Jarno Sokka

Development of an Application for Surveying Rail Traffic Clearances

Metropolia University of Applied Sciences

Bachelor of Engineering

Information and Communication Technology

Bachelor’s Thesis

21 August 2019

Author Title Number of Pages Date	Jarno Sokka Development of an Application for Surveying Rail Traffic Clear-ances 47 pages + 1 appendix 21 August 2019
Degree	Bachelor of Engineering
Degree Programme	Information and Communication Technology
Professional Major	Software Engineering
Instructors	Janne Ahonen, Head of Department, Metropolia Pasi Kräknas, Head of Training, NRC Group Finland

This thesis describes the development of the ATU App, an augmented reality-based measuring tool for mobile devices. The prototype version of the application was developed in the Innovation Project of fall 2018 for NRC Group Finland.

The purpose of the application is to visualize the clearance requirements of rail vehicles using augmented reality. The visualization is used to determine if the requirements are met or if an installation is intruding into the clearance space. A reliably accurate visualization-based check can serve as a substitute for current measuring methods in many cases, thus saving work and time.

The thesis describes the requirements of the application, the features developed to fulfill the requirements, the creation of assets and tools needed to deploy AR content and the field testing of the application. Additionally, some of the ideas and features that were left outside the scope of the project are discussed briefly.

The project was focused on developing the Android version of the application. The features were implemented in a platform-agnostic manner to make porting the application to other platforms require as little additional work as possible.

Keywords	Unity, Vuforia, Augmented Reality

Tekijä Otsikko Sivuja Päiväys	Jarno Sokka Rataliikenteen turvaulottumien tarkastelusovelluksen kehitys 47 sivua + 1 liite 21 elokuuta 2019
Tutkinto	Insinööri(AMK)
Tutkintoohjelma	Tieto- ja Viestintätekniikan Insinööri (AMK)
Ammatillinen Pääaine	Software Engineering
Ohjaajat	Janne Ahonen, Tutkintovastaava, Metropolia Pasi Kräknas, Koulutuspäällikkö, NRC Group Finland

Tässä opinnäytetyössä kehitettiin lisätyn todellisuuden sovellus älypuhelimille rataliikenteen turvaulottumien tarkasteluun. Työn toimeksiantaja on NRC Group Finland. Sovellus perustuu Metropolian Syksyn 2018 Innovaatioprojektissa kehitettyyn prototyyppiin.

Sovellus käyttää lisättyä todellisuutta näyttämään käyttäjälle turvallisen rataliikenteen edellyttämiä avoimen tilan ulottumia. Sijoitetusta mallista käyttäjä voi tarkistaa ulottuman esteettömyyden laitteensa näytöltä. Lisätyn todellisuuden sijoittamiseen perustuva tarkastelu voi korvata perinteisiä mittalaitteilla tehtäviä tarkastuksia, jos se saavuttaa riittävän tarkkuuden. Älypuhelimella tehtävä tarkistus säästäisi aikaa ja vaivaa.

Opinnäytetyössä käydään läpi ratkaistavan ongelman tekniset vaatimukset, sovellukseen toteutetut ominaisuudet, lisätyn todellisuuden edellyttämien 3D-mallien ja maalilaitteiden kehittäminen sekä sovelluksen kenttätestaus. Lisäksi työ käsittelee lyhyesti projektin skaalan ulkopuolelle jääneitä ideoita ja ominaisuuksia.

Projekti keskittyi sovelluksen Android version kehittämiseen. Sovelluksen ominaisuudet pyrittiin toteuttamaan alustasta riippumattomilla tavoilla, jotta sen vienti muihin ympäristöihin olisi mahdollisimman vaivatonta.

Keywords	Unity, Vuforia, Augmented Reality

Contents

List of Abbreviations

ATU	“Aukean Tilan Ulottuvuus”, an area of open space required for the vehicle to pass safely through a section of tracks.
AR	Augmented Reality, the technique of overlaying virtual content in images or video of the real world.
IDE	Integrated Development Environment, a program that provides tools for software development such as a code editor, debugger, build tools, etc. Supported languages vary by IDE.
GIMP	Gnu Image Manipulation Program, a raster graphics image editing program.
MFA	Multi-factor authentication
MSAL	Microsoft Authentication Library
RSU	“Ratatyön Suojaulottuma”, an area around a rail track that is considered dangerous, with associated safety regulations.

1 Introduction

This thesis details the development of the ATU App, an AR-based safety clearance surveying application for mobile platforms. The thesis briefly covers the initial requirements and prototyping of the application in the Innovation Project course of fall 2018, but the focus of the thesis is on the development of the second, full version of the application.

The application is developed for NRC Group Finland. NRC Group Finland is an infrastructure construction company. It is the largest railway construction company operating in Finland.

The application is intended to serve as a substitute for current methods of measuring ATU. ATU refers to “Aukean Tilan ulottuvuus”, an area of open space required for the vehicle to pass safely through a section of tracks. The situations where it can serve as a substitute depend on the accuracy and reliability the application is able to attain.

The thesis primarily covers features of the application, their implementation and theory as it affected their development. The technologies used and features implemented are primarily approached from the perspective of how they contribute to the application. How they could be adapted for similar projects is considered in some cases.

2 Origins & Requirements

This chapter covers the origins of the application in the Innovation Project, the requirements it must meet and the technical specifications from which those requirements are derived.

2.1 The Innovation Project Task

The task offered to the teams in the Innovation Project was to develop a practical portable solution that would be able to replace the current tools and methods for checking the

ATU for at least some measurement situations. Ideally, the application would be dependable enough to replace all but the final measurements made for documentation purposes.

The workflow in place at the time required the measurement team to check the work of the installation team using specialized equipment such as tachyometers. The measurements would be used to calculate ATU placement, which in turn determined whether the installation needed to be adjusted. As the work would often happen at night, a follow-up measurement was required during daytime to check the earlier measurement was correct and to measure the values for documenting the work.

The overall solution that both Innovation Project teams working on this task arrived at was an augmented reality application for Android. The main advantages the application can provide are speed and convenience. The ATU is displayed immediately and responds to user input. The application runs on a smartphone and the only additional equipment it needs is a portable target with which to place and position the AR content.

2.2 Original Development

The first version of the application was made as part of the Innovation Project course during fall semester 2018. That version was a ‘proof of concept’, focused on implementing the core features of the application to a degree that allows them to be demonstrated. That version included

- An accurate and adaptable model of the Finnish train ATU.
- A simple color palette of four colors options for the ATU.
- Toggleable transparency and outline-only modes for the ATU.
- Four measuring grids of varying grain, in a cyclic toggle.
- The ability to take a screenshot with the ATU settings automatically included.

The prototype also included an auto-setup component. The component would automatically configure a properly structured ATU model to work with the application. All that was required was adding the component to the root object of the model. Its purpose was to make it simple to swap out the original model so that a non-technical worker could solve the main licensing issue with the prototype.

At the start of this thesis project, it was decided to rebuild the application from scratch. The “rapid prototyping” style in which the first version was developed would not have made for a good foundation to build on. Recreating the application also ensured there would be no asset ownership issues. The asset with the most reuse value, the ATU model, was unavailable for use in commercial applications due to the licensing of the software used to create it.

2.3 ATU Specifications

The application includes two different ATU models. The application is in theory capable of accommodating an arbitrarily large selection of different ATU models. In practice the two-directional cycling selection as well as memory and processing power limits of mobile devices would eventually limit the selection.

All rail vehicles consist of one or more cars, each with a rigid body resting on two sets of wheels. This means that in turns each car behaves similarly to a short chord on a circle, causing the outer corners and inner mid-section of the car to protrude further from the track than on a straight rail. Figure 1 illustrates how this impacts the effective width of the ATU.

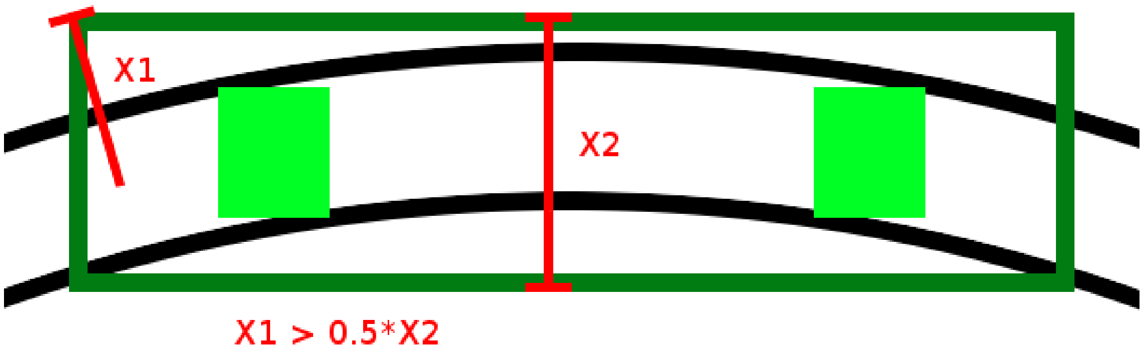


Figure 1 - ATU width scaling

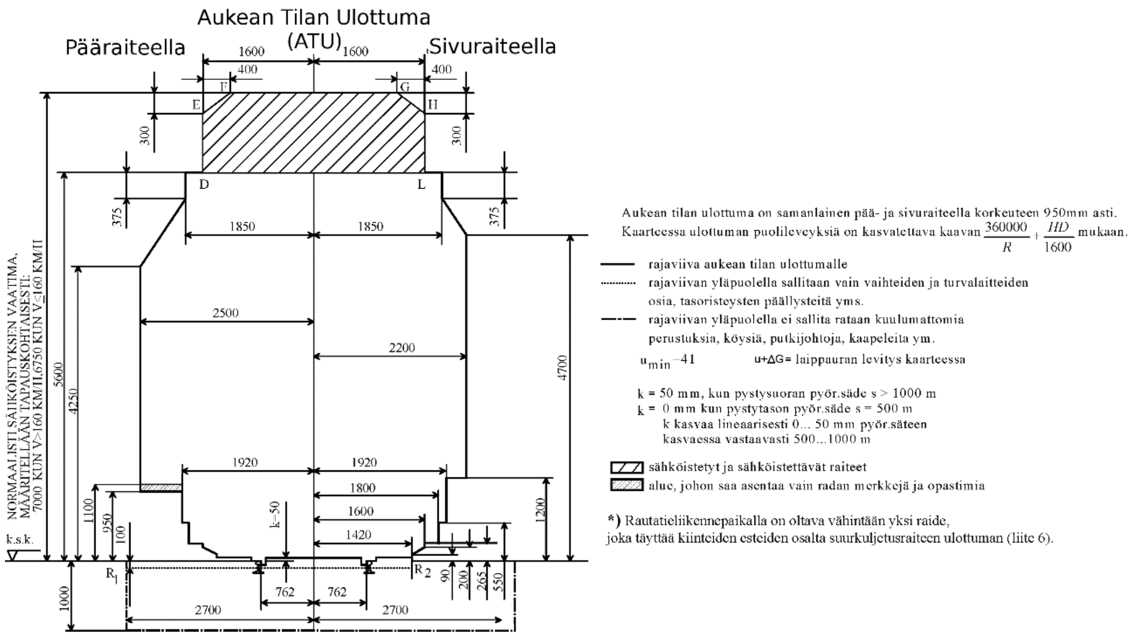
As shown in Figure 1, the bounds of the train car body are closer or further from the center of the rail depending on the direction and sharpness of the turn. As the entire train car must be able to pass through a given segment of rail safely, the scaled width of the ATU is based on the highest effective width of the body.

To model this behavior of the cars in turns the ATU models are scaled based off the radius of the turn and the cross-section tilt in turns. The degree of width scaling caused by tilt depends on the height at which the potential obstacle is. The exact formula varies by ATU.

2.3.1 Finnish Train ATU

The train ATU is defined in RATO part 2 [1 appendix 2]. It is a public document produced by the Finnish Transport Infrastructure Agency. The document specifies the dimensions of the ATU under the various conditions of the Finnish railway network.

