University of Birmingham

— The Outfielder Problem —

A Virtual Reality Testing Environment

Alexandru Bracau (1646036)

Supervision: Dr. Sang-Hoon Yeo & Robert J. Hendley

B.Sc. in Computer Science

School of Computer Science, University of Birmingham

April 12, 2019

Contents

Abstract

The Outfielder Problem has been a cause for debate among the scientific community ever since Seville Chapman first proposed a catching strategy in 1968. The problem with testing the proposed strategies is that they all produce similar and successful strategies in the real world. Therefore, in an attempt to settle this debate, researchers have turned to virtual environments where they can create "impossible" situations for subjects attempting to catch virtual fly balls and accurately monitor their behaviour. This report presents a highly configurable virtual environment, designed to be used with the state-of-the-art FOVE head-mounted display which provides eye-tracking capabilities that can be used to further study the Outfielder Problem. A short experiment was conducted to test whether the environment is fit for purpose. Following the experiment, it was concluded that subjects initially behave similarly to real outfielders, but a flaw in the motion control system may cause them to adopt an unnatural behaviour pattern after becoming more accustomed to the environment.

Acknowledgements My supervisors have provided me with a wealth of support throughout this past year. For this, I am very grateful. I would like to thank Dr. Sang-Hoon Yeo in particular for sharing his expertise on the subject and for always keeping me on track, helping to structure my progress, and Robert J. Hendley for the guidance he has provided for the project overall and for our insightful discussions. I am also very thankful to all of the participants who took part in the experiment and in preliminary tests, whose feedback has proven invaluable for the development of this tool.

Repository All software for this project can be found at https://git.cs.bham.ac.uk/axb1212/outfielder-environment.

1 Introduction

The purpose of the work described in this report was to build a highly configurable Virtual Reality (VR) testing environment which can be used to further explore the Outfielder Problem, a particular example of the problem of interception.

In the game of baseball, the outfielder plays in a defensive position farthest from the batter. The outfielder's duty is to catch the ball hit by the batter before it hits the ground. This includes two distinct tasks: firstly, arriving at the landing spot of the ball at a specific point in time and secondly, reaching out to catch the ball in their glove. The outfielder problem and consequently, the testing environment, concerns the first part.

It is, conceptually, a very simple task, but difficult nevertheless, given the size and speed of the ball relative to the baseball field, with average fly balls landing over 100m away from the batter. Human vision has been found to be unable to offer accurate distance measurement over 30 meters away (Cutting and Vishton, 1995), meaning the catcher is also unable to determine the velocity of the ball (particularly vertical velocity while situated below the ball (Shaffer and McBeath, 2005)). Furthermore, even experienced outfielders cannot accurately determine when the ball has reached the apex of its parabolic trajectory (Shaffer and McBeath, 2005). Therefore we ask ourselves the question, how is this task performed successfully every day in spite of these difficulties. This is the "Outfielder Problem".

This problem has received some level of attention in recent decades, therefore a number of proposed theories explaining the behaviour of the catcher have obtained support. However, due to the nature of these strategies, distinguishing between them in actual practice has proven difficult. With "natural" ball flights, the proposed models all predict successful (and similar) paths. This is why most recent studies turn to VR environments where the ball's trajectory can be manipulated in such a way that would cause the predictions of the competing models to diverge, hence allowing researchers to more accurately judge the validity of these models.

Thanks to recent advances in VR technology such as higher pixel count displays as well as integration with eye tracking at lower costs than ever, such studies in virtual environments allowed for great strides in all areas of vision research. The tool presented in this report makes use of these advances to provide a virtual environment suitable for such studies in the context of the Outfielder Problem. Furthermore, it implements two of the proposed catching strategies so that they can be compared against the trajectory of the subjects inside the virtual environment.

This report will begin by outlining the debate surrounding the Outfielder Problem, presenting the most supported theories and the arguments for and against each of them, as well as the most recent research on the subject. It will then discuss other studies using VR environments and whether or not using a VR environment can be used to gather information about the task of the outfielder.

Afterwards, the design and implementation of the application will be presented, focusing on the structure of the experiment, the design of the virtual environment, the configurability of the system and the user interface, the implementation of motion control, data collection and finally the implementation of the prediction models.

Finally, the experiment ran to test the validity of the virtual environment will be described and the results will be discussed. Following this, the observed limitations and insufficiencies of the tool and further work will be addressed.

2 The Outfielder Problem

The following subsections will be detailing the catching strategies that have gained the most support and which were taken into account when designing this tool. Discussion of these results and their relevance to the development of the tool are primarily reserved for later sections.

These strategies can be grouped into two categories: ”online” and ”offline”. The offline approach posits that the catcher possesses an internal physical representation of the world, and by observing a set of initial parameters of the ball, can predict with a reasonable degree of accuracy where the ball will land. The online approach, on the other hand, assumes that the catcher uses continuous visual input to control motion, without the need for an internal physical model.

2.1 Trajectory Prediction

One approach proposes that the catcher performs an accurate trajectory prediction (TP) (Saxberg, 1987), under the assumption that humans are capable of such a feat. This ”offline” approach assumes the player possesses the initial conditions of the ball's launch (i.e. velocity, direction, spin, wind speed and direction), acquired via visual input, audio input of the bat hitting the ball and tactile input offering wind information altogether coupled with the catcher's experience of how each input affects the trajectory of the ball. The catcher is then able to make a prediction based on an internal physical model of the world and head towards the landing area.

However, players have not been found to be able to accurately predict ball trajectories (Shaffer and McBeath, 2005), either due to limitations of human perception or the inaccuracy of the internal model. Furthermore, extensive research on the topic of model-based control in the past decades has shown that some tasks can be performed normally with longer periods of visual occlusion, while others become impossible with even short periods of missing visual input.

Take the classic blind walking test for example, where subjects are required to walk to a target after briefly viewing the environment. Most findings suggest humans are capable of performing this task successfully with targets up to 30 meters away (Loomis et al., 1992; Rieser et al., 1990). This is also performed successfully even when participants have only viewed the path for 150ms, even without directly fixating the target (Philbeck, 2000).

On the other hand, in more complex tasks involving catching, increasing occlusion time resulted in exponentially increasing error rates. Bennett, Ashford and Elliott (2003) and Bennett, Elliott, Weeks and Keil (2003) study participants attempting to catch a tennis ball, with visual samples of 20ms at 60ms or longer intervals, and performance appeared to decrease significantly with periods of occlusion longer than 60ms to 80ms. The paper by Zhao and H.Warren (2015) offers a more in-depth discussion on this subject.

Therefore, most research seems to suggest that an internal model is used for short periods of time where visual information of the ball is either unavailable or unreliable, for example when running and not facing the ball or when it is difficult to clearly assess the ball's position in conditions of poor visibility such as having the sun behind the ball or dense fog. Nevertheless, initial information is not sufficient by itself and regular visual input is needed to perform a successful catch. The following two strategies reflect this need.

2.2 Optical Acceleration Cancellation

In light of the concerns listed previously, Chapman (1968) proposes an ”online” approach in the form of the Optical Acceleration Cancellation strategy (OAC), which uses current visual input to dictate movement towards the landing point. This strategy involves the player moving radially toward or outward from the ball in such a way that cancels the perceived acceleration of the ball's optical projection. If the ball were to increase in velocity, the catcher should accelerate backwards and if the ball's velocity decreases, the catcher accelerates forwards. In the original model, lateral motion is controlled by keeping the azimuth angle of the ball relative to the player constant.

Following this strategy, the catcher should be in the right place at the right time to catch the ball. This strategy has seen notable support (McLeod and Dienes, 1996; Michaels and Oudejans, 1992) particularly in the form of the Generalized Optical Acceleration Cancelling theory (McLeod et al., 2006).

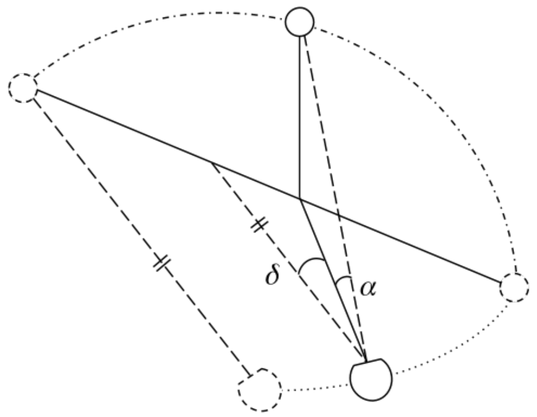


Figure 1: Diagram of the outfielder task. Commonly, the elevation angle is labeled α, and the rotation angle δ, as marked in this figure

The GOAC strategy improves upon the OAC strategy by providing a means of controlling lateral motion and posits that lateral motion is controlled by cancelling the acceleration of the rotation angle. More specifically, the catcher should move in a such a way that maintains the acceleration of α, the elevation angle, as well as δ, the angle of rotation from the initial orientation, constant.

The main argument against it is that humans were observed to possess quite poor abilities to detect acceleration (Calderone and Kaiser, 1989; Schmerler, 1976), so poor in fact that in certain contexts, detection of acceleration may require a change in velocity of about 20% of the average velocity of the projectile (Babler and Dannemiller, 1993).

However, this could be heavily dependant on visibility conditions and the skill level of the fielder. Furthermore, the strategy is only required to bring the player in close to the landing point with enough time left to perform final adjustments and catch the ball, since with distances of less than 30m perception is much more helpful and the OAC strategy is error-nulling, so the deviation from the right path is corrected given enough time.

2.3 Linear Optical Trajectory

In response to the observed inability of humans to accurately measure acceleration, McBeath et al. (1995) propose the Linear Optical Trajectory (LOT) model. It posits that the catcher runs in such a way as to maintain the optical trajectory of the ball linear, as perceived by the player, appearing as if it is moving in a constant direction relative to the home plate. Therefore the LOT model does not require detection of acceleration, but only detection of deviation from the straight path.

Mathematically, LOT requires the catcher to maintain the picture plane angle ψ constant. This automatically implies that the acceleration of α and β are kept constant and equal, where α and β are the angles depicted on the right.

The issue of the LOT model by itself is that it does not provide a means of translating visual input directly into movement, but only "advises" towards a family of paths to be taken. This arises because any picture plane angle can be obtained from infinitely many pairs (α, β). Aboufadel (1996) discusses the mathematical limitations of this model thoroughly.

So far, neither the LOT nor the OAC strategies provide an explanation for the many examples of skilled outfielders turning their back on the ball (in order to run faster towards the landing point) only to turn around to face the ball with only moments left to catch the ball. This, of course, supports the existence of an internal model used for short-term predictions.

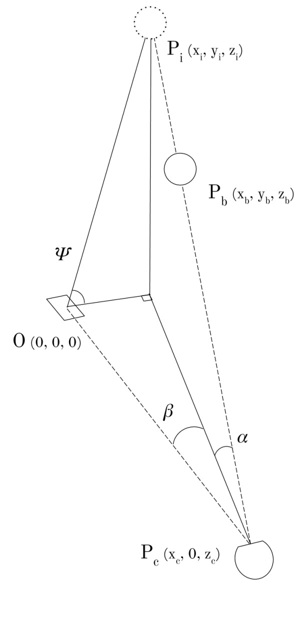


Figure 2: The picture plane angle is commonly labeled ψ

2.4 Reconciling the three

Until now it has become clear that each strategy, on their own, is lacking in some areas, therefore most recent studies have attempted to reconcile them to provide a sound model for the problem at hand. Although not strictly relevant to the development of this application, it is worth mentioning a few prominent examples for completeness sake.

Zhao and H.Warren (2015) propose that in the case where visual information is readily available, the "online" strategy by itself is sufficient for normal performance. However, where visual input is unavailable or unreliable, action is controlled by "offline" strategies, but not without a severe decrease in performance. They also observed that a world model is neither sufficient nor required for normal levels of performance.

Similar results are reported by Belousov et al. (2016). They conclude that in the outfielder case, the catcher will turn to face the ball as soon as possible to gather information given they have enough time to do so (i.e. they don't have to sprint towards the landing point) and continue tracking until the end. If they must sprint, during this time they will rely on an internal model. A study by de la Malla and Lopez-Moliner (2015) also found evidence that people use a predictive model that is continuously integrated with visual information when faced with the task of estimating the time of interception of a ball that is visually occluded for selected segments of its flight.

2.5 Virtual Reality and the Outfielder Problem

Almost all of the recent research on this topic has been performed in some type of virtual setting. As previously mentioned, the main problem in testing these theories is that they provide relatively similar paths to interception. Hence, most experiments provide physically impossible ball trajectories to create cases where the models would expect divergent paths.

Zaal and Michaels (2003) are among the first to test this problem inside a virtual environment. They do this inside a CAVE, a 3x3x3m space where the subject can move freely, while the virtual environment is projected on the walls surrounding the subject. Manipulating the angular size of the ball allowed them to obtain evidence supporting the OAC model.

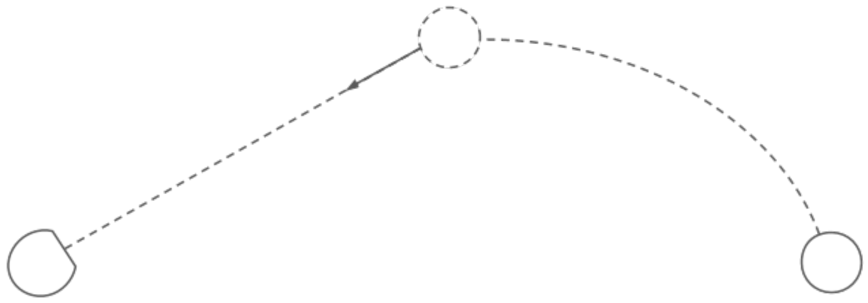


Figure 3: Artificial path manipulation in the study by Fink et al. (2009). At specific times during the the ball's parabolic flight, it would change direction to follow a straight line path towards its initial destination.

Fink et al. (2009) similarly attempt to test these strategies in a VR environment. They used the VENLab, a 12m by 12m area where the participant can move freely, together with a head-mounted display (HMD) (640 by 480-pixel resolution for each eye at a 60Hz refresh rate). By manipulating the ball's trajectory artificially (Figure 3), they found evidence refuting the LOT model in favour of OAC and also supporting the theory that with constant visual information available, no internal model is required and action is guided by coupling visual information to movement.

Diaz et al. (2009) also carry out their experiments in a virtual setting. They investigate the constant bearing angle (CBA) strategy of interception. They artificially alter the ball's velocity mid-flight and observe the subject's reaction. However, instead of using an HMD, they use a projector with 1280 x 1024 resolution onto a 1.8 x 1.2m screen at 60Hz (inside a carefully constructed dark room) with the participants seated 1m in front of the screen. The subjects controlled the virtual catcher using a foot pedal.

