Bachelor Informatica

University of Amsterdam

Projection-Based Interaction using Tangible Objects

Aaron Metzelaar

January 26, 2024

Supervisor(s): Dr. R.G. Belleman

Abstract

This thesis presents a flexible framework for projection-based interaction in Mixed Reality (MR), utilising tangible objects to merge the physical and digital environments. The framework aims to simplify MR application development in Unity and is structured around the design principles of ease of use, adaptability, and real-time responsiveness. The framework is designed with modularity in mind, allowing for separate use of calibration, object initialisation, and object detection components. Additionally, the framework provides classes for initialised and detected objects. A demonstrative RGB colour mixing application is used to showcase the framework's effectiveness and adaptability. The framework's reliability is limited by its dependence on RGB data, which enhances accessibility for varying setups but becomes unreliable in changing lighting conditions or when using similarly shaded colours. Future research could address this issue by incorporating depth image data.

Contents

CHAPTER 1

Introduction

Mixed Reality (MR) aims to blend the physical and digital worlds together by combining physical interactions with the versatility of computing power, ultimately leading to experiences that are both intuitive and seamlessly integrated [4]. When implemented thoughtfully, these types of integration can enrich our interactions with the world around us in unprecedented ways. It is crucial to differentiate this form of reality from other closely related technologies, namely Virtual Reality (VR) and Augmented Reality (AR). According to Milgram and Kishino [10], we can represent all these technologies in a "virtuality continuum", where AR and MR are situated between the real and virtual environments. AR can be considered a subset of MR, located closer to the real world, as it overlays digital information onto a physical environment but does not allow for physical interactions. VR can be viewed as a completely virtual environment, visually transforming the user's surroundings and providing a means to engage with its environment. Nonetheless, both of these technologies lack the capability to interact with physical objects that MR can offer. Unlike AR, which merely projects a digital element onto the real world, MR can enable both a physical object and a digital counterpart to coexist at the same time, essentially producing a "digital double" of the physical object. Interacting with the physical object then converts to identical interaction with the digital version.

By using a central MR system comprised of a camera and a projector, virtually any surface can be transformed into an interactive environment. This significantly increases MR's scalability for public installations while simultaneously reducing the barrier of entry for users. Due to these factors, MR is often considered an excellent medium for both educational and entertainment purposes [7, 2]. No specialised equipment is required on the user's side, making it suitable for demonstrations such as in the project Kreyon City [8]. In this application, LEGO blocks served as literal "building blocks" for a city. The positions of these blocks determined how specific parameters of the city functioned, subsequently impacting the application's results. Using these non-electronic and simple tangible objects as points of interaction allows the MR system to create a highly accessible and engaging experience that is intuitive to interact with. This intuitiveness is essential in achieving a believable experience, as virtual interactions should closely mirror real ones [6]. When this is achieved, the physical and virtual realities can work seamlessly together. One of the main obstacles to creating a realistic VR experience is the lack of physical, hands-on object interaction [5]. However, this problem can be solved in MR by using tangible objects detected by a camera and incorporated into the projector and system.

Continuing from this understanding of the distinctive capabilities of MR, this research aims to address an under-explored but crucial aspect: the development of a flexible framework that can enable both the creation and execution of custom projection-based interaction applications. By developing such a framework, the objective is to enhance accessibility to the field of MR for both creators and developers. By giving users the starting point of the proposed framework, the aim is to substantially reduce the time and technical challenges associated with setting up projection-based interaction applications. Reducing development complexity is crucial in en-

couraging innovation in the field of MR. A more accessible developing environment opens up the door to a wider audience of potential creators without being held back by the lack of extensive programming experience. The research question for this thesis is as follows: How can a system be designed with which users can implement a projection-based interaction application based on tangible objects placed in the field of the projection? To answer this question, we can split it up into two sub-questions:

1. How can the software architecture be designed to effectively support the development of interactive, projection-based applications within a Mixed Reality environment? This sub-question emphasises the need for a software framework that accommodates the functionalities of MR in a logical and effective way, with a focus on simplifying MR development and enhancing user experience.

2. How can the system be designed to be generic and adaptable for various types of projection-based interaction applications involving tangible objects within a Mixed Reality environment? This sub-question seeks to investigate the design choices necessary for creating a framework that allows for a wide variety of MR applications.

The structure of this thesis is designed to approach these questions methodically. Therefore, the theoretical background necessary to understand the complexities and potential of MR technology will first be examined. Following this will be the system design and the implementation of this design. A demonstrative application is created to show the framework in action in the results chapter. Lastly, the results of the research questions are discussed, and a conclusion is made.

CHAPTER 2

Theoretical background

2.1 Fundamentals of Mixed Reality

Mixed reality (MR), as a concept, emerged from the intersection of the physical and digital world, creating a blended environment containing elements from both realities. This idea was first distinctly defined in the works of researchers like Milgram and Kishino [10], which has laid the groundwork for many technological advancements in the field. Within this widely acknowledged paper, the idea of representing forms of reality within a "Virtuality Continuum" is proposed. Within this representation, as can be seen in Figure 2.1, this continuum illustrates a spectrum, ranging from the completely real environment at one end to a fully virtual environment at the other. Real environments refer to the unmodified, physical world we know around us, as can be seen without the help of any digital devices. On the right side of the figure, real environments are defined as an entirely digital world consisting of only simulated objects. VR technology falls in this category, as here, the user also completely emerges into the virtual world displayed within (most commonly) a headset. Therefore, MR can be best described as any combination of real-world and virtual-world environments or objects within a single display.