Figure 2 displays the dimensions and the conditions under which they apply.



Aukean tilan ulottuma on samanlainen pää- ja sivuraiteella korkeuteen 950mm asti.
Kaarteessa ulottuman puolileveyksiä on kasvatettava kaavan $\frac{360000}{R} + \frac{HD}{1600}$ mukaan.

——	rajaviiva aukean tilan ulottumalle
········	rajaviivan yläpuolella sallitaan vain vaihteiden ja turvalaitteiden osia, tasoristeysten päällysteitä yms.
—·—	rajaviivan yläpuolella ei sallita rataan kuulumattomia perustuksia, köysiä, putkijohtoja, kaapeleita ym.

u_{min} –41 $u+\Delta G$= laippauran levitys kaarteessa

k = 50 mm, kun pystysuoran pyör.säde s > 1000 m
k = 0 mm kun pystytason pyör.säde s = 500 m
k kasvaa lineaarisesti 0... 50 mm pyör.säteen
kasvaessa vastaavasti 500...1000 m

▨	sähköistetyt ja sähköistettävät raiteet
▨	alue, johon saa asentaa vain radan merkkejä ja opastimia

*) Rautatieliikennepaikalla on oltava vähintään yksi raide,
joka täyttää kiinteiden esteiden osalta suurkuljetusraiteen ulottuman (liite 6).

Figure 2 - The Finnish train ATU specification [1 appendix 2]

The conditions consist of

- Rail Type – Main Rail or Side Rail.
- Electrification – High Speed, Low Speed or Not Electrified.

for a total of 6 different states. Width scaling for the ATU applies evenly to both sides. The specification only shows half of each of Main Rail and Side Rail specification, as the full ATU is symmetrical. How this specification translates to the 3D model used by the application is covered in Chapter 4.3

2.3.2 Tampere Tram ATU

The tram ATU is defined in the design documents for the Tampere Tramway. The documents are confidential, so Figure 3 instead shows the model derived from the specifications.

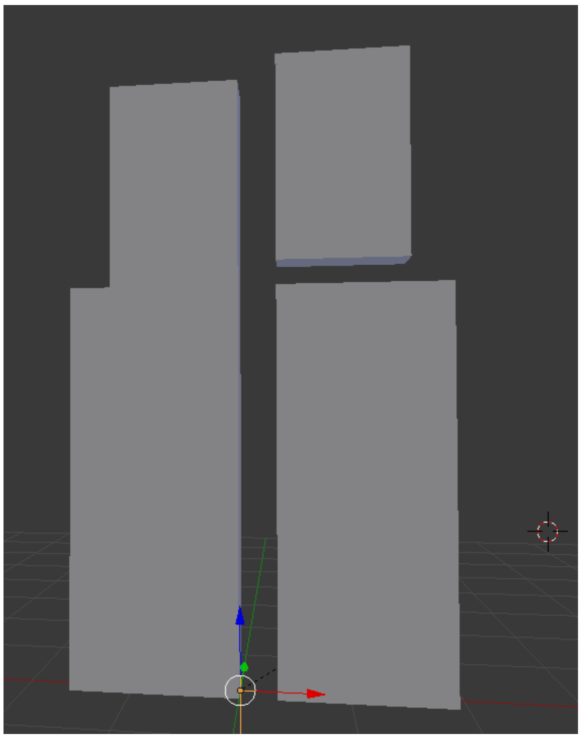


Figure 3 - Tampere tram ATU 3D model

The tram ATU is simpler than the train ATU, as the conditions under which the tram operates are more uniform. The creation of the tram ATU model is also covered in Chapter 4.3

Note that while the train ATU works for the entire rail network of Finland, this tram ATU is only for Tampere as there is no national standardization for trams. Some lines need wider, higher-capacity passenger cars to accommodate the volume of passengers traveling on that line. The ATU reflects the tram car which in turn reflects the needs of local commuters.

3 Technologies

This chapter describes the technologies and tools used to create the application. How to install these tools is not covered, as it does not take long for that kind of information to become outdated and up-to-date information is typically available if the tool itself is available.

3.1 Unity Engine

The visualization of the ATU requires a 3D engine as a platform in order to have the degree of responsiveness and adaptability expected of the application. The Unity Engine is ideal for this as it already supports the relevant mobile platforms and supports Vuforia. It is also familiar from past projects.

Unity Engine offers Visual Studio community edition as its default IDE but the licensing for it does not permit working on enterprise applications. The sole free C# IDE available was Visual Studio Code.

In order to build for Android applications, Unity needs both the Java Development Kit (JDK) and the Android SDK Tools. The application also needs a bundle identifier. In this case it is com.NRCGroupFinland.ATUApp. Bundle identifier is the unique identifier for the application on Android. Only one application on the Google Play Store can have a given identifier. When installing from an .apk, it will install over an existing installation with the same bundle identifier if one exists.

Building for iOS requires either running Unity on a Mac or using a specialized Unity add-don. A physical Mac was not available. As the user base of the case company is mostly using android devices, setting up a second Unity installation on a virtual Mac was deemed unnecessary. That such a venture had high potential to become a serious time-sink was a factor in the decision.

3.2 Vuforia

For the visualization of the ATU to be useful it needs a means of relating itself to the real world. Vuforia provides the augmented reality component that fulfills this need. Vuforia has multiple options for deploying AR content. Deployed content is effectively attached to a location in the world, as tracked in a model of the scene by the application. Vuforia primarily deploys content in response to detected targets. Targetless methods that deploy based off the user’s location, such as Ground Plane, are unsuited for this application as they have no means of providing precise placement.

The simplest type of target is the Image Target, which in theory can be any arbitrary image the developer chooses to use. In practice, how good an Image Target a given image is can vary a lot. Multiple Image Targets grouped together form a Multi-Target, typically a box. Object Targets are effectively the next step in Multi-Targets. Where Multi-Targets have a defined set of faces, an Object Target has a collection of pictures from different angles. Targets will be discussed in greater detail in Chapter 4.4 Vuforia Development.

Augmented Reality is typically marketed with its interactability and the merging of the virtual with the real. This can create unrealistic expectations for the technology, particularly for lay users. The basic structure of the augmentation in action in Unity consists of three parts

- The viewpoint camera.
- The augmentation object(s).
- The input of the device camera.

This arrangement can be compared to a movie theater. The viewpoint camera is the moviegoer, the input of the device camera the movie and the augmentations take place between the moviegoer and the movie. The view the application has of the real world is as flat as the movie screen. The augmentations cannot interweave with objects in it and instead will always appear to be in front of what is shown. In order to present true mixed reality, the application would need to accurately and quickly calculate the shapes it needs to omit from the augmentations based on the camera input feed. Such a feat is well beyond what mobile devices can do for the foreseeable future. While fully mixed reality would be useful, the application can work within the limitations of the technology.

Vuforia provides four types of licenses. The development license is free and allows you to fully finish your product before making a financial commitment. In order to deploy it, either as an internal application for an organization or for sale, you must purchase a license. Until recently Vuforia provided single payment licenses for simple uses for small businesses, but currently only subscription licenses are available. The pricing is based on revenue, businesses with revenue less than 10 million $/year can choose between two fixed price licenses while larger businesses will have to negotiate their price.

3.3 3D Modelling - Blender

The application needs an accurate and well-organized model of every type of clearance it needs to be able to visualize. The basic shape of the train ATU model is straightforward. The various states and kinds of scaling the model needs to able to handle place some requirements on how the model is structured. The final model is a large and relatively complex hierarchy of mostly basic shapes. The tram ATU is considerably simpler, as it has to deal with just one very specific operating environment.

There are many different 3D modelling programs on the market. The ideal program that meets and slightly exceeds their requirements for minimal costs, both monetary and time spent learning. Free software is a solid starting point, since very little is committed and thus lost if the software fails to fulfill the needs of the project.

Blender is free and Unity can natively import blender files. This allows you to edit the .blend file directly in the project folder, save and have your changes load in to the project without any extra steps required. The tools and features were more than adequate for the needs of the project. The creation of the 3D models for the project is detailed in Chapter 4.3

3.4 Miscellaneous Assets – GIMP and Inkscape

Most applications need at least some custom graphics assets. In this case the UI needed some icons and the ATU model needed a measuring grid. The program used for these needs to support transparency in images. Gnu Image Manipulation Program is free and adequate for most of the tasks. What it handles poorly is the creation of measuring grids. A grid that is both large enough and sharp enough results in an inconveniently large file in a raster graphics format. For that reason, the grids are best done in a vector graphics program. Inkscape is free and can handle the task of creating the grids.

4 The Application – Features & Development

This chapter describes the features of the application and their implementation on a general level. UI design comes first as it affects the entire application. Next are the contents of the side and top bar in a top-down left to right order. Then the creation of the 3D assets used by the application and finally the technology used to deploy them.

4.1 UI and Visual Design

The native user interface implementation of Unity is built on the Canvas element. All UI elements must be children of a Canvas, otherwise they simply are not displayed. UI elements are drawn in the order they appear from the top down [2]. Elements later in the draw order will cover elements earlier in the order if they overlap. This includes active elements with no visual component or small visual components relative to their size. Ordering of elements within a canvas is how the sidebar menu buttons are available in the menus opened from the side bar but covered by the menus opened from the top bar.

Canvases determine their draw order by a Sort Order variable [3]. A higher Sort Order means being drawn later and covering Canvases with lower Sort Order. In this application, the canvas for Access Control has a higher Sort Order than the main Application. This way, in the unintended event both were active, Access Control would effectively block access to the main Application.

The application is built around a ‘Main View’, the view the user sees immediately after successfully signing in. The Main View is where the user views the ATU model once deployed. The UI in the Main View is designed to minimize the amount of screen space used up while providing immediate access to useful features. As shown in Figure 4, the icons in the view are grouped together at the top and right of the screen with two exceptions.

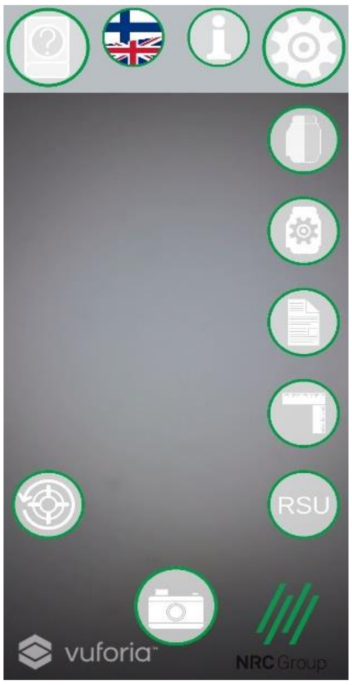


Figure 4 - The Main View of the Application, Unlabeled and Labeled

The responsibilities of the top Icon bar can be summarized as providing information and configuring the application. It includes the in-app user manual, language settings, toggleable description labels for the icons and the main settings menu. The side Icon bar covers the practical use of the application. It includes controls for the ATU models and additional tools as well as a notepad. The two isolated Icons are for the documentation camera and for resetting the targeting of Vuforia. The division into these groups and the deeper menu structure is discussed in more detail in 4.2, as this subsection is about presentation.

The Main View uses Icon-based buttons as one of its ways to conserve screen space. On a small screen buttons with text labels would have to make sacrifices in clarity of meaning, legibility or both. Icons preserve both at the cost of having at best an informal agreement on their exact meaning. There is also some added challenge from the unique nature of the features the application must communicate in Icon form. The solution is to offer toggleable descriptor labels for the Icons. These labels can be toggled on and off from the icon “(i)”, widely used to stand for “information”.

UI Icons consist of 3 Image components, from the bottom up

- The Fill, a simple partially transparent circle
- The Outline, a solid circular outline for the Fill
- The Icon, an image communicating the function of the button

Figure 5 shows an example of how a complete Icon stacks up.

