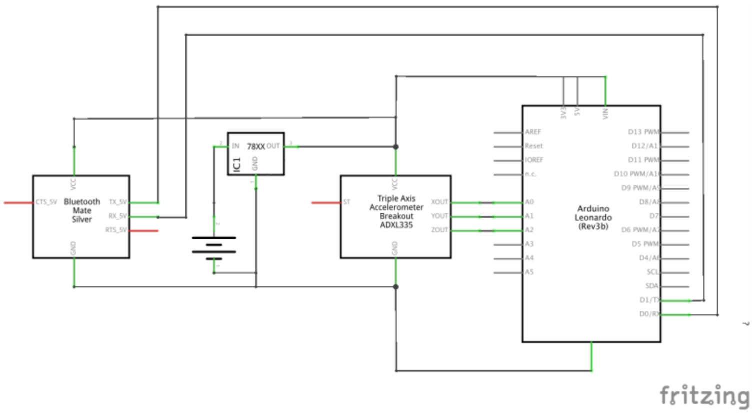


Figure 20. Circuit diagram for final prototype.

Components have been soldered together to fit in a 3D printed rectangular case with dimensions 7cm × 3cm × 4.5cm. Individual components and materials used for the assembly are shown in Figure 21. The total cost of the final prototype is AUD $97.40, priced from Jaycar. Functional improvements may include incorporating sensory feedback such as lights, sound or vibration through LEDs, sound modules, or vibration motors. For example, LEDs can be programmed to change colour at an increasing RPS.

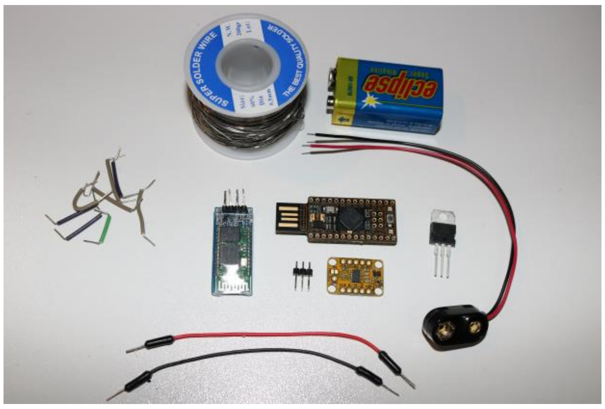


Figure 21. Components for final prototype.

Table 9. Bill of materials for final prototype.

Component	Cost ($)
1x Freetronics LeoStick	47.95
1x AM3X Accelerometer	22.95
1x HC06 Bluetooth	19.95
1x 5V Voltage Regulator	1.85
1x 9V Battery	3.95
1x 9V Battery Snap	0.75
Total	97.4

Alternative cases have been considered, which includes a plastic capsule or wristband pouch that can be purchased online. The wristband pouch offers a wearable take on the controller. However, a 3D printed case has been preferred for a custom design and fast turnover. An alternative 3D printed case using a spherical design is demonstrated in Appendix 4B – Hardware Miniaturisation. However, the spherical case has not been used since it is more difficult to fix the accelerometer axes in a sphere.

Decision Matrix 9. Circuit.

Decision	Opportunities	Constraints	Req.
Breadboard	No soldering, modular.	Large size.	1, 5
Prototyping Shield	Small size, modular.	Requires soldering.	
Wired	Small size.	Requires soldering, not modular.	

Decision Matrix 10. Casing.

Decision	Opportunities	Constraints	Req.
3D Printed Case	Fast turnover, custom design.	3D printer, filament quality.	6
Plastic Capsule	Premade.	Custom size difficult to source.	
Wristband Pouch	Wearable design.	Custom size difficult to source.	

Decision Matrix 11. Shape.

Decision	Opportunities	Constraints	Req.
Rectangular	Fix controller orientation.	Not ergonomic.	6
Circular	Ergonomic.	Hard to fix controller orientation.	

Rotation Data & Analysis

Raw data has been collected to determine how the program will measure RPS and detect rotation direction. There are two parts to this process: understanding the basics of rotation, and collecting clockwise and anticlockwise acceleration readings. This chapter will also detail the models developed to identify anticlockwise and clockwise rotation gestures.

5.1 Unravelling Rotation

In the following study, the controller is assumed to be travelling in a circular path at constant speed without changing orientation. The movement can then be described as uniform circular motion (The Open University, 1994). Uniform circular motion lies in two axes in the 3D coordinate system. Since it exhibits periodic behaviour about an equilibrium, the one-dimensional (1D) projection of the movement over time is in simple harmonic motion (The Open University, 1994). In simple harmonic motion, the displacement of an object produces a sinusoidal wave as demonstrated in Figure 22.

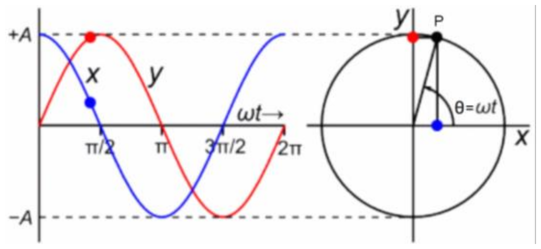


Figure 22. Displacement of an object in uniform circular motion over time at an angular velocity of 1 rad/s (Hespenheide, 2016).

As P completes one full trip around the circle, θ varies from zero to 2π measured in radians. The angle, θ, increases with time according to $\theta = \omega t$, where ω is the angular velocity in radians per second, and t is time in seconds. Since P is defined by two components, x and y, right-angle trigonometry can be used to model each component's position, linear velocity, and linear acceleration over time as follows.

Equation 4. y component position, velocity, and acceleration functions for anticlockwise uniform circular motion (The Open University, 1994)

$$\sin\theta = \frac{y}{A} \rightarrow y = A\sin\theta = A\sin(\omega t) = A\sin(2\pi f t)$$
$$\frac{dy}{dt} = 2\pi f A\cos(2\pi f t)$$
$$\frac{d^2y}{dt^2} = -4\pi^2 f^2 A\sin(2\pi f t)$$

Equation 5. x component position, velocity, and acceleration functions for anticlockwise uniform circular motion (The Open University, 1994)

$$\cos\theta = \frac{x}{A} \rightarrow x = A\cos\theta = A\cos(\omega t) = A\cos(2\pi f t)$$
$$\frac{dx}{dt} = -2\pi f A \sin(2\pi f t)$$
$$\frac{d^2x}{dt^2} = -4\pi^2 f^2 A \cos(2\pi f t)$$

Where A = radius of rotation or amplitude (m), θ = the angle point P makes with the X-axis, ω = angular velocity (rad/s), f = frequency (Hz), t = time (s) for both Equation 4 and Equation 5.

Equation 4 and Equation 5 shows the behaviour of the x and y components of point P as it moves anticlockwise. If point P moves clockwise, θ is negative because the reference line from the centre of the circle to P is now below the X-axis according to unit circle conventions (Cohen, 2005). Consequently, the relationship between position, linear velocity, and linear acceleration for clockwise rotation can be calculated as follows.

Equation 6. y component position, velocity, and acceleration functions for clockwise uniform circular motion

$$\sin(-\theta) = \frac{y}{A} \rightarrow y = -A\sin(\theta) = -A\sin(\omega t) = -A\sin(2\pi f t)$$
$$\frac{dy}{dt} = -2\pi f A \cos(2\pi f t)$$
$$\frac{d^2y}{dt^2} = 4\pi^2 f^2 A \sin(2\pi f t)$$

This calculation uses the trigonometric identity $\sin(-\theta) = \sin(\theta)$.

Equation 7. x component position, velocity, and acceleration functions for clockwise uniform circular motion

$$\cos(-\theta) = \frac{x}{A} \rightarrow x = A\cos(\theta) = A\cos(\omega t) = A\cos(2\pi f t)$$
$$\frac{dx}{dt} = -2\pi f A \sin(2\pi f t)$$
$$\frac{d^2x}{dt^2} = -4\pi^2 f^2 A \cos(2\pi f t)$$

This calculation uses the trigonometric identity $\cos(-\theta) = \cos(\theta)$.

For a given frequency and amplitude, it is possible to find the acceleration function for the *x* and *y* components (Equation 8 and 9). Since frequency is equivalent to revolutions per second (RPS), and amplitude is equivalent to rotation radius, Equation 4 to Equation 7 can be directly applied to the motion of the controller. The following graphs show the expected acceleration values for 0.67RPS (f = 0.67Hz) and a rotation radius of 12.5cm (A = 12.5cm). The graphs have been created using the following functions.

Equation 8. Anticlockwise acceleration functions for *x* and *y* components (f = 0.67Hz and A = 12.5cm)

$$\frac{d^2y}{dt^2} = -4\pi^2 \times 0.67^2 \times 0.125\sin(2\pi \times 0.67 \times t) = -2.2\sin(4.2t)$$
$$\frac{d^2x}{dt^2} = -4\pi^2 \times 0.67^2 \times 0.125\cos(2\pi \times 0.67 \times t) = -2.2\cos(4.2t)$$

Equation 9. Clockwise acceleration functions for *x* and *y* components (f = 0.67Hz and A = 12.5cm)

$$\frac{d^2y}{dt^2} = 4\pi^2 \times 0.67^2 \times 0.125\sin(2\pi \times 0.67 \times t) = 2.2\sin(4.2t)$$
$$\frac{d^2x}{dt^2} = -4\pi^2 \times 0.67^2 \times 0.125\cos(2\pi \times 0.67 \times t) = -2.2\cos(4.2t)$$

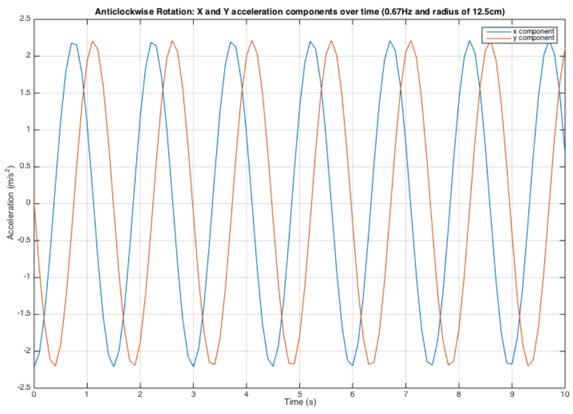


Figure 23. Anticlockwise Rotation: *x* and *y* acceleration components over 10s for 0.67RPS and a radius of 12.5cm.

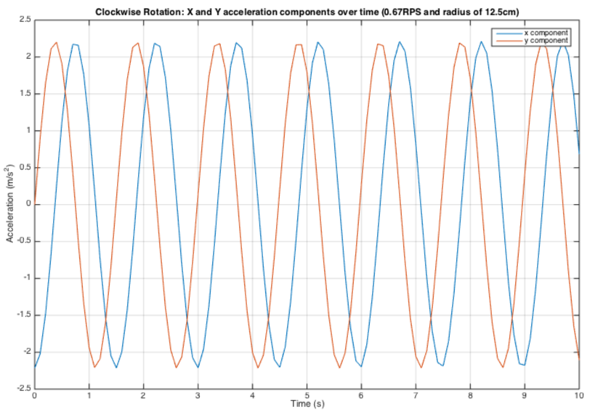


Figure 24. Clockwise Rotation: *x* and *y* acceleration components over 10s for 0.67RPS and a radius of 12.5cm.

Figure 23 and Figure 24 displays the acceleration of *x* and *y* components over a duration of 10 seconds. It can be seen that the *y* component waveform lags the *x* component waveform in anticlockwise rotation. However, it leads the *x* component waveform in clockwise rotation. The number of rotations is countable by the number of successive peaks or troughs of the waveform. Therefore, for 0.67RPS, there should be 7 peaks and 6 troughs in the first 10s of acceleration readings. Furthermore, the maximum and minimum acceleration readings are expected to be $\pm$2.2m/s^2.

Plotting *x* and *y* acceleration components at each time instant results in a two-dimensional (2D) projection of acceleration for anticlockwise rotation, as shown in Figure 25. In this graph, consecutive data points are 0.1s apart, and have been drawn in the same colour to show one full revolution before the next begins. The projection of clockwise rotation has been omitted since it displays a similar graph, but in the reverse direction.

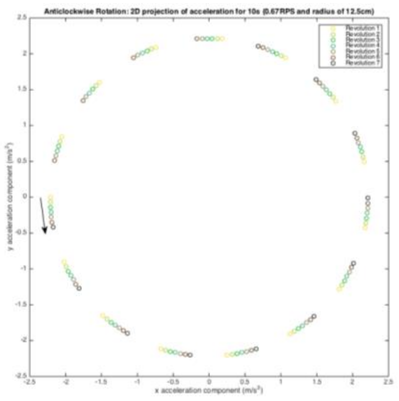


Figure 25. Anticlockwise Rotation: 2D projection of *x* and *y* acceleration components.

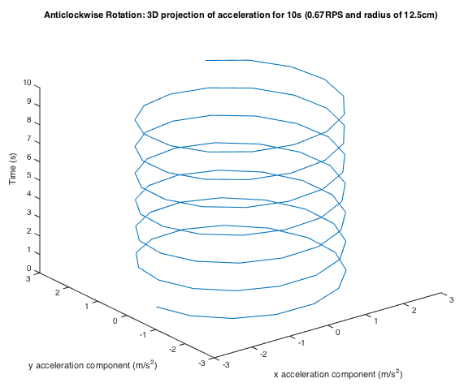


Figure 26. Anticlockwise Rotation: 3D projection of *x* and *y* acceleration components.

The time taken to complete one revolution, also known as the period, *T*, is the inverse of frequency. For this waveform, the period is $1/0.67Hz = 1.49s$. Since each data point is recorded in 0.1s increments, it takes $1.49s/0.1s \approx 15$ data points to show one full revolution. This explains why there are 15 clusters of data in Figure 25. The acceleration components of anticlockwise rotation have been projected against time to create a 3D graph displayed in Figure 26. The 3D plot helps to visualise the resultant motion of the *x* and *y* acceleration readings, so that it can be easily mapped onto the displacement of the device.

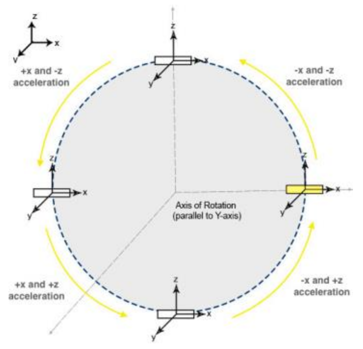


Figure 27. Anticlockwise Rotation: acceleration mapped onto displacement.

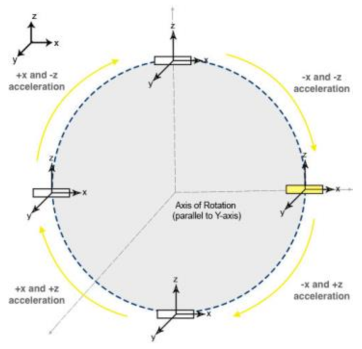


Figure 28. Clockwise Rotation: acceleration mapped onto displacement.

The mapping process commences on the highlighted starting point in Figure 27 and Figure 28. The accelerometer orientation is fixed throughout the rotation, with the *z* component equivalent to the *y* component in the analysis. It can be observed that there is no difference between anticlockwise and clockwise rotation in terms of sign changes in acceleration.

The observations can be verified by considering the resultant vector of the *x* and *z* acceleration components. The resultant vector is always pointing towards the centre, which is in line with the direction of centripetal acceleration (Giordano, 2009). This study shows that it is possible to model rotation gestures using sinusoidal functions. Other models considered include integrating acceleration readings to track the position of the controller in space, known as dead reckoning. However, the former model is easier to implement and ensures the program will only detect rotation gestures. The key assumptions that have been made is constant speed and fixed device orientation. Raw data has been collected in the following subsections to verify this behaviour. The procedure for collecting rotation data is available in Appendix 5A – Procedure for Data Collection.

Decision Matrix 12.Analytics Model.

Decision	Opportunities	Constraints	Req.
UCM & SHM	Detect only rotation gestures.	Limited to sine properties.	1
Dead Reckoning	Detect any positional movement.	Errors from integration drift.	