Both Augmented Reality (AR) and Augmented Virtuality (AV) are subsets of MR, but they have distinct properties that explain their placement on the spectrum. According to Milgram and Kishino [10], AR is more closely related to our real-world environment and only augments it by displaying virtual objects on top of it. A modern-day example of this is Google Maps' Live View feature. It uses this technology to display digital directions to a given location on top of the live camera feed from the user's camera [12]. On the other hand, AV consists of a virtual environment enhanced with real-world elements. This can be useful to enhance the experience within the virtual world by adding interactivity and tangibility through the use of the physical elements. An example of this is flight simulators, which consist of real cockpit controls and instruments combined with a virtual environment to simulate flying more realistically.

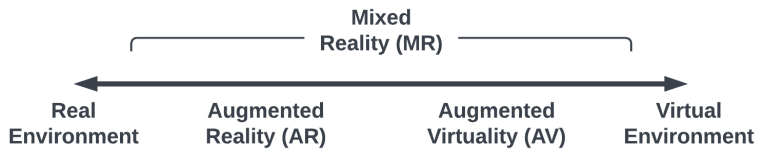


Figure 2.1: The Virtuality Continuum, a spectrum ranging from real environments to virtual environments. This continuum highlights the positioning of AR, AV and VR and depicts the gradual transition from the real world to digital environments.

2.2 Projection-Based MR Technology

Projectors can be used to overlay digital information onto a real-world environment, eliminating the need for head-mounted displays typically used in AR and VR applications. This makes the environment more accessible to a wider audience. The system manages the visualisation centrally, creating a shared digital environment. All users in this environment will view the same augmented space. Therefore, there is no need to synchronise individual devices per user. This setup enables multiple users to interact simultaneously within the same space.

One of the main challenges in projection-based MR is calibrating and aligning the system and its projection. Achieving this as precisely as possible is crucial for creating a believable and immersive experience. Therefore, it is essential to calibrate digital projections accurately with physical objects. This is especially important in systems where users can interact with the objects within the application, changing their position and rotation, as their actions must be reflected in the corresponding digital projection in real-time. A chequerboard pattern is commonly used to calibrate a projector and camera. This pattern is projected over the entire width and height of the projected area, allowing for the precise location of all square intersections [11]. Because all squares in the pattern are of equal size, a projection of it can show any distortions the camera or surface might have. The data set of square intersections that is produced using this method can be used to generate a camera matrix and distortion coefficients that can transform images to be shown in the correct perspective.

In the field of education, projection-based MR has shown considerable potential. Research by Yuan et al. [15], for example, shows how this technology can be used to transform educational environments. In their study, information is displayed on top of physical objects, with a simulation taking place around these objects. This approach creates a more engaging and visually appealing experience to convey the learning material.

2.3 Role of Tangible Objects in MR

Physical, tangible objects in the context of MR deliver a way to interact with an application in a natural way, bridging the gap between the digital and physical aspects of the environment. Because users interact with actual physical objects, no physics has to be simulated for these objects, making the interactive system less expensive to render and more intuitive to users. The objects coexist in the real and digital world at the same time, with the latter version being a "digital twin" of the actual object. This virtual object must closely follow all movements and rotations of the real object to maintain this coexistence; otherwise, the application will not correctly base its following steps on accurate data.

In educational settings, applying tangible objects within MR has been used to show that it can help with numeric comprehension for children [13]. In this research, a simple interface and objects are used to help preschool children understand the relations between the sizes of the objects, their numerical representation and some basic numerical competencies. This approach underlines the potential of MR within the field of education, offering intuitive, hands-on experiences that can lead to a deeper understanding of the material.

CHAPTER 3

System Design

The design of an MR environment heavily influences the user's experience and immersion. Beginning with a high-level overview, this chapter will gradually go down to specific design details. The reasoning behind certain design choices will be explained to highlight their contribution to the overall goal of creating an intuitive, easily accessible and efficient system for both application implementation and user interaction.

3.1 System Architecture

The system architecture of the created framework serves as both an outline of the design used in this research and a blueprint for how a system should appear when using the given framework. Therefore, it is crucial to keep the overall design as simple as possible for easy comprehension and implementation. The hardware components layout, depicted in Figure 3.1, consists of three major parts: the camera, projector, and tangible objects. Note that the camera's field of view fully encompasses the projected field, allowing the entire area to function as a detection zone for tangible objects. The projector is positioned almost perpendicular to the longer edge of the field to counteract the vertical projector offset commonly found in modern-day projectors, such as the one used in the creation of this framework. Cameras, however, are not typically designed with this offset. Therefore, it is placed centrally above the projected field to maximise coverage.

Environmental factors should be considered when setting up the MR system to ensure that the system can properly function. Bright or inconsistent lighting can cause issues in detecting edges of the projected field or objects due to a lack of contrast or an overload of reflections. It is also recommended to ensure that the positioning of the projector and camera remain stationary during the entire duration to ensure the calibration remains accurate.

Regarding the projector, it is important to consider the position for which it is designed. Attempting to project downwards using a projector designed for horizontal projection can result in system overheating. Another solution for this could be to point both the projector and sensors to a wall instead while using magnetic objects to function as tangible objects in the MR experience.