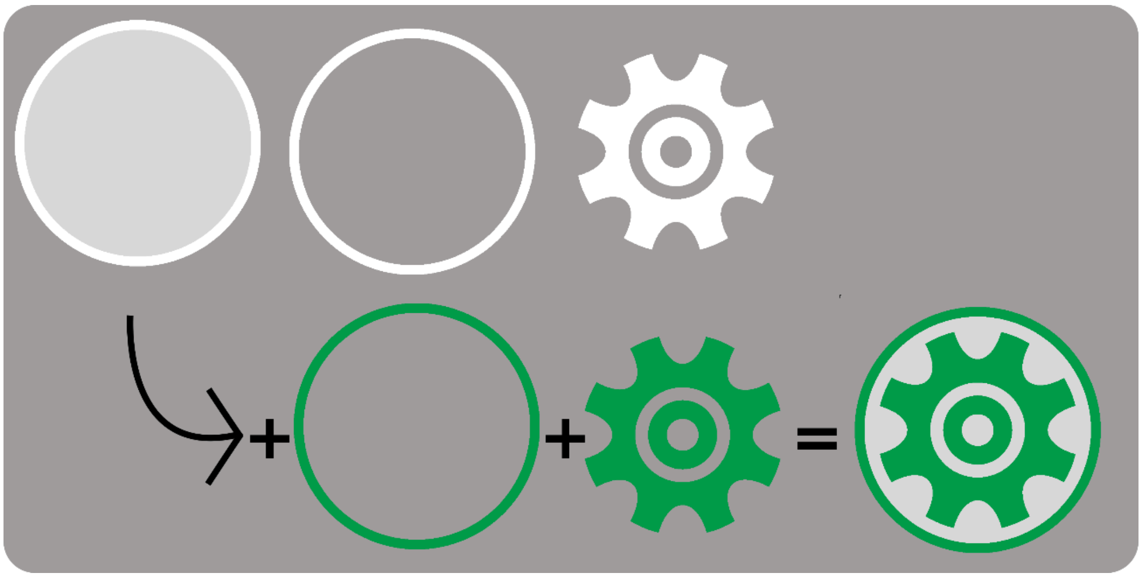


Figure 5 - UI Icon Composition

As shown in Figure 5, the base color of each of the components in their source image is white. The components get their final color from the Material assigned to the Image. Each component holds a reference to the Material for its type of UI element. Changing the color of that Material will change it for all elements referencing the Material. This allows easy implementation of alternate UI color palettes, including an option for a fully custom palette. Buttons that correspond to a menu or other function with a “held” or “ongoing” state will switch their Material reference to signal changes in state. Icons and labeled buttons use the same Material for the fill and the body. While the color for an active labeled button is typically the same as the color of the rim of the icon button, the fill itself is partially transparent and the resulting color distinct. Labeled buttons also switch their text color for better contrast with the new fill while icons do not change their color.

The application has a total of four different color palettes for the UI. Three are presets and the last is customizable. Presets consist of the “Default”, “Dark” and “Autumnal” themes. Themes can be changed from the Color Scheme Settings menu. The change is automatically saved to Display Settings. The color switch is immediate, allowing a limited preview from the menu itself, as seen in Figure 6.

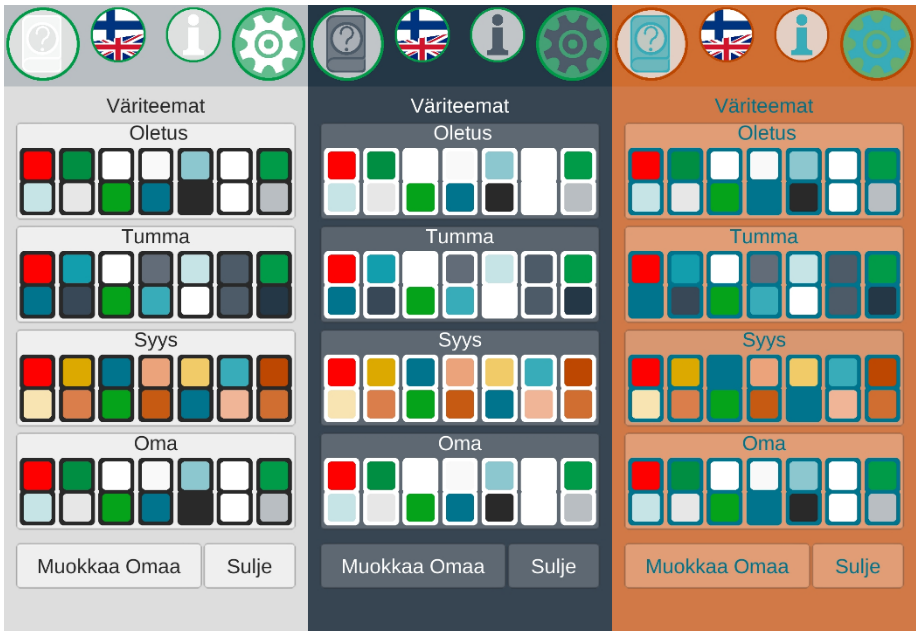


Figure 6 – The Color Scheme Presets

While not all of the colors are in use in the menu, enough of them are to give a good preview of the theme. The preset themes mostly use colors and their variations outlined in the case company style guide [4], shown in Figure 7.

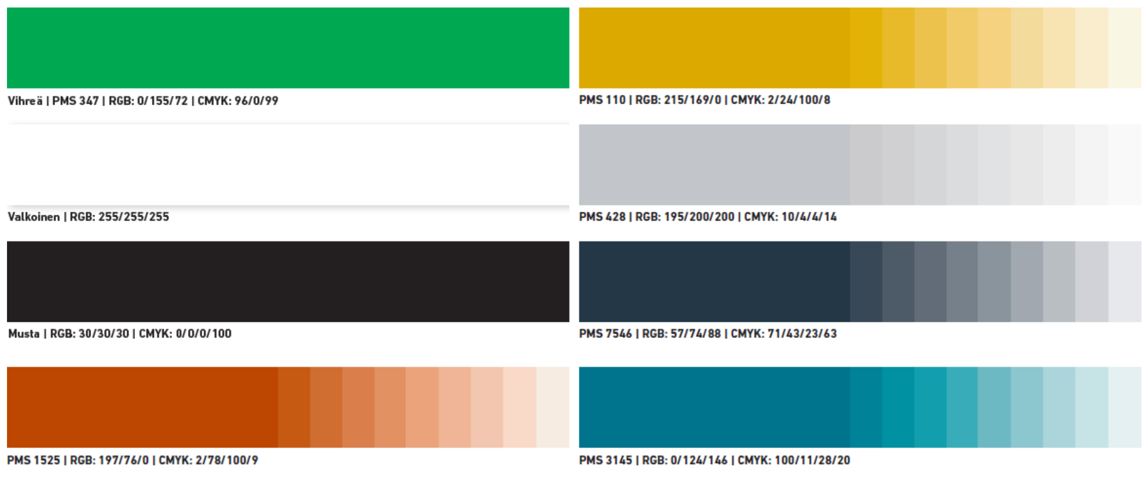


Figure 7 - NRC Visual Guidelines Color Selection [4]

The Default theme is primarily white and light greys with the green used for emphasis. Shades of the supporting blue are used for labels and sliders. The Dark theme takes its main colors from the dark grey and blue but keeps red and greens from the Default for alert colors and UI Icon boundaries. The Autumnal theme is composed from the shades of red and yellow with blue shades for contrast. The final theme, Custom, is entirely up to the end user. Custom has no limitations and makes no attempt to enforce readability. Its role is to accommodate special use conditions and user-specific eyesight issues. The editing menu for the Custom theme offers a button for resetting it to match the Default theme. The Default theme is also the starting point of the custom theme on default settings, so the user could also delete their saved color settings to reset the theme.

The UI themes are implemented in a way that makes changing the colors of the presets easy and intuitive. When the application starts, it loads all settings, including settings for UI themes. These consist of the colors of the Custom theme and which theme is in use. The theme that is in use is then applied to the UI, the same as if it had been switched into. When switching themes, the colors from the theme palette are copied into the shared UI materials. Simply changing the colors displayed on those preset palettes will be enough to alter the theme.

4.2 Application Structure

How the parent-child relationships of objects work in a Unity Scene is an important part of how the application is structured. The position, rotation and scale of a child object is always in relation to its parent. Additionally, if the parent object is set to be active or inactive in the scene, the child object will inherit that state. This continues down the hierarchy. An inactive object makes all the objects descended from it inactive, whether they would normally be so or not. If an object is inactive, then any scripts attached to it do not call their Update method. In effect, the scripts on an inactive object are frozen. In games this can be used to recycle projectiles by deactivating and repositioning the object for the next use as a “new” shot.

This allows us to activate, deactivate, move and scale objects as groups. Among the uses for this application are opening and closing menus, scaling the ATU model to match the real-world scale of targets and making utilities such as measuring grids stay with the ATU model.

4.2.1 Overall Structure

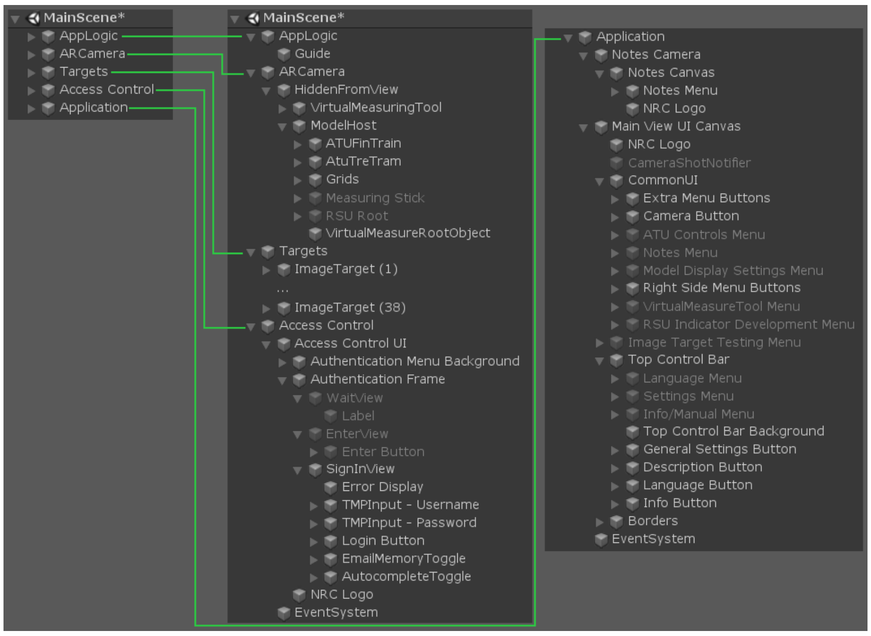
The application consists of a single Scene. In Unity game development context, a Scene is typically a level or the main menu. In a more general sense, a Scene is a collection of assets that consistently work together. For example, the main menu of a game is often a Scene on its own because there is no guarantee that you will always go to a specific level from it, therefore including any given level in the same Scene is generally a waste of resources.

Another way of looking at Scene boundaries is user expectations. The user expects to be able to move freely and seamlessly around things that are related. A Scene change and its associated loading time interrupting the user every time they move between two similar parts of a logical whole is unacceptable. Continuing the example, while most of the menus included under main menu are not used often, users expect to be able to seamlessly navigate them. If the menus were split between Scenes, the resource savings from it would be more than negated by the loss in user experience. Ultimately, good structure is designed to enable a smooth and intuitive flow for the user.

