Technical University of Denmark

BSc Project Report

Detection of Impaired Eye Tracking Using Mobile Devices

Detektion af Svækkede Følgebevægelser i Øjnene med Mobilenheder

Daniel Everland
s192081

Noah Bro-Jørgensen
s204500

Supervisor
Per Bækgaard

June 5th, 2023

Project duration
30/1 2023 - 5/6 2023

Project workload
20 ECTS

Abstract

This report details the development of an application designed to utilize mobile eye-tracking technology in the detection of ocular-motor impairments related to concussions. It serves as an exercise in exploring modern, lightweight eye-tracking technologies, as well as how a user-centric design approach can allow any user to easily perform complex tasks such as screening for a concussion.

We define the problem to be solved as the need to quickly and accurately detect signs of concussion. We researched existing screening techniques, both analog and digital, and consulted an expert in the field. A two-pronged, iterative approach examining both technical limitations and user satisfaction was used to evaluate our design candidates. We settled on a digital King-Devick test, using gaze tracking as an additional vector of analysis.

User tests generally conclude that the resulting prototype is simple to use, and our fatigue test confirms that the app is able to detect a change in cognitive faculties. Mobile gaze tracking works, at least to the extent of usually being able to track the horizontal eye movements that are of interest during the King-Devick test. However, we cannot conclusively state how helpful this data is to screening without showing the application to more experts in the field.

We believe our prototype serves as an excellent foundation for future applications to further explore the potential of gaze-tracking and other methods of data analysis as additional vectors of evaluation in the screening of concussions.

Contents

1 Introduction

In Denmark, concussions affect roughly 25,000 people every year. Most victims recover naturally after a few months, but roughly 35% suffer from long-term symptoms that can last months or even years. Long-term symptoms present themselves both somatically and psychologically, and can significantly impair a person's ability to live their daily life. This incurs a large societal cost in terms of using medical services and increases the risk of unemployment for the afflicted[1]. It is important that concussions are diagnosed and treated when present. As such, this project will address the possibility of using mobile technology to ease the task of screening for concussions.

When it comes to the diagnosis and treatment of concussions, we consider three important aspects of the issue:

Concussions Are Serious:
Traumatic brain injuries (TBIs) can be classified as mild, moderate, or severe, with concussions classified as mild forms of TBI.[2] Despite the name, mTBIs can still have severe consequences for the afflicted, leading to long-lasting symptoms such as chronic headaches, light and noise sensitivity, and problems with vision and concentration.

Concussions Must Be Diagnosed Quickly:
Concussions may worsen if not treated properly, and repeated trauma may significantly lengthen recovery time or lead to severe or fatal complications.[3] In situations such as military operations or athletic competitions, it's important to be able to quickly assess if someone has suffered a concussion in order to determine if they can continue fighting or competing.

In these situations, the tools and expertise necessary to make such an assessment may not always be present or fully reliable, complicating attempts at diagnosis.

Concussions Can Be Difficult to Diagnose:
Concussions aren't immediately apparent to outside observers or even the person suffering from the condition. In fact, symptoms may not manifest until some time after the inciting injury, making immediate diagnosis next to impossible.[4] Symptoms may also vary from case to case, and concussions may even be clinically categorized by the symptoms they cause. For example, some concussions may affect a patient's light-sensitivity or ocular-motor functions, while other concussions affect memory or other cognitive functions. This means that any given concussion can require one of multiple different methods for screening and treatment.[5]

Physicians often rely on the symptoms reported by or observed in the patient in order to diagnose mTBIs. This of course requires that the examiner is familiar with the possible symptoms of concussions and that the patient is honest in self-reporting their symptoms, which is not always the case.[6][7]

The Centers for Disease Control and Prevention (CDC) estimated in 2003 that mTBIs constitute at least 75% of all cases of TBI in the United States[8], and the University of Pittsburgh Medical Center (UPMC) found that among student athletes, somewhere between 1.7 and 3 million concussions occur each year as a result of sports-related events. Furthermore, an estimated 50% of concussions are never reported.[9]

In summary, concussions are difficult to accurately test for, and they must be caught quickly to prevent repeat injuries from worsening symptoms or lengthening recovery times. Further complicating matters is that those who suffer from concussions may sometimes be unaware of their own symptoms or attempt to hide them if they are eager to return to the field.

We propose that we can help alleviate these problems with the help of modern smartphone technology. We formulate the following initial problem statement:

Is it possible to use a mobile device to help a user quickly and objectively detect symptoms of a concussion, and can such a solution be made simple enough that a user with little to no experience with screening for concussions can make use of it?

2 Related Works and Scoping

The following section describes tools and disciplines related to the problem described in the introduction. How do medical professionals currently screen for concussions, what concussion screening apps are currently on the market, and how does this knowledge further shape our initial problem statement?

2.1 Screening Methods

At a glance, there are a huge number of methods to screening and assessing concussions, too many to describe in detail here. Kutcher and Giza[4] mention three physical and three cognitive evaluation tools used in pre-injury testing, i.e. testing done before an injury to establish an objective baseline of performance. These are the clinical reaction time test (RT_{clin}), Balance Error Scoring System (BESS), and King-Devick test (K-D), as well as the Axom Sports Computerized Cognitive Assessment Tool (CCAT) Immediate Post-Concussion Assessment and Cognitive Testing (ImPACT), and Standardized Assessment of Concussion (SAC).

Murray et al.[10] highlight the impact on ocular motor function as a result of concussion and mention Vestibular Ocular Motor Screening (VOMS) and eye-tracking as methods of assessing such impairments.

Additionally, one of the apps described in the next section makes use of the Westmead Post-Traumatic Amnesia Scale.

Although there are many other assessment tools, we limit the scope of this section to providing a brief overview of the above-mentioned tools.

BESS
Balance Error Scoring System (BESS) tests the patient's balance by having them assume different stances on firm and unstable surfaces. As such, this test requires equipment that may not be readily available in any given situation. The modified BESS (mBESS) test includes only the first half of the test performed on a firm surface, and therefore requires no special equipment.[11]

CCAT
The Axom Sports Computerized Cognitive Assessment Tool (CCAT) is derived from the CogState Sport computerized tasks, which is used in many places around the world to help make concussion-related return-to-play decisions. CCAT tests the patient's reaction time, attention, learning, and working memory using a series of digital card games. The tool establishes a pre-injury baseline and is used to evaluate cognitive impairments in athletes that have suffered head injuries.[12][13]

GCS, WPTAS, and A-WPTAS
The Glasgow Coma Scale (GCS) and Westmead Post-Traumatic Amnesia Scale (WPTAS) are standardized tests used to assess the onset of amnesia in patients that may have suffered a concussion. While they screen for a symptom of concussion, the tests are designed not to diagnose concussion, but to monitor recovery. The subject must answer basic questions such as what their name is, what month it is, and where they are, as well as perform simple tasks such as identifying and recalling objects in an image. GCS is recommended as an initial assessment of the projected recovery time of a concussed individual, after which WPTAS is performed repeatedly in daily intervals, or the Abbreviated Westmead PTA Scale (A-WPTAS) is performed in hourly intervals. The former is used for severe injuries, and the latter for milder concussions where recovery is expected to be relatively quick.[14]

ImPACT
Immediate Post-Concussion Assessment and Cognitive Test (ImPACT) is an electronic test. It consists of three parts: 1) A survey detailing relevant medical history such as drug use, learning disabilities, and previous concussions. 2) A self-reported list of 22 symptoms, ranked by severity. 3) A number of exercises meant to test memory, reaction time, learning ability, and more.[15] As mentioned in the introduction, diagnosis through self-reporting is fallible in certain situations. The concussed individual may not be aware of anything being wrong, or they may downplay symptoms if they are e.g. an athlete eager to continue competing.[6][7]

King-Devick
King-Devick (K-D) is a saccade-based test, which is carried out by asking patients to read a series of

numbers out loud from flashcards that get progressively harder by removing line indicators and clumping numbers together. Once the patient has completed all three flashcards, their score is based on the amount of time it took for them to do so, and subsequently compared against a baseline score from tests they performed prior to the injury. If the screening score is worse than the baseline, it could indicate the presence of a concussion.[16]

RT_{clin}
Clinical Reaction Test (RT_{clin}) is a tool meant for assessing the reaction time of a patient, designed to be part of return-to-play decisions in a sports context. The test is performed using a weighted, vertical shaft, held aloft by the examiner. The patient is seated with their forearm resting on a desk and dominant hand around the base of the shaft (without touching the shaft). At randomly determined times, the examiner releases the shaft, and the patient must grab it without moving their arm. By measuring where the patient grabbed the shaft, the time between release and grab can be calculated.[17]

SAC
Standardized Assessment of Concussion (SAC) tests the cognitive abilities of the patient and can be administered on the sidelines of an athletic competition or in an emergency room. The test evaluates short- and long-term memory, concentration, and mental bearings, and is performed by asking the patient several questions and giving them problems such as listing the months of the year in reverse order.[11]

VOMS
Vestibular Ocular Motor Screening (VOMS) assesses balance and vision impairment. It tests smooth pursuit and rapid eye movements, reflex, motion sensitivity, and near point of convergence. According to UPMC, vestibular and vision problems are accurate markers for long-term concussion symptoms, but the test is best performed in combination with a standard symptom assessment and interview.[18]

Summary
Table 1 summarizes the assessment tools described in this section. Our problem statement highlights the need for quick and at least partially objective assessments of concussion, and as such our interest lies in assessment tools that are mobile, objective, and require no equipment other than a mobile device.

Of those listed, only BESS and RT_{clin} do not fulfill the requirements. Many existing tests seem to have been developed with similar requirements in mind, as a common application of concussion screening is return-to-play decisions in sports contexts. When considering only the fully objective tests, each of these assessments screen one or more of the following faculties: Orientation, memory, concentration, balance, and ocular-motor function.

Tool	Mobile	Objective	Requires Equipment
BESS	X	X	X
mBESS	X	X	
CCAT	X	X	*
GCS	X	X	*
WPTAS	X	X	*
A-WPTAS	X	X	*
ImPACT	X	*	*
King-Devick	X	X	*
RT_{clin}	X	X	X
SAC	X	X	
VOMS	X	X	

Table 1: Summary table of the assessment tools presented in this section. **Mobile:** Can the test be performed anywhere? **Objective:** Does the test rely on objective observations? (* = the test relies partially on objective observations) **Requires Equipment:** Does the test require any special equipment to perform, not including notes and instructions? (* = the test requires a mobile device or equipment replaceable by a mobile device)

2.2 Existing Mobile Applications

There are several mobile apps on the market that profess to serve as concussion screening tools. In order to determine how existing technologies utilize mobile apps to screen for the neurological faculties listed at the end of the previous section, we investigated three of the most popular apps available on app stores. It is worth noting that CCAT, described in the previous section, is also available as a paid mobile app.

HeadCheck[1]
This Android app is designed for use by both medical and non-medical personnel, with the use case of assessing concussions in athletes. The app can maintain a registry of different athletes for different personnel to review and report potential injuries, and supports both baseline testing and sideline injury assessment. The app is based on the ImPACT assessment tool, but simplified to a questionnaire that requires no equipment. As such, the app relies on incident details, self-reported symptoms, and simple memory tests. The app doesn't conclusively state whether the subject is concussed, but provides a score that indicates the likelihood of concussion.

FirstResponder[2]
This Android/iOS app is intended to be used in assessing concussions in athletes. Its main selling point is its ability to provide a recommendation in less than 90 seconds, that it can email summary reports to relevant parties, such as doctors, schools, and parents, and that it can show you directions to nearby qualified doctors or emergency rooms. The test features simple yes/no questions for assessing symptoms related to memory and orientation, and a picture-recognition test, all based on WPTAS.

HitCheck[3]
This iOS app features several practical tests for the patient to take. Instead of a questionnaire, the test is designed as a series of short, gamified activities that assess various cognitive, vestibular, and ocular functions. The app makes use of the phone's gyroscope and accelerometer to test e.g. the user's balance, reflexes, and coordination. Like HeadCheck, HitCheck also offers a dashboard for monitoring multiple athletes and teams, and recording baseline test results. HitCheck doesn't assess the likelihood of concussion directly but provides an overview of how the user scored in each activity and what brain function that activity correlates to.

2.3 Final Problem Statement

Screening for concussions is very complicated, and something that even professionals struggle to do accurately, depending on the setting. It's difficult to objectively assess many of the symptoms without simply asking the concussed person what symptoms they are experiencing.

Since concussions can have many different symptoms, there are a wide array of tools used to assess the condition, usually revolving around testing various cognitive, ocular, and vestibular functions. None of these tests appear infallible on their own, and the surest assessment of concussion is therefore a combined approach utilizing multiple assessment tools.

Existing tools and mobile apps highlight the importance of baseline testing. Many sideline tests only work when there is a relevant baseline to compare against. For example, if a patient performs poorly on a reflex test, it's important to know whether the result is an anomaly, or if the subject in question usually performs badly in such tests. As such, most objective tests should ideally produce a quantifiable metric describing the subject's performance, so that it might be compared to previous or future tests.

It's clear that a fully functional app for screening concussions would be very wide in scope. Tests such as assessing reflexes, balance, or long- and short-term memory might be relatively simple to design, but we determine that the work of this project is better spent focusing on a single, challenging test. We will then limit the scope of the project to designing this one type of test, with the understanding that it may not serve as a fully stand-alone assessment of concussion, but rather one test in a suite of tools used to screen for concussion.

According to neuro-optometrist Dennis Abildgaard (see Section 4), the brain processes approximately 11

[1] `https://www.headcheckhealth.com/`
[2] `https://www.headsmart.me/firstresponder/`
[3] `https://www.hitcheck.com/`

million bits of sensory impressions per second, of which visual information accounts for 10 million bits. It stands to reason that a concussed individual is very likely to suffer dysfunctions related to vision. Despite this, none of the apps researched in Section 2.2 feature any ocular screening outside of HitCheck's color recognition test. An ideal focus of this project would therefore be to perform an ocular test of a patient, similar to e.g. the King-Devick or VOMS tests.

Given the fact that we are software engineering students with no background in medicine or neurology, the goal of this project will not be to develop a new type of screening technique. A more sensible goal will be to adapt an existing screening technique and determine how we may improve upon it using modern smartphone technology. We lack the expertise to assess the efficacy of any product we create as a tool for screening concussion. For this reason, a sufficient goal will be to determine what type of improvement will elevate an existing screening technique, and whether we can implement it.

Using modern smartphone technology, is it possible to adapt and improve upon an existing ocular-motor concussion screening technique, and can such a test be made simple enough that a user with little to no experience with concussions can perform the assessment using only a mobile device?

2.4 Anatomical Nomenclature

With the goal in mind of adapting a concussion screening technique, it's prudent to define anatomical terms related to eye movement and neurology that are used throughout the report.

Cognitive
Related to a person's ability to think, reason, and remember. Concussions affecting cognition can cause symptoms such as trouble learning and retaining new information, decreased ability to focus and decreased multitasking skills.[3]

Ocular-motor
The term ocular means of or related to the eyes or vision. The ocular-motor system is a part of the entire central nervous system and controls a person's eye movement. Concussions affecting the ocular-motor system can cause symptoms such as making it difficult to read long passages or to look at computer screens.[3]

Vestibular
Relates to balance. Concussions affecting the vestibular system can make it difficult to interpret motion, to remain steady when moving your head, and impairs head and eye movement.[3]

Saccade
Saccades are extremely quick eye movements that occur when the eyes shift focus from one point to another.[19]

Smooth pursuit
Smooth pursuit is eye movement that is done subconsciously while tracking a moving point or object. While some people can make a smooth pursuit motion without tracking a moving point or object, most people who try to do so will instead make saccades.[19]

Vergence
Vergence refers to both convergent and divergent movement of the eyes, and is performed when keeping objects at varying distances in focus.[19] An example of convergence is illustrated in Figure 1. Here, the gaze direction of a pair of eyes converge in order to focus on a nearby point of interest. If the point were to move further away, the gaze directions would diverge in order to maintain focus on the point.

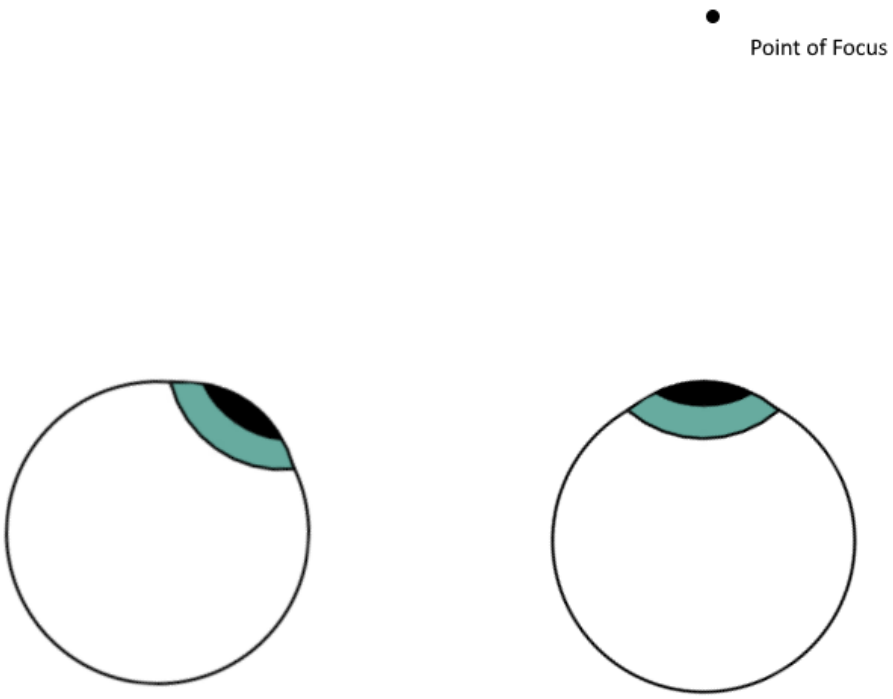


Fig. 1: Illustration showing how convergent eye movement keeps a nearby point in focus.

Vestibulo-ocular Movements
Vestibulo-ocular movement occurs when you keep a point or object in focus while moving your head. This can be demonstrated by looking at something while rotating your head from side to side. The object will remain in focus, and your eyes will rotate in the opposite direction of your head.[19]

Fixation
We define fixation as the brief period where visual information is "recorded" by the eye. In the context of saccadic movement, fixations usually occur between each larger movement of the eye.[20]

3 Methodology

3.1 Project Planning & Task Management

At the beginning of the project, we decided to structure our tasks and work delegation using an agile project management service. This would allow us to easily keep track of what we needed to work on, and who had been assigned the responsibility of making sure it got done. We considered three different services, one of which we have used previously during our studies at DTU, but all of which we have experience using either professionally or for personal projects:

1. Jira[4] is arguably one of the largest and most widely used services on the market. It has everything we would need for our project but is also built for large enterprise projects, which has the drawback of, at times, being a bit too rigid and overwhelming for a project as small as ours.
2. Trello[5] is another very well-known service and essentially the antithesis of Jira. Where Jira has large systems built for scaling with thousands of team members, Trello simply provides Kanban-esque boards you can use in whichever way you please.
3. ClickUp[6] provides a happy medium between the two. It is simple to set up and get going, while still having traditional issue tracking we can use to monitor our progress.

We settled on using ClickUp, and while we utilized it extensively the first couple of weeks, we gradually slipped into simply meeting a couple of times a week to discuss how things were going and what to work on. Additionally, we kept in touch almost daily through the messaging service Discord[7] to keep each other appraised of our progress and to make joint decisions regarding the direction of the project. Towards the end of the project, while we were spending most of our time proofreading the report and making minor improvements, we started utilizing Overleaf's comment feature to delegate tasks.

3.2 Double Diamond

The double diamond is a model used for structuring a process with which a product or solution is created based on an initially defined problem statement or idea. The process is split into two major parts, which are comprised of each of the diamonds, each subdivided into two smaller parts called phases that broaden or narrow the scope of the process.

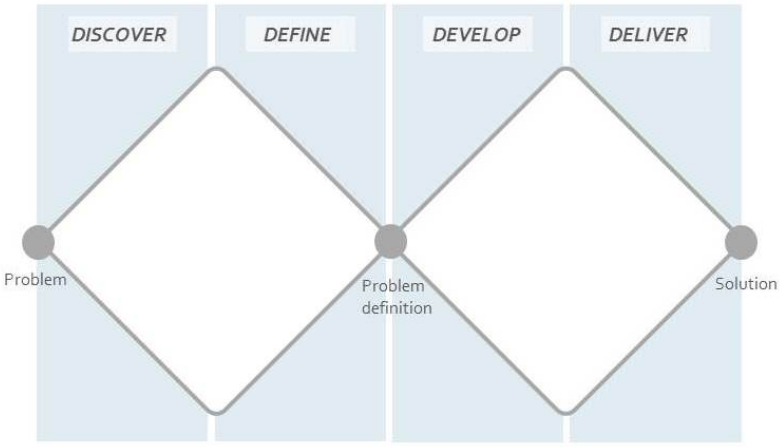


Fig. 2: Illustration of the double diamond process. Source[21]

The first diamond focuses on properly identifying the problem which needs a solution, whereas the second diamond focuses on determining how to best solve said problem.

[4] https://www.atlassian.com/software/jira
[5] https://trello.com/
[6] https://clickup.com/
[7] https://discord.com/

The first phase broadens the scope of the problem statement, gathering information through research and analysis to consider all possible angles from which to understand the initial problem statement. This phase occurs in Sections 1, 2, and 4, as we define the problem and research existing methods in approaching it.

The second phase narrows the scope, boiling the problem space down into a very specific and well-defined final problem definition. This refinement occurs at the end of Section 2, and the scope is further narrowed by our meeting with a neuro-optometrist (Section 4).

The third phase expands the solution space by workshopping and brainstorming ways of solving the problem definition. We develop and present our solution candidates in Section 5, and we explore options for implementations in Sections 6 and 7.

The fourth and final phase narrows the solution space until a concrete and optimal solution is found that best solves the problem definition. As we draw conclusions from experiments carried out in Sections 6 and 7, we gradually arrive at a single implementation that solves our problem, described in Sections 8 and 9.

3.3 Primary Data Sources

3.3.1 Surveys

Our approach to conducting surveys is to favor quantitative data over qualitative data wherever possible, with additional optional qualitative fields that allow the subjects to elaborate upon their answers. For instance, instead of asking open-ended questions like "What do you think about the following 2 designs", we prioritize asking questions like "Which of the two options do you prefer", with an optional field permitting them to elaborate if they wish to do so.

We find that the benefit of writing surveys this way is that you always get actionable data that is easy to compare and contrast from the mandatory questions, while still getting in-depth answers in the optional questions when test subjects wish to do so. This results in us being able to make decisive choices based on the survey's responses while still getting a broader understanding of why the subjects have the opinions that they do.

3.3.2 Field Specialists

Early on in the project, we reached out to several experts related to the screening and treatment of concussions. One of these experts was a neuro-optometrist, who works with the rehabilitation of patients suffering from vision-related disorders that stem from neurological issues. We arranged a meeting with Dennis Abildgaard, employed at Privatsyn, and had an exploratory dialogue with him about how to use mobile devices and vision to test for concussions.

We were also in contact with the medic of the Danish national team in American football, who is involved in return-to-play decisions in case of injuries such as concussions. Unfortunately, we were unable to schedule a meeting with them during the project period.

We reached out to several other people and organizations, but all of these either turned down the meeting or never replied. These contacts include: A research associate professor of neurology, the neuro-ophthalmological team at Rigshospitalet, and the Royal Danish Defence College.

While it's a shame we could only arrange one expert meeting, we also believe that additional meetings would have had a reduced influence on our project. More meetings would certainly have benefited us, but our one meeting with Dennis Abildgaard undoubtedly had a major impact on the direction of our project.

Toward the end of our project, we attempted to reach out to Dennis again in order to get an expert opinion on our final product. This would have served as a valuable means of evaluation for our final testing, described in Section 9. Unfortunately, we were unable to get in touch with him again.

3.4 Secondary Data Sources

A major part of this project involves using and referencing terms and concepts related to vision and concussion, which is a scientific subject neither of us have any formal training or experience working with. Because of this, we had to gather a lot of data in order to attain a level of knowledge sufficient to understand the problem statement and propose potential solutions. All of this requires reading a lot of scientific articles, and as such, we have compiled a list of all of our references in our paper's bibliography, where every citation follows the IEEE citation style.

In regards to technical information, which mostly pertains to using third-party frameworks and the Android SDK, we have chosen not to use the same methods of citation. We made this decision in order to avoid cluttering the bibliography to an unnecessary degree, as we reference a lot of different technical documentation throughout this report. As such, we will utilize footnotes for links to technical documentation. We also use footnotes when referring to institutions, people, and products.

3.5 Build/Measure/Learn Cycles

Our approach to development has followed an iterative Build/Measure/Learn (BML) cycle similar to that described in Eric Ries' *The Lean Startup*[8].

In this approach, we begin each cycle by building a new design, then test and measure the design according to some relevant metric, and finally analyze our findings in order to learn how or whether to move forward.

The rationale behind this approach is to ensure efficient use of development resources by experimenting and gathering feedback frequently, rather than potentially wasting time on components that don't pan out. In our development, we prefer simple, minimum-viable products that communicate intent and can be further built upon in future iterations after gathering data on the prototype's performance.

We have used BML cycles for the work described in Sections 6 and 7. Doing so has allowed us to quickly test out ideas and discard or move forward with them as deemed appropriate.

For the experiments in Section 6, rather than utilize BML to iterate on a single design, our approach has been to develop multiple components independently of each other. We then iterate on these parts until we either deem the component unsuited to our needs or ready to be integrated into the final product. For the design considerations in Section 7, we have focused iterations on a single design that transitions into the final product iteration described in Sections 8 and 9.

[8] Eric Ries. The Lean Startup. *Crown Business*, 2011

4 Expert Interview

We met Dennis Abildgaard, who is a neuro-optometrist at PrivatSyn[9], to conduct a loosely-structured informal interview about how TBIs affect vision in general, which existing IT solutions are currently used, and proposals for potential improvements. The primary takeaway from the interview was how you can test a patient's vision in three different ways: vergence, saccades, and pursuit. All of our questions and notes regarding this meeting can be found in Appendix 12.7.

Vergence is the concept of your eyes moving independently of each other to focus on an object. Dennis showed us several different methods of assessing a patient's ability to converge, amongst them a test where the patient wears polarized glasses while looking at two plastic sheets with symbols on each sheet, see Figure 3.

Fig. 3: Photo taken of one of the vergence testing tools Dennis introduced. The polarized glasses make only certain symbols visible to either eye in such a way that the wearer's eyes must to converge to align the two columns of symbols. The two sheets can be moved around within the frame to test the limits of the patient's convergence.

Dennis highlighted vergence as an excellent indicator of concussion. However, he also warned that testing for this type of eye movement without physical tools or prior experience in the field of neuro-optometry was going to be exceptionally difficult and well beyond the scope of this project.

Instead, Dennis recommended we study saccades as there is a myriad of ways to evaluate a patient's ability to perform saccades. He mentioned that King-Devick is used in sport contexts to screen for concussions. When we mentioned the idea of using eye-tracking for smartphones, Dennis again recommended King-Devick, stating that gaze tracking information could serve as a useful additional means of analysis for such tests.

We did not spend a lot of time discussing pursuit with Dennis, as he stated it wasn't a good measurement of TBI's and that it has a weak correlation with concussions, specifically compared to testing saccades.

In summary, Dennis strongly recommended we use saccades for screening instead of smooth pursuit. He also emphasized how important comparing against a baseline is, as biological traits differ greatly from person to person, and as such creating a one-size-fits-all algorithm for analysis is very difficult.

[9] `https://privatsyn.dk/om-os/klinikkens-medarbejdere/`

5 Preliminary Designs

Based on the research presented in Section 2 and the insights provided by our interview with Dennis Abildgaard, we have limited the scope of the project to focus on one of two types of eye movement: Smooth pursuit or saccade.

While smooth pursuit isn't recommended by Dennis, there is enough evidence to suggest it may still be a feasible strategy[10][22][23], and we also move forward with saccades on Dennis' recommendation.

To align our solution with existing and established methods of screening, we limit our scope to utilizing the ocular-motor screening techniques found in VOMS and King-Devick tests. Both types of solutions would rely on gaze tracking to detect the direction of the patient's gaze in order to assess their performance on the test in question. In this section, we detail the basic design of these two tests, and further explore and develop each idea in later sections.

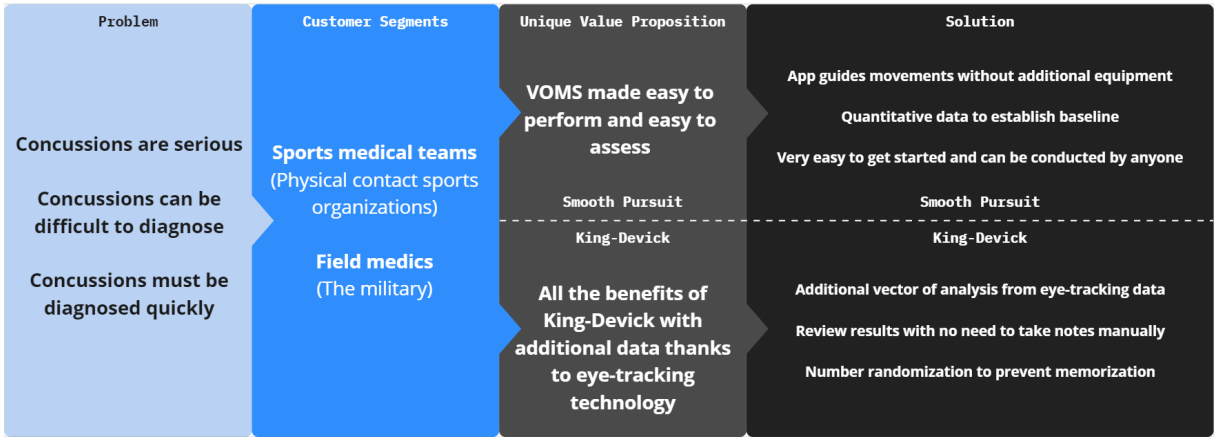


Fig. 4: Lean canvas of the two proposed designs discussed in this section. Both are based on the same core problem and geared towards the same customers, but differ in execution.

5.1 Smooth Pursuit

The smooth pursuit test is based on one of the modules of VOMS. In this module, the assessor sits at a distance of 3 feet from the patient and has them focus on the tip of their index finger (see Figure 5). The assessor then performs a horizontal movement by moving their finger 1.5 feet left of center, 1.5 feet right of center, then back to center. The movement takes approximately 4 seconds. In the second part of the test, the assessor performs an equivalent vertical movement.

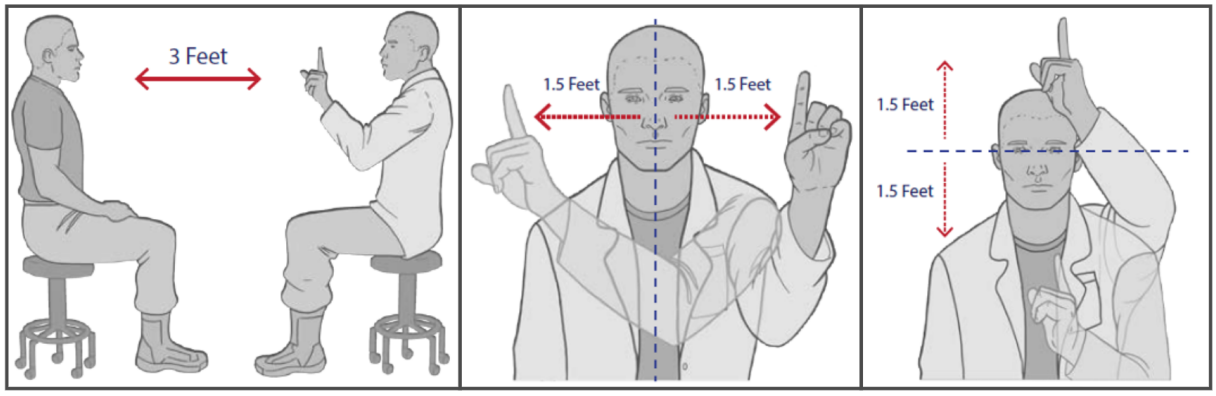


Fig. 5: Illustration of how the smooth pursuit module of VOMS is carried out. Source: [24]

UPMC recommends a tape measure and metronome to keep track of the distances and timings of the test[18], however, it can be expected that an experienced medical professional can perform the assessment

without such guides. The aim of our solution would be to assist those inexperienced in the assessment to carry it out using only their mobile device.

The most obvious solution would perhaps be to carry out the whole interaction on screen, i.e. rather than having an assessor wave a finger around, display a moving dot on the phone screen and use it to guide the patient's smooth pursuit eye movements.

Since this module of VOMS tests the patient on their smooth pursuits out to a certain angle, it's important that we preserve this angle in the transition to phone screen. Since the movement distance in VOMS is equal to the distance between patient and assessor, this would require the user of a phone app to hold the phone at a distance equal to the length of the phone screen. Conducting an assessment at such a short distance poses two problems: First, the phone's front-facing camera is unlikely to be able to fully capture the patient's face, making gaze tracking impossible. Second, at such a close distance, the patient's eyes will move at different angles from each other when focusing on the phone screen.

Using the phone screen for VOMS poses too many problems to be viable. Instead, one might consider the possibility of the assessor using a phone instead of their finger to guide the patient's smooth pursuits.

If the assessor points the phone camera at the patient during the test, gaze tracking technology can detect when the patient is looking at the phone. If they're not looking at the phone, gaze tracking may also be able to detect how far off-target the subject's gaze is. In this way, the smooth pursuit module of VOMS is improved by providing solid, quantitative data on the patient's smooth pursuits. This helps an inexperienced assessor uncover a potential visual dysfunction, and allows for tests to be compared against each other in order to establish an objective baseline performance.

Furthermore, it may be possible to use the phone's sensors to track the movements of the device. Doing so may allow a potential app solution to guide the assessor's movements during the test, so that they need neither equipment nor experience to perform the correct gestures. We would need further user testing to determine how exactly the app would guide a user's movements, but modern phones provide many different tools to explore, such as vibration, on-screen indicators, or sounds.

Figure 6 shows the user story map of the proposed solution. Besides the app, there are two actors: The user, responsible for assessing TBI, and the patient, who is the potentially concussed subject. The map describes two overall goals in using the app: First is taking the test, which can either be with the aim of establishing a baseline or to perform a post-injury assessment. Second is reviewing the results, wherein the user can either add the results to their baseline, or assess the possibility of concussion based on the presented statistics.

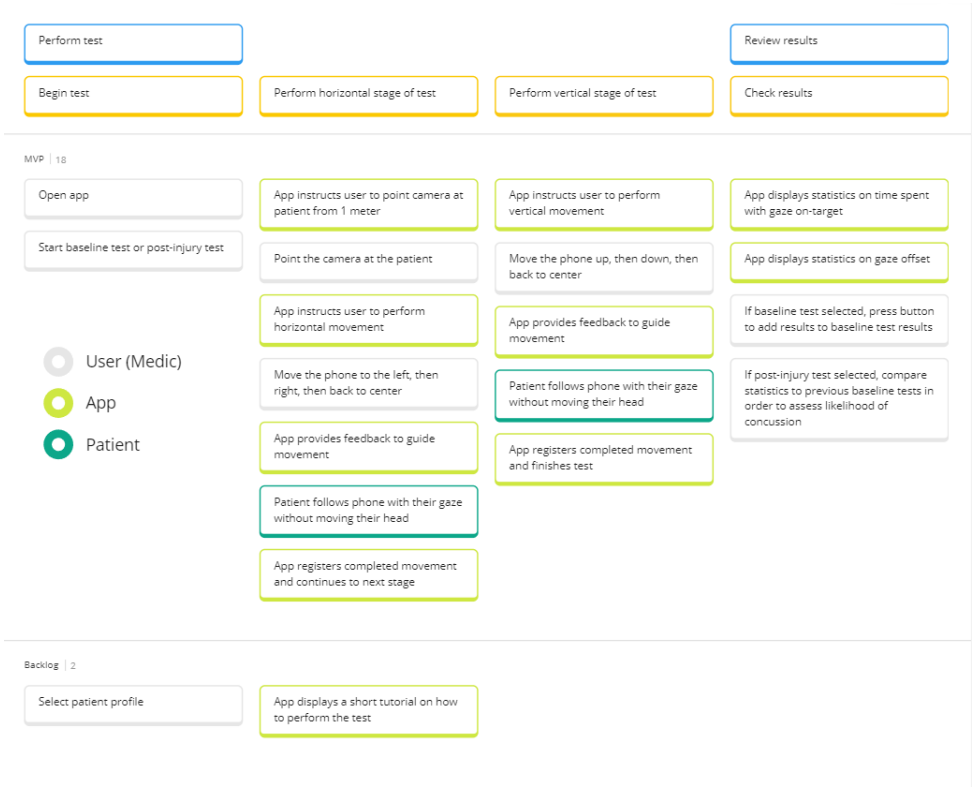


Fig. 6: Initial user story map for the proposed smooth pursuit design.