3.2 Framework Design

To create a framework that aligns with our research goal, we must establish some guiding principles. The framework should prioritise ease of use, adaptability, and effectiveness, which are kept in mind throughout the framework design.

The Unity game engine is one of the most popular and beginner-friendly game engines currently available [3]. To account for the goal of simplifying MR development, the framework is developed using this engine and its programming language, C#. As this framework revolves around linking

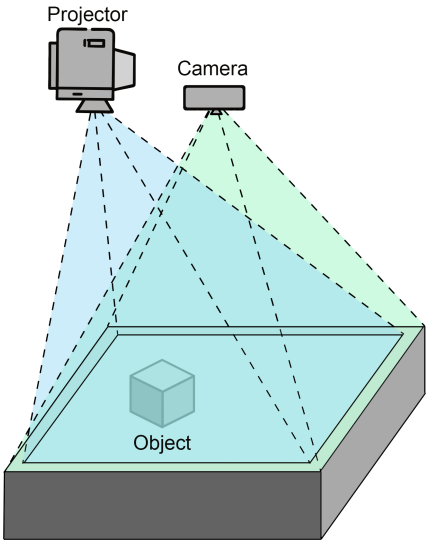


Figure 3.1: A schematic overview of the MR system architecture, showing the spatial arrangement of the projector, camera and tangible object. The dashed lines indicate the projection of the projector and the detection field of the camera. The computing system that compiles the data from the camera and calculates the actual data to be projected has deliberately been left out of this figure to focus on the spatial configuration of the system.

physical objects to digital counterparts, a programming language that supports object-oriented programming like C# is a suitable choice. The Unity game engine has been widely adopted due to its accessibility for both beginning and experienced developers, aligning with our principle of ease of use. To support developers of all levels, different levels of customisation will be offered in the framework's structure. Basic functionalities can be easily modified through checkboxes, numerical inputs, and sliders in Unity's inspector menu without requiring any coding knowledge. However, more structural or functional changes can be implemented through code. The framework categorises all functionality based on its function, making it modular. This modularity is not a design feature but a fundamental aspect of the framework's setup; all steps can also be used independently. The following subsections will cover these modular components in order of appearance in the workflow of the MR framework.

3.2.1 System Calibration

The first step of using the MR framework is the calibration of the system's hardware components. Real-world coordinates captured by the camera are mapped to virtual points in the projection with the goal of precisely aligning the physical and digital spaces. In order to view objects in every location within the camera field the same, no matter the perspective, we need to apply a perspective transformation to the image drawn from the camera. After determining the corners of the rectangle (source points), we can map these points to the corners of the projection itself (destination points) using the following matrix equation:

$$\begin{bmatrix} d_1 \\ d_2 \\ z \end{bmatrix} = \begin{bmatrix} s_1 \\ s_2 \\ 1 \end{bmatrix} \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix}.$$

In this equation, the elements denoted with s are the dimensions of a source point and d for a destination point. These dimensions of the destination point must be divided by the scale factor, z, to acquire the correct Cartesian point. The actual perspective transformation matrix can be seen on the right. After this transformation is performed, the aforementioned chequerboard calibration method is used to account for any distortions in the camera or surface being projected on.

The calibration process is done automatically, and the user gets a visual indication of the projection area found with the camera, together with some guidance for more accurate calibration and how to navigate further. The user can recalibrate or continue to the next step by pressing a single button on the keyboard. This way, minimal user input is required, improving the ease of use.

3.2.2 Object Initialisation

Initialising objects in a way that follows the design principles set at the start of this section leads to an interface closely resembling the calibration step; guidance is shown in text form on the screen, informing the user how to initialise the object and progress further. When an object is recognised within the calibrated area, a colour gets projected onto it, visually informing the user about the detected shape of the object. If the user wants to choose a colour to project onto the object themselves, this can be done in the Inspector tab in Unity of the object; otherwise, a contrasting colour will be chosen automatically. Information about the object, like its colour and shape, must be saved in a way that can efficiently be referenced and compared to later possible occurrences of the same object in the application runtime.

Depending on the user's needs, this step can be done every time the application gets started or the data can be saved after initialising once to use in other runs later. This feature can save users time and enhance convenience, especially when the same objects are used every time. The same is the case for the system calibration, but it is strongly recommended to do this every time the application is started up, as smaller changes can lead to offsets in calibration.

3.2.3 Object Recognition

To make a projection-based interaction application engaging, the framework's object recognition plays a crucial role. Effectiveness, one of our principles in this design, must be addressed through response time and the ability to track objects correctly. The recognition, tracking and projecting of objects must happen as close to real-time as possible to ensure the MR experience is as believable as possible. Each object's position and rotation must be tracked at user-defined intervals and saved so that only one digital object can represent each physical object. When this is achieved, the user can put their full attention towards the application in progress.

3.2.4 Application architecture

Once the objects have been initialised, saved and detected, the focus can shift to the core of the MR experience: the application itself. The application is designed around an event-based architecture which ensures that changes occur only when triggered by a specific event. User-created scripts listen for and handle events related to the objects. These scripts are linked to objects during their initialisation and are configured to take action when a condition is met. An alternative application architecture design could be an update-based model, as used during the object recognition process. However, this architecture is inefficient due to the continuous monitoring even in the absence of changes. Additionally, this approach is less scalable as performance is impacted by the introduction of more events, as each action triggers on every update.