The application is compact enough to work as a single Scene. The top-level objects in the Scene are

- AppLogic
- ARCamera
- Targets
- Access Control
- Application

Figure 8 shows the structure of the application.



```
MainScene*
  AppLogic
  ARCamera
  Targets
  Access Control
  Application
```

```
MainScene*
  AppLogic
    Guide
  ARCamera
    HiddenFromView
      VirtualMeasuringTool
      ModelHost
        ATUFinTrain
        AtuTreTram
        Grids
        Measuring Stick
        RSU Root
        VirtualMeasureRootObject
  Targets
    ImageTarget (1)
    ...
    ImageTarget (38)
  Access Control
    Access Control UI
      Authentication Menu Background
      Authentication Frame
        WaitView
          Label
        EnterView
          Enter Button
        SignInView
          Error Display
          TMPInput - Username
          TMPInput - Password
          Login Button
          EmailMemoryToggle
          AutocompleteToggle
      NRC Logo
    EventSystem
```

```
Application
  Notes Camera
    Notes Canvas
      Notes Menu
      NRC Logo
  Main View UI Canvas
    NRC Logo
    CameraShotNotifier
    CommonUI
      Extra Menu Buttons
      Camera Button
      ATU Controls Menu
      Notes Menu
      Model Display Settings Menu
      Right Side Menu Buttons
      VirtualMeasureTool Menu
      RSU Indicator Development Menu
    Image Target Testing Menu
    Top Control Bar
      Language Menu
      Settings Menu
      Info/Manual Menu
      Top Control Bar Background
      General Settings Button
      Description Button
      Language Button
      Info Button
    Borders
  EventSystem
```

Figure 8 - Scene Structure Expanded

The role of “Targets” is to keep editor hierarchy clean and easy to navigate. Without it there would be 38 additional objects cluttering the window. AppLogic hosts many of the upper level Manager-classes for the features and functions of the application. ARCamera is the main camera for the scene. It contains elements under HiddenFromView that have a visible physical model that is not intended to show up on screen randomly. This includes the templates for the measures deployed with the Virtual Measuring Tool as well as ModelHost. ModelHost is how the application can use a single shared set of models for all targets. It is described in more detail in Chapter 4.4. Access Control has a relatively narrow but important role in hosting the Canvas for the application sign-in process. As an “outer shell” of the application it has no other functionality.

The “Application” object is where the majority of the work is done. The main canvas is what the user spends most of their time in. Its contents are arranged such that the top bar menus and their contents will draw over the rest of the application, as suits their role as ‘main’ menus intended to be available from anywhere in the application. Both main and side bar menus are organized to always draw their controls over the content they serve, so the same navigation UI is used in all menus instead of duplicated in each of them. “Application” also contains a secondary camera with a dedicated canvas for the documentation function. This is detailed in 4.2.3.

4.2.2 ATU Interaction

Interacting with ATU model is split into two categories - visualization and control. Both are accessed from the sidebar. ATU visualization is the top button and ATU control is the one right below it. The visualization menu is divided into two sections, one section being essentially a submenu of the other section, as shown in Figure 9.

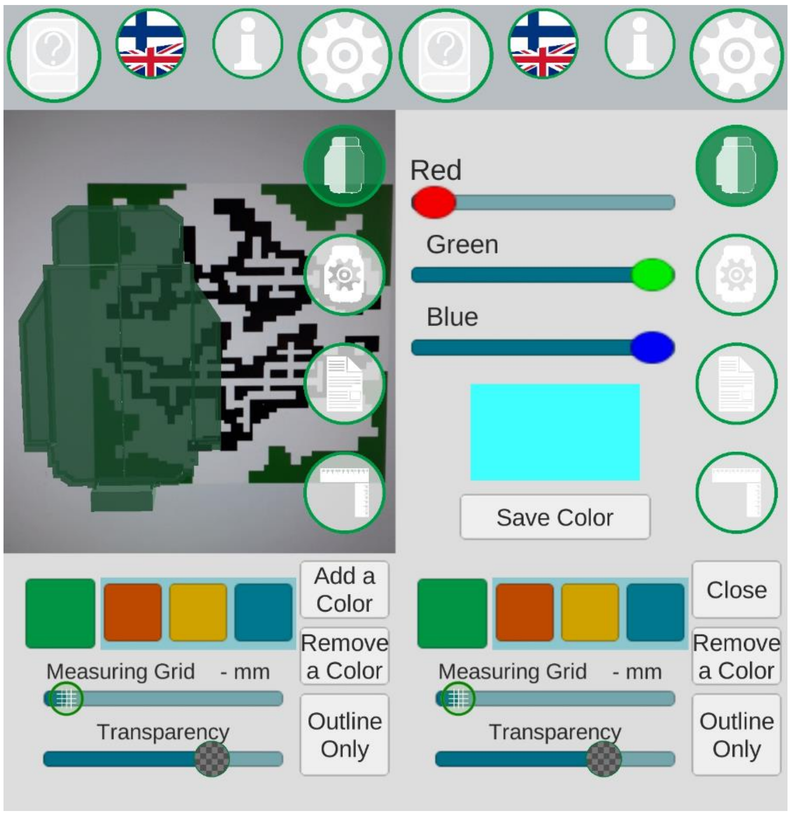


Figure 9 - The Two Stages of Display Menu

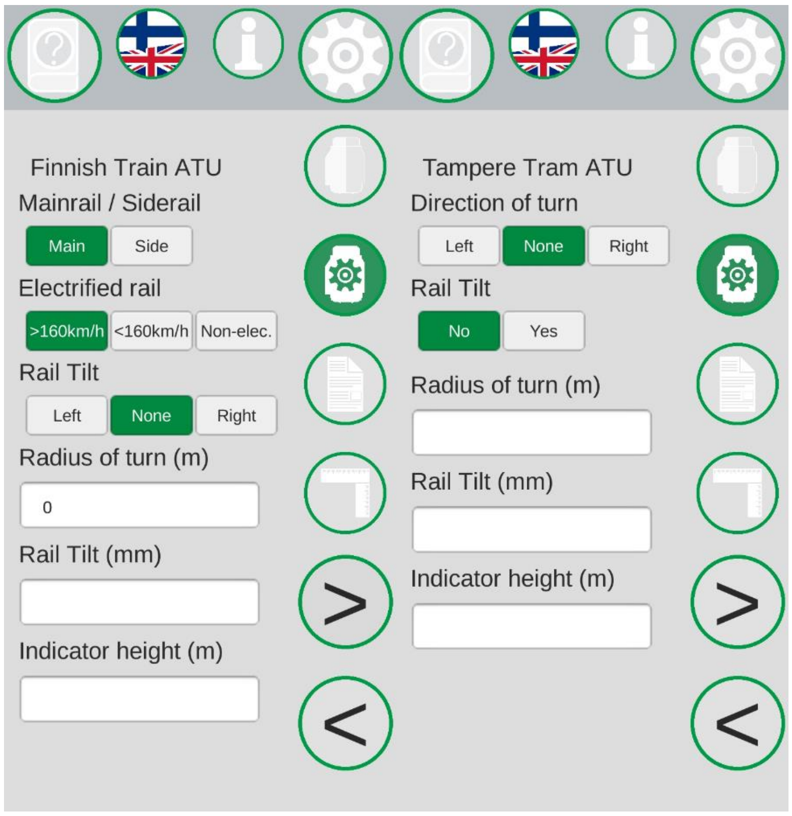
By dividing the menu into two partial menus that are accessed in stages, the user can view the ATU when applying changes to how it is displayed. The first section of the menu allows the user to make changes. The second section allows the user to tailor the color options to meet their needs.

Colors are selected by tapping them. The default color selection available is based off the company colors [4] of the case company. Their primary color, green, is always available from a standalone button. The other colors are in a sliding tray. The color selection in the tray can be customized. The “Add Color” button opens the submenu for doing so. As the user is now customizing their color selection instead of the ATU, blocking their view of the model is fine. The “Delete Color” button works as a toggle. When it is on any colors in the tray that you tap are removed from the selection. If the tray is left empty when the user exits the application, it will be restored to the defaults the next time the application is started.

The transparency of the ATU model can be changed with a slider. In addition, the “Outline Only” button can be used to activate or deactivate the filler pieces of the ATU model segments.

The ATU Control menu consists of a single menu with changing contents. It has a common base of two direction buttons in the lower right that are used to browse through the different ATU models. The buttons deactivate the current model and Control UI for it, then activate the next model in the collection and the corresponding UI. The application only has two ATU models in it but would be able to handle a larger selection. If the number of ATUs included by default in the application grows unwieldy, it would be best to introduce a menu for setting individual ATUs active or inactive as needed. This would keep the browsable selection down to a manageable size.

Each ATU model has its own controls but shares a common ordering as shown in Figure 10.


Finnish Train ATU
Mainrail / Siderail: [✓ Main] Side
Electrified rail: [✓ >160km/h] <160km/h Non-elec.
Rail Tilt: Left [✓ None] Right
Radius of turn (m): 0
Rail Tilt (mm):
Indicator height (m):

Tampere Tram ATU
Direction of turn: Left [✓ None] Right
Rail Tilt: [✓ No] Yes
Radius of turn (m):
Rail Tilt (mm):
Indicator height (m):

Figure 10 - ATU State Controls

The ATU states displayed in Figure 10 are the default states for these models. The buttons for setting the state of the ATU model are listed first as the state determines what inputs are needed to modify the ATU. The input fields are in order of expected frequency of use. Buttons representing the active states have a different color than inactive buttons, green with white text in the Default theme.

Input fields for turn radius and rail tilt permit integer inputs. Indicator height permits decimal input in order to have the correct scale and precision. While decimal input mode allows for negative values, values greater than zero are required by the ATU Manager to apply tilt scaling. If the requirements are not met only regular turn scaling is applied. for regular turn scaling a turn radius of zero is treated as ‘no scaling’. For the tram ATU no scaling is applied if either radius is zero or turn direction is set to ‘None’. Empty fields for values required for the type of scaling the state of the ATU calls for are marked with a red star.

4.2.3 Documenting the ATU

The application allows the user to document the state of the ATU using in-application camera and note-taking tools. The camera function uses the back camera of the device to take a picture that shows the 3D model of the ATU but omits the UI. The application displays a shutter icon over the screen indicating when it takes a picture. The final picture also includes a notes field that contains both automatically gathered and freeform notes. The picture is created by combining the views of the main camera and a secondary notes camera that observes a duplicate of the Notes menu.

Freeform notes are accessed from the Notes menu, third from the top in the sidebar. The menu itself is simple, consisting of a header and a blank page with a field for text. The text field itself does not do anything beyond its basic display function with the notes written on it. The notes are copied over to a similar form viewed by the secondary notes camera. The copying happens when the camera button is pressed, as part of the overall notes update, prior to taking the screenshots.

In addition to the freeform notes, the application takes automatic notes as appropriate to the state of the application. Automatic notes are for recording information that the application has access to, expects to be useful in most cases and would be difficult to record manually in a timely fashion. Such information includes the state of the ATU and the spacing of the measuring grid. Each auto-documenting component is responsible for composing its own notes. The NotesManager component simply calls for them and writes them down for the camera.

4.2.4 Other Utilities

Measuring Tool

The Measuring Tool was possibly the most important feature for the development of the application. It is effectively a virtual tape measure. Units in the Unity scene of the application correspond to meters in the real world. By giving the user control over one of the dimensions of a basic cube, we turn that cube into an improvised measuring tool.

The tool coordinates a collection of measuring tools. Each tool consists of two cubes and a plane. The cubes form the body of the measure. The first cube handles vertical distance, the second horizontal distance. The plane is at the very end of the horizontal measure. The plane is aligned such that it has height and depth. When the user is lined up directly with the end of the horizontal measure, this lack of width on the plane turns it into a razor-thin line. This allows the user to iterate the length of the horizontal segment until the plane lines up with what the user is measuring. The Measuring Tool is accessed from a minimal partial menu so the user can see the measures while adjusting them.

Testing the Measuring Tool against an ordinary tape measure on short table top distances was instrumental in determining that the application is capable of the degree of precision required for inspecting ATUs.

RSU Indicator

“RSU” stands for “Ratatyön Suojaulottuma” [5 appendix 1], which roughly translates to “track work safety zone”. It refers to a zone around the rail that has safety restrictions that must be observed. For the purposes of this application, the relevant restriction is a prohibition on storing tools or materials within the RSU. To help determine whether a given item is correctly outside the RSU the application provides two tools.