These examples are presented here, one one hand, to outline the recent rise in popularity of VR settings as vehicles for studies in human vision, but also to illustrate the variety of different types of environment and motion control utilised, as well as different kinds of manipulation that can be used to study the validity of the proposed theories. Nevertheless, the popularity of this practice is not enough to prove that these results comprise proper clinical evidence, as will be discussed in the following subsection.

2.6 Conclusions

The previous subsections have outlined the reasons why The Outfielder Problem is still a cause for debate to this day and how different studies attempt to use VR systems to find definitive answers. However, one important issue is wheter or not this approach can provide meaningful information about real outfielders and the problem of interception in general.

In psychology, the term *ecological validity* is often used to refer to how well the findings from an artificial or highly simplified context can translate to the complex real world. The debate around the ecological validity of tests in laboratory studies has been ongoing since the early 1990's (Conway, 1991), and a special issue of the American Psychologist was devoted to this debate in 1991. This is, of course, referring to laboratory studies in general, not specifically VR environments.

The highly realistic virtual environments used in recent studies further complicate this debate, with the apparent line between the real world and the virtual one thinning with every leap in virtual reality technology. The task of the outfielder specifically is not necessarily an example of a controversial application of VR environments as a research tool from an ecological validity point of view, since the simulated task is, although simplified, largely true to life, because it is not highly abstracted but merely replicated.

The question of the validity of the results from a virtual environment remains open, with analysis generally restricted to qualitative assessments and on a case-by-case basis Zaal and J.Bootsma (2011). As such, a qualitative test was performed to compare the paths of subjects inside this virtual environment to the paths of real outfielders. See the Evaluation and Testing section for details. For now, however, we can only conclude that the relevance of the data from this environment is largely dependant on the configuration of the experiment and the burden of proof is passed on to the researcher.

3 Design and Implementation

This section will describe the implementation of the tool and will present short discussions related to the design choices made. In the following subsections, different components of the tool will be elaborated in reasonable detail, beginning with the structure of an experiment, followed by the virtual environment as experienced by the subject, the control input system, the configuration and data collection systems and finishing with the implementations of the proposed strategies.

The system requirements could be summarised as follows: the tool must provide the means for conducting highly customisable experiments inside a virtual environment simulating the task of the outfielder. The tool should collect all data relevant to the task and output it in a usable format. Furthermore, it should implement two of the proposed online strategies and output their predictions on a case-by-case basis.

The virtual environment was developed with the Unity3D engine exclusively for the *FOVE 0* HMD. The FOVE headset uses a WQHD OLED display with a resolution of 2560x1440 and a refresh rate of 70Hz, offering a field of view of up to 100°. The head tracking camera tracks the headset's position at 100Hz and is augmented with acceleration values and rotation values at sampled at 1000Hz. The two infrared eye tracking sensors track at 120Hz with an accuracy of $> 1°$(link to official specifications).

The physics updates inside the virtual environment are performed at 100Hz with a lower limit of 10Hz for the worst-case scenario (i.e. a fixed timestep of 0.01s and a maximum allowed timestep of 0.1s), resulting in a sampling rate for data collection of approximately 100Hz.

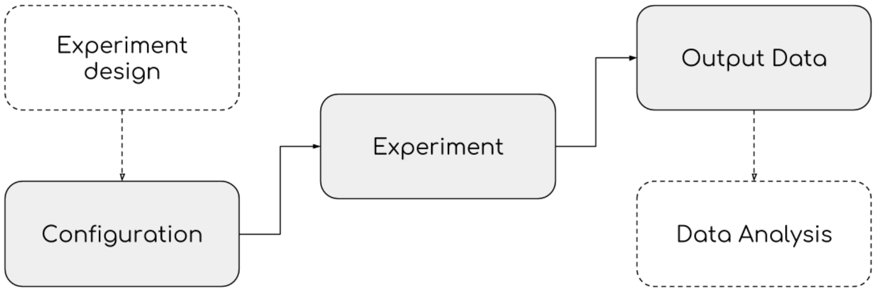


Figure 4: Typical usage of the tool

The figure above provides a basic overview of the typical usage of this tool. After designing an experiment, the user may configure the environment either via the UI or a configuration file, then conduct the experiment with any number of subjects inside the environment and finally analyse the data output by the environment to draw conclusions. Throughout the development process, these three core functions were kept in focus.

3.1 Structure of the Experiment

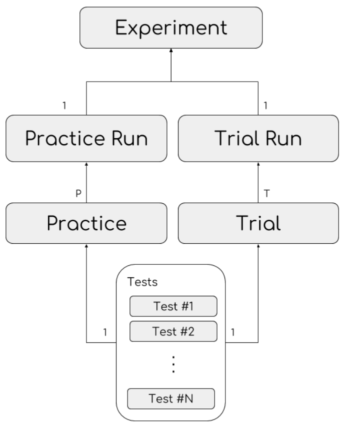


Figure 5: Experiment Structure

The diagram on the left breaks down the structure of an experiment. This section is important because it explains the terms used throughout the rest of this paper.

Firstly, a *test case* represents a collection of initial parameters for one catching task. Specifically, a test case contains information like the initial velocity vector of the ball, the target landing point, etc. as specified in the configuration file. This is detailed in the Configuration chapter. There are N such test cases, which form the set of test cases.

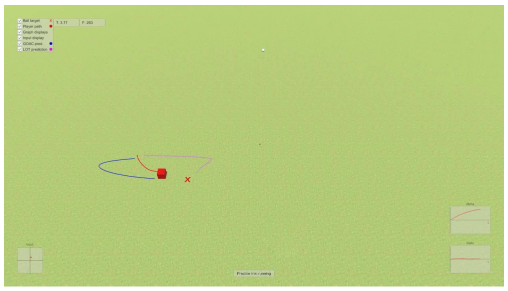
Running a test case refers to setting up and conducting a catching task as described by the parameters stored in the test case. A test case can be run as either a *trial* or *practice*. This is only a conceptual layer and is only recorded as an `enum`. From here on, unless otherwise specified, the term *trial* will be used to refer to both trials and practices.

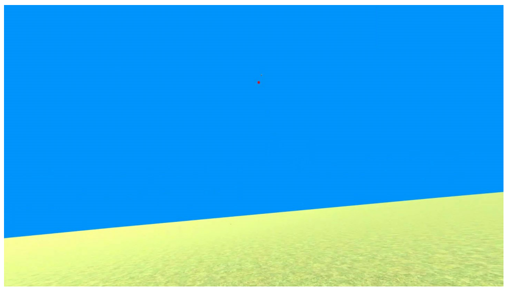
Note: Unless the user configures any special behaviour, there is no difference between a trial and a practice from the subject's perspective. Furthermore, data from the environment is collected for trials and practices alike. The only practical difference between them is that data collected is stored in two separate folders, one for the practices and one for the trials. This feature was implemented, on one hand, to allow for custom behaviour, but also for clarity, providing another layer of structure to the collected data and to the experiment as a whole.

A *practice run* or *trial run* consists of N practices or trials such that a practice run runs exactly one copy of each configured test case. There are P or T practice or trial runs respectively. The order of the practices is randomised across all practice runs and the order of trials is randomised across all trials. Finally, an *experiment* consists of conducting P practice runs followed by T trial runs.

A *trial manager* stores the list of test cases and is responsible for managing the data collector and catching strategy models, as well as implementing the functionality for starting and completing trials. The responsibility of running and keeping track of trials and practices lies with the *trial runner*. The trial runner also manages starting, pausing and stopping the experiment, as well as updating the user interface (specifically the *Experiment* panel).

3.2 The VR Environment


(a) Bird's eye view of the ongoing experiment with all overlays active.


(b) The approximate view from the subject's perspective. Red dot shows gaze convergence. In this example, the subject's head is tilted to the right.

The level of realism employed in the simulation needs to be discussed. Figure 6b shows an example of what the subject might see during the trial. By most definitions, this would not be labelled as "realistic", but the aim of the environment was not to completely recreate the conditions on a baseball stadium.

Rather, it should simulate the outfielder's *task* and only the task, maintaining realistic visual cues and respecting the laws of physics (unless intentionally manipulated). For this reason, during the catching task, all physics and graphics are left in the hands of Unity's physics and graphics engines. It has proven to be reliable, consistent and also customisable to the required level.

The simulation should, however, feel as natural as possible to the subject, despite the simplified visual component. In accordance with common practices and considering the scope of this environment, the only elements present in the subject's view during the catching task are the ball, a solid colour skybox and the grass textured ground.

One sensitive aspect of the environment is, of course, motion control. The first problem is that most studies so far have seen the subjects having trouble catching the virtual balls using virtual hands: Zaal and Michaels (2003) observed the participants (in a CAVE system) fail to perform a successful catch with their hands, captured using a motion glove, 76% of the time, while Fink et al. (2009)) observed subjects catching the balls approximately half the time.

However, much higher success rates were obtained when subjects attempted instead to intercept the ball with their forehead. Notably, the same study by Zaal and Michaels (2003) observed an increase in successful catches to about 50%. In a similar task, McLeod et al. (2008) reported a success rate of 75% with unperturbed ball trajectories, albeit in a soccer context. For this reason, in order to catch the virtual fly balls inside *this* virtual environment, the subject must attempt to intercept the ball with their forehead. The size of the actual hitbox may be configured manually.

In an attempt to increase immersion, the option of bringing the subject's hands inside the virtual environment was explored. This would have been done using a LEAP Motion device — a very compact system mounted on the front of the VR headset that uses an infrared camera to capture the hands of the player. The integration process turned out to be quite straightforward, but one critical issue prevented this arrangement from functioning correctly. As described in the official white paper by A. Savkin et al. (2017), the FOVE headset uses a novel approach to position tracking, for which the headset is fit with infrared LEDs which are picked up by the tracking camera.

Unfortunately, the LEAP Motion infrared camera interferes dramatically with this system, to the point where the motion of the headset could not be accurately determined anymore and drift would accumulate until the headset orientation would be abruptly reset (even though the headset managed to perform admirably using the internal accelerometer data). For this reason, the LEAP Motion system could not be integrated with the virtual environment.

Due to technological limitations, all studies using a virtual environment where the subject can move freely so far restricted the "baseball field" to an area a few meters across. In order to replicate the scale of the real task inside a virtual environment, an alternative means of controlling the virtual catcher has to be employed. The environment implements two methods of motion control: either using the tracked position of the FOVE headset (in practice, by leaning in the direction they want to move) or via *any* joystick input, virtual or physical. An in-depth description of the input methods is reserved for the Motion Control section.

The tool also provides a couple of ways for the user to monitor the virtual environment from the computer screen. Apart from the UI view, two more views are available from the main display: a bird's eye view of the catching task (figure 6a) and an approximation of the subject's view which also features a red dot to observe the subject's gaze (figure 6b).

The bird's eye view also provides a few live displays that may offer some insight into the task at hand, such as the display of the ball's target destination, the path of the virtual catcher within the current trial, the paths predicted by the catching strategies, a display showing the subject's controller input as well as live graphs of the α and δ angles.

An attempt was also made to display the accelerations of the two angles in a similar fashion. However, it meant extracting the second derivatives of the two angles, which did not yield any usable results, even after smoothing.

3.3 Configuration

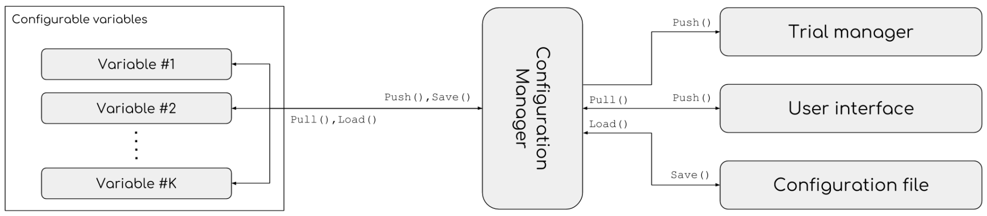


Figure 7: Illustration of the configuration system

The primary strength of this tool is its configurability and the ability to easily add configurable elements in the future. The figure above shows the structure of the configuration system.

The *configuration manager* stores a list of pre-defined `Variable<T>` objects. Each of these variables stores its own name (to be referenced in a configuration file), a reference to the variable it configures, a reference to the user interface container for it (if applicable) where to push updated values or from where to pull new values if set via the UI, a default value as well as an accepted range and optionally an `Action` to be executed when the set value is changed.

In reality, for primitives to be configurable via a `Variable<T>`, they need a wrapper class. This is because C# only allows the use of pointers in very limited circumstances and primitives are value types, therefore `Variable<T>` objects cannot store references to them. This is done using the `Configurable<T>` class.

In order to provide a smooth experience while configuring the experiment from the user interface, the environment is reconfigured any time certain variables are changed from the menu. To this effect, the `ConfigurationManager`'s `Refresh()` method implements a three-step cycle executed for each `Variable<T>`:

1. `Pull()`: Parse the variables from their referenced containers (if applicable) from the user interface and store them inside the `Variable<T>` objects, regardless of their value.

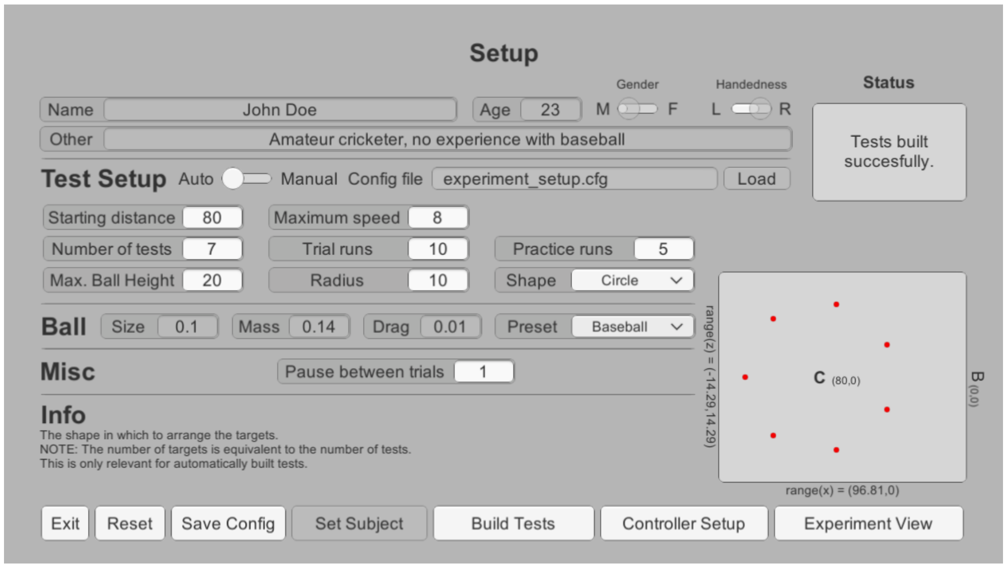
2. `Check()`: Check if the value stored is in the accepted range. This step is separated from the `Pull()` step in order to be able to offer more accurate error messages to the user.

