ISSN 2959-6157

Augmented Reality Text Translation: A Unity-Based Real-Time Approach

Youding Yin[1,+ *], Guanzheng Liu[2+], Shiqi Zhang[3+]

[1]Department of Computer Science and Software Engineering, Rose-Halman Institute of Technology, Terre Haute, IN, 47803, United States, 008023you@gmail.com
[2]Department of Information and Communication Engineering, Hubei University of Economics, 430070, 20130223@email.hbue.edu.cn
[3]College of Information Science and Technology, Beijing University of Chemical Technology, Beijing, 100029, China, 2021020171@buct.edu.cn
+These authors contributed equally to this work and should be considered co-first authors.
*Corresponding author email: 008023you@gmail.com

Abstract:

Language barriers have remained as one of the biggest challenges when it comes to global communication. The need for text translation to happen in real time has been on the rise due to the idea of globalization in the past decades. However, there are very few well-developed applications on the market right now to serve the purpose of real-time text translation. This paper presents a Unity-based application that seamlessly translates text in real time in an augmented reality environment. It successfully combined Unity engine and C# scripts with services such as Google Cloud Vision and Google Translate to address the problem of cross-lingual understanding. The application utilizes a pre-trained machine learning model from Google Cloud Vision to recognize and translate texts and pins the translated text in relative positions on the camera as objects. The overall outcome of the application demonstrated the possibility of breaking down text language barriers. It also proved the potential usage of such technology in daily life. Users were able to view translated text objects through the perspective of augmented reality thus enabling a seamless translation experience in various environments.

Keywords: Translation, Augmented Reality, Computer Vision, Machine Learning, Unicode Classification.

1. Introduction

In today's increasingly interconnected world, language continues to pose a significant barrier to effective communication. Despite the growing capabilities of various translation applications, there remain substantial limitations in the realm of text translation. Even with features like photo translation introduced by companies like Google, translation convenience has been greatly improved. Photo translation combines image recognition technology and translation technology to offer a more convenient language translation solution. During its usage, users align their devices with the text they wish to translate, take a picture, and the application uses text recognition to extract text from the image. It then translates the extracted text into the user's chosen language, displaying the translation. This process eliminates the need for manual text input, resulting in intuitive results and significantly enhancing user experience.[1] Existing photo translation applications include Google Translate, Baidu Translate, Youdao Translate, among others. However, despite these advancements, challenges persist in various real-life scenarios. For instance, situations like driving a vehicle may not afford users the time to capture images with their devices, rendering them unable to access translated text. Moreover, while the process of using photo translation has been streamlined to a great extent, it can still be cumbersome during activities like traveling where frequent picture-taking for translation information remains inconvenient.

To address these issues, we require a more convenient and real-time translation solution. After a series of considerations, we've decided to introduce Augmented Reality (AR) technology into the field of translation to create an almost real-time translation effect. As an emerging technology, AR blends virtual and real elements, enabling users to interact with virtual elements in real time while retaining their perception of the real world. We believe that AR technology can be used to translate captured text

in real time, projecting the translated text directly near the original text's location. [2] This approach could potentially be implemented on smart glasses, freeing the user's hands and delivering a real-time translation experience. This paper aims to explore the feasibility, implementation methods, and limitations of AR-based real-time translation.

In this paper, we use a method of capturing images through successive screenshots, extracting text using the Google Vision API, and performing translation using the Google Translate API. The paper will delve into the practical approach of employing these technologies and highlight the limitations of relying solely on these APIs.

2. Method

2.1 . *Overview*

The overall flowchart of the working application is presented in figure 1. The whole project is done with Unity. Once the application is started, the camera will start capturing and sending screenshots to Google Cloud. With a pre-trained machine learning model that recognizes texts in images, Google Cloud is able to provide results with potential languages characters and each of their bounding box coordinates. The application will receive those results and store them in different data structures waiting for parsing. After the parsing, the application then sends those texts that require translation to Google Translate and waits for the response. Google Translate will translate the received texts and return the result to the application. Finally, the application will use C# scripts to spawn text objects on Unity canvas which are visible to the user with the bounding box coordinates and the translated texts. The application will perform the processes mentioned above continuously until the application is eventually stopped by either the user or the system.