3.3 Data Flow

The framework's data flow is designed to create a functional MR system efficiently. Figure 3.2 illustrates the flow of the data, which is divided into various processes and stages. The first step is to calibrate the RGB data captured by the camera if it has not been done yet. Otherwise, this step can be omitted. By processing the data to detect only initialised objects, we can utilise the coordinate mapping resulting from the calibration process to determine the precise location of physical objects in both the physical world and the digital representation. This information enables us to generate a virtual object that takes on the same position as its physical counterpart. Combining these elements creates a digital replica of the environment within the projected area,

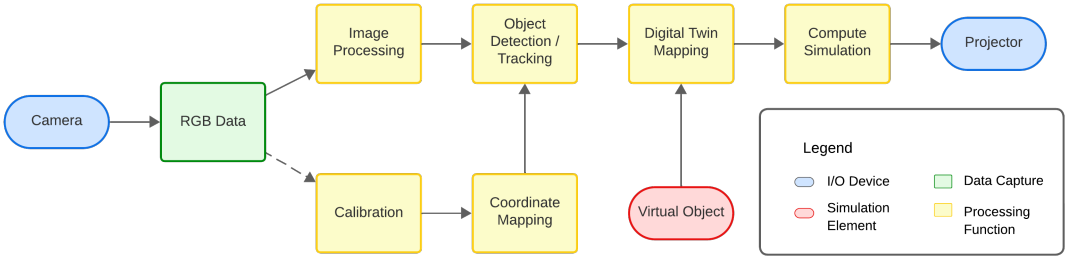


Figure 3.2: The flow of data through the MR system. The dotted line represents a process that is only performed if it has not been performed yet. All other processes are performed during the entire application's runtime.

which can be further used in the application. The result is then projected using the projector again, and the process repeats.

CHAPTER 4

Implementation

In this chapter, we transform the abstract ideas from the design chapter into a functional, tangible system. During this implementation, the design fundamentals and research goals are kept in mind. The framework's code is open-source and available at https://github.com/AaronMetzelaar/MRFramework. This GitHub repository is set up as a template repository, making it easier to create a repository based on the created framework as a starting point. All features discussed in this chapter are available in this framework and can be used by following the steps outlined in the provided README.md file. For a detailed guide on using the framework and demonstration application, please refer to Appendix A.

4.1 Hardware Implementation

In implementing the framework, multiple types of hardware were tested to ensure that the system works in different hardware environments. An Azure Kinect DK camera was mostly used for testing, but a smartphone camera was also tested to confirm that the system works with multiple camera setups. The Kinect camera was initialised at a resolution of 720 by 1280 pixels. This resolution is sufficient for detecting the shape and colour of objects as long as the projected field fills up most of the camera's field of view. However, when the projected area is a smaller portion of what the camera sees, shape-based object detection becomes more challenging, requiring a higher resolution. This, in turn, comes at the cost of the need for more processing power.

The framework was created using an Azure Kinect DK camera which provides depth data. Therefore, the possibility of combining both depth and RGB data was considered. However, due to time constraints and a lack of accessible linking of these two data types, this idea was abandoned.

The application was projected using a Mitsubishi XD500U-ST. This model was chosen because it was the only available projector at hand while creating this framework. Other display types, such as monitors, have also been tested and have been shown to function equally well, if not better.

The setup was assembled using two tripods and lashing straps, as shown in Figure 4.1. This setup is suboptimal, as the camera is not positioned directly above the projected area. Therefore, a smaller portion of the camera's field of view is filled with data we care about, causing a greater loss of image quality since we need to zoom in more. Furthermore, because the camera looks at the projection field at an angle, taller objects can block the area behind them. Due to no optimal way to mount either the camera or projector to a higher point in the room, this setup was chosen anyway.

Figure 4.1: The hardware setup of the system used to create the MR framework. Lashing straps are used to firmly attach the projector to the tripod. The camera is set up on a separate tripod. This allows for easy adjustment of the hardware's position and height when needed.

4.2 Software Implementation

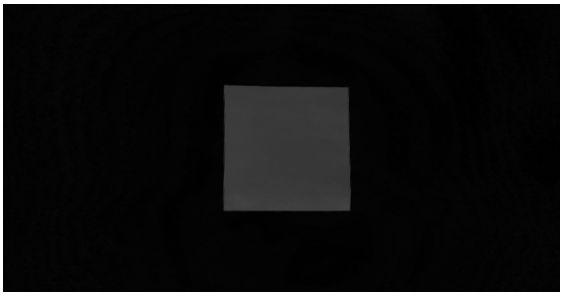
The software and development environment used to develop this framework was chosen for its stability and long-term support. The latest Long-Term Support (LTS) version of Unity, version 2022.3.16f1, was selected as this is the recommended version to use for maximum stability and support [9]. For image processing and object recognition tasks, a Unity package implementing OpenCV functions in C# was used. This package must be installed for the framework to work, as is mentioned and linked to in the README.md file in the GitHub repository.

In the following subsections, we will discuss all the modular components of the framework, as done in the previous chapter. The user is guided through the different stages of the framework when using it by displaying instructions in text form in the projected field. In the calibration phase, we need to align the corners of the camera and projector. This crucial step guarantees that any point detected by the camera will be precisely translated to the same position in the application. Next, we initialise objects to identify the specific shape and colour combinations that should be detected during the application. Lastly, the initialised objects are detected using an object detection algorithm.