3. `Push()`: Push the values from the `Variable<T>` objects to their referenced `Configurable<T>` objects and also push other affected variables to their UI containers (e.g. when setting a ball preset, also push the new ball size, mass and drag to the UI containers for size, mass and drag respectively)

Note: As will become clear in the following sections, a significant portion of the user interface for this tool is provided primarily as a way of sanity checking the setup from a configuration file. In that sense, dynamically configuring the environment from the interface at runtime may seem a bit excessive. However, it should be noted that the user interface itself is built into the virtual environment, therefore the configuration will effectively be done at runtime either way.

3.3.1 User Interface

The user interface provides a set of controls for configuring basic functionality. As previously mentioned, the user interface is provided for convenience, and primarily as an aid during the configuration process.


Setup

Name: John Doe
Age: 23
Gender: M / F
Handedness: L / R
Other: Amateur cricketer, no experience with baseball

Status: Tests built succesfully.

Test Setup: Auto / Manual
Config file: experiment_setup.cfg

Starting distance	80	Maximum speed	8		
Number of tests	7	Trial runs	10	Practice runs	5
Max. Ball Height	20	Radius	10	Shape	Circle

Ball: Size 0.1, Mass 0.14, Drag 0.01, Preset Baseball

Misc: Pause between trials 1

Info
The shape in which to arrange the targets.
NOTE: The number of targets is equivalent to the number of tests.
This is only relevant for automatically built tests.

Figure 8: Setup menu view with a basic configuration loaded.

Before continuing to the *Experiment menu*, the user must first set a subject inside the *Setup menu*. Via the fields at the very top, the user *must* set the subject's name and can *optionally* set the subject's age, gender, handedness and other information they might require. The subject's handedness and gender are set by default to *right-handed* and *male* respectively and can be changed via the sliders.

The auto-manual slider switches between the option of using the user interface to automatically generate a set of test cases and loading a configuration from a file.

The *Auto* setting can be used to generate a given number of tests such that the landing points of the ball in each test form a specified shape around the catcher, either a circular shape or a square shape of specified radius (for a square shape, the number of tests must be divisible by 4 and the radius represents the distance from the virtual catcher to a side of the square). By specifying the maximum height reached by the ball and using the generated destination points, the trajectories for the ball can be uniquely determined. When the Auto setting is selected, the user has access to the variable containers (i.e. the labelled input boxes).

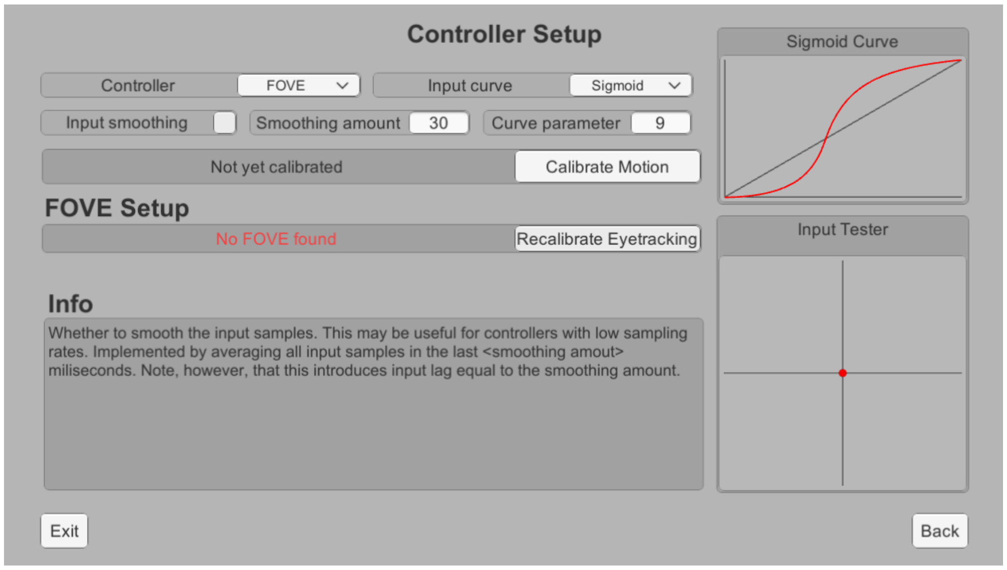
When the *Manual* setting is selected, access to the input boxes is blocked, and the configuration must be loaded from an input file specified via the input box to the right of the slider. Any errors encountered while loading the configuration file will be displayed in the status box in the upper right of the screen. Any variable not assigned a value in the configuration file will be assigned the default value. The variable containers will briefly change colour, either to green if a value was configured for that variable or red otherwise. This is meant to help identify any variables whose values were not set in the configuration file and provide some visual feedback that the configuration loaded correctly and completely.

Apart from the access it provides to some configurable variables, the setup menu also offers a basic display showing the tests loaded in the experiment, as shown in figure 8. The red dots represent the destination points of the configured ball launches. Due to the nature of the system for ball trajectory manipulation, it is not possible to display where these manipulated trajectories may take the ball. However, this still adds value to the user interface by providing a means of easily finding errors in the configuration file.

From this menu, the user can save the configuration set via the variable containers to a configuration file by pressing the *Save Config* button. Note that this feature is only available in *Auto* mode when the containers are accessible.

After loading a configuration file or having configured the variables via the containers, the user must build the test cases by pressing the *Build Tests* button. Only after this is done the user can navigate to the *Experiment* menu.

The *Info* box at the bottom provides some information about the container the user's cursor is currently hovering over. This infobox is also present in the *Controller* menu and is not meant as a complete, detailed explanation of the item being described, but does provide some useful information in most circumstances.



Controller: FOVE
Input curve: Sigmoid
Smoothing amount: 30
Curve parameter: 9
Not yet calibrated
No FOVE found

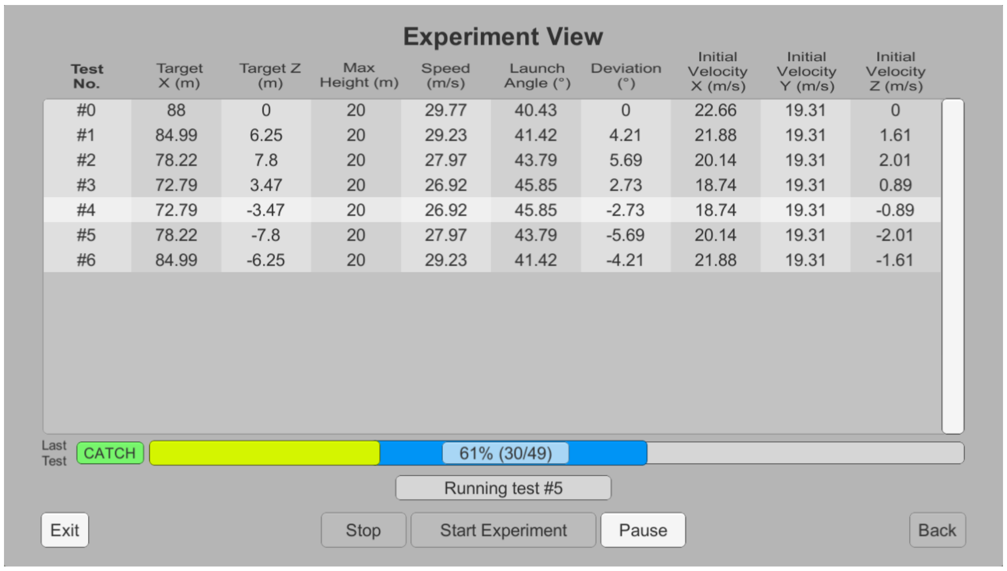
Whether to smooth the input samples. This may be useful for controllers with low sampling rates. Implemented by averaging all input samples in the last <smoothing amout> miliseconds. Note, however, that this introduces input lag equal to the smoothing amount.

Figure 9: Controller menu view.

The *Controller* menu provides functionality for configuring the input method for motion control. A detailed description of the input methods implemented can be found in the Motion Control section, along with a description of the input curve functions, the implemented input smoothing and the process of calibration.

The curve display in the upper right allows for a visualisation of the input curve and is updated dynamically when the input curve dropdown or the curve parameter values are changed. The input tester below that provides a means of testing the selected input method after calibration. Both of these are intended to help with making sure that the intended effect of the input smoothing and input curve are obtained.

The *Experiment* menu serves as the main control panel for the actual experiment being run. The content panel at the centre of the menu shows the list of all configured test cases and their parameters. The test currently being run is highlighted during the experiment.


Experiment View

Test No.	Target X (m)	Target Z (m)	Max Height (m)	Speed (m/s)	Launch Angle (°)	Deviation (°)	Initial Velocity X (m/s)	Initial Velocity Y (m/s)	Initial Velocity Z (m/s)
#0	88	0	20	29.77	40.43	0	22.66	19.31	0
#1	84.99	6.25	20	29.23	41.42	4.21	21.88	19.31	1.61
#2	78.22	7.8	20	27.97	43.79	5.69	20.14	19.31	2.01
#3	72.79	3.47	20	26.92	45.85	2.73	18.74	19.31	0.89
#4	72.79	-3.47	20	26.92	45.85	-2.73	18.74	19.31	-0.89
#5	78.22	-7.8	20	27.97	43.79	-5.69	20.14	19.31	-2.01
#6	84.99	-6.25	20	29.23	41.42	-4.21	21.88	19.31	-1.61

Figure 10: Experiment menu view.

Below this, a bicolour progress bar shows the completion percentage, with the lime coloured bar representing the practice runs and the blue bar representing the trial runs. To its left, the result from the previous test is shown, as either a *Catch* or a *Miss*. Below the progress bar, a status box relays any relevant information to the user.

The *Start Experiment* button brings the virtual catcher to the field and starts the experiment. The experiment cannot be started unless the selected controller has been calibrated first. The *Pause* button causes the experiment to pause after the current trial is finished. This should not be done normally, but if a subject is feeling unwell and needs to take a longer break, the user running the experiment has the option of pausing the experiment without losing the progress within the experiment and having to start over, in turn only sacrificing the data from one trial in a worst case scenario. The *Stop* button becomes available when the experiment is in a *paused* state, terminating the experiment prematurely.

3.3.2 Configuration File

Configuration should preferably be conducted via the configuration file. The configuration manager implements a basic parser and allows for 2 types of input (see table 1): setting a configurable environment variable and creating a test case, as well as C-style double slash comments (beginning anywhere on a line).

Function	Syntax	Description
Variable	`$<variable_name>=<variable_value>`	Sets the variable `<variable_name>` with the value `<variable_value>`.
Test Case	`@<type_specifier>=<param1>,<param2>,<param3>`	Creates a test case as specified by the `<type_specifier>` and given parameters.

Table 1: Configuration functions. See the appendix for a full list of configurable variables.

The accepted range for variable values and test parameters are, unless otherwise specified, set to the maximal values that produce expected functionality of the configured variable. This implies that a user may not set, for example, the mass of the ball to a value less than 0, but they could set the initial velocity of the ball to the maximum value supported by the `Vector3` type.

Evidently, this would *most likely* cause many of the components involved to fail in unexpected ways. This conscious decision was made in order to provide complete freedom to the user when configuring their environment. All variables were tested to work for *reasonable* and even excessively *unrealistic* values (e.g.

speeds far exceeding the realm of possibility), but extreme values are not recommended. See the Conclusions and Observations section for a more detailed discussion on this topic.

For specifying the test cases, the user has two options: either generate a set of tests automatically as described in the previous subsection, by specifying a shape, radius and maximum ball height or manually configure each test case. There are 3 ways of configuring a test case as described in the table below.

Type	Description	Parameters	
T	Specify by the target destination and maximum height to reach	x	target X coordinate
		z	target Z coordinate
		h	maximum ball height
V	Specify by the initial velocity of the ball in 3 dimensions	x	initial velocity on the X axis
		y	initial velocity on the Y axis
		z	initial velocity on the Z axis
P	Specify by a set of initial parameters for the ball	s	speed of the ball at launch
		l	vertical launch angle of the ball
		d	horizontal launch deviation of the ball

Table 2: Description of the test specifiers

Note that the advantage of specifying tests by target destination via the configuration file over using the automatic test generation, apart from the higher level of control over the positions of the targets, is that this allows for specifying individual maximum ball heights for each test case.

3.3.3 Ball Script

As a means of providing virtually unlimited access to variables of the environment to the user at runtime, a custom C# script is compiled at runtime. The script is compiled using a greatly reduced version of the Mono C# compiler. This is an open-source project maintained by Neuman (2015), the source code for which can be found at this link. Initially, the scripting component used the full Mono SDK install to compile the script, but this meant a 400MB needed to be packaged with the application for this to work. With the version used, only a 2MB .dll is required.

Nevertheless, this is not ideal, for a couple of reason: for one, although scripting in C# allows for maximum interoperability between this script and the rest of the application, it is not the most user-friendly scripting language — a Python implementation would have been preferred; furthermore, the compiled script is not explicitly ”sandboxed” in any way, which could lead to unstable configurations.

The table below details the methods available for editing inside the script. The decision was made to expose the full script file instead of hard-coding headers, helper functions and directives and only pasting in the implementation for these methods from text files. This way, the user has unlimited options of affecting the parameters available to the script, as long as care is taken not to affect the structure of the class.

Method Name	Variables available exclusively within the scope of the method	
`WhileRunning()`	`int currentTest`	number of the current trial
	`float currentTime`	current time within the current trial
	`Vector3 catcherPosition`	position of the virtual catcher
	`Vector3 headsetPosition`	position of the FOVE headset
`After()`	`bool caught`	the result of the completed trial
`BeforeCountdown()`	None	
`BeforeLaunch()`	None	

Table 3: Methods available for configuration via the C# script.

Furthermore, the script object stores references to the `Rigidbody` and `Transform` components of the ball. This means that throughout any of the functions described above, physics manipulations can be performed on the ball as well as potentially attaching other Unity components to the ball `GameObject` if needed. The script also stores a reference to the light source of the scene, which can be modified to change the colour of the light, its direction, intensity, shadows, etc.

Helper Method	Description
`RANDOM_INT(int from, int to)`	Returns a random integer in the interval `[from,to]`
`MAKE_WALL(Vector3 position, Vector3 rotationDegrees, bool visible)`	Instantiates and returns a prefabricated wall game object, which can be resized and manipulated in any way, either visible to the subject or not
`SET_VISIBLE(GameObject gameObject, bool visible)`	Makes the object visible or invisible to the subject
`RECORD_LINE(string data, string fileName)`	Writes the data string to the specified file

Table 4: Methods available for configuration via the C# script.