5.2 King-Devick

During our meeting with neuro-optometrist Dennis Abildgaard, he recommended the King-Devick test as a foundation for a screening app utilizing gaze tracking. King-Devick tests the subject's ability to perform saccades, and according to Dennis, this type of eye movement is an excellent indicator of brain injury.

The test is performed by having the subject read a sequence of numbers from a series of test cards, like those shown in Figure 7. After the subject has familiarized themselves with the exercise using the demonstration card, they are then asked to read the numbers of the remaining cards aloud as fast as they can, left to right, top to bottom.

The assessor tracks the time it takes for the subject to read each card and also tracks whether the subject makes any errors without immediately correcting themselves.

For the finer details of how to run the King-Devick test, we used the procedure described in the Methods section of the paper by Whelan et al.[25]. Based on this description, we formulated the following procedure for baseline testing and screening:

1. Explain to the patient the basics of performing the test (read numbers aloud as quickly as they can without performing errors).
2. Provide the patient with a demonstration card, and let them know that they won't be scored on this card.
3. Explain to the patient that the next flashcards will be scored.
4. Provide the patient with the first test card and record the time it takes to read it aloud.
5. Repeat the previous step for the 2nd and 3rd test cards. If the patient makes an error without immediately correcting themselves, restart from the 1st test card if doing the baseline. If doing the screening, the test is failed in case of an uncorrected error. Record the total time spent on all three test cards.
6. If screening, end the test here and provide a warning if total time is greater than the baseline. If performing baseline, repeat steps 2 through 6, and record the fastest of the two total times as the final result.
7. If the final result of the baseline or screening test is lower than the current baseline, record the result as the new baseline.

Our proposed design implementing this test would have the cards displayed on the phone screen for the user to read aloud. The app would then handle taking time on the user's attempts, as well as record audio as they read the numbers aloud. The recorded audio can be used to verify that the subject has made no errors, either after or during the test. The front-facing camera can be used to record eye-tracking data as the test is carried out, providing additional insights that a regular King-Devick test would not. An additional benefit of replacing the test cards with a phone screen is the ability to change the sequence of numbers, preventing users from memorizing the test.

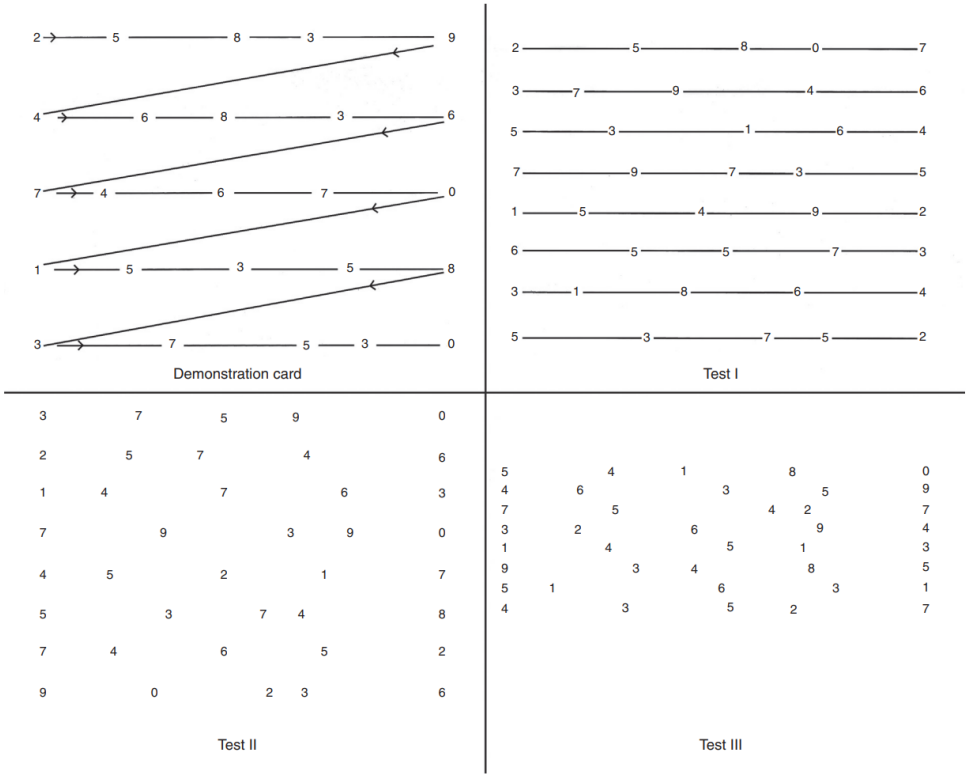


Fig. 7: Example test cards used in King-Devick. Source[16]

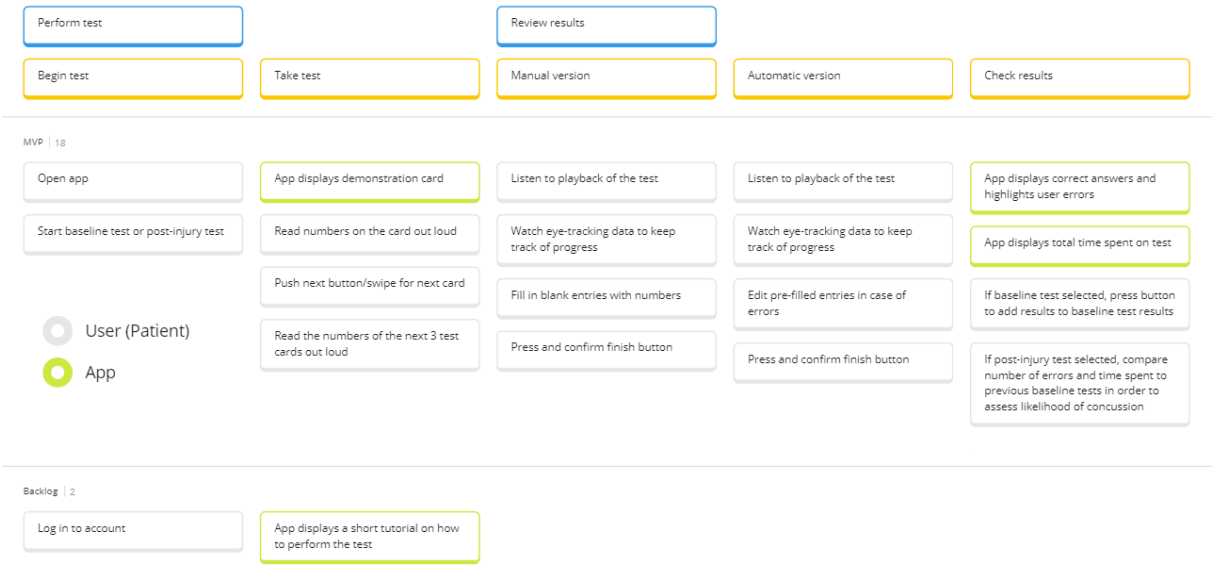


Fig. 8: Initial user story map for the proposed King-Devick design.

Figure 8 shows the user story map of this solution. As with the previous solution, goals are divided into taking one of two types of tests and reviewing their results. There are two proposed methods of verifying the spoken sequence of numbers, represented by the "Manual version" and "Automatic version" tasks. This solution doesn't necessitate that a medic/assessor is involved in the user flow. We do however suspect that a certain degree of competence is necessary to properly interpret collected gaze tracking data, so we still recommend that a professional assessor performs the "Review Results" goal.

5.3 Moving Forward

Now that we have properly defined how our two chosen solution candidates work, it is time to determine which of the candidates is best suited to be developed into our final prototype.

Our design descriptions give a general impression of what technical features are necessary for a successful implementation, and also allow us to present the concept of our solution to people outside the project.

To best select a solution candidate, we employ two separate approaches: First, we perform a series of technical experiments involving the features we require for implementation, thus assessing our ability to properly adapt the given screening technique. Second, we assess the user experience of both solutions, in order to determine what screening technique is preferred by users, and how our adaptation can be further improved.

These two approaches are documented in the coming sections.

6 Preliminary Technical Experimentation

With preliminary designs narrowed down to two solution candidates, work begins on selecting the best of these solutions. Our selection process involves two overall considerations: On a purely technical level, what are we capable of implementing (Section 6) and what provides the best user experience (Section 7)?

This section details experiments carried out with the aim of researching implementations of features for both VOMS and King-Devick. We are interested in implementing the smooth-pursuit screening technique from VOMS, and as such will require the patient to track an object using their eyes. We will explore how using the phone as such an object can be combined with eye-tracking to determine whether the patient is tracking the movement of the phone correctly or not. King-Devick will likewise include an exploration of eye-tracking technology, as we will use this to analyze a patient's saccade eye movement, but we will also look into converting audio recordings of a patient reading flashcards aloud into data that can be used to determine whether they did so correctly or not.

6.1 Wheel Motion Analysis

6.1.1 Goal

This experiment relates to the smooth pursuit solution described in Section 5. A big part of this solution is combining the motion of the device with an eye tracking feed that can detect whether a user is correctly tracking the device with their eyes. The initial goal of this experiment is to perform both a camera and a motion sensor recording and see if the two can be combined and synchronized. We wish to do so by measuring a simple, predictable movement, in this case the motion of a spinning wheel. We will examine the following aspects:

1. Is it possible to synchronize sensor data and camera footage?
2. Can we use the gathered data to create a reliable model for the position of the device while it's being rotated?
3. Can we detect the eyes of the subject while the device is being rotated?
4. Can we detect the eyes of the subject even if they do not look directly into the camera?
5. Are there any types or degrees of motion that affect the accuracy of our analysis?

6.1.2 Setup

We will utilize two devices, each gathering a separate set of data, in order to ensure running several data-gathering apps on the same device does not reduce performance and negatively affects the data collected. The first device will record a video using the default Android camera app. The second device will utilize a third-party app[10] to gather accelerometer and gyroscopic data. The two devices will be taped together to the bicycle wheel, to ensure their positions are identical.

[10] `https://play.google.com/store/apps/details?id=de.lorenz_fenster.sensorstreamgps&gl=US`

Fig. 9: Front and side view of the bicycle wheel setup

The experiment involves two actors: A user actor in charge of moving the device, and a patient actor who is the subject of the camera recording. The patient actor sits in front of the wheel, while the user actor spins the wheel. The experiment is conducted several times, each time varying the following parameters:

- Distance between the sitting person and the wheel.
- The angular velocity of the wheel.
- Whether the sitting person looks into the camera or at the center of the wheel.

We will conduct tests at three distances, and for each distance we rotate the wheel at three different speeds. This is done twice for both of us; once where we follow the camera with our eyes, and once where we constantly look at the center of the wheel. This is done to simulate a person who is not able to properly track the motion of the devices and can be used to determine whether we can detect this.

6.1.3 Results

We encountered several issues doing the experiment this way. The first issue was that the video recording stopped when the device locked, leading to us conducting the entire experiment once before realizing none of it was recorded.

We changed the experimental setup so the video-recording device was placed behind the sensor-recording device, letting the user actor see the screen as the experiment was conducted. This unfortunately led to another issue, as we discovered the tape used to adhere the phones to the wheel would trigger the touch screen, causing the camera to rapidly and uncontrollably zoom in and out during the experiment.

All in all, the experiment failed to produce any meaningful camera data, and the motion sensor data had multiple problems of its own, as described later in Section 6.2.

6.1.4 Conclusion

The difficulties encountered during this experiment made us realize that the experiment needed to be simplified. Incorporating a bicycle wheel in the experiment led to too many issues, and we immediately moved on to another version of the experiment using handheld motions instead.

6.2 Handheld VOMS

6.2.1 Goal

This experiment was performed immediately after the one described in the previous section, and its goal remains the same as the previous: Can we somehow unify the data sets obtained from a camera recording and a motion sensor recording? We now wish to see if a change in setup allows us to obtain the data we need.

6.2.2 Setup

In this experiment, we still move two devices around as one in order to collect separate sensor and camera data. Instead of using a wheel, however, the assessor instead holds both devices in their hand and moves them around manually. Since it would be extremely difficult to produce a circular motion in this way, we resort to simpler, linear motions based on those performed by the smooth pursuit module of VOMS (see Figure 5).

We performed this movement as smoothly as we could while keeping the patient's face in frame. As recommended by VOMS, each movement was performed over 4 seconds.

While VOMS recommends the subjects stand roughly 90 cm apart, we tested three different distances in order to determine how this affects the difficulty of carrying out the test and how it affects the quality of our image analysis. We also did one test for each of us where we did not look at the camera, once again attempting to simulate a person who is unable to track the motion of the devices.

Rather than start and end a recording for each step of the experiment, the whole experiment was carried out in a single recording. To mark each step, the devices were quickly flicked 90 degrees to cause an easily identifiable spike of activity in the motion sensor logs.

6.2.3 Results

Sensor logs obtained from the experiment were unfortunately quite unreliable. The update frequency for the gyroscope was highly irregular compared to the accelerometer, resulting in large parts of this and the previous experiment having no gyroscope data. As this experiment was carried out immediately after the previous, we had not noticed this issue before carrying out this experiment.

Because we relied on a quick flick of the device to mark the beginning of observations, the missing data rendered us unable to determine any markers with which to synchronize sensor data and video. Both accelerometer and gyroscope were also subject to a high degree of noise that made reading the data next to impossible.

There are multiple sources of error in these experiments, the two most notable being: The device used to record sensor data was an older model, Samsung Galaxy S7 Edge, released in 2016. It's very probable that an older model has worse sensor capabilities, and that seven years of use have degraded the sensors further. For the experiment described in this section in particular, a second error source was the way in which the experiment was conducted. The devices were moved around freely by hand with no guide, resulting in irregular movements.

6.2.4 Conclusion

We failed to record any sensor data worth analyzing, due to the issues mentioned in the previous section. In retrospect, both this and the previous experiment were far too complex for our needs, and the failure helped us determine more accurately how to proceed toward a viable product. Rather than trying to combine camera and sensor data immediately, we should adopt a simpler, divided approach to each type of data collection on its own.

We know that to use a phone to assess smooth pursuits, we need to 1) ensure the phone is moving as prescribed (horizontally or vertically) and 2) perform image analysis of the recorded footage to determine if the subject's gaze is following the device.

The simplest approach is to develop and experiment with both functions independently and combine them into one app once finished. To this end, we updated our plan with the following tasks:

- Collect viable sensor data: Measure sensor data for simple, single-axis linear motions and rotations. Then produce a simple app for live display of the direction of motion and rotation for further testing.
- Determine how to run gaze tracking analysis on phone hardware. Also determine if the analysis is fast enough to run parallel to the recording, or if it needs to be performed as a post-processing task.

Despite no viable sensor data being collected, the experiments led to a few unexpected but valuable UX-related findings:

The distance at which the assessment is performed is very important. The recommended distance at which to perform the smooth pursuit test according to VOMS is 3 feet (approximately 90cm).[24] We reached the same conclusion during testing. At further distances (200cm), the assessor needed to move the device much further to cover the same angle of the subject's vision, which the assessor couldn't do while standing still. At shorter distances (50cm), the proximity between the patient and the assessor made the test awkward and uncomfortable.

While moving the device, the assessor will need to tilt the camera slightly to keep the subject in frame. A simple way of doing this is by letting the assessor view the camera feed and make the necessary adjustments themselves during the test. On the other hand, looking directly at the subject's face through a phone screen at odd angles presents its own difficulties. It may be worth experimenting with different forms of feedback to ensure the subject remains in frame.

Tilting the device like this during the test also presents some challenging wrist movements for the assessor. In analog VOMS tests this isn't a problem, as the subject of focus is a finger, where orientation doesn't matter. We see this in the center image of Figure 5, where the assessor's palm turns with their arm during the movement. Attempting to keep the palm oriented toward the patient during this movement results in an uncomfortable twist of the wrist, and the same thing happens when trying to maintain proper orientation of a phone.

Besides this issue, we found that smooth horizontal and vertical movements of the device are generally difficult to perform. Hands may shake a little, the device may dip briefly, and it's unlikely that the assessor will manage to maintain a constant speed through the test. These aren't things we can prevent, but they are things we will need to consider and adapt our solution to.

All in all, these findings seem to speak against the viability of smooth pursuit as a solution candidate.

6.3 Accelerometer and Gyroscope Experiments

6.3.1 Goal

Due to the faulty data obtained from motion sensors in previous experiments, we now wish to devise an experiment involving analysis of the simplest possible movements, in order to determine whether the goal of tracking a phone's movements is at all possible. We leave the task of eye-tracking to later experiments and instead focus solely on motion sensor data.

The goal of this experiment is to determine whether we can obtain a device's change in position and orientation by processing data from its accelerometer and gyroscope, as well as how accurate and reliable this data is.

6.3.2 Setup - Linear Accelerometer

We use Physics Toolbox Suite[11] on Android to perform a series of experiments using the accelerometer, and the gyroscope. As mentioned in the previous experiment, a concern was the performance of an older model of phone. These experiments are carried out on newer models: a OnePlus Nord and a Samsung S22 Ultra. The data obtained was very similar across both models, but only the data from the OnePlus Nord is presented in this section.

We perform five experiments using the linear accelerometer. Each experiment is carried out with the phone face-up on a horizontal surface, without changing the orientation of the phone. Aside from the first experiment, each experiment starts and ends with the phone at rest for at least 5 seconds.

1. Leave the phone at rest for about 10 seconds.
2. Slowly slide the phone along its x-axis (parallel to the top and bottom edges of the phone).
3. Slowly slide the phone along its y-axis (parallel to the left and right edges of the phone).
4. Slowly slide the phone along its z-axis (perpendicular to the phone's screen).
5. Slowly slide the phone along a path incorporating movements in both the x- and y-axes.

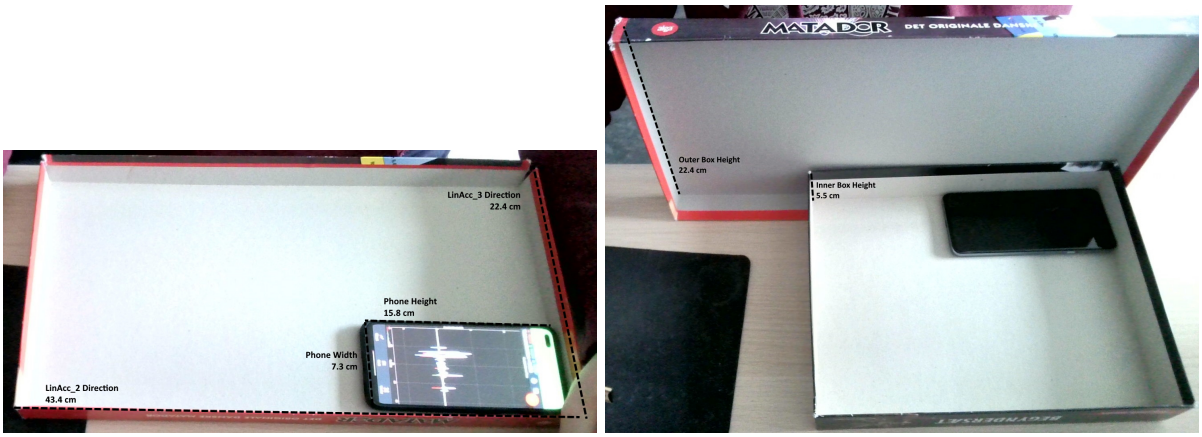


Fig. 10: Setup of the single-axis experiments and relevant dimensions.

Figure 10 shows the setup of the single-axis experiments. A cardboard box lid is used as a guide to move along each axis. To get the total distance in the x-axis and y-axis, the box's dimensions are subtracted from the phone's: $x = 22.4 - 7.3 = 15.1$cm and $y = 43.4 - 15.8 = 27.6$cm.
For the z-axis, the phone is placed inside another box in order to keep it level throughout the movement. The distance is calculated by subtracting the smaller box's height: $z = 22.4 - 5.5 = 16.9$cm.

[11] `https://play.google.com/store/apps/details?id=com.chrystianvieyra.physicstoolboxsuite`

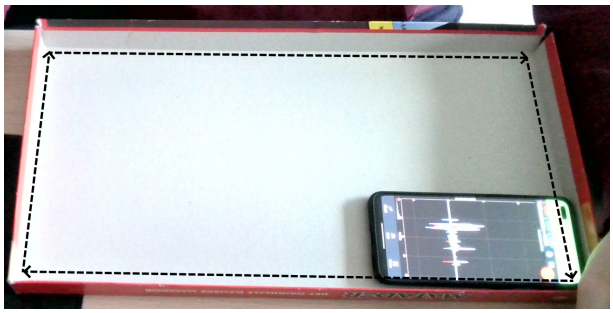


Fig. 11: Setup and movement path of the final experiment.

Figure 11 shows how the final experiment was carried out. Using the same cardboard box as a guide, the phone is first moved left (negative y), up (negative x), right (positive y), and then down (positive x) to its starting position.

By measuring accelerometer data for all these experiments, the distance traveled can be calculated using the trapezoidal rule to estimate the area under the curve, first for the accelerometer data, and again for the resulting velocity data.

6.3.3 Setup - Gyroscope

After the accelerometer experiments, we perform four gyroscope experiments. As with the previous set, each experiment starts and ends with the phone at rest for at least 5 seconds. Each experiment also begins with the phone lying face-up on a horizontal surface.

1. Leave the phone at rest for about 10 seconds.
2. Slowly turn the phone 90 degrees along its x-axis. Hold still for about 5 seconds. Slowly return the phone to its original position.
3. Slowly turn the phone 90 degrees along its y-axis. Hold still for about 5 seconds. Slowly return the phone to its original position.
4. Slowly turn the phone 90 degrees along its z-axis.

Rotation along the x- and y-axes is performed by turning the phone up against a vertical surface and lowering it again. Rotation along the z-axis is performed by turning the phone around 90 degrees while laying on a flat surface with the screen face-up.

Since the gyroscope measures angular velocity (rad/s), the data was integrated once using the trapezoidal rule to obtain the change in angle (rad).

6.3.4 Results - Linear Accelerometer

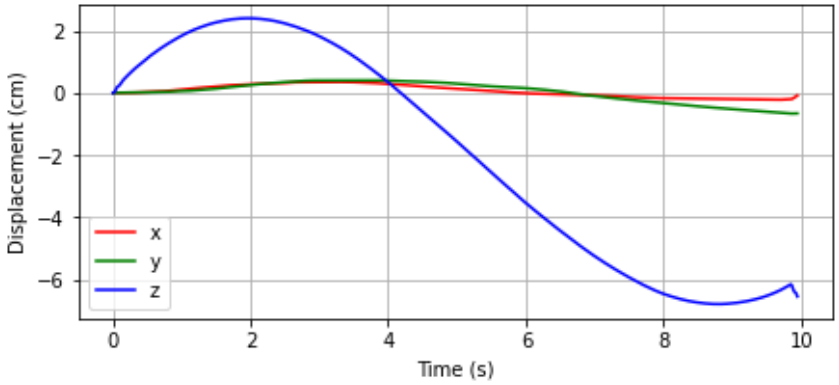


Fig. 12: Using measurements from the accelerometer, this graph shows the displacement of the device in an experiment with no movement.

Figure 12 shows displacement over time for the first experiment, where the phone is not moved. We see less than a centimeter of movement in the x- and y-axes, as expected, but there is a lot of displacement happening in the z-axis. We would expect a final displacement in all axes of 0cm, but we instead obtain a displacement of 0.08, 0.66, and 6.55 cm in the x, y, and z directions respectively.

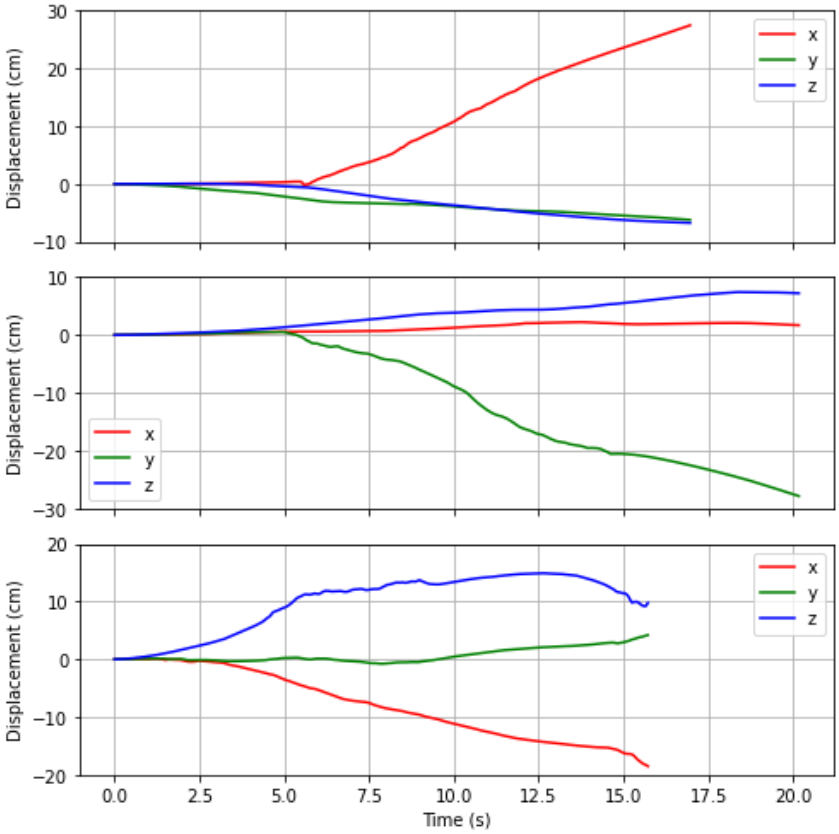


Fig. 13: The displacement of the device in three experiments: Movement along the x-axis (top), y-axis (middle), and z-axis (bottom).

Figure 13 shows displacement data for the single-axis experiments. We observe a high degree of inaccuracy here. Notably, despite the last five seconds of each experiment being spent with the phone at rest, we still regularly observe movement along most axes. The recorded movement along the x-axis in the first of

these experiments is much greater than anticipated (27.5cm vs. 15.1cm). For the second experiment, we obtain a value close to the expected value (27.9cm vs. 27.6cm). For the third experiment, we undershoot the expected value (9.8cm vs. 16.9cm), and obtain an almost 20cm displacement along the x-axis.

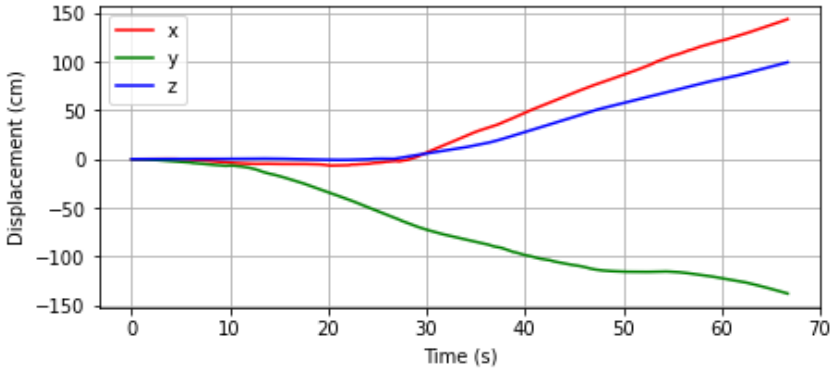


Fig. 14: The displacement of the device in the experiment with movement along both the x- and y-axes.

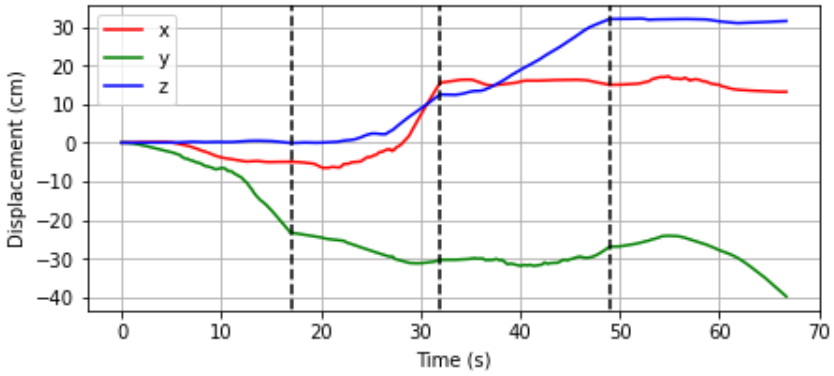


Fig. 15: The displacement of the device in the experiment with movement along both the x- and y-axes. The vertical lines mark times where the phone is held still and velocity has been reset when calculating displacement.

Figure 14 shows displacement data for the experiment involving movement along both the x- and y-axes. We expect a maximum displacement of 15.1cm for x and 27.6cm for y, and no displacement for z. We also expect total displacement to be 0 at the end of the experiment. However, the data obtained from the experiment ends up with displacement values in the range of 1.0-1.5 meters.

In Figure 15, data has been amended by resetting the velocity every time the phone is at rest during the experiment. This prevents noise from the accelerometer readings building up over the course of the experiment. While this reduces deviation significantly, it still isn't enough to obtain accurate displacement values.

This part of the experiment confirms that using the accelerometer to detect the phone's movements will be much more difficult than simply reading the accelerometer data.

6.3.5 Results - Gyroscope

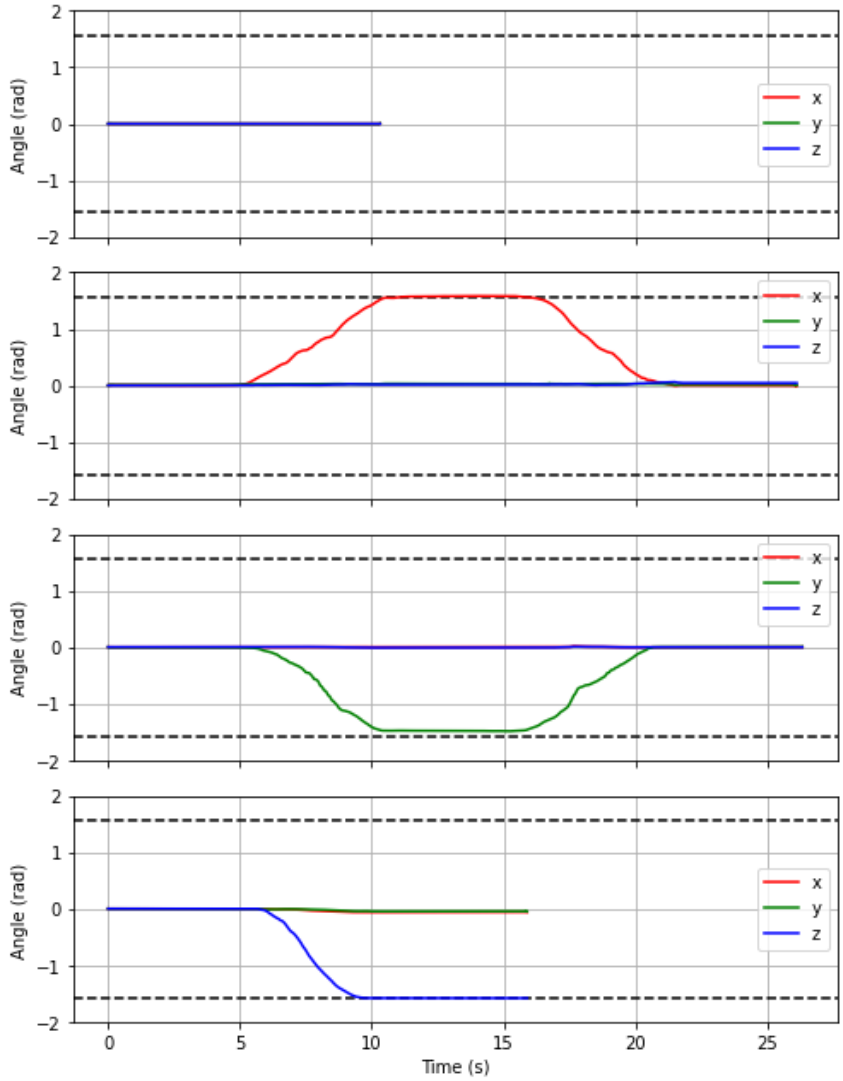


Fig. 16: The change in angle recorded by the gyroscope for each of the 4 gyroscope experiments. The dashed lines mark $\pi/2$ and $-\pi/2$.

Figure 16 shows the recorded angles for each of the four experiments. In each case, the data is almost exactly as we expect it: There is no change in angle for the first experiment, and the change in angle is almost exactly $\pi/2$ radians for the remaining experiments. For the experiments with rotation about x and y, the angle also returns to its original orientation.

6.3.6 Conclusion

We can quickly ascertain from the linear accelerometer experiments that the accelerometer is not sufficient in determining the movement of a phone, at least not the type of movement we aim to perform as part of a smooth pursuit eye tracking test.

Further research into the topic reveals that this is not only a common problem, but also reveals why our data deviate so significantly from our expectations. In a tech talk concerning Android sensors[26], David Sachs explains that noise in accelerometer readings translates into drift in velocity data when integrated. Even if no acceleration occurs, integration will eventually accumulate enough noise that velocity starts growing positively or negatively. This drift is exacerbated greatly when the data are integrated again to obtain displacement data. Figure 17 shows an example of this drift for a device at rest. Despite accelerometer data fluctuating about 0, the noise causes a slight drift in velocity, which in turn causes

an even larger inaccuracy in displacement data. These inaccuracies pile up even faster when the device is moving, as we see in the data on Figure 14.

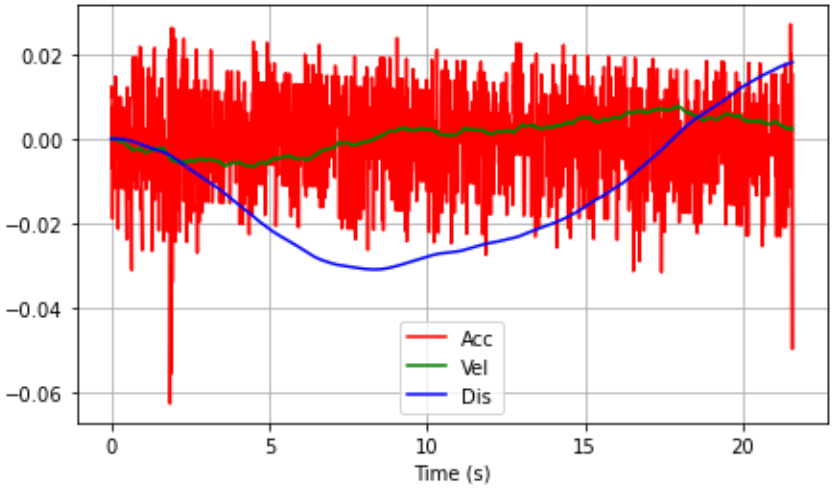


Fig. 17: Acceleration, velocity, and displacement data for a device at rest. Measurements are in m/s^2, m/s, and m, respectively.

Accounting for accelerometer noise is a difficult problem to solve. Signal filters can reduce noise, but they risk warping data and are dependent on the type of movement recorded. Another approach, suggested by David Sachs, is to combine multiple sensor data to reduce the impact of noise and error. The linear accelerometer used in our experiments already does this, as it combines magnetometer and gyroscope data to eliminate gravitational acceleration.

If we were to move forward with a smooth pursuit test involving movement of the phone, a possible solution would be to either partially or entirely rely on camera input to detect correct movement. This could be done by applying a facemesh or similar to the subject being examined and measuring the viewing angle. Alternatively, perhaps a framework exists that better integrates sensor data for more accurate movement tracking.

Unlike the accelerometer data, the gyroscope experiments produced results that aligned very well with our expectations. This is due to measurements having significantly less noise, and data processing requiring only one level of integration. It would seem that it is much easier to use gyroscope data compared to the accelerometer, but we should still keep in mind that integration of a noisy signal is likely to cause drift.

Our takeaway from this iteration is that tracking a phone's movements is much more difficult than anticipated, and will require significantly more research and effort to function properly.

6.4 MediaPipe Iris Detection

6.4.1 Goal

The previous experiment picked up on the motion sensor portion of our initial experiments. Having reached a conclusion on that front, we now move onto researching eye and gaze tracking for mobile phones.

In this and the next experiment, we explore the capabilities and limitations of two different eye detection frameworks when used on mobile phone platforms. The primary goal of these experiments is to find a gaze tracking framework to move forward with, and determine which of the solutions the software is best suited for.