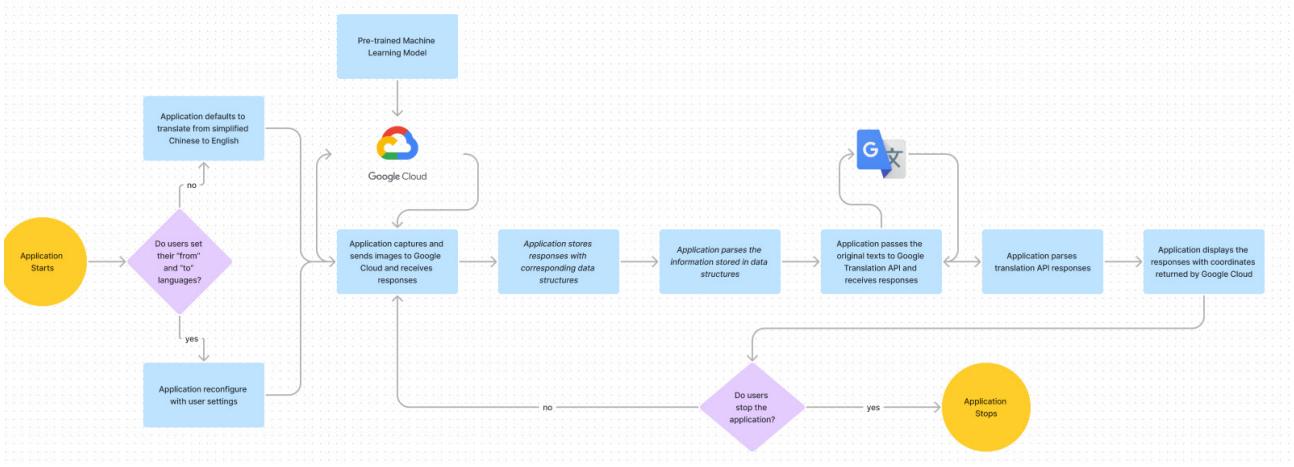


Figure 1 Flowchart of overall application

2.2 . *Application User Interface*

In the current version, the application's user interface contains merely a starting menu whenever the user tries to boot up the application. The starting menu provides all the basic information and configuration regarding the application, this includes application naming title, input and output language settings, both start and quit functionality buttons, and finally a link that directs the user with keywords that can help set the translating languages. For a clear graphic demonstration, refer to the image 2 presented below.