A number of functions are provided from inside the script file, some for convenience, others because the functionality implemented is not generic, in the sense that the implementation details are specific to the application. Of course, introducing other such functions in the future is trivial.

Apart from the variables already mentioned, the script also has access to the singleton *FoveInterface* class which serves as a way to obtain information about the subject's gaze and perhaps modify behaviour accordingly as well as complete access to the `UnityEngine` package, which in practice gives access to variables such as gravity, time scale, etc.

3.4 Motion Control

As mentioned previously, two input methods were implemented, along with a third provided exclusively for aiding with configuration:

1. FOVE control — The virtual catcher is controlled by the tracked position of the FOVE headset. In practice, the subject can move the virtual catcher by leaning in the direction they wish to move.
2. Joystick control — The environment is also built to receive input from any available joystick device. Apart from physical joysticks, almost any other device can be mapped to a virtual joystick (for example *vJoy*) and be used to provide input.
3. Mouse control — For testing while configuring the input curve, the user may use the mouse to provide input. This *was not meant* to be used by the subject for controlling the virtual catcher during real experiments, but that is also an option if the user wishes to configure the environment in this way.

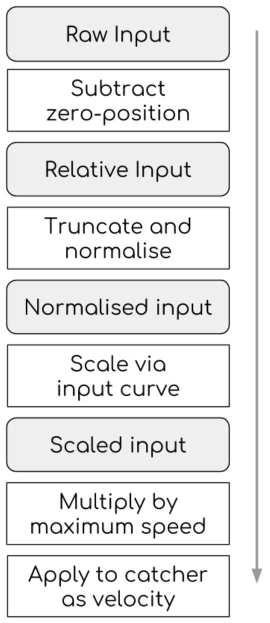


Figure 11: Representation of the input handling.

The diagram in figure 11 illustrates how input is handled by the application. The raw input is read from the input device as an (X, Y) position from which the input relative to the current zero-position is extracted, which is then truncated to the values set via the *calibration* step and normalised, then scaled according to the input curve function to obtain the scaled input vector which has a magnitude in the `[0,1]` range. Finally, the magnitude of the input vector is multiplied by the set maximum speed and applied as a velocity vector to the catcher.

The *calibration* step *must* be effectuated before the experiment can begin. After the zero-position subtraction step, all input resembles input from a joystick. As such, during calibration, the maximum and minimum values for the input are set. For example, if using FOVE control, the subject must lean in all four directions as much as they are comfortable doing. Similarly, with a joystick type device, the subject must handle the device in such a way that maximal inputs are attained.

The zero-position of the input is reset once within every trial, just before the ball is launched, but the calibration bounds are kept until a new calibration is performed manually. This is because for controllers or joystick devices that do not physically return to the zero-position (e.g. the FOVE controller) the subject will always return the controller to a slightly different neutral position and will have trouble finding the zero-position set initially.

The decision was made to apply instantaneous velocity vectors to the virtual catcher instead of applying a force since it produced a much more responsive and natural motion. Since the type of joystick-like input device well suited to this kind of task will always produce smooth, progressive transitions from one direction to the next, the velocity based system proved to be the right choice.

Input smoothing is provided in case the sampling rate of the controller device is insufficient, or the reduced granularity of the input produces jittery motion. The smoother works by collecting input samples every frame (upwards of 120Hz on powerful enough computers) and returning the average of all the samples from the previous `smoothing_amount` milliseconds. This technically introduces an input delay equal to `smoothing_amount`, but in practice, the delay is usually shorter due to the resulting input approaching the current input and largely depends on the sampling rate of the input device.

A number of input curves were implemented for the convenience of the user. The mathematical implementation of the input curves and the appropriate ranges for the parameters in each case can be found in Appendix C. These are functions $f_p(x) : [0, 1] \mapsto [0, 1]$ that manipulate the input values in an attempt to provide a more natural experience. In most cases, linearly mapping the input is the best choice, but are provided for the versatility they offer. A configurable input curve may prove most useful when using a joystick type controller.

Note: The initial plan was to use a Wii Balance Board as a means of controlling the virtual catcher. However, after some preliminary testing, some problems were found. Firstly, it was insufficiently granular and much too sensitive, meaning that although it was found to be quite intuitive, it required longer periods of practice to adjust to the movements required to control the virtual catcher. Furthermore, it did not feel like walking or running, but more like trying to use a hoverboard. For this reason, this method of input was deemed unfit for the purpose of this environment, and the FOVE control method was introduced. Nevertheless, the Wii Balance Board option is still available to use via the joystick input if it is needed.

3.5 Data Collection

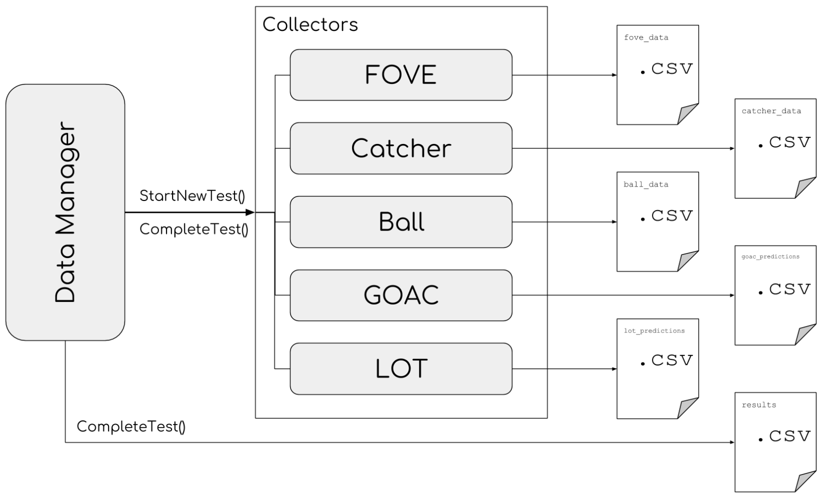


Figure 12: Diagram of data collection management

The data collection is handled by the `DataManager`. The manager holds a list of all active `Collector`s in the scene. On application startup, every `Collector` announces itself to the manager which registers them on the list.

Once the user has set a subject via the Setup menu, the `DataManager` creates a folder for the subject (if one does not exist) and writes two `.csv` files, one containing the list of test cases and one containing the information about the subject, as described previously.

When requested by the `TrialManager` at the beginning of a trial, the data manager instructs all collectors to start collecting. Whenever a collector is instructed to start collecting data from the object it is attached to, it creates a new `.csv` file in the current trial folder then launches a `Coroutine` (i.e. the Unity wrapper for a thread) that collects data at 120Hz (i.e. on every physics update), storing it inside a `StringBuilder`.

In order to minimise storage access, the collector only writes the contents of the `StringBuilder` once every second. The collection coroutine is stopped when the trial is completed. On completion of a trial, the data manager also writes a results file, containing the result of the trial as well as the total time and the total number of frames.

There are five collectors in the virtual environment: a FOVE collector attached to the FOVE object, two generic `Rigidbody` collectors attached to the virtual catcher and the ball and two strategy collectors attached to the strategy objects. Every collector stamps each entry with the current time and frame. For a complete list of the values collected by each data collector, as well as a diagram of the folder structure generated by the data manager see appendix B.

For the ball and catcher, only position and velocity are collected, which may seem insufficient. This is not the case — the orientation of the catcher's game object does not change, only the orientation of the headset (which *is* recorded), and neither does the ball's unless acted upon via the script. Similarly, the strategy collectors only record position, since the models output positions exclusively.

The FOVE collector is by far the largest file in the trial folder, collecting a total of 9 types of data, including the status of the eyes (is any eye closed), the position of the ball and the subject's gaze as projected on the screen, the position and look vectors of the headset, the look vectors for each eye and finally, as processed by the FOVE interface, the approximate point the user is currently gazing at, i.e. the convergence point of the two eye vectors, as well as the direction of the gaze. In order to correctly capture eye-tracking information, the subject goes through a calibration process every time a new *experiment* begins. This is handled by the FOVE interface software.

As a final note, it is worth mentioning that no acceleration data is collected from the environment. Unity does not implement the concept of *acceleration*. Initially, acceleration was inferred from changes in velocity using a finite difference method and recorded, but the resulting data was too noisy to be useful and the same method could easily have been used when analysing the data at a later date, perhaps with better results.

3.6 Prediction Models

Two proposed models were implemented: the Generalised Optical Acceleration Cancellation (GOAC) and the Linear Optical Trajectory (LOT) strategies, as described previously.

The strategies are managed by the `TrialManager`, which initialises them every trial and updates them every physics update with information about the ball position during trials. The process of initialisation is normally delayed an amount of time equal to the reaction time of the catcher. By default, these are set to 0.5s (Fink et al., 2009), but this is, of course, configurable via the configuration file.

The predictions from neither of these models represent real catcher trajectories since they are not constrained to human speeds or accelerations, as will become apparent when discussing their implementations. They produce *absolutely* accurate positions of *ideal* catchers that strictly respect the strategy.

During development, a "realistic" approach was implemented for the GOAC model by moving a pseudo-catcher towards the predicted correct value via linear interpolation, considering human limitations such as reaction time and maximum acceleration. This was later removed from the application, since although the resulting path appeared realistic and "human-like", it would have made it impossible for the pseudo-catcher to correctly reflect the subject's maximum acceleration on a test-by-test basis. Therefore, the model predictions are not meant to match exactly with the path taken by the subject, but rather to be paired with a type of damped spring model, parametrised according to each subject's recorded maximum acceleration and speed (in effect, producing a pseudo-catcher that always "chases" the absolute point at any time t).

3.6.1 Generalised Optical Acceleration Cancellation

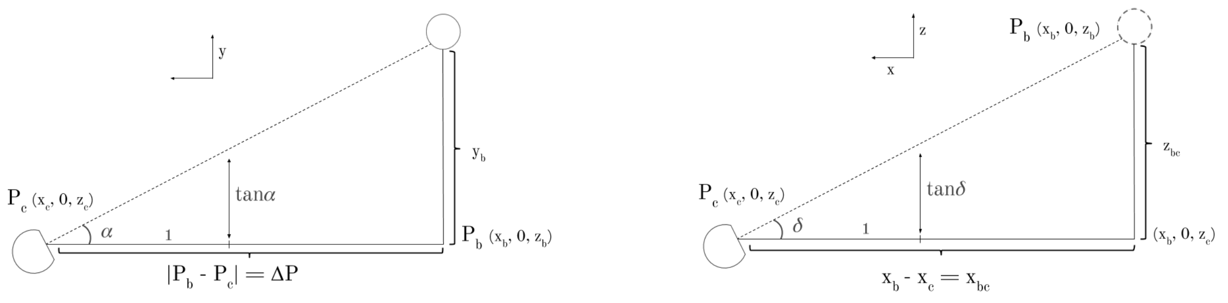


Figure 13: Left: side view of the of ball's plane of flight and the elevation angle, depicting the ball (top right) and the catcher (bottom left). Right: top-down view of the rotation angle and the XZ plane.

The strategy was implemented as described in the original paper by McLeod et al. (2006). The implementation of OAC (i.e. movement along the x-axis) follows an implementation by Rozendaal and van Soest (2003) (which only models the catcher moving along the x axis), expanded to planar motion.

Let $P_b = (x_b, 0, z_b), P_c = (x_c, 0, z_c)$ be the positions of the ball and catcher projected onto the XZ plane and y_b the y coordinate of the ball, as illustrated in figure 13. The catcher is assumed to be grounded throughout the catch for the purpose of computing the predictions, i.e. $y_c = 0$. Also, let α and δ be the elevation and rotation angles as established earlier. Then we can conclude:

$$tan\alpha(t) = \frac{y_{bc}(t)}{\Delta P(t)} \text{ where } \Delta P = |P_c - P_b| \quad (a)$$

$$tan\delta(t) = \frac{z_{bc}(t)}{x_{bc}(t)} \quad (b)$$

where $z_{bc} = z_b - z_c$, $y_{bc} = y_b - y_c$ and $x_{bc} = x_b - x_c$. This is evident from figure 13. We would like to know what the exact position of the catcher should be at each time t. Therefore, we compute the position of an ideal catcher that always maintains the acceleration of α and δ null. Then, if $\ddot{tan}(\alpha) = \ddot{tan}(\delta) = 0$, both these functions must be linear functions of t.

$$\tan\alpha(t) = a_0 + a_1 \cdot t \quad (1)$$
$$\tan\delta(t) = d_0 + d_1 \cdot t \quad (2)$$

If we find these parameters from the initial conditions of the ball and catcher, then from (1) we can deduce the equation for the distance between P_b and P_c:

$$\Delta P(t) = \frac{\Delta P_0 \cdot y_b(t)}{y_{b0} \cdot (1 + Kt)} \text{ , } K = \frac{\dot{y_{b0}}}{y_{b0}} - \frac{\dot{\Delta P_0}}{\Delta P_0} \quad (3)$$

Furthermore, from (2) we find that the catcher must also respect:

$$z_{bc}(t) = H \cdot x_{bc}(t) \cdot t \text{ , } H = \frac{\dot{z_{bc0}}}{x_{bc0}} \quad (4)$$

Note that equation (3) yields a circle of points with radius ΔP, and equation (4) yields a line. This makes intuitive sense — a certain elevation angle can be achieved from anywhere on a circle around the ball and a certain rotation angle can be obtained from anywhere on a line intersecting the ball. Therefore, we have the system of equations

$$\begin{cases} \Delta P^2(t) = z_{bc}^2(t) + x_{bc}^2(t) \\ z_{bc}(t) = H \cdot x_{bc}(t) \cdot t \\ y_{bc} = y_b - y_c \\ x_{bc} = x_b - x_c \end{cases}$$

Knowing $P_b = (x_b, 0, z_b)$, we can find the point $P_c = (x_c, 0, z_c)$, i.e. the correct position of the ideal catcher.

3.6.2 Linear Optical Trajectory

The LOT strategy implements the strong LOT model as described in the paper by Aboufadel (1996) which is based on the original paper by McBeath et al. (1995). For a much more in-depth discussion on the mathematics behind this implementation, reference Aboufadel's work.