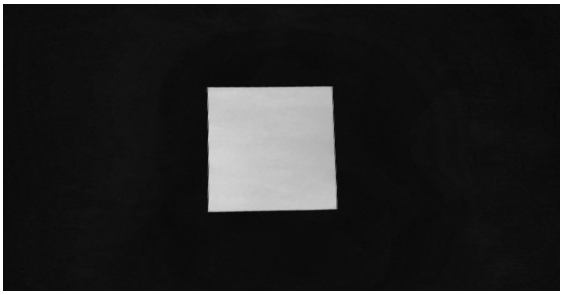
4.2.1 System calibration

To calibrate the camera and projection area, a methodical approach was taken by iterating through threshold bounds to determine the rectangular projection area. As the environment surrounding the projection area can be diverse, and the software does not know how big the area should be, adaptive threshold methods cannot be performed on the camera image to consistently find the rectangle. Therefore, a more manual approach was chosen, singling out one small threshold area at a time to find the largest four-sided contour. This technique may struggle if other rectangular shapes are in view of the camera, but this can be easily dealt with by obscuring part of the edge of the rectangle.

After detecting a rectangle, a transformation matrix is applied to map its corners to the corners of our projection. To account for any surface or camera distortions, a chequerboard pattern is projected within the projection area. OpenCV's `FindChessboardCorners` and `CalibrateCamera`



(a) Greyscale representation of a yellow object.



(b) HSV (Hue, Saturation, Value) greyscale channel of the same yellow object.

Figure 4.2: Comparison of colour representations for object detection. As can be seen, the HSV representation does a better job of highlighting the object from the background, as the focus lies more on the hue and saturation characteristics.

functions are then used to obtain a camera matrix and distortion coefficients, which we use to correct these distortions. Going forward, we will apply the transformation and distortion corrections to all images received from the camera. This will allow us to focus solely on the projection area and ensure that it appears as intended.

A "base image" is saved together with the other calibration data to be referenced later by other functions. This image is an undistorted, cropped version of how the camera views the projection area without any objects that should be detected later. Using this image, we can later find the true difference between the background and the new image captured by the camera. This approach also allows us to place certain props in the projection field without them falsely getting detected for any object.

When thresholding an image, it must first be converted to a greyscale image. This conversion can cause different colours to fade out, making it more difficult to detect objects. To address this issue, we convert the image to the greyscale channel of the HSV (Hue, Saturation, Value) representation of the image. Figure 4.2 illustrates the significant difference this can make for objects with paler colours.

4.2.2 Object initialisation

The initial step in object initialisation is determining the shape of the object. To achieve this, the saved base image is subtracted from the new image containing only the object. This eliminates any consistent details in the projected area, isolating the object itself. The resulting image of this subtraction is then subjected to various operations to enhance the object's visibility. First, a bilateral filter is applied to the image. This filter is known for reducing noise while preserving the edges in an image [14], which is essential for accurate edge detection. Subsequently, Canny edge detection is used to outline the object [1]. As no other edges can appear during this process, unlike in the calibration step, Canny edge detection has a high probability of successfully identifying the object's outline. OpenCV's `MorphologyEx` function is used to perform some ad-

vanced morphological transformation on the result of the edge detection. This refines the lines and removes any small imperfections, resulting in a cleaner representation of the object's shape. This makes it easier to detect the object.

The detection of the object is done using methods similar to those used for identifying the projection area. However, the detection is not limited to four-sided contours. Instead, a convex hull algorithm is applied to combine all grouped contours found, as there is a possibility that multiple smaller contours are found instead of a singular contour for the entire object. This approach ensures the entire object is captured while often reducing the complexity of the contour, but has the downside that no concave objects can be used.

Once an object has been correctly identified, its contour is normalised to ensure consistency in its representation. This is achieved by finding the minimum area that encloses the contour and aligning this rectangle horizontally. The initialised object saves two different hues: the hues of white and the chosen colour projected onto the centre of the physical object. These hues represent how the object could appear and can be compared to later. The normalised contour, along with the two hues, the colour to be projected onto the object, and an optional name, are saved to a ScriptableObject in Unity. This allows for easy reference to the data when comparing contours to initialised objects, as well as the ability to save an object's data between sessions.

The final stage of object initialisation involves providing feedback to the user. In this system, we provide this feedback by projecting the identified contour onto the detected object. The object's real-world position is converted to its corresponding location on the canvas in Unity. A mesh is subsequently created based on the contour, which is positioned at the angle determined during the normalisation process.

It is important to note the difference between the coordinate systems of Unity and OpenCV. Unity uses a traditional Cartesian coordinate system with the origin at the bottom left-hand corner, whereas OpenCV considers the origin to be at the top left-hand corner, resulting in an increasing downward y-axis. The latter coordinate system is more commonly used in computer graphics, where positions are set using screen coordinates. Due to this difference, it is necessary to transform all coordinates before they can be accurately represented in the Unity environment. This transformation involves inverting the y-axis values after processing them with OpenCV but before rendering them in Unity.

4.2.3 Object detection

When detecting objects in a running application, the user can adjust the detection strictness using sliders. These sliders, as shown in Figure 4.3, affect the detection and comparison of initialised and detected objects. Changing values allows users to configure the framework to function as needed for their respective use cases. These sliders follow our design principle of making the program accessible to different kinds of users, as they require no code input to operate.