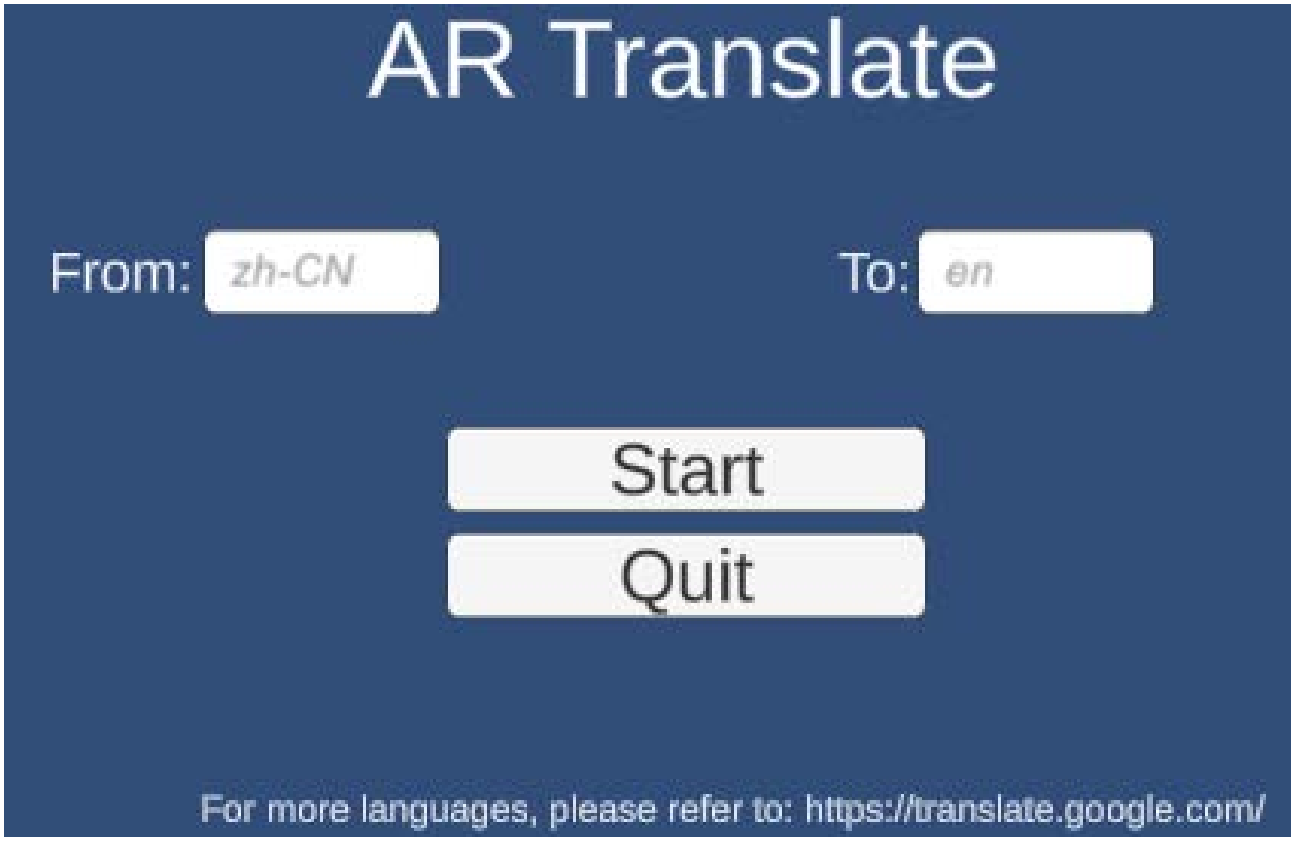


Image 1 Starting Menu User Interface

The input and output languages option prompts the user to enter their desired translation languages. As image 2 has shown, the "From" user input is used for prompting the user to enter the keyword that is wished to be detected and translated. By default, the setting for the "From" input is "zh-CN", which represents Simplified Chinese as the original goal of the project is only for the application to translate Simplified Chinese to other languages of choice. At the current state of the application, the application can only set the "From" languages to CJK languages explicitly due to limitations of the code base. CJK language, in the current context, refers to Chinese, Japanese, and Korean characters. It is a goal in the future for the team to make extensions to the current code base so that the application can support a wider range of "From" input languages. As for the "To" user input, it indicates the configuration of the language that the user wants their texts to be translated to. Different from the "From" user input, the application can support all common languages that are provided in the Google Translated API. The bottom link provided in the starting menu will take the user to Google Translate's website so the user can select the desired language keywords from the web link.

2.3 . *Communication with Google Cloud Vision*

After the user finishes the configuration and enters the main application camera, the application will start taking screenshots of the camera capture and sending them to Google Cloud Vision. Although the application aims for real-time translation, it is still difficult to achieve video result text recognition and extraction. Therefore, we took the approach of letting the application take continuous screenshots as a substitute and hence mimicking frames in a video capture. We then analyze each frame of the video with Google Cloud Vision to constantly take in responses. Once the frame is sent to Google Cloud Vision, a pre-trained machine learning model is then applied to analyze the input. The model is able to recognize different characters from all common languages and send back results. The results, which are compacted in a JSON response, contains the language that the model recognized, the texts extracted from the image, and each corresponding bounding-polygon of texts. This JSON response will be received and stored in our application scripts for further use. It is also worth mentioning that the overall communication process requires a Google Cloud Vision API key to function.