Let $P_b = (x_b, y_b, z_b)$ be the position of the ball and $P_c = (x_c, 0, z_c)$ the position of the catcher at a time t. Then let $P_i = (x_i, y_i, z_i)$ be the image of the ball from the catcher's perspective, as shown in diagram 2. Thus, we define two functions of time

$$p = \frac{z_i}{x_i} \text{ and } q = \frac{y_i}{x_i}$$

Therefore, the tangent of ψ can be expressed as

$$\tan\psi = \frac{y_i^2}{x_i^2 + z_i^2} = \frac{q^2}{1+p^2}$$

The strong LOT model posits that the catcher follows a path that keeps both p and q constant. Considering $t = 0$ at the initial response time of the catcher, for any time $t > 0$ and until the ball either hits the ground or is caught, p and q are found using

$$p = \frac{x_b - x_c}{z_c - z_b} \quad (5)$$

$$q = \left(\frac{z_b}{x_b + pz_b}\right)\left(\frac{x_c(p^2+1) - (x_b + pz_b)}{x_c - x_b}\right)$$

By fixing p and q at the initial time, x_c and z_c can be found from

$$x_c = \frac{(y_b - qx_b)(x_b + pz_b)}{y_b(p^2+1) - q(x_b + pz_b)}$$

$$z_c = \frac{(py_b - qz_b)(x_b + pz_b)}{y_b(p^2+1) - q(x_b + pz_b)}$$

which yield the position of the catcher P_c.

A modification was made to the mathematical model by Aboufadel that allows it to function correctly inside the environment. Unity uses a rather strange (although not unmotivated) axis system, with x and z denoting the horizontal plane and y denoting the height, as may have become clear from the previous section. Hence, the coordinate system used in Aboufadel's implementation can be obtained by simply interchanging y and z.

3.7 Conclusions and Observations

There is an argument to be made that, in some ways, the user may have too much liberty when configuring the environment. It should be noted that the process of developing this tool was governed by the assumption that the user is a researcher attempting to push the limits of human perception in order to draw pertinent conclusions. This implies that the user is not actively trying to "break" the application.

Because of that, it has been assumed that the user will look to configure the environment with stability and reliability in mind and in such a way that produces repeatable results, hence, it has been assumed the user will test their specific configuration thoroughly before attempting to run the experiment with their participants. That being said, there is the possibility that the user may not be aware of certain implementation details that may interact with their configuration, therefore some precautions were taken to minimise the risk of user error in such circumstances.

Firstly, the conscious decision was made *not* to provide the custom script with access to critical components of the environment, such as the data collection, input management or trial management components.

Those components, in particular, were *not* designed to be directly manipulated at runtime in unexpected ways, despite the fact that, for example, manipulating the user input during a trial might produce interesting results.

Furthermore, as mentioned in the Configuration File section, although the maximum accepted values for configured variables far exceed the realm of practicality in most instances, care has been taken as to not allow values that would not produce intended functionality, but also values that would produce unexpected results. For example, Unity tends to be somewhat inconsistent with its handling of negative values in variables related to the *size* of objects. Setting the *scale* variable of the ball's transform to 1 or -1 produces the same result (for the purposes of this application), but the *size* of the ball's sphere collider can only be a *positive* value. For this reason, the application also enforces positive values only for the scale of the ball.

One other decision should be addressed: the choice to move away from the Wii Balance Board input method in favour of FOVE control. As mentioned previously, the initial plans were to use the Wii Board for controlling the virtual catcher. Although the control is less intuitive, it still has a big advantage over the FOVE control, in that the motion control is decoupled from the head position, and consequently eye movement.

The issue here is that the FOVE control method will cause interference in the eye-tracking data, since the subject needs to move their head not only to gather information about the ball, but also to reposition themselves inside the virtual environment. Due to this, subjects would often be leaning their head to a side while tracking the ball, which evidently affects the eye-tracking data.

However, the source of this interference is completely and accurately determined, since all orientation and position data of the headset, global as well as relative to the subject's neutral position, is available at any time. Therefore, it may very well be possible to remove this interference when analysing the data, depending on how the data is going to be used. Nevertheless, this problem is acknowledged and this is why the environment provides means of using any preferred input method other than the FOVE control via a virtual joystick.

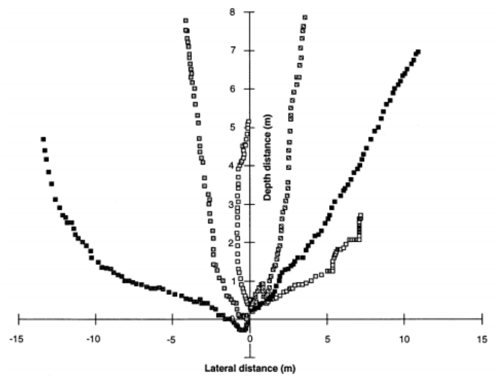
4 Evaluation and Testing

This section will present a short experiment meant to qualitatively compare the paths chosen by subjects inside the virtual environment with the data obtained by McBeath et al. (1995) when observing real outfielders and discuss the verbal feedback from the participants regarding the overall experience, as well as evaluating other unstructured tests that were run before the experiment. The observed limitations of the system will also be discussed.

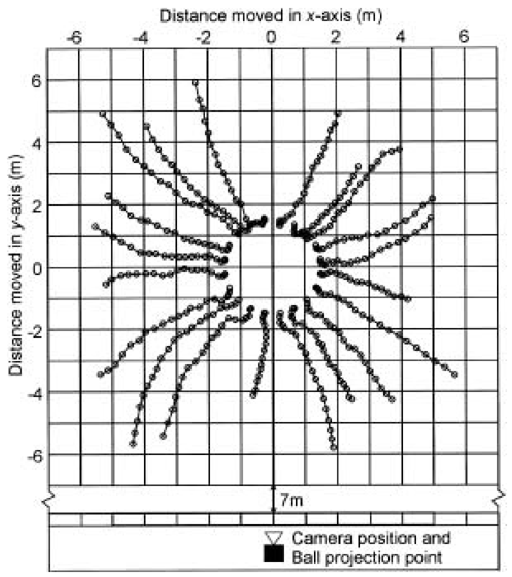
As mentioned previously, the ecological validity of the testing environment, otherwise put, how much can the *virtual* environment be trusted to provide information about the *real* task, is not something that can easily be tested.

Although it is a fact that a virtual environment will never be able to perfectly replicate the task of a real outfielder (or any task, for that matter), an argument can be made that the same *strategies* are used on a real field as well as in a virtual setting. Therefore, the behaviour of an outfielder should be comparable *qualitatively* to the motion of a subject in a virtual environment.

In order to test this assumption, a short experiment was run to qualitatively compare a subject's motion inside the virtual environment to the motion of a real outfielder. The data obtained by McBeath et al. (1995) was employed as an example of real outfielder trajectories.



(a) The data set used as an example of real outfielder trajectories, extracted from the paper by McBeath et al. (1995)



(b) The other option considered, extracted from the paper by McLeod et al. (2001)

Finding examples of real outfielder trajectories in a usable format proved to be very difficult. Even if the only collected data the paper provides is one single graph shown in figure 14a, it is still the most appropriate piece of data that could be found at the time of writing this report. One other paper was considered, by McLeod et al. (2001) (data shown in 14b), which does provide more data than the chosen paper, however, the distance between the catcher and ball's landing point was, on average, about 5 meters. During previous tests, participants seemed to exhibit approximately linear trajectories on such short distances, which is also the case with most of the paths presented in that paper, and this would not have provided much information about the true behaviour of the system.

4.1 Experiment

Task In order to replicate the ball trajectories from the paper, the test cases were configured to land at the targets extracted from the graph above, as specified in table 5. Apart from the 6 tests replicating the original task, 6 additional tests were configured to land at the 6 symmetric points with respect to the x-axis. There was no information about each individual ball launch in the paper, but another graph suggests the

	I	II	III	IV	V	VI	I*	II*	III*	IV*	V*	VI*
Target X	45.35	42.3	44.81	42.13	43.03	47.25	45.35	42.3	44.81	42.13	43.03	47.25
Target Z	-13.3	-4.11	0	3.62	10.91	7.18	13.3	4.11	0	-3.62	-10.91	-7.18

Table 5: Configuration of the test cases

average height reached by the ball was around 30 ± 10 meters, so the balls were configured to reach a height of 30 meters. Each participant went through the tests twice for practice before four trial runs. The virtual catcher was controlled using the FOVE controller.

Setup The environment was run on a Windows 10 machine with an Intel Core i7-6700K @ 4.00Ghz processor and a GeForce GTX 1060 6GB GPU. The subjects controlled the virtual catcher by leaning in the direction they wished to move, with a set maximum speed of 8.5m/s, while sat in a chair approximately 1 meter in front of the infrared position tracker. The FOVE headset is vertically synchronised, therefore the screen refreshed at a constant 70Hz. The ball was considered "caught" if the subject came within 1m of it (i.e. the hitbox was configured with size (2,2,2)). Data was collected at 100Hz and all of the participants provided informed consent.

Participants Four participants took part in the experiment, none of which had any experience playing baseball, but one participant had considerable experience playing badminton. Another participant suffered from myopia and did not wear glasses or contact lenses for the experiment. The results from each participant will be discussed independently due to the small amount of data, as well as significant variations in behaviour from one participant to the next.

General conclusions and feedback The paths of real outfielders always exhibit some curvature. Whether the path curves towards or away from the home plate seems to depends on how confident the catcher is with their chances of catching the ball, as well as personal preferences and habits, as also observed by Shaffer and McBeath (2002), but most of the time, their path would curve away from the home plate.

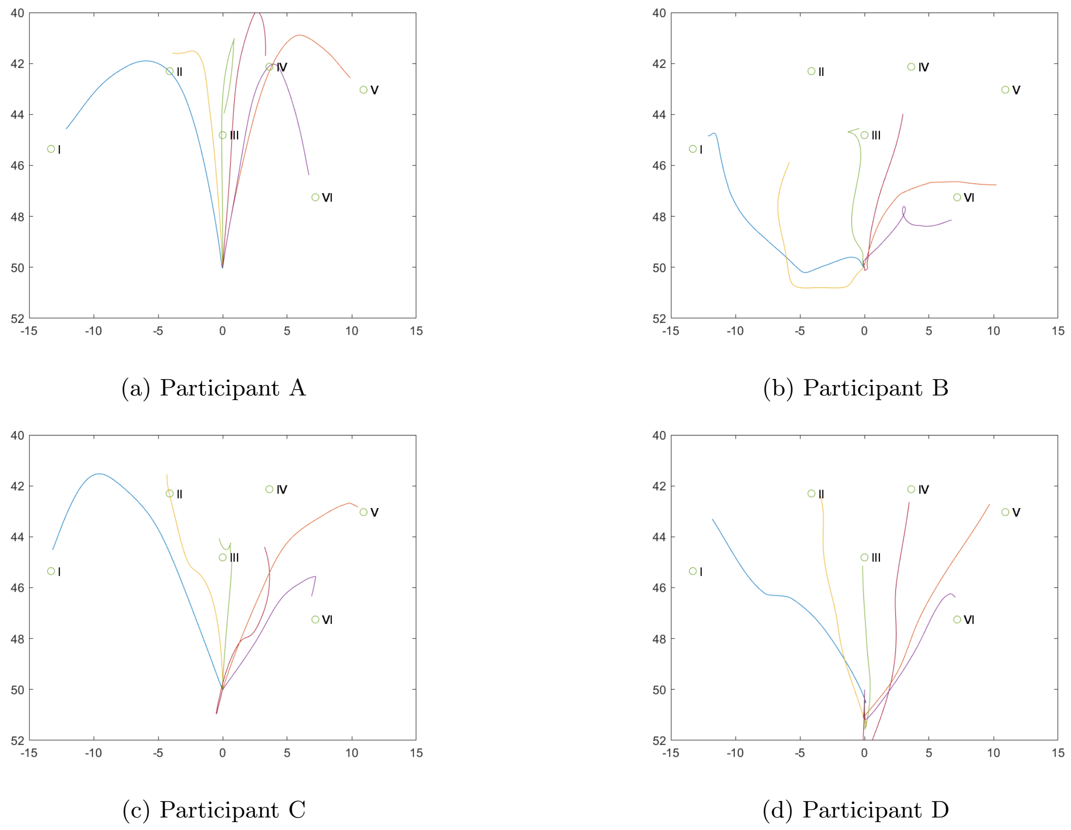
This is observed for all participants in most trials, but participants were observed to switch from an outward curving pattern to an inward curving pattern as they progressed through the trials and gained more experience with the environment. This observation is discussed thoroughly later in this section.

None of the participants reported feeling motion sick and mentioned that the control method was intuitive and easy to get used to. However, all of them mentioned that the ball would appear to accelerate sideways during the last portion of its flight, which would not actually happen. This sensation can be explained by the fact that the subjects can accelerate quite abruptly inside the virtual reality with little physical feedback (little, because the leaning motion does provide some feedback), therefore it was not the ball accelerating sideways, but the subjects not realising they were, in fact, accelerating. In reality, a catcher would be able to gauge their own acceleration and velocity quite accurately and would intuitively *expect* the ball to appear to curve sideways. All of the participants reported they only observed this effect once or twice, which could mean they learned to adjust to this effect after the first time they encountered it.

It should be noted that, as was to be expected, some subjects found it easier to control the virtual catcher than others. This may be connected to their experience with video games and the abstraction of motion control, but more data is needed to draw this conclusion.

Participant A This participant has had extensive experience playing badminton, but also, perhaps more importantly, this experiment is the 3rd time this participant has interacted with this environment. This is most likely the reason their catching technique is very consistent using this specific input method, and also why their trajectories do not match the paths of a real outfielder in this experiment, neither do they match the behaviour of the other participants. As a result of their experience, they managed to catch the ball in 100% of the trials. Similar behaviour is exhibited by Participant C, although to a lesser degree.

Participant C This experiment was the 2nd time this participant has interacted with the virtual environment. This further illustrates the progression from normal behaviour, comparable to a real outfielder, to a behaviour adapted to the specifics of the motion control implemented in this environment. This behaviour is less accentuated than Participant A, but still evident from the paths chosen. They still exhibit normal behaviour for some of the tests that are launched more towards the catcher's initial position (II, III and IV), but it is reasonable to assume that with enough practice, they would eventually replicate Participant A's technique for all tests. They achieved a success rate of 95%, catching the ball 23/24 times.



(a) Participant A

(b) Participant B

(c) Participant C

(d) Participant D

Figure 15: Average paths for the four participants, across four trial runs each. The labelled circles mark the landing point of the ball for each trial. Note that in some instances, the subjects failed to return to a neutral position in the pause between trials, which caused the "zero-position" of the controller to be set incorrectly, leading to an inability of the subject to properly control their motion for that trial. This would lead to the virtual catcher speeding off uncontrollably toward the edges of the field, which is why some of the paths appear skewed and far from the ball's landing.

Participant B The second participant had a much harder time getting used to the motion control. For about half of the trials, they could not confidently control the virtual catcher, but they eventually learned to control it towards the end of the experiment. An example like this was to be expected. Although many people seem to adjust to the input method after 5-6 practice trials, some people may need more time to practice. As a direct result of this, they only managed to catch the ball approximately 46% of time, or 11/24 trials. However, they did seem to exhibit natural behaviour for successful catches, most clearly observed for test I.