5.2 Raw Data Analysis

The subsequent graphs show acceleration readings when the accelerometer is rotating clockwise and anticlockwise. Accelerometer readings have been received as a digital signal, and converted to acceleration values in metres per second squared using Equation 10. The Matlab code to convert acceleration values and produce graphs is in Script 2 of Appendix 5B – Code for Matlab Graphing.

Equation 10. Digital output to acceleration conversion

$$acceleration\,\frac{m}{s^2} = \frac{\left(digital\ output\ \times \frac{5V}{1023}\right) - 1.65V}{0.8\,\frac{V}{g}} \times\ 9.8\ \frac{\frac{m}{s^2}}{g}$$

5.2.1 Anticlockwise Rotation Acceleration Data

Figure 29 displays the accelerometer outputs when rotated at 0.67RPS, which is equivalent to 40RPM, around a circle with a radius of 12.5cm. There is clear periodicity in the signal, and observations of frequency, vertical shift, and amplitude can be made on the acceleration readings.

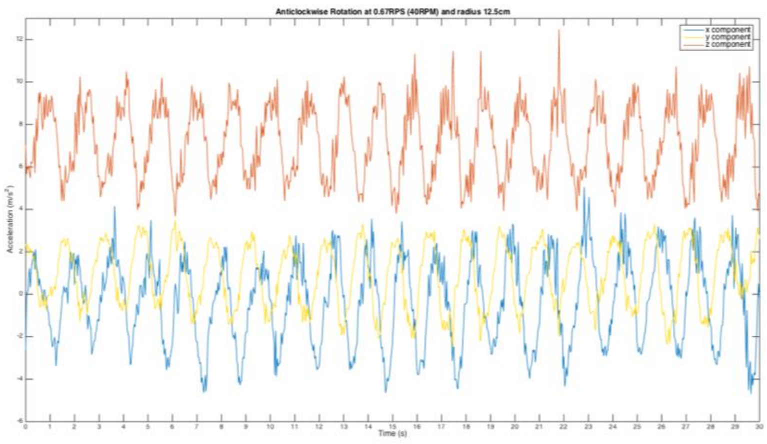


Figure 29. Anticlockwise rotation readings for 0.67RPS and a rotation radius of 12.5cm over 30s.

Frequency:

- There are approximately 20 peaks and troughs for all waveforms in a timeframe of 30s, which corresponds to 20 revolutions in 30s, or 40RPM.

Vertical shift:

- The vertical shift of the *x*, *y*, and *z* component waveforms is approximately constant for the 30s time interval. This indicates that the accelerometer has been held in a relatively fixed orientation throughout the rotation.
- The *x* and *y* component waveforms are vertically shifted by -0.2m/s^2 and 1.2m/s^2 respectively. This has been found by taking the average of each component's acceleration values.
- The *z* component waveform is vertically shifted by 7.2m/s^2. It is higher than the *x* and *y* component waveforms since acceleration in this direction also experiences gravity.

Amplitude:

- The *x* and *z* acceleration components have similar amplitudes of approximately $\pm$3m/s^2. In comparison, the *y* acceleration component has a smaller amplitude of approximately $\pm$2m/s^2. Since this movement occurs in 3D space, there is motion in the Y-axis as the user moves left or right while rotating the device.

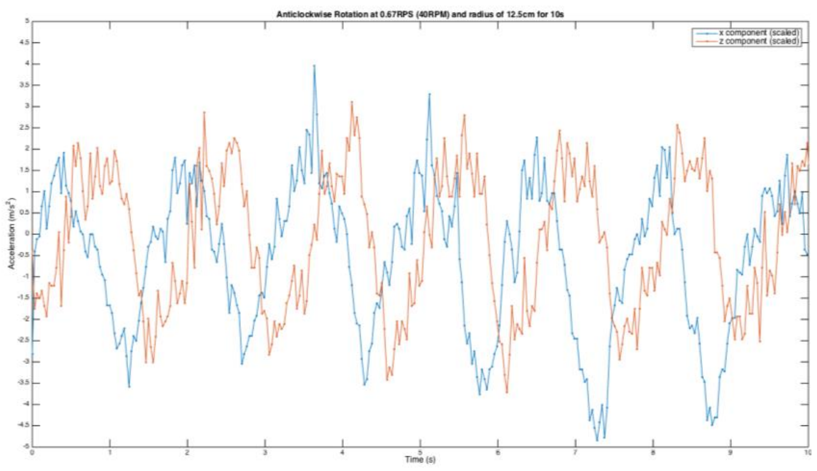


Figure 30. Anticlockwise rotation readings for 0.67RPS and a rotation radius of 12.5cm over 10s.

The *x* and *z* acceleration components for the first 10 seconds have been normalised in Figure 30 to compare against the expected sinusoidal functions of Equation 8. The values have been normalised by adding 0.2m/s^2 to the *x* components, and subtracting 7.2m/s^2 from the *z* components, resulting in a vertical shift of zero for both waveforms. It can be observed that rotation is anticlockwise since the *z* component waveform lags the *x* component waveform. However, the maximum and minimum values do not comply with the predicted values of $\pm$2.2m/s^2. This may be due to imperfect rotation about the 25cm diameter circle, and variable speed of rotation. Noise in the dataset is caused by vibration in the human arm, which results in spikes throughout the wave.

5.2.2 Clockwise Rotation Acceleration Data

Since the rotation readings for clockwise rotation acceleration data over a period of 30s are similar to anticlockwise rotation acceleration data, only the normalised data graph has been shown below.

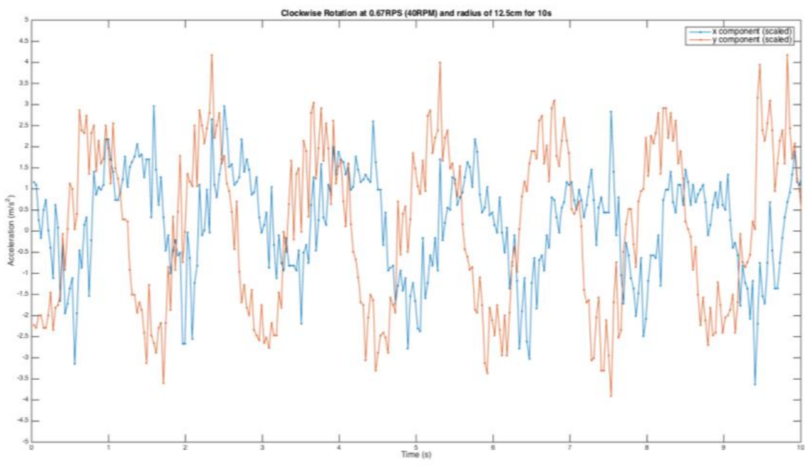


Figure 31. Clockwise rotation readings for 0.67RPS and a rotation radius of 12.5cm over 10s.

The normalisation of *x* and *z* components for the first 10 seconds in Figure 31 shows that the *z* component waveform leads the *x* component waveform. This matches the predicted behaviour in Equation 9. Once again, the maximum and minimum values do not comply with the expected $\pm 2.2\text{m/s}^2$ for a rotation of 0.67RPS around a radius of 12.5cm. However, the maximum and minimum values are considered to be reasonably close to the predicted values, given human error in maintaining a constant rotation diameter, speed, and accelerometer orientation.

A more accurate procedure to capture data would be to develop a robotic arm for a consistent rotation speed. This would reduce noise, and decrease discrepancy between predicted and observed values. Additionally, the *y* acceleration component may measure zero acceleration in this condition, since an ideal rotation would not deviate from the X-Z plane. However, this experiment shows that it is possible to model rotation using sinusoidal functions, which can be achieved through fundamental frequency estimation and curve fitting.

5.3 Models for Gesture Recognition

Raw data examined in the previous section will be used to develop models that detect anticlockwise and clockwise rotation. These models will be implemented in the server, which will read values from the microcontroller and output the RPS and rotation direction for use in the game.

5.3.1 Fundamental Frequency Estimation

Fundamental frequency estimation methods will be applied in this project to determine RPS from the acceleration readings. The fundamental frequency, f_0, is the lowest frequency in a periodic waveform, also known as the first harmonic (Serway et al., 2008). It is used in the field of audio signal processing, such as speech recognition, to process signals that have harmonically related components (Gerhard, 2003).

There are a number of methods that can be used to measure f_0. They fall into two main categories: 1) time domain based estimators, and 2) frequency domain based estimators (Robel, 2006; Gerhard, 2003). While time domain approaches are computationally simple, they lack robustness when noise is present in the input signal (Gerhard, 2003). In comparison, frequency domain approaches show more resilience to noise, but are computationally heavier (Gerhard, 2003). This project will use a time domain approach since the expected waveform does not have significant high-frequency components, making f_0 relatively easy to detect.

The underlying principle behind time domain based estimators is the determination of period, and the inverse of this value to give f_0 (Gerhard, 2003). Within this class, there are two popular methods: 1) zero-crossing rate (ZCR), and 2) autocorrelation. In this project, the former two methods have been tested before determining that autocorrelation would be suitable to measure RPS.

Decision Matrix 13. Rotation Detection.

Decision	Opportunities	Constraints	Req.
ZCR	Simple concept and implementation.	Sensitive to noise.	
Autocorrelation	Efficient, more resilient to noise.	Requires at least two periods.	1

5.3.1.1 Zero-Crossing Rate

The ZCR measures how often a waveform crosses zero per unit time (Gerhard, 2003). The rising or falling edge crossings are counted, and the rate at which they occur is used to estimate f_0, as shown in Figure 32. The benefit of this approach is that it is computationally inexpensive, and can be implemented in O(N) time complexity (Bogason, 2015). This means that the performance of the algorithm will grow linearly to the size of the input dataset (Bell, 2015). However, the main drawback is that it is very sensitive to noise (Bogason, 2015).

There are various open source algorithms to determine ZCR. The Python function in Snippet 1 has been obtained from a GitHub Gist by Endolith (2016). The parameters of the function are *sig* and *fs*, which corresponds to the signal, and the sampling frequency respectively.

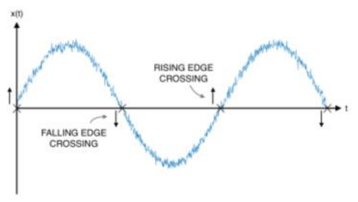


Figure 32. ZCR counts the number of rising or falling edge crossings to determine $\boldsymbol{f_0}$.

Snippet 1. ZCR Python function (Endolith, 2016).

```
1.  def freq_from_crossings(sig, fs):
2.      indices = find((sig[1:] >= 0) & (sig[:-1] < 0))
3.      crossings = [i - sig[i] / (sig[i+1] - sig[i]) for i in indices]
4.  return fs / mean(diff(crossings))
```

In this application, the signal is the x and z acceleration components, and the sampling frequency is the number of acceleration readings received in one second. The Freetronics Eleven utilised in this project has a maximum reading rate of 30ms, which means that the sampling frequency is $1000/30 = 33.3\ Hz$. In line 4 of Snippet 1, the sampling frequency is divided by the average of the difference between the indexes of consecutive rising-edge crossings to get f_0. Since the ZCR method requires positive and negative values, acceleration readings for each axis must be normalised by subtracting the mean from each component.

Acceleration readings in Figure 33 are a subset of anticlockwise rotation data in Figure 29. The dataset has been partitioned to 4 seconds of readings to show approximately two wavelengths. It is easier to estimate f_0 and perform curve fitting on a 4s time window since there is less variation on frequency in a smaller sample. Two programs have been made to partition the data, and perform f_0 estimation. They are available in Script 5 and Script 6 of Appendix 5D – Code for Fundamental Frequency Estimation and Curve-Fitting.

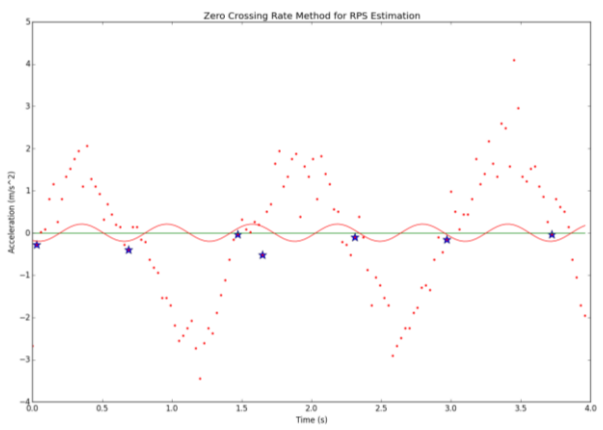


Figure 33. Fitted sine function using $\boldsymbol{f_0}$ from ZCR on anticlockwise rotation data from 0 to 4s. The rising crossing indices have been marked as blue stars.

Table 10. Output parameters for ZCR and curve fitting on anticlockwise rotation data from 0 to 4s.

Parameter	Value	Units
Rising Crossing Indices	1, 23, 49, 55, 77, 99, 124	-
Amplitude	-0.21	m/s^2
Frequency	1.6	Hz
Phase Shift	1.1	s
Vertical Shift	0	m/s^2

The ZCR method estimates f_0 to be 1.9Hz, which has been used to produce the fitted sine wave in Figure 33. The remaining parameters of the fitted sine wave are displayed in Table 10. It is clear that the fit is poor due to an incorrect f_0 estimation. This is due to noise in the dataset, which causes multiple rising crossings within one cycle.

The ZCR method only works for a few partitions, such as from 16 to 20s shown in Figure 34. In this dataset, there is only one rising crossing per cycle with indices 25, 70, and 113, as summarised in Table 11. The average of the differences between the indices is 44, and dividing the sampling frequency of 33.3Hz by this value returns 0.76Hz. Since the data was collected from a 0.67RPS rotation, the estimated value of f_0 is considered to be accurate given imperfect human rotation during the procedure.

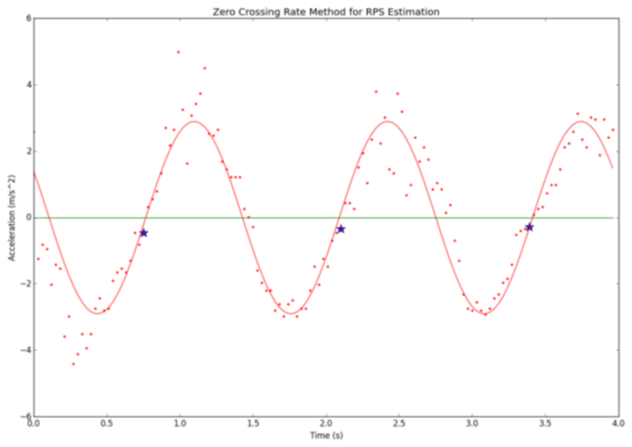


Figure 34. Fitted sine function using f_0 from ZCR on anticlockwise rotation data from 16 to 20s. The rising crossing indices have been marked as blue stars.

Table 11. Output parameters for ZCR and curve fitting on anticlockwise rotation data from 16 to 20s.

Parameter	Value	Units
Rising Crossing Indices	25, 70, 113	-
Amplitude	2.9	m/s^2
Frequency	0.76	Hz
Phase Shift	2.6	s
Vertical Shift	0	m/s^2

The fitted sine wave has an amplitude of 2.9m/s^2, which is close to the predicted value of 2.2m/s^2 in Equation 8. The phase shift is 2.6, which means that the sine wave is shifted 2.6 units to the left. There is no vertical shift since the acceleration readings have been normalised. Overall, the ZCR method provides an accurate estimation of f_0 only when there is minimal noise in the dataset. It is not an appropriate fundamental frequency estimator for this project.

5.3.1.2 Autocorrelation

Autocorrelation is a measure of how similar a waveform is compared to a delayed version of itself (Gerhard, 2003). This concept is demonstrated in Figure 35, which shows a signal contained in a time window. The signal is copied in Step 1, and because there is no time delay, the correlation of the original and copied signal is at a maximum. Step 2 shows that when the copy is delayed significantly, it does not look similar to the original signal in the overlapping area. Therefore, the autocorrelation value of this delay is small. When the copy is delayed even more in Step 3, it looks very similar to the original because the signal is periodic. Hence, the autocorrelation value of this delay shows a peak.

The time difference between the maximum peak and the first peak is equal to the period of the waveform, and in consequence the frequency of the signal can be found. Autocorrelation can be computed at O(Nlog(N)) time complexity, which means that doubling the size of the input data will have little effect on its performance (Bogason, 2015; Bell, 2015). This method is efficient when dealing with large datasets, and can be used to find f_0 even when the incoming signal contains noise (Bogason, 2015). However, a limitation is that it requires at least two periods to detect frequency (Rabiner, 1977). This means that to process a 0.67Hz signal, at least 3 seconds of the waveform must be analysed.