Since the previous experiment made it clear how difficult motion tracking is, we are now mostly interested in applications involving the device screen. As such, gaze tracking will likely have to be performed using the front-facing camera.

In this experiment, we wish to determine:

- How difficult is it to track the eyes of a person using the front-facing camera?
- What data do we need to collect in order to determine where they are looking on the screen?
- Are there any other factors we need to be aware of when collecting data in future iterations such as ambient lighting affecting the recording?

6.4.2 About the Model

The experiment detailed in this section revolves around the MediaPipe Iris framework.[27] This landmark detection model excels at detecting the outlines of a subject's eyes and irises.

The model uses machine learning to fit a facial mesh over a user's face, then isolates the eye region for further analysis in order to outline the irises. The model is also able to provide an approximate distance between the subject's eyes and the camera by utilizing the fact that most irises are of the same size.

The model does not provide any information on where the subject is looking, but part of this experiment will be to explore how we can analyze the position and shape of an iris to detect gaze direction.

6.4.3 Setup - Physical Setup and Measurements

We will record a few short videos of a person's face while they look at different points on a mobile device, conduct some rudimentary image analysis on their eyes and compare this to data gathered by manually measuring the device, the person's face, and their distance to the device.

We created an image specifically for the device used in the experiment with four dots placed at each corner of the screen.

Fig. 18: Image displayed on a OnePlus Nord device during the test

The participant uses a ruler to measure the distance from their eyes to the device's screen. The ruler should be placed in the middle of one of the device's sides, which also helps align the participant's face correctly to be parallel with the given side of the device. This lets us roughly calculate the position of their eyes relative to the camera. After this, the participant fixates on each dot for a few seconds, starting with the bottom left dot and switching in a clockwise direction. This is repeated three times; once at a distance of 20cm, once at a distance of 25cm, and once at a distance of 30cm.

To calculate the position of the eyes, the distance from the participant's eyes and to the right side of their face is measured. We use the right side of the face, as this is the side used with the ruler to align the face with the device. The position of the camera and all of the dots respective to the bottom left of the device, which has been designated the origin, is also measured. From these measurements the following calculation can be done to ascertain the horizontal and vertical angle in degrees of each eye when looking at each dot at a given distance:

$$\theta = tan^{-1}\left(\frac{P_{eye} - P_{camera} - P_{dot}}{z}\right) \cdot \frac{180}{\pi} \tag{1}$$

Where all the positions are along the axis that is being measured, i.e when measuring the horizontal angle, P_{eye} is the horizontal offset of the eye measured relative to the origin, and where z is the distance between the eyes and the device.

Figure 19 shows the measured position relative to the origin of all relevant objects. These are the values used in Equation 1 in order to calculate the gaze angles manually for all distances and dots.

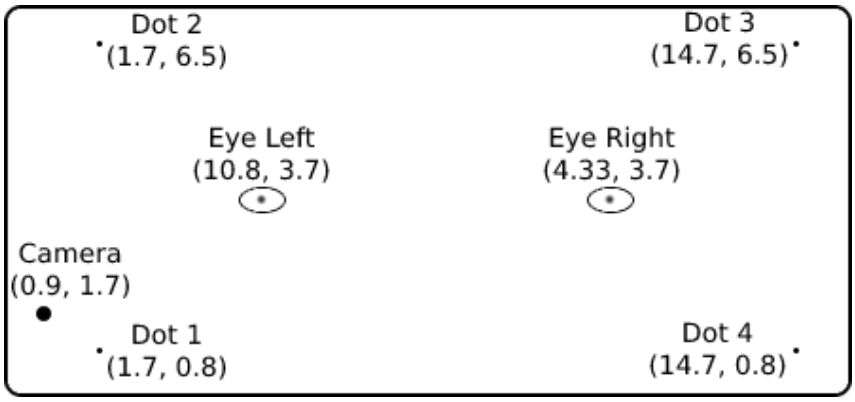


Fig. 19: Measured points on the screen. All measurements are in centimeters

Table 2 shows all the calculated data using Equation 1 for all permutations of distance to the device and positions of dots and eyes.

Distance	Dot Index	Eye (L/R)	Angle (Horizontal/Vertical)
20cm	1	L	(4.94°, 3.33°)
		R	(22.36°, 3.33°)
	2	L	(4.94°, −12.86°)
		R	(22.37°, −12.86°)
	3	L	(−29.47°, −12.86°)
		R	(−13.50°, −12.86°)
	4	L	(−29.47°, 3.33°)
		R	(−13.50°, 3.33°)
25cm	1	L	(3.95°, 2.67°)
		R	(18.22°, 2.67°)
	2	L	(3.96°, −10.35°)
		R	(18.22°, −10.35°)
	3	L	(−24.32°, −10.35°)
		R	(−10.87°, −10.35°)
	4	L	(−24.32°, 2.67°)
		R	(−10.87°, 2.67°)
30cm	1	L	(3.30°, 2.22°)
		R	(15.34°, 2.22°)
	2	L	(3.30°, −8.65°)
		R	(15.34°, −8.65°)
	3	L	(−20.64°, −8.65°)
		R	(−9.09°, −8.65°)
	4	L	(−20.64°, 2.22°)
		R	(−9.09°, 2.22°)

Table 2: Calculations made using data from Figure 19 and Equation 1

6.4.4 Setup - BLOB Analysis of Iris Detection

MediaPipe Iris doesn't track the subject's gaze, but we were interested in researching whether BLOB analysis of the detected iris could offer some estimate of gaze direction.

Since the iris is a constantly sized circle, we expect it's possible to approximate the gaze direction by interpreting the iris outline as a tilted, perspective-warped circle.

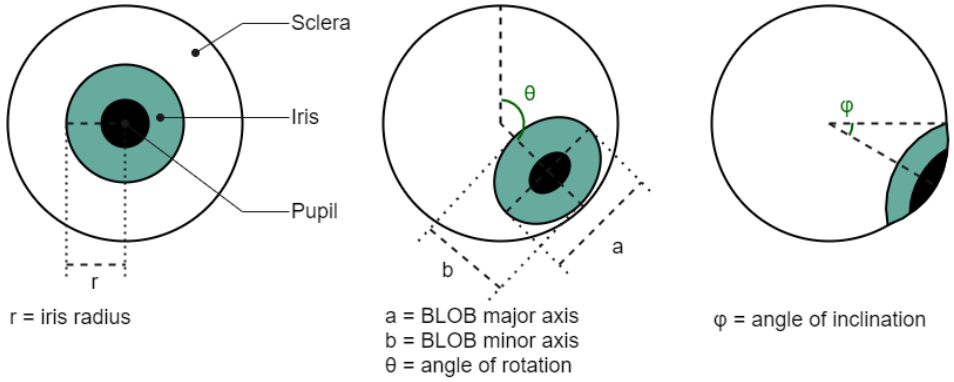


Fig. 20: Simplified geometric sketches of the eye. The center and right sketches show the angles of interest: the angle of rotation (θ), and the angle of inclination (φ).

Figure 20 shows how the direction of a subject's gaze can be described in polar coordinates: An angle of rotation (θ) around the axis between eye and camera, and an angle of inclination (φ), describing the degree to which the subject is looking away from the camera. Once these are obtained, it's simple to convert to a Cartesian angle measure, if necessary. Interpreting gaze direction to a point on the screen of a device would require additional calculations, but obtaining the angle was sufficient for this iteration.

By having the MediaPipe framework detect the iris outline, BLOB analysis can obtain the major and minor axes of the iris. The angle of rotation (θ) can then be obtained through the angular offset of these axes compared to the image axes.

The angle of inclination (φ) is found through the ratio between the major and minor axis lengths. Since φ is a rotation around the major axis, it does not affect the perceived length of the major axis, but it does affect the minor axis: The greater the angle of inclination, the shorter the minor axis appears on the image.

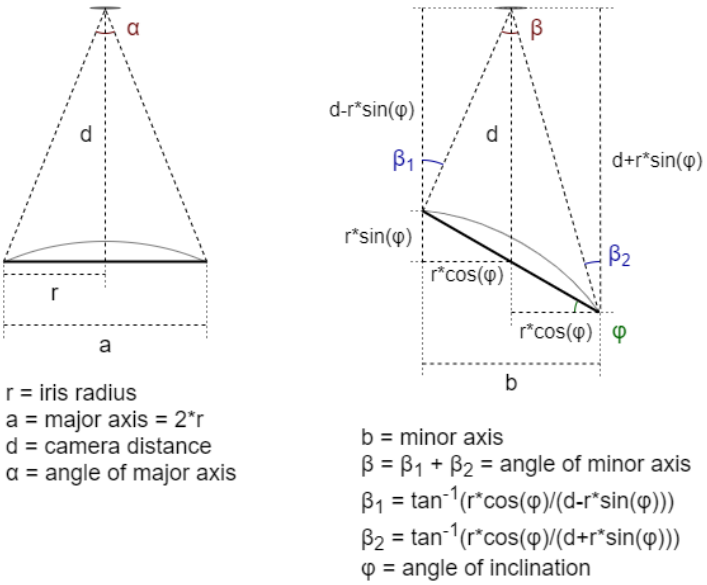


Fig. 21: Sketches showing the iris and camera lens as viewed along the minor axis (left) and major axis (right). The right sketch shows the relationship between β and φ.

As mentioned earlier, MediaPipe relies on the fact that the majority of humans have a constant iris diameter in order to determine the distance between eyes and camera. This is done by using the focal length of the camera to interpret distances on the image as angles, and then using the constant iris diameter to determine the distance.

In a similar vein, by expressing the minor axis as an angle (β), we can use the distance and iris diameter assumption to approximate the angle of inclination, see Figure 21.

6.4.5 Results

Unfortunately, we were unable to obtain any data from this experiment. Despite multiple attempts, we were unable to implement the MediaPipe framework on any device due to a multitude of problems. These include spaces and special characters in folder names breaking the installation, issues hooking up Linux subsystems to connected webcams and phones, and problems building the example projects provided by MediaPipe[12].

6.4.6 Conclusion

MediaPipe Iris is too troublesome of a solution. As of the writing of this project, the MediaPipe team is working on an alternate solution[13], but for the time being, we move on to explore a different framework.

We made the mistake of investing a lot of preparatory work in a solution we couldn't get working. At this stage, we only need a simple experiment that can help us determine whether pursuing a King-Devick-style test is feasible, which we can do in an easier way using an SDK that offers a more complete solution.

[12] `https://github.com/google/mediapipe/tree/master/mediapipe/examples/android`
[13] `https://google.github.io/mediapipe/solutions/iris`

6.5 SeeSo Gaze Tracking

6.5.1 Goal

We turn our eyes to more complete gaze tracking solutions, and now wish to explore the capabilities of a different framework: SeeSo.

The goal now is to successfully implement SeeSo, and show that it can track a subject's gaze on the screen of a mobile device.

6.5.2 About the Model

SeeSo is an easy to implement SDK that uses the front-facing camera to determine where the user is looking on the mobile device's screen, and outputs gaze information (including coordinates and the estimated type of eye movement) 30 times a second.

SeeSo calculates the distance between the camera and the subject's face by constructing similar triangles between the image plane and the subject[28]. The first of these triangles is composed of a known, or rather assumed, width of the subject's face w_s and the distance d to the subject's face. The second triangle is composed of the focal distance f of the camera and the width of the image plane w_i.

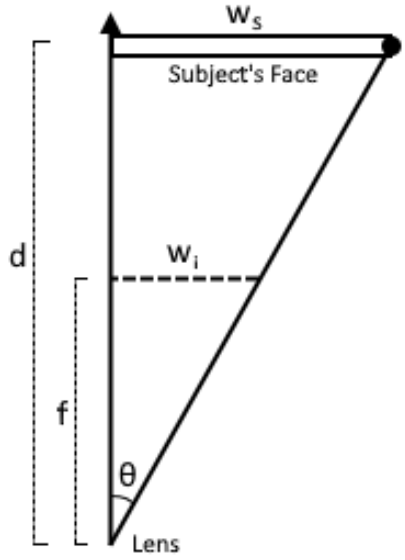


Fig. 22: Diagram showing the similar triangles and the variables used when calculating the depth to the subject's face

$$tan(\theta) = \frac{w_i}{f} = \frac{w_s}{d} \Leftrightarrow d = \frac{w_s}{tan(\theta)} \tag{2}$$

Equation 2 shows how the distance d to the subject's face is obtained from the focal length f and feature sizes in the image w_i and subject w_s. The focal distance can be calculated using SeeSo's camera calibration, or from lens information obtained from the device. SeeSo's own documentation is unclear, but subject features likely rely on constant widths similar to how MediaPipe relied on a constant iris width among subjects. The image features are identified by SeeSo's deep learning model.

While SeeSo does not provide any information regarding how their deep learning framework functions, we can look at other gaze tracking projects such as Pupil Invisible Supportive Tool (Pistol) and explain roughly how it works under the assumption that SeeSo follows a similar model. We will keep this explanation cursory and limited strictly to components relevant to gaze tracking on a flat screen.

The Pistol model has several processing steps, of which those relevant to gaze tracking are: Feature detection, eyeball and vector computation, mapping and synchronization of frames, and gaze estimation.[29] See Figure 23 for an overview of the workflow.

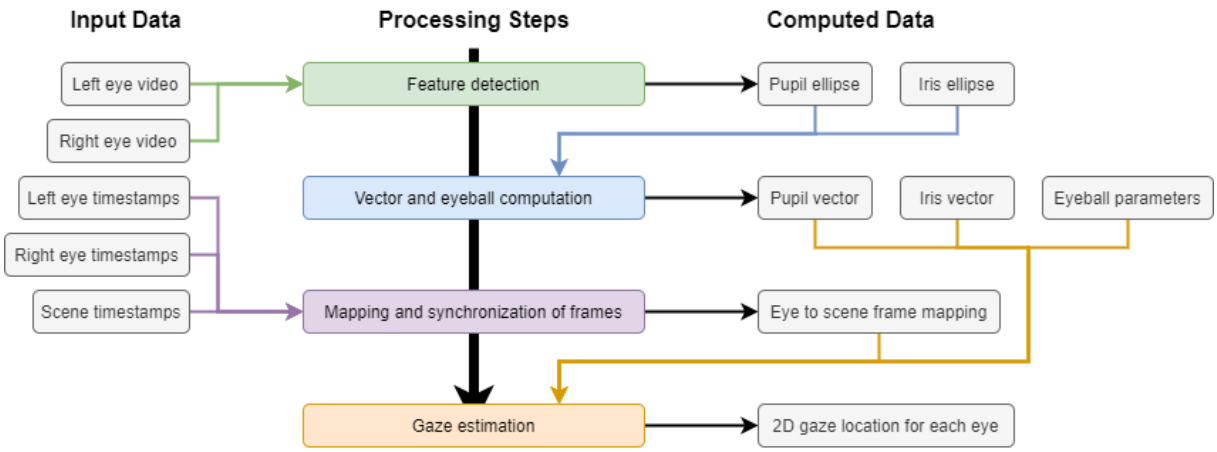


Fig. 23: An excerpt of the workflow model presented in the paper by Fuhl et al. on Pistol.[29] The model is edited to show only the data and process steps relevant to 2D gaze tracking.

Feature detection feeds a gray-scale image to a deep neural network which extracts data for the ellipses describing pupil and iris shapes.

Eyeball and vector computation is again performed using deep learning. 100 of the most different pupil ellipses are selected from the previous step, which the deep neural network uses to compute approximate models of the eyeball. Pupil and iris vectors are computed as the vector intersecting the center of the eyeball model and the pupil/iris.

Pistol uses two different processing models for estimating gaze, and separately uses pupil center, iris center, pupil vector, and iris vector to produce a total of 8 gaze point estimators for each eye. It's unclear how these different estimators are rendered into a final estimation, but we assume the 8 estimators are used to validate the accuracy of data in order to filter out any errors that occur during processing.

In summary, Pistol (and likely SeeSo as well) perform gaze point estimation by supplying image data to a complicated series of deep learning models that extract relevant features, compute eye models, and estimate gaze direction.

All of this is done automatically by the SDK and calculated locally on the device without sending data to an external server. The process by which gaze information is provided to the developer is outlined in the figure below, where all the technical calculations mentioned above happen in Step 3. Authentication is only used to validate that the application has a valid license to use the SeeSo SDK.

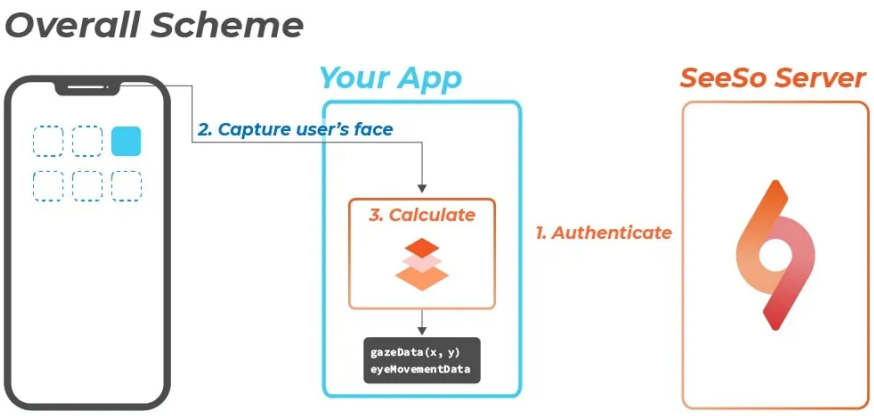


Fig. 24: SeeSo's execution pipeline[30]

6.5.3 Setup

The SeeSo SDK will be used to create a simple Android application that gathers and exports the gaze data to a .csv file, which we will then be able to use to determine whether the SDK is accurate enough to render where the user is looking. Specifically, two brief experiments will be run. In the first experiment, we will look at the four corners of the device. In the second experiment, we will slowly shift our gaze along the edges of the device. As there is no motion along the edge, this eye movement of course won't be smooth pursuit, but rather a series of fixations along each edge.

Once we export the data, we will draw density heat maps for each experiment, where we should be able to clearly see four corners and a square representing the edges of the device, respectively. It should be noted that we will not utilize SeeSo's calibration system, so we don't expect the experiments to provide perfectly accurate data, but it should be good enough to roughly validate its use for tracking the user's gaze during a King-Devick test.

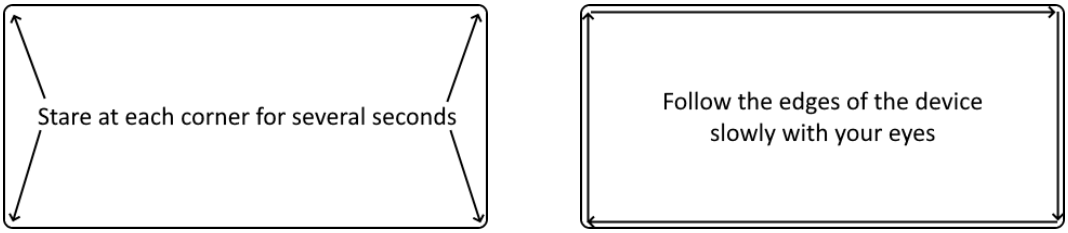


Fig. 25: Illustrations showing how the two experiments were carried out

6.5.4 Results

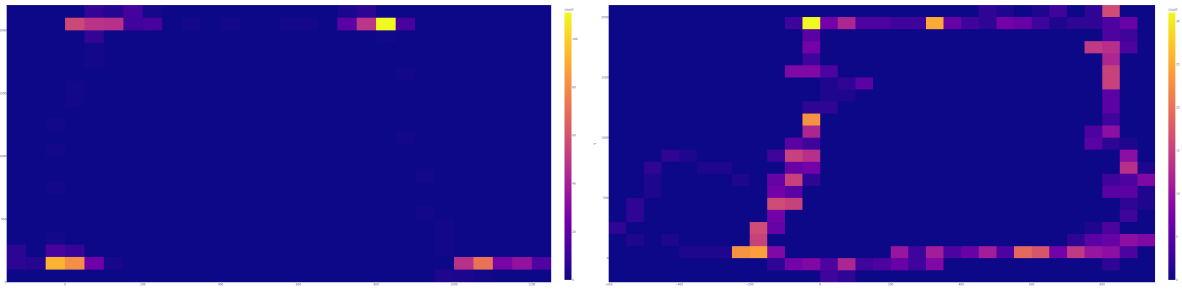


Fig. 26: Density heatmaps representing the coordinates in screen space collected from SeeSo generated using a Python script[31]

In Figure 26, the left plot shows staring at the 4 corners of the screen, whereas the right plot shows following the edges of the device with the user's gaze.

While it is clear from the plot that the gaze tracking is not perfectly accurate, it is good enough to be very evident what the user is looking at in both cases. The plot where the user is staring at four points has an issue tracking the two right-hand points properly, as they seem to be offset horizontally by a relatively large margin. The other plot where the user followed the side of the device has the same issue, which is evident by the left line that should be relatively vertical but appears to be slightly diagonal.

The right plot has another artifact that needs to be explained, which is the small amount of data to the left of the rectangle. SeeSo reports gaze data even if the user looks outside the dimensions of the device's screen. This data can be explained by the test person looking off-screen briefly.

6.5.5 Conclusion

The MediaPipe SDK turned out to be a dead-end, but we had much more success using the SeeSo SDK. It was very straightforward to get up and running in Android Studio and is reasonably well documented. The density heatmaps gathered from our experiment verify that roughly tracking the gaze of the user is

possible without calibration, and we expect this to become much more accurate if we decide to pursue this type of test and properly implement calibration.

The conclusion from this experiment is that performing gaze tracking on a mobile device is indeed viable. This type of gaze tracking, which outputs a position on the screen, is especially suited for King-Devick-style testing, since that would require describing how a subject's gaze moves across each number displayed on the screen.

Although we can find other uses for this method of gaze-tracking, we will not be able to use this for analyzing saccades automatically. This is due to SeeSo having an update rate of 30Hz. Saccade movements usually occur on a scale of 15-100ms, i.e. approximately 10-70Hz.[19] In the worst case, we would be losing more than half of all relevant data points, rendering analysis faulty and potentially misleading.

It's possible that this gaze tracking would also work for smooth pursuit tests, for example by displaying a dot on the screen for the patient to focus on. However, since the screen would have to be turned toward the patient, the assessor would potentially have a hard time determining if they are keeping the patient in frame while moving the device back and forth.

6.6 Speech-to-Text

6.6.1 Goal

A proposed feature of the King-Devick design is using the smartphone's audio recording capabilities to verify that the user makes no errors during a test. The consequence of an uncorrected error is that a patient must restart their baseline test, or that they fail their screening test, so it's important such errors are caught.

We can do this by simply recording the audio and playing it back to either the patient or an assessor, allowing them to determine whether an error was made after a test card has been read out loud.

Alternatively, we may attempt to streamline this process by automatically transcribing the number sequence spoken by the user and comparing it against the actual sequence. The experiment described in this section is meant to assess the possibility of implementing such a feature in our solution.

Speech recognition has been a common staple of smartphones for a while now and continues to improve. As such, unlike more complicated tasks such as eye detection and image analysis, speech recognition is a built-in feature of the Android class library.[14] The question to be answered by this experiment is not "Can we recognize a user's speech?" but rather "How accurate is Android's native speech recognition?"

6.6.2 Setup

We design a small app implementing Android's native `SpeechRecognizer` that can be started and stopped with button presses, and that displays interpreted text on screen. See Figure 27.

Fig. 27: Screenshots of the test app for performing speech to text.

In English, we read aloud each of the four King-Devick test cards from Figure 7, and compare the output of the app with the contents of the test card. The experiment is performed while seated with the phone lying on a desk, approximately 40cm from the speaker.

6.6.3 Results

The experiment results in the app returning the following four strings:

```
250 8394 66047 01535 837 530
258 07794 6364 7973 554 92657 33186 475-2
37-590 257 4647 6379 3904 5275 37048 75290 236
54180 46590 7547 3269 453 93455 16314 30527
```

Trimming the strings of whitespace and hyphens is a simple enough task, but the resulting sequences still do not match the sequences found in Figure 7.

[14] `https://developer.android.com/reference/kotlin/android/speech/SpeechRecognizer`

Demo		Test 1		Test 2		Test 3	
E	R	E	R	E	R	E	R
2	2	2	2	3	3	5	5
5	5	5	5	7	7	4	4
	0	8	8	5	5	1	1
8	8	0	0	9	9	8	8
3	3	7	7	0	0	0	0
9	9	3		2	2	4	4
4	4	7	7	5	5	6	6
6	6	9	9	7	7	3	
8		4	4	4	4	5	5
3		6	6	6	6	9	9
6	6	5		1			0
7	0	3	3	4	4	7	7
4	4	7		7	7	5	5
6		6	6	6	6	4	4
7	7	4	4	3	3	2	
0	0	7	7	7	7	7	7
1	1	9	9	9	9	3	3
5	5	7	7	3	3	2	2
3	3	3	3	9	9	6	6
5	5	5	5	0	0	9	9
8	8	1		4	4	4	4
3	3	5	5	5	5	1	
7	7	4	4	2	2	4	
5	5	9	9	1		5	5
3	3	2	2	7	7	1	
0	0	6	6	5	5	3	3
		5	5	3	3	9	9
		5		7	7	3	3
		7	7		0	4	4
		3	3	4	4	8	
		3	3	8	8	5	5
		1	1	7	7	5	5
		8	8	4		1	1
		6	6	6		6	6
		4	4	5	5	3	3
		5		2	2	1	1
		3		9	9	4	4
		7	7	0	0	3	3
		5	5	2	2		0
		2	2	3	3	5	5
				6	6	2	2
						7	7

Table 3: Table comparing the values of each King-Devick test card (see Figure 7) to the recorded response from the app utilizing Android's `SpeechRecognizer` class.
E = Expected value, R = Recorded value.

The expected and recorded sequences have been lined up for comparison in Table 3. Two notable observations can be made here: First, whenever an extra value appears in the observed results, the number is 0. Second, all errors can potentially be classified as missing values or extra values. The only exception is the (7|0) error in the Demo column, which can be argued to in fact be a sequential missing value error and extra value error. Table 4 shows the types of errors observed in the speech-to-text experiment and at which rates they occur.

	Demo	Demo*	Test 1	Test 2	Test 3	Avg	Avg*
Total Error Rate	20.0%	24.0%	17.5%	12.5%	20.0%	17.5%	18.5%
Missing Value Rate	12.0%	16.0%	17.5%	10.0%	15.0%	13.6%	14.6%
Extra Value Rate	4.0%	8.0%	0.0%	2.5%	5.0%	2.9%	3.9%
Wrong Value Rate	4.0%	0.0%	0.0%	0.0%	0.0%	1.0%	0.0%

Table 4: Various error rates for each test card and averages. Columns marked with * treat the (7|0) error in the demonstration card as two separate missing value and extra value errors.

To determine if these are the only types of errors we can expect to see, we conducted the same experiment for two other speakers. In both cases, the total error rate was 0, suggesting that the initial experiment was not necessarily indicative of the behavior we could expect. It would seem that there are perhaps certain speech patterns or voice types that Android's speech recognition struggles to comprehend.

6.6.4 Conclusion

It would be prudent to repeat this experiment on more subjects to determine how common errors truly are, but during implementation (see Section 8.8), we found that audio recording and Android's built-in speech-to-text feature were mutually exclusive, rendering the finer details of this experiment moot.

In theory, given a speech-to-text framework that is compatible with our needs and is at least as accurate as that of Android's, the feature could be a valued quality of life feature capable of accurately highlighting potential errors for most users. Certain users would frequently experience false positive errors, which may be confusing, but user experience testing could reveal whether those drawbacks are outweighed by the benefits of automatic validation.

6.7 Conclusion

On a technical level, we find King-Devick-based screening methods to be more viable than VOMS-based screening methods, since we were unable to implement the phone movement model required for VOMS. SeeSo gaze tracking works fairly well, and while we no longer plan to use this to automatically analyze saccade movement, we can still use it to roughly record the user's gaze on the screen and play it back during a review process. This will help assessors of the application make rough judgments as to whether the patient was looking at the digit they were reading aloud.

Before we make the final decision to go with either VOMS or King-Devick, we will need to complete a user experience assessment to establish which of the two methods users prefer.

7 Preliminary User Experience Experimentation

While the technical implementation of our two potential solutions is crucial for the final decision regarding which to pursue, an equally important aspect is the experience while using either. User experience is important to consider, as potential stakeholders are not going to use our product if they don't find it an enjoyable experience. As such, this section of the report will focus on this aspect without considering what was concluded in Section 6.

Of the two solution candidates presented in Section 5, we wish to determine what solution is preferred among users, and what can be done to improve user experience of our designs.

The users we test in this section are all part of our social network: Friends, family, and colleagues. We realize that we would obtain more valuable data testing our customer segment, but doing so is impractical for several reasons. We have not been able to contact members of the military or sports medics, as described in Section 3, and there are a number of ethical issues surrounding testing on people who suffer or have suffered from concussions. For example, according to Dennis Abildgaard, VOMS and King-Devick are effectively designed to provoke discomfort in patients suffering from ocular-motor dysfunctions. Testing our solution on these people could provoke major discomfort and fatigue, possibly affecting them for the rest of the day.

7.1 Roleplay & Questionnaire

We have decided to create a roleplay scenario where we take the test participants through a series of scenarios mimicking the real-life use case of our potential solutions. After each scenario, we present them with a few questions to gather insight into how the experience was. This will help us compare and contrast the different solutions, and determine what they do well and in which aspects they need to be improved.

Each testing session starts by briefly introducing the subject to the overall thesis and goal of our project in order to give them a surface-level understanding of why we are asking them to perform these tests, while still keeping the descriptions vague enough as to not introduce any biases towards any particular screening method.

The experiment is set up in three different sections, each section testing a separate potential solution discussed earlier in the report:

1. Testing smooth pursuit where movement is linear
2. A variation of testing smooth pursuit where movement is circular
3. Testing saccades based on methods from King-Devick

Linear Smooth Pursuit Test

The first section tests smooth pursuit using methods learned from VOMS as explained in Section 5. We start by explaining to the test subject how the test procedure is executed. The subject initially roleplays the patient, and as we execute the procedure, we once again explain what we are doing. After this, we switch roles so we become the patient in the session, and the test subject becomes the assessor. This way, they get to experience both how it feels to conduct the test, and also what the experience is like for the patient.

The VOMS-based linear smooth pursuit test is executed as follows:

1. Open the camera app on your phone. Don't record anything, but make sure the camera's output is visible on screen
2. Ask the patient to look into the camera at all times during the test.
3. Keep the phone roughly a meter away from the patient's face.

4. Move the phone half a meter to the left and back to the center.
5. Move the phone half a meter to the right and back to the center.
6. Steps 4-5 should take 4 seconds in total.
7. Repeat steps 3-5 once more.
8. Repeat steps 3-7 once more, but where the linear movement is up and down instead of left and right.

Circular Smooth Pursuit Test

A variation of this test is also conducted where we do circular motions instead of linear motions, but everything else remains exactly the same, including the questions which are asked after the test has been conducted. The procedure is therefore as follows:

1. Open the camera app on your phone. Don't record anything, but make sure the camera's output is visible on screen
2. Ask the patient to look into the camera at all times during the test.
3. Keep the phone roughly a meter away from the patient's face.
4. Move the phone a half a meter up
5. Move the phone in a clockwise circle, retaining a half-meter radius
6. Steps 4-5 should take 4 seconds in total.
7. Repeat steps 3-5 once more.

Saccade Test

The final section tests saccades using standard King-Devick flashcards. Like the smooth pursuit tests, we explain to the test subject how the procedure is executed, this time while showing them a demonstration flashcard. The flashcards are simply saved as images on our phones, so when we conduct the test, it is done through the default photo app on our phones. After explaining the test to the test subject, the phone is handed over to them, and they are free to do a practice run on the demonstration card if they please. Once they are ready, they swipe to the next flashcard and the test commences following standard practices when executing a normal King-Devick test:

1. Read each digit out loud from left to right, starting in the top left corner
2. After reading the digit in the bottom right corner out loud, switch to the next flashcard
3. Once the test has been finished a score can be calculated, although this was not done during the UX experiment

Key Takeaways

The full summary of all responses can be found in Appendix 12.2, but we will present a brief overview of the key takeaways from the experiments here.

Users generally reported that both smooth pursuit movements were doable, although there seemed to be a general consensus that it requires practice to do comfortably. Furthermore, subjects report that the circular motion was more difficult than the linear motion.

It should be noted that we also noticed several subjects rated the movements to be easy, even though it was obvious to us as observers that they didn't do the movement correctly, both by spending more than 4 seconds completing the movements and also by not extending the device a half meter. We think

this is caused by it being difficult to self-evaluate whether you're doing the movement correctly, which we intend to address using some of the feedback methods mentioned in the questionnaire. Seeing as all subjects know us personally, it could also be caused by a conscious or subconscious positivity bias in terms of wanting to view your colleague or family member's work in a positive light.

Subjects reported that the King-Devick test was more difficult than the smooth pursuit tests because they were more stressed, which we assume was probably caused by having to read the numbers aloud while under time pressure. Several subjects also noted that they found the test to be interesting, which we believe is due to the test requiring more participation from the testee than the smooth pursuit test where you simply have to look into the camera. Finally, all test subjects reported that they would want the application to provide gaze data when reviewing the results for a test. The responses were split 50/50 in terms of whether the app should attempt to automatically provide a result, or if that should be entered manually by a doctor.

Of the three tests, the King-Devick test was overwhelmingly preferred by test subjects.

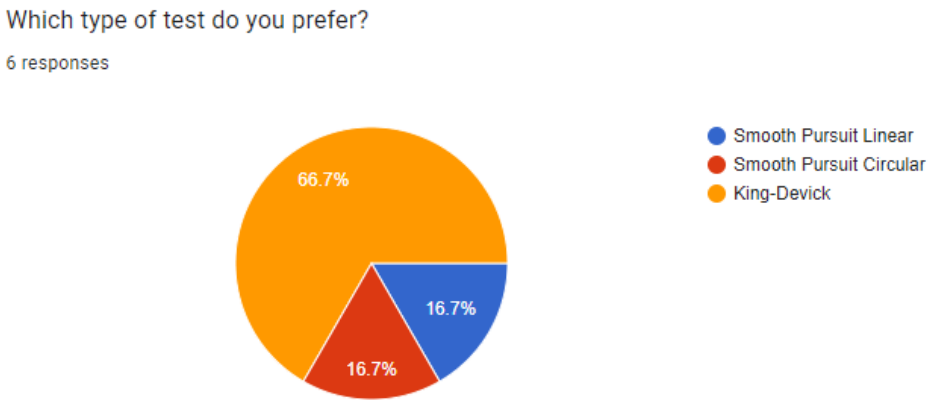


Fig. 28: Results for the question "Which type of test do you prefer?"

For the remainder of Section 7 we will therefore focus on more detailed designs for the King-Devick test.

7.2 King-Devick Wireframe Questionnaire

We decided to create a quantatively-focused questionnaire featuring wireframes for the most important parts of the application. Each question presents the subject with two or three variations of the same wireframe and asks them which they prefer with an optional qualitative question where they can elaborate on why they chose the wireframe they did. This not only tells us exactly how several aspects of our application should look, but it also provides us with a clear guiding star in terms of what experience users are looking for in a broader sense. We received 11 responses in total for 6 different questions, all of which can be seen in detail in Appendix 12.3 and 12.4.

Front Page
The front page wireframes have two variations. The first variation splits the front page into two separate screens. The first page contains a "Start Test" button which navigates the user to the second page where they can select between a baseline test or a post-injury test. The second variation removes the "Start Test" button and moves the baseline- and post-injury test buttons to the first page, eliminating the need for a second page and navigation.

Responses indicate that users are split almost evenly between the two options, although the second option with a single page has a slight lead. The qualitative answers indicate that subjects who selected Option 1 prefer it to keep the user interface cleaner, whereas subjects who prefer Option 2 do not think it looks cluttered and find the "Start Test" button to be confusing as it navigates to a new page instead of starting the test immediately.

As Option 2 is slightly preferred and we want to avoid confusion at all costs due to the serious nature of

the application's use case, we have opted to go with Option 2.

Demonstration cards
The demonstration card wireframe has two variations. One of the variations simply shows the demonstration flashcard, whereas the other has a fullscreen popup with instructions on how to proceed with the test. Results indicate that people overwhelmingly prefer the fullscreen popup with instructions as 92.3% selected that option. The qualitative feedback indicates that users find the instructions to be necessary, including the one person who selected the option without the fullscreen popup, as they stated they would only prefer it if there was someone there telling them how to complete the test verbally.

Center Play Button
After the test subject has completed a flashcard, a review window appears where they can review their attempt. This window includes a few UI elements which let the user of the application start and stop the playback and scrub along a timeline to playback specific parts of the test attempt. Two variations are presented in the questionnaire, one where the play/stop button is large and along the side of the window, and another where it is small and located in the center of the screen.