Participant D The last participant suffered from myopia and took part in the experiment without wearing glasses or contact lenses. Expectedly, they reported not being able to see the ball for the first half of its flight but managed to move in time to catch it an impressive 83% of the time (or 20/24 times). Because of this, they were forced to adopt more linear trajectories in order to reach the landing point in time to catch the ball, although a very slight outward curvature is exhibited for all test cases.

The next few paragraphs discuss the issue of the participants' behaviour changing throughout the experiment after gaining more experience with the system. The source of the problem is most likely the fact that subjects can move backwards just as fast as they can move forwards. Almost all of the participants seem to naturally realise this at some point during the experiment and they started following paths similar to the paths exhibited by Participant A, who has had more experience with the environment, namely by moving towards the ball's plane until the ball was above them and follow it backwards to its destination.

Of course, on a real field, a catcher would never be able to do this, since they could never move backwards fast enough to keep up with the ball going above their head. Furthermore, another issue was the maximum speed the participants could attain in the environment. A maximum speed of 8.5m/s is an average speed for a sprinting outfielder, but the participants seemed to maintain maximum speed for most of the duration of the trial, even at the moment of catching the ball.

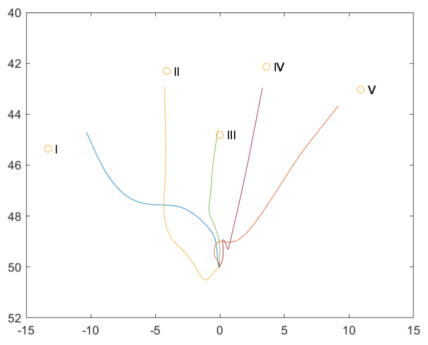
Therefore, a solution is immediately apparent: the environment should be modified to respect the maximum speed of an outfielder when moving backwards, as well as limit the maximum acceleration of the virtual catcher since, at the moment, the acceleration is only limited by how quickly the subject can change their position.

However, as previously mentioned, all participants do initially behave naturally. Hence, after resolving the motion control issue and by carefully configuring the length of the experiment, participants can be expected to qualitatively replicate the movement of real outfielders.

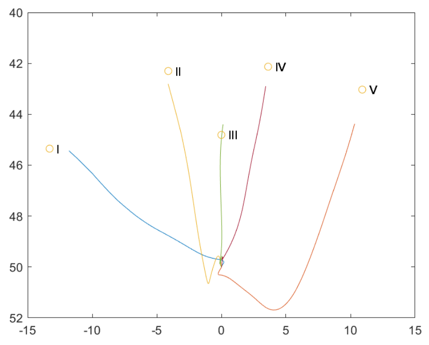
One more detail worth mentioning is that in test III, where the ball was headed straight towards the catcher, all participants initially moved sideways, similarly to how a real outfielder may move, perhaps in an attempt to gather more visual information about the trajectory of the ball.

4.2 Excluded data

Data from four other participants was excluded from the experiment due to an unforeseen failure. Unfortunately, due to an unnoticed bug in the trial management code, four of the participants did not execute each test the correct number of times, in some cases skipping some tests altogether. Furthermore, test VI and its symmetrical counterpart were both missing from the experiment. However, the environment itself, as well as the system setup described for the experiment, were otherwise identical, so there is some information to be gained by look over this data. Only the data from two out of four subjects could be salvaged.



(a) Participant E*



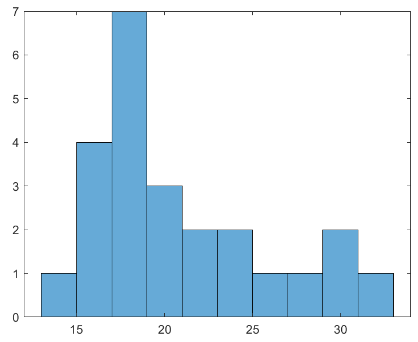
(b) Participant F*

Figure 16: Due to the bug mentioned earlier, for these participants some tests were run only once, while some were run multiple times. With the exception of test I, which was run 11 times for one participant and 15 times for other, tests were run a random number of times between 1 and 4.

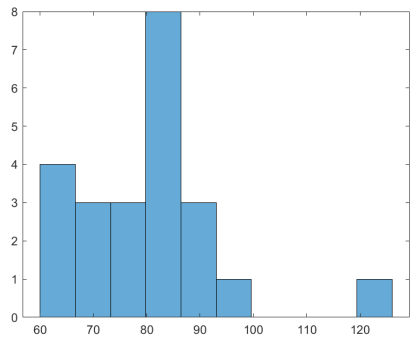
Both subjects were interacting with the environment for the first time, had no considerable sports experience and had normal or corrected to normal vision. Looking at the trajectories taken by the subjects, the curvature is clearly present for all the tests. Similarly to the previous participants, they also reported the appearance of the ball accelerating sideways. Participants E* and F* achieved a success rate of 66% (16/24) and 74% (20/27) respectively.

4.3 Eye tracking

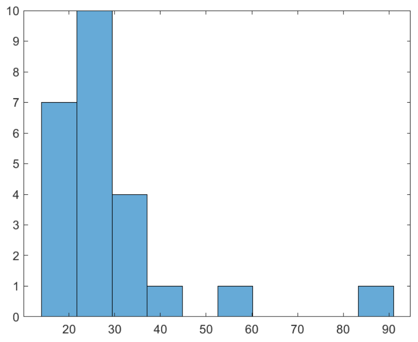
During the experiment, the red dot that shows the user's gaze appeared to be visibly off-target for most of the experiment, under the assumption that the subject would be looking at the ball most of the time, which can also be observed in the example footage provided. For this reason, the eye-tracking data from the experiment was checked for accuracy.



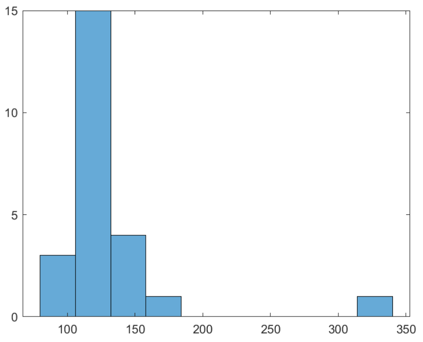
(a) Participant A



(b) Participant B



(c) Participant C



(d) Participant D

Figure 17: Histograms of the average distance per trial between the subject's gaze and the ball position. One extreme outlier (distance > 400 pixels) was removed from the histogram for Participant B, which occurred when the participant asked for a break and they removed their headset towards the end of a trial.

Perhaps somewhat predictably, the average distance between Participant D's gaze and the ball is very large (upwards of 100 pixels for every trial), but this information is not very useful since they did report not being able to see the ball for a large part of its flight. Manually examining the data files, the error seemed to hover around 70 pixels for periods when the participant actively tracked the ball.

Participants A, B and C tracked the ball throughout the trial. The histograms show that the error of the eye tracking system is quite significant, usually around 30 pixels for Participants A and C, but may be as dramatic as 80-90 pixels for Participant B. Participant B did not report suffering from any condition that would affect their sight, therefore the eye tracking calibration may have been performed poorly.

The calibration is handled by the FOVE interface and the only information available to the user is whether the calibration is running and whether is finished successfully. What exactly "successfully" means is impossible to know. Therefore, the risk of participants not performing the calibration properly does exist. As such, one very simple solution to safeguard against this would be to introduce a small moving object inside the "white room" where the virtual catcher is stationed before the start of the experiment, so that the user may verify, with the aid of the red dot displaying the subject's gaze, whether the results of the calibration are acceptable.

At the same time, it may not be a problem of poor execution of the calibration process, but simply that the eye tracking functions better for some people than others, depending on interpupillary distance, eye colour, etc.

Furthermore, it seems it is not simply a drift or scaling issue, so a simple transform cannot be applied to fix the problem. It turns out that the FOVE API for Unity is quite infamous for its poor eye-tracking capabilities, mostly due to the fact that it is outdated, and has not received any updates in the past year (last update, as of writing this report: 2018-03-07), with the C++ API seeing an update as recently as February 3rd of this year.

As things currently stand, it seems there is no solution immediately available to fix this issue on the side of the application, since the FOVE API does not expose the mechanisms involved in collecting and correcting the eye-tracking information to the developer. However, it may be possible to find which variables cause an increase in error of the eye tracking system and use it to "fix" the data after it has been collected.

4.4 Conclusion

Following the experiment, one critical issue has been found — insufficient restrictions on the subject's motion in the virtual environment eventually lead to the subject changing from natural behaviour to a strategy that takes advantage of the motion control system in a way that does not reflect the behaviour of a real outfielder.

However, considering the fact that prior to discovering this strategy, intuition drives the subjects to move in a way that is qualitatively consistent with real-life behaviour, the environment, at its core, has proven to be a valid tool for studying the task of the outfielder, provided the issue described above is addressed.

Another problem is presented by the low (and inconsistent) *accuracy* of the eye-tracking system. Although precise, gaze data was found to be less reliable than expected and the usability of this data is largely dependant on the goals of the experiment. Therefore, thanks to the somewhat commendable *precision* of the system, the data could be used to draw qualitative conclusions, but any numeric analysis should be performed with caution.

Finally, the system does not currently offer a way for the user to verify that the calibration was performed correctly before running the experiment, possibly in the form of a small moving object in the virtual pre-experiment area which can be used to visually assess the effectiveness of the eye-tracking, thus minimising the risk of miscalibration by the subject.

5 Limitations and Further work

This section will reiterate and expand on the problems and insufficiencies of this application and discuss potential solutions. First of all, the critical issues need to be addressed.

During testing, all subjects were observed to initially follow a trajectory qualitatively similar to a real outfielder's, but some subjects would switch to an unnatural trajectory after longer periods of time spent inside the virtual environment, preferring to move forwards and into the plane of the ball and position themselves approximately under the ball then move backwards with the ball until it was caught. One participant, after having previously interacted twice with the virtual environment, had become very accustomed to this strategy and behaved completely differently from a real outfielder in the experiment.

The problem may be that the way input is translated into motion inside the virtual environment is lacking in its current implementation. The current system does not limit maximum speed depending on orientation, i.e. the catcher can move backwards as fast as they can move forwards. Hence, an obvious and necessary solution would be to implement a (preferably configurable) way of limiting the speed of the catcher when moving backwards of to the sides, hence prohibiting the subjects from behaving unnaturally.

The other critical problem concerns the quality of the eye-tracking data. As discussed in the previous section, eye-tracking data is provided by the FOVE interface and the API does not provide any way of interacting with the calibration process. One way to work around this would be to try to find which variables produce an increase in the error of the eye-tracking and adjust the recorded data after the experiment.

When discussing this with a fellow student also struggling with this issue, he mentioned that he found the orientation of the headset to affect the eye-tracking data, with increasingly larger errors for a larger change in the headset's deviation from the initial orientation. It may be possible to find a relationship between the two and use it to reduce the error, but this is still only an idea and a much deeper analysis of this problem needs to be performed.

The source of the problem is most likely the outdated API provided for the Unity engine. As such, the optimal solution would be to recreate the virtual environment using OpenGL or a similar graphics library and use the C/C++ API for FOVE, which is more reliable and up-to-date. This is in fact not that big of a task since all of the code was written in C# which can be integrated quite easily with a C++ application. Furthermore, the visual component of the environment is simple by design and would be easy to replicate using a graphics library.

On the topic of the simple design of the visual component: currently, the environment does not provide any means of configuring the complexity of the scenery. It would be interesting to see how the behaviour of a subject changes once elements from the scenery interfere with the subject's tracking of the ball, such as moving objects in the background or even loud noises. Also related, a minor limitation may be the fact that the virtual "outfield" is fixed in size. This limits the initial maximum distance between the catcher and the ball to about 200 meters. Although more than enough for the task of the outfielder, some experiment configurations may benefit from a larger field.

Lastly, and perhaps most importantly in terms of further work, this tool should actually be used to study the Outfielder Problem. Plans for this were already made with the supervisor and an experiment will hopefully be run over the summer.

6 Conclusion

This report presented a virtual reality tool which can be used to study the Outfielder Problem, a case of the problem of interception in psychology. An overview was given of the current competing solutions with arguments for and against each of them, as well as similar attempts at using a virtual environment to study this task.

The key strength of this tool is its high degree of configurability, which allows a user to construct custom experiments and monitor the behaviour of the subjects inside the virtual environment. The tool also implements two of the proposed solutions for the Outfielder Problem in order to allow the user to test the subjects' resulting trajectories against their predictions and draw conclusions in favour of one or the other.

Following a short test, it has been shown that the environment can reproduce the task of the outfielder in such a way that subjects initially behave similarly to real outfielders. However, subjects appeared to adopt an unnatural behaviour after longer periods of time spent within the environment. A potential cause of this problem has been determined to be a flaw in the motion control system and a solution was proposed.

7 Reflections

Developing a tool for scientific research has been one of the most interesting and diverse tasks I have ever tackled. Throughout the development of this tool, I had the opportunity to apply my knowledge of Computer Science as well as Mathematics in the field of Sports Sciences, which I had never encountered before.

At every step I had to keep in mind the interests of the user, a researcher, most likely an expert on the topic of the visual control of action, who may need the environment to perform in ways I had not considered previously, as well as the correctness of the environment, since the virtual environment must replicate the task of the outfielder reduced to its most basic elements, but at the same time it must not require any less from the subject. Walking this thin line meant always adding and removing parts from the virtual environment as I researched the topic further and found new things.

Thanks to this I now know that the study of human behaviour is a treacherous subject: I would often find my assumptions to be not only inaccurate but at times completely incorrect. This taught me a very valuable, practical lesson — always thoroughly study the problem I am trying to solve before attempting to find a solution. This is, of course, something I have always known, but I only realised the true importance of this lesson after tackling a project of this scale.

At one point it had become hard to accept the fact this project cannot be everything it could be — after all, I am just one person and the time I have is limited. It was not very easy for me to get over this. In the weeks before the demonstration, I tried to squeeze in as many features as I could, to make sure I had something impressive to show for the actual presentation, but, of course, rushing through the structural elements of the application meant I had to rework most of the code I wrote in that period. This taught me the value of a solid framework and had I done this from the beginning, I could have saved a lot of time.

Looking back on this project, if I were tasked with building this tool again, I would make sure to test it on people other than myself periodically throughout the development process. The experiment presented in section 4 was really the only time a reasonable number of people tested one iteration of the system. The glaring issue that was discovered regarding the motion control system could have been found earlier, but I did not notice any such behaviour with myself nor with the one or two other friends who I had asked to come in and try it. I know now that people can behave in strange and unexpected ways when adapting to things that are new to them, and this taught me not to trust my own uninformed assumptions as much as I have during this project.