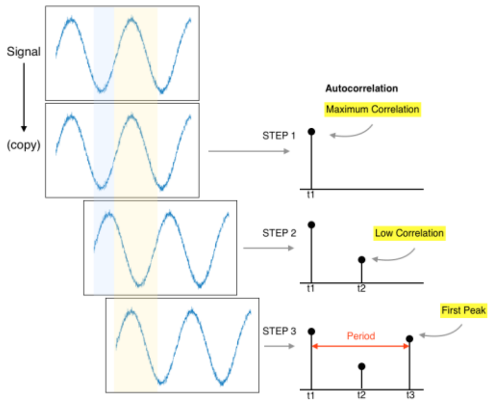


Figure 35. Autocorrelation is a function of how similar a signal is to a delayed version of itself.

Similar to the ZCR method, there are several open source algorithms to implement frequency autocorrelation in Python. The function applied in this project is also from the GitHub Gist by Endolith (2016). It takes the parameters *sig* and *fs* to return f_0. The function is shown in

Snippet 2, and has been used in Script 7 of Appendix 5D – Code for Fundamental Frequency Estimation and Curve-Fitting, to produce the following graphs.

Snippet 2. Frequency Autocorrelation Python Function (Endolith, 2016)

```python
def freq_from_autocorr(sig, fs):
    corr = fftconvolve(sig, sig[::-1], mode='full')
    corr = corr[len(corr)//2:]
    d = diff(corr)
    start = find(d > 0)[0]
return fs / px
```

The function uses quadratic interpolation to estimate the index of the first peak after the first low point in the autocorrelation. Similar to the ZCR method, it divides the sampling frequency by this index to find f_0. Figure 36 shows the fitted sine wave for the x component of anticlockwise rotation acceleration readings from 0 to 4s. The autocorrelation plot of this dataset is displayed in Figure 37 where the first peak has been marked.

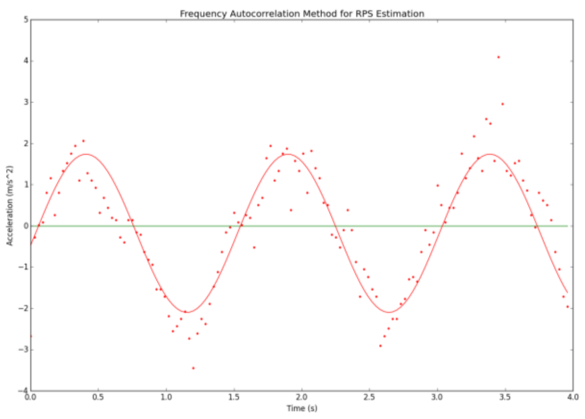


Figure 36. Fitted sine function using f_0 from frequency autocorrelation on anticlockwise rotation data from 0 to 4s.

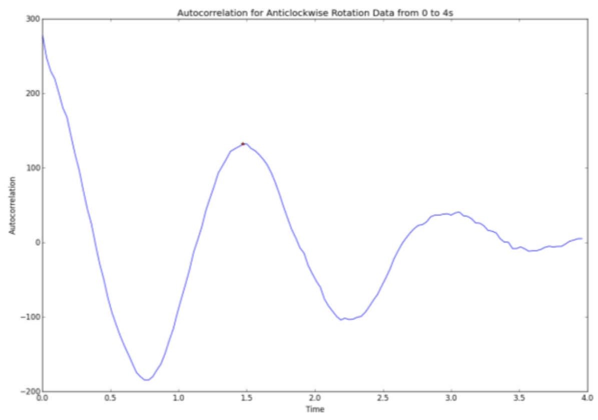


Figure 37. Autocorrelation graph of anticlockwise rotation data from 0 to 4s. The first peak is marked as a blue star.

Table 12. Output parameters for autocorrelation and curve fitting on anticlockwise rotation data from 0 to 4s.

Parameter	Value	Units
Amplitude	1.9	m/s^2
Frequency	0.67	Hz
Phase Shift	-0.16	s
Vertical Shift	-0.18	m/s^2

Frequency autocorrelation estimates f_0 to be 0.67Hz, which is the targeted RPS from the data collection procedure. The curve fitting function uses this frequency to model a sine wave that correctly fits the dataset, as opposed to the ZCR estimation in Figure 33. The amplitude of the sine wave is 1.9m/s^2, and is close to the predicted amplitude of 2.2m/s^2. Since frequency autocorrelation performs well with noisy data, it was used to estimate f_0 for a longer time window, as demonstrated in the following graphs.

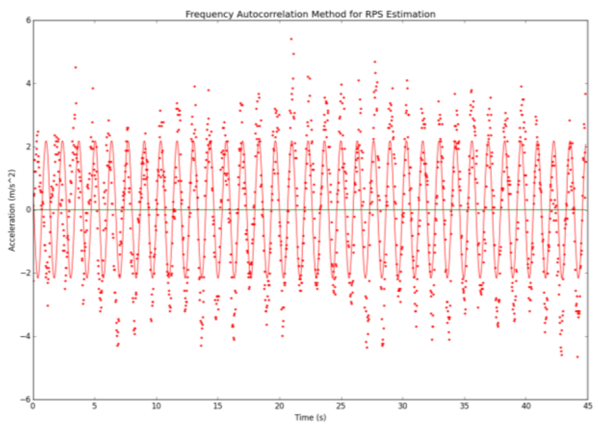


Figure 38. Fitted sine function using $\boldsymbol{f_0}$ from frequency autocorrelation on anticlockwise rotation data from 0 to 45s.

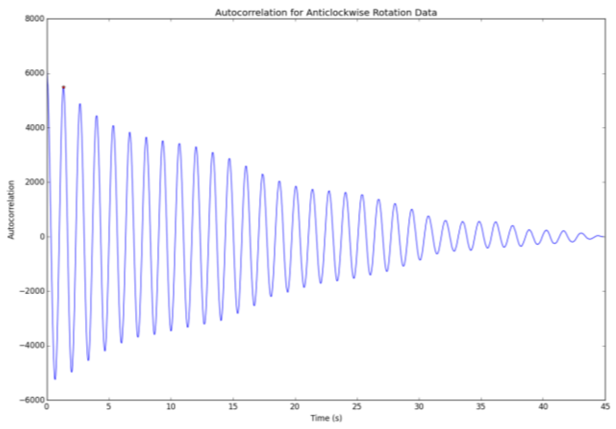


Figure 39. Autocorrelation graph of anticlockwise rotation data from 0 to 45s. The first peak is marked as a blue star.

Table 13. Output parameters for autocorrelation and curve fitting on anticlockwise rotation data from 0 to 45s.

Parameter	Value	Units
Amplitude	2.2	m/s^2
Frequency	0.75	Hz
Phase Shift	2.8	s
Vertical Shift	0	m/s^2

Given the estimated f_0 of 0.75Hz, the fitted sine wave in Figure 38 appears to closely model the acceleration readings. This implies that the controller was rotated at $0.75RPS \times 60s = 45RPM$, which is higher than the targeted value of 0.67RPS or 40RPM. The discrepancy may be due to the influence of noise, and a larger time window, resulting in a frequency that must satisfy the entire waveform. Figure 39 shows the autocorrelation plot of the signal, which exhibits an alternating sequence of positive and negative values. From this pattern it can be confirmed that the observed time series is not random, and is indeed sinusoidal (Nist Sematech, 2013).

5.3.2 Curve Fitting

The curve fitting function used to generate the sine waves in the previous section is shown in Snippet 3. It uses non-linear least squares to fit a function, f, to a dataset (Scipy.org, 2016). The relevant parameters are *f*, *x data*, and *y data*. In this application, *f* is the model sinusoidal function, shown in Snippet 4. Since the curve fitting function is not able to suitably estimate the frequency of the signal, frequency autocorrelation was used to output f_0. This value was then used to calculate the angular frequency $\omega = 2\pi f_0$ for input into the model. The variables *a*, *b*, and *c* are the amplitude, phase shift, and vertical shift of the sine wave respectively. These variables are returned by the curve fitting function below.

Snippet 3. Curve fitting function (Scipy.org, 2016)

```
scipy.optimize.curve_fit(f, xdata, ydata, p0=None, sigma=None, absolute_sigma=False, ch
eck_finite=True, bounds=(-inf, inf), method=None, jac=None, **kwargs)
```

Snippet 4. Model sinusoidal function

```
def func(acceleration_time_values, a, b, c):
    return a*np.sin(angular_f*acceleration_time_values + b) + c
```

The remaining parameters *x data* and *y data* is the time, and X, Y, or Z-axis acceleration readings respectively. It was found that time must be linearly spaced for both the frequency autocorrelation, and the curve fitting function to accurately determine the model variables. This limitation has been accommodated in the program by interpolating the acceleration readings to linearly spaced time with an interval equivalent to the reading rate. The acceleration readings have been linearly interpolated because the microcontroller sends data in uneven intervals with a variation of approximately $\pm 2ms$. For example, a reading rate of 30ms would send data in an increment of 0, 31ms, 60ms, and 92ms, instead of 0, 30ms, 60ms, and 90ms.

Other curve fitting functions considered includes Matlab EzyFit, and MS Excel Solver. However, similar to the SciPy curve fitting function, it was found that both Matlab EzyFit and MS Excel Solver does not accurately perform sinusoidal curve fitting without a fundamental frequency estimator. Furthermore, only SciPy and Matlab can process data in real time.

Decision Matrix 14. Analytics Suite.

Decision	Opportunities	Constraints	Req.
SciPy Curve Fitting	Real-time, open-source.	Inaccurate without f_0 estimator.	1, 4
Matlab EzyFit	Real-time, GUI interface.	Proprietary, inaccurate without f_0 estimator.	
MS Excel Solver	GUI interface.	Not real-time, requires manual calculation.	

Decision Matrix 15. Linearise Time.

Decision	Opportunities	Constraints	Req.
Linearised Data	Required for models.	Acceleration values must be interpolated.	1
Non-linearised Data	Receive raw time data.	Not suitable for models.	

Decision Matrix 16. Linear Interpolation.

Decision	Opportunities	Constraints	Req.
Linear Interpolation	Accurate for small increments.	Inaccurate for large increments.	1
No Interpolation	Receive raw acceleration data.	Mismatch between linearised time and acceleration.	

5.3.3 Direction Determination

The model to determine rotation direction requires two pieces of information:

1) The rotational position of the controller.
2) The change in acceleration vector, called *jerk*, for the *x* and *z* components.

In Figure 27 and Figure 28, the sign of the *x* and *z* acceleration components have been mapped onto the rotational position of the controller. It was observed that the sign of *x* and *z* acceleration components are the same regardless of rotation direction. This is because centripetal acceleration always acts towards the centre in uniform circular motion. Therefore, given the sign of the *x* and *z* acceleration values, it is possible to determine the position of the controller at any point in the rotation.

The change in acceleration vector for any given time interval can be found by subtracting the *x* and *z* acceleration components at time t_1, from the *x* and *z* acceleration components at time t_2. This vector is also known as jerk, and is a function of the derivative of acceleration with respect to time. Differentiating the acceleration functions for anticlockwise rotation and clockwise rotation results in Equation 11 and Equation 12. The sign changes of *x* and *z* jerk components can also be mapped onto the rotational position of the controller by referring to Figure 40 and Figure 41, where f has been defined to be 0.67Hz, and A to be 12.5cm.

Equation 11. *x* and *y* component jerk functions for anticlockwise uniform circular motion.

$$\frac{d^3y}{dt^3} = -8\pi^3 f^3 A\cos(2\pi f t)$$
$$\frac{d^3x}{dt^3} = 8\pi^3 f^3 A\sin(2\pi f t)$$

Equation 12. *x* and *y* component jerk functions for clockwise uniform circular motion.

$$\frac{d^3y}{dt^3} = 8\pi^3 f^3 A\cos(2\pi f t)$$
$$\frac{d^3x}{dt^3} = 8\pi^3 f^3 A\sin(2\pi f t)$$

Note: the *y* component in the equation is equivalent to the *z* component in the rotation.

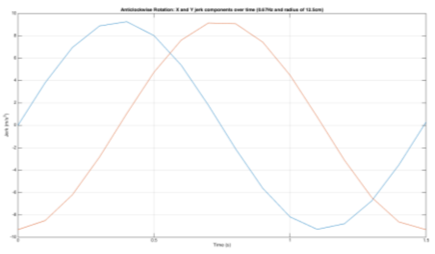


Figure 40. Anticlockwise Rotation: *x* and *y* jerk components for 0.67RPS and a radius of12.5cm.

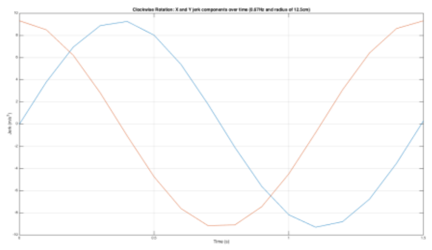


Figure 41. Clockwise Rotation: *x* and *y* jerk components for 0.67RPS and a radius of 12.5cm.

While the *x* component of jerk is the same for both rotation directions, the *z* component is different in terms of sign, as shown in Figure 42 and Figure 43. In anticlockwise rotation, the *z* component is positive in the left semicircle, and negative in the right semicircle. However, in clockwise rotation, the *z* component is positive in the right semicircle, and negative in the left semicircle.

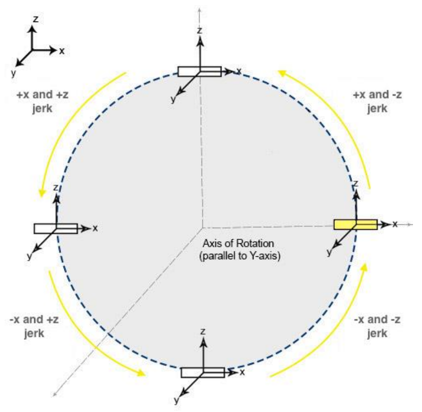


Figure 42. Anticlockwise Rotation: jerk mapped onto displacement.

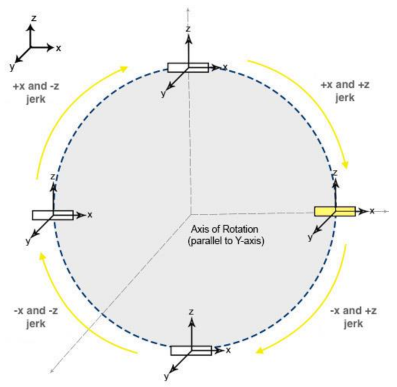


Figure 43. Anticlockwise Rotation: jerk mapped onto displacement.

The resultant vectors at each quadrant in the rotation are shown in Figure 44. For example, positive jerk for both *x* and *z* components would have a resultant vector directed towards the North East. This would occur in the left top, or right bottom quadrant of rotation. Therefore, to determine rotation direction, the rotational position must be known. If the position is on the left top quadrant, then a North East jerk vector implies the controller is moving anticlockwise. This is reasonable since the direction of jerk in uniform circular motion is parallel to the velocity vector, but opposite in direction.

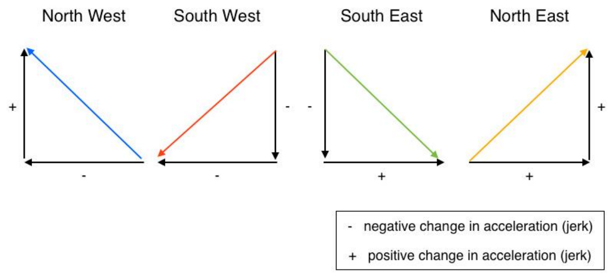

North West
South West
South East
North East
- negative change in acceleration (jerk)
+ positive change in acceleration (jerk)

Figure 44. Possible resultant vectors showing direction of jerk.

Rotational position and jerk is mapped in Snippet 5, where anticlockwise rotation has been recorded as -1, and clockwise rotation as 1. If the output is not -1 or 1, the program will return an unknown direction, which ensures that only rotation gestures will be detected. Additionally, since the user may change rotation direction at any instant, a majority vote is taken to determine the rotation direction within an analysis frame. However, a drawback of this method is lag, since the most recent output may not have the greatest weighting.