At each interval, the "expected" image, which is the still frame of the application placed on the base image, is compared to the actual picture. Any differences between the two can be attributed to the objects. This image, which contains the differences, is processed using the same steps as during the initialisation of the objects. Subsequently, each detected shape is compared to each initialised object to determine whether they match shape and hue. The detected shapes are arranged in descending order based on their contour area. This allows us to stop searching for objects when the area is less than the smallest contour area of the initialised objects.

When a detected object matches an initialised object, we generate an identifier based on its location and contour length. If this identifier closely matches another identifier saved in the active object dictionary, we update the position and rotation of the saved object. If the identifier is not yet known, we save it in the dictionary and create a new GameObject for this object. The initialised object's contour is applied to the new object, along with the detected object's position

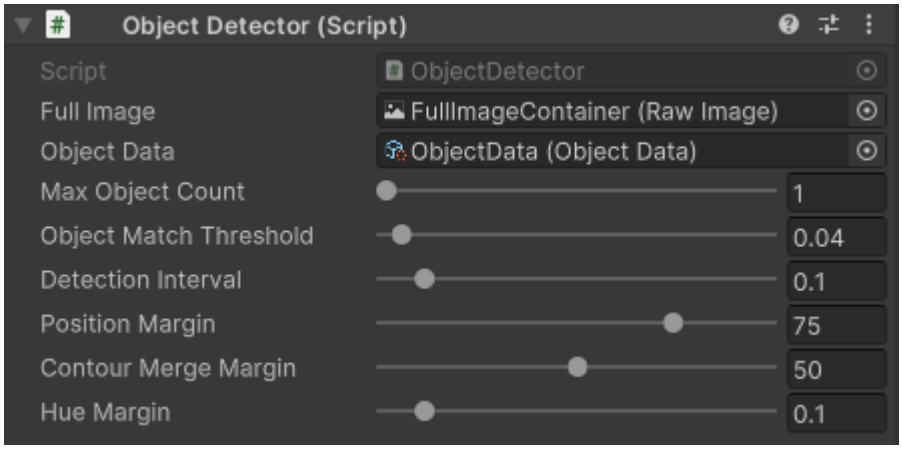


Figure 4.3: The available sliders on the object detector script, as shown in the Unity editor. Using these inputs, the user can easily change the strictness of certain parts of the detection algorithm without the need for any code. Hovering over these sliders reveals a tooltip explaining what the input means in more detail and what changing the value translates to in the detection of objects.

and rotation. Any entry in the active objects dictionary that is not called within a certain interval is removed, as the object is no longer in that location and should not be displayed anymore. The corresponding GameObject is also deleted.

CHAPTER 5

Results

A demonstration application was created to showcase the framework's functionality and provide an example of how the framework could be used. The creation of this application served as a feedback loop on the functionality of the framework, emphasising which parts of the framework needed improvement or change in order to function as expected. The following sections will outline the aim of this application, followed by some highlighted points as to why this was chosen.

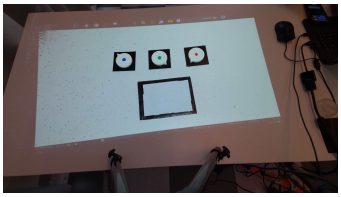
5.1 RGB Color Mixing Application

The goal of the application is to display as much of the framework's functionality as possible while highlighting its features. With this in mind, we have created a demonstrative application that uses tangible objects to visualise RGB colour combinations. In this application, the four objects visible in Figure 5.1 are initialised. Three of the objects have the same shape and are therefore differentiated by the colour in their centre. This demonstrates how the framework can distinguish objects based on multiple factors. Turning a knob upwards increases the respective red, green or blue colour value to increase, resulting in a different colour being displayed on the fourth object.

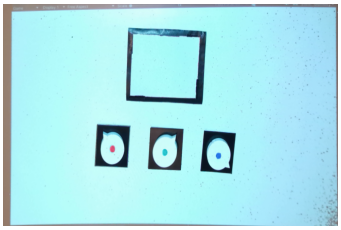
All the objects are white, as we quickly learned that it is difficult to detect a colour projected onto a different coloured object. Since the table on which the objects are placed is also white, we placed some black squares in the base image to make the objects stand out in the detection process. These black squares, as can be seen in Figure 5.1c, have a lower middle section where the knobs can be placed. This allows the knobs to rotate freely without changing their position. A colour is drawn in the centre of each knob. This allows them to be distinguished during the object detection process, even though they have the same shape.

The object used to visualise the RGB colour combination has been configured to skip the hue-matching step in the object detection process. This decision was made due to inconsistencies between the image capture and the projection of the new RGB colour combination onto the object. An attempt was made to replace one of the saved hue values on the initialised object with the newly calculated colour on every update in order to match with the detected colour, but it was unsuccessful. However, this is not an issue in the context of this demonstrative application, as it does not affect the outcome of the application.

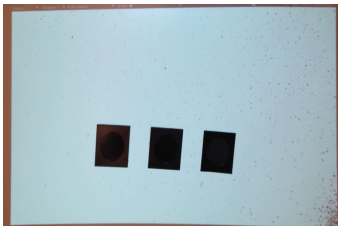
Object detection occurs at defined intervals while the application operates on an event-driven basis. Updates in the application only occur based on specific events, in this case, a detected change in the rotation of an object. This approach ensures that no unnecessary calculations are performed using similar or identical repetitive data, resulting in a faster system.