The first is an internal distance calculation. When the target scale is correctly assigned units in the Unity scene correspond to meters in the real world. In the scene the device the application is installed on is represented by the ARCamera object. By measuring the horizontal distance between it and the target in the scene, it is possible to determine whether the device is within the RSU. Using this information, the application can either display a warning triangle if the device is within the RSU or a green checkmark if the device is outside of it.

The other approach is technologically simpler. A 2.5 scene units long bar is extended from the target. This bar can be observed from the side to determine whether it reaches the item we are checking or not. Which approach the end user chooses will likely depend on the conditions at the work site. Obstacles like ditches and other rough terrain can make one approach or the other much harder without the obstacle itself breaking RSU.

4.2.5 Languages and Translation

The languages available in the application by default are English and Finnish. The available languages were limited to the fluent languages of the developer as a dedicated translator was not available for the project. The languages menu has a simple column of buttons from which a language is selected. At the foot of the menu is an input field and a button for adding a language from a text file. A second button generates a ready-to-use template for adding a language. The name of the text file that is loaded is also the name the language is listed with.

Translations are done by auto-translation components added to each text element in the UI. Each component compares its current language to the language of the application. When the two do not match the component adopts the new language and retrieves the matching translation from the TranslationManager component. The auto-translation components are all derived from a base Translatable component. Their sole difference is in the enumerator they use. As the application has over 90 translated elements, a single enumerator would be unwieldy to use in Unity Editor. Instead the application uses a single main enumerator to manage all translations but uses smaller subset enumerators to define the scope of auto-translation components. The subset enumerators reference the main enumerator to define their values.

The translations are stored in an array of lists set to the size of the main enumerator. The indexes of the array correspond to values of the main enumerator, an index per translatable concept. The enumerator values of Translatables correspond to the content they are responsible for translating. New languages are appended after the existing ones, one entry per list with “null” for entries that are not missing from the language file.

4.2.6 Settings

The main settings menu is accessed from the upper right corner of the application UI. The main menu itself does not directly host any settings but offers an array of sub-settings menus. The sub-settings are divided by their role

- Target settings
- Color themes
- Sign-in settings

Color Themes were covered with UI in Chapter 4.1. Sign-in settings handles a small set of ease-of-use related settings. As the application is mostly used in the field it is preferable to minimize the amount of input required for signing in. This is done by offering two auto-completion options for sign-in email. The first auto-completes the email from ‘@’ onwards, ex. “@samplecompanymail.fi”. The user only needs to input their personal uniquely identifying segment of the full email address. The second setting allows users to save the full email for future sign-ins. The email is saved in full even if auto-completed. The auto-completion is relatively simple. It appends the full @ending.fi to input with not @ symbols, just the “ending.fi” if input ends in @ and nothing to input containing an @ but not ending in it. The auto-completed segment can be changed as part of the sign-in settings. The two toggleable options are also accessible from the sign-in screen but the auto-completed segment can only be modified from the sign-in settings menu. The default is set to the company email address for the case company.

Target settings consist of size scaling, displacement mode and additional X- and Y-displacement. The size scaling assumes that the size given for the target is its real size and it is intended to deploy a full-size ATU. A single unit of size in the ATU model corresponds to 1 m in its specification and the dimensions of the targets are given as 1 m x 1m when uploading them to a database. The scaling is applied to a dedicated “scaler” component that is a child of a target and a parent of the AR content deployed by the target. To arrive at the correct scale, the scaler is divided with the user-inputted size of the target converted to meters, ex. a 200 mm target would set the scale to 1 / .2.

Displacement settings are part of how the application secures a dependable point of reference for deploying the content. For the state of the ATU to correctly represent the situation the user intends to survey the ATU model must be positioned in the center of the track. To do so it needs a point of reference from which to reposition itself. For the use case of this application that reference point is rail width, a standard 1524 mm for train tracks and 1435 mm for tram tracks. The rail width is defined as the distance between the inner sides of the rails. There are varying sizes of train track but for all of them this distance is the same. The size of the target and its position relative to the rail can be considered known. From this it is possible to derive

center of rail = target position -+(target size / 2) +- (rail width / 2)

where size and width are positive or negative depending on which rail and which side of the target combine to serve as the reference point. Aligning the left side of the target with the left rail results in

center of rail = target position - (target size / 2) + (rail width / 2)

The application provides a displacement mode for each of these variations, saving the user from having to calculate and manually apply them in the field. The X- and Y-displacement inputs would also be capable of handling them but are intended to handle a narrower range of use situations. In the event none of the preconfigured displacement modes apply to a situation, if the position of the target relative to the center of the rail can be established the ATU model can be positioned correctly using manual displacements. In this role they are the final safety net. The other, preferred role for them is developing new target deployment methods without requiring a rebuild of the application for each prototype.

The settings can be reset from the button at the bottom of the main settings menu. The process creates a fresh copy of each Setting type, then saves those settings, overwriting the existing previously saved settings with the defaults. It is recommended to do this when updating to a new version of the application as any changes made to the settings will typically make them incompatible.

4.2.7 Access Control

When the application starts, the user is first shown the case company logo. After that the application starts in Access Control. Access control consists of three views

- Wait View
- Enter View
- Sign-In View

as shown in Figure 11

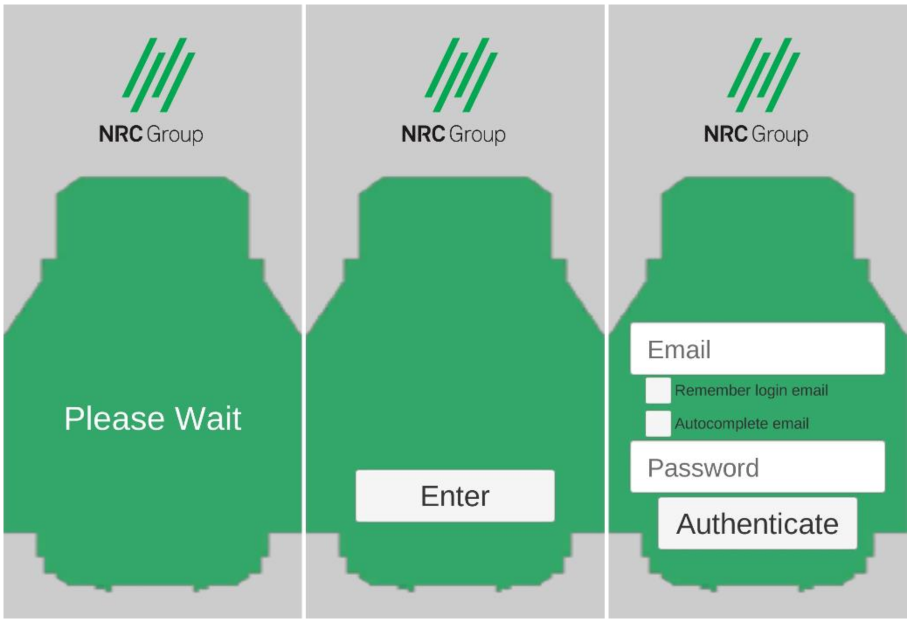


Figure 11 - Access Control Views

The first view visited is the Wait View. In this view the application checks if there is a stored authentication that is still valid. In that case the user is directed into the Enter View and onwards into the application Main View. If no stored authentication is available or the stored authentication is found to be expired or false, the user is directed into the Sign-In View. The Sign-In View prompts the user for the credentials they use to sign in to company applications. The sign-in settings options for simplified email input are also accessible from this View.

The authentication is handled with Azure Active Directory sign-in with the Microsoft.Identity.Client library. The flow used is MSAL Username / Password [6] for desktop applications. The users sign-in with company email and password. The factors that led to choosing this flow

- Platform independent, so the application is portable with minimal platform-specific coding.

- Internal to the application. Device code flow [7] would have satisfied platform independence but bouncing the user in and out of the application simply to sign in would be a poor user experience.

- Add-on independent, as while Unity can implement in-application web browsers developing one is outside the scope of the project and making the application dependent on a third-party web browser component for secure sign-in is questionable at best.

The flow is restricted to organizational accounts, such as school and work accounts. Personal Microsoft accounts are not supported. This is not an issue for the expected user base of this application but may be an obstacle for enterprise applications that would be expected to serve Guest users. The flow is also unsuited for general consumer applications for the same reason.

Other limitations are the inability to support for multi-factor authentication and single sign-on. Single sign-on is not important for a specialized stand-alone mobile application. Multi-factor authentication is used by many organizations and is applied to the account, not to individual applications. If the case company were to adopt MFA the application would need to change authentication flow to device code.

4.3 3D Modelling

The ATU models used by the application are composed of multiple parts in order to properly implement their scaling. Concrete steps are described on the assumption of using Blender but as nothing particularly specialized is required it should be possible to create comparable models in most 3D modelling software. As the individual ATU pieces are regular shapes with clearly defined dimensions, creating the individual pieces should be a reasonably simple task. How to divide the ATU model into these pieces is slightly more complex and described for both models later in the chapter.

As the scaling is exclusively horizontal the first step is a vertical split of the model. For manipulating the model in Unity, it is best if the root coordinates for the pieces match the zero point of the ATU. There are two relatively simple ways to accomplish this. First option is to create symmetric pieces and use a large cutter piece bordering on the x=0 plane. The halves of the piece are created with Intersect and Difference Boolean operations. The second approach is to make an asymmetric piece, duplicate it and rotate it to create the other half. As the asymmetric pieces are unlikely to be properly centered, two large boxes positioned at (0, 0, 0) are turned into new centered copies of the pieces using the Intersect operation. The goal of centering the pieces to the zero point is to make it possible to reset the full ATU model to its correct neutral state by setting all local position and rotation values for all pieces to 0. If the correct neutral state requires all pieces to be in specific arbitrary positions, then any disruption of the model can potentially create a permanent measuring error as the piece cannot be put back in the exact same spot.

The application supports a hollowed outline-only state for the ATU models. The purpose is to have a solid sharp outline while still being able to see what is behind the ATU. This could not be done by simple scaling the transparency of the material. Hollowing a part is done by creating a cutter piece at least the size of the hole you intend to make, preferably going over a bit on the side where it should leave no outline. A copy of the piece to be hollowed is made. The cutter is then placed into position and used in two Boolean operations on the identical pieces, one Intersect operation and one Difference operation. The Difference operation creates the outline piece and the Intersect operation creates the fill piece for that outline piece. In the model hierarchy fill pieces should be children of outline pieces. The two are expected to move together, but only the outline piece is expected to be available independently. Therefore, the fill piece is the child of the relation.

As the ATU is scaled by moving the parts away from each other, a gap naturally forms in the center. The gap is very narrow under most conditions and since the ATU is surveyed at the edges the gap has very little impact on the core functionality of the application. Nevertheless, it can be distracting. The solution is the introduction of a middle column to fill in the space left by the scaling. It should consist of a stack of boxes of the same depth as the main model it fills in. The stack should have pieces corresponding to the outlines and the filler pieces. The middle filler pieces are grouped together in the ATU Manager with the fillers that they connect. The width of the filler stack should be one unit. This allows it to be auto-scaled with the distance between the pieces it is connecting, provided they are both rooted at (0, 0, Z). If the scaling is uneven, the middle filler will also need to reposition itself exactly in the middle of the two by

$$M = L + (L - R) / 2$$

where L, R and M are 3-dimensional vectors representing the positions of Left side, Right side and Middle filler, respectively.