2.4 . *Google Translate API*

Once the application scripts have the extracted texts from the Google Cloud Vision responses, it will then recognize the parts of the response texts that require transla-

tion based on the “From” language user inputs set in the starting menu. This is accomplished by breaking the CJK letters and characters into Unicode blocks that contain graphic characters used specifically in CJK. These blocks are used in Ideographic Description Sequences to describe a potential ideograph which contains graphic sign language characters that are not encoded in the original Unicode system.

In the application’s code base, the team has written an infrastructure that works as a CJK Ideograph identifier. Once a CJK character’s unicode has been detected, the line of text is then flagged as a potential CJK sentence. Since the team doesn’t want the translation process to lose the meaning of possible mixed languages sentences, these flagged texts will all be translated as a regular CJK sentence. This way, the application not only reduced the workload by not having to translate all texts, it also kept the completeness in translated meaning of some mixed sentences

After the application finishes identifying all potential sentences that need to be translated, it will then send those texts to Google Translate API to conduct the actual translation process. Results of the translation will likely be the same as if the translation is done through Google Translate’s online services, all results received from the translation process will be passed on as responses from Google Translate to the application. It is also worth mentioning that in order to reduce the run-time of the application, it is one of the team’s future goals to move the whole translation process to a local server. This can be done by running an application image using services such as Docker, more details will be discussed in the later sections of this report.

2.5 . *Spawning result texts in real time*

One of the final stages of the application flowcharts is to show the translated text over the camera canvas in Unity. This can be achieved in many different ways but the team chose the approach of spawning dummy texts with a prefab text object hidden under the main camera canvas. In Unity, the team set up a prefab text object that has the example fonts, text size, and text color. The team then uses a C# script to spawn other dummy text objects with the input of translated text results received from the Google Translate API. These spawned text objects then have their position adjusted with the bounding-polygon coordinates returned by the Google Cloud Vision API. With the coordinates, these text objects can then be attached to a relative position of the original texts in the image taken by the application.

However, there can be the problem that whenever the user decides to walk away and aim the camera at a different object and try to translate, the previously spawned dummy text objects will then populate the entire camera canvas regardless of the current contents captured by the camera. To solve this problem, the team decided to take the approach of destroying all previous text objects on the frame and start fresh with the new frame. This way, each time translated results are received, a new wave of dummy text objects will be spawned on the screen as the previous text objects are cleared. The user will only see one set of floating translated text objects on the main camera canvas and thus solving our problem, and when no texts are detected, the main camera canvas still destroys all text objects leaving a clean canvas shown to the user.

2.6 . *Scene Analysis*

In real-time or scenario translation, the scenes captured by the device are subject to constant change as the user’s perspective shifts. Consequently, the content requiring translation also varies accordingly. Examples include outdoor scenarios like billboards and road signs, as well as indoor scenarios such as bulletin boards and materials on a desktop. Some scenes involve only a single translation point, such as a sticky note on a desk, while others have multiple instances for translation, such as traffic signs on a highway, where multiple signs may appear at the same intersection. The translated results presented to the user need to consider the correspondence between translated text and its original position. This intuitive correspondence aids users in determining the alignment between translated text and the original content, reducing the likelihood of errors. This aspect is especially crucial for scenarios like highway signage, which demand quick and accurate reading and translation. Various methods can be employed to establish the correspondence between the original and translated texts (positions), such as using the same color for both. In this project, we have employed a method where translated text follows the original content, meaning that the translated text is projected just below the corresponding original text, ensuring utmost clarity.