Results indicate that 92.3% prefer the larger button off to the side, although some mention that it is a bit too large. Another user in favor of the large buttons states that they did not realize the variation with the small button even had a playback feature. The one user who preferred the small button stated that they did so because they would be less likely to press it by accident.

Gaze Indicator
In extension to the previous question regarding the review window, we presented three variations of the gaze indicator in the questionnaire. The first option has an outline and a large radial gradient, the second is a small dot, and the third option is a large radial gradient without the outline. 76.9% preferred the first option which has an outline and a large radial gradient. Responses indicate this is because it has higher contrast and due to its size, it will probably also encompass the error we may have in our gaze prediction. They think the second option is too small to accurately describe where we have detected the person is looking.

Active Frame Indicator
Continuing with questions regarding the review window, the replay system lets you author the results by clicking on a number on the flashcard. When you've done so, an indicator may appear showing you which number you're editing. We presented them with three variations where the first draws a line from the sidepanel to the currently selected number, the second draws an additional frame around the currently selected number, and the third does not draw any indicators for the currently selected number.

Results indicate a preference for Option 1 with 69.2% in favor, with the remaining 30.8% preferring Option 2. The third option, which does not draw any indicators for the currently selected number, was not preferred by anybody. One response, which preferred Option 2, suggested we blur the rest of the window when selecting a number to more clearly indicate which you had selected.

Final Results Page
The final results page is intended to communicate to the patient what the result of their test is. We presented two variations where the first displays some raw data regarding what the current test score is and what the threshold is for flagging a score as being worrisome, while the second option draws the same data in a graph to communicate it visually.

53.8% preferred Option 1, which does not draw the graph, with the remaining 46.2% preferring Option 2. One response, which was in favor of Option 2, states that it gives the user a more intuitive understanding of how the application arrives at the conclusion. Another user, who favors Option 1, states that a number in accordance with a scale is easier to relate to than a graph.

7.3 Conclusion

We have determined that users prefer to conduct a King-Devick test, rather than VOMS-based smooth pursuit tests. Furthermore, we have designed several wireframe variations and have received very clear feedback on which users prefer and why they do so, giving us a solid foundation from which to start developing the application.

8 Application Implementation

The user experience tests conducted in the previous section and the technical experiments of the section before that provide us with an excellent foundation to begin development of the King-Devick baseline and screening app. We know what is and isn't possible, and we know what is and isn't preferred among users. What's left is the implementation of ideas in developing a functioning app.

Our intention is to develop a minimum viable product that adapts the specifications of a King-Devick test accurately, while improving upon it with some additional features provided by modern smartphones. In total, we are looking to implement the following:

- A demonstration flashcard and three test flashcards whose layout matches that of an analog K-D flashcard.
- Flashcard digits should be randomized to avoid patients memorizing the sequence.
- The patient's gaze should be recorded and be able to be played back in a performance review mode.
- The patient's voice should be recorded and be able to be played back in a performance review mode.
- The patient's voice should automatically be analyzed, and the application should make an assessment as to whether the patient correctly read all the digits out loud.
- A baseline test implemented in accordance with K-D specifications. The baseline score should be recorded in permanent storage, so it can be used at a later time.
- A post-injury screening implemented in accordance with K-D specifications. Post-screening, the application should make an assessment as to whether the patient should seek medical attention.

In order to make development smooth, the app was divided into two parts that could be worked on independently of each other. Daniel handled the implementation of the wireframe presented and refined in Section 7.2, ensuring a proper visual design and user flow (described in Section 8.3). Noah was responsible for the implementation of more advanced functions such as gaze tracking and speech recognition (described in Sections 8.4-8.8).

8.1 Android Development Background Information

The Android SDK contains terms for many concepts that a developer must understand in order to successfully develop an app. The concepts and terms used in the production of our application are explained in this section. Most of this information can be found in the Android Developer Guides website[15].

Activity
Equivalent to a window within the application that contains elements of the user interface.

Business Logic
This term refers to functionality not strictly related to the UI, such as managing the data from a gaze tracker recording or handling the media player for an .mp3 file.

Manifest File
XML file which describes the content of the application, such as which activities it has and which permissions it requires to execute.

Fragment
A group of UI elements that has its own lifecycle and can handle its own input. Fragments cannot exist independently and must be contained within an activity or other fragments.

View
Base UI class used to draw elements on the screen.

[15] `https://developer.android.com/guide`

DRY (Don't Repeat Yourself)
A principle in programming that aims to reduce the amount of duplicate code in an application[16]

Intent
Data class which contains information regarding how to start an activity

Test Instance
This term is defined by us for this project, and denotes a data entity that contains information regarding a single instance of the King-Devick test, from the first demonstration flashcard is shown until the final test flashcard has been completed and reviewed.

Test Session
This term is defined by us for this project, and denotes a data entity that contains information regarding a test from the moment the given test button is pressed until the test has concluded. A baseline Test Session will run through 2 Test Instances before it has been completed.

8.2 Kotlin vs. Java

Android Studio, one of the primary IDEs used to develop Android apps, supports developing Android applications using both Java and Kotlin. While Kotlin and Java are technically interoperable, letting you develop an application that leverages both languages, we would like to focus on a single language throughout our development. We have made this decision partly because it simply seems like good practice to stick to using one language, but also because Java/Kotlin interoperability comes with caveats[17] that we would rather avoid having to deal with. As such, we have to weigh the costs and benefits of each language and choose one before we start our implementation.

Kotlin is an open-source, statically typed object-oriented language used by a majority of Android developers. Android Studio has built-in tools for Kotlin and will select it as the default language over Java when creating a new project. While neither of us have any formal experience using Kotlin, we find the syntax to be familiar to other statically typed object-oriented languages we do know like Java, C#, and C++.

Java is known for its ubiquity and multi-platform support. We both have experience using Java from several courses at DTU, including a course that taught Android development in Java. We are both comfortable writing code in Java and would find re-learning Android SDK concepts a larger obstacle than refamiliarizing ourselves with the Java syntax.

Since Kotlin is a relatively new language and seems to be favored by Google, we get the impression that it is the de facto preferred language to use for Android development and that Java support will slowly be phased out over the coming years. As we are more familiar with Java and its syntax, using Kotlin definitely poses a risk to the development of our application, as we might hit certain roadblocks which we would more easily overcome in a Java environment. We also find a lot of value in learning what, to us, seems to be the future of Android app development, rather than choosing the safe, but probably somewhat deprecated, option of continuing to use Java. For this reason, we will primarily use Kotlin to develop our Android app.

8.3 Wireframe Implementation

8.3.1 Use of Fragments

Fragments can help reduce code bloat and prevent violations of the principles of DRY by letting you reuse groups of user interface elements across the application. Communicating information to fragments is more cumbersome than simply retrieving references to the individual elements that may exist in an activity using `findViewById`, and as such there is a trade-off when deciding to use fragments for the user interface. We found that the balance lies in making the user interface modular, so we use fragments for groups of elements that we know need to be used in several places.

[16] `https://www.geeksforgeeks.org/dry-dont-repeat-yourself-principle-in-java-with-examples/`
[17] `https://kotlinlang.org/docs/java-interop.html#escaping-for-java-identifiers-that-are-keywords-in-kotlin`

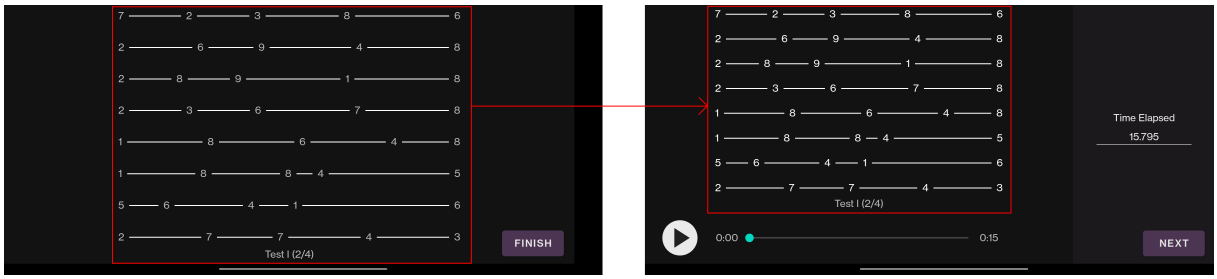


Fig. 29: Two different activities in the application, both of which using the same fragment for displaying a King-Devick flashcard

While using viewmodels does provide an elegant way of communicating with fragments, as seen in Section 8.6, we did not discover this until later in the project, at which point we had already implemented most of the user interface using a less elegant solution. This solution relies on the callback `onAttachFragment`, which is invoked on the owning activity when a fragment is created. The ID of the given fragment is queried to identify which instance in the layout the invocation is regarding, at which point the reference can be downcast into the fragment's type and used directly.

The main menu uses this method to initialize the two buttons it contains. Fragments can be initialized using a `Bundle`[18], which is a simple data container of key-value pairs that is passed to the fragment. The main menu uses this method to initialize its buttons, and while an argument could be made that this is cruder than the viewmodel method, it is also a lot simpler. Since the buttons need only be initialized and enabled after SeeSo has been initialized, this may be preferred over the viewmodel method in this case. This definitely played a large role in our decision to keep this implementation instead of prioritizing cleaning it up with a more elegant solution.

8.3.2 Application-Spanning Data

The state of the application often has to be communicated across several different activities. Simple primitive data can easily be passed from one activity to another by adding it to the `Intent` that instantiates the new activity. Passing complex data objects between activities requires additional care, and there are several ways of accomplishing this, each of which is useful in different situations.

During runtime, our application has to store data regarding the currently running test, which includes information regarding whether it's a baseline or not, the scores across several different flashcards, and so forth. We also need to be able to easily wipe data that is no longer needed. This is used when an entire baseline or post-injury test has been completed (Test Session), or between two different screenings during the baseline test, as it has to run twice (Test Instance). If a Test Session is wiped, any ongoing Test Instance data must also inherently be wiped. Finally, data should be easily accessible from anywhere in the application, and adding new data should be easy.

To summarize, our application-spanning data storage solution needs to support the following features:

1. Arbitrarily complex data
2. Clear Test Session data
3. Clear Test Instance data
4. Automatically clear Test Instance data if Test Session data is cleared
5. Easily add new data fields

We will now examine several different potential solutions and explain their usages, benefits, and drawbacks.

[18] `https://developer.android.com/reference/android/os/Bundle`

Use Primitives
In theory, we could flatten the data structures and store all the data as primitives in the `Intent` using key-value pairs. While technically feasible, this would be extremely cumbersome to work with and lead to bugs where some of the Test Session or Test Instance data was cleared, but not others.

Parcelable
While the `Intent` class only supports storing data from primitives out of the box, it can also handle complex data types if they implement the `Parcelable` interface. This also lets us utilize data composition, which can be useful when clearing Test Session and Test Instance data, as Test Instance data could be a field in the Test Session data, thus seamlessly handling requirement 4.

A drawback of this approach is that you have to be extremely careful when passing data to the next activity, as forgetting to put data into the `Intent` will implicitly clear it.

JSON
JSON supports hierarchically categorizing data into objects, which we can use to support requirement 4. Kotlin also has native support for converting objects into JSON strings and back, which lets us easily add data fields and thereby supports requirement 5.

Unfortunately, this method suffers the same drawback as the `Parcelable` interface, as we would need to pass the JSON string along to activities using `Intent`. In essence, this is just a slightly more complicated way of doing the same as with `Parcelable`.

Static Data
Kotlin does not have a static keyword as similar languages like Java and C# do. Instead, it relies on what it calls a companion object[19] to achieve what is essentially the same goal, i.e. data and functions that can be accessed without explicitly using an instance of the given type.

This can be used to store data across activities seamlessly. Once the data has been generated, it remains in memory until it is cleared manually, which resolves the issues present when using `Intent`. Furthermore, since it is just a normal class, we can still take advantage of data composition to support requirements 2-4.

Persistent Data On Disk
Google seems to push the use of Room[20] to store persistent data on Android, which is a wrapper framework for using an SQLite database. There is little reason to spend time considering how we would effectively utilize an SQL database to support the requirements stated in this section, as storing runtime data in persistent storage is something we can reject outright. Not only is it bad practice, but it can also create a lot of issues with data not being cleared correctly, causing bugs even across restarts of the application.

Application Class
This solution has many similarities to the Static Data solution, except that it does not use the companion object and will exist as an actual instance. The Android SDK lets you specify a class type which it will instantiate when the application is launched. The instance persists throughout the application's lifetime, so it is always valid and can be accessed from any class within the application.

One minor benefit to the Static Class solution is in terms of semantics, as this is clearly the intended way to manage global behavior across an application.

Conclusion
While many of these solutions could work in practice, we decided to use the `Application` subclass method, as it supports arbitrarily complex data, lets us clear and structure the data hierarchically, makes it easy for us to add new data fields and also follows proper and expected code conventions within the Android development environment.

[19] `https://kotlinlang.org/docs/object-declarations.html#companion-objects`
[20] `https://developer.android.com/codelabs/basic-android-kotlin-training-persisting-data-room#0`

8.3.3 Persistent Data Serialization

In this prototype-phase of the application, we don't need to store much persistent data. In reality, we only really need to store the baseline score, which is simply a `float`. Nevertheless, we will go over some of the common methods of storing persistent data on Android and explain which fits our needs the best.

Room
Room runs a local SQLite server on the device which can be accessed through an abstraction layer. It's pushed quite heavily by Android, as they seemingly want you to use this for persistent data storage. Since our application, or at least the prototype application, only needs to store a single float, setting up an entire SQLite server is overkill.

Future versions of this application might however benefit from doing this, especially since a full release of the app would need to be able to handle multiple profiles for different patients' screening data. An SQL server would fit this purpose very nicely, although you would probably also want to connect it to a cloud-based server, which Room is not designed to do.

Flow
Flow is a framework built for Android applications that use Kotlin which lets you pipe data through your application while producing it asynchronously. This shares many of the same problems as Room in terms of being overkill and seems to focus more on easing communication with network layers without blocking the main thread, as those are inherently asynchronous.

This seems like it might be a good fit for a final version of the application, as it can handle communicating with a cloud-based SQL database.

DataStore
`DataStore` lets you store data as key-value pairs or as typed objects using both synchronous or asynchronous methods. The documentation[21] recommends you do not use synchronous calls and instead rely on the asynchronous methods powered by Flow.

This does seem like a relatively comprehensive solution for storing data locally if you want to avoid using Room for whatever reason, but it's still a bit cumbersome to get up and running since our goal is storing a single floating point number, especially the asynchronous methods which require the use of Flow as well.

SharedPreferences
Shared Preferences are much more simple than any of the other methods explored in this section. It provides a single function call for getting a reference to an object you can write and read key-value pairs to, which are then subsequently stored persistently. The documentation[22] does warn developers of using it instead DataStore, but we will disregard that for the reasons outlined in this section.

Conclusion
We have decided to use Shared Preferences as we were looking for the simplest possible API to store a single value persistently, which is what Shared Preferences provides.

8.4 Gaze Data Recording

Implementation of gaze tracking is critical to the app: Not only is it one of the most complex components, but it is also a vital part of the app's unique value proposition. Because of the potential complexity of the technology, implementation is approached in as small steps as possible, starting with the simplest of tasks and gradually building on more and more complexity until the desired level of functionality is achieved.

Due to the success of the technical experiments detailed in Section 6.5, we move forward with SeeSo as our solution to gaze tracking.

Fortunately, implementation of SeeSo is straightforward, as its quick start guide[23] is written with the

[21] `https://developer.android.com/topic/libraries/architecture/datastore`
[22] `https://developer.android.com/training/data-storage/shared-preferences`
[23] `https://docs.seeso.io/nonversioning/quick-start/android-quick-start/`

same idea of gradual complexity in mind. Initial implementation is done on a blank activity.
First step is to simply have this activity import the requisite library and log the version number, proving the import is successful.
For the next step, the SeeSo `GazeTracker` class is initialized, which is once again verified via console log. Next is running the gaze tracker and logging recorded coordinates. By setting the `translationX` and `translationY` properties of an image view, this quickly turns into an app that can move a small image around on screen according to where the user is looking, as shown in Figure 30.

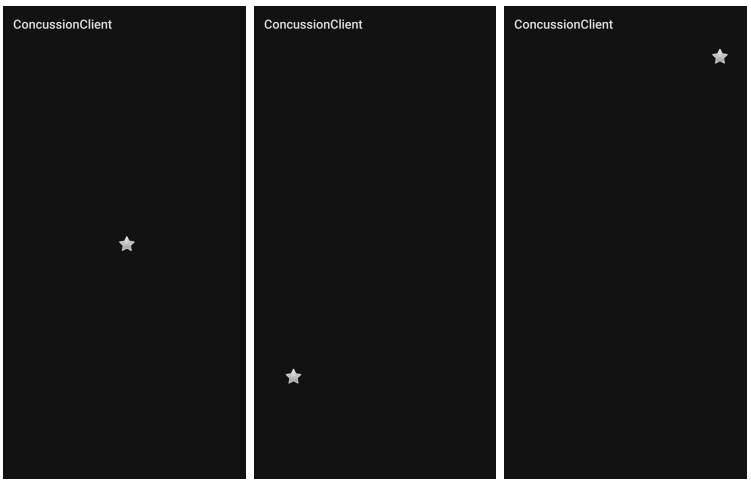


Fig. 30: Screenshots of initial SeeSo implementation showing a star icon that moves to where the user is gazing.

With the basics of gaze tracking working, development moves on to user control features and playback. See section 8.5 for details on playback implementation. The simplest implementation of user control here is a button that toggles between "start recording" and "stop recording", which implements SeeSo's `startTracking` and `stopTracking` methods. When the gaze tracker processes information, it returns a `GazeInfo` object which among other things contains the UNIX timestamp in milliseconds and a pair of x- and y-coordinates in pixels. Observed gaze data is stored in a map that matches coordinates to timestamps, and the timestamps are offset so that the first entry in the map has timestamp 0.

While the code was since moved around between other classes, the functionality of gaze recording remained the same for the remainder of app development. In its final iteration, the gaze recorder is placed in the application class, described in Section 8.3.2. This makes it accessible to all activities, so the test activity can perform recordings and the review flashcard activity can read those recordings.

Since it can take a couple of seconds to initialize the gaze tracker, this also has the added benefit of allowing us to initialize the gaze tracker in the main activity when the application is first launched. This way we avoid interrupting the user during the test.

8.5 Gaze Data Playback

With recording of the gaze data handled, we move on to playback of gaze data. To display the coordinates, we use an image view as a gaze indicator, moving its center to wherever the coordinates indicate. What's really of interest, however, is how the position of the gaze changes over time. To properly display the recorded gaze data, a form of playback is required.

The implementation of this kind of playback presents some problems: The application should run code that continuously updates the user interface by moving the gaze indicator around, but the user should remain able to interact with other parts of the app while playback occurs. This means playback should be handled by another thread that doesn't block user input. However, we also want to allow the user to pause or scroll through the playback, as is typical for a music or video player. The playback thread should be able to be paused, resumed, and manipulated by calls from the main user thread.

Similar to how gaze recording was implemented, the initial implementation is as primitive as possible. It

creates the desired playback effect but doesn't allow user input. This is done with the `Looper`[24] class, and a corresponding `Handler`[25].

In this initial implementation, starting playback positions the gaze indicator, then launches a delayed task to call itself recursively with a delay matching the difference between the current and next timestamps. This method produces the desired playback effect, but the repeated, recursive callbacks is not very elegant and cannot be easily interrupted.

The next implementation uses a coroutine job to handle playback. The `Job`[26] class defines a piece of code that runs in the background and can be canceled. A job is tied to the life cycle of another object, and in this first implementation of the class, the job is launched using the activity's life cycle. This means the job lasts as long as the activity, and is destroyed when the activity is destroyed.

Where the looper-handler implementation launched hundreds of serial callbacks, this new implementation launches a single thread that loops until it reaches the end of recorded gaze data or is canceled. This implementation now allows the user to manipulate playback, as illustrated by the presence of the seek bar on Figure 31. This seek bar is updated by the running job, and user input on the seek bar cancels the current job and relaunches it from the indicated progress.

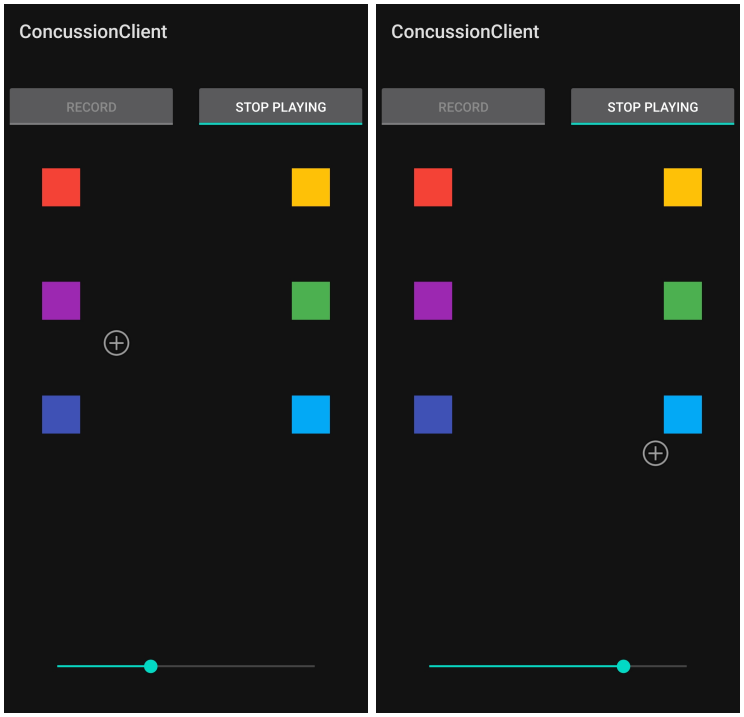


Fig. 31: Screenshots of playback implementation using `Job` and featuring a seek bar to provide user control of playback. The circle with a crosshair indicates gaze position - the colored squares simply serve as markers for testing gaze tracking accuracy.

At this point, gaze data playback is integrated into the wireframe implementation described in Section 8.3, with the goal of further refining it once audio playback is introduced (see Section 8.7).

8.6 View Model Implementation

With the introduction of gaze data playback, the code of the activity for reviewing flashcard performance starts getting somewhat cluttered and difficult to maintain. When playback is started, code is run to transform the "play" button into a "pause" button, and the gaze indicator position needs to be continually updated, as well as the progress shown on the seek bar. Similar updates are required when

[24] `https://developer.android.com/reference/android/os/Looper`
[25] `https://developer.android.com/reference/android/os/Handler`
[26] `https://kotlinlang.org/api/kotlinx.coroutines/kotlinx-coroutines-core/kotlinx.coroutines/-job/`

pausing playback, scrolling on the seek bar, or when playback finishes.

An elegant solution to this problem is Android's view models[27]. The view model is a class meant to handle business logic and preserve UI state. This means we can take many of the functions not directly related to the UI - in this case most of the gaze playback functionality - and move them into the view model. From there, we expose a simple set of interfacing functions for the activity to use when the user interacts with various input elements.

The view model contains a simple state object that contains necessary information on the current state of the UI and can be used to determine the appearance of mutable elements of the UI. The bottom of Figure 32 lists these properties.

The UI state object determines much of the activity's appearance using only a few primitive data types. The view model is hooked into the activity by giving the activity a special method that tells it how to update the view in response to the UI state object. An example of how the UI state object shapes an activity's appearance is shown in Figure 32.

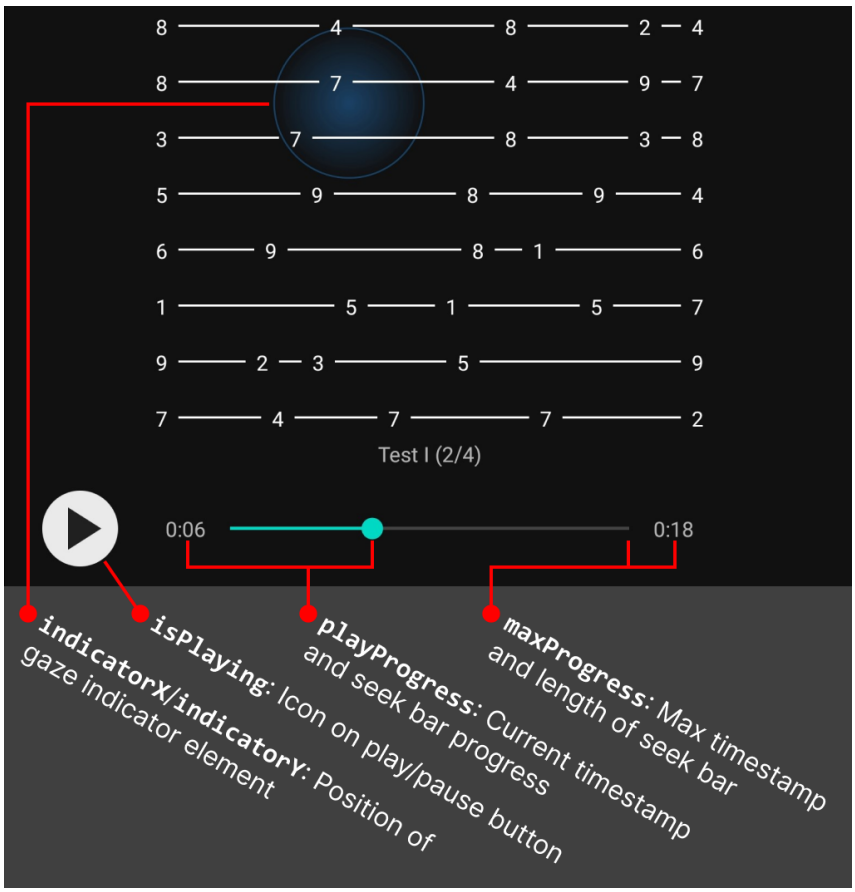


Fig. 32: Partial screenshot of review flashcard activity, edited to show how elements of the UI state object determine dynamic UI elements.

8.7 Combined Audio and Gaze Tracker Playback

While gaze data is an important part of the project's unique value proposition, recording the user's reading aloud of the flash card is also a vital usability feature for reviewing performance of a flashcard. It also elevates the value of gaze tracking data by showing what the user was reading at any given time.

Android provides an easy-to-use framework for recording and playing audio in the form of the `MediaRecorder`[28]

[27] `https://developer.android.com/topic/libraries/architecture/viewmodel`
[28] `https://developer.android.com/reference/android/media/MediaRecorder`

and `MediaPlayer`[29] classes. To store the audio, a temporary file is created, with a file path retrievable through the application class. Similar to how the application class exposes the gaze tracker to all activities, this exposes the audio file to all activities to read and write.

8.7.1 Recording

One of the biggest problems in tackling combined recording of gaze data and audio is that neither recorder class can begin recording instantaneously. Each class has some amount of overhead that causes the start and end of recording to take a non-zero amount of time.

This means that the two recordings of audio and gaze data are going to be of different lengths, which can cause problems in synchronizing the playback. In order to determine the severity of this issue, some impromptu experiments are carried out to measure the lengths of recordings under different configurations.

A decision that is made early in the process is choosing to fully encapsulate one recording within the other. By beginning the outer recording before the inner, and ending the outer recording after the inner, we ensure that the outer recording spans both sets of data, and can use it to determine the full length of playback. The question left to answer is whether the audio or gaze tracking recording is best suited for the outer recording.

To that end, the app is run in both configurations a couple times each, while measuring the duration of each recording. `AudioPlayer` has a `duration` attribute for this purpose, and for gaze data, we can simply extract the last timestamp, given that all timestamps are offset so that the initial timestamp is 0. However, a strange observation is made: Starting audio recording within gaze data recording leads to the smallest difference in recording durations, but the audio recording sometimes has a longer duration, despite beginning later and ending sooner.

The experiment revealed a flaw in how playback for the gaze data was treated: Whenever the SeeSo framework is unable to produce gaze data, such as when the user's eyes aren't visible on the camera, no data is recorded.
If this occurs in the middle of a recording, the only observable effect is that playback lingers on a given coordinate longer than usual, which isn't a big problem, albeit slightly misleading as the user isn't necessarily looking at that spot at the time.
If the lack of recorded data happens at the end of a recording, the duration will appear shorter if measured using the last timestamp. This explains how gaze recording sometimes has a shorter duration despite being the outer recording.
Similarly, if the start of the recording lacks data, the duration will not only be shorter, but the timestamps of the entire recording will be inaccurate, as the first timestamp is used as the offset.

By instead marking the timestamp of when the gaze recording starts and stops, it is confirmed that the optimal configuration is an outer audio recording and inner gaze recording. This is confirmed by running the application five times for each configuration and recording the durations. This results in an average duration disparity of 150 ms, whereas the other configuration has a 241 ms duration difference. To fix the offset problem caused by lack of initial gaze recording data, the gaze recorder now offsets data according to a timestamp taken once recording starts.

8.7.2 Playback

Implementing audio playback is more straightforward than gaze data playback. Since the measured difference in duration between the two recordings is so small, it's unnecessary to synchronize playback in any major way. At worst, the two playbacks will be offset from each other by ca. 150 ms.

We change the logic of the playback coroutine to better align with the continuous playback of the audio file, and to better reflect our findings from the experiment carried out in the previous section. Rather than calculate the time between each timestamp in gaze data and delaying the thread for that duration, the coroutine instead delays by 33 ms, corresponding to approximately 30 updates per seconds.

Previously, non-empty gaze data would always have data for timestamp 0, allowing the gaze player to always be able to look backwards through the data map if there is no data for a given timestamp. Since

[29] `https://developer.android.com/reference/android/media/MediaPlayer`

data is no longer offset according to a data point, but a timestamp taken as recording begins, the player will need different logic for selecting data points. The player is changed to only look back 33 ms, and show nothing if no coordinates are found. This also fixes playback "freezing" when there is a hole in the data, as playback will now hide the gaze indicator in the case of missing data.

8.8 Speech-to-Text

With both gaze and audio recording working, we move onto integrating speech-to-text into the application. This amounts to setting up a recording using the `SpeechRecognizer`[30] class described in Section 6.6. This type of recording produces a string of numbers instead of something to play back, so there are no concerns regarding synchronizing a third type of playback. The first step of this integration is ensuring that speech-to-text can be made to work concurrently with the other types of recordings that occur during tests using the flashcards.

Unfortunately, this step fails. Both the audio recording and the speech-to-text recording use the device's microphone, causing one service to be blocked by the other. The result is the `SpeechRecognizer` returning nothing.

We make several attempts at solving this problem. Unfortunately, `SpeechRecognizer` cannot transcribe recorded audio, so it cannot be fed the audio obtained from the `MediaRecorder`. Additionally, the audio used by `SpeechRecognizer` isn't accessible in a way that allows for later playback.

As it turns out, Android's built-in speech-to-text isn't a viable option if we also want to present the user with audio playback. Presumably, we have a choice between one or the other, but we didn't test the combination of gaze tracking and speech-to-text with no audio, as we already had a clear preference for audio recording for several reasons. Firstly, playback of audio greatly elevates the value of gaze playback, our unique value proposition. Second, implementation of audio playback is already finished and implemented. Third, implementation of speech-to-text would require additional behind-the-scenes features, such as logic for matching up uneven sequences of numbers (such as what's done to align the values in Table 3).

Instead, an external speech-to-text solution is required, one that can take an existing audio recording and transcribe it. Unfortunately, it isn't possible to implement such a solution in time, but the experiment carried out in Section 6.6 confirms that any speech-to-text software with the same accuracy as Android's may be of value to the application, depending on user feedback.

[30] `https://developer.android.com/reference/android/speech/SpeechRecognizer`

8.9 Final Application

Figure 33 provides a simplified overview of the overall user flow of the final application, as well as what additional resources are used and when. For a more detailed overview of the classes, see the class diagram in Appendix 12.8.

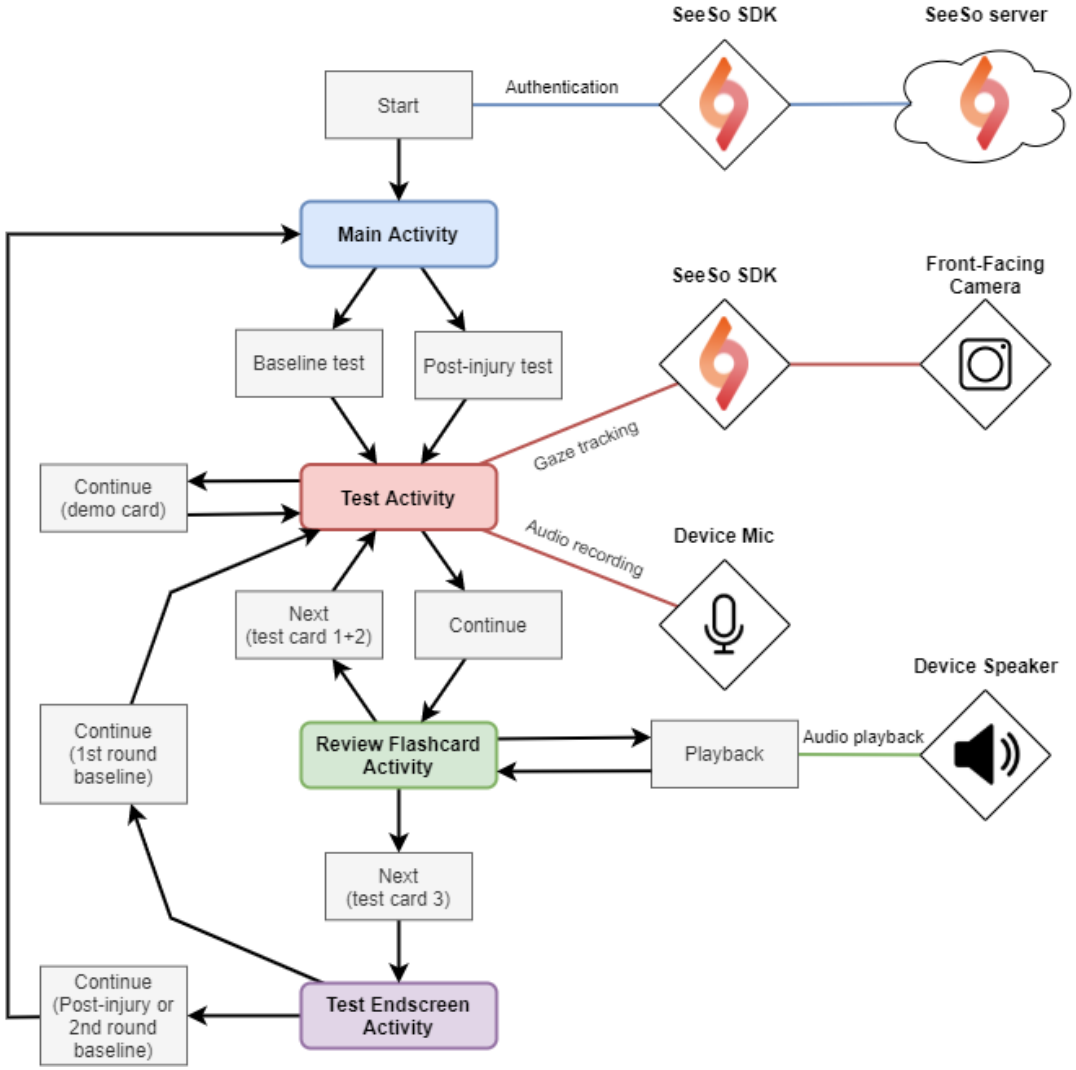


Fig. 33: Flow diagram of the final application, including usage of resources such as hardware or internet connection.

The application is structured around 4 activities, each with their own layout.

8.9.1 Main Activity

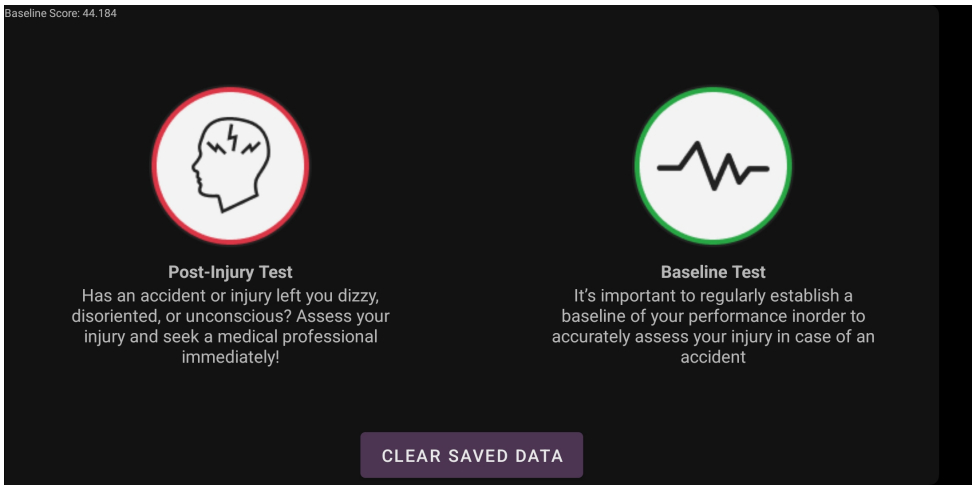


Fig. 34: Screenshot of the main activity.