Snippet 5. Rotational position and jerk mapping

```
accel_vector_direction_and_position_mapping = {
    ('NE', 'LT'): -1,
    ('NE', 'RB'):  1,
    ('NE', 'L' ): -1,
    ('NE', 'B' ):  1,
    ('SE', 'LB'):  1,
    ('SE', 'RT'): -1,
    ('SE', 'L' ):  1,
    ('SE', 'T' ): -1,
    ('SW', 'LT'):  1,
    ('SW', 'RB'): -1,
    ('SW', 'R' ): -1,
    ('SW', 'T' ):  1,
    ('NW', 'RT'):  1,
    ('NW', 'LB'): -1,
    ('NW', 'R' ):  1,
    ('NW', 'B' ): -1,
    ('N' , 'L' ): -1,
    ('N' , 'R' ):  1,
    ('S' , 'L' ):  1,
    ('S' , 'R' ): -1,
    ('E' , 'T' ): -1,
    ('E' , 'B' ):  1,
    ('W' , 'T' ):  1,
    ('W' , 'B' ): -1
}
```

Two alternative models have been developed in the process of determining rotation direction. These methods rely on the concept of a leading waveform; if the *x* component leads the *z* component waveform, the rotation is anticlockwise. The opposite is true in clockwise rotation.

The first method compares the phase shift of both signals to determine the leading waveform. If the *x* component waveform has a phase shift, θ, greater than the *z* component waveform, the rotation is anticlockwise. To compare phase shift, the fitted sine functions must have a positive amplitude, and the same frequency (AspenCore, 2016). If the fitted sine wave has a negative amplitude, the phase angle must be added by π. However, a drawback to this method is that the current frequency autocorrelation algorithm does not always return the same frequency for both *x* and *z* acceleration components. Furthermore, it will not be able to detect dynamic changes in rotation direction due to the curve-fitting algorithm. Consequently, it is not a robust way to determine rotation direction.

The second method compares the time of consecutive peaks in both signals. If the times at which peaks occur in the *x* component waveform are smaller than the times at which peaks occur in the *y* component waveform, the rotation is anticlockwise. However, it was found that this method is error prone due to the implicit assumption of order. For example, a correct comparison of anticlockwise rotation requires the *x* component peak and consecutive *y* component peak to be present at the beginning of the analysis frame.

Decision Matrix 17. Rotation Direction Detection.

Decision	Opportunities	Constraints	Req.
Rotation Position & Jerk	Robust, rotation gesture exclusive.	Majority vote, lag.	2
Phase Shift Comparison	Conceptually simple.	Changing rotation direction.	
Peak Times Comparison	Conceptually simple.	Error prone, assumes order.	

Decision Matrix 18. Normalise Data.

Decision	Opportunities	Constraints	Req.
Normalised Data	Sign changes in acceleration.	Not real acceleration readings.	2
Raw Data	Real acceleration readings.	No sign changes in acceleration.	

5.3.4 Axes Conventions

The models analyse the X and Z-axis outputs of the accelerometer. The underlying assumption is that the Y-axis is the axis of rotation. To generalise the model, the axes defining the plane of rotation is labelled as east and up, and the axis of rotation is labelled as north in the program. This is an adaptation of the East, North, Up (ENU) axes convention used in flight dynamics for a standard reference frame (Koks, 2008).

The program asks the user for input on the ENU axes, which allows rotation gestures to be defined in six different planes. For example, if the rotation gesture lies on the X-Z plane, where the positive Z-axis points to the ceiling, and the positive X-axis points forwards, east is ‘+x’, north is ‘+y’ and up is ‘+z’. In the eye of the user, clockwise rotation occurs as a forward rotating motion, whereas anticlockwise rotation occurs as a backward rotating motion.

While ENU provides flexibility in the way the user holds the controller and defines the plane of rotation, it still requires the accelerometer orientation to be fixed throughout the rotation. In the future the case should be designed so that the controller can only be held in one orientation for a universal clockwise and anticlockwise rotation gesture. For now, user testing should be done to determine the most intuitive plane of rotation for users.

Chapter 6

The Controller, Server and Game Interface

This chapter focuses on data flow in the program, and the communication interface between the controller, server, and game. The user interface has also been presented to show key features of the program.

6.1 Data Flow in the Controller, Server and Game

The purpose of the system is to detect rotation gestures, and rotation direction for input to a game. Three subsystems have been created to achieve this: the controller, server, and game.

1. The controller reads and sends accelerometer values to the server.
2. The server processes acceleration readings and returns the RPS value, and the rotation direction, which is either anticlockwise or clockwise.
3. The game accepts the RPS value and rotation direction as input.

The interface between the controller, server, and game is shown in Figure 45, and will be explained in the following sections. The code is available here: https://github.com/mariafoo/Building-a-Physical-Analogue-for-Feedback-Rich-Systems.

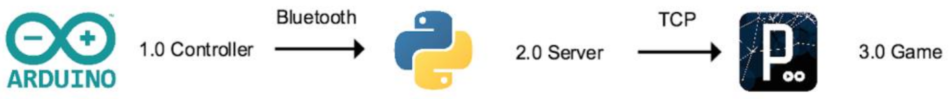


Figure 45. Communication interface between the controller, server, and game.

6.1.1 The Controller

The controller is programmed through the Arduino IDE. The sketch gives instructions to the microcontroller to read acceleration at a sampling delay of 30ms. A 30ms delay has been used to accommodate the total processing time of the system. Acceleration readings are sent via a Bluetooth connection between the controller and server. The Freetronics LeoStick requires Serial1 to communicate using Bluetooth, whereas Serial is reserved for USB communications device class (CDC). Functionality to switch between Serial and Serial1 has been incorporated, so that both USB CDC and Bluetooth connections are supported.

Decision Matrix 19. Controller server-communication standard.

Decision	Opportunities	Constraints	Req.
Serial Communication	USB communication.	No Bluetooth communication.	4, 5
Serial1 Communication	Bluetooth communication.	No USB communication.	4, 5

Decision Matrix 20. Sampling delay.

Decision	Opportunities	Constraints	Req.
30ms	Covers total system processing time.	Less data points.	4, 5
0ms	Instantaneous data transmission.	Limited by accelerometer bandwidth.	

6.1.2 The Server

The server is programmed using Python 3 and is dependent on several Python libraries. It asynchronously analyses acceleration readings, and plots the acceleration values and fitted sine curves on a graph in real time. The flow of acceleration data is shown in Figure 46.

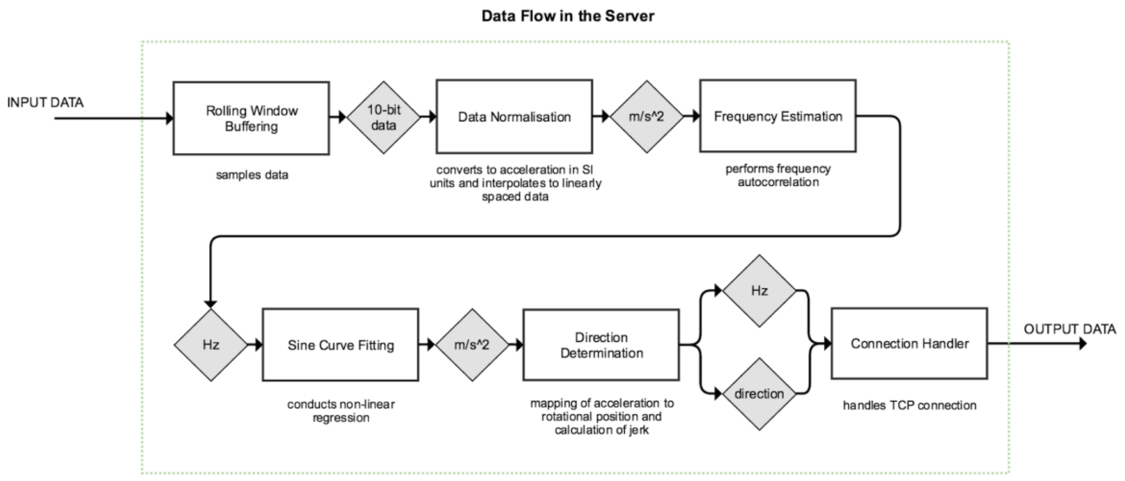


Figure 46. Flow of acceleration data in the server.

6.1.2.1 Data Sampling

Input acceleration data is received from the controller and is processed in a rolling window. Rolling window is a sampling technique that contains two separate buffers of different intervals. One buffer is the current working buffer, and the second buffer is the historical buffer. The current working buffer is updated as acceleration readings are received. At a specified interval, accumulated acceleration readings in the current working buffer are sent to the historical buffer. The current working buffer is then cleared, along with an equivalent interval of the earliest acceleration readings in the historical buffer.

The specified time interval of the current working buffer is 150ms. A sampling delay of 30ms means the current working buffer accumulates 5 sets of acceleration readings to send to the historical buffer before it is cleared. The historical buffer has a time interval of 4s, and holds 133 sets of acceleration readings.

Sampling techniques are used in order to perform the fundamental frequency estimation and curve-fitting algorithm, which requires a signal of at least two periods for analysis. Consequently, with a time window of 4s, the minimum rating of the controller is 0.5RPS or 30RPM. A limitation to the rolling window is that analysis can only be started once the historical buffer accumulates 4s of acceleration readings. However, the advantage is that it provides an up-to-date representation of the RPS and rotation direction, as opposed to simple periodic sampling. In simple periodic sampling, the historical buffer and current working buffer have the same interval. Hence, a time window of 4s will produce an equivalent lag for the rotation analysis. In contrast, the rolling window sampling technique will only lag by 150ms.

The sampling technique can be improved by placing a higher weighting on the latest acceleration readings in the historical buffer to reflect the RPS and rotation direction for the most up to date acceleration readings.

Decision Matrix 21. Software.

Decision	Opportunities	Constraints	Req.
Python 3	Open source, large community.	Limited to libraries available.	1

Decision Matrix 22. Graphing Method.

Decision	Opportunities	Constraints	Req.
Static	Data collection for post processing.	Not interactive, no feedback.	
Real Time	Interactive, rapid feedback.	Lag caused by sampling technique.	1, 4

Decision Matrix 23. Sampling Method.

Decision	Opportunities	Constraints	Req.
Simple Periodic	Easy to implement.	Lag in RPS and rotation direction output.	
Rolling Window	Faster processing.	Data in historical buffer has greater weighting.	1, 4

Decision Matrix 24. Rolling Window.

Decision	Opportunities	Constraints	Req.
150ms	Shorter lag.	Smaller interval ⇒ less accurate RPS value.	

Decision Matrix 25. Time Window.

Decision	Opportunities	Constraints	Req.
4s	30RPM minimum rating.	Minimum RPS may be higher in practice.	1, 4
3s	40RPM minimum rating.	Minimum RPS may be higher in practice.	

Decision Matrix 26. Initialise Analysis.

Decision	Opportunities	Constraints	Req.
Immediately	No delay after initisalisation.	Incompatible with frequency estimation.	1, 4
Buffer First	Compatible with frequency estimation.	4s delay between start and analysis.	

6.1.3 The Game Client

The system dynamics game has been created using Processing, which is an open source computer programming language and IDE built for the electronic arts. The game accepts RPS and rotation direction information from the server using a transmission control protocol (TCP). TCP is the only communication method available for Processing applications.

The game has been designed following a Processing tutorial on Flappy Bird. Key features of the game include a constant downwards acceleration due to gravity, and a scoring system based on the number of walls that has been passed. The user controls the addition or removal of heat to influence the hot air balloon’s height. Feedback occurs in terms of the discrepancy between the current and desired height. The hot air balloon does not have a horizontal velocity, rather the walls are moving at a constant speed to the left to create an illusion of movement.

This game can be used as an educational tool in two ways. Firstly, it can be used to show the behaviour of a stock to flows. Secondly, it can be used as a platform to create the behaviour of a dynamic system through programming.

Decision Matrix 27. Game.

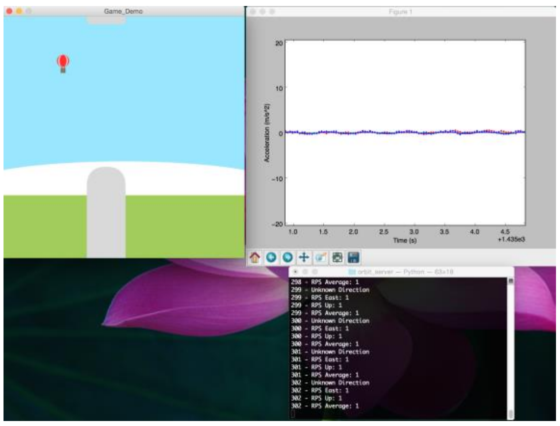
Decision	Opportunities	Constraints	Req.
Processing	Open source, programming for electronic arts.	Small community, limited documentation.	3

Decision Matrix 28. Communication standard.

Decision	Opportunities	Constraints	Req.
TCP	Standard functions available.	Required by Processing application.	3

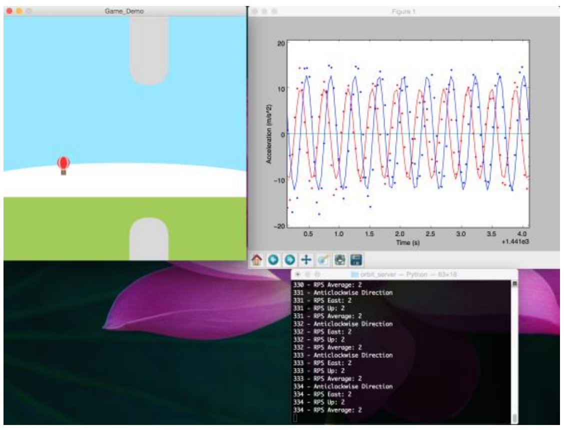
6.2 User Interface

The controller, and server runs from the terminal, and the game client runs from the Processing IDE. To launch the system, the controller must be connected to the computer. The following figures provide a working example of the system.



```
298 - RPS Average: 1
299 - Unknown Direction
299 - RPS East: 1
299 - RPS Up: 1
299 - RPS Average: 1
300 - Unknown Direction
300 - RPS East: 1
300 - RPS Up: 1
300 - RPS Average: 1
301 - Unknown Direction
301 - RPS East: 1
301 - RPS Up: 1
301 - RPS Average: 1
302 - Unknown Direction
302 - RPS East: 1
302 - RPS Up: 1
302 - RPS Average: 1
```

Figure 47. User interface showing when accelerometer is at rest.



```
330 - RPS Average: 2
331 - Anticlockwise Direction
331 - RPS East: 2
331 - RPS Up: 2
331 - RPS Average: 2
332 - Anticlockwise Direction
332 - RPS East: 2
332 - RPS Up: 2
332 - RPS Average: 2
333 - Anticlockwise Direction
333 - RPS East: 2
333 - RPS Up: 2
333 - RPS Average: 2
334 - Anticlockwise Direction
334 - RPS East: 2
334 - RPS Up: 2
334 - RPS Average: 2
```

Figure 48. User interface showing when the accelerometer is moving in an anticlockwise direction.

Figure 47 and Figure 48 show the graphical user interface in three screens. The first screen is the system dynamics game, the second screen is the plot of acceleration values and curve-fitted data, and the third screen is the terminal output of rotation direction, and acceleration. As the user rotates the controller in an anticlockwise rotation, this will bring the hot air balloon down, faster than the constant gravitational acceleration.

Discussion

The demonstration and use of Agile methodology in this project was no less in value than the prototype gaming system it produced. Three development streams were initially defined: the controller, server and game. It was found that while high-level requirements have been retained, more detailed requirements have evolved, as expected for an unfamiliar problem and solution domain. Further, there was increasing need to have feedback loops in the design process since the development streams are interdependent of each other. A downstream design decision or test finding in one stream may require changes to an earlier decision in another work stream.

The use of Agile allowed for requirements to be progressively introduced to enhance the system's capabilities. For example, simple periodic sampling was replaced by rolling window sampling because lag was observed when the controller and game was tested. Simple periodic sampling was implemented at first, since it was an easier sampling method to program, even though it did not offer the best solution. In this manner, faster delivery of proof of concepts was achieved, and provides new avenues for further development of the prototype that meets the remaining requirements.