2.7 . *Temporal Characteristics*

It is important to consider the duration for which a piece of text must be visible before translation is required. Not all text necessarily needs to be translated, as users are often only interested in specific portions. A simple approach is to determine a temporal characteristic for collection: if a particular piece of text is captured three times consecutively, translation is initiated; otherwise, it is skipped.

2.8 . *Unity Implementation of Translation Text Following Process*

The translation process is outlined as follows:

1. Obtain an image through the camera.
2. Determine if there is any text in the image. If text is present, further analyze whether the text requires translation.

3. If translation is required, invoke cloud-based translation software to translate the text, resulting in the corresponding translated text.
4. Calculate the original text's position and project the translated text just below it.

2.9 . *Data structure*

In this project, we utilize `System.Serializable` to store information for each line of text.
System.Serializable is an attribute in C# that indicates that a class or struct can be serialized. Serialization is the process of converting an object's state into a format that can be easily stored or transmitted and later reconstructed.

```
[System.Serializable]
12 个引用
public class TextLineInfo
{
    public string text;
    public Vector2 topLeft;     // coordinates of the top left corner
    public Vector2 topRight;
    public Vector2 bottomLeft;
    public Vector2 bottomRight;
    public bool containsChinese;
    public string translatedText;

    3 个引用
    public TextLineInfo(string text, Vector2 topLeft, Vector2 topRight, Vector2 bottomLeft, Vector2 bottomRight)
    {
        this.text = text;
        this.topLeft = topLeft;
        this.topRight = topRight;
        this.bottomLeft = bottomLeft;
        this.bottomRight = bottomRight;
        containsChinese = false;
        translatedText = "";
    }
}
```

Image 2 TextLineInfo class code

We have defined a `public class TextLineInfo` to store various pieces of information for each line of text. Within this class, we include the following elements:
1. A `string` element named `text` to store the actual text content of the line.
2. A `Vector2` element to store the coordinates of the four vertices of the text bounding box.
(In C#, Vector2 refers to a structure that represents a 2D vector or point in a Cartesian coordinate system. It is part of the System.Numerics namespace)
3. A `bool` element named `containsChinese` to determine whether the text contains CJK characters (Chinese, Japanese, Korean).
4. A `string` element named `translatedText` to store the translated version of the line's text.

To enhance usability within this group, we have implemented a constructor named `TextLineInfo()`. This constructor accepts two parameters: the original text content of a specific line and the coordinates of the four vertices of the text bounding box. Within this constructor, the corresponding elements of the class are initialized by copying the provided values. The two optional elements, `containsChinese` and `translatedText`, are assigned default values if not provided. Specifically, `containsChinese` defaults to `false`, indicating that CJK elements are not present by default. `translatedText` is assigned an initial value of an empty string.

2.1 *0. Algorithm implementation*

The overall process of the algorithm used for processing texts to be translated is presented here in figure 2.