It has been a long, stressful year, and I can comfortably say this project has been the largest piece of work I have ever undertaken, but it also marked the first time I was asked to present something I was truly proud of. In the end, I do believe I can say I am proud of my work, and once the highlighted issues are addressed, I am very eager to deliver this tool to the School of Sports Science and work with them study the problem of the outfielder.

References

A. Savkin, P., Saito, S., Jarich, V., Fukusato, T., Wilson, L. and Morishima, S. (2017), ‘Outside-in monocular ir camera based hmd pose estimation via geometric optimization’, pp. 1–9.

Aboufadel, E. (1996), ‘A mathematician catches a baseball’, *The American Mathematical Monthly* **103**(10), 870–878.

Babler, T. G. and Dannemiller, J. L. (1993), ‘Role of image acceleration in judging landing location of free-falling projectiles’, *Journal of Experimental Psychology* **19**(1), 15–31.

Belousov, B., Neumann, G., Rothkopf, C. A. and Peters, J. (2016), ‘Catching heuristics are optimal control policies’, *Advances in Neural Information Processing Systems* pp. 1426–1434.

Bennett, S., Ashford, D. and Elliott, D. (2003), ‘Intermittent vision and one-handed catching: The temporal limits of binocular and monocular integration’, *Motor Control* **7**, 378–387.

Bennett, S., Elliott, D., Weeks, D. and Keil, D. (2003), ‘The effects of intermittent vision on prehension under binocular and monocular viewing’, *Motor Control* **7**, 46–56.

Calderone, J. B. and Kaiser, M. K. (1989), ‘Visual acceleration detection: Effect of sign and motion orientation’, *Perception & Psychophysics* **45**(5), 391–394.

Chapman, S. (1968), ‘Catching a baseball’, *American Journal of Physics* **36**(10), 868–870.

Conway, M. A. (1991), ‘In defense of everyday memory.’, *American Psychologist* **46**(1), 19–26.

Cutting, J. E. and Vishton, P. (1995), ‘Perceiving layout and knowing distances: The integration, relative potency, and contextual use of different information about depth’, *Perception of space and motion* pp. 69–117.

de la Malla, C. and Lopez-Moliner, J. (2015), ‘Predictive plus online visual information optimizes temporal precision in interception’, *Journal of Experimental Psychology: Human Perception and Performance* **41**(5), 1271–1280.

Diaz, G. J., Phillips, F. and Fajen, B. R. (2009), ‘Intercepting moving targets: a little foresight helps a lot’, *Exp Brain Res* **195**, 345–360.

Fink, P. W., Foo, P. S. and Warren, W. H. (2009), ‘Catching fly balls in virtual reality: a critical test of the outfielder problem’, *Journal of Vision* **9**(13).

Loomis, J., DaSilva, J., Fujita, N. and Fukusima, S. (1992), ‘Visual space perception and visually directed action’, *Journal of Experimental Psychology* **18**, 906–921.

McBeath, M. K., Shaffer, D. M. and Kaiser, M. K. (1995), ‘How baseball outfielders determine where to run to catch fly balls’, *Science* **268**, 569–573.

McLeod, P. and Dienes, Z. (1996), ‘The generalized optic acceleration cancellation theory of catching’, *Journal of Experimental Psychology* **22**, 531—543.

McLeod, P., Reed, N. and Dienes, Z. (2001), ‘Toward a unified fielder theory: What we do not yet know about how people run to catch a ball.’, *Journal of Experimental Psychology: Human Perception & Performance* **27**(6), 1347–1355.

McLeod, P., Reed, N. and Dienes, Z. (2006), ‘The generalized optic acceleration cancellation theory of catching’, *Journal of Experimental Psychology* **32**, 139–148.

McLeod, P., Reed, N., Gilson, S. and Glennerster, A. (2008), ‘How soccer players head the ball: A test of optic acceleration cancellation theory with virtual reality’, *Vision Research* **48**(13), 1479 – 1487.

Michaels, C. F. and Oudejans, R. D. (1992), 'The optics and actions of catching fly balls: Zeroing out optical acceleration', *Ecological Psychology* **4**, 199—222.

Neuman, J. (2015), 'mcs-ICodeCompiler', `https://github.com/aeroson/mcs-ICodeCompiler`. Accessed: 12 April 2019.

Philbeck, J. (2000), 'Visually directed walking to briefly glimpsed targets is not biased toward fixation location', *Perception* **29**, 259–272.

Rieser, J., Ashmead, D., Talor, C. and Youngquist, G. (1990), 'Visual perception and the guidance of locomotion without vision to previously seen targets', *Perception* **19**, 675–689.

Rozendaal, L. and van Soest (2003), 'Optical acceleration cancellation: a viable interception strategy?', *Biological Cybernetics* **89**(6), 415–425.

Saxberg, B. V. (1987), 'Projected free fall trajectories', *Biological Cybernetics* **56**(2–3), 159—-175.

Schmerler, J. (1976), 'The visual perception of accelerated motion', *Perception* **5**(2), 167–185.

Shaffer, D. M. and McBeath, M. (2002), 'Baseball outfielders maintain a linear optical trajectory when tracking uncatchable fly balls', *Journal of Experimental Psychology: Human Perception & Performance* **28**(2), 335–348.

Shaffer, D. M. and McBeath, M. K. (2005), 'Naive beliefs in baseball: Systematic distortions in perceived time of apex for fly balls', *Journal of Experimental Psychology* **31**, 1492—-1501.

Zaal, F. T. J. M. and J.Bootsma, R. (2011), 'Virtual reality as a tool for the study of perception-action: The case of running to catch fly balls', *PRESENCE: Teleoperators and Virtual Environments* **20**(1), 93–103.

Zaal, F. T. and Michaels, C. F. (2003), 'The information for catching fly balls: Judging and intercepting virtual balls in a cave', *Journal of Experimental Psychology. Human Perception & Performance* **29**(3), 537.

Zhao, H. and H.Warren, W. (2015), 'On-line and model-based approaches to the visual control of action', *Vision Research* **110**, 190–202.

A Configurable Variables

Name	Unit	Type	Default	Range
`auto`	˜	`bool`	false	`true, false`
`number_of_tests`	˜	`int`	8	$[0, \infty)$
`practice_runs`	˜	`int`	3	$[0, \infty]$
`trial_runs`	˜	`int`	10	$(0, \infty]$
`max_ball_height`	m	`float`	20	$(0, \infty]$
`radius`	m	`float`	8	$(0, \infty]$
`targets_shape`	˜	`int`	0	$(0, 1]$
`pause_between_trials`	s	`float`	2	$(0, \infty]$
`max_speed`	m/s	`float`	5	$(0, \infty]$
`starting_distance`	m	`float`	30	$(0, 250)$
`ball_size`	m	`float`	0.1	$(0, \infty]$
`ball_mass`	kg	`float`	0.14	$(0, \infty]$
`ball_drag`	none	`float`	0.01	$(0, 1)$
`ball_preset`	˜	`int`	0	$[0, 5]$
`controller_type`	˜	`int`	0	$[0, 2]$
`input_smoothing`	˜	`bool`	false	`true, false`
`smoothing_amount`	ms	`int`	30	$[1, \infty]$
`input_curve`	˜	`int`	0	$[0, 3]$
`curve_parameter`	none	`float`	1	$(0, \infty]$
`lot_start_time`	s	`float`	0.5	$[0, \infty]$
`goac_start_time`	s	`float`	0.5	$[0, \infty]$
`skybox_on`	˜	`bool`	false	`true | false`
`light_rotation`	deg	Vector3	(75, -90, 0)	$x, y, z \in (-360, 360)$
`light_color`	RGB	Vector3	(255, 244, 214)	$x, y, z \in [0, 255]$
`hitbox_size`	m	Vector3	(2, 3, 3)	$x, y, z \in (0, \infty)$
`skybox_color`	RGB	Vector3	(0, 149, 255)	$x, y, z \in [0, 255]$

Note: In this table, ∞ refers to the maximum value the specified data type can hold, i.e. $2^31 - 1$ for integer values and approximately $3.4028235 \cdot 10^38$ for floating point values. Also, the Vector3 class stores 3 `float`s, so the same rule applies.

Name	Description
`auto`	Set auto or manual mode
`number_of_tests`	The number of test cases.
`practice_runs`	The number of times to run each test as practice.
`trial_runs`	The number of times to run each test as a trial.
`max_ball_height`	The maximum height reached by the ball in generated tests, in meters
`radius`	The radius of the selected shape, in meters
`targets_shape`	The shape of the targets for the generated test cases.
`pause_between_trials`	Time to pause between consecutive trials in seconds.
`max_speed`	Maximum speed attainable by the catcher, in meters/s.
`starting_distance`	Initial distance between the ball and catcher, in meters.
`ball_size`	Radius of the ball, in meters.
`ball_mass`	The mass of the ball, in kilograms.
`ball_drag`	The drag parameter of the ball. This is NOT a drag coefficient in the ordinary sense. Reference the Unity documentation for more information.
`ball_preset`	Presets for the ball: (0) Custom (1) Baseball, (2) Tennis, (3) Shuttlecock, (4) Football, (5) Basketball. Note that these are simply value presets.
`controller_type`	Type of controller: (0) FOVE, (1) Joystick, (3) Mouse
`input_smoothing`	Whether to smooth the input or not.
`smoothing_amount`	The time window used for smoothing, in miliseconds.
`input_curve`	Type of input curve to be applied: (0) Linear, (1) Sinusoidal, (2) Exponential, (3) Sigmoid. Reference the appendix for the implementation.
`curve_parameter`	The parameter for the curve.
`lot_start_time`	Initial time for the LOT strategy, in seconds.
`goac_start_time`	Initial time for the GOAC strategy, in seconds
`skybox_on`	If true, use a basic skybox, otherwise use a solid color.
`light_rotation`	The rotation of the directional light source, in degrees.
`light_color`	Color of the lighting, in RGB format.
`hitbox_size`	Dimensions for the catcher hitbox, in meters.
`skybox_color`	Color of the skybox, if a solid color, in RGB format.

B Output Data Format

results.csv		
Field	Type	Description
Catch	bool	Result of the trial, represented as either 0 or 1.
Total_time	`float`	Total duration of the trial, in seconds.
Total_frames	`int`	Total number of frames for the trial.

test_list.csv		
Field	Type	Description
No	`int`	The number of this test.
Speed	`float`	The launch speed of the ball
Angle	`float`	The vertical launch angle of the ball.
Deviation	`float`	The horisontal launch angle of the ball.
TargetX	`float`	The point at which the ball will land.
TargetZ	`float`	
MaxHeight	`float`	The maximum Y coordinate attained by the ball.
VelocityX	`float`	The initial velocity of the ball in 3 dimensions.
VelocityY	`float`	
VelocityZ	`float`	

ball_data.csv		
Field	Type	Description
Time	`float`	Time within the current trial.
Frame	`int`	Frame number within the current trial.
Position_X	`float`	Position of the ball at the current time.
Position_Y	`float`	
Position_Z	`float`	
Velocity_X	`float`	Velocity of the ball at the current time.
Velocity_Y	`float`	
Velocity_Z	`float`	

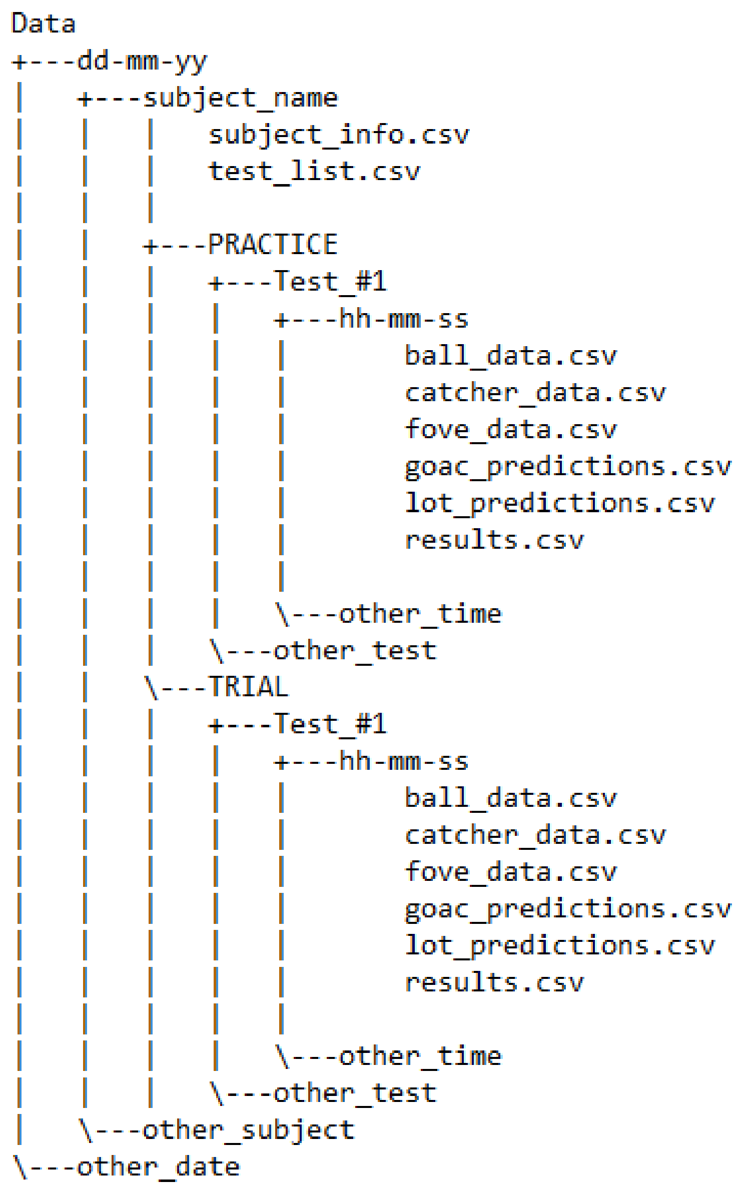
catcher_data.csv		
Field	Type	Description
Time	`float`	Time within the current trial.
Frame	`int`	Frame number within the current trial.
Position_X	`float`	Position of the catcher at the current time. Position_Y will always be 1.
Position_Y	`float`	
Position_Z	`float`	
Velocity_X	`float`	Velocity of the ball at the current time. Velocity_Y will always be 0.
Velocity_Y	`float`	
Velocity_Z	`float`	

goac_predictions.csv		
Field	Type	Description
Time	`float`	Time within the current trial.
Frame	`int`	Frame number within the current trial.
Position_X	`float`	Position predicted by the GOAC strategy.
Position_Y	`float`	
Position_Z	`float`	

lot_predictions.csv		
Field	Type	Description
Time	`float`	Time within the current trial.
Frame	`int`	Frame number within the current trial.
Position_X	`float`	Position predicted by the LOT strategy.
Position_Y	`float`	
Position_Z	`float`	

fove_data.csv		
Field	Type	Description
Time	`float`	Time within the current trial.
Frame	`int`	Frame number within the current trial.
LeftEyeClosed	`bool`	Whether the eye is closed. Represented as either 0 or 1.
RightEyeClosed	`bool`	
BallScreenPosition_X	`float`	The position of the ball in screen coordinates, as observed by the subject. Note: Screen coordinates represent pixels and (0,0) is the bottom left corner.
BallScreenPosition_Y	`float`	
GazeScreenPosition_X	`float`	Where the subject is currently looking, in screen coordinates. Note: Screen coordinates represent pixels and (0,0) is the bottom left corner.
GazeScreenPosition_Y	`float`	
HeadsetPosition_X	`float`	The position of the FOVE headset in world coordinates.
HeadsetPosition_Y	`float`	
HeadsetPosition_Z	`float`	
HeadsetFacing_X	`float`	Unit vector representing the forward direction of the headset.
HeadsetFacing_Y	`float`	
HeadsetFacing_Z	`float`	
LeftEyeVector_X	`float`	The look vector of the left and right eyes respectively. This may be useful if the subject is suffering from any condition which might render gaze data unusable, such as strabismus.
LeftEyeVector_Y	`float`	
LeftEyeVector_Z	`float`	
RightEyeVector_X	`float`	
RightEyeVector_Y	`float`	
RightEyeVector_Z	`float`	
GazeDirection_X	`float`	The direction the subject is currently looking in. This is constructed by the FOVE interface using the left and right eye vectors.
GazeDirection_Y	`float`	
GazeDirection_Z	`float`	
GazePoint_X	`float`	The world point the subject is currently focusing on (i.e. the position on which the left and right eye vectors converge).
GazePoint_Y	`float`	
GazePoint_Z	`float`	
GazeAccuracy	`float`	The accuracy of the gaze information.*