Through the development, porotypes were built on ad-hoc basis, although only two prototypes have been shown in Figure 17 and Figure 19, the first was configured with a breadboard connection, and was intended to test the sensors, collect rotation data, and confirm the possibility of measuring rotation gestures. The second was configured with a wired connection, and was intended to test the Wireless communication interface. Remaining prototypes that have been created but discarded during development is available in

The drawback of the design process was the lack of documentation. A proper testing strategy would have minimised the number of prototypes and tests conducted, although given the scope of the project, rework was minimal. For larger and more complex systems, a table of decisions made at each deployment phase will ensure that the project progresses on track in spite of changing requirements.

Chapter 8

Conclusions and Further Work

8.1 Conclusions

The outcome of this design project is a minimum viable product of an educational tool to help young people learn system dynamics. Agile development principles have been used to overcome uncertainty in the problem and solution domain. In particular, a development, test, and deployment cycle has been used to incrementally add hardware components and software features to meet the system requirements. Foundation work has been created for future refinement of the platform. Specifically, four key artefacts has been delivered:

1. *System Dynamics Game:* a single player game that allows the player to control the inflow and outflow of heat to influence the height of a hot air balloon. This is a typical stock and flow problem, and allows the player to visualise the effects of flows on a stock in real time.

2. *Game Controller:* a wireless game controller that the player can intuitively use to control inflows and outflows via anticlockwise or clockwise rotation gestures.

3. *Game Server:* a server that processes acceleration readings from the game controller in real time to output the RPS value and rotation direction. It also has the feature to plot acceleration readings in real time, to show the performance of the curve-fitting function.

4. *ANU Survey:* a survey that highlights the phenomenon of stock-flow failure, even among people with technical backgrounds. In particular, a tendency to identify flows as stocks, termed as ‘stock bias’, was observed.

5. *Decision Tree:* a map detailing key hardware components and software features of the system, and a decision path that culminated in the final prototype. This tool provides a case study on agile development principles applied to complete a project with hardware and software requirements.

8.2 Lessons Learned

Agile development principles proved to be beneficial in the project due to the unfamiliar problem and solution domains, and small project size. In particular, the Pareto principle was utilised for rapid delivery of a proof of concept at each configuration in the decision tree.

Multiple pathways have been explored in the process of finding a solution that meets the requirements. In the future, the use of Agile should be combined with a configuration management practice. The lack of documentation due to rapid build and test phases made it difficult at times to track design decisions. Furthermore, testing occurred on an ad-hoc basis, which presents a risk of unnecessary rework for incoming honours students. It is strongly recommended that a journal of design decisions be kept, in addition to using a version control system for the software.

8.3 Future work

This section has been split into short-term, and medium-term recommendations. Short-term recommendations have a timeframe of less than one year, with increasing uncertainty as the project progresses toward the release of a beta prototype.

8.3.1 Short-Term

The most urgent recommendation is to verify and validate the rotation detection algorithm. The key challenge for real time interaction is that the frequency autocorrelation function requires at least two periods to detect RPS of a signal. Hence, there will always be a time delay between the rotation gesture, and its visualisation on the game due to the analysis. Rolling window sampling reduces this delay to 150ms, but because data in the historical buffer is not weighted to the most recent data received, new acceleration readings will not have a large effect on the output RPS value. The remaining methods for fundamental frequency estimation should be tested, in particular those based on the frequency domain, for more robust alternatives.

In terms of rotation direction detection, the algorithm returns unknown directions in some cases, even when a rotation direction is defined. This may also contribute to lag in the game, because the game has been programmed so that unknown directions do not affect the stock. Areas to test are the frequency autocorrelation function, and SciPy curve-fitting function, since fitted acceleration readings are used to detect rotation direction. Analysing the real time graph output of the server, and matching behaviour that produces an unknown rotation direction should also be considered to identify potential sources of error.

Improvements to the game can be conducted mostly independently of the controller and the server. Features to be included are the incorporation of time delay, feedback, and nonlinearities. A virtual ‘speedometer’ can be created on the game screen to demonstrate the current RPS value for better user feedback. To test the game without the game controller, input can be received through the keyboard or mouse.

The controller hardware can be miniaturised using a small Arduino board with integrated Bluetooth communication. Furthermore, the 9V battery adds inefficiencies to size and power in the current prototype. Small LiPo batteries can be used with a LiPo StepUp or Boost converter board to reduce size and increase power efficiency. Additionally, sensory feedback through lights, sound, or vibration can be incorporated to the controller to create a more interactive feedback system.

8.3.2 Medium-Term

User experience testing is required to ensure that the system works as intended and is intuitive. The simulation should work so that users will not lack procedural knowledge when interacting with the system. Therefore, hypothesis testing should be completed after user experience testing, so that user interface issues do not contribute to error in the experiment.

In the medium-term, an important functionality for the game is to create a multiplayer system that can be used in a classroom context. This would fulfil its purpose as an educational tool for eventual distribution to primary and secondary schools for the promotion of system dynamics.

Bibliography

ANALOG DEVICES 2009. The Five Motion Senses: Using MEMS Inertial Sensing to Transform Applications.

ANG, W. T., KHOO, S. Y., KHOSLA, P. K. & RIVIERE, C. N. 2004. Physical model of a MEMS accelerometer for low-g motion tracking applications. *IEEE International Conference on Robotics and Automation,* vol. 2**,** 1345-1351.

ARONSON, D. 1996. Overview of Systems Thinking.

ASPENCORE. 2016. *Phase Difference and Phase Shift* [Online]. Available: http://www.electronics-tutorials.ws/accircuits/phase-difference.html [Accessed 2 November 2016].

BALAJI, S. & SUNDARARAJAN MURUGAIYAN, M. 2012. Waterfall Vs V-Model Vs Agile: A Comparative Study on SDLC. *International Journal of Information Technology and Business Management,* vol. 2**,** 26-31.

BELL, R. 2015. *A Beginner's Guide to Big O Notation* [Online]. Available: https://rob-bell.net/2009/06/a-beginners-guide-to-big-o-notation/ [Accessed 31 October 2016].

BLANCHARD, B. S. & FABRYCKY, W. J. 1998. *Systems Engineering And Analysis 3rd Ed,* United States of America, Prentice Hall.

BOGASON, O. 2015. *Fundamental Frequency Estimation and Supervised Learning* [Online]. Available: http://obogason.com/fundamental-frequency-estimation-and-machine-learning/ [Accessed 31 October 2016].

BROWNE, C. 2013. Building better mental models with physical analogues. *Digital poster presented at the First Global Conference on Research Integration and Implementation, September 8-11, 2013.* Canberra, ACT.

BRUNSTEIN, A., GONZALEZ, C. & KANTER, S. 2010. Effects of domain experience in the stock-flow failure. *System Dynamics Review,* vol. 26**,** 347-354.

COHEN, D. 2005. *Precalculus: With Unit Circle Trigonometry*, Cengage Learning.

CRONIN, M. A., GONZALEZ, C. & STERMAN, J. D. 2009. Why don't well-educated adults understand accumulation? A challenge to researchers, educators, and citizens. *Organisational Behaviour and Human Decision Processes,* vol. 108**,** 116-130.

DADAFSHAR, M. 2015. Accelerometer and Gyroscopes Sensors: Operation, Sensing, and Applications. *Application Note 5830.* Maxim Integrated.

DELOITTE UNIVERSITY PRESS 2016. Tech Trends: Innovating in the digital era.

DIMENSION ENGINEERING LLC. 2016. *A beginner's guide to accelerometers* [Online]. Available: http://www.dimensionengineering.com/info/accelerometers [Accessed 16 October 2016].

DUTT, V. & GONZALEZ, C. 2007. Slope of Inflow Impacts Dynamic Decision Making. *Proceedings of the 25th International Conference of the System Dynamics Society.* Boston, Massachusetts.

ENDOLITH. 2016. *Frequency Estimation Methods in Python* [Online]. Available: https://gist.github.com/endolith/255291 [Accessed 31 October 2016].

FEATHERSTON, C. R. & DOOLAN, M. 2012. A Critical Review of the Criticisms of System Dynamics. *The 30th International Conference of the System Dynamics Society.* St Gallen, Switzerland.

FISCHER, H., DEGEN, C. & FUNKE, J. 2015. Improving Stock-Flow Reasoning With Verbal Formats. *System Dynamics Simulation & Gaming,* vol. 46**,** 255-269.

FORRESTER, J. W. 2007. System Dynamics - A Personal View of the First Fifty Years. *System Dynamics Review,* vol. 23**,** 345-358.

FREESCALE SEMICONDUCTOR 2010. 1.5g, 6g Three Axis Low-g Micromechanical Accelerometer. *Document Number MMA7361LC.* Freescale Semiconductor.

FREETRONICS. 2016a. *3-Axis Accelerometer Module for Arduino* [Online]. Available: http://www.freetronics.com.au/products/3-axis-accelerometer-module - .WASy19yqz8d [Accessed 17 October 2016].

FREETRONICS. 2016b. *AM3X Quickstart Guide* [Online]. Available: http://www.freetronics.com.au/pages/am3x-quickstart-guide - .V_yjadyqz8c [Accessed 11 October 2016].

FREETRONICS. 2016c. *Eleven (100% Arduino Uno Compatible)* [Online]. Available: http://www.freetronics.com.au/products/eleven - .WAN46dyqz8c [Accessed 17 October 2016].

FREETRONICS. 2016d. *LeoStick (Arduino Compatible)* [Online]. Available: http://www.freetronics.com.au/products/leostick - .WASlHNyqz8c [Accessed 17 October 2016].

GERHARD, D. 2003. Pitch Extraction and Fundamental Frequency: History and Current Techniques. *Technical Report TR-CS 2003-06.* University of Regina Department of Computer Science.

GIORDANO, N. 2009. *College Physics: Reasoning and Relationships*, Cengage Learning.

GONZALEZ, C. & DUTT, V. 2011. A generic dynamic control task for behavioral research and education. *Computers in Human Behaviour,* vol. 27**,** 1904-1914.

GRAHAM, B. B. 2000. *Using an Accelerometer Sensor to Measure Human Hand Motion.* BS & MNG Electrical Engineering and Computer Science Master Thesis, Massachusetts Institute of Technology.

HESPENHEIDE, J. 2016. *Iterative Design and Creating on a Small Budget* [Online]. Available: http://www.julian-h.de/sketch/ [Accessed 19 October 2016].

HOEFFNER, J. & SHAH, P. 2002. Review of Graph Comprehension Research: Implications for Instruction. *Educational Psychology Review,* vol. 14.

ISTQB. 2016a. *What is Spiral model- advantages, disadvantages and when to use it?* [Online]. Available: http://istqbexamcertification.com/what-is-spiral-model-advantages-disadvantages-and-when-to-use-it/ [Accessed 11 November 2016].

ISTQB. 2016b. *What is V-model- advantages, disadvantages and when to use it?* [Online]. Available: http://istqbexamcertification.com/what-is-v-model-advantages-disadvantages-and-when-to-use-it/ [Accessed 11 November 2016].

JEZNY, J. & CURILLA, M. 2013. Position Measurement with Hall Effect Sensors. *American Journal of Mechanical Engineering,* vol. 1**,** 231-235.

KOKS, D. 2008. Using Rotations to Build Aerospace Coordinate Systems. *DSTO-TN-0640.* DSTO Systems Sciences Laboratory.

MEADOWS, D. 2009. *Thinking in Systems*, Earthscan.

MELLIS, D., BANZI, M., CUARTIELLES, D. & IGOE, T. 2007. Arduino: An Open Electronics Prototyping Platform.

MISLEVY, R. J. 2013. Evidence-Centered Design for Simulation-Based Assessment. *Military Medicine,* vol. 178**,** 107-114.

MIT MEDIA LAB. 2013. *Kids coding in the cloud* [Online]. Available: http://news.mit.edu/2013/scratch-two-released-0514 [Accessed 11 November 2016].

MITCHESON, P. D., YEATMAN, E. M., RAO, G. K., HOLMES, A. S. & GREEN, T. C. 2008. Energy Harvesting From Human and Machine Motion for Wireless Electronic Devices. *IEEE Proceedings,* vol. 96**,** 1457-1486.

MUMENTHALER, M. & MATTLE, H. 2006. *Fundamentals of Neurology: An Illustrated Guide*, Thieme.

NEWELL, B. 2012. Simple models, powerful ideas: Towards effective integrative practice. *Global Environmental Change,* vol. 22**,** 776-783.

NIELSEN, J. & MOLICH, R. 1990. Heuristic evaluation of user interfaces. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, April 1-5, 1990.* Seattle, Washington.

NIST SEMATECH 2013. e-Handbook of Statistical Methods. *Engineering Statistics Handbook.*

PAPAGEORGIOU, G. & ABROSIMOVA, K. 2007. An Overview of System Dynamics Methods for Developing Management Flight Simulators. *6th WSEAS Int. Conference on Computational Intelligence, Man-Machine Systems and Cybernetics, December 14-15, 2007.* Tenerige, Spain.

RABINER, L. R. 1977. On the Use of Autocorrelation Analysis for Pitch Detection. *IEEE Transactions on Acoustics, Speech, and Signal Processing,* vol. ASSP-25**,** 24-33.

RICHARDSON, G. P. 2011. Reflections on the foundations of system dynamics. *System Dynamics Review,* vol. 27**,** 219-243.

ROBEL, A. 2006. Fundamental Frequency Estimation. *Summer 25 August 2006, Lecture on Analysis, Modeling, and Transformation of Audio Signals.* Institute of Communication Science TU-Berlin IRCAM Analysis/Synthesis Team.

RUPARELIA, N. B. 2010. Software Development Lifecycle Models. *ACM SIFSOFT Software Engineering Notes,* vol. 35**,** 8-13.

SCIPY.ORG. 2016. *scipy.optimise.curve_fit* [Online]. Available: https://docs.scipy.org/doc/scipy-0.18.1/reference/generated/scipy.optimize.curve_fit.html [Accessed 2 November 2016].

SERWAY, R., FAUGHN, J. & VUILLE, C. 2008. *College Physics Volume 10*, Cengage Learning.

SPARKFUN ELECTRONICS. 2016a. *Accelerometer Basics* [Online]. Available: https://learn.sparkfun.com/tutorials/accelerometer-basics [Accessed 12 October 2016].

SPARKFUN ELECTRONICS. 2016b. *Accelerometer, Gyro and IMU Buying Guide* [Online]. Available: https://www.sparkfun.com/pages/accel_gyro_guide [Accessed 15 October 2016].

SPARKFUN ELECTRONICS. 2016c. *Bluetooth Basics* [Online]. Available: https://learn.sparkfun.com/tutorials/bluetooth-basics/how-bluetooth-works [Accessed].

STERMAN, J. D. 2002. System Dynamics: Systems Thinking and Modeling for a Complex World. *Working Paper Series.* MIT, Engineering Systems Division.

STERMAN, J. D. 2006. *Management Flight Simulator* [Online]. Available: http://www.systemdynamics.org/DL-IntroSysDyn/flsim.htm [Accessed 11 November 2016].