Finnish Train ATU

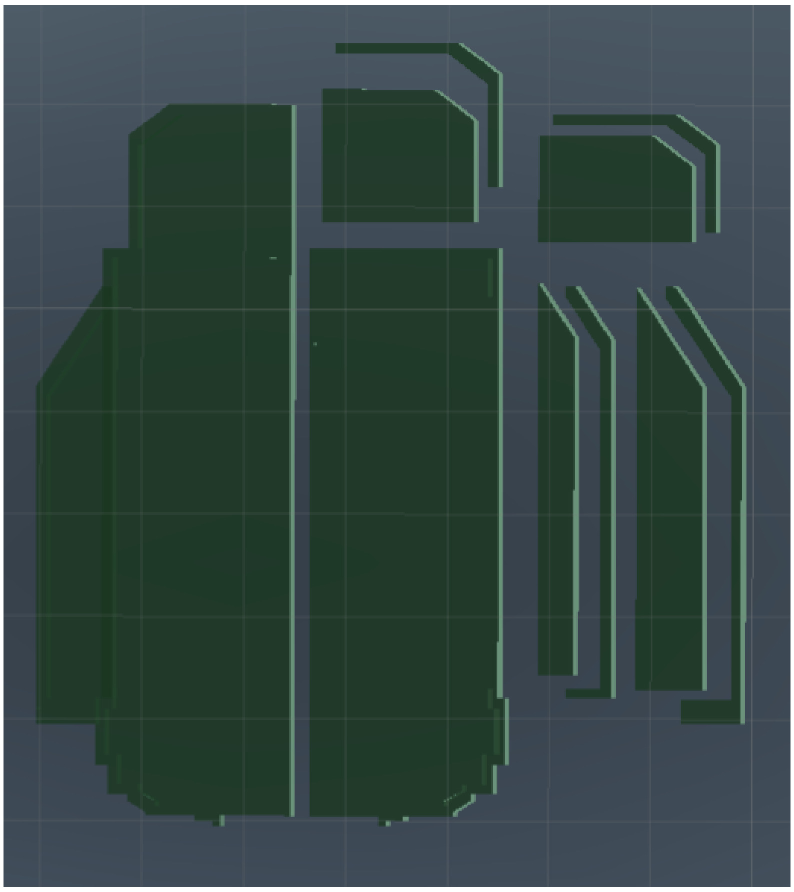


Figure 12 - Train ATU Model

The train ATU is the more complex of the two. It has different outlines for siderail and main rail connections. Additionally, electrified connections have an additional segment added to the top of the ATU, the height of the segment depending on the speed of the train in that section of rail. As shown in Figure 12, the model is reduced to a shared base component and four add-on segments to account for all six state variations in one model. The train ATU model has eleven combined outline and fill pieces per side and seven middle connector pieces for a total of 29 pieces.

Width scaling applies the same to all parts in a side, so it is applied on the organizational object level. Each of Left, Right and Middle section has an “empty” organizational object that allows the pieces to be moved as a group. The empty objects do not have a 3D model mesh. For the train ATU the Left and Right objects are used to implement width scaling. The Middle object references them to set its own position and width to fill the scaling gap.

Tampere Tram ATU

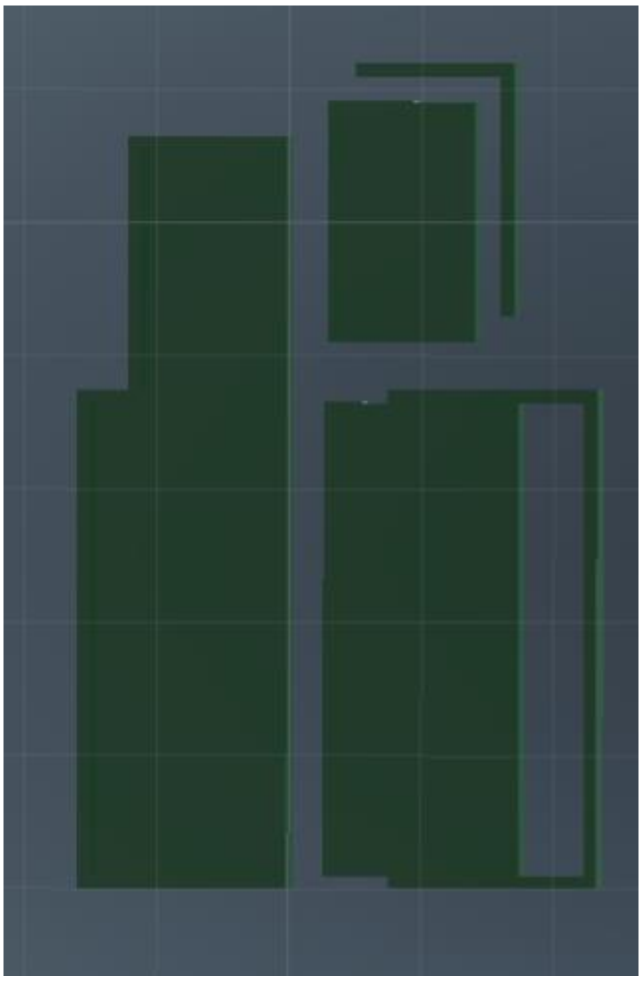


Figure 13 - Tram ATU Model

The tram ATU is very simple, consisting of just the arm and the body. That separation, the split into halves and the implementation of the outline mode results in a total of eight core pieces. Additionally, the middle connector pieces contribute a further four: one per part, doubled by outline mode support, for a total of twelve pieces.

The main distinction from the train ATU is that the width scaling function for the arm is very different from the one for the body. As such scaling is not applied at the ATU half level but at the level of the arm and body outline pieces. The same holds for the middle connectors, they need to reference the corresponding outline pieces for their scaling instead of the ATU halves.

4.4 Vuforia Development

The application depends on Vuforia Targets to deploy the AR content with which it checks rail traffic clearances. For the application to serve as a useful tool and a dependable substitute for current methods, the results it produces must be accurate, stable and available.

Accuracy is the measure of the application's ability to give precise results. If the application is given an accurate scale for the deployed target it needs to be able to deploy a correctly scaled ATU model into the correct position in the camera view of the device.

Stability refers to the ability to get consistent results from the application. The deployed content must remain correctly positioned for the duration of the survey work. If the application has trouble with detecting or tracking the position of the target, the model can

- Become tilted or otherwise misaligned and useless for surveying the ATU.
- Shake in place, in what is essentially a rapidly changing slight placement error.
- Disappear altogether if tracking is fully lost
- Become attached to unrelated elements and move with them

The above can be minimized with high quality targets as well as proper movement away and around the target.

Availability refers to the application being able to deploy the AR content under the current conditions. For example, the application cannot provide useful results if a Target cannot be detected. This might be caused by darkness or heavy snowfall concealing the Target or high winds causing the Target to sway uncontrollably. In practice, most of such extreme conditions are issues for existing methods as well. One known exception, darkness, is detailed in 5.1 Future Developments.

Vuforia has a feature called Extended Tracking [8]. Extended tracking requires that a target has already been detected. When that target is no longer in view Vuforia tries to track its position using the environment. The reliability of this feature is heavily dependent on the surroundings. On a plain office table extended tracking is quickly lost. Even if it has additional items scattered to help with tracking or is used in otherwise feature-rich environment it has a tendency to lose or gain distance randomly. This makes extended tracking by itself too unreliable for any kind of precision work.

When precision matters the correct approach is to have both the Vuforia Target and the item the user wishes to survey on the screen. This is best accomplished in a U-motion. The user starts at the target to detect it. The user then backs away some distance before moving sideways to be in line with the item. The user then approaches the item while keeping the target in view. As the ATU models have some depth to them the user can tell if they are in line with the edge of the ATU by seeing it as a clean cut. A degree of transparency on the model allows the user to see through it to confirm that there is no visible surface on either side of it. This confirms the user is lined up exactly with the edge. From there it is plain to see if the item would be inside the ATU or not.

In most use cases Vuforia targets have individualized content. This is done by simply including the content as children of the target in the scene hierarchy. The content is then deployed when the target is detected. The content deployed by this application is relatively complex. Duplicating that content for each of the supported targets would put an unnecessary strain on the application. Instead the application uses a single copy of each of those elements, grouped together under the “ModelHost” object. This includes all ATU models, the measuring grid they share and a root object for the Measuring Tool.

The ModelHost itself does not implement the sharing, it is simply an organization tool. The sharing is done by TargetController components added to all Vuforia targets. Once a target is detected the TargetController attaches ModelHost to the scaler object of the target and applies the target settings to the deployed content. When the ModelHost is moved from one target to another it will by default seek to maintain a consistent size in the scene by automatically changing its own local scale. If the local scale is not set back to 1 after each step it will result in an accumulating scaling error.

4.4.1 Target Types

Image Targets [9] are the most basic type of target. For most of its development the application dealt exclusively in Image Targets. Image targets are simply regular 2D images used as targets. The quality of the target depends on how recognizable it is to image recognition technology. While the content can be deployed in front or behind the target, the target itself will not obscure the content.

Multi-Targets [10] are a collection of Image Targets that have defined positions relative to one another. A group of Image Targets simple placed together would all track as individual targets. Each of them would deploy their individual content, up to the limit of simultaneously tracked targets the application supports. As a Multi-Target, the group works as a single target and detecting one or more of the faces of the target deploys their shared content exactly once. A Multi-Target is useful when you expect to move around the target to some degree, as when you start losing track of one face you will gain traction with another. Users of the application want to be in line with the obstacles they survey. increased reliability when at an angle to the target is therefore very useful.

Object Targets [11] are essentially the next stage of Multi-Targets. Instead of a collection of Images as the faces of a cube, they have a collection of angles that form a sphere. This approach allows Object Targets to be created from a large variety of objects. It also makes them take up more space: Target Databases with only Image Targets can handle 1000 targets, whereas one with only Object Targets can handle only 20.

Cylinder Targets [12] were deemed to not be particularly useful for this application. They offer no advantages that Multi-Targets cannot also provide. As with Object Targets, their main advantage is in being able to apply AR content to enhance an existing product or service. They are not particularly useful when your main product is the AR content.

4.4.2 Target Databases

Targets in Vuforia are organized into target databases [13, 14]. There are device, cloud and VuMark databases. Device and VuMark databases are stored and queried locally on the device while cloud databases are stored remotely and queried over the internet.

Device databases are the “general purpose” database, supporting most target types. The recommended number of Image Targets in a single device database is 1000. How many targets a single database can support depends on the images used. A device database dedicated to object targets can hold 20 of them. Presumably Multi-Targets and Cylinder Targets require at least slightly more space than simple image count would imply, as they also involve data about how the images relate to one another. Unfortunately, detailed information about it is not available from Vuforia and discovering it is outside the scope of the project. This application is unlikely to need more than 100 Image Targets, even accounting for various weather conditions and similar circumstances under which a specialized target is much more visible than an ordinary one. As a local database, the response time for detection is a few frames. Since the size limits are not an issue for this application, the other database types would have to offer considerable advantages in order to be chosen over device database.

VuMark [15] databases are a specialized device database for VuMark targets. Vumarks are a variation of Image Targets that follow a specific set of rules. Among them is reserving a section of the mark for encoding information as a pattern of alternating bright and dark squares. This allows a single clearly branded design to differentiate between a large range of individual items. An example use for the case company would be to apply marks to construction machines and query related information with the identification from the VuMark. While there might be a use case for the company, VuMarks offer no unique advantages to this application and have additional constraints in their design, so they were not used.

Cloud databases are hosted by Vuforia as a service that is made available based on the license used. The main advantages of Cloud databases are the volume of targets stored and the ability to change both the targets and the AR content those targets deploy. This application does not need targets in excess of what a device database can support. The content it deploys is complex and closely linked to the rest of the application. It might not be outright impossible to use the flexible content of cloud databases to swap out ATU models, but it would come at the cost of massively increased complexity and reduced responsiveness of the application. The cloud content would need to be equipped with materials and scripts, then connected to the rest of the application before it is ready to use. This would cause a considerable spike in complexity in the rest of the application that has to make the content complete. Those complexity issues would scale sharply with the flexibility of content that such a setup is expected to support. The amount of work needed to make such a system would be more efficiently spent implementing the models manually one by one as needed. As such this application would not benefit from being able to deploy content from the cloud.