*The FOVE Unity API, as well as its documentation are quite poor. In the case of the `accuracy` variable, the only information available was a post on the support forums, stating that ”accuracy is the amount to which immersion-breaking effects should be allowed to rely on this value [...] In single-eye setups, this will always be zero; if the user makes their eyes diverge, the value will likely be zero”. Furthermore, it seems newer APIs for Python, C and C++ have omitted it entirely. Nevertheless, it was included here just in case anyone found a need for it.



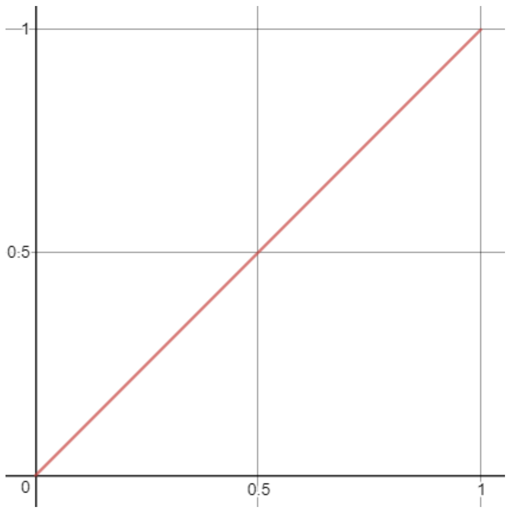
```
Data
+---dd-mm-yy
|   +---subject_name
|   |   |   subject_info.csv
|   |   |   test_list.csv
|   |   |
|   |   +---PRACTICE
|   |   |   +---Test_#1
|   |   |   |   +---hh-mm-ss
|   |   |   |   |       ball_data.csv
|   |   |   |   |       catcher_data.csv
|   |   |   |   |       fove_data.csv
|   |   |   |   |       goac_predictions.csv
|   |   |   |   |       lot_predictions.csv
|   |   |   |   |       results.csv
|   |   |   |   |
|   |   |   |   \---other_time
|   |   |   \---other_test
|   |   \---TRIAL
|   |   |   +---Test_#1
|   |   |   |   +---hh-mm-ss
|   |   |   |   |       ball_data.csv
|   |   |   |   |       catcher_data.csv
|   |   |   |   |       fove_data.csv
|   |   |   |   |       goac_predictions.csv
|   |   |   |   |       lot_predictions.csv
|   |   |   |   |       results.csv
|   |   |   |   |
|   |   |   |   \---other_time
|   |   |   \---other_test
|   \---other_subject
\---other_date
```

Folder structure of the output data.

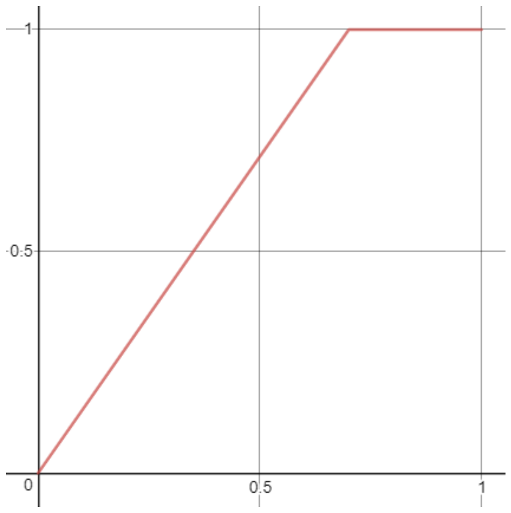
C Input curve functions

Linear

$$y = \begin{cases} \frac{x}{p}, & \text{for } 0 \le x \le p \\ 1, & \text{for } p < x \le 1 \end{cases} \quad , p \in [0,1]$$



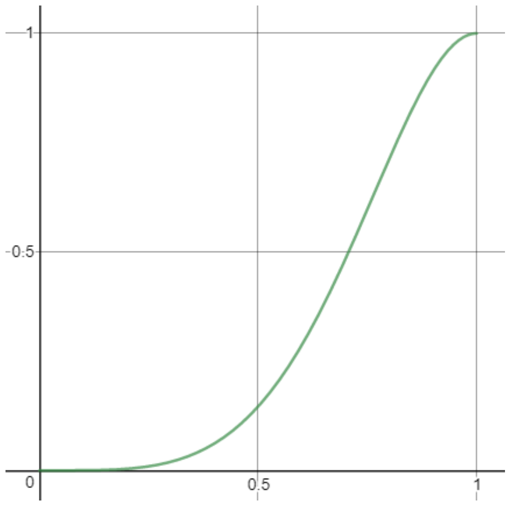
(a) Linear curve with $p = 1$



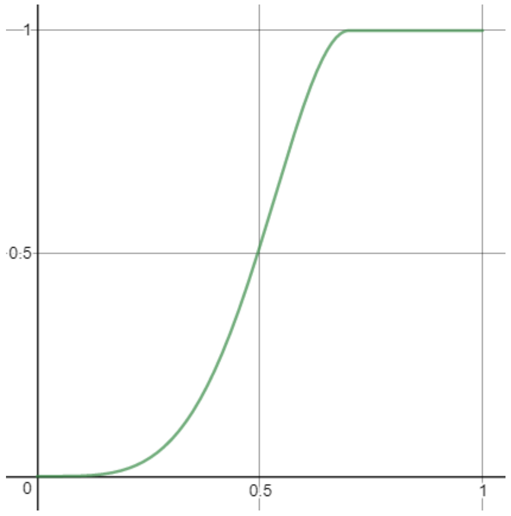
(b) Linear curve with $p = 0.7$

Sinusoidal

$$y = \begin{cases} -\frac{1}{2}\cos\frac{\pi x^2}{p^2} + \frac{1}{2}, & \text{for } 0 \le x \le p \\ 1, & \text{for } p < x \le 1 \end{cases} \quad , p \in [0,1]$$



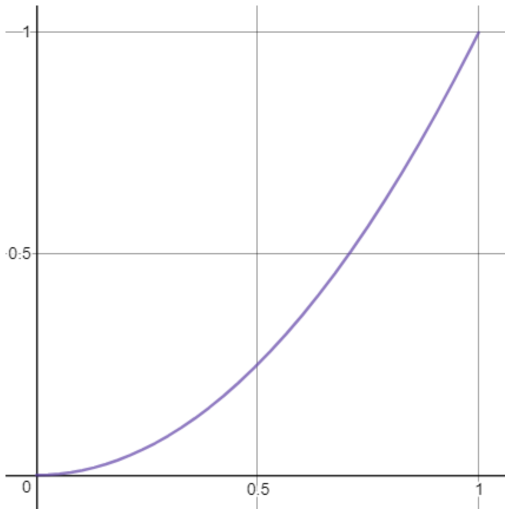
(a) Sinusoidal curve with $p = 1$



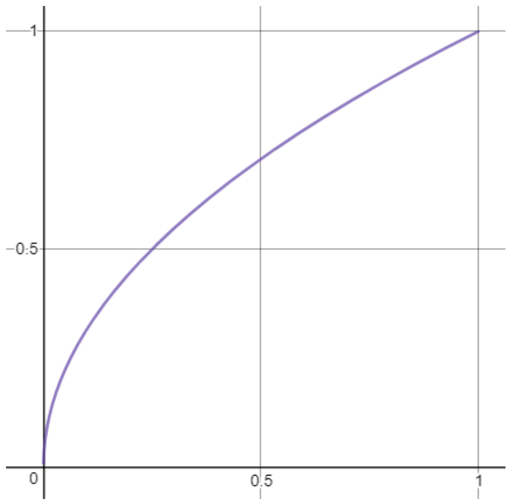
(b) Sinusoidal curve with $p = 0.7$

Exponential

$$y = x^p \ , \ p \in [0, \infty)$$



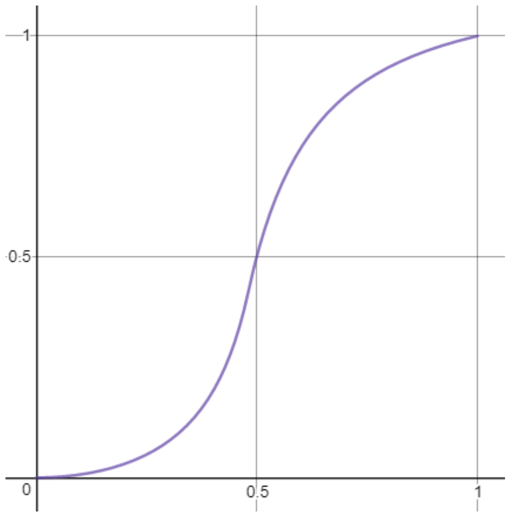
(a) Exponential curve with $p = 2$



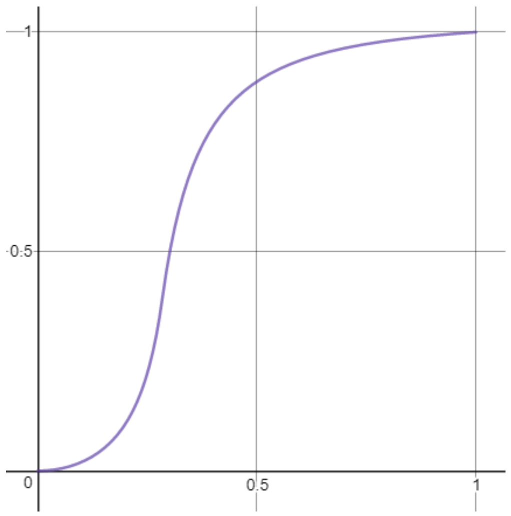
(b) Exponential curve with $p = 0.5$

Sigmoid

$$\frac{1}{k}\left(\frac{0.809(x^2p - 1.618)}{1 + |x^2p - 1.618|} + \frac{1}{2}\right) \text{ where } k = \frac{0.809(p - 1.618)}{1 + |p - 1.618|} + \frac{1}{2} \ , \ p \in [1, \infty)$$



(a) Sigmoid curve with $p = 7$



(b) Sigmoid curve with $p = 0.7$

Note: The ”fuzzy maths” there are just to get the modulus sigmoid function to map $[0, 1]$ to $[0, 1]$. It is not mathematically rigurous, i.e. $y \neq 0$ for $x = 0$ and $y \neq 1$ for $x = 1$ but it is close enough for this application.

Contents of the .zip file and instructions

In these instructions, "/" refers to the root of the contents of the .zip file. A tree view of the folder structure can be found inside the .zip file at /folder_structure.txt

Git repository The /git.txt file contains the link to the git repository. This was transferred from a github repository which was used for version control. You can find a link to that on the gitlab repository.

Running the application A release build of the application can be found in the folder /ReleaseBuild/. From there you can start the application via the executable OutfielderEnvironment.exe. If you would like to configure the environment via a configuration file, you should write a text file as described in the report and save it as a .cfg. If you would like to use the custom script, simply follow the instruction in the comments inside CustomBallScript.cs.

In-app controls Inside the application, to switch between the UI, birds-eye view and subject point of view you can use the num pad 1, 2, and 3 respectively (note that top row 1, 2 and 3 will not work). If they do not work, check that num-lock is on.

Source files All of the source files can be found in the UnityProject folder, under /UnityProject/Scripts/. If you use Microsoft Visual Studio, you should be able to load the entire solution via the solution file at UnityProject/OutfielderEnvironment.sln

Building the project If you would like to build the application yourself, you need to import the project inside Unity. The /UnityProject/ folder is a direct copy of my own project folder. You can open this project in Unity by double-clicking the scene file found in /UnityProject/Assets/Scenes/OutfielderEnvironmentMain.unity

Note: Unity does not really like importing projects from different versions. If you try to load this project in a version other than my own, well, your mileage may vary. I used version 2018.3.1f1, so this guaranteed to work, but any version 2018.3.1x should in theory work just as well. I tried importing my project on the computer in the Sports Science labs which have a version 2017.x and that did not work.

Sample footage Sample videos of two participants using the virtual environment can be found in the submission folder, /sample_footage_1.mp4 and /sample_footage_2.mp4. Sometimes, it may look like there is a second image of the ball stuck on the screen - this is an artifact of compression. Note that the red dot marks the gaze point of the subject.

Collected data The anonymised data from the experiment described in the report is attached here with the permission of the participants, as well as the configuration file used for the experiment. The name fields in the subject_info.csv files were manually cleared.

Other contents Some other resources are also added here, for completeness sake:

- The MATLAB scripts used to process the data from the experiment in /MatlabScripts/
- The LaTeX source files in /ReportLatexSource.zip
- The slides from the demo at /Demo_Slides.ppt
- A copy of these instructions is also available inside the .zip at /instructions.txt