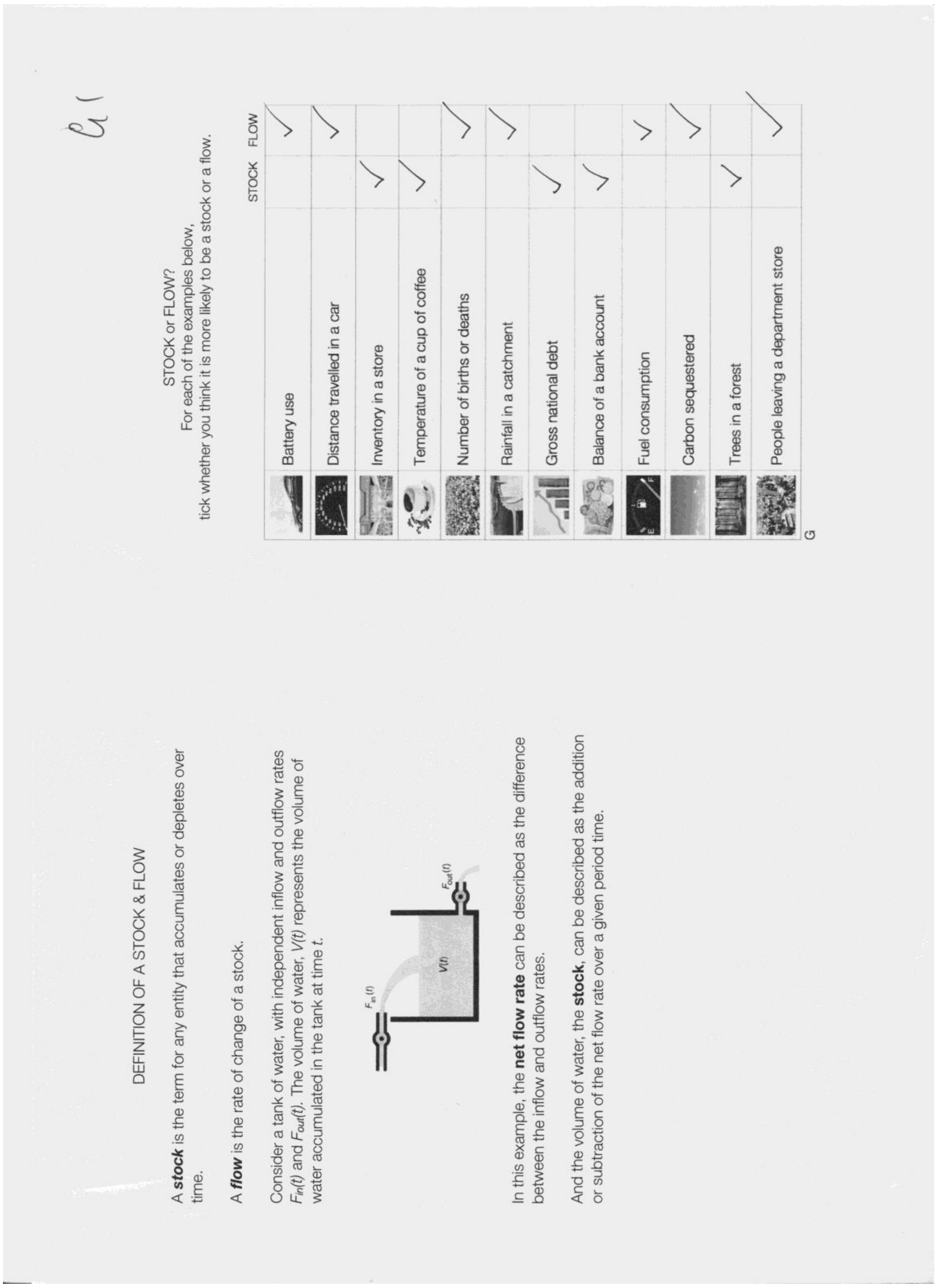
THE OPEN UNIVERSITY 1994. *Flexible Learning Approach to Physics Module P5.1 Simple harmonic motion*, The Open University.

WALDRON, L. 2015. Project Management, PowerPoint Presentation. ENGN3221: Engineering Management, The Australian National University.

XIN, L., HUANG, E. & MOK, S. 2011. HC-06 Product Data Sheet.

Appendices

Appendix 2A – Survey Questions



DEFINITION OF A STOCK & FLOW

A ***stock*** is the term for any entity that accumulates or depletes over time.

A ***flow*** is the rate of change of a stock.

Consider a tank of water, with independent inflow and outflow rates $F_{in}(t)$ and $F_{out}(t)$. The volume of water, $V(t)$ represents the volume of water accumulated in the tank at time t.

$F_{in}(t)$
$V(t)$
$F_{out}(t)$

In this example, the **net flow rate** can be described as the difference between the inflow and outflow rates.

And the volume of water, the **stock**, can be described as the addition or subtraction of the net flow rate over a given period time.

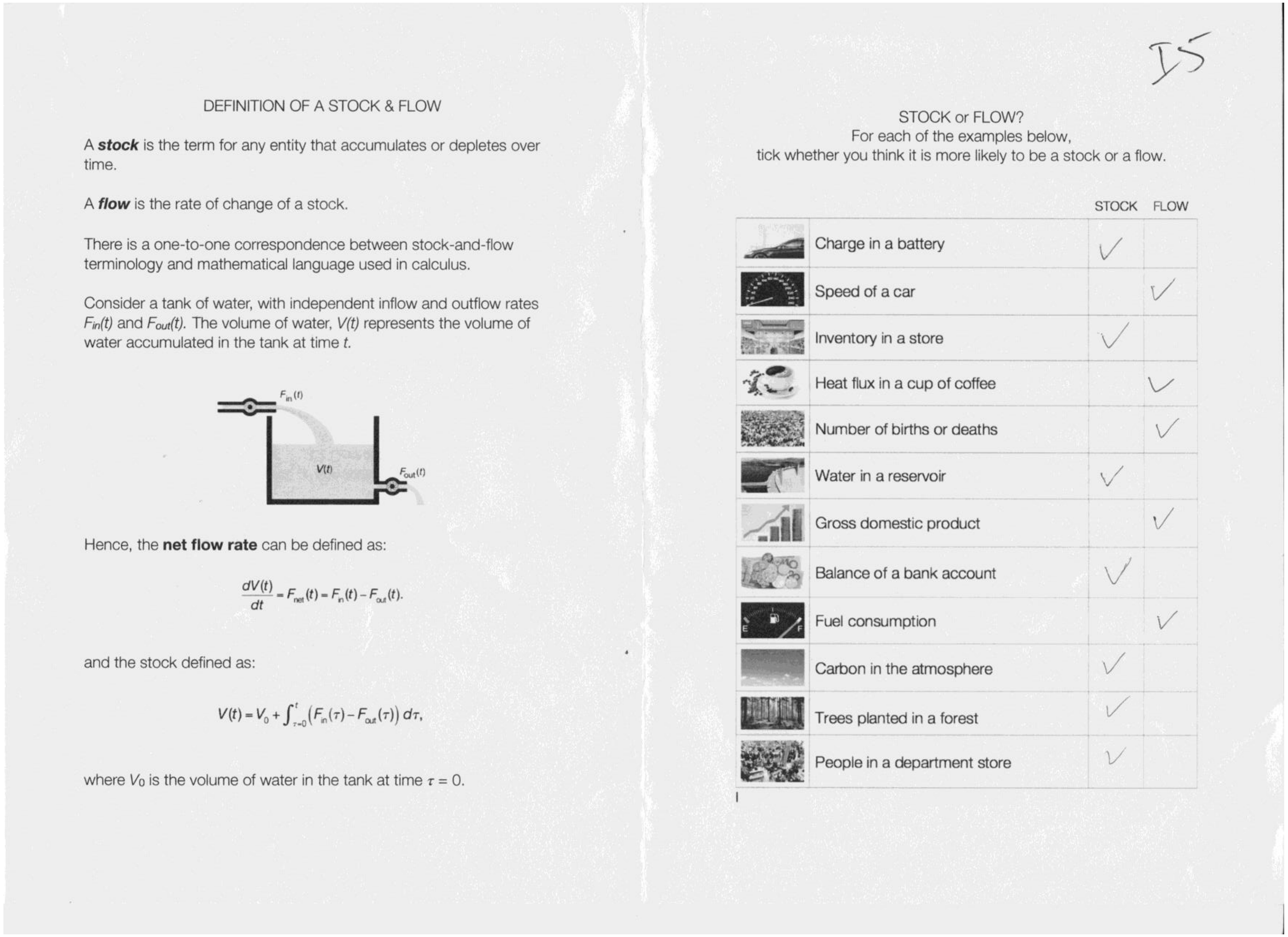
G1

STOCK or FLOW?
For each of the examples below,
tick whether you think it is more likely to be a stock or a flow.

	STOCK	FLOW
Battery use		✓
Distance travelled in a car		✓
Inventory in a store	✓	
Temperature of a cup of coffee	✓	
Number of births or deaths		✓
Rainfall in a catchment		✓
Gross national debt	✓	
Balance of a bank account	✓	
Fuel consumption		✓
Carbon sequestered		✓
Trees in a forest	✓	
People leaving a department store		✓

G

Figure 49. Survey variation with verbal explanation of stocks and flows



I5

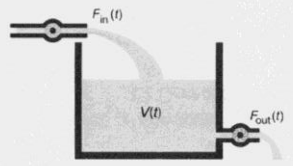
DEFINITION OF A STOCK & FLOW

A ***stock*** is the term for any entity that accumulates or depletes over time.

A ***flow*** is the rate of change of a stock.

There is a one-to-one correspondence between stock-and-flow terminology and mathematical language used in calculus.

Consider a tank of water, with independent inflow and outflow rates *$F_{in}(t)$* and *$F_{out}(t)$*. The volume of water, *$V(t)$* represents the volume of water accumulated in the tank at time *t*.



Hence, the **net flow rate** can be defined as:

$$\frac{dV(t)}{dt} = F_{net}(t) = F_{in}(t) - F_{out}(t).$$

and the stock defined as:

$$V(t) = V_0 + \int_{\tau=0}^{t} \left(F_{in}(\tau) - F_{out}(\tau)\right) d\tau,$$

where V_0 is the volume of water in the tank at time $\tau = 0$.

STOCK or FLOW?
For each of the examples below,
tick whether you think it is more likely to be a stock or a flow.

		STOCK	FLOW
	Charge in a battery	✓	
	Speed of a car		✓
	Inventory in a store	✓	
	Heat flux in a cup of coffee		✓
	Number of births or deaths		✓
	Water in a reservoir	✓	
	Gross domestic product		✓
	Balance of a bank account	✓	
	Fuel consumption		✓
	Carbon in the atmosphere	✓	
	Trees planted in a forest	✓	
	People in a department store	✓	

Figure 50. Survey variation with mathematical explanation of stocks and flows

Table 14. Survey stock and flow questions

Question No.	S	F
1	Charge in a battery	Battery use
2	Distance travelled in a car	Speed of a car
3	Inventory in a store	Sales in a store
4	Temperature of a cup of coffee	Heat flux in a cup of coffee
5	Population of a country	Number of births or deaths
6	Water in a reservoir	Rainfall in a catchment
7	Gross national debt	Gross domestic product
8	Balance of a bank account	Interest in a bank account
9	Fuel in a tank	Fuel consumption
10	Carbon in the atmosphere	Carbon sequestered
11	Trees in a forest	Trees planted in a forest
12	People in a department store	People leaving a department store

Table 15. Survey demographic questions

Question No.	
1	Have you completed any course pre-reading?
2	Have you ever studied system dynamics before?
3	Have you completed any university-level calculus before?
4	Are you an international or domestic student?
5	Is this a required course or elective for your degree or major?
6	Are you an undergraduate or graduate student?

Appendix 2B – Significant Outcomes from Fisher's Exact Test

Table 16. Question 6 Stocks versus Flows

Q6	Stocks	Flows	Total	Fisher's Exact Test	
Correct	18	10	28	Two-sided p-value	0.0050
Incorrect	3	13	16		
Total	21	23	44		

Table 17. Question 8 Stocks versus Flows

Q8	Stocks	Flows	Total	Fisher's Exact Test	
Correct	20	12	32	Two-sided p-value	0.0421
Incorrect	3	9	12		
Total	23	21	44		

Table 18. Question 10 Stocks versus Flows

Q10	Stocks	Flows	Total	Fisher's Exact Test	
Correct	17	6	23	Two-sided p-value	0.0022
Incorrect	5	16	21		
Total	22	22	44		

Table 19. Question 11 Stocks versus Flows

Q11	Stocks	Flows	Total	Fisher's Exact Test	
Correct	19	1	20	Two-sided p-value	0.0000
Incorrect	3	21	24		
Total	22	22	44		

Appendix 4A – Summary of IC Datasheet (MMA7361LC)

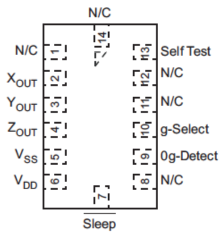


Figure 51. Top view of accelerometer with pinout descriptions (Freescale Semiconductor, 2010)

Table 20. Pin descriptions (Freescale Semiconductor, 2010)

Pin No.	Pin Name	Description
1	N/C	No internal connection Leave unconnected
2	X_{OUT}	X direction voltage output
3	Y_{OUT}	Y direction voltage output
4	Z_{OUT}	Z direction voltage output
5	V_{SS}	Power Supply Ground
6	V_{DD}	Power Supply Input
7	$\overline{\text{Sleep}}$	Logic input pin to enable product or Sleep Mode
8	N/C	No internal connection Leave unconnected
9	0g-Detect	Linear Free-fall digital logic output signal
10	g-Select	Logic input pin to select g level
11	N/C	Unused for factory trim Leave unconnected
12	N/C	Unused for factory trim Leave unconnected
13	Self Test	Input pin to initiate Self Test
14	N/C	Unused for factory trim Leave unconnected

Table 21. Maximum ratings (Freescale Semiconductor, 2010)

(Maximum ratings are the limits to which the device can be exposed without causing permanent damage.)

Rating	Symbol	Value	Unit
Maximum Acceleration (all axis)	g_{max}	$\pm$5000	g
Supply Voltage	V_{DD}	-0.3 to +3.6	V
Drop Test$^{(1)}$	D_{drop}	1.8	m
Storage Temperature Range	T_{stg}	-40 to +125	°C

1. Dropped onto concrete surface from any axis.

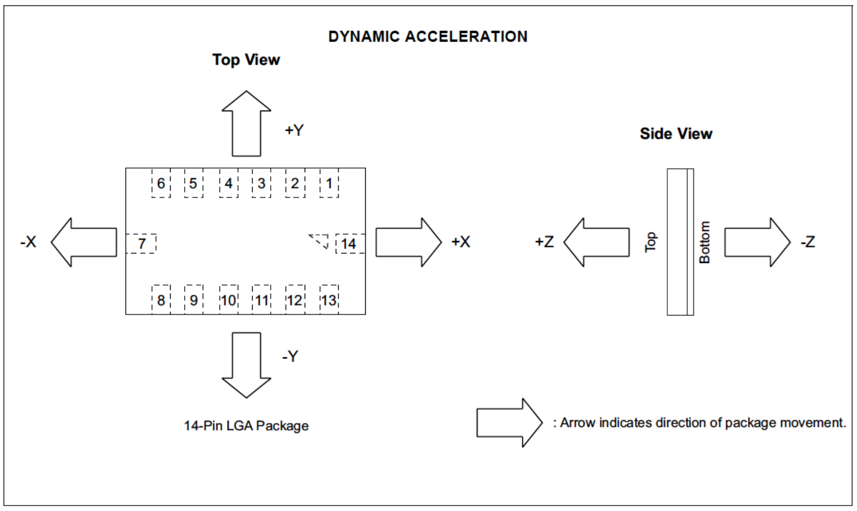


Figure 52. Positive and negative direction of package movement in the X, Y, and Z-axis (Freescale Semiconductor, 2010).

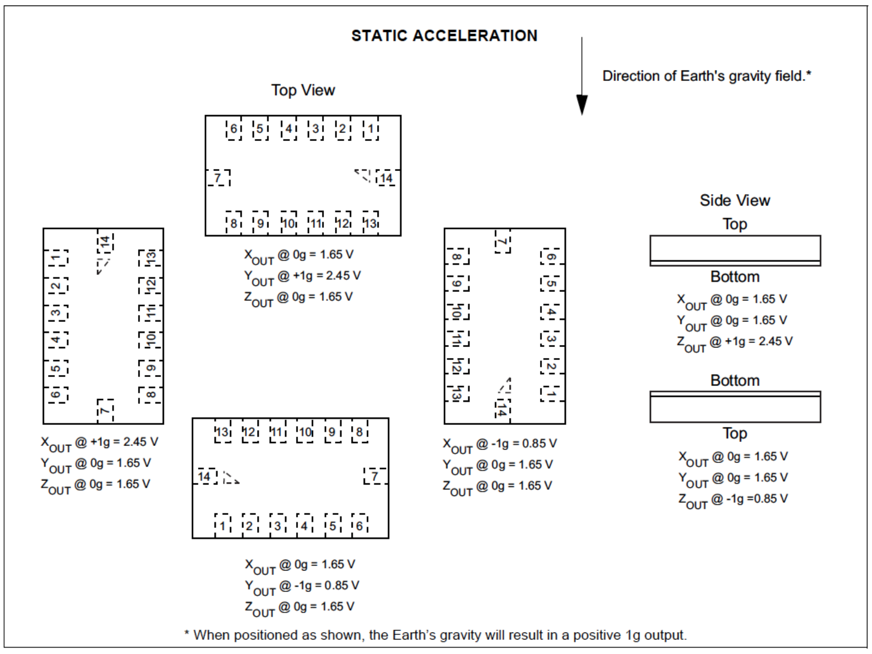


Figure 53. Voltage outputs when the accelerometer is stationary at various positions (Freescale Semiconductor, 2010).