The main activity is the launcher activity, i.e. the first thing that a user will see when opening the app. On the front-end, this activity has 4 overall components (See Figure 34):
The center holds the primary feature of the main activity: Buttons that begin the two types of tests. These are implemented using fragments, and both lead to the test activity, described in the next section.
In the top-left corner, we display the currently stored baseline score. This was implemented mostly as a debugging feature, allowing us to see what data the system was working with, and since we determined implementing a proper history of baseline data was out of scope for this project.
The "Clear Saved Data" button at the bottom of the screen is similarly intended to be a debugging feature, as we often need to test how the application performs if no baseline score is recorded, for instance, to ensure the Post-Injury Test button becomes inactive.

Being the launcher activity, the main activity also handles two important tasks when the app is first launched: First, the activity requests permission to use the camera and use the microphone for recording audio. Second, assuming camera permission was granted, the activity initializes the SeeSo gaze recorder. This initialization requires an internet connection, as a license key must be sent to and verified by SeeSo's servers in order to use their product. Once initialized, the gaze recorder is ready for use by the test activity. The initialization can take a couple of seconds, so we start it here in the main activity so that a slow initialization doesn't disrupt the test.

8.9.2 Test Activity

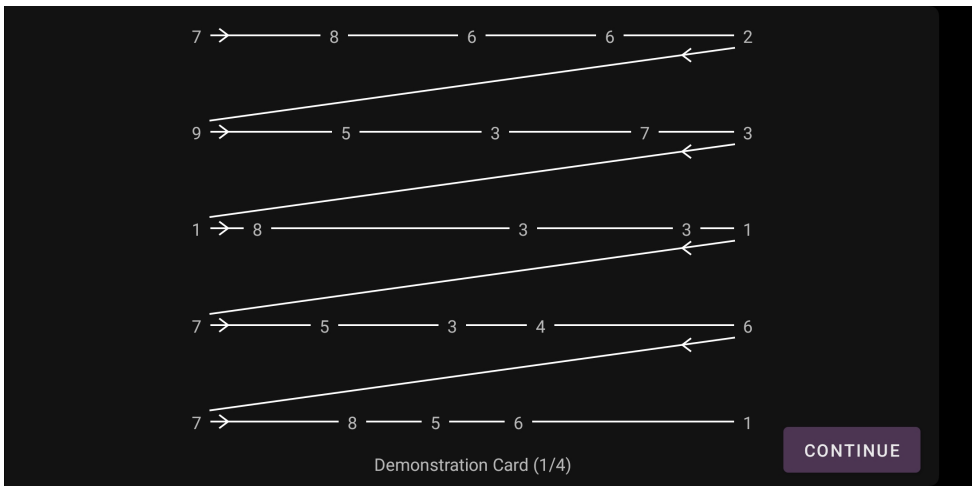


Fig. 35: Screenshot of the test activity, showing the demonstration flashcard.

Whether the user chooses baseline or post-injury test, they are brought to the test activity, which Figure 35 shows a screenshot of.

The front-end of this activity consists primarily of the flashcard fragment, whose appearance varies based on how far the user has progressed in the test (see Figure 36 for the other three configurations).

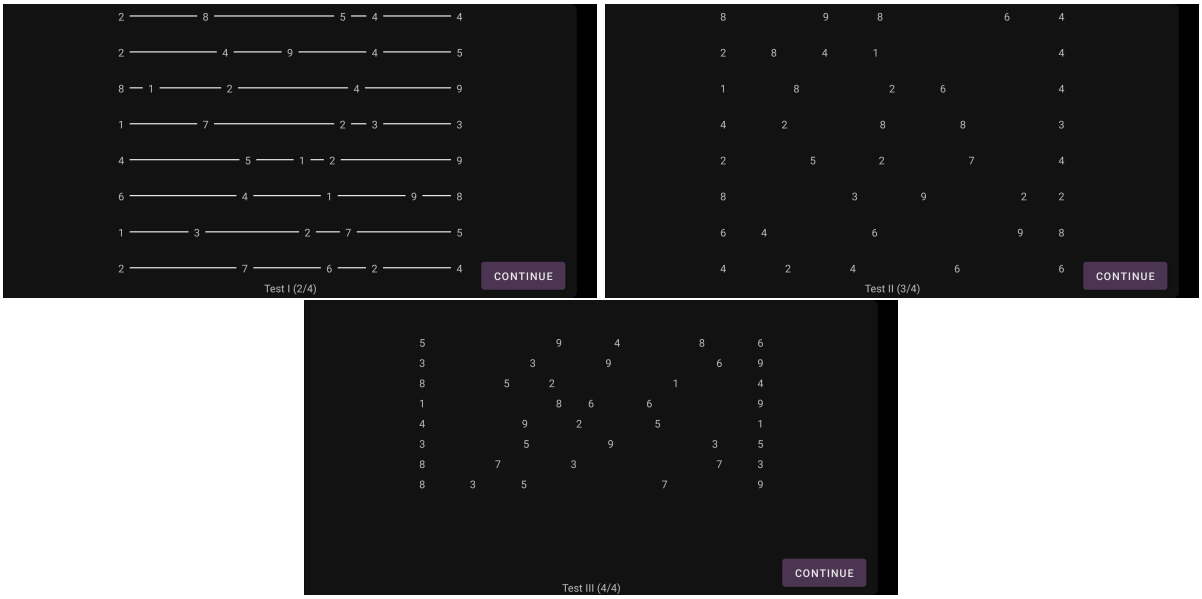


Fig. 36: Screenshots of the test activity, showing the three test flashcards.

The numbers on the flashcard, as well as the amount of space between each number, is randomly generated when the fragment is created. The index of the flashcard is then used to determine layout features such as the presence of diagonal and horizontal lines and how close the rows are positioned. At the bottom of the flashcard is a display showing which of the four flashcards the user is currently on, giving them an idea as to their progress.

For the demonstration and first test flashcard, the user is shown a set of instructions preparing them for the upcoming flashcard. These instructions are displayed in Figure 37. Before the test begins, the instructions appear as an overlay that can be dismissed by tapping anywhere on the screen. The instructions serve to teach the user how to perform the King-Devick test, how to hold the phone, and how to advance once the flashcard is complete.

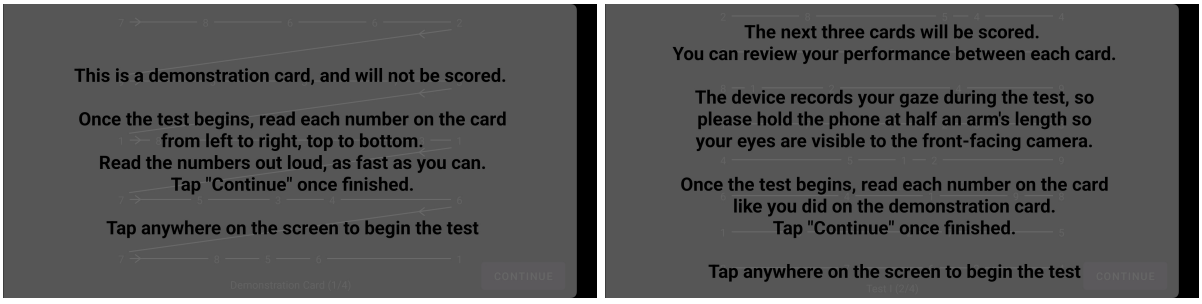


Fig. 37: Screenshots of the test activity, the two different instruction popups.

The final component of the test activity layout is the continue button, which takes the user to the review flashcard activity.

When displaying any of the three test cards (not the demonstration card), the test activity records the user as they read through the flashcard. Two recordings are made: A gaze recording using the SeeSo SDK which in turn uses the front-facing camera, and an audio recording that uses the microphone. Both

of these recordings start when the activity is first rendered (for the first test card, recording instead starts when the instruction popup is dismissed), and stop once the user taps the continue button.

Gaza data is stored in a map accessible to the review flashcard activity, and the audio recording is stored in a temporary .mp3 file, also accessible to the review flashcard activity.

8.9.3 Review Flashcard Activity

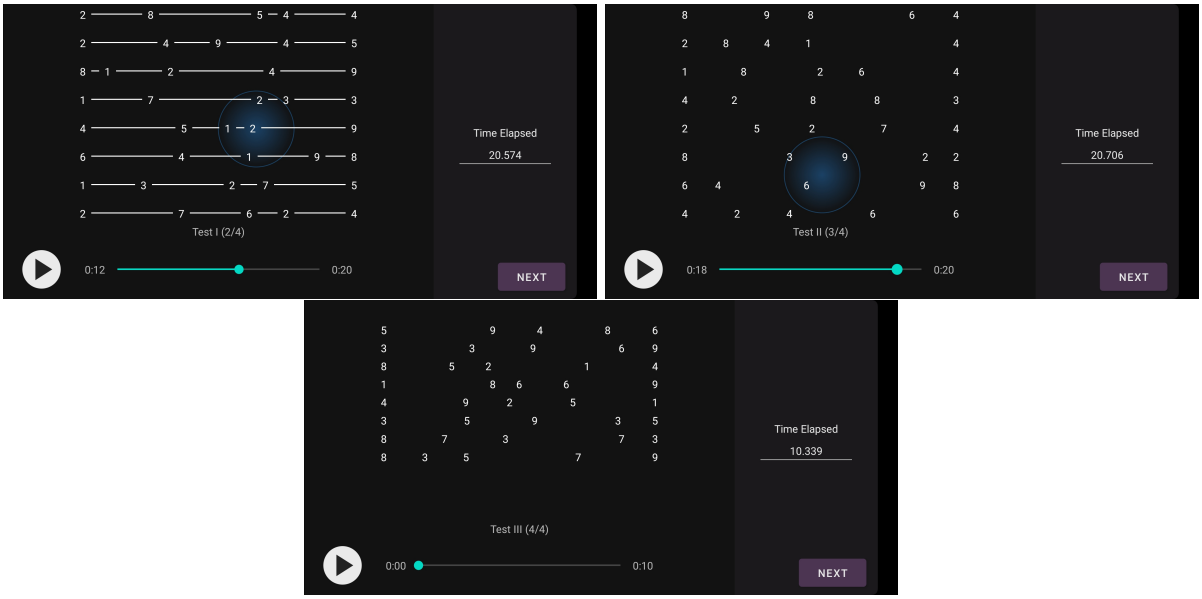


Fig. 38: Screenshots of the test review activity, showing the three test flashcards and the playback system in various states.

The review flashcard activity is arguably the most complex of the application, featuring several interconnected systems that must work together in order to form a cohesive whole.

The flashcard display uses the same fragment as the test activity, which utilizes a seed in order for the randomly generated appearance and data of the flashcard to be identical to the test activity the user navigated from.
On the right-hand side of the screen, the recorded time elapsed during the test can be seen and edited. The edit feature was implemented to let any medical professional supervising the test author the test's outcome. Not implementing Speech-to-Text caused a cascading effect, making us re-evaluate whether implementing error handling was a good idea or not, as the assessor would now have to manually enter all of these responses individually for every digit read aloud. This ultimately resulted in the entire scope of the review activity being significantly reduced, as the only data point the assessor could author was the time it took for the patient to complete the test. The button at the bottom of the side panel is used to advance to the test activity for the next flashcard. If reviewing the fourth and final flashcard, this button instead takes the user to the end screen activity.
The bottom of the screen contains several elements pertaining to the playback system. When the user presses play, they will be able to see and hear recorded gaze and audio data, which lets medical professionals review the test in order to more closely examine the subject's performance. The progress bar is also interactable, letting the user select a specific time during the test to start the playback. When the user interacts with the progress bar, the gaze indicator is dynamically updated to show where the user was looking at the given time.

8.9.4 End Screen Activity

The end screen activity is split into two distinct modes, depending on if the user is currently conducting a baseline test or has just completed a post-injury screening. Each mode is implemented using a fragment in order to simplify the distinction, reduce code clutter, and avoid one mode affecting the other.

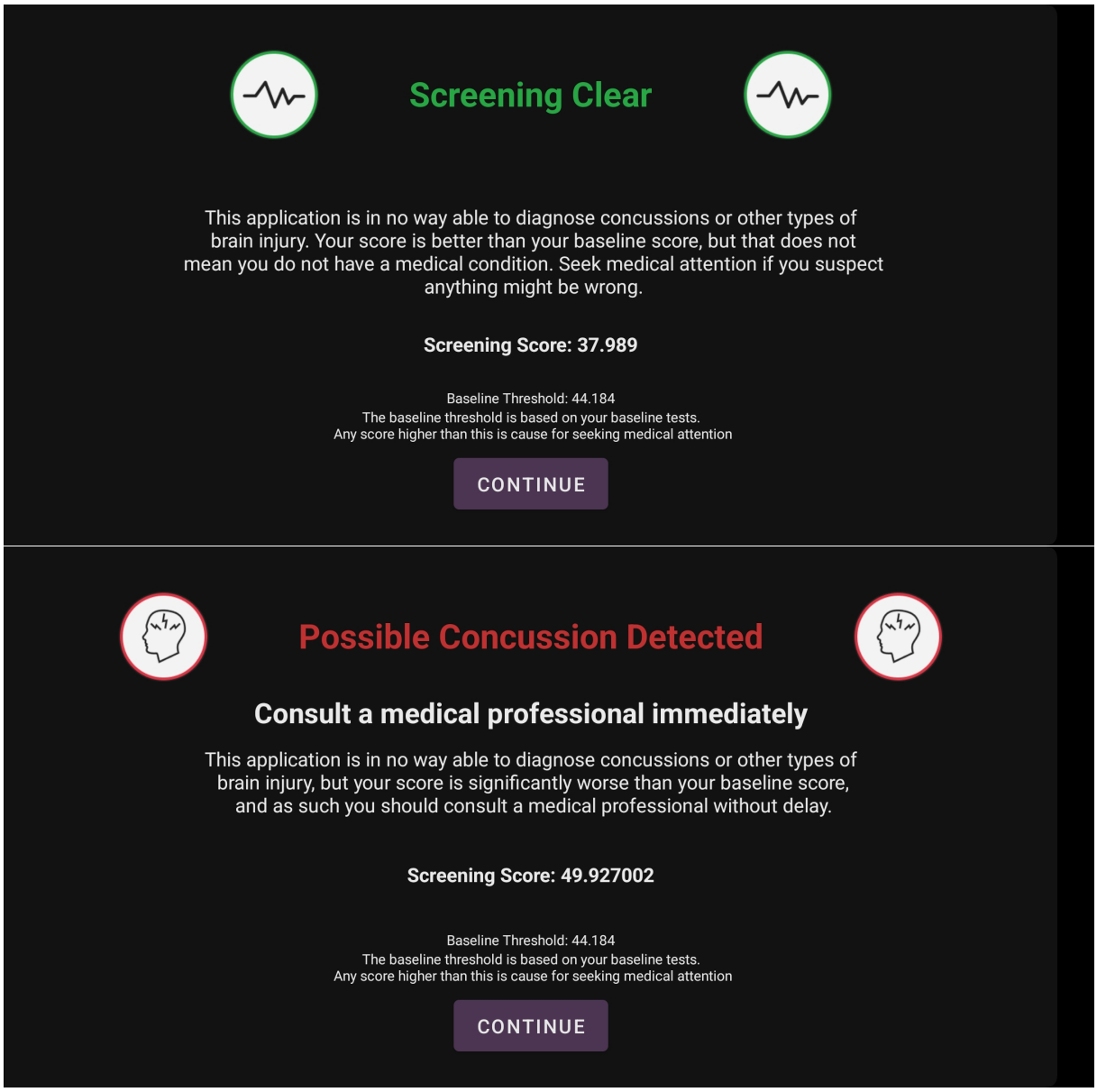


Fig. 39: Screenshots of the end screen after a post-injury test has been completed, both in the case where the subject passed and where the subject failed

The post-injury version of the end screen (Figure 39) contains several different text elements, all of varying sizes depending on how important the information they contain is for the test subject. In the case where the screening failed and our application detected they may have sustained a TBI, large text urges the test subject to seek professional medical attention. Less important text, such as the text at the bottom which explains how baselines are compared to screening scores, is rendered in a smaller size, so as to not distract the user from what is most important for them to read. Color coding is also utilized in order to convey information to the reader, where a safe green is used when the subject passes the screening, and a dangerous red is used when the subject fails.
As this fragment contains a lot of text, we opted to use a string resource[31] in order to store all the different variations. This helps keep the code clean, as each string variation can be retrieved using a simple ID, and all the text can be edited from the same XML file.
Since the entire view changes depending on whether the subject may have sustained a concussion or not, we opted to implement the fragment by creating a lot of getter functions for the various aspects (`getIcon` for the icon resources in the top of the screen, `getHeaderColor` for the text color using the

[31] `https://developer.android.com/guide/topics/resources/string-resource`

top of the screen, etc.) which all queried a simple Kotlin property[32] that compared the baseline score to the screening score. This further helped keep the code clean, as the view-generating code did not have to contain any logic for handling the differences between the two cases.

The Continue button at the bottom of the screen takes the user back to the main activity.

Fig. 40: Screenshots showing the end screen when completing a baseline test. Since the baseline tests have two rounds of King-Devick, two screens are shown

The baseline version of the end screen (Figure 40) has a simpler layout. Like the main screen, this fragment contains some debugging information in the top-left corner, which shows the cached data in memory of both Test Instances of King-Devick. This was useful during development to verify that the correct baseline score was chosen among the two.
Unlike the post-injury fragment, this fragment does not contain a lot of text variations, and as such, we were able to get away with simply hardcoding strings into the fragment, where the missing "Final Baseline" text in the first round of the end screen is accomplished by simply rendering an empty string.

This version has two variations of the button at the bottom of the screen. After the first round of testing, Continue takes the user to the test activity of the first flashcard to begin the second round. After completing the second round, Finish takes the user back to the main activity.

8.10 Conclusion

Besides the issues caused by not being able to implement Speech-to-Text, the implementation continued as we had hoped, and we feel that we have managed to arrive at a sensible prototype that is able to complete the initial goals as described in this section's introduction. We will now conduct experiments to determine whether the application fulfills the intended goals of the project as a whole or not.

[32] `https://kotlinlang.org/docs/properties.html#declaring-properties`

9 Final Testing

We have created a functional prototype that is capable of gathering data during a King-Devick test and making recommendations regarding whether or not the patient should seek medical attention based on this data. While we have not implemented any automation tests during development in accordance with best practices due to time constraints, we have manually tested intended user flows and verified that the application is functional on a technical level. In order to answer our final problem statement, we now wish to validate whether the application possesses the screening capabilities of a King-Devick test, and whether it's simple enough that any user can perform the assessment using the app. We specifically wish to evaluate the following attributes of our application:

1. Can users operate the app properly when they have only been given instructions by the application?
2. Do users understand whether they should seek medical attention after screening?
3. Is our application sensitive to suboptimal brain function in the same way that the King-Devick test is?

These goals will be evaluated by conducting two experiments. The first experiment will validate our application's sensitivity to ocular-motor performance deterioration by comparing baseline test scores conducted while rested and awake to screening test scores conducted while sleep deprived. The second experiment will validate the intuitiveness of our application by asking test subjects who have no prior experience with using the application to achieve certain goals. After this, they are asked to fill out a questionnaire that will help us determine whether they found the experience to be intuitive or not.

9.1 Fatigue Test

9.1.1 Goal

We wish to validate that our application is able to correctly attain similar test results as a normal King-Devick screening test. Issues posed by ethical dilemmas regarding subjecting real patients with mTBIs to our tests leave us looking for alternative ways of performing this validation.

9.1.2 Setup

We base this test on a paper that was able to draw a correlation between sleep deprivation and deterioration of King-Devick test scores.[32] The test subjects were all neurology residents and staff from the University of Pennsylvania Health System and were grouped depending on whether they were on-call or not, as this affected their sleep significantly prior to taking the screening test. The paper found that residents and staff not on-call and after ≥ 6 hours of sleep had a median improvement of 3.8 seconds compared to their baseline performance, which is consistent with the learning effects of repeatedly taking King-Devick tests. The on-call group had a median deterioration of about 0.23 seconds, completely negating the learning effect. Both groups were typically screened 24-72 hours after completing their baseline.

While the test score deterioration in sleep-deprived individuals is not as significant as with patients suffering from mTBIs, we find this to be a suitable simulation to validate the efficacy of our application.

Unlike the subjects who were tested in the paper, both of us have taken a myriad of King-Devick tests already, and as such the learning effect might not be as pronounced on us as on the test subjects. To correct for this unknown factor, we will start by executing the baseline test for three days while we are fully rested, in order to determine whether our score improves or not. After this three-day period, we will repeat the same procedure, but executing the screening in a sleep-deprived state, which we will induce by waking ourselves after < 6 hours of sleep.

To summarize, the following test will be completed for the first 3 days while fully rested:

1. If the device already has a saved baseline score, clear it from the main menu.
2. Complete the full baseline test.
3. Write down the final baseline score.

For the last 3 days where we wake up after getting < 6 hours of sleep:

1. Sleep for < 6 hours
2. Wake up from an alarm
3. Complete the screening test.
4. Write down the final screening score.
5. At this point, we are permitted to go back to sleep.

9.1.3 Results

The experiment resulted in the following raw data points:

	Daniel	**Noah**
Rested 1	47.567	40.813
Rested 2	52.834	40.430
Rested 3	44.252	40.045
Tired 1	49.348	47.663
Tired 2	48.818	59.571
Tired 3	54.786	52.155

Table 5: Overview showing all the raw test results from our fatigue tests

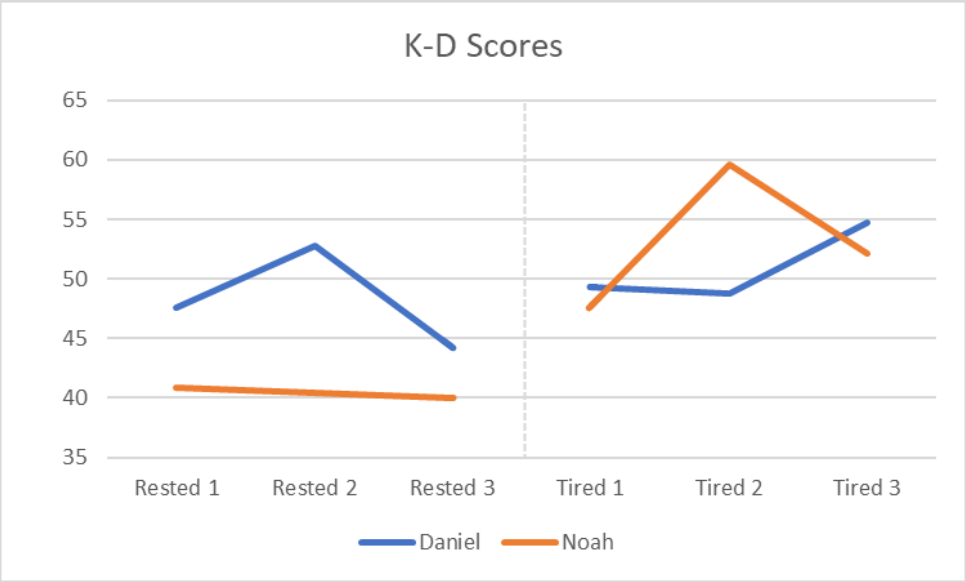


Fig. 41: Graphical overview of our test results

We can compile the raw data into something that is easier to interpret, namely by defining an average rested score, the slope of a linear trend of the rested scores, an average tired score, and a percentage

increase of the two averages. The rested trendline, which describes our daily change in score where negative scores are an improvement, is calculated in Excel using the `LINEST` function[33] and helps us determine whether we are affected by the learning effect, as mentioned in the sleep deprivation paper[32]. Only the slope is shown in this table, as we are not interested in the y-intercept. The average percentage score increase helps us determine if the application is able to detect an increase in score while we are tired and is calculated using the following formula:

$$\text{Percentage Increase} = \frac{\text{Tired Average Score} - \text{Rested Average Score}}{|\text{Rested Average Score}|} \tag{3}$$

	Average Rested Score	Rested Trendline Slope (LLS)	Average Tired Score	Average Percentage Score Increase
Daniel	48.218	-1.66	50.984	5.74%
Noah	40.429	-0.38	53.130	31.41%

Table 6: Overview of the interpreted data

9.1.4 Conclusion

It is very clear to us that we need more data points to properly analyze the performance of our application. For instance, it is very difficult to tell whether Daniel's second rested score is an outlier or not, rendering any assessment of the entire dataset ambiguous. With this in mind, we will nevertheless work with the data we have collected and try to interpret its meaning to the best of our ability.

Looking at Figure 41, it is evident that Daniel's rested baseline is higher than Noah's. This is not uncommon, partly because every person will have a different baseline, but also because Noah has spent more time running K-D tests while implementing the gaze tracking system, and will therefore most likely have a lower baseline score than Daniel, even if we initially started out with roughly the same baseline. This is further indicated by the Rested Trendline Slope (LLS), as Daniel's is markedly lower than Noah's, which could indicate that Noah is reaching the limit of how much practicing can affect his score, whereas Daniel can still significantly improve his score. Our final conclusion on the matter of the learning effect is therefore that both Noah and Daniel are still affected by it, although Noah is significantly less affected than Daniel is.

The sleep deprivation paper[32] measures its results relative to the learning effect. Since the rested group saw an improvement in-line with the learning effect, and the sleep-deprived on-call group saw a deterioration in their score, rather than the expected improvement, the paper concluded that K-D screening is sensitive to sleep deprivation. Similarly, our experiment shows that our application is sensitive to sleep deprivation and by extension when referencing the paper, also sensitive to the effects of mTBIs.

9.2 Usability Test

9.2.1 Goal

We wish to verify whether we have successfully solved the second half of our final problem statement: "(...) and can such a test be made simple enough that a user with little to no experience with concussions can perform the assessment?" We hope the instructions provided by the application itself are sufficient for new users to feel the application is intuitive and easy to use. The goal of this experiment is, therefore, to hand over the application to a test subject who has been given no prior information besides a brief introduction of the app's intended use case, and the goals the test person has to achieve using the application.

While we will not have time to iterate upon this experiment, it will provide invaluable data which we can use to validate the usability of our product as-is and to reflect upon which changes we could make, given we had more time to do so.

[33] `https://support.microsoft.com/en-us/office/linest-function-84d7d0d9-6e50-4101-977a-fa7abf772b6d`

9.2.2 Setup

The experiment starts by giving the subject a brief introduction that explains the application is intended to help screen people for concussions, and how a baseline has to be established while the subject is healthy before they can be screened post-injury. After this, we hand over a device with the application launched. The application should not have any data from previous experiments saved, as its state should be as if it had just been installed. Furthermore, the volume on the phone should be turned on so they are able to listen to themselves read the numbers out loud. We ask the subject to achieve the following goals:

1. Complete a baseline test.
2. Complete a post-injury screening test.
3. Assess whether the application suggests you seek medical attention or not.

At this point, the test subject should start attempting to achieve these goals, while the test giver simply observes without aiding or guiding in any way, even when the test subject asks questions.

After the subject has either completed all goals or given up on doing so, they will be asked to fill out a questionnaire. The full questionnaire can be found in Appendix 12.5, but in essence, most of the questions are quantitative in nature and ask the subject whether they understood what they were supposed to do and whether they understood what the application was attempting to communicate to them. There are also some optional qualitative questions, which the subject can choose to answer if they wish to elaborate upon their experience.

9.2.3 Results

The full summary of all responses can be found in Appendix 12.6, but we will present a brief overview of the key takeaways from the questionnaire here, along with our own impressions gained from observing people using the application and talking to them after conducting the test and answering the questionnaire.

All test subjects report being able to successfully complete the given instructions, although a couple of people were confused while completing the baseline test. One of them got confused by the review activity and thought it was another flashcard they had to complete. They did not realize this was not the case until they saw the playback button. Another person did not understand whether they had to do it in Danish or English. They completed their first Test Instance in English but then switched to Danish during the second Test Instance.
25% did not understand why the application flagged them as potentially being concussed, whereas the remaining 75% understood their screening results. This question was followed by a qualitative question intended to test whether subjects who claimed to understand their screening results actually did so. In reality, most of the answers we received were not as elaborate as we had hoped, and cannot be used to determine whether the subjects actually understood their results one way or the other. While filling out the survey, some subjects did verbally express their understanding, even if they didn't fully convey this through the questionnaire.
25% thought they held the device correctly during tests, but had an unspecified problem. We, unfortunately, did not include a qualitative question after this, so we do not know what exactly the reported issues were, although we assume it is probably related to the gaze tracking playback not being entirely accurate.
All but one subject reported that the instructions were helpful. One subject recommended reducing the amount of text in the card and instead relying more on icons and inspiring colors. They reported that they accidentally, out of habit, dismissed the instruction page the first time, and recommends perhaps including a quiz to ensure the subject actually understands the instructions. Another recommended we specify which language the test should be conducted in, or explicitly state that doing it in any language is fine.
All subjects but two were able to use the playback feature but reported that the gaze indicator was not entirely accurate. The two subjects who were not able to use the playback feature reported this was due to not noticing it existed. No one reported any issues when trying to listen to the audio recording of their test.

We noticed that one of the subjects was squinting a lot while doing the test, which resulted in either wildly inaccurate gaze data, or no gaze data at all. We also noticed that gaze tracking performed poorly for people with glasses and did not work at all in the dark. Most of the subjects did not understand that they did not have to read the demonstration card out loud every time they started a test instance. Some of the subjects were also a bit frustrated by the instruction cards appearing several times, as they had understood how to complete the test after the first Test Instance.

9.2.4 Conclusion

Test subjects were able to understand the goals we asked them to complete and execute them correctly without any further instructions or interference by us. Some of the hiccups experienced could be improved by making the following changes to the application:

- Review flashcards after they have all been completed. This can however cause a situation where the subject fails the first flashcard but is instructed to continue with the remaining two, even though they will have to redo the Test Instance no matter what. Further testing is required before we can make any decisions regarding this.
- The test result cards contain a lot of information at once. We should boil this down to clearly communicate what their result is and why they have received that result. Other information such as the disclaimer regarding the efficacy of the application could be moved elsewhere. In this instance, it could be a fullscreen popup the subject has to agree to before starting any of the tests.
- Improve the instruction cards by including more visual aids, rather than showing a massive neutrally-colored wall of text. We should also explicitly state that the test should be conducted in English.
- Since the current iteration isn't able to distinguish between different users, we cannot determine when a person has seen and understood the demonstration card and instruction popup, and thereby avoid showing them unnecessarily. We could however only show it when the baseline is run without any previous data, and assume this means a person who is unfamiliar with the application has started the test.

We consider the experiment to be successful on two accounts. First, we gathered a lot of actionable feedback which very clearly outlines the weak points of the application, helping us determine where we could focus our attention on hypothetical future iterations. Second, we were able to confirm that the application is intuitive enough for users to use the app without requiring instructions from an expert.

9.3 Additional Reflections

Upon reflection on our validation tests, we have also identified two error sources worth mentioning. The first error source stems from us not considering that many of our test subjects speak Danish as a primary language, and would therefore need specific instructions as to which language they should conduct the test in. This has led to some subjects reading flashcards aloud in English, some in Danish, and some switching languages midway through the test. All tests should have been conducted in English exclusively, as K-D has not been validated for other languages[33]. The second error source is that since our speech-to-text implementation would regularly include extra `0`s, as explained in Section 6.6.3, we decided early on to not include `0`s in our randomly generated flashcards. We did not revert this change after deciding to abandon speech-to-text, and as such all validation tests were performed with a version of the app that only showed digits 1-9. We have since patched the application, but as validation tests were not performed to official specifications, their results might be influenced negatively.

We do not believe these error sources have significantly affected our results, as they both appear to be relatively minor deviations from the official specifications.

9.4 Conclusion

The two tests were completed as expected and we gathered the amount of information we had planned from both. We retroactively realized our fatigue test could have used more data points to be more

accurate, but the data provided does indicate that our application is sensitive to sleep deprivation, and thereby potentially also sensitive to detecting some forms of mTBIs. We discovered several minor issues during the usability test, which in theory could be used for future iterations, but overall the test confirmed that users do find the application to be intuitive.

This concludes the practical development and assessment of the app and we will now move on to reflecting upon the project as a whole.

10 Discussion

In this section, we discuss our findings from throughout the project and our final thoughts on our product and its performance.

10.1 Results

10.1.1 Product Success

To evaluate the success of our product implementation, we look back at the lean canvas presented in Figure 4 and determine how we have solved problems, considered customer segments, and provided a solution and unique value.

We believe we have put together an app that provides the 3 solution features described in the lean canvas: Our app provides the user with a set of King-Devick flashcards that are always unique and randomized, preventing users from improving their score by memorizing a given set of flashcards.
The playback feature allows assessors to review the results after the test, without having to manually keep track of errors while the test is carried out.
We successfully implemented eye tracking and integrated it into the playback feature. While this analysis isn't always entirely accurate, it usually at least conveys the horizontal scanning motion of the patient, giving you an idea of where the patient falters, backtracks, or skips numbers.

This all leads to the product providing its proposed unique value: We believe our app accurately reflects a standard King-Devick test, and it has the additional, optional benefit of providing information on the patient's gaze during the test.

These features come together to nicely solve the top 3 problems presented in Figure 4. Given that King-Devick tests are used in return-to-play decisions and that our app accurately depicts a King-Devick test, our app could feasibly be used in the screening of concussions. A patient familiar with the test can go through all three test cards in less than a minute, and so the whole test including assessment could take as little as two minutes.
As mentioned in Section 2, of course, King-Devick is only one avenue of testing, and as such our app cannot singlehandedly solve the problems as presented in the lean canvas. The test would need to be included in a greater toolbox to properly fulfill its function. Still, within the limits of our scope, we have provided a solution to the problem.

Our research throughout the project has given us insights into how screening is performed for athletes, but we failed to gain much of a perspective on how the military performs these kinds of assessments, and whether our product is a good fit for them.
We weren't able to get in talks with either customer segment, but we have done general user experience testing as part of development of our product. We would describe our tests as being good indicators of civilian use of our app and therefore reflect the use case of athletes and return-to-play decisions. In this context, our app performs well with users, who for the most part only report minor issues such as gaze tracking accuracy.
In the context of military use in warzones, however, we can't say for certain whether or not this app would be a valuable tool. One issue we see is that the current iteration of our app requires an internet connection in order to authenticate and initialize the SeeSo gaze recorder. This may not always be desirable or possible in a combat situation, and would require a different gaze tracking framework in order for the app to provide its unique value. A possible benefit to our solution is that the test is very flexible with regards to what position the patient is in. For example, if necessary, the test can be performed by someone lying in a trench or other tight quarters, unlike tests such as BESS, RT_{clin}, or VOMS.

10.1.2 Project Success

To evaluate the success of this project, we refer to our final problem statement, presented in Section 2.3:

Using modern smartphone technology, is it possible to adapt and improve upon an existing ocular-motor concussion screening technique, and can such a test be made simple enough that a user with little to no experience with concussions can perform the assessment using only a mobile device?

We can consider this statement as three separate questions to be answered in this section: Have we adapted an existing ocular-motor concussion screening technique using smartphone technology? Using this technology, does our adaptation offer some improvement to the technique? Finally, have we made this adaptation simple enough that someone with no experience in concussion screening can complete it?

The first two questions can be difficult to conclusively answer, as we lack the background in neurology and proper testing sample to confirm whether our application accurately adapts the King-Devick test, or indeed improves upon it.

We have extensively studied descriptions of how the King-Devick test is conducted from various scientific research papers[16][25][32][33] as well as our conversation with Dennis Abildgaard. Due to this, we have a high degree of confidence that our adaptation of the test is accurate to the original. We have made conscious departures from the original in the element of randomization we have introduced to the horizontal spacing and number sequences, but we believe the resulting test cards still qualify for use in the King-Devick test.

We have also conducted a fatigue test that revealed our version of the King-Devick test is sensitive to sleep deprivation, a trait it shares with the original test. According to the paper[32] cited in Section 9.1, this behavior implies sensitivity to mTBIs as well. It is worth noting that our fatigue test was conducted on only two people, and we collected only three data points for either state of rest. As such, any results obtained from this experiment cannot be considered conclusive.

Nevertheless, due to our prior research, we believe we have succeeded in reproducing the King-Devick test on a smartphone.