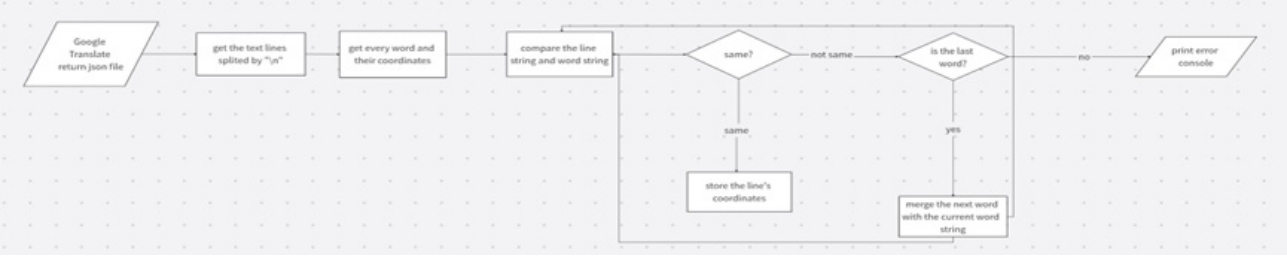


Figure 2 Translation algorithm flowchart

Firstly, we perform preprocessing on the data returned by the Google Translate API. The original content from the API recognition is initially stored in the string `fullText`. The API's response contains data separated by newline characters `\n` to represent different lines. After a series of processing steps, each line of text separated by `\n` is split and stored into a string array named `textLines`. Simultaneously, an empty `List<TextLineInfo>` named `textLineInfos` is created. This list serves the purpose of storing information for each word along with its bounding box.

```
string responses = www.text.Replace("\n", "").Replace(" ", "");
JSONNode res = JSON.Parse(responses);
string fullText = res["responses"][0]["textAnnotations"][0]["description"].ToString().Trim('"');


JSONArray textAnnotations = res["responses"][0]["textAnnotations"].AsArray;

if (fullText != "")
{
    fullText = fullText.Replace("\\n", ";");
    string[] texts = fullText.Split(';');
    List<TextLineInfo> textLineInfos = new List<TextLineInfo>();

    fullText = res["responses"][0]["textAnnotations"][0]["description"].ToString().Trim('"');
    string[] textLines = fullText.Split(new string[] { "\\n" }, StringSplitOptions.None);
```

Image 3 Translated text processing code

Here, within each piece of information returned by the API, the characters of each word along with the coordinates of the four bounding box vertices for that word are stored in a single `textLineInfo` object using a constructor. Subsequently, the `lineInfo` object is added to the `List<TextLineInfo>` named `textLineInfos`.

```
for (int i = 1; i < textAnnotations.Count; i++)
{
    JSONNode textAnnotation = textAnnotations[i];
    string text = textAnnotation["description"];


    JSONNode boundingPolyNode = textAnnotation["boundingPoly"]["vertices"];
    Vector2 topLeft = new Vector2(boundingPolyNode[0]["x"].AsFloat, boundingPolyNode[0]["y"].AsFloat);
    Vector2 topRight = new Vector2(boundingPolyNode[1]["x"].AsFloat, boundingPolyNode[1]["y"].AsFloat);
    Vector2 bottomLeft = new Vector2(boundingPolyNode[3]["x"].AsFloat, boundingPolyNode[3]["y"].AsFloat);
    Vector2 bottomRight = new Vector2(boundingPolyNode[2]["x"].AsFloat, boundingPolyNode[2]["y"].AsFloat);


    TextLineInfo lineInfo = new TextLineInfo(text, topLeft, topRight, bottomLeft, bottomRight);
    lineInfo.containsChinese = containsCJK(text);
    if (lineInfo.containsChinese)
    {
        lineInfo.translatedText = translate(text, InputFieldScript.inputText, OutputFieldScript.inputText);
    }

    textLineInfos.Add(lineInfo);
}
```

Image 4 Bounding ploy box to TexLineInfo code

Here, a new empty `List<TextLineInfo>` named `newTextLineInfos` is created. This list is distinct from the previous one and is intended to store various information for each line.

```
List<TextLineInfo> newTextLineInfos = new List<TextLineInfo>();
textCreator.ClearAllTextOnCanvas();
```

Image 5 List of TextLineInfo class

In this context, the integer variables `aIndex` and `bIndex` respectively represent the indices of the string array `texts` (which stores each line of text) and the string array `textLineInfos[].text` (which stores the text of each word). Both of these indices are initially set to 0, indicating the start of the traversal. If at the current indices, the strings in the two arrays are exactly the same, it signifies that a line contains only a single word. In this case, the information of this word is stored in the `newTextLineInfos` list, which holds information for each line.

```
textCreator.ClearAllTextOnCanvas();
int aIndex = 0;
int bIndex = 0;
while (aIndex < textLines.Length && bIndex < textLineInfos.Count)
{
    if (textLines[aIndex] == textLineInfos[bIndex].text)
    {

        TextLineInfo lineInfo = new TextLineInfo(textLineInfos[bIndex].text, textLineInfos[bIndex].topLeft, textLineInfos[bIndex].topRight,
        textLineInfos[bIndex].bottomLeft, textLineInfos[bIndex].bottomRight);
        lineInfo.containsChinese = textLineInfos[bIndex].containsChinese;
        lineInfo.translatedText = textLineInfos[bIndex].translatedText;
        newTextLineInfos.Add(lineInfo);

        aIndex++;
        bIndex++;
    }
```