Appendix 4B – Hardware Miniaturisation

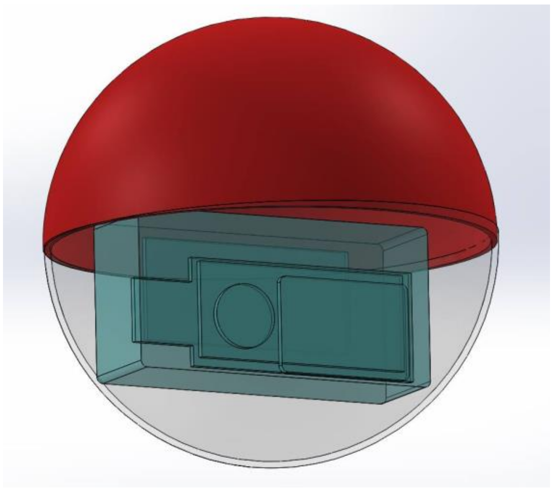


Figure 54. CAD Concept Spherical Case (70mm diameter)

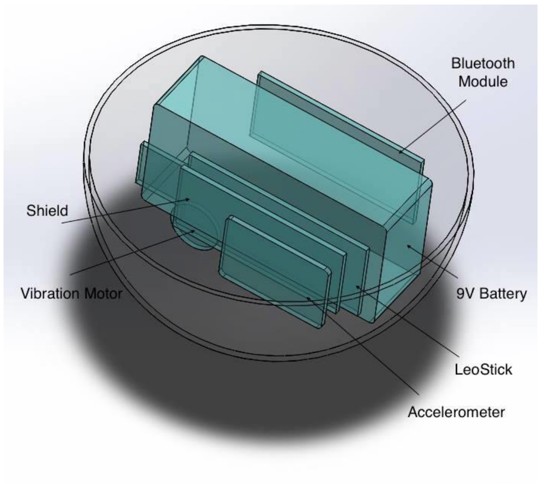


Figure 55. CAD Concept Spherical Case (70mm diameter)

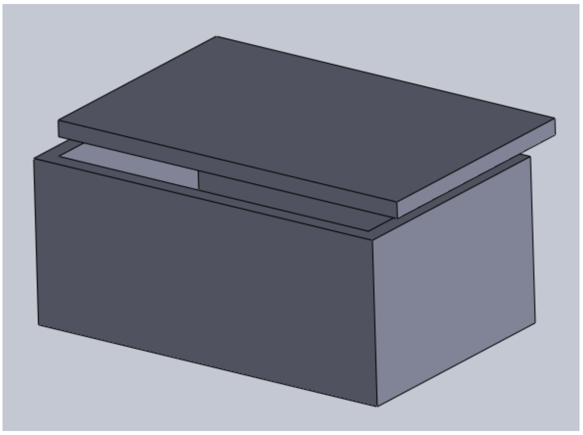


Figure 56. CAD Concept Rectangular Case (7cm × 3cm × 4.5cm)

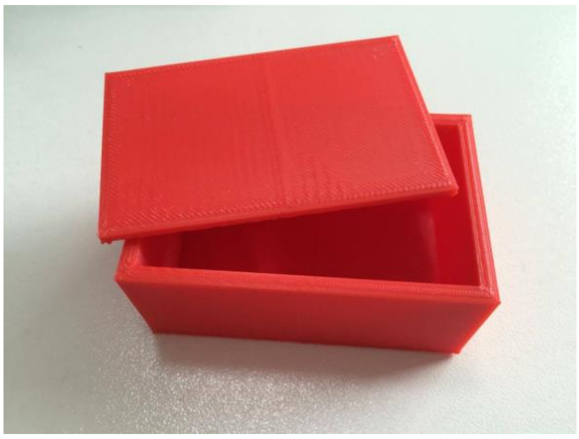


Figure 57. 3D Printed Rectangular Case (7cm × 3cm × 4.5cm)

Appendix 4C – Unused Prototypes

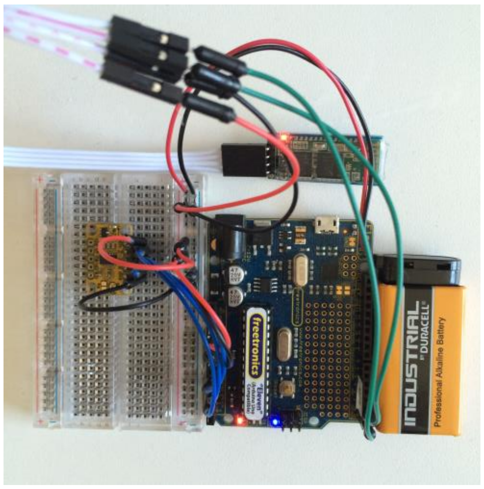


Figure 58. Freetronics Eleven V0.2 connected wirelessly

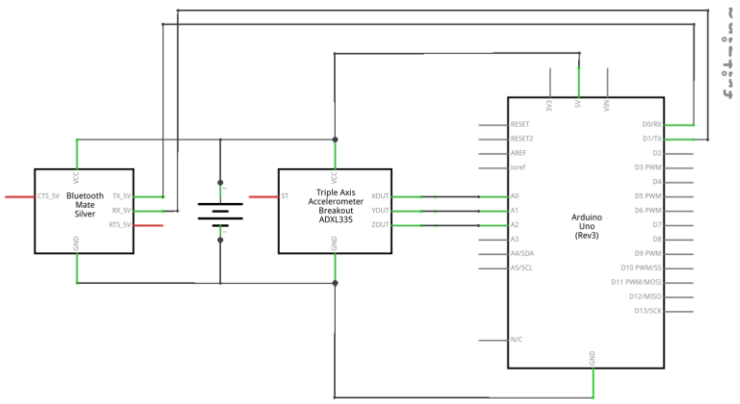


Figure 59.Circuit diagram for V0.2

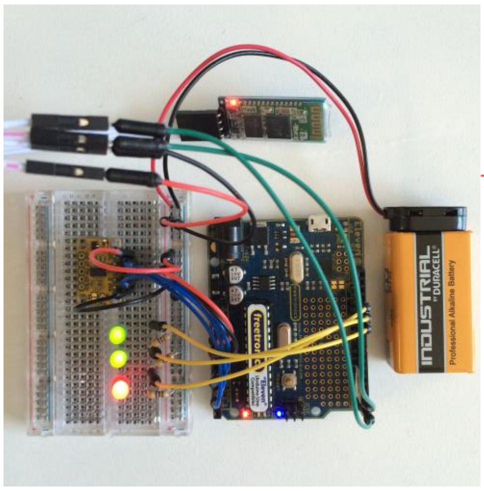


Figure 60. Freetronics Eleven V0.3 connected wirelessly with visual feedback

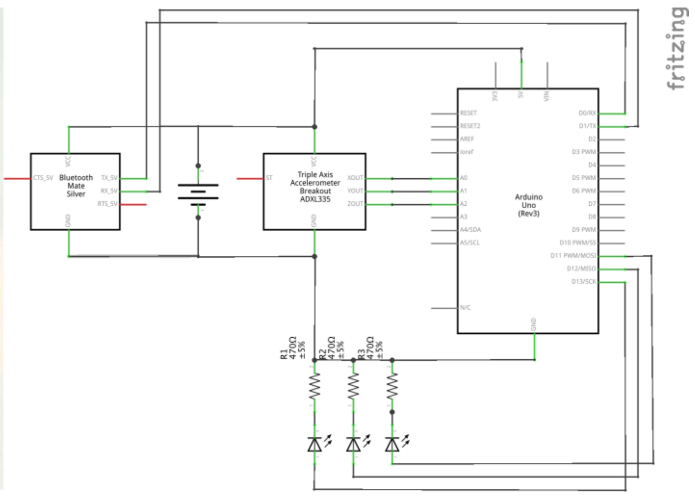


Figure 61.Circuit diagram for V0.3

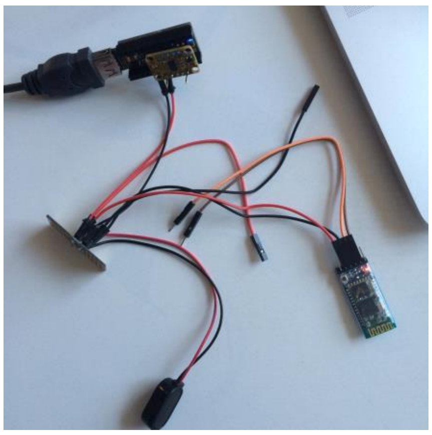


Figure 62. Freetronics LeoStick V0.4 connected by USB

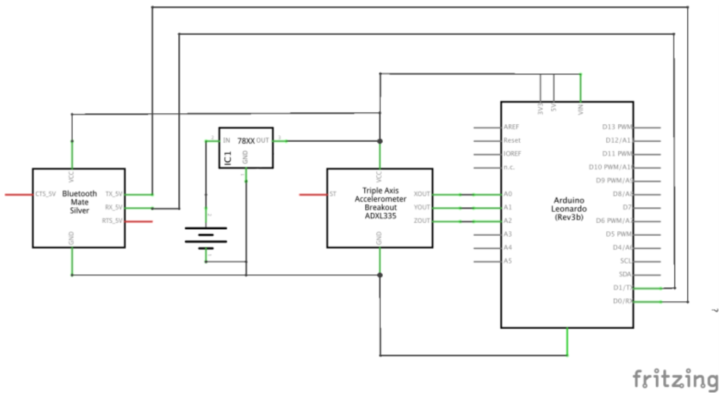


Figure 63.Circuit diagram for V0.4

Appendix 5A – Procedure for Data Collection

This procedure describes how data is collected from the accelerometer for analysis. The prototype used to collect raw data is shown below.

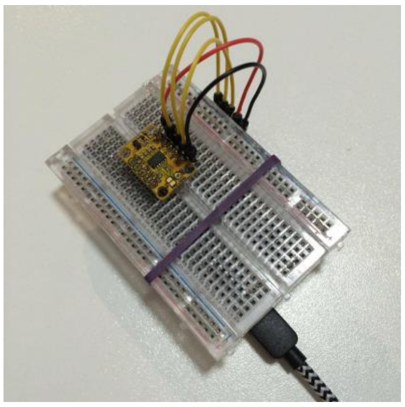


Figure 64. Prototype 1 connection.

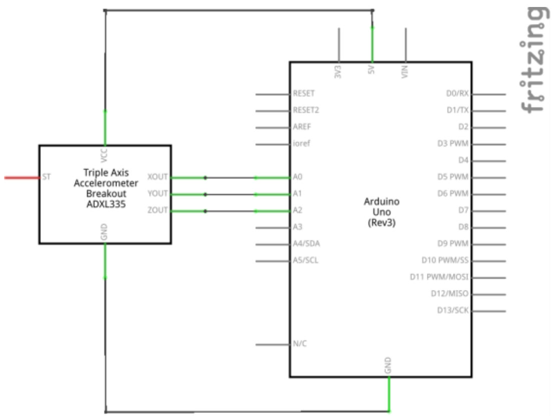


Figure 65. Prototype 1 schematic.

Step 1: Connect the accelerometer to the microcontroller. The accelerometer range is set to $\pm$1.5g by default. It is configured properly when the X, Y, and Z-axis outputs are in the range of 338, 338, and 501 respectively. See Figure 64 and Figure 65 for set up and connection to the microcontroller.

- Note: Figure 65 displays an Arduino Uno and ADXL 335, not a Freetronics Eleven and AM3X. However, the connections for the two devices are the same. The application used to draw the circuit diagram, 'Fritzing', does not have schematics for the components used in this project.

Step 2: Connect the microcontroller to a computer using a 1.5m micro USB cable. The cable length should be greater than 1m to allow for sufficient arm movement.

Step 3: Open the Arduino IDE and upload the script read_acceleration.ino to the microcontroller. See Script 3 in Appendix 5C – Code for Data Collection. This script reads and converts analogue outputs of the accelerometer to digital outputs.

Step 4: Print a 25cm diameter circle onto an A3 page. Attach to a wall to define the rotation diameter. Mark an arbitrary starting point on the circle.

Step 5: Set an online metronome to 40 beats per minute, which is equivalent to 40RPM.

Step 6: Start at the mark on the circle and practice rotating the accelerometer in an anticlockwise direction around the Y-axis so that one revolution matches to the beat of the metronome.

Step 7: When ready, open the terminal and run the script get_data_based_on_iterations.py. See Script 4 in Appendix 5C – Code for Data Collection. This script writes the acceleration readings onto a csv file for a specified number of data points. Choose 1000 data points for the computer to record. This will take approximately $2000 \times\ 30ms = 60\ seconds = 1\ minute$. 30ms is the fastest time the Freetronics Eleven reads data from the accelerometer.

Step 8: Continue rotating the accelerometer in the same orientation until the program terminates. Acceleration readings are stored in the specified csv file for analysis.

Step 9: Repeat for clockwise rotation. Disconnect the microcontroller once complete.

Appendix 5B – Code for Matlab Graphing

Script 1 rotation_study.m

This script is for Matlab. It creates the sinusoidal functions for x and y acceleration components at f=0.67Hz and A=0.125cm.

```
t = 0:0.1:10;
x = -4*(pi^2)*(0.67^2)*0.125*cos(2*pi*0.67*t);
% activate formula for anticlockwise rotation;
% y = -4*(pi^2)*(0.67^2)*0.125*sin(2*pi*0.67*t);
% activate formula for clockwise rotation;
y = 4*(pi^2)*(0.67^2)*0.125*sin(2*pi*0.67*t);

c1 = [242 220 48] ./ 255;
c2 = [169 199 58] ./ 255;
c3 = [14 180 24]  ./ 255;
c4 = [53 155 112] ./ 255;
c5 = [117 128 72] ./ 255;
c6 = [92 53 19]   ./ 255;
c7 = [6 16 8]     ./ 255;

t1 = t(1:15);
t2 = t(16:30);
t3 = t(31:45);
t4 = t(46:60);
t5 = t(61:75);
t6 = t(76:90);
t7 = t(91:101);

x1 = x(1:15);
x2 = x(16:30);
x3 = x(31:45);
x4 = x(46:60);
x5 = x(61:75);
x6 = x(76:90);
x7 = x(91:101);

y1 = y(1:15);
y2 = y(16:30);
y3 = y(31:45);
y4 = y(46:60);
y5 = y(61:75);
y6 = y(76:90);
y7 = y(91:101);

hold on;
box on;
% activate plots below to graph x components and y components versus time
% grid on;
% plot(t,x);
% plot(t,y);

% activate plots below to graph x components versus y components
% plot(x1,y1,'o','MarkerEdgeColor',c1);
% plot(x2,y2,'o','MarkerEdgeColor',c2);
% plot(x3,y3,'o','MarkerEdgeColor',c3);
% plot(x4,y4,'o','MarkerEdgeColor',c4);
% plot(x5,y5,'o','MarkerEdgeColor',c5);
% plot(x6,y6,'o','MarkerEdgeColor',c6);
% plot(x7,y7,'o','MarkerEdgeColor',c7);

% activate plots below to graph x components vs y components vs time
figure
```

```
plot3(x,y,t);
```