Next, we consider the question of improvement. We have implemented two improvements in our adaptation of King-Devick. The first is the aforementioned randomization of flashcard elements. The second is our implementation of gaze data recording and playback. In our own testing, this feature appears to work reasonably well, although there is room for improvement in terms of overall accuracy.

Generally, even in cases of reduced gaze tracking accuracy, we can usually see the horizontal movement of a subject's gaze as they perform the test. Even if the vertical position is off, the accompanying audio playback can clue the assessor into which row is being read.

During our own testing and user testing, we discovered a number of conditions that greatly reduce the information available in terms of gaze data. For example, one subject wearing glasses experienced very poor gaze tracking results, and another who was squinting due to screen glare experienced similar results. Additionally, tests performed at night as part of our fatigue test revealed that no gaze data is recorded when the test is performed in darkness.

We find these conditions acceptable for the intended use of the app. For example, athletes of physical contact sports don't tend to wear glasses on field, and bright lights that cause squinting can be avoided in most cases. A lack of lighting may come up in certain military use cases, but there's only so much a smartphone can do.

We would assert that we have to at least a certain degree of success implemented gaze tracking as part of our adaptation of the King-Devick test. We do not have the qualifications or testing resources to assert to what degree this feature is an improvement to the test, but we can point to neuro-optometrist Dennis Abildgaard suggesting the feature as a possible improvement to the test. With this expert opinion in mind, we assert that we have indeed offered an improvement of the test using modern smartphone technology. More testing among qualified assessors is necessary to determine how big of an improvement this feature is.

Finally, we consider the question of ease of use. Our user testing shows that the majority of users were able to use the basic features of our app, including completing both a baseline test and a post-injury test. Some were a little confused regarding the review portions of the test, which is both understandable and acceptable, given that this part of the app is intended for advanced users such as medics and other professionals involved in return-to-play decisions.

User testing returned mostly positive results on all topics regarding usability of the application. Our

sample size is rather small, and doesn't allow for fully conclusive statements on the matter, and we expect some measure of positivity bias since the subjects were all part of our social network.

Nevertheless, we observed many of the tested subjects as they tried out the application, and didn't notice any signs of them outright struggling with using the product. For that reason, we assert that we have created an application that is easy to use, even for people with no experience in screening for concussions.

All in all, we consider this project a success. Not only have we fully answered the questions posed by our final problem statement, but we have answered affirmatively that yes, we can adapt an existing concussion screening technique, we can improve it using smartphone technology, and we can make it easy to use.

10.2 Future

The app presented as part of this project is a satisfactory prototype, but in projects like this, there will always be a list of features that would benefit the product if we had had more time and resources. Obvious features include a calibration activity and functioning speech-to-text, but we have also given thought to more comprehensive features, such as the user journey after a completed test and the handling of multiple users.

10.2.1 Post-Test User Journey

An important aspect to consider for the future of the app is how it should report the results of user tests. In the case of a concussion, medical staff may be interested in the details of the test performance, as it may shed light on the severity or type of injury.

This requires a feature that can export data related to the test, and the question is how this data should be encoded and how it should be presented. The performance of each test card can be fully described using four overall types of data:
The flashcard itself, described by its seed and flashcard index.
The King-Devick score, represented by the time in seconds it took to complete the flashcard.
The gaze tracking data, represented as a sequence of timestamps and coordinates.
The audio, encoded as an .mp3 file.

Full test data could be exported as three instances of each of these four data types (one for each test card) and reconstructed by a program into the form we see in the review flashcard activity. Alternatively, the whole test could be exported as a video file for more universal access.

An alternative method of presenting gaze tracking data would be displaying the path of the gaze in a single image. This would grant an immediate overview of the overall scanning path of the patient, but lacks synchronization between gaze and audio which is useful for identifying when mistakes occur.

A more advanced reporting feature could take advantage of SeeSo's saccade analysis feature. Given that SeeSo only outputs data at a frequency of 30Hz, we doubt that this feature is wholly accurate, but assuming it works, it may be able to provide a shorthand overview of where fixations occurred, and how well they line up with the positions of numbers on the flashcard. This could be presented either as a statistic of missed and accurate fixations, or an overlay on top of the flashcard highlighting missed fixations.

We don't expect detailed reports to be hugely important to the future of this app, as it is intended to be used as part of a larger suite of screening tools that detect different concussion-related impairments. When an assessor must consider the results of several other tests, a simple stat such as the King-Devick score and percentage of missed saccades may provide a smoother user experience compared to watching a minute-long video file.

10.2.2 Multiple Users

The current prototype saves only one baseline score, and as such would not satisfy the use case of a medic using the app to screen patients belonging to a sports team or combat unit. The primary user, i.e. the medic, should be able to set up and select profiles for the patients they are responsible for, and the app should track separate baseline scores for each of these profiles.

When we deliberate on the design of a multi-user setup, we consider existing screening apps on the market, such as those described in Section 2.2. HeadCheck ties staff and athlete profiles to organizations, such as colleges or sports teams. That way, medics that work for the same organization can all access each athlete associated with that same organization. Not only can any medic registered to the organization select a registered athlete for testing, they can also look at that athlete's previous test results and baselines if further context is needed.

We imagine a future multi-user setup to work in much the same way, with the main users being medical staff registered to an organization, and patients being a secondary sort of profile that medical staff can create, select, inspect, and conduct tests for. These secondary profiles would be tied to the same organization as the medical staff, allowing other medics to interact with those same patient profiles.

The specific details of implementation are of course more difficult to hash out. How are organizations registered to the app in the first place? Who is in charge of registering medical staff to each organization? Does the app require organization administrator roles? What is the level of involvement of the developers when it comes to moderating/maintaining these organizations?

Answering these questions would have taken a lot of attention away from the minimum viable features of this app, and as such have been left outside our scope.

10.3 Personal Reflections

Kotlin Comprehension
Utilizing Kotlin has definitely been a challenge for the two of us, although being familiar with other statically-typed object-oriented languages helped us a lot. It is very evident from reading the source code that we have learned a lot as we progressed through the implementation, as parts of the codebase utilize Kotlin features that other parts could have benefitted from greatly. A concrete example of this is the Kotlin keyword `lateinit`[34], which lets you initialize an otherwise non-nullable type outside the constructor. This is very useful when developing Android applications, as many variables have to be initialized in Android SDK callbacks. Much of our application, therefore, use nullable types, even though we never really expect them to be nullable - in these cases, using `lateinit` is preferred, as it correctly communicates the intent that the variable is not supposed to be `null`, while still letting us initialize it in the correct part of the Android SDK initialization flow. While this has deteriorated the quality of the codebase, we are happy to have chosen Kotlin, as we appreciate having some familiarity with another language.

King-Devick Test Concerns
We have several concerns regarding the K-D test specifications. We are concerned about how passing a screening test updates the baseline test score, as this can introduce false positives. Players who are frequently tested, but who continually pass, will get better and better baseline scores until they inevitably fail the test and are removed from play. Since patients only need to re-do their baseline annually[25], this can be a significant failure point during a football season or on the front lines of war. We also have concerns regarding the cutoff requirements for failing screening tests, as there is still no medical consensus for what the cutoff should be. The official website suggests any deterioration is grounds for failure, whereas some clinical papers have used an error margin of 3-7 seconds[33]. Despite our concerns, we wanted to implement our King-Devick test as close to the official specifications as possible, as we are not qualified to make these medical assessments. We have therefore not done anything to address these concerns in our application.

Hardware Reflections
Our application is currently reliant on running on a device that supports the Android operating system. This is beneficial in terms of the ubiquity of such devices but has several significant drawbacks in the context of being utilized in war. Smartphones are quite expensive, especially since our test does not require many of the features modern smartphone hardware provides. Furthermore, smartphone hardware is not designed to be used in the harsh conditions warfare often entails. A primary concern in this regard is the fragility of the glass screen interface, and the short battery life of smartphones, which is problematic during sustained combat. Smartphones are also designed to be connected to a mobile network, which can be problematic as they can be used to disclose the location of troops. It would therefore make sense to

[34] `https://kotlinlang.org/docs/properties.html#late-initialized-properties-and-variables`

design hardware that is more suited for a military environment, while also keeping costs low - this will probably also involve having to improve the performance of the codebase, partly to improve the battery life of the device, but also since cheap hardware often imposes serious limitations on computational power and the size of memory and permanent storage.

Project Management Approach

We've been able to maintain a fairly relaxed approach to project planning during our work. We were for the most part able to keep up with our initial project plan, as any delays that occurred were offset by other tasks finishing ahead of schedule or falling out of scope. Our familiarity and experience working together the past 3 years have been a huge advantage, as we could confidently rely on each other's discipline and skill to see tasks through, but also clearly communicate when things weren't working out. For example, when Noah was getting frustrated and stuck trying to get MediaPipe working, he handed the task over to Daniel and moved on to other technical experiments without loss of momentum.

11 Conclusion

In our final problem statement, we wanted to explore whether it is possible, using modern smartphone technology, to adapt and improve upon an existing ocular-motor concussion screening technique and whether such a test can be made simple enough that a user with little to no experience with concussions can perform the assessment.

We went through several iterations of exploratory experiments to determine which type of medical screening tool to use, eventually deciding on implementing a King-Devick test in our mobile application. We were able to successfully implement a virtual version of this test, including some additions such as gaze tracking and randomized flashcards that we believe improve upon the analog version of the test and aid medical professionals in making correct assessments when screening patients.

While we are very satisfied with the final version of the application, we did have to scale back the scope somewhat in order to finish our prototype in time. Some of the features we would have liked to implement include an automatic analysis of the gaze data, as a secondary vector of examination that medical professionals would be able to utilize. We also had to cut some quality-of-life features that would make using the application easier. This includes automatically registering the numbers spoken by test subjects, which can be used to highlight potential errors and ease the task of assessment. We also wanted to implement calibration of SeeSo's gaze tracking to improve its accuracy.

Our validation tests showed that users were able to successfully use the application with little to no experience, but we are not able to validate that the application screens mTBIs as well as conventional K-D testing does. This is in part due to the insufficient amount of data gathered during our fatigue experiment, but also due to the ethical and time restrictions of validating our prototype on real patients with mTBIs. It should however be noted that since our implementation follows the analog K-D specifications, we cannot determine any reason why it should be less effective than utilizing analog flashcards.

In conclusion, we believe our prototype is able to practically assess any irregular visual dysfunctions in a patient, and that any person is able to use it successfully even if they have little to no experience with concussions. We have also identified several aspects of the prototype which can be iterated upon to further improve the efficacy of the application. The project as a whole can furthermore be used as a foundation for further research to explore gaze-tracking and other methods of data analysis as additional vectors of evaluation in the screening of concussions.

References

[1] Dansk Center for Hjernerystelser. Tal og statistik. https://dcfh.dk/viden-om-hjernerystelse/tal-og-statistik/. Accessed 02/06/2023.

[2] Centers for Disease Control and Prevention. Get the facts about tbi | concussion | traumatic brain injury | cdc injury center. https://www.cdc.gov/traumaticbraininjury/get_the_facts.html, March 2022. Accessed 6/2 2023.

[3] Institute of Medicine (US) Board on Neuroscience and Behavioral Health. Is soccer bad for children's heads? summary of the iom workshop on neuropsychological consequences of head impact in youth soccer. *Washington (DC): National Academies Press (US)*, 2002. https://www.ncbi.nlm.nih.gov/books/NBK220606/.

[4] Jeffrey S. Kutcher and Christopher C. Giza. Sports concussion diagnosis and management. *Continuum Lifelong Learning in Neurology*, 20(6 Sports Neurology), 2014. https://pubmed.ncbi.nlm.nih.gov/25470160/.

[5] University of Pittsburgh Medical Center. Types of concussions | upmc sports medicine concussion program. https://www.upmc.com/services/sports-medicine/services/concussion/symptoms-diagnosis/types. Accessed 6/2 2023.

[6] Chizuk et al. Concussion reporting behaviors in student athletes across sexes and levels of contact. *Journal of Concussion*, June 2021. https://journals.sagepub.com/doi/full/10.1177/20597002211015093.

[7] Meier et al. The underreporting of self-reported symptoms following sports-related concussion. *J Sci Med Sport*, 18(5), July 2014. https://pubmed.ncbi.nlm.nih.gov/25150463/.

[8] National Center for Injury Prevention and Control. Report to congress on mild traumatic brain injury in the united states: Steps to prevent a serious public health problem. *Atlanta, GA: Centers for Disease Control and Prevention*, September 2003. https://www.cdc.gov/traumaticbraininjury/pdf/mtbireport-a.pdf.

[9] University of Pittsburgh Medical Center. Concussion statistics and facts | upmc | pittsburgh. https://www.upmc.com/services/sports-medicine/services/concussion/about/facts-statistics. Accessed 6/2 2023.

[10] Murray et al. Smooth pursuit and saccades after sport-related concussion. *Journal of Neurotrauma*, 2020. https://pubmed.ncbi.nlm.nih.gov/31524054/.

[11] Cleveland Clinic. Concussion test: Assessment types & how to interpret results. https://my.clevelandclinic.org/health/diagnostics/22267-concussion-test, December 2022. Accessed 5/2 2023.

[12] National Institute of Neurological Disorders and Stroke. Ninds cde notice of copyright: Axon sports computerized cognitive assessment tool (ccat). https://commondataelements.ninds.nih.gov/report-viewer/24123/Axon%20Sports%20Computerized%20Cognitive%20Assessment%20Tool%20(CCAT). Accessed 12/05 2023.

[13] David Darby. Re: Validity of axon sports computerized cognitive assessment tool (ccat). https://theconcussionblog.files.wordpress.com/2011/02/axon-sports-ccat-cmo-validity-letter.pdf. Accessed 12/05 2023.

[14] Shores et al. Glasgow coma scale (gcs) & abbreviated westmead pta scale (a-wptas). https://www.mq.edu.au/about/about-the-university/our-faculties/medicine-and-health-sciences/departments-and-centres/department-of-psychology/glasgow-coma-scale-gcs-and-abbreviated-westmead-pta-scale-a-wptas. Accessed 3/3 2023.

[15] ImPACT Applications, Inc. Impact concussion test | impact applications. https://impactconcussion.com/, January 2023. Accessed 6/2 2023.

[16] Leong et al. The king—devick test for sideline concussion screening in collegiate football. *Journal of Optometry*, 2015. https://findit.dtu.dk/en/catalog/564b12f336b20a153b026d98.

[17] Eckner et al. Effect of concussion on clinically measured reaction time in 9 ncaa division i collegiate athletes: A preliminary study. *PM&R*, 3(3), 2011. https://www.sciencedirect.com/science/article/pii/S1934148210013456.

[18] University of Pittsburgh Medical Center. Vestibular ocular motor screening (voms) | concussion diagnosis. https://www.upmc.com/services/sports-medicine/services/concussion/symptoms-diagnosis/voms. Accessed 6/2 2023.

[19] Augustine GJ et al. Types of eye movements and their functions. https://www.ncbi.nlm.nih.gov/books/NBK10991/. Accessed 24/03 2023.

[20] Hessels et al. Is the eye-movement field confused about fixations and saccades? a survey among 124 researchers. *R. Soc. open sci*, 5, 2018. https://royalsocietypublishing.org/doi/10.1098/rsos.180502.

[21] University of Copenhagen. Model double diamond. https://innovationenglish.sites.ku.dk/model/double-diamond-2/. Accessed 25/05 2023.

[22] Hunfalvay et al. Smooth pursuit eye movements as a biomarker for mild concussion within 7-days of injury. *Brain Injury*, 2021. https://pubmed.ncbi.nlm.nih.gov/34894915/.

[23] Kaae et al. Vestibulo-ocular dysfunction in mtbi: Utility of the voms for evaluation and management – a review. *Neuro Rehabilitation*, 2022. https://content.iospress.com/articles/neurorehabilitation/nre228012.

[24] Traumatic Brain Injury Center of Excellence. Visual guide to performing a vestibular/ocular-motor screening (voms). https://health.mil/Reference-Center/Publications/2020/07/31/Vestibular-Ocular-Motor-Screening-VOMS. Accessed 10/02 2023.

[25] Whelan et al. King-devick testing and concussion recovery time in collegiate athletes. *Journal of Science and Medicine in Sport*, 25, 2022. https://www.sciencedirect.com/science/article/pii/S1440244022002237.

[26] David Sachs. Sensor fusion on android devices: A revolution in motion processing. https://www.youtube.com/watch?v=C7JQ7Rpwn2k. Accessed 14/3 2023.

[27] Google LLC. Iris | mediapipe. https://google.github.io/mediapipe/solutions/iris. Accessed 28/2 2023.

[28] SeeSo. How to measure distance from a camera. https://blog.seeso.io/how-to-measure-distance-from-a-camera-4ebc36151323. Accessed 19/03 2023.

[29] Fuhl et al. Pistol: Pupil invisible supportive tool to extract pupil, iris, eye opening, eye movements, pupil and iris gaze vector, and 2d as well as 3d gaze. *CoRR*, abs/2201.06799, 2022. https://arxiv.org/abs/2201.06799.

[30] SeeSo. Beginner's guide to eye tracking using seeso. https://blog.seeso.io/beginners-guide-to-eye-tracking-using-seeso-bf6c7a849898. Accessed 19/03 2023.

[31] Plotly. 2d histograms in python. https://plotly.com/python/2D-Histogram/. Accessed 14/03 2023.

[32] Davies et al. Residency training: the king-devick test and sleep deprivation: study in pre- and post-call neurology residents. *Neurology*, 78, April 2012. https://pubmed.ncbi.nlm.nih.gov/22529208/.

[33] Chrisman et al. Impact of factors that affect reading skill level on king-devick baseline performance time. *Annals of Biomedical Engineering*, 47, 2019. https://pubmed.ncbi.nlm.nih.gov/30341738/.

12 Appendix

12.1 UX Preliminary Questionnaire

Concussion Questionnaire

* Required

Smooth Pursuit Roleplay 1

1. How easy is it to assess the correct distance? *

Mark only one oval.

Impossible

1
2
3
4
5

Not an issue

2. How easy is it to assess the correct timing? *

Mark only one oval.

Impossible

1

2

3

4

5

Not an issue

3. How easy is it to keep the patient in the camera's field of view? *

Mark only one oval.

Impossible

1

2

3

4

5

Not an issue

4. How was the experience in general? *

5. Do you see any issues completing the procedure?

Smooth Pursuit Roleplay 2

6. How easy is it to assess the correct distance? *

Mark only one oval.

Impossible

1

2

3

4

5

Not an issue

7. How easy is it to assess the correct timing? *

Mark only one oval.

Impossible

1

2

3

4

5

Not an issue

8. How easy is it to keep the patient in the camera's field of view? *

Mark only one oval.

Impossible

1

2

3

4

5

Not an issue

9. How was the experience in general? *

10. Do you see any issues completing the procedure?

Smooth Pursuit Feedback

11. Which of the following methods of providing timing feedback would you prefer? *

Check all that apply.

- [ ] Metronome vibration
- [ ] Vibration if you move to quickly
- [ ] Visual countdown timer
- [ ] Sound which counts down out loud to 4
- [ ] Metronome sound
- [ ] Other: _____

12. Which of the following methods of providing movement feedback would you prefer? *

Check all that apply.

- [ ] Vibrates more the closer you are to the correct position
- [ ] Visually points an arrow in the right direction
- [ ] Visually shows a facemesh preview of the patient's face at the correct angle
- [ ] Sound which tells you which direction to move ("Left", "Stop", "Right", etc.)
- [ ] Other: _____

13. Additional feedback

King-Devick Roleplay

14. How was the experience in general? *

15. Do you see any issues completing the procedure?

King-Devick Feedback

16. Which of the following methods of analyzing the test would you prefer? *

Mark only one oval.

- Replaying of audio file
- Replaying of audio file and gaze data
- Other: _____

17. Which of the following methods of entering the results would you prefer? *

Mark only one oval.

- ◯ Manually enter the results. (Audio file provided)
- ◯ Results entered automatically, you can edit them
- ◯ Other: _____

18. Additional feedback

_____

_____

_____

_____

_____

Wrap-Up

19. Which type of test do you prefer? *

Mark only one oval.

- ◯ Smooth Pursuit Linear
- ◯ Smooth Pursuit Circular
- ◯ King-Devick

12.2 UX Preliminary Questionnaire Response Summary

Concussion Questionnaire

Questions Responses 6 Settings

6 responses

Link to Sheets

Accepting responses

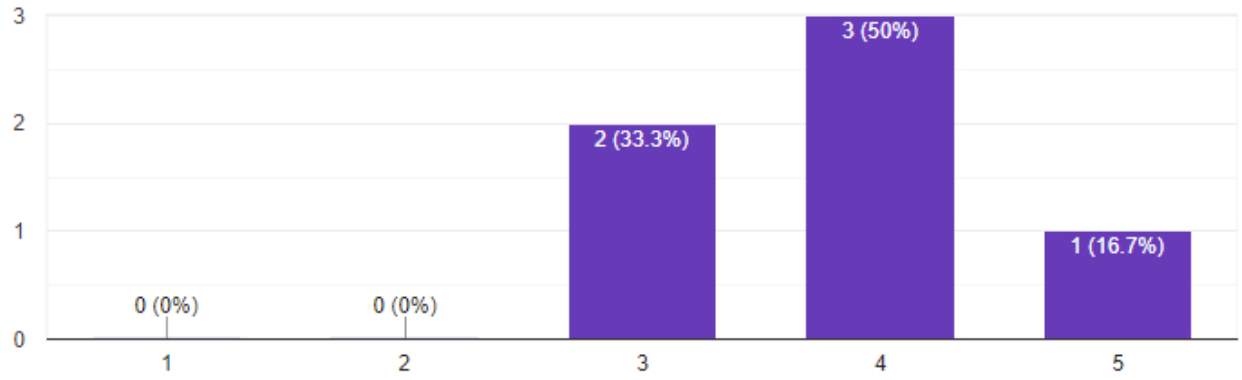
Summary Question Individual

Smooth Pursuit Roleplay 1

How easy is it to assess the correct distance?

Copy

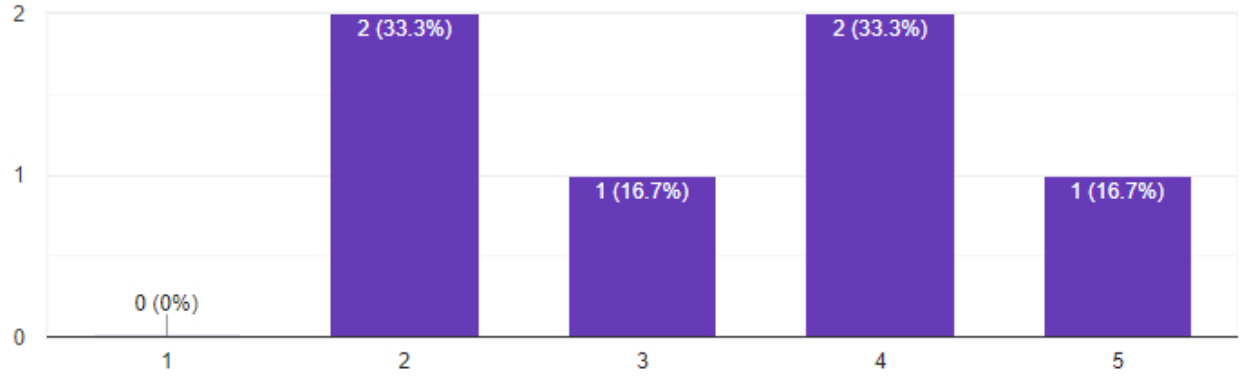
6 responses



How easy is it to assess the correct timing?

Copy

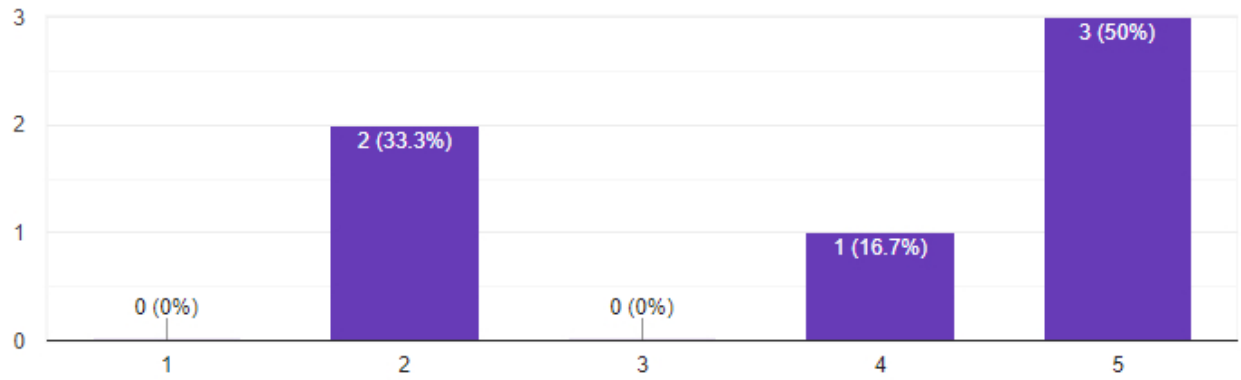
6 responses



How easy is it to keep the patient in the camera's field of view?

Copy

6 responses



How was the experience in general?

6 responses

It was a slightly negative experience

Like a normal camera pan

You have to get used to it. Should possibly include a test scenario you can test on before doing it in the field. There's a lot of different aspects to keep in mind

It was weird and unique and not something I have tried before. You had to jungle several aspects to do it properly

I think once you've tried it and understand what you're meant to do, then it makes sense.

Instructions Werge easy to follow, the instruction to keep patient in view White moving the camera could have ben more outspoken

Do you see any issues completing the procedure?

3 responses

No

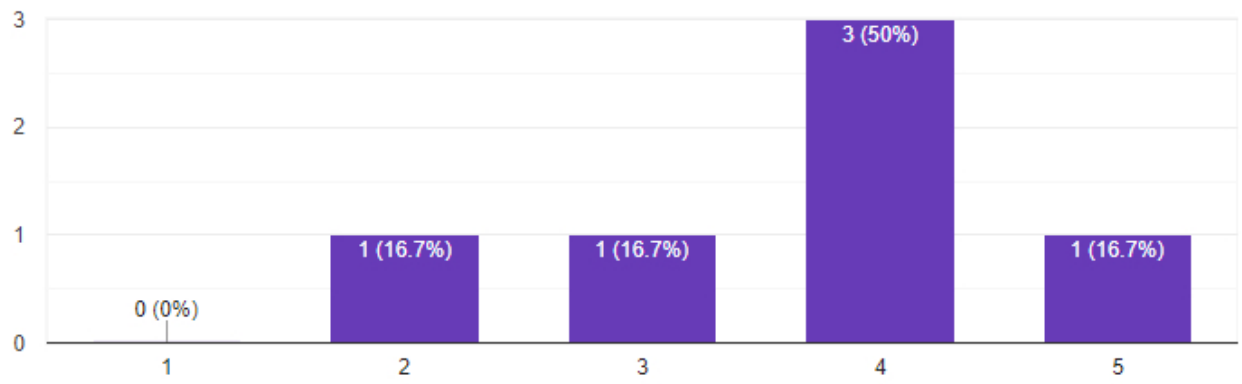
It can be difficult to properly assess the distance and timing. Large chance of human error

Smooth Pursuit Roleplay 2

How easy is it to assess the correct distance?

Copy

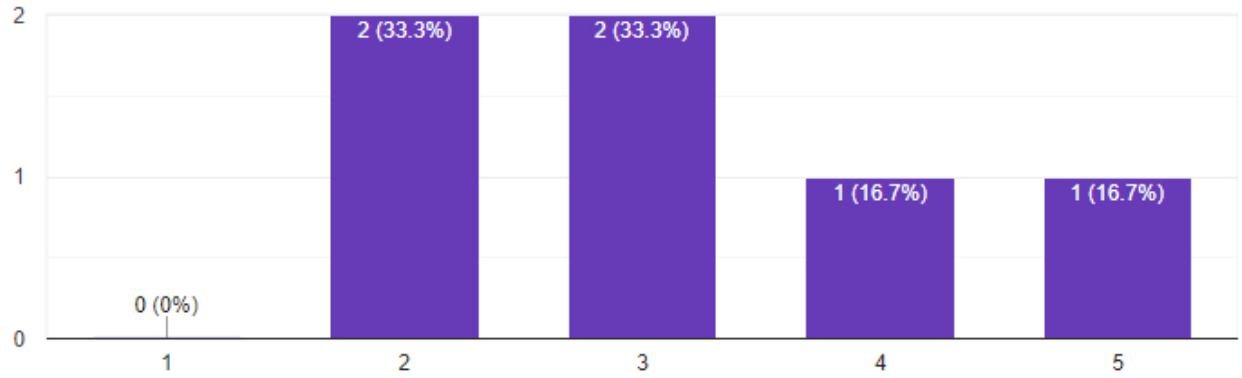
6 responses



How easy is it to assess the correct timing?

Copy

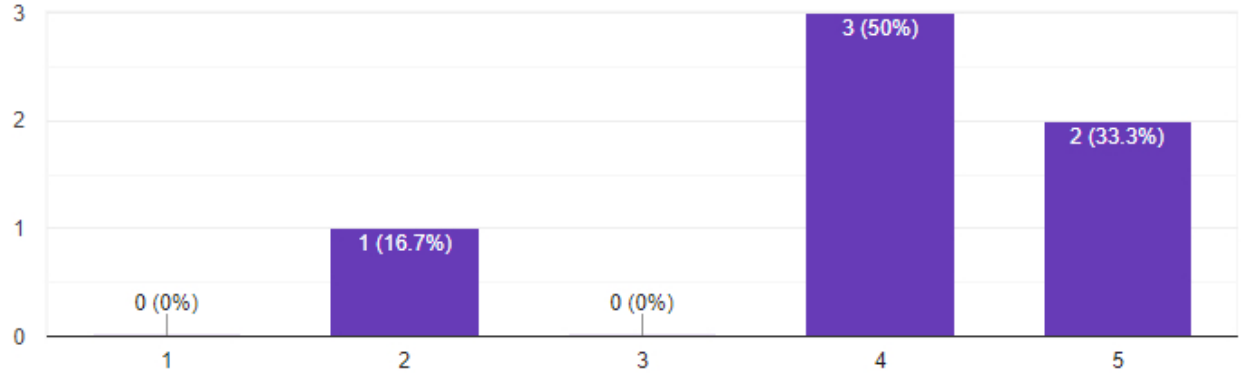
6 responses



How easy is it to keep the patient in the camera's field of view?

Copy

6 responses



How was the experience in general?

6 responses

It is cumbersome

The motion is more advanced so getting it all together is a bit more difficult

It's easier because you don't have to switch hands, but it's more difficult keeping the patient in focus than before.

It was more difficult to keep the patient in field of view and to do it in four seconds.

Cine

Easy to follow instruction, emphasis on keeping patients ene in fokus could have ben more outspoken

Do you see any issues completing the procedure?

3 responses

No

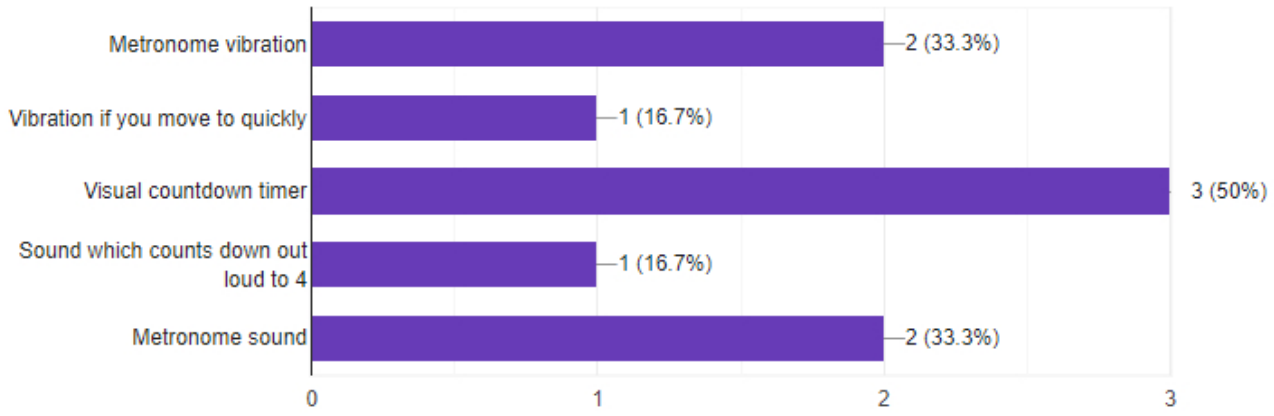
It is difficult to do the circle movement in four seconds

Smooth Pursuit Feedback

Which of the following methods of providing timing feedback would you prefer?

Copy

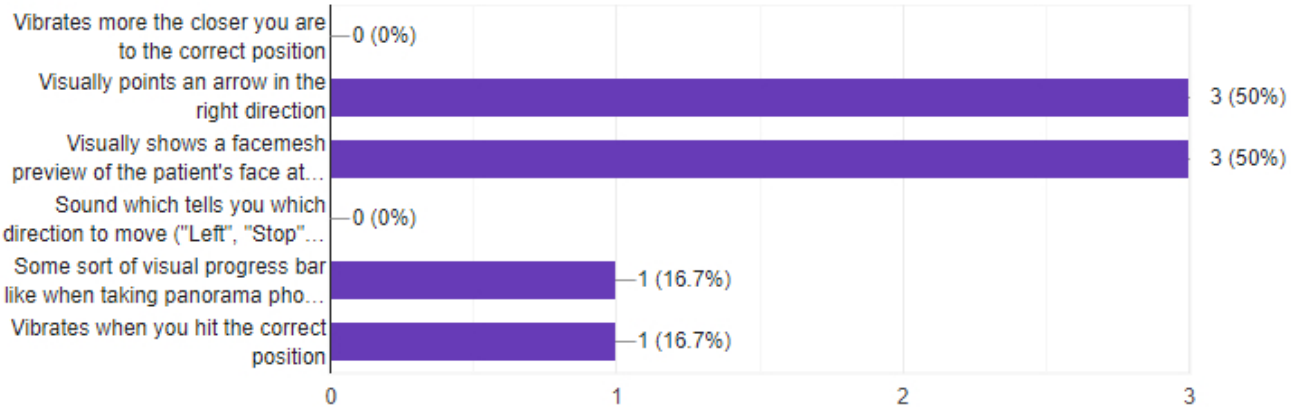
6 responses



Which of the following methods of providing movement feedback would you prefer?

Copy

6 responses



Additional feedback

0 responses

No responses yet for this question.

King-Devick Roleplay

How was the experience in general?

6 responses

- It was more difficult because it was stressful
- A tinge of nervousness when I fucked up. It got increasingly more difficult
- Very intuitive, difficult to do incorrectly. Very nice with the demonstration card
- It was very challenging but at the same time very interesting
- Interesting
- It was easy to understand instructions and Carrey Them out

Do you see any issues completing the procedure?

4 responses

No

If you cannot carry the device

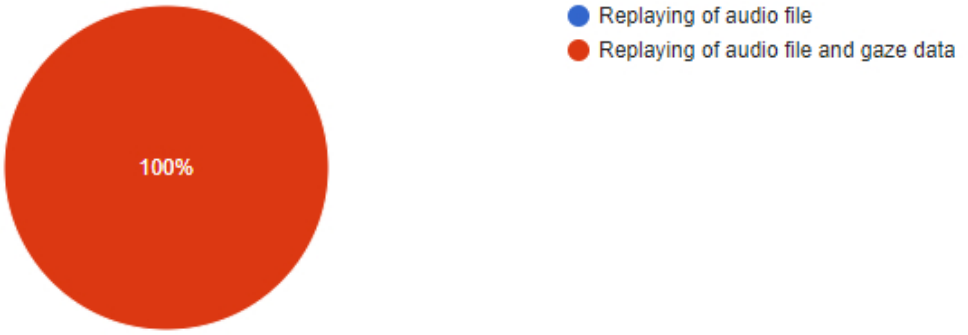
It might be more difficult if someone else holds the phone, so I would like to hold it myself

King-Devick Feedback

Which of the following methods of analyzing the test would you prefer?