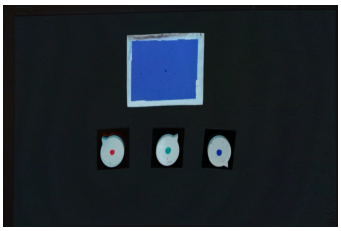
(a) Original image captured by the camera.



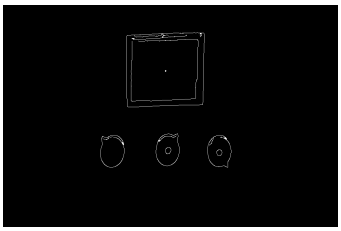
(b) The image after undistorting and cropping.



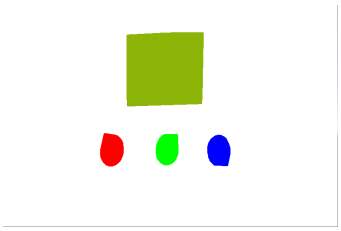
(c) The base image.



(d) The result of subtracting the base image from the undistorted and cropped image.



(e) The transformed image, emphasizing object edges.



(f) Final visualization from the game, which gets projected back onto the objects.

Figure 5.1: Sequential image processing and rendering stages of the framework, running the demonstrative application.

A false assumption was that objects could be detected consistently during user interaction and throughout the application runtime. Because parts of the object are hidden from the camera when a user interacts with them, for example, a user grabbing the knob to rotate it, the object cannot be recognised during this period. As a consequence, the visualised colour does not change gradually like the rotation of the knob; instead, the colour is updated when the knob is detected in its new rotation. This issue could potentially be solved by creating a design for the knobs that allows them to be interacted with without blocking their top-down shape. Another solution could be to use positional trackers placed at both the centre and the point of the knob. In this case, the rotation can be determined by the angle of the line drawn between the two points in the same way it is currently calculated.

CHAPTER 6

Discussion

In this thesis, we have presented a framework for projection-based interaction using tangible objects within a Mixed Reality (MR) environment. The main objective of the framework, as illustrated by the research question and its sub-questions, is to offer an effective but adaptable platform for developing MR applications. The developed MR framework delivers on this by allowing users ways to create an application without worrying about external factors such as calibration, object initialisation or detection. This ensures that the users can focus on creating the MR experience itself. By offering ways to fine-tune the underlying processes using intuitive sliders and other inputs, no coding experience is needed in order for the system to function under different circumstances. Adaptability has been one of the key principles during the creation of the framework; the design and implementation allow for the configuration of any object and the ability to use or alter any part of the framework separately.

Despite the achievements, the framework has several limitations. One of the challenges we encountered while using the framework was the resolution constraint of the Azure Kinect DK camera used during testing. Due to the camera being locked to a 720p resolution and the need to crop into the image, inconsistencies in object recognition and tracking arose. This limitation highlights the need for a quality camera positioned in a location that minimises the space captured around the projected area. Additionally, the system may struggle to track objects that blend in with the background, are partially obstructed, or are affected by significant changes in lighting conditions after calibration.

Further research could explore the integration of depth data to cross-check RGB data in object initialisation and detection or eliminate the need for RGB data in shape detection altogether. This would address the issue of not being able to detect objects with a similar colour to the background, as well as changes in lighting conditions, as depth data is not affected by these factors. Another potential addition is the use of infrared data for gesture recognition, enabling users to interact with the MR environment through specific hand or body poses or movements. This would provide additional interaction options without the need for any initialisation.

6.1 Conclusion

The created framework addresses the need for a practical and adaptable platform for MR application development. The objective of the research has been achieved, as the framework functions as promised. This research provides a more in-depth explanation of certain choices and structures found in the project. The usage of this framework could give creators of future MR applications a head start in their development, offering a solid foundation upon which to build. The created framework fulfils its purpose when it becomes invisible, immersing users in the experience it creates without drawing attention to the underlying complexities.

Bibliography

[1] John Canny. "A Computational Approach to Edge Detection". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-8.6 (1986), pp. 679–698. DOI: 10.1109/TPAMI.1986.4767851.

[2] Adrian David Cheok et al. "Mixed reality entertainment and art". In: *International Journal of Virtual Reality* 8.2 (2009), pp. 83–90. DOI: 10.20870/IJVR.2009.8.2.2729.

[3] Eleftheria Christopoulou and Stelios Xinogalos. "Overview and Comparative Analysis of Game Engines for Desktop and Mobile Devices". In: *International Journal of Serious Games* 4.4 (Dec. 2017). DOI: 10.17083/ijsg.v4i4.194.

[4] Céline Coutrix and Laurence Nigay. "Mixed Reality: A Model of Mixed Interaction". In: *Proceedings of the Working Conference on Advanced Visual Interfaces*. AVI '06. Venezia, Italy: Association for Computing Machinery, 2006, pp. 43–50. ISBN: 1595933530. DOI: 10.1145/1133265.1133274.

[5] Elham Ebrahimi et al. "An empirical evaluation of visuo-haptic feedback on physical reaching behaviors during 3D interaction in real and immersive virtual environments". In: *ACM Transactions on Applied Perception (TAP)* 13.4 (2016), pp. 1–21. DOI: 10.1145/2947617.