The ability to easily update the database could be included in an application using a device database. To do so the application is set up to load a database from a folder when it starts. The application can then query a server to check whether an updated database is available. When an update is available it is downloaded and replaces the previous database. For this application the ability to upgrade the database with new targets could prove useful when existing targets prove a poor fit for some use conditions. The need to do so is typically user-specific, so there is little reason to set up a server to pass regular updates to all users. The obvious downside in this implementation is redownloading the whole database instead of just the updates.

4.4.3 Image Target Development

Vuforia Target Manager grades targets on a scale of zero to five stars [16]. The first set of six image targets all scored full five stars in that evaluation. In practice their performance was unreliable and there were noticeable differences between them in detection distance, tracking distance and stability. All image target batches discussed in this chapter are included in Appendix 1.

Target Quality

The quality of an Image Target refers to its overall recognizability by Vuforia under a variety of conditions. This includes the distance at which it can be detected, the distance to which it can be tracked, how reliably the content holds still, etc. overall performance.

Vuforia is based on image recognition technology. It works by learning recognizable features of images and using those features to recognize it [17 5.1.1]. Features are essentially patterns, including patterns of patterns and the different degrees of recursion that permits. As an example, a sharp angle is a feature. Two such features together, one nested inside the other is its own feature, possibly the tip of a cat ear. Two such patterns along with a selection of other specific patterns is where detecting cats in images begins. The patterns used this way were not given to the image recognition software but were instead learned from a body of data that had images with and without cats. Images with cats had specific patterns of patterns that over extended training allowed the software to form a flexible “blueprint” for “this image has one or more cats in it”.

The takeaway for the purposes of Vuforia targets is that image recognition looks at an image at a very coarse level, at all levels. The image is split into tiny snippets which are combed for patterns and are in turn are grouped to form new snippets that are combed for patterns. This means that a good quality image target needs to perform well on all levels of this process. An abstract shape consisting of clean lines making soft curved turns with minimal crossings is unlikely to work well as a target. It can be very distinct on the highest level, but it has very few features at the lower levels. Image recognition will struggle to gain traction on the curves and Vuforia will struggle to figure out the orientation of the image.

Initial batch

The first set of targets was created in a very freeform fashion, picking a single GIMP tool and using it to create an irregular pattern with sharp, clean corners and a variety of colors. The first four were created with the Paths tool, the fifth with the square select. Finally, the target “NRZ” was created using the case company logo. The targets were all scored 5/5 by the Vuforia Target Manager.

In practice there were noticeable differences in their performance in testing. The simplest of the targets, “NRZ”, was also a top performer. This led to the creation of the second half of the set, simple black and white targets with solid fill. The purpose of these targets was to test how simple a target could be and still perform well. The simpler targets performed comparably to the previous targets, with some standouts and some underperformers. The testing was somewhat hampered by lack of proper measuring tools.

Second Batch

The second set of images was divided into two subsets to pursue two different questions. The first part, 2.1, experimented with “pinning” corners of an image. The idea is to use some of the space on an image to mark the corners as corners and use the rest for more random detail. There was some variance in the overall design in the hopes of gaining insight into how fine or coarse level of detail makes for a good balance of detectability and trackability. The standout targets from 2.1 are positioned fifth and eight in the grouping in appendix 1.

Group 2.2 A & B were intended to test whether using a variety of colors made a difference compared to a black and white image. The results were very evenly matched with color targets having a slight edge overall.

The test case had the target displayed on a computer screen and used the measuring tool feature to measure distance to the edge of a mouse pad. The distances involved were under a meter, changed pseudo-randomly for each measurement to minimize the impact of educated guesses. The errors average error in five tests was less than 5 mm for each target and as low as 1.5 mm for the best performing one. As the differences in average error between the targets were 0-3 mm it is difficult to conclusively call one design or other superior.

Third Batch

The goals of the third batch were to test different approaches to generating suitably irregular target patterns. The batch also involved some usability patches compared to the previous one, such as a clear outline and the inclusion of an arrow element to indicate which side of the target should be upwards.

Despite the efforts to create a coarse and irregular outline, the A-type targets had performance issues that were probably related to rotation symmetry. In hindsight it makes sense, as images are invariably somewhat coarse at least on the level on which machine vision parses them. The circular patterns essentially cast doubt on which side was up. The degree to which there is a curve is the degree to which there can be doubt about the direction on some levels of the image recognition process.

B-type targets were more dependable than A-type targets. Both target types performed mostly comparably to targets from the second batch in similar accuracy tests, with one standout averaging 1 mm of error. From this it was concluded that the beyond a few basic ideas the improvement space for target design gets narrower and narrower while the returns are less noticeable.

Conclusions

On later review, the issues with the first targets was a lack of contrast between the colors used and a relative lack of good trackable features. The only target with a good amount of coarse detail also re-used the same outline, reducing the uniqueness of the detail. The second half of the first batch did much better on the outline detail front and served to prove that relatively simple shapes can make for good targets if properly designed.

The design of Batch 3 type B is the currently understood best practices. A clear outline is necessary for correctly deploying the target. A direction indicator allows the remainder of the image to be a feature-rich jumble of irregular random patterns. A branching pattern of rectangles seeking to form T-shapes will create a good balance of dark and light. The right corners in such a pattern are easily detectable detail for image recognition. Creating the pattern by hand with rectangle select tool does a good job of ensuring no two elements have quite the same dimensions, keeping the patterns unique. It is unclear just how fine-grain the linework should be for best detection and tracking distances.

4.4.4 Target Deployment

In order to produce accurate results, the application needs a reliable point of reference from which to deploy the AR content. As noted in 4.2.6 over displacement settings, this point of reference is the inner edge of either rail. Ideally the target can be deployed with its bottom on level with the top of the rail and either side in line with the inner edge. The means by which it is deployed need to be able to hold it in place in a variety of conditions. The mechanism responsible for anchoring a target to the rail in some way or form will be referred to as a rail clamp.

The first prototype of the clamp was simply gravity based, somewhat similar to a scales in how it was intended to work. One end was heavy and would naturally swing downwards if unsupported. Braced against the side of the rail it would hold the target in place. While it worked with the added help of an external brace holding it in place, it had issues placing the image level and would handle forces applied along the rail, such as wind, poorly.

The second prototype attached to the rail with a magnet. it consisted of two metallic corner plates attached with screws to each other. One corner plate would hug the rail while the other presented a surface for attaching a backing board for targets. The magnet would be on the inside of the first plate, holding it to the top of the rail.

It held onto the rail well but as the top of the rail is far from even it too had issues with placing the target level. The magnet made the issue worse as it would pull to maximize its grip, which would not place the target correctly. The corner plate hugging the rail would need to be braced against the rail to hold the target level. It also needed another measuring device to help set it level as the rail itself was insufficient reference point.

The third and final prototype upgraded the previous version with an additional magnet, used to add a rotation joint between the clamp proper and the backing board for targets. The basis for this change was that the levelness of the clamp only mattered to the degree it contributed to the levelness of the target. If the target could be righted independently of the clamp, then how crooked the clamp attaches to the rail would not matter. A small water level was added to the backing board to help set it level without external tools.

The magnet joint in combination with the water level allowed for righting the target to the degree that a tachyometer could not declare it to be leaning in either direction. This was deemed sufficient degree of independently attainable levelness.

5 Field Testing

Field tests of the application were for the most part tests of the rail clamp solutions. For the first field test there was some uncertainty if natural environments would be worse for the application than the office environment it had been tested in until that point. The concern was unfounded as natural environments proved to be very feature rich and easily capable of supporting extended tracking.

Field test 1 – Hyvinkää

The first field test was carried out at Hyvinkää train yard. The choice of location was based on having an available section of rail that saw little to no traffic. Ease of travel for the parties involved in testing was the other factor. The testing was done using the first prototype clamp. As the rails at the train yard were smaller than the type of rail the clamp was designed for it had to be adjusted in the field. It also required a brick to brace it against the rail to hold properly still. In these tests the application was still relying on extended tracking. The tracking was maintained without issue, but the results were off varying amounts of centimeters. The magnitude and variance of errors meant that as it was the application was unusable in all but the obviously in the clear cases. The users would simply not have any means of determining how off and in which direction the results were. The takeaway from this test was that a better clamp and more stable measuring results were needed.

Field test 2 - Järvenpää

The second field test was carried out alongside measurement work done north of Järvenpää. The clamp used was the first version of the magnet clamp. The obstacle measured was simple square target on a stick held up with a tripod near the rail. In this test both the target and the obstacle were kept in view while surveying. Despite this the initial measurements were consistently off by 5 cm. When the clamp was checked using the tachyometer it was found to be set at an angle. After this was corrected the measurements were within 0.5 cm of error. These results led to the development of the magnet joint for the clamp.

Field test 3 – Riihimäki

The last field test carried out as part of the project. The goal was to confirm that the updated prototype rail clamp could set a target level without the aid of other measuring devices. Being reliant on a specialized piece of measuring equipment would largely defeat the purpose of the application. The test site was a substitute track assembled from small pieces of rail next to a lamp post at the station parking lot. The lamp post also served to test the ability of the application to survey obstacles at greater heights.

There were some initial difficulties as in the process of upgrading the rail clamp the target board had effectively switched sides. This caused it to deploy the ATU model out of position to measure distance to the lamp post. Displacement settings were added into the application between field tests two and three, so the issue was solved by using manual displacement to correct the positioning. This event led to expanding the displacement modes to include all combinations of rail edge and target side.

Another issue was with retaining tracking. Initially the tracking was lost after moving approximately four to five meters away from the target. This was sufficient to measure the distance to the post at eye level but would be insufficient for surveying the upper parts of the seven meters tall train ATU. After some experimentation the issue was found to be a user error. The starting position had the phone facing the target in a downwards angle. This likely gave the application a warped initial impression of the target, causing it to lose track once there was enough distance for the angle between phone and target to level out. By placing the phone level with the target and holding it level for a while when backing away the tracking held for twenty meters, the full available distance at the parking lot.

6 Future Development

This section covers ideas and features that while interesting did not make the scope of the project.

Targets for Low-Light Conditions

One of the conditions where regular image recognition would struggle to perform is low-light conditions. There is simply not enough contrast for it to perform to its full ability. Using an external light source illuminate a target has issues with securing support for the light source. Additionally, most durable and waterproof materials are at least somewhat reflective, which can effectively blank out the target from some angles.

A potential solution for it is to use a target that is illuminated from the inside, similarly to a paper lantern. With the right combination of light source and thin plastic the target could look approximately the same under bright daylight and in pitch-black darkness. Some degree of adjustable output in the internal light source would bridge the gap between the two extremes. Such a target device should also use a set of distributed light sources to ensure even lighting instead of a single bright spot. Careful varying of the thickness of the material can serve the same purpose as light source distribution.

User-Added ATU models

The ability for the application to automatically parse a combination of a 3D model file and a text file schematic into a ready-to-use ATU model in the application was one of the possible future-proofing features put forth early in the project. The application would load the model and place it in the ModelHost. A general-purpose ATUManager component would be added to the object highest in the model's hierarchy.

The ATUManager would use the schematic to generate collections for managing the different states and implementing scaling. A mathematics system would interpret the scaling functions supplied in the schematic and convert them into linked lists of operations. These operation collections would be used to apply scaling, with a scaling calculator object iterating through the collection to produce the scaling value.

An UI would be added to ATU controls for the new model. The UI is generated based off the schematic with the expectations that all full states of the ATU can be expressed as a combination of sub-states and that sub-states can be expressed as one-or-none of a collection of one or more states. For example, electrified rail is in a one-or-none of two possible states. Main or side rail is in one of two possible states. Outline-only mode is a one-or-none of one possible state.

Uncertainty of the pitfalls of importing 3D models in runtime held the feature back, as it would have been a commitment of unknown scope in a project that has no shortage of features to implement. There are few scenarios where the feature would be truly irreplaceable. The application would need to be serving a global market as a single solitary product, with each individual client slotting in their own specifications. In more modest scopes or where it is possible to deliver tailored copies to each client the ATU models are best implemented the same way train and tram ATU were. Potentially greatest value from implementing this feature would have been as groundwork for other applications involving AR content.