Copy

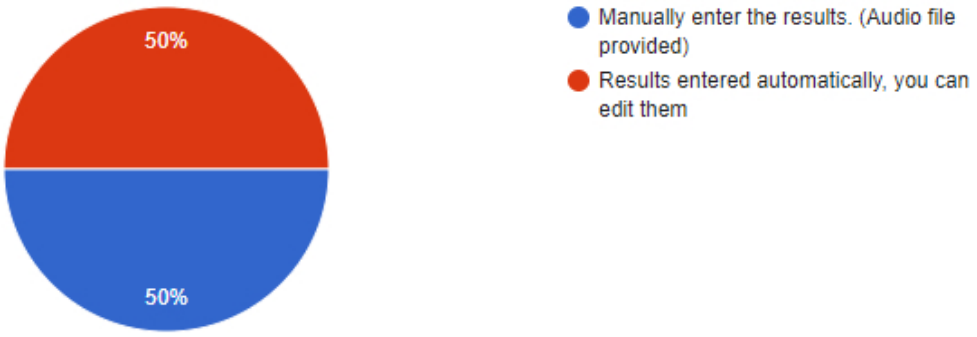
6 responses



Which of the following methods of entering the results would you prefer?

Copy

6 responses



Additional feedback

1 response

Draw red boxes around incorrect numbers on the flashcard. If the box is pressed, the replay of that given moment is played.

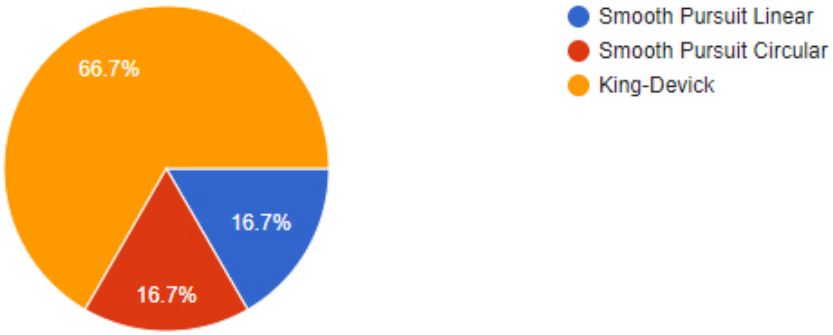
Wrap-Up

Wrap-Up

Which type of test do you prefer?

Copy

6 responses



12.3 King-Devick Wireframe Questionnaire

Saccade Test Wireframe Questionnaire

danieleverland@gmail.com (not shared) Switch account

* Required

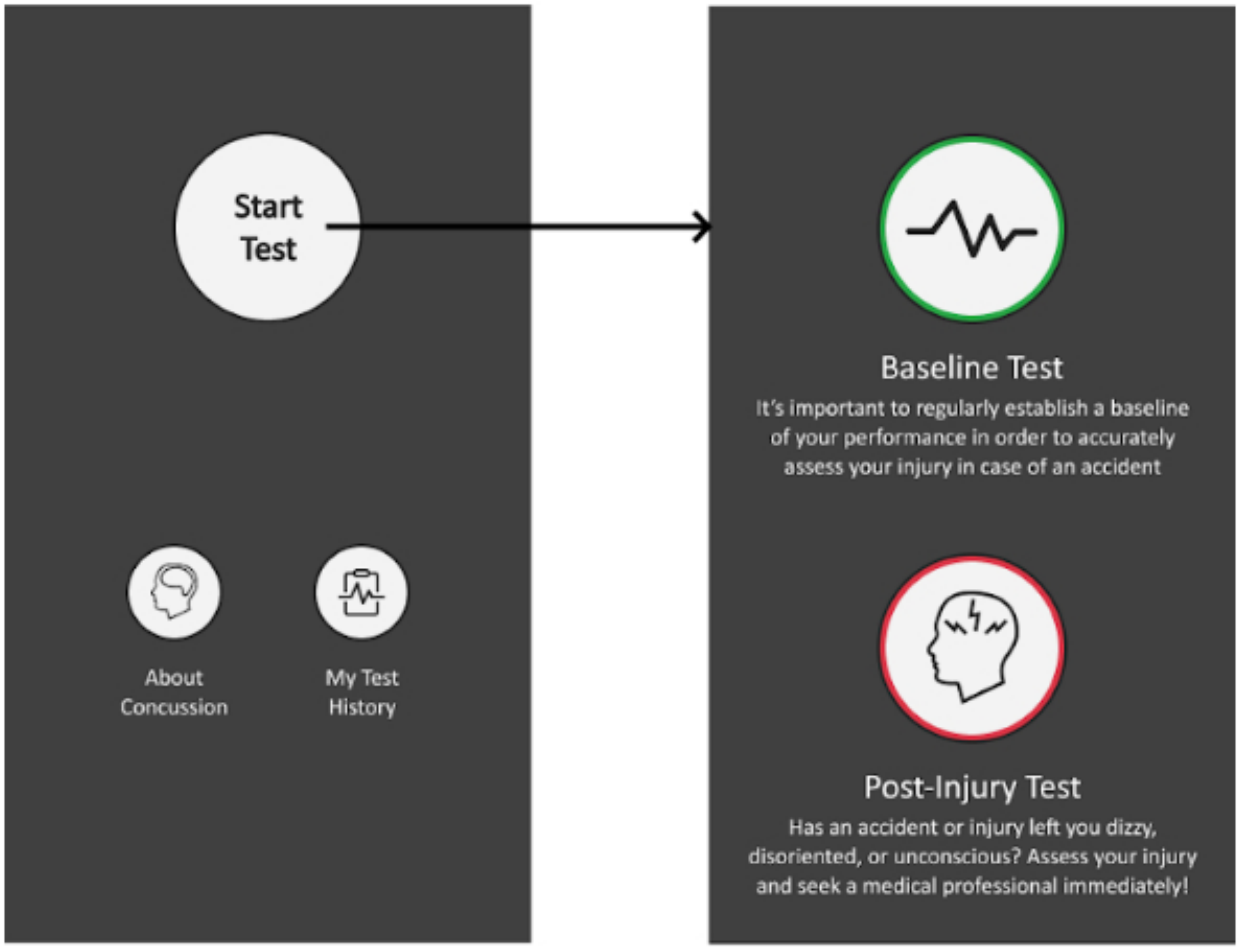
Wireframe Feedback

Front Page

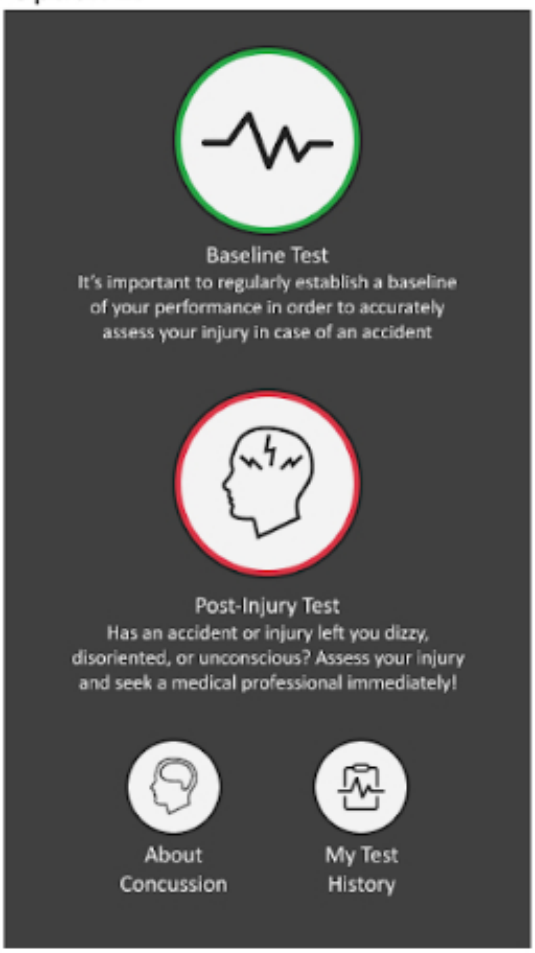
This page shows what the user will be presented with when they open the application. The first option has split the possible actions into two pages, whereas the second option displays all actions on a single page.

Which of the following options do you prefer? *

Option 1:



Option 2:



- ◯ Option 1
- ◯ Option 2

Additional Feedback (Optional)

Your answer

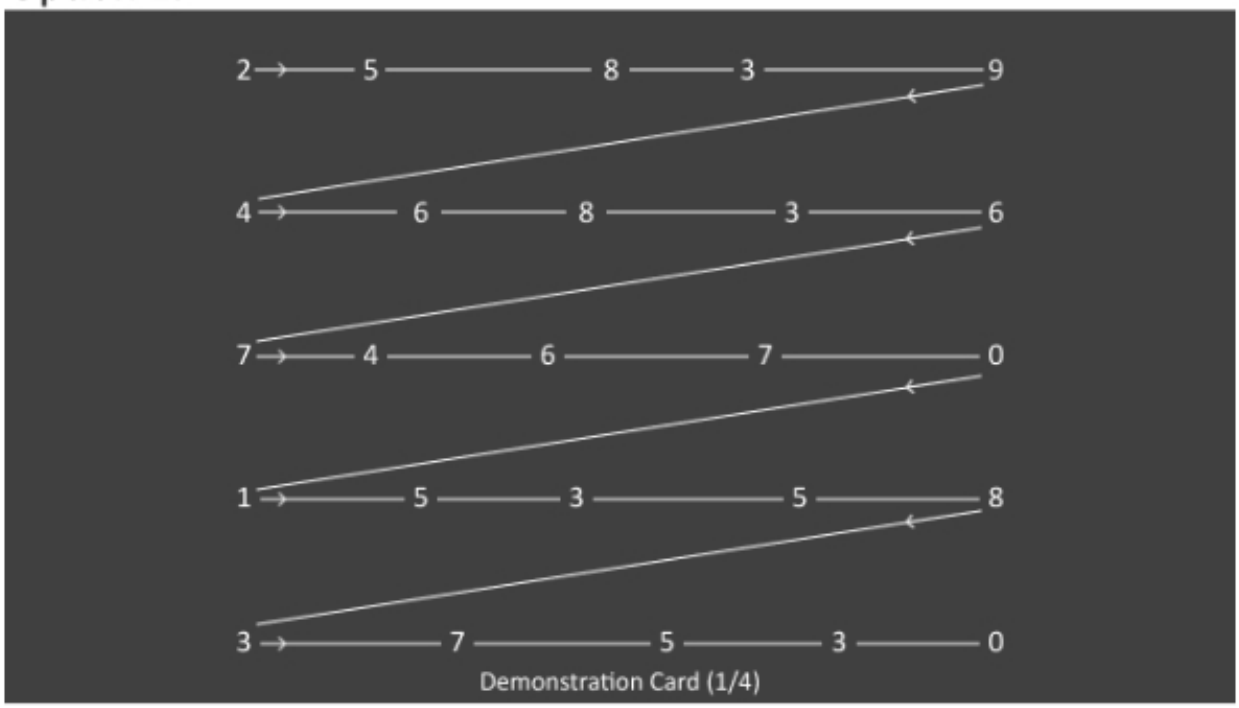
Demonstration Cards

Our application is based on a real concussion screening test called King-Devick, which screens patients by asking them to read single-digit numbers out loud from a flashcard. Before the test is executed, the patient is shown a demonstration flashcard, which the patient can use to familiarize themself with the test without affecting their score.

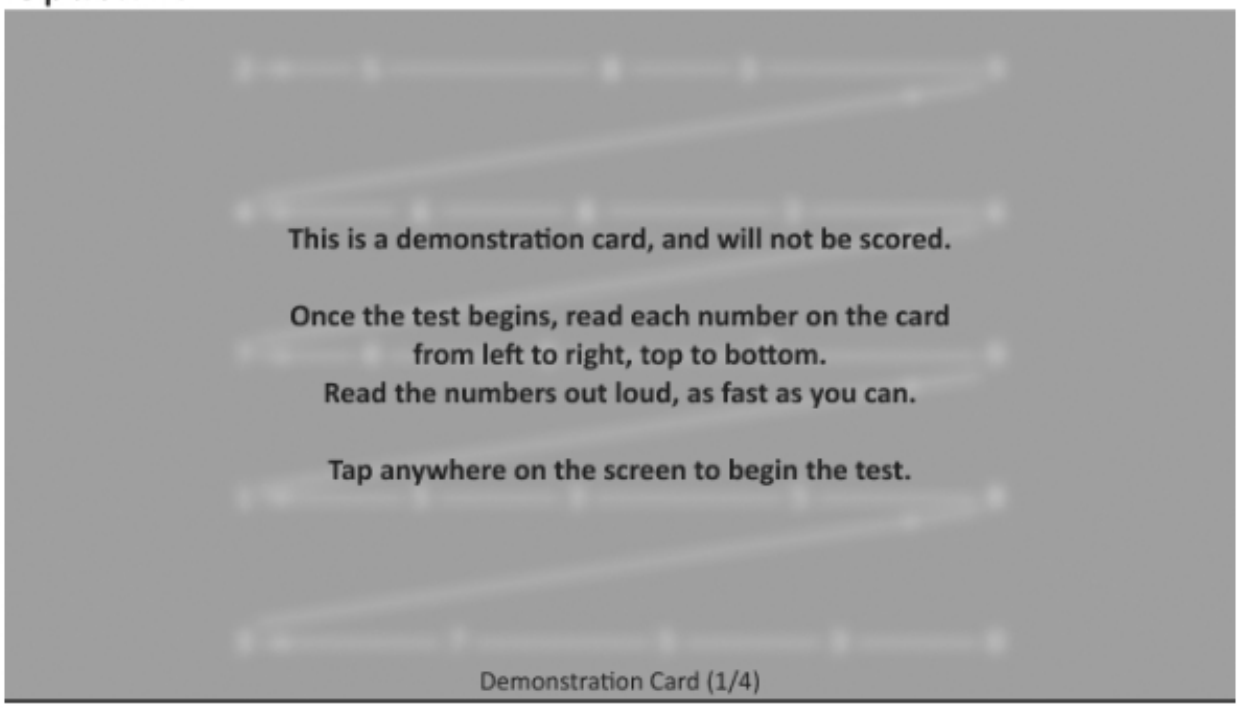
The first option only displays the flashcard itself, whereas the second option contains a popup that explains how to execute the test. In option 2, once you tap the screen, you will be presented with the same screen as in option 1.

Which of the following options do you prefer? *

Option 1:



Option 2:



- ◯ Option 1
- ◯ Option 2

Additional Feedback (Optional)

Your answer

Center Play Button

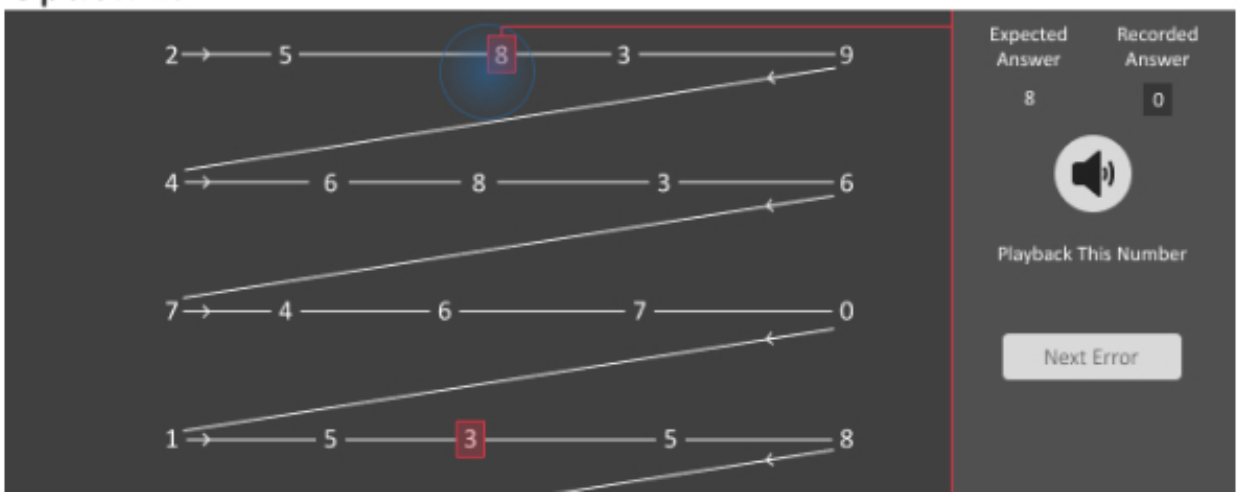
When a patient has completed a flashcard, they are presented with a screen where they can review their attempt. If any mistakes were detected during the test, the app presents the user with an interface to review their answers. The app may be prone to incorrectly flag some answers as mistakes, so this allows the user to correct any false negatives.

Part of this screen is a replay system that lets the user play an audio recording of the attempt so they can listen to each number being read aloud and determine if it was done correctly or not. The app will also display an indicator showing where the patient was looking at the given moment.

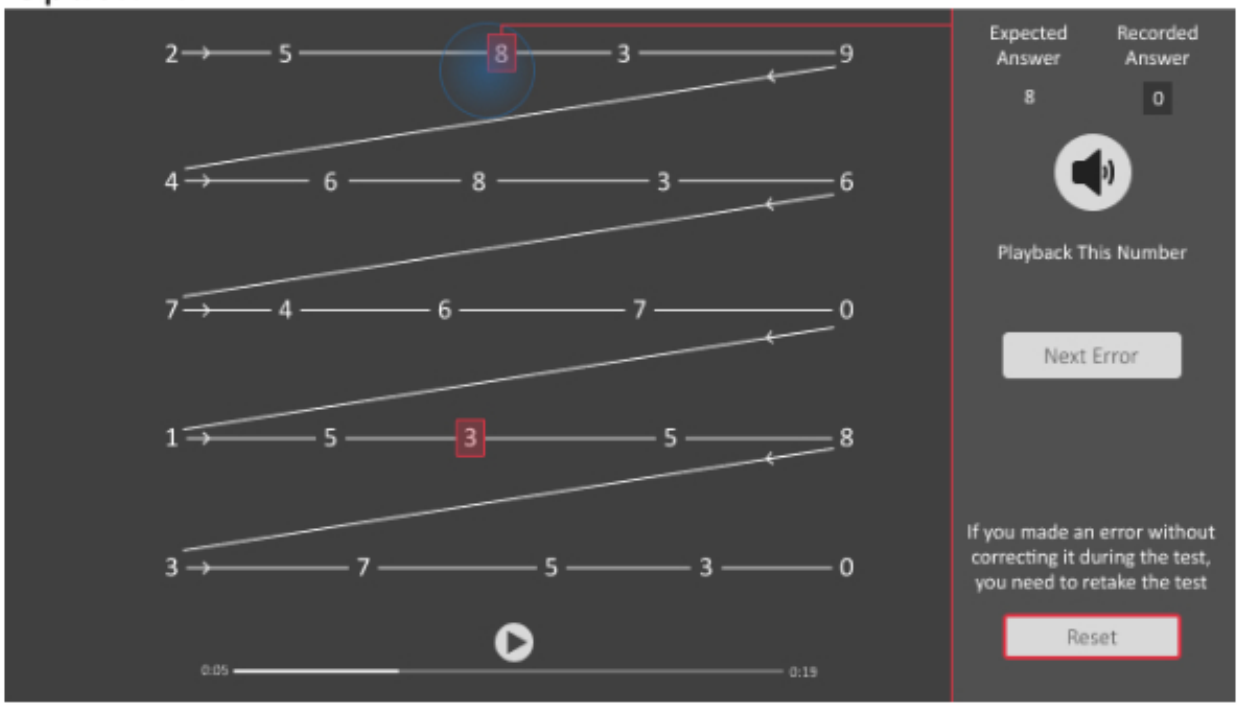
The first option has a large play button on the left side and a larger time indicator. The second option has smaller elements where the play button is centered on the screen.

Which of the following options do you prefer? *

Option 1:



Option 2:



- ◯ Option 1
- ◯ Option 2

Additional Feedback (Optional)

Your answer

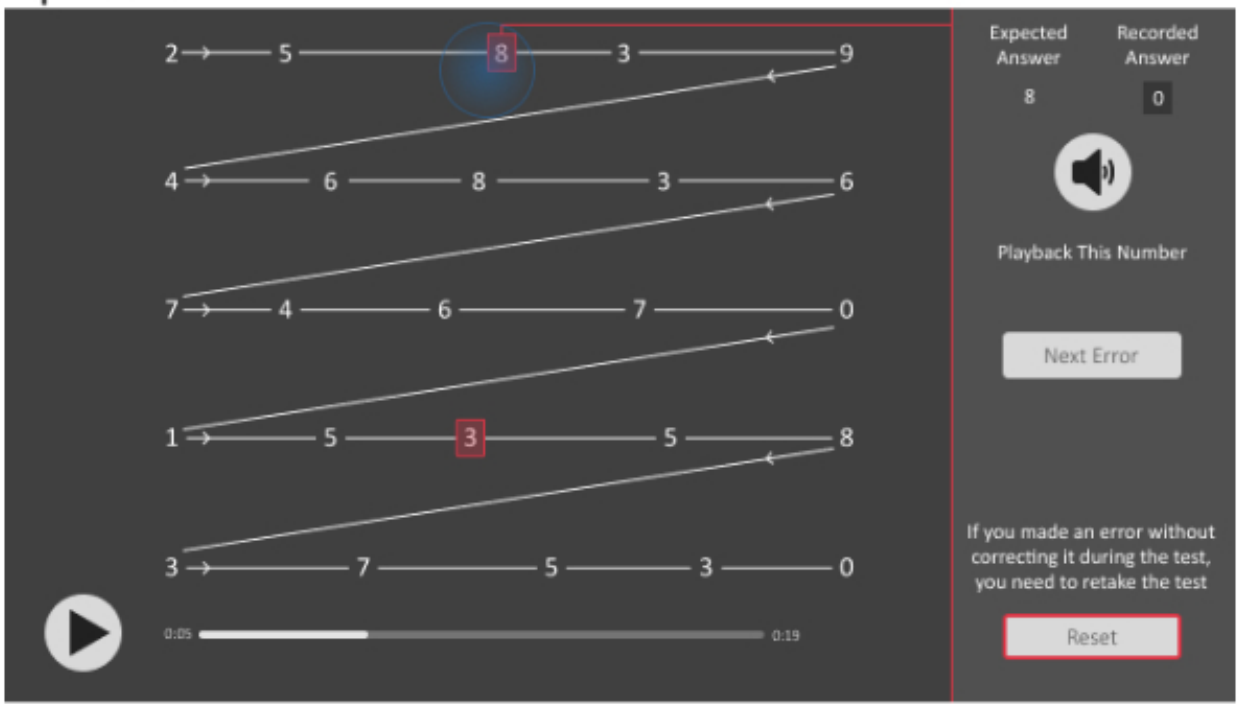
Gaze Indicator

As mentioned, part of the replay system includes an indicator which shows where the patient was looking at a given moment.

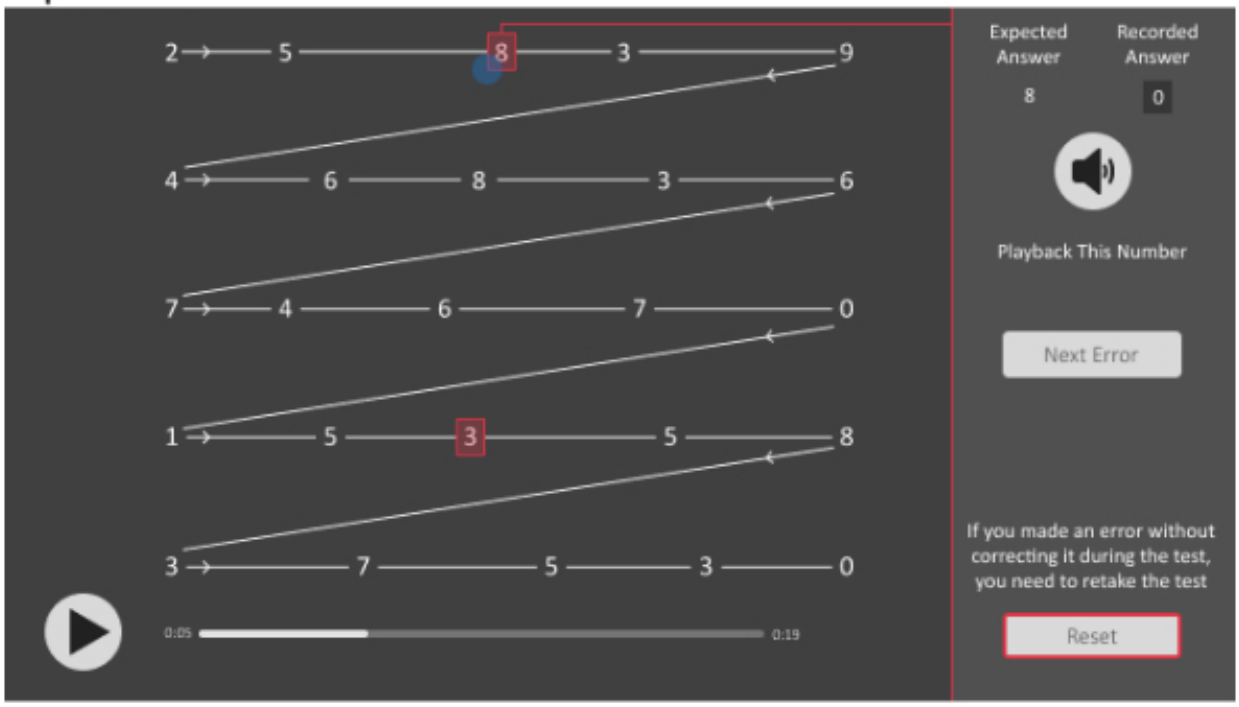
The first option has an outline and a large radial gradient.
The second option is a small dot.
The third option is a large radial gradient without the outline.

Which of the following options do you prefer? *

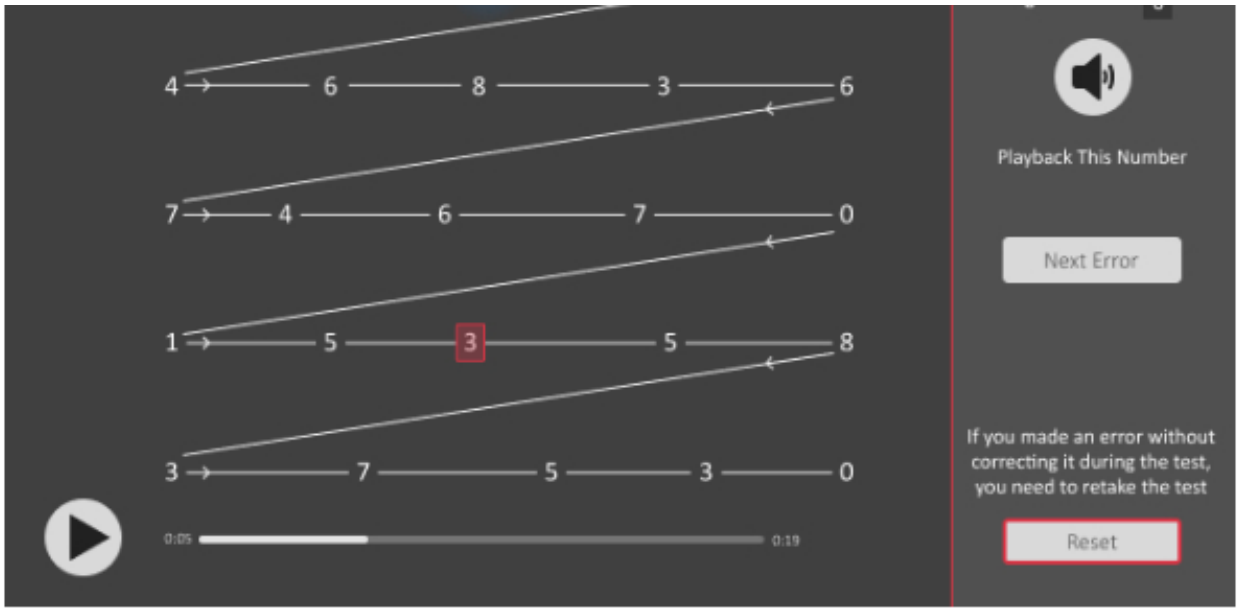
Option 1:



Option 2:



Option 3:



- ◯ Option 1
- ◯ Option 2
- ◯ Option 3

Additional Feedback (Optional)

Your answer

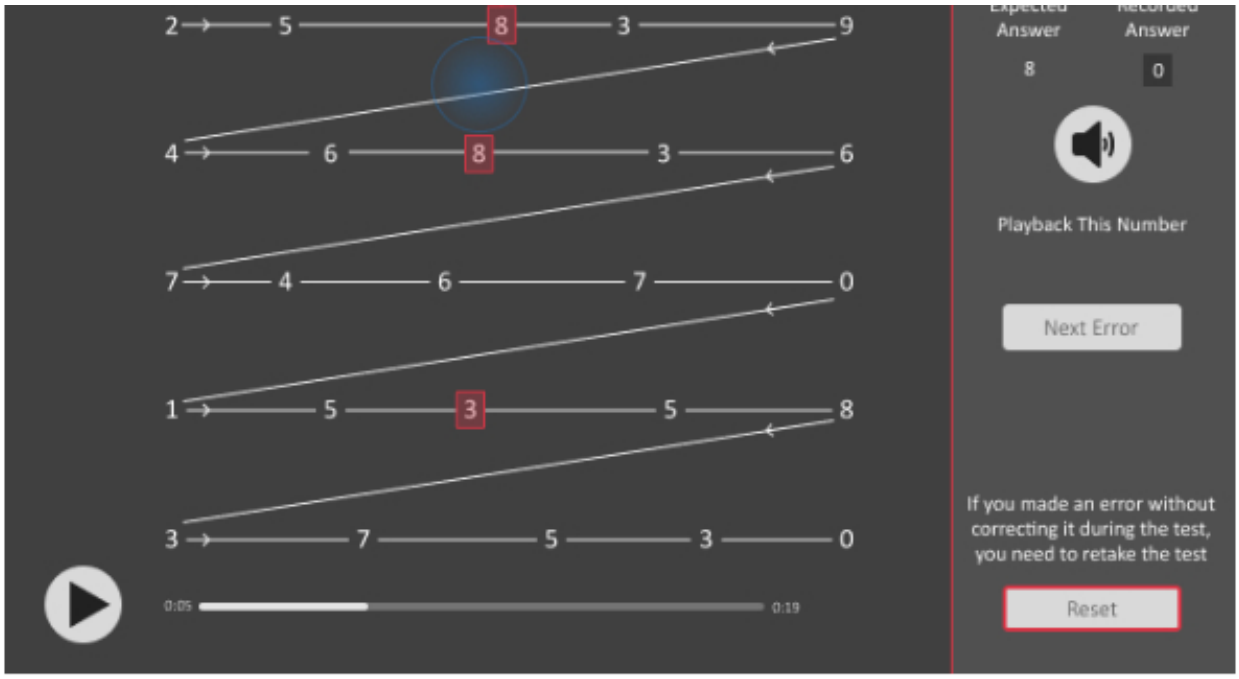
Active Frame Indicator

The replay system lets you author the results by clicking on a number on the flashcard. When you've done so, an indicator may appear showing you which number you're editing.

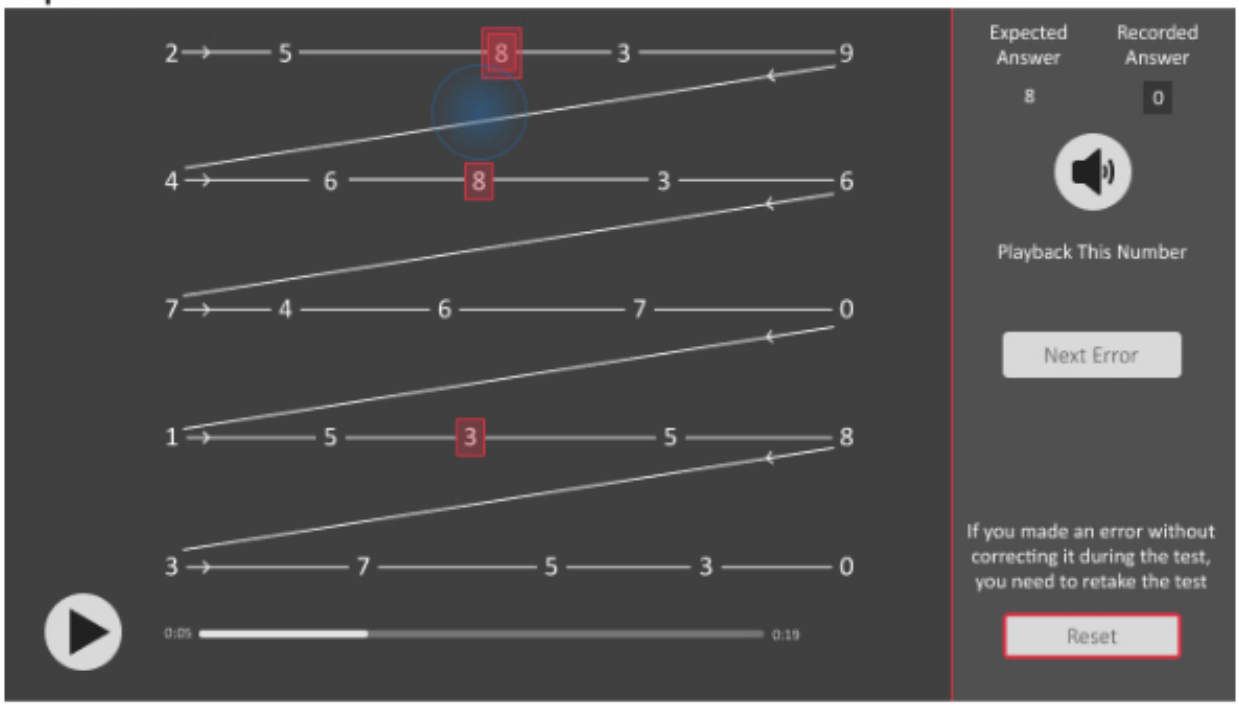
The first option draws a line from the sidepanel to the currently selected number
The second option draws an additional frame around the currently selected number
The third option does not draw any indicators for the currently selected number

Which options do you prefer? *

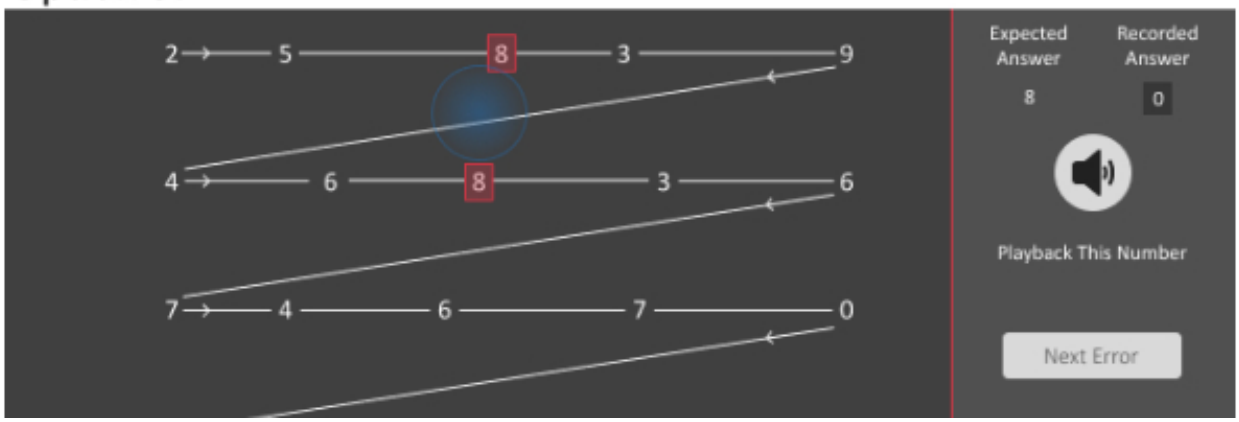
Option 1:

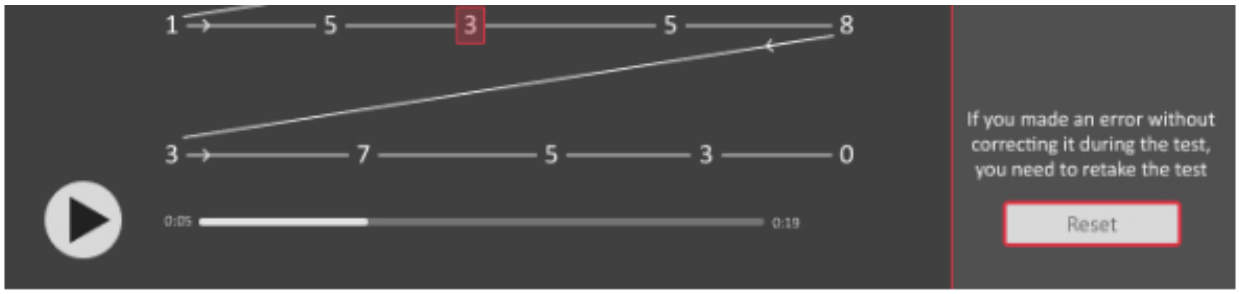


Option 2:



Option 3:





- ◯ Option 1
- ◯ Option 2
- ◯ Option 3

Additional Feedback (Optional)

Your answer

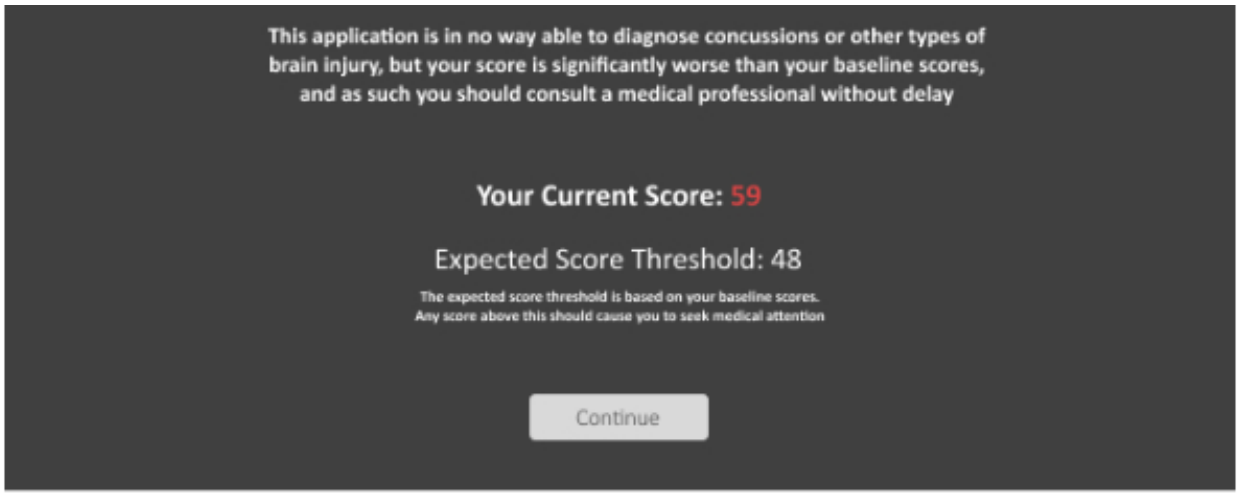
Final Results Page

This page is intended for cases where a test has resulted in the application suspecting the user may have a concussion. This is determined by conducting several tests while the user is healthy, which the application will use to establish a baseline performance metric. If the test score is significantly worse than the baseline scores, the user is suspected of having suffered a concussion.