Image 6 Checking whether the text should be translated

If the strings are not exactly equal, it indicates that the line contains more than one word. In this case, a loop is used to combine the next word string from the word array with the existing word string. This combined string is then compared with the line information until all corresponding words in that line have been merged. At this point, the top-left and bottom-left coordinates of the first word in the merged set of words are taken as the top-left and bottom-left coordinates of the line, while the top-right and bottom-right coordinates of the last word in the merged set are taken as the corresponding line's top-right and bottom-right coordinates. Since the text within the line has been merged, if the line contains CJK characters, the `translate()` method is called again to translate the text within the line. Subsequently, all the relevant information is stored in the list `newTextLineInfos` for line texts.

```
else
{

    string mergedString = textLineInfos[bIndex].text;
    int tempBIndex = bIndex + 1;
    int originIndex = bIndex;

    while (tempBIndex < textLineInfos.Count)
    {
        mergedString += textLineInfos[tempBIndex].text;

        if (textLines[aIndex] == mergedString)
        {

            TextLineInfo lineInfo = new TextLineInfo(mergedString, textLineInfos[originIndex].topLeft, textLineInfos[tempBIndex].topRight,
            textLineInfos[originIndex].bottomLeft, textLineInfos[tempBIndex].bottomRight);
            lineInfo.containsChinese = containsCJK(mergedString);
            if (lineInfo.containsChinese)
            {
                lineInfo.translatedText = translate(mergedString, InputFieldScript.inputText, OutputFieldScript.inputText);
            }
            newTextLineInfos.Add(lineInfo);
            aIndex++;
            bIndex = tempBIndex + 1;
            break;
        }

        tempBIndex++;
    }
```

Image 7 Checking correspondence with line text

```
if (tempBIndex == textLineInfos.Count)
{
    Debug.Log("Failed to match A[" + aIndex + "]");
    aIndex++;
    bIndex = bIndex + 1;
}
```

Image 8 Error log when no correspondence found

2.1 *1. Translation Recording*

To further expand the application's functionalities and usability, the team chose to integrate a local MySQL database system into the application to record every translation result along with relevant information. This will aid in recording translation history, analyzing data, and providing persistent data storage. Here is an introduction of how the local database is integrated into the system.

2.1 *1.1. Local Database Table Structure*

Before integrating the local database, a suitable database table structure needs to be designed to store translation records. Below is the database table structure used in the system:

Table 1. History Table of Database

Column name	Data type	Comment	Primary
id	int	Primary key	√
source	varchar(255)	Source language	
target	varchar(255)	Target language	
original	varchar(255)	Original text	
translation	varchar(255)	Translated text	
time	timestamp	Translation time	

The history table is designed to store translation history records in the database. Each row represents a record of one translation. The id column is defined as the primary key of the table, indicating that each record has a unique identifier named id. It has a data type of int, which is an integer with a size of four bytes. The source column stores the source language name of the translation, represented by the abbreviation name used in Google Translate, which represents a language and cultural code. For example, Chinese is represented by zh-CN. This column has a data type of varchar(255), allowing storage of text data with a maximum length of 255 characters. The target column stores the target language name of the translation, represented by the abbreviation name used in Google Translate. It is also a varchar(255) column. The original column is used to store the untranslated original text, i.e., the text to be translated. The translation column is used to store the translated text, representing the translation result. The time column is of type timestamp and is used to record the timestamp of the translation operation. This allows you to track when each translation operation is executed.