Translating User-Added Content

Since user-added content is primarily intended for the person creating and adding it, it would be perfectly fine to exclude it from the translation system with the expectation that the user understands their own content. However, there is a case for creating content to be added and used by other users of the application. In those cases translations for user-added content become relevant.

The main TranslationManager cannot meaningfully speculate on what kind of content users would add. It would also need to take care to work around space reserved for user-added translations, both in further development and in runtime. It would also need a protocol for when user-added content has filled all the space reserved for it and needs more.

The solution is to give each piece of user added content its own translation manager. This translator would essentially be a smaller version of the main translator that the autotranslators for custom content use. It would support whatever languages the user added and default to the first language included.

Coordinating translations between the two translators would involve an enumerator for all supported languages. Each translator has a list of the languages it supports and their position in that list matches the position of the translated terms in their translation lists. This allows all translators involved to serve translations in the same language regardless of what other languages they support. In both cases if the translator is asked for a language it does not support it offers the default instead.

As there is no support for user-added content there was no need to add support for coordinated translations. The current main TranslationManager works using a shorter enumerable consisting of just the default languages.

Room-style Multi-Target

Another potentially interesting Multi-Target use case would be as an inverted target. By turning the target from a box into a room the user can survey larger structures imposed on the surroundings from within the box. With proper controls for advancing the timeline and changing visibility settings by groups this could be used to create AR-enhanced “project survey posts” for work sites. For rail work projects such a survey post could include ATU specifications. The limitations of an application like that would be the amount of 3D content a mobile device can support, the information available to accurately model the site and the effort needed to model the site. Evaluating whether such an application provides good value would require industry-specific expertise and is outside the scope of this work.

7 Summary and Conclusions

The goal of the project was to develop a field-ready product from the original Innovation Project prototype. This included developing the core AR content, the means to deploy it and document the results as well as supporting features to help users use it under various conditions. Detailed feedback is not available at the time of writing, but the test results so far indicate that the application can meet expectations.

Implementing the authentication was possibly the most challenging part of the project. The case company was already committed to using Azure AD, so the authentication had to work in that same context. While there are typically many discussions about how to solve or work around an issue in Unity, Azure AD only had a couple of dead ends. After implementing the sign-in it was easy to see why. The best fitting flow is niche and not recommended by the MSAL authors due to its legacy nature. The limitations in the types of accounts it supports and the usability issues with the workarounds could be a deal-breaker for many commercial Unity projects.

Due to the available scheduling for field tests acceptance testing was left outside the scope of the project. The goal for that testing would be to create a larger body of data about the degree of accuracy the application can reach in field conditions. The data would then be used to decide in what situations the application can replace current measuring methods. The amount of data needed requires multiple testers to gather it in a timely fashion. To do this the testers need to be provided with not just the application but also a suitable rail clamp. Supplying those may take some time.

Low-light targets would definitely be worth developing as the ideal time for work that requires closing off the track is at night when traffic is lower. The challenges with them are mostly manufacturing and logistics as they should work the same as ordinary Multi-Targets from the application’s perspective.

The project had many moments of discovery that led to simple yet effective solutions such as the measuring tool and the magnet joint on the clamp. Other ideas occurred that were only somewhat related to the topic and were only pursued briefly to determine if they would have value to the case company if fully realized. Augmented reality appears to have far more utility applications than expected. That such potential is untapped may be due to the need to understand both the technology and the use case on a specialist level. Without that combined know-how both discovering potential use cases and evaluating their viability in business use would prove difficult.

References

Details of the references are given here. Use the referencing system required in your degree programme or as agreed with your supervisor.

Layout of this page in the number (Vancouver) referencing system:

1 Ratatekniset ohjeet (RATO) osa 2 – Radan geometria, appendix 2, Väylä 2010

2 Unity Canvas Draw Order [Cited 21.8.2019] https://docs.unity3d.com/Manual/UICanvas.html

3 Unity Canvas Sort Order [Cited 21.8.2019] https://docs.unity3d.com/ScriptReference/Canvas.html

4 NCR Groupin suunnittelukäsikirja 2018, NRC Group Finland Viestintä

5 Radanpidon Turvallisuusohjeet (TURO), appendix 1, Väylä 2018

6 Azure AD Username-Password Flow [Cited 21.8.2019] https://github.com/AzureAD/microsoft-authentication-library-for-dotnet/wiki/Username-Password-Authentication

7 Azure AD Device Code Flow [Cited 21.8.2019] https://github.com/AzureAD/microsoft-authentication-library-for-dotnet/wiki/Device-Code-Flow

8 Extended Tracking [Cited 21.8.2019] https://library.vuforia.com/content/vuforia-library/en/articles/Training/Extended-Tracking.html

9 Image Targets [Cited 21.8.2019] https://library.vuforia.com/content/vuforia-library/en/articles/Training/Image-Target-Guide.html

10 Multi-Targets [Cited 21.8.2019] https://library.vuforia.com/content/vuforia-library/en/articles/Training/Multi-Target-Guide.html

11 Object Targets [Cited 21.8.2019] https://library.vuforia.com/content/vuforia-library/en/articles/Training/Object-Recognition.html

12 Cylinder Targets [Cited 21.8.2019] https://library.vuforia.com/content/vuforia-library/en/articles/Solution/Cylinder-Targets-Guide.html

13 Basics on databases, performance comparison [Cited 21.8.2019] https://library.vuforia.com/content/vuforia-library/en/articles/Training/Getting-Started-with-the-Vuforia-Target-Manager.html

14 Database Comparison [Cited 21.8.2019] https://library.vuforia.com/content/vuforia-library/en/articles/Solution/Comparison-of-Device-and-Cloud-Databases.html

15 VuMarks [Cited 21.8.2019] https://library.vuforia.com/content/vuforia-library/en/articles/Training/VuMark.html

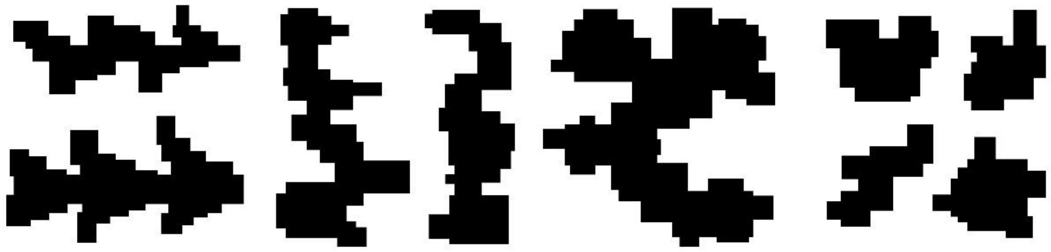
16 Vuforia Target Manager [Cited 21.8.2019] https://library.vuforia.com/articles/Solution/Optimizing-Target-Detection-and-Tracking-Stability

17 Chollet,F. 2017. Deep Learning with Python. Manning Publications Company

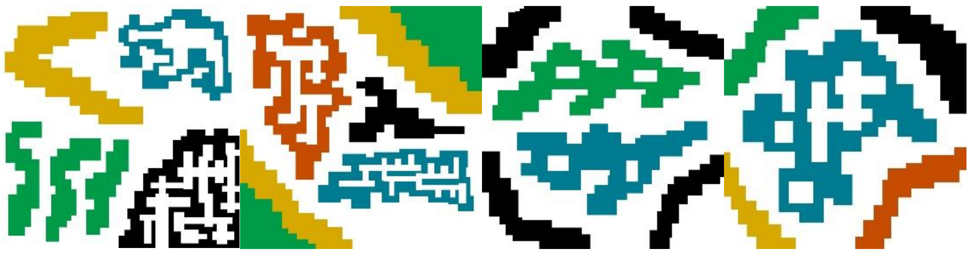
Vuforia Image Targets

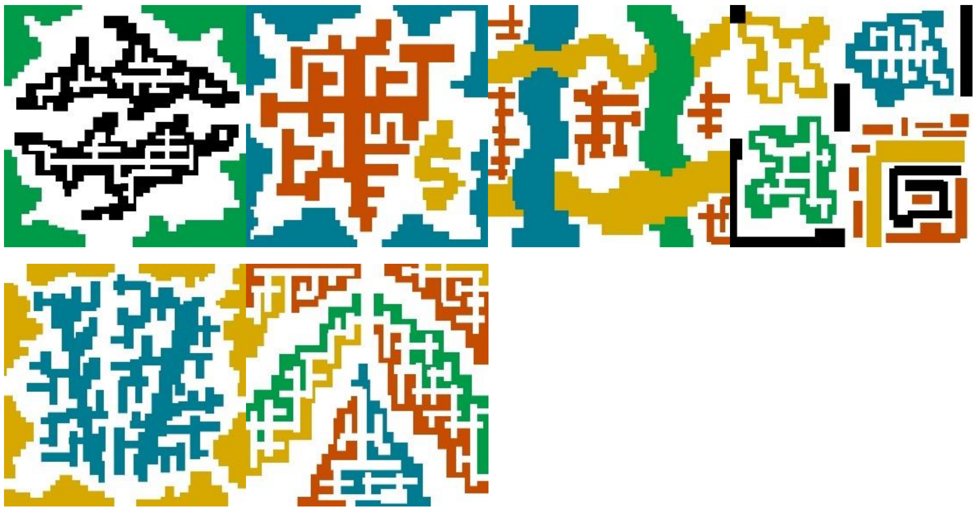
Batch 1 A

Batch 1 B

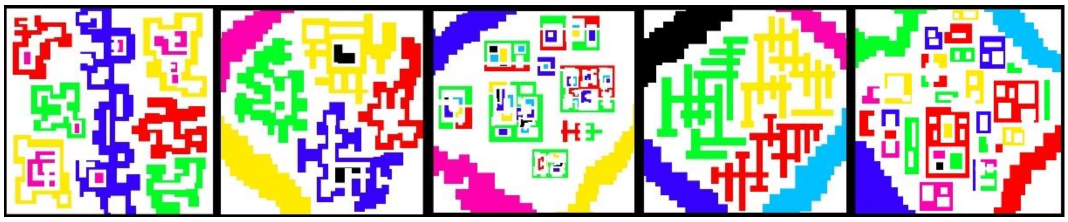


Batch 2.1

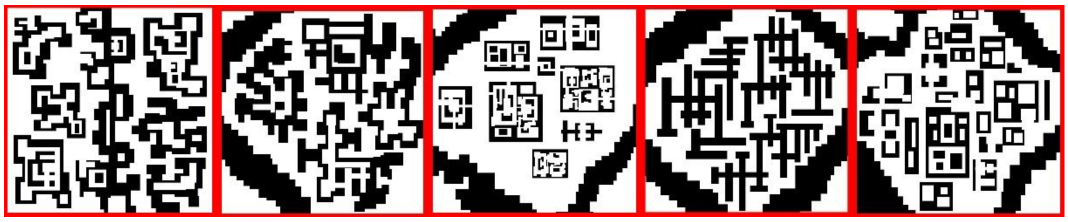




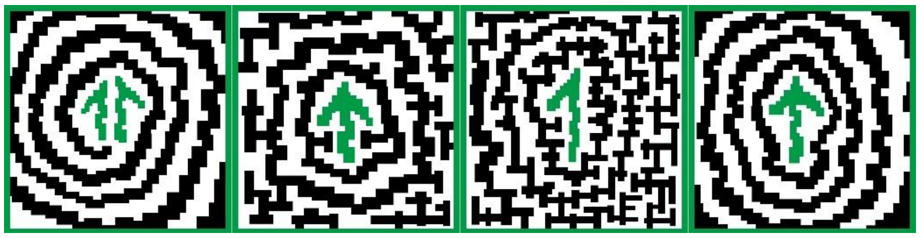
Batch 2.2A



Batch 2.2B



Batch III A



Batch III B