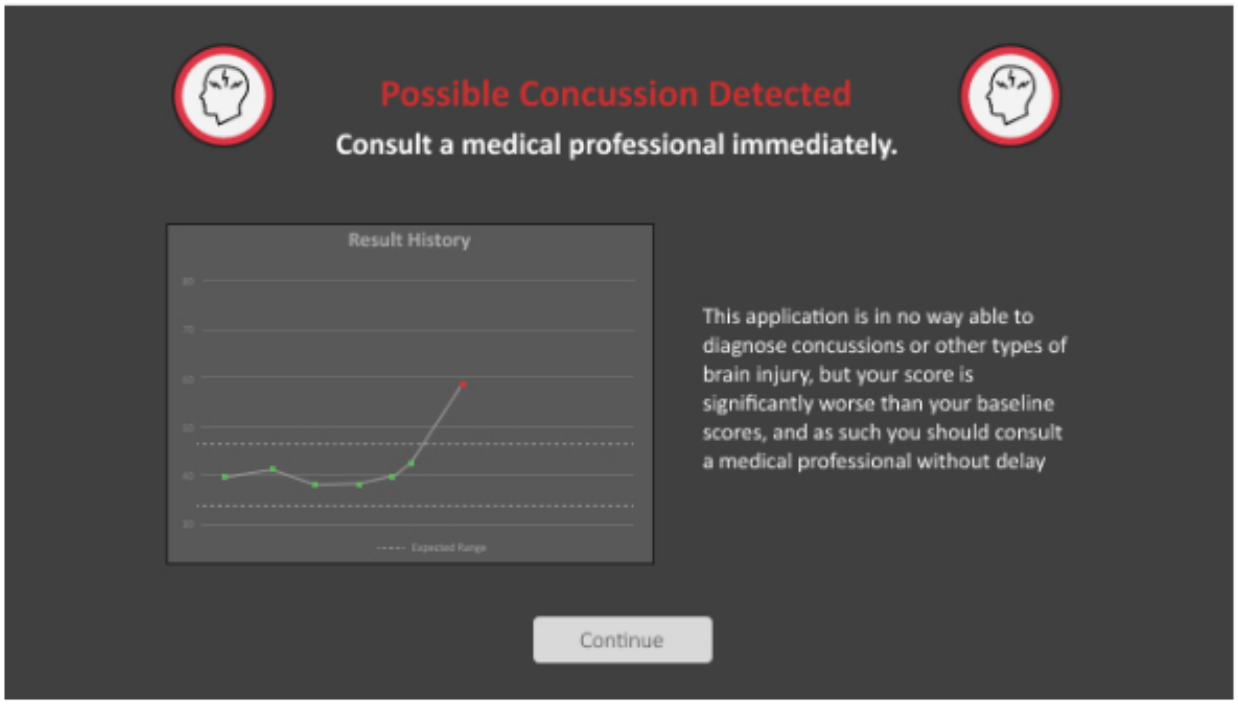
The first option displays some raw data regarding what the current test score is and what the threshold is for flagging a score as being worrisome.
The second option draws the same data in a graph to communicate it visually

Which of the following options do you prefer? *

Option 1:



Option 2:



- ◯ Option 1
- ◯ Option 2

Additional Feedback (Optional)

Your answer

Back Next Clear form

12.4 King-Devick Wireframe Questionnaire Response Summary

Saccade Test Wireframe G

Questions | Responses 11 | Settings

11 responses

Link to Sheets

Accepting responses

Summary | Question | Individual

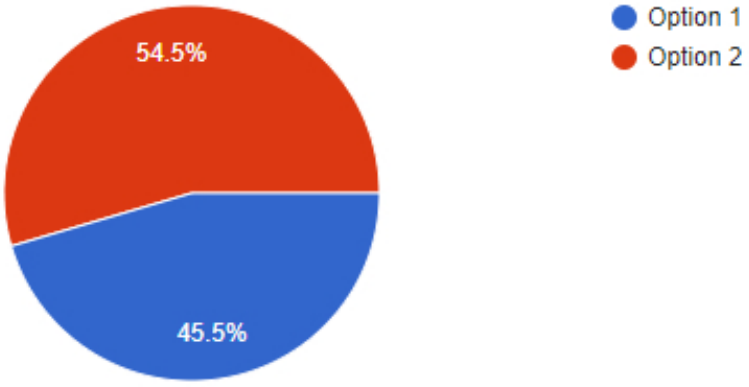
Wireframe Feedback

Front Page

Which of the following options do you prefer?

Copy

11 responses



Additional Feedback (Optional)

6 responses

Less clicks are always preferred and I don't personally think there is too much information in option 2.

With an obvious back button i'd prefer option 1. Don't wanna clutter the UI too much for someone trying to get help.

Start test button is obsolete

The first page has more empty space which is visually more appealing, and allows for more polished UI graphics to be added without cluttering the view. The second page has less elements on it, so the text can be bigger for better readability. The second screen needs a back button though.

Perhaps put Post-injury test at the top instead of baseline test, since the post-injury is more important in case of an emergency.

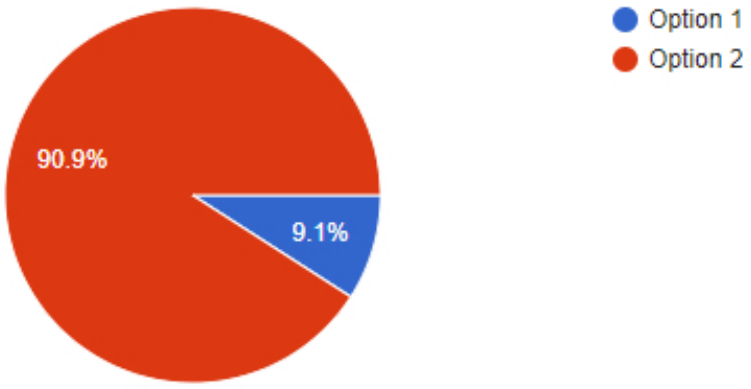
I do not think the screen becomes too cluttered, and, form the perspective of the user, i would think pressind start test would immidieatly start the test.

Demonstration Cards

Which of the following options do you prefer?

Copy

11 responses



Additional Feedback (Optional)

6 responses

Someone holding your hand and telling you what is about to happen is usually nice

There's a risk that the user accidentally presses the popup away, maybe a hold to begin could be

There's a risk that the user accidentally presses the popup away, maybe a hold to begin could be better and would better the chances that the user actually reads the text.

Nr 2 as it shows a description

I prefer option 1 only if there is a test conductor assisting the patient. Main reason is that its easier for a larger demographic of the population to read from a physical card than having to interact with a computer/mobile device. For self-testing i much prefer option 2 over option 1.

Good with some context and information in option 2

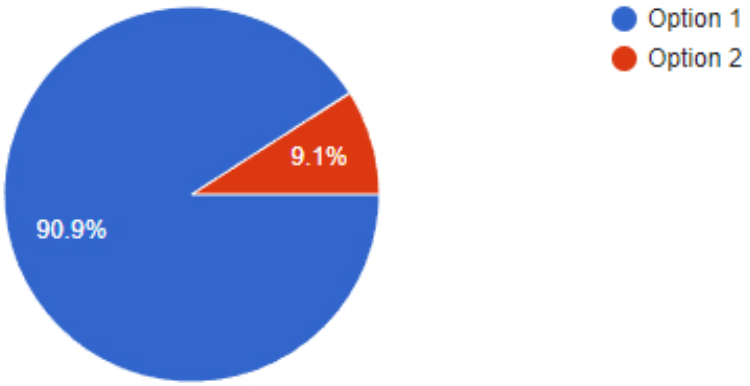
I find the explanation nessecary

Center Play Button

Which of the following options do you prefer?

Copy

11 responses



Additional Feedback (Optional)

4 responses

I like the placement of the button on option 1 more, but I think the size is a bit too large.

Nr 2 as you're less likely to press the play button on your touch device by accident

It was much more obvious to understand option 1's playback UI than option 2's - in fact i think it didnt occur to me that option 2 even had a playback UI for like 20 seconds

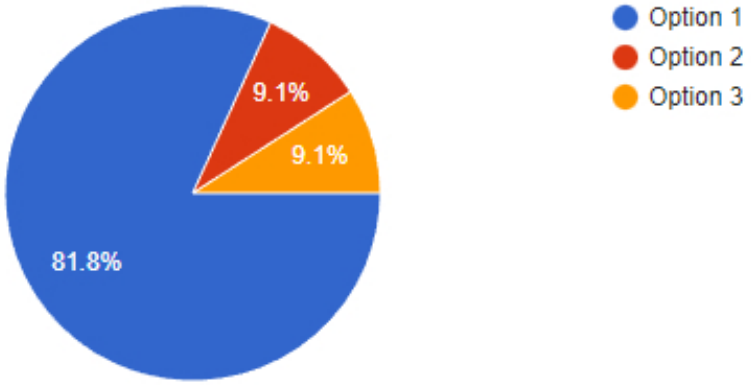
I like the bigger button

Gaze Indicator

Which of the following options do you prefer?

Copy

11 responses



Additional Feedback (Optional)

3 responses

Option 3 seems more fluent

I prefer option 1 because it's easier to see the blue element due to higher contrast, and it is big a blurry enough that it doesn't lie to the user about the precision of the eye tracking feature. Ideally it should encompass the predicted look position and the error. Option 2 looks "too precise" for what the precision probably is. However, if the precision is actually so good that the error doesn't spread more than option 2, i'd start to prefer that more. Although the design for option 1 is more visually pleasing.

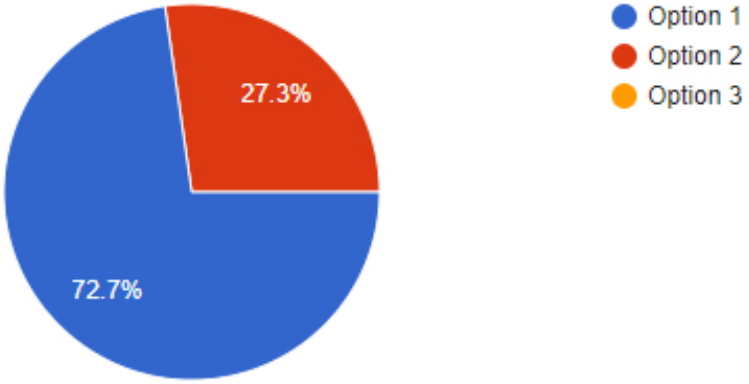
Perhaps increase the constrast between the blue color and the background

Active Frame Indicator

Which options do you prefer?

Copy

11 responses



Additional Feedback (Optional)

3 responses

Nr 2 is easier to see

Option 1 adds too much clutter to the screen. Option 3 is unclear about how far into the errors we currently are. I'd preferred to blur/darken all of the remaining UI to draw focus to a specific element, rather than add more UI elements. I don't think option 2's highlight/current is very clear to understand, making the effect more prominent/larger can alleviate this problem

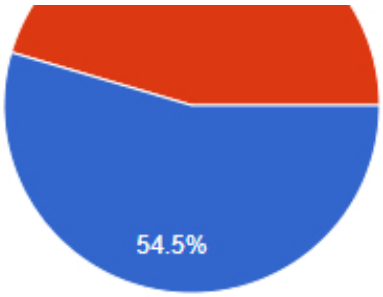
i would be confused without an indicator. I find oprion one the most destinct, but 2 could be functional

Final Results Page

Which of the following options do you prefer?

Copy

11 responses



Additional Feedback (Optional)

7 responses

A number in accordance with a scale is often easier to relate to than a graph. The graph on option 2 does not have much explanation, but could work if a bit of text was added to it.

bruh id prefer not to have a concussion :(

I think the graph provides needed clarity to why the app decided to warn you about a possible concussion

Nr 1 as Nr 2 is difficult to understand

Option 2 gives a much better intuitive understanding of how the application arrives at the conclusion. Consider not using the color red this much as it at first glance is quite alarming (both read header and red outlines). Yellow or orange might be better to avoid potentially panicking the user.

If both could fit in the frame it would be nice

The visual is a little more dramatic and gets the point across better.

General Feedback

Do you have any other feedback regarding the application and its use? (Optional)

2 responses

Awesome idea! Hope it works out!

Looks good!

12.5 Validation Questionnaire

Validation Questionnaire

danieleverland@gmail.com Switch account

Not shared

* Indicates required question

Were you able to start the baseline test from the main menu? *

- Yes
- No

Were you able to start the post-injury test from the main menu? *

- Yes
- No

Were you able to clear the data for your baseline score? *

- Yes
- No

Baseline Test Questions

The following questions relate to the baseline test you had to complete.

Were you able to complete the baseline test successfully? *

- Yes
- No

Were you ever confused about what to do while attempting to complete the baseline test? *

- ◯ Yes
- ◯ No

If you were ever confused about what to do while completing the baseline test, please elaborate:

Your answer

Post-Injury Test Questions
The following questions relate to the baseline test you had to complete.

Were you ever confused about what to do while attempting to complete the post-injury test? *

- ◯ Yes
- ◯ No

If you were ever confused about what to do while completing the post-injury test, please elaborate:

Your answer

Do you understand why you got the outcome you did from the post-injury test? *

- ◯ I got "Screening Clear", and I understand why
- ◯ I got "Screening Clear", but I don't understand why
- ◯ I got "Possible Concussion Detected", and I understand why

- ◯ I got "Possible Concussion Detected", but I don't understand why
- ◯ I did not finish the post-injury test

If you understood why you got the result you did from your post-injury test, briefly explain why:

Your answer

General Test Questions
The following questions relate to the general testing experience.

Did you understand how to hold the phone during tests? *

- ◯ Yes, there was no problem
- ◯ I thought I did, but something went wrong
- ◯ No, I did not understand that part of the instructions
- ◯ No, I skipped the second set of instructions

Were the instructions during the tests helpful? *

- ◯ Yes
- ◯ Only the first set that appeared before the demonstration flashcard
- ◯ Only the second set that appeared after the demonstration flashcard
- ◯ No, I didn't need them

Do you have any additional feedback on the instructions that appeared before and after the demonstration flashcard?

Your answer

While reviewing tests, were you able to use the playback feature to see where you had been looking on the flashcards during the test? *

- ◯ Yes
- ◯ Yes, but the gaze indicator was a little off
- ◯ Kind of, the gaze indicator matched where I was looking horizontally, but not at all vertically
- ◯ Not really, the gaze indicator didn't match where I was looking at all
- ◯ No, there was no gaze indicator when I used the playback feature
- ◯ No, I didn't notice the playback feature

While reviewing tests, were you able to use the playback feature to listen to yourself read the numbers out loud? *

- ◯ Yes
- ◯ No. I tried to, but I could not get it to work
- ◯ No. I did not think of using that feature

Submit Clear form

Never submit passwords through Google Forms.

12.6 Validation Questionnaire Response Summary

Validation Questionnaire

Send

Questions Responses 8 Settings

8 responses

Link to Sheets

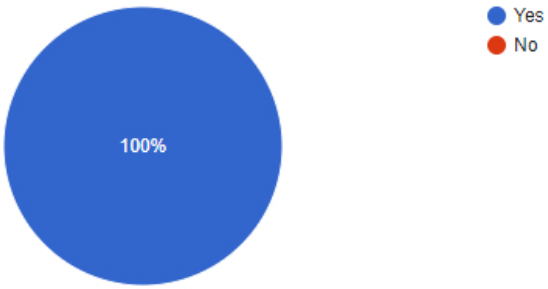
Accepting responses

Summary Question Individual

Were you able to start the baseline test from the main menu?

Copy

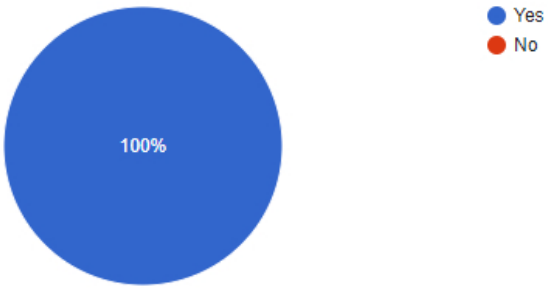
8 responses



Were you able to start the post-injury test from the main menu?

Copy

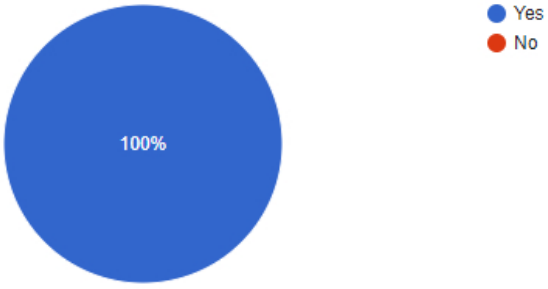
8 responses



Were you able to clear the data for your baseline score?

Copy

8 responses

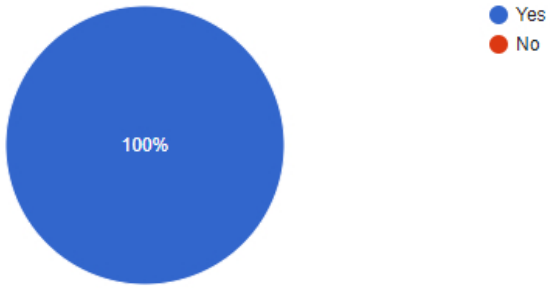


Baseline Test Questions

Were you able to complete the baseline test successfully?

Copy

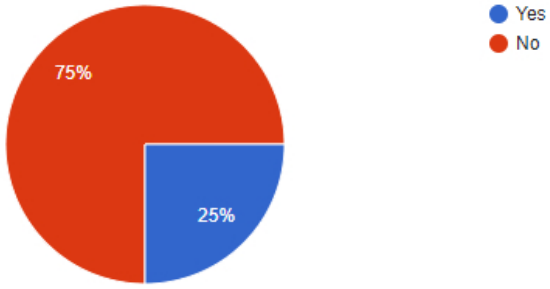
8 responses



Were you ever confused about what to do while attempting to complete the baseline test?

Copy

8 responses



If you were ever confused about what to do while completing the baseline test, please elaborate:

2 responses

I didn't understand that it was a test followed by a recording of that same test. I thought the second was a new test, but then was confused by the "play"button.

The first time when I tried the baseline test I also used the first 'non-graded' card as a test and did it fully. After I had tried the test a couple of times i skipped the non-graded test.

I tried speaking to it both in danish and english and I am unsure if it makes a difference. I ended up doing it in danish since I felt more comfortable speaking numbers out loud in rapid succession in my mothers tounge.

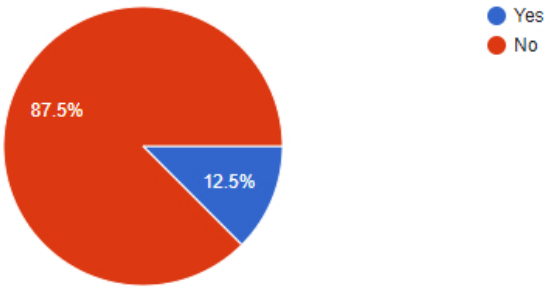
Post-Injury Test Questions

Were you ever confused about what to do while attempting to complete the post-injury

Copy

test?

8 responses



If you were ever confused about what to do while completing the post-injury test, please elaborate:

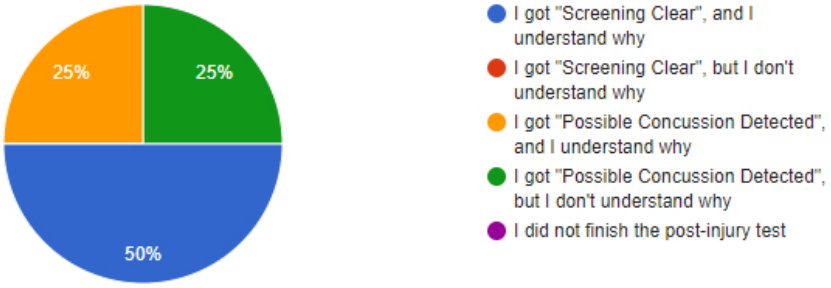
1 response

Only if speaking in a differnce. Else, no.

Do you understand why you got the outcome you did from the post-injury test?

Copy

8 responses



If you understood why you got the result you did from your post-injury test, briefly explain why:

5 responses

I did pretty good I think

My score was slightly higher

Baseline score was very low

I got the score because I don't have a concussion

My understanding is, that I performed better than my baseline and that even though I performed better than my baseline I could still have an ill-fit. If I suspected to have a medical condition I should still seek medical aid since the app cannot for sure conclude that I don't have a medical condition and it could be a false positive that it gave me a Screening Clear.

General Test Questions

Did you understand how to hold the phone during tests?

Copy

8 responses

Were the instructions during the tests helpful?

Copy

8 responses

Do you have any additional feedback on the instructions that appeared before and after the demonstration flashcard?

2 responses

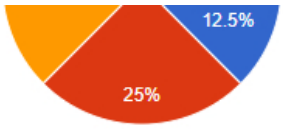
There was a lot of text, perhaps some icons and more inspiring colors would be beneficial. I accidentally, out of habit, dismissed the first instruction page. If it is integral to the test that the participant understands the instructions clearly, perhaps a quiz or test to ensure they have read the instructions is needed.

Maybe add if a specific language is needed, but if isn't relevant which language, then it doesn't matter, since in my experience I would just try both and conclude that I didn't have to do it in a specific language and thus just stuck to whatever I felt comfortable with anyway.

While reviewing tests, were you able to use the playback feature to see where you had been looking on the flashcards during the test?

Copy

8 responses

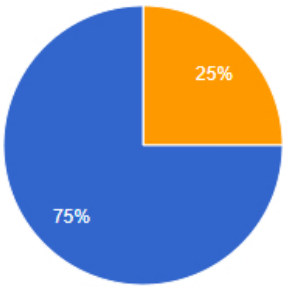


match where I was looking at all
No, there was no gaze indicator when I used the playback feature
No, I didn't notice the playback feature

While reviewing tests, were you able to use the playback feature to listen to yourself read the numbers out loud?

Copy

8 responses



Yes
No. I tried to, but I could not get it to work
No. I did not think of using that feature

12.7 Dennis Interview Questions & Notes

Dennis Interview

Må vi optage interviewet? Det bliver muligvis delt med vores vejleder og censor, men ellers ikke med nogen andre.

Lad være med at forklare lidt om hvem vi er og hvad vi laver. Vent til efter Q6

Eksisterende app: Reflex

De arbejder meget med vergens (konvergens og divergens af synet) og hvordan det påvirker koncentration. Øvelser med at splitte øjnene op.

Brug polariserede briller til test af vergens.
Tjek undersøgelse af amerikansk militær.
Baseline testing er rigtig vigtig.
En god måling: polariserede briller og to billeder med anderledes og fælles symboler man langsomt separerer. Måler vergens og akkumulation.
Hvor meget kan du dreje øjet før du belaster øjet?
Brug prismer til at måle vinkler man kan dreje øjet. Man kalder det biomarkører.
Teknologiske løsninger er ikke så interessante for Dennis, men ville være god til at give brugere noget at sammenligne med.
En måde at måle vergens kræver typisk eksterne værktøjer som prismer
Sakkader er bedst at måle på. Større sammenhæng mellem sakkader og hjernerystelse end pursuit.
Pursuit er ikke godt at kigge på.
King devick test ville være godt til at test sakkader.
Readalyzer laver noget lignende VOMS?
Når man bruger 3D skal man ligge indenfor Panum's area
10-15 grader er en god vinkel at test sakkader
EyeBab: Suite af øjenøvelser.
Vergenstræning (ikke diagnosticering): To billeder på en skærm der flytter fra hinanden. Kryds øjnene for at danne tredje billede.
11,000,000 bits sanseindtryk i sekundet. 10,000,000 af dem er synsindstryk

Q1: Hvordan diagnosticerer du typisk hjernerystelser?
Det gør de ikke. Man slår hovedet, og udviser symptomer. Så har man sikkert hjernerystelse. Ser man skarpt, kaster man op, kraniebrud, osv. Hurtig udtrætning er et klassisk symptom.

Q2: Hvis ikke allerede svaret: Hvilke symptomer skal en patient have for at I kan give en hjernerystelse diagnose?

Q3: Hvis ikke allerede svaret: Udviser personer med hjernerystelser nogen visuelle karakteristika?

Q4: Hvis ikke allerede svaret: Kan du demonstrere hvordan du typisk diagnosticerer hjernerystelser?

Q5: Hvilke udfordringer har I normalt med at diagnosticere hjernerystelser?

Q6: Hvilke IT løsninger bruger I til at udføre jeres arbejde?

Q7: Kan du forestille dig nogen IT løsninger som kan hjælpe dig med at stille hjernerystelse diagnoser?

Q8: Kan du se nogen etiske problemer ift. brugen af vores app? Hvad skal vi være særligt opmærksome på i den forbindelse?
Det er meget svært at reducere en bestemt måling til en diagnose.

12.8 Class Diagram

Activities

Fragments

SeeSo classes

Data classes

Other classes

MainActivity : AppCompatActivity

- concussionApplication: ConcussionApplication
- preferences: SharedPreferences
- hasBaseline: Boolean
- permissions: Array<String>
- permissionRequestCode: Int
- postInjuryTestFragment: FrontPageTestButtonFragment
- baselineTestFragment: FrontPageTestButtonFragment

- checkPermission(): void
- onCameraAndAudioPermissionGranted(Boolean, Boolean): void
- enableTestButtons(): void
- onGazeRecorderInitFail(String): void
- hasPermissions(Array<String>): Boolean
- onGazeRecorderInitFail(String): void
- getTestButtonFragmentBundle(String, String, Int, Boolean, Boolean): void
- updateDebugData(): void

FrontPageTestButtonFragment : Fragment

- titleText: String?
- bodyText: String?
- imageResource: Int?
- isScreening: Boolean?
- isEnabled: Boolean?
- concussionApplication: ConcussionApplication
+ testButton: ImageButton

+ newInstance(String, String, Int, Boolean, Boolean): FrontPageTestButtonFragment
+ enableTestButton(Boolean): void

SeeSoGazeRecorder

- appContext: Context
- initSuccessCallback: () -> Unit
- initFailCallback: (String) -> Unit
- licenseKey: String
- modelName: String
- screenOriginX: Float
- screenOriginY: Float
- flashcardWidth: Int?
- flashcardHeight: Int?
- flashcardXOffset: Int?
- flashcardYOffset: Int?
- gazeTracker: GazeTracker?
- oneEuroFilterManager: OneEuroFilterManager
- gazeData: MutableMap<Int, Pair<Float, Float>>
- timestampOffset: Long
- errorDescriptions: MutableMap<InitializationErrorType, String>
- initializationCallback: InitializationCallback
- gazeCallback: GazeCallback

+ SeeSoGazeRecorder(Context, () -> Unit, (String) -> Unit)
+ getGazeData(): Map<Int, Pair<Float, Float>>
+ startTracking(Int, Int, Int, Int): void
+ stopTracking(): void
- initGaze(): void
- initSuccess(GazeTracker): void
- initFail(InitializationErrorType): void

SeeSo

camp.visual.gazetracker.GazeTracker

camp.visual.gazetracker.callback.InitializationCallback

camp.visual.gazetracker.constant.InitializationErrorType

camp.visual.gazetracker.callback.GazeCallback

camp.visual.gazetracker.filter.OneEuroFilterManager

TestActivity : AppCompatActivity

- getInstance: TestingInstance
- getFlashcardIndex: Int
- isDemonstrationCard: Boolean
- isFirstTestCard: Boolean
- seed: Int
- concussionApplication: ConcussionApplication
- startTime: Long
- fragmentContainerView: FragmentContainerView

- createFullscreenPopup(Int, () -> Unit): void
- startRecording(): void
- stopRecording(): Float

FullscreenPopupFragment : Fragment

- bodyTextId: Int
- onClickCallback: () -> Unit
- rootLayout: FragmentFullscreenPopupBinding?

+ FullscreenPopupFragment(Int, () -> Unit)
- onClicked() : void

FlashcardConfiguration

+ line: LineConfiguration
+ name: String
+ rows: Int
+ headerWeight: Float
+ footerWeight: Float

+ FlashcardConfiguration(LineConfiguration, String, Int, Float, Float)

FlashcardFragment : Fragment

- configuration: FlashcardConfiguration?
- seed: Int?
- randomGenerator: Random?
- clickListener: OnClickListener?
- concussionApplication: ConcussionApplication
- getInstance: TestingInstance
- flashcardData: FlashcardData
numberData: MutableMap<Int, FlashcardNumberData>
numberUILookup: MutableMap<View, Int>
numberIndexLookup: MutableMap<Int, View>
- flashcardIndex: Int
- allConfigs: Array<FlashcardConfiguration>

+ newInstance(Int, Int): FlashcardFragment
- createNumberRows(Int, LinearLayout, Int): void
- createNewTextView(Int): TextView
- createFooterText(LinearLayout): void
- addFooterText(LinearLayout): void
- addFillingSpace(ViewGroup): void
- addSpace(ViewGroup, Float): void
- addDiagonalLine(ViewGroup, Float): void
- addLine(ViewGroup, Float): Line

LineConfiguration

+ showLines: Boolean
+ showArrows: Boolean
+ showDiagonal: Boolean

+ LineConfiguration(Boolean,Boolean, Boolean)

Line : View

+ showArrow: Boolean
+ reverseArrowDirection: Boolean
+ startXOffset: Float
+ startYOffset: Float
+ endXOffset: Float
+ endYOffset: Float
+ arrowOffset: Float
+ isDiagonal: Boolean
- linePaint: Paint()
- debugPaint: Paint()
- margin: Float
- arrowWidth: Float
- arrowHeight: Float
- arrowHalfHeight: Float
- diagonalHorizontalOffset: Float

+ drawDiagonal(Canvas): void
+ drawMidline(Canvas): void
+ drawArrow(Canvas, Float, Float, Float, Float): void

BaselineTempDataCache

+ firstAttempt: Float
+ secondAttempt: Float
+ min: Float (get)

+ BaselineTempDataCache(Float, Float)

ConcussionApplication : Application

+ gazeRecorder: SeeSoGazeRecorder?
+ audioRecorder: MediaRecorder
+ audioFilePath: String
+ isGazeRecorder: Boolean (get)
+ session: TestingSession? (get)
+ isScreening: Boolean? (get)

+ getBaselineTempData(): BaseLineTempDataCache
+ getInstance(): TestingInstance
+ getPreferences(Context): SharedPreferences
+ initializeNewSession(Boolean): void
+ clearInstance(): void
+ calculateFinalScore(): Float
+ calculateFlashcardScore(FlashcardData): Float
+ createNewInstance(): TestingInstance
+ initGazeRecorder(() -> Unit, (String) -> Unit): void
+ initAudioRecorder(): void

ReviewFlashcardFragment: FlashcardFragment

ReviewFlashcardActivity : AppCompatActivity

- seed: Int
- getFlashcardIndex: Int
- getTimeElapsed: Float
- concussionApplication: ConcussionApplication
- getFlashcardData: FlashCardData
- isFinalFlashcard: Boolean
- selectedIndex: Int
- playbackControlContainer: LinearLayout
- playPauseButton: ImageButton
- playbackBar: Seekbar
- currentTimeText: TextView
- maxTimeText: TextView
- gazeIndicatorView: ImageView
- flashcardContainer: ConstraintLayout

ReviewFlashcardViewModel : AndroidViewModel

- audioGazePlayer: AudioGazePlayer?
- _uiState: MutableStateFlow<SeeSoUiState>
+ uiState: StateFlow<SeeSoUiState>

+ ReviewFlashcardViewModel(Application)
+ initGazePlayer(): void
+ togglePlay(): void
+ blockPlayback(): void
+ unblockPlayback(): void
+ stopPlayback(): void
+ seekTo(Int): void
+ updateForProgress(Int): void
+ getTimestampText(Int): String

SeeSoUiState

+ isPlaying: Boolean
+ playProgress: Int
+ maxProgress: Int?
+ indicatorX: Float?
+ indicatorY: Float?

TestingSession

+ instance: TestingInstance
+ baselineTempDataCache: BaselineTempDataCache?

+ TestingSession(TestingInstance, BaselineTempDataCache?)

TestingInstance

+ flashcardData: MutableMap<Int, FlashcardData>

+ TestingInstance(MutableMap<Int, FlashcardData>)
+ createFlashcardData(Int): void
+ getFlashcardData(Int): FlashcardData

FlashcardData

+ numbers: MutableMap<Int, FlashcardNumberData>
+ elapsedTime: Float

+ FlashcardData(MutableMap<Int, FlashcardNumberData>, Float)

FlashcardNumberData

+ index: Int
+ expectedValue: Int
+ actualValue: Int

+ FlashCardNumberData(Int, Int, Int)

AudioGazePlayer

- viewModel: ReviewFlashcardViewModel
- gazeData: Map<Int, Pair<Float, Float>>?
- audioPlayer: MediaPlayer
- updateDelay: Int
- maxProgress: Int
- playBack: Job?
- progress: Int
- isBlocked: Boolean
- resumePlaybackOnUnblock: Boolean

+ AudioGazePlayer(
ReviewFlashcardViewModel,
Map<Int,Pair<Float, Float>>?,
MediaPlayer)
+ getCoords(Int): Pair<Float, Float>
+ seekTo(Int): void
+ startPlayback(): void
+ pausePlayback(): void
+ blockPlayback(): void
+ unblockPlayback(): void
+ stopPlayback(): void
- getValidTimestamp(Int): Int

TestEndscreenActivity : AppCompatActivity

- concussionApplication: ConcussionApplication
- isScreening: Boolean
- isBaseline: Boolean
- baselineData: BaselineTempDataCache
- isFirstAttempt: Boolean

- handleBaselineData(): void
- addTestTypeFragment(): void
- addScreeningFragment(): void
- addBaselineFragment(): void

EndscreenFragment_Baseline : Fragment

- concussionApplication : ConcussionApplication
- preferences: SharedPreferences
- baselineData: BaselineTempDataCache
- isFirstAttempt: Boolean
- savedBaselineScore: Float

- setDescription(TextView): void
- setFinalScoreText(TextView): void
- setButtonText(Button): void
- getSessionBestScore(): Float

EndscreenFragment_Screening : Fragment

- concussionApplication : ConcussionApplication
- preferences: SharedPreferences
- baselineData: BaselineTempDataCache
- isFirstAttempt: Boolean
- isConcussionDetected: Boolean

- getDisclaimerText(): String
- getHeaderColor(): Int
- getHeaderText(): String
- getIcon(): Int