2.1 *1.2. Database Operation Flow*

The integration process entails a structured workflow of database operations carried out during and after translation instances:

Establish Connection: Firstly, the application will establish a connection with the local MySQL database, employing appropriate credentials and connection parameters such as the name of server, port number, database name, username and password.

Create Insert Statement: The application will get the translation-related information, including source language, target language, original text, translated text, and translation time after using Google Translate. Then, the application will construct an insert statement based on the information.

Execute Database Operation: After all the preparation work is completed, an insert query operation will be executed and a new record will be inserted into the History table.

Close Connection: After completing the database operation, the application will close the connection to the local database to ensure resources are appropriately released.

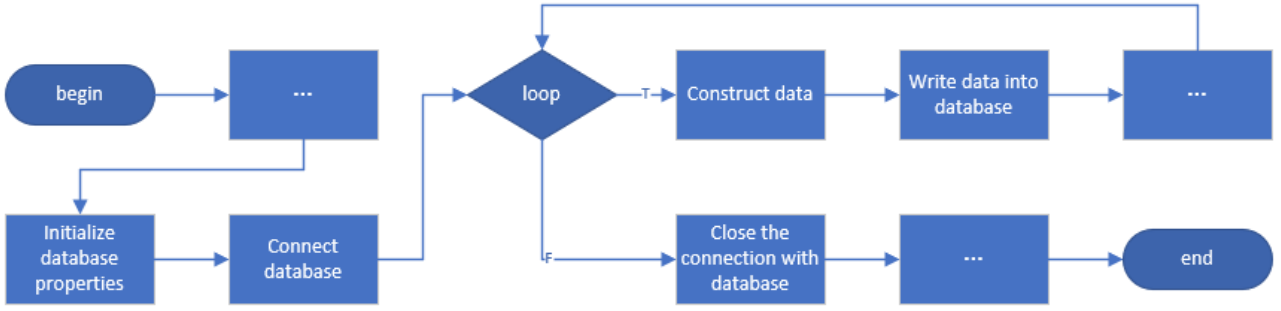


Figure 3 Database operation flowchart

2.1 *1.3. Enhanced User Experience*

By storing translation records in the local database, the system can offer the following enhanced user experience:

Historical Record Query: Users can query previous translation records, reviewing past translation content and times.

Analysis and Statistics: The application can leverage stored data for analysis and statistics, such as identifying which language combinations have higher translation demands or frequently used translation content.

Improved Model and Algorithm Accuracy: The stored data can be utilized to enhance the accuracy of models and algorithms. By analyzing historical translation data, patterns and trends can be identified, leading to insights that could refine the translation process and contribute to the continual improvement of the translation application's accuracy.

Offline Access: Since data is stored in the local database, users can access previous translation records even when not connected to the internet.

Incorporating a local MySQL database system into the translation application demonstrates a strategic approach to enhance functionality and user experience. This part has highlighted the intricacies of the database's structural design, the sequence of database operations, and the resultant benefits for users. Moreover, practical considerations for implementation have been underscored, which augments the application's capabilities and data management prowess.

3. Result

Although the method section already demonstrated the concepts behind the application, a clear walkthrough with application screenshots is still necessary for a better understanding. The overall performance of the application will be shown and discussed here.

3.1 . *User Interface and Experience*

User interface is a critical part of the over application. Starting from the very beginning, users will be able to provide both "from" language and "to" language as part of the settings. Users can also visit the link provided at the bottom of the menu page for more language keywords. If users choose to set the application to default, they can simply leave the both language input text fields blank and continue from there. This will set the "from" language to Simplified Chinese and "to" language to English. After setting the language preferences, users can either choose to start the translation process by clicking the "Start" button or exit the application by clicking the "Quit" button.

Image 9 Menu UI Screenshot

Image 10 The user setting the languages to Japanese and Simplified Chinese

3.2 . *Translation Accuracy*

For the translation process, the team is confident that the accuracy is high enough to support common words and sentences translation regardless of the language. Since the application uses Google Translate API, as long as the API