Script 2 acceleration_raw_data_plots.m

This script is for Matlab. It converts digital outputs to linear acceleration (m/s^2). Import the 'Time', 'X', 'Y', and 'Z' columns of the csv file produced in **Script 4. get_data_based_on_iterations.py**.

```
digital_x = X;
digital_y = Y;
digital_z = Z;
t = Time/1000;

voltage_0g = 1.65;
g = 9.8;

% x axis outputs
analogue_x = digital_x * (5/1023);
g_force_x = (analogue_x - 1.65)/0.8;
acceleration_x = g_force_x * g;

% y axis outputs
analogue_y = digital_y * (5/1023);
g_force_y = (analogue_y - 1.65)/0.8;
acceleration_y = g_force_y * g;

% z axis outputs
analogue_z = digital_z * (5/1023);
g_force_z = (analogue_z - 1.65)/0.8;
acceleration_z = g_force_z * g;

hold on;
box on;
% activate plots below to graph acceleration versus time
% plot(t,acceleration_x);
% plot(t,acceleration_y);
% plot(t,acceleration_z);

% activate plots below to graph x vs y vs z acceleration
% plot(acceleration_x,acceleration_y,'o');
% plot(acceleration_x,acceleration_z,'o');
% plot(acceleration_y,acceleration_z,'o');

% activate figure and plots below to graph two components vs time;
% figure
% plot3(acceleration_x,acceleration_y,t);
% plot3(acceleration_x.acceleration_z,t);
% plot3(acceleration_y,acceleration_z,t);
```

Appendix 5C – Code for Data Collection

Script 3. read_acceleration.ino

This script is for the Arduino IDE. Upload onto the Freetronics Eleven microcontroller with connection to a 3-axis accelerometer. It reads X, Y, and Z-axis analogue inputs from the accelerometer and converts them to digital outputs.

```
int xAxis = A0;
int yAxis = A1;
int zAxis = A2;
int axisValue = 0;

unsigned long time;

void setup() {
  Serial.begin(9600);
  // block until the current transmit buffer is emptied (from the reader)
  Serial.flush();
}

void loop() {
  time = millis();
  Serial.print("S");
  Serial.print("Time=");
  Serial.print(time);
  axisValue = analogRead(xAxis);
  Serial.print(",X=");
  Serial.print(axisValue);
  axisValue = analogRead(yAxis);
  Serial.print(",Y=");
  Serial.print(axisValue);
  axisValue = analogRead(zAxis);
  Serial.print(",Z=");
  Serial.print(axisValue);
  Serial.println("E");

  delay(0);
}
```

Script 4. get_data_based_on_iterations.py

This script runs on Python 3. It takes data from the microcontroller and writes the output to a csv file according to the columns: 'Time', 'Delta Time', 'X', 'Y', and 'Z'. To execute from the command line run on the same directory: `python3 get_data_based_on_iterations.py filename.csv n` *where filename.csv is the csv file to write to, and n is the number of acceleration readings (e.g. 1000).*

```python
import serial, re, time
import csv
import sys

# 2 arguments, file to record into, and number of iterations to record
csv_file = open(sys.argv[1], 'w')
iterations_to_record = int(sys.argv[2])

acceleration_writer = csv.writer(csv_file, delimiter=',')
acceleration_writer.writerow(["Time", "Delta Time", "X", "Y", "Z"])

matching_regex = re.compile('Time.(\d+).X.(\d+).Y.(\d+).Z.(\d+)', re.I)

iterations_done = 0
prev_ms = 0

arduino = serial.Serial('/dev/cu.usbmodem1411', baudrate=9600, timeout=1) # connection via USB

first_time = True
while (iterations_done < iterations_to_record):

    # if the starting_byte is not S, discard it until we reach an S
    starting_byte = arduino.read(1)
    if starting_byte != b"S":
        continue

    # read a byte until we reach an E
    intermediate_byte = arduino.read(1)
    message_buffer = b""
    while (intermediate_byte != b"E"):
        message_buffer += intermediate_byte
        intermediate_byte = arduino.read(1)

    # now we have the message buffer
    # start from Time 'T'
    message_buffer = message_buffer.decode("utf-8")
    if message_buffer and message_buffer[0] == 'T': # start from Time

        coordinates = re.match(matching_regex, message_buffer) # parse only values

        if not bool(coordinates):
            continue

        time_ms = int(coordinates.group(1))
        x_accel = int(coordinates.group(2))
        y_accel = int(coordinates.group(3))
        z_accel = int(coordinates.group(4))

        if first_time:
            prev_ms = time_ms
            first_time = False

        delta_ms = time_ms - prev_ms

        prev_ms = time_ms

        acceleration_writer.writerow([time_ms, delta_ms, x_accel, y_accel, z_accel])
```

```python
            csv_file.flush()

            iterations_done += 1

        else:
            continue

arduino.close()
csv_file.close()
```

Script 5. bucket_data_into_time_windows.py

This script runs on Python 3. It takes the csv file produced in Script 4. get_data_based_on_iterations.py, *and partitions the data into 4 second time windows for output onto separate csv files. To execute from the command line run on the same directory:* `python3 bucket_data_into_time_windows.py filename.csv` *where filename.csv is the csv file to read.*

```python
# python bucket.py path-to-csv-to-read path-to-directory-to-hold-bucketed-csv-files

import csv
import sys

def read_csv_to_objects (csv_file_path):
    with open(csv_file_path, 'r') as csv_file:
        reader = csv.reader(csv_file, delimiter=',')
        header = next(reader)
        readings = []
        for row in reader:
            reading = {}
            for index, heading in enumerate(header):
                reading[heading] = row[index]
            readings.append(reading)
    return readings

acceleration_readings = read_csv_to_objects(sys.argv[1])
partition_directory = sys.argv[2]

# bucket the data in 4 second segments
# 4 seconds is going to be our time window
# 4 seconds is 2 * 2 seconds
# 2 seconds is the period for 30 revolutions per second

time_window_ms = 4000
time_interval_per_reading_ms = 30
data_points_per_window = time_window_ms // time_interval_per_reading_ms

# now we can bucket the data by data_points_per_window

file_index = 0
partitioned_csv = open(partition_directory + '/' + str(file_index) + '.csv', 'w')
partition_writer = csv.writer(partitioned_csv, delimiter=',')
for i, row in enumerate(acceleration_readings):
    partition_writer.writerow([row['Time'], row['X'], row['Y'], row['Z']])
    if ((i + 1) % data_points_per_window == 0):
        file_index = file_index + 1
        partitioned_csv.close()
        partitioned_csv = open(partition_directory + '/' + str(file_index) + '.csv', 'w')
        partition_writer = csv.writer(partitioned_csv, delimiter=',')

partitioned_csv.close()
```

Appendix 5D – Code for Fundamental Frequency Estimation and Curve-Fitting

Script 6. static_curve_fitting-zcr.py

This script runs on Python 3. It takes the csv file produced in **Script 5. bucket_data_into_time_windows.py**, *and estimates the frequency of the signal using zero-crossing rate. The frequency is used in the curve-fitting function to perform a regression over the acceleration readings. To execute from the command line run on the same directory:* `python3 static_curve_fitting_zcr.py`. *The program will ask for the csv filename to read.*

```
# following instructions from http://www2.mpia-hd.mpg.de/~robitaille/PY4SCI_SS_2014/_static/15.%20Fitting%20models%20to%20data.html

import csv
import sys
import math as m
from matplotlib.mlab import find
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit

your_csv_file = input('Enter a csv filename: ')

def read_csv_to_table_without_header (csv_file_path):
    with open(csv_file_path, 'r') as csv_file:
        reader = csv.reader(csv_file, delimiter=',')
        readings = []
        for row in reader:
            readings.append(row)
    return readings

table = read_csv_to_table_without_header(your_csv_file)

# assume 30 ms increments between data points
# for a 4s time window, there will be a maximum of 133 data points
acceleration_time_values = np.arange(0, 3.99, 0.03)

# table data is in this format:
# [
#   [T, X, Y, Z],
#   [T, X, Y, Z],
#   ...
# ]

digital_acceleration_x_values = np.array([int(row[1]) for row in table])
digital_acceleration_y_values = np.array([int(row[2]) for row in table])
digital_acceleration_z_values = np.array([int(row[3]) for row in table])

analogue_acceleration_x_values = (digital_acceleration_x_values * 5/1023)
g_force_x_values = (analogue_acceleration_x_values - 1.65)/0.8
acceleration_x_values = g_force_x_values * 9.8

analogue_acceleration_y_values = (digital_acceleration_y_values * 5/1023)
g_force_y_values = (analogue_acceleration_y_values - 1.65)/0.8
acceleration_y_values = g_force_y_values * 9.8

analogue_acceleration_z_values = (digital_acceleration_z_values * 5/1023)
g_force_z_values = (analogue_acceleration_z_values - 1.65)/0.8
acceleration_z_values = g_force_z_values * 9.8

# normalise it to 0 mean
normalised_acceleration_x_values = acceleration_x_values - np.mean(acceleration_x_values)
normalised_acceleration_y_values = acceleration_y_values -
```

```
    np.mean(acceleration_y_values)
normalised_acceleration_z_values = acceleration_z_values - 
    np.mean(acceleration_z_values)

# estimate the frequency for curve fitting
def freq_from_ZCR(sig, fs):
    indices = find((sig[1:] >= 0) & (sig[:-1] < 0))
    print("Rising Crossing Indices: " + str(indices))
    crossings = [i - sig[i] / (sig[i+1] - sig[i]) for i in indices]
    return (indices, fs / np.mean(np.diff(crossings)))

indices, f = freq_from_ZCR(normalised_acceleration_x_values, 33.33)
angular_f = f * 2*np.pi

marked_time = [acceleration_time_values[i] for i in indices]
marked_indices = [normalised_acceleration_x_values[i] for i in indices]

plt.plot(marked_time, marked_indices, 'b*', markersize=14)

def func(acceleration_time_values, a, b, c):
    return a*np.sin(angular_f*acceleration_time_values + b) + c

popt, pcov = curve_fit(func, acceleration_time_values, normalised_acceleration_x_values
)

a = popt[0]
b = popt[1]
c = popt[2]

print("Amplitude: " + str(a))
print("Frequency: " + str(f))
print("Phase Shift: " + str(b))
print("Vertical Shift: " + str(c))

print('Curve fitting formula: Acceleration(t) (m/s^2) = ' + str(a) + 'sin(' + str(angul
ar_f) + 't + ' + str(b) + ')' + ' + ' + str(c))

plt.plot(acceleration_time_values, acceleration_time_values * 0, 'g-')
plt.plot(acceleration_time_values, normalised_acceleration_x_values, 'r.')
plt.plot(acceleration_time_values, func(acceleration_time_values, a, b, c), 'r')
plt.xlabel("Time (s)")
plt.ylabel("Acceleration (m/s^2)")
plt.title("Zero Crossing Rate Method for RPS Estimation")
plt.show()
```

Script 7. static_curve_fitting-ac.py

This script runs on Python 3. It takes the csv file produced in **Script 5. bucket_data_into_time_windows.py**, *and estimates the frequency of the signal using autocorrelation. The frequency is used in the curve-fitting function to perform a regression over the acceleration readings. To execute from the command line run on the same directory:* `python3 static_curve_fitting_ac.py`. *The program will ask for a csv filename and the time delay between each data point, which is approximately 30ms for Freetronics Eleven connected to a MacBook Pro 12 GHz Intel Core i7.*

```
# curve fitting algorithm from http://www2.mpia-hd.mpg.de/~robitaille/PY4SCI_SS_2014/_static/15.%20Fitting%20models%20to%20data.html
# frequency estimation algorithm from https://gist.github.com/endolith/255291
# parabolic function algorithm from https://github.com/endolith/waveform-analyzer/blob/master/common.py

import csv
import sys
import math as m
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.mlab import find
from scipy.optimize import curve_fit
from scipy.signal import fftconvolve

your_csv_file = input('Enter a csv filename: ')
delta_t = int(input('Enter time delay (in milliseconds): '))/1000

def read_csv_to_table_without_header (csv_file_path):
    with open(csv_file_path, 'r') as csv_file:
        reader = csv.reader(csv_file, delimiter=',')
        readings = []
        for row in reader:
            readings.append(row)
    return readings

table = read_csv_to_table_without_header(your_csv_file)
n = len(table)

# assume 30 ms increments between data points
# for a 4s time window, there will be a maximum of 133 data points
acceleration_time_values = np.arange(0, n*delta_t, delta_t)

# table data is in this format:
# [
#   [T, X, Y, Z],
#   [T, X, Y, Z],
#   ...
# ]
#
digital_acceleration_x_values = np.array([int(row[1]) for row in table])
digital_acceleration_y_values = np.array([int(row[2]) for row in table])
digital_acceleration_z_values = np.array([int(row[3]) for row in table])

analogue_acceleration_x_values = (digital_acceleration_x_values * 5/1023)
g_force_x_values = (analogue_acceleration_x_values - 1.65)/0.8
acceleration_x_values = g_force_x_values * 9.8

analogue_acceleration_y_values = (digital_acceleration_y_values * 5/1023)
g_force_y_values = (analogue_acceleration_y_values - 1.65)/0.8
acceleration_y_values = g_force_y_values * 9.8

analogue_acceleration_z_values = (digital_acceleration_z_values * 5/1023)
```

```
g_force_z_values = (analogue_acceleration_z_values - 1.65)/0.8
acceleration_z_values = g_force_z_values * 9.8

# normalise it to 0 mean
normalised_acceleration_x_values = acceleration_x_values -
 np.mean(acceleration_x_values)
normalised_acceleration_y_values = acceleration_y_values -
 np.mean(acceleration_y_values)
normalised_acceleration_z_values = acceleration_z_values -
 np.mean(acceleration_z_values)

# estimate the frequency for curve fitting

def parabolic(f, x):
    """
    Quadratic interpolation for estimating the true position of an
    inter-sample maximum when nearby samples are known.
    f is a vector and x is an index for that vector.
    Returns (vx, vy), the coordinates of the vertex of a parabola that goes
    through point x and its two neighbors.
    Example:
    Defining a vector f with a local maximum at index 3 (= 6), find local
    maximum if points 2, 3, and 4 actually defined a parabola.
    In [3]: f = [2, 3, 1, 6, 4, 2, 3, 1]
    In [4]: parabolic(f, argmax(f))
    Out[4]: (3.2142857142857144, 6.1607142857142856)
    """
    xv = 1/2. * (f[x-1] - f[x+1]) / (f[x-1] - 2 * f[x] + f[x+1]) + x
    yv = f[x] - 1/4. * (f[x-1] - f[x+1]) * (xv - x)
    return (xv, yv)

def freq_from_autocorr(sig, fs):
    """
    Estimate frequency using autocorrelation
    """
    # Calculate autocorrelation (same thing as convolution, but with
    # one input reversed in time), and throw away the negative lags
    corr = fftconvolve(sig, sig[::-1], mode='full')
    corr = corr[len(corr)//2:]

    # Find the first low point
    d = np.diff(corr)
    start = find(d > 0)[0]

    # Find the next peak after the low point (other than 0 lag).  This bit is
    # not reliable for long signals, due to the desired peak occurring between
    # samples, and other peaks appearing higher.
    # Should use a weighting function to de-emphasize the peaks at longer lags.
    peak = np.argmax(corr[start:]) + start
    px, py = parabolic(corr, peak)

            # activate to draw autocorrelation graph
            # plt.plot(acceleration_time_values, corr)
            # plt.plot(acceleration_time_values[px], py, 'r*')
            # plt.xlabel("Time (s)")
            # plt.ylabel("Autocorrelation")
            # plt.title("Autocorrelation for Anticlockwise Rotation Data")
            # plt.show()

            return fs / px

        f = freq_from_autocorr(normalised_acceleration_x_values, 1/delta_t)
        angular_f = f * 2*np.pi

        def func(acceleration_time_values, a, b, c):
            return a*np.sin(angular_f*acceleration_time_values + b) + c
```

```python

        popt, pcov = curve_fit(func, acceleration_time_values, normalised_acceleration_x_values)

        a = popt[0]
        b = popt[1]
        c = popt[2]

        print("Amplitude: " + str(a))
        print("Frequency: " + str(f))
        print("Phase Shift: " + str(b))
        print("Vertical Shift: " + str(c))

        print('Curve fitting formula: Acceleration(t) (m/s^2) = ' + str(a) + 'sin(' + str(angular_f) + 't + ' + str(b) + ')' + ' + ' + str(c))

        plt.plot(acceleration_time_values, acceleration_time_values * 0, 'g-')
        plt.plot(acceleration_time_values, normalised_acceleration_x_values, 'r.')
        plt.plot(acceleration_time_values, func(acceleration_time_values, a, b, c), 'r')
        plt.xlabel("Time (s)")
        plt.ylabel("Acceleration (m/s^2)")
        plt.title("Frequency Autocorrelation Method for RPS Estimation")
        plt.show()
```