[6] Wen Fuan and Cao Gang. "Selection of Terminal Device for Virtual Simulation Experiment". In: *2020 5th International STEM Education Conference (iSTEM-Ed)*. IEEE. 2020, pp. 38–41. DOI: 10.1109/iSTEM-Ed50324.2020.9332616.

[7] Charles E Hughes et al. "Mixed reality in education, entertainment, and training". In: *IEEE computer graphics and applications* 25.6 (2005), pp. 24–30. DOI: 10.1109/MCG.2005.139.

[8] *KREYON CITY - Build Your Ideal City.* URL: http://whatif.cslparis.com/kreyon.html.

[9] *Lts vs. Tech Stream: Choose the Right Unity Release for you.* URL: https://unity.com/releases/lts-vs-tech-stream#:~:text=What%20version%20do%20you%20recommend,about%20our%20different%20releases%20here..

[10] Paul Milgram and Fumio Kishino. "A taxonomy of mixed reality visual displays". In: *IEICE TRANSACTIONS on Information and Systems* 77.12 (1994), pp. 1321–1329. ISSN: 0916-8532.

[11] Daniel Moreno and Gabriel Taubin. "Simple, Accurate, and Robust Projector-Camera Calibration". In: *2012 Second International Conference on 3D Imaging, Modeling, Processing, Visualization Transmission*. 2012, pp. 464–471. DOI: 10.1109/3DIMPVT.2012.77.

[12] Chris Phillips. *New ways Maps is getting more immersive and sustainable.* 2023. URL: https://blog.google/products/maps/sustainable-immersive-maps-announcements/.

[13] Taciana Pontual Falcão et al. "Tangible Tens: Evaluating a Training of Basic Numerical Competencies with an Interactive Tabletop". In: New York, NY, USA: Association for Computing Machinery, 2018. ISBN: 9781450356206. DOI: 10.1145/3173574.3174125.

[14] Carlo Tomasi and Roberto Manduchi. "Bilateral filtering for gray and color images". In: *Sixth international conference on computer vision (IEEE Cat. No. 98CH36271)*. IEEE. 1998, pp. 839–846. DOI: 10.1109/ICCV.1998.710815.

[15] Qingshu Yuan et al. “TIPTAB: A tangible interactive projection tabletop for virtual experiments”. In: *Computer Applications in Engineering Education* (2022). DOI: 10.1002/cae.22524.

APPENDIX A

Using the MR Framework and Demonstration Application

The framework contains multiple steps to create the MR simulation: calibration, initialization and detection. These steps are also separated in the source files and are called `Calibrator.cs`, `ObjectInitializer.cs` and `ObjectDetector.cs` respectively. The demonstrative application can be found in `RGBDemoApplication.cs`. The following sections will provide a step-by-step guide on using the framework and demonstration application, going more in-depth than the instruction text shown in the framework.

A.1 Requirements

- Unity version 2022.3.16f1 (latest LTS version as of writing)
- OpenCV plus Unity package
- Chequerboard calibration image

A.2 Step-by-step guides

A.2.1 Before starting

1. Click on the `ApplicationManager` object in the `Hierarchy` tab and link the `Calibration Image` parameter found in the `Calibrator (script)` component to the chequerboard calibration image downloaded.
2. Make sure that the entire projection area is clear of any objects and that the camera's view of the projection area is not obstructed.
3. Press `Play` in the Unity editor and wait a few seconds for the calibration process to start automatically.

A.2.2 System calibration

1. If the green rectangle does not match the projection area correctly or if it is not visible at all, please ensure that the camera is able to capture the entire projection area. Additionally, make sure that there is enough contrast between the projection area and the surface underneath. Press the `Spacebar` key to recalibrate.
2. Any objects that should not be detected but should be in the projection area should now be placed in the projection area. Then press the `B` key to set the "base image". Check

if the base image appears as desired in the image showing on the screen. If not, repeat the calibration process without these objects in the projection area, and try again. Make sure the objects in the base image are always present in the same location throughout the runtime of the application.

3. Press the `Enter` key to continue to the following step in the process.

A.2.3 Object Initialisation

1. Place the object that should be initialised in the centre of the projection area to ensure any deviations in corner positions are minimal. Make sure only this object is visible in the projection area. If any objects were placed in the projection area during the base image, make sure these are still in the same location. Press the `Spacebar` key to let the framework find the shape of the object and project this back onto the object. If the shape is incorrect, press the `Spacebar` key again to retry.

2. To change the colour projected onto the object, open the `ApplicationManager` object in the `Hierarchy` tab, go to the `Object Initializer` component and change the `Object Color` parameter. Changes will only apply when the object is reinitialised.

3. Press the `Enter` key to save the object's data and continue.

4. Repeat the steps above for every object that needs to be initialised. Previously initialised objects should be removed from the projection area before initialising a new object. Every object should be linked in the application to the `ApplicationFlowManager.cs` file in a list called `objectsToInitialize`, as can be seen in `RGBDemoApplication.cs`.

A.2.4 Demonstration Application

Strictness of the object detection can be altered using the sliders found on the `Object Detector` component on the `ApplicationManager` object.

1. Press the `Spacebar` key to start the application and object detection.

2. Place all objects in the projection area, ensuring no objects overlap and have enough contrast compared to the base image.

3. Turn a knob upwards to increase the value of their colour; turn a knob downwards to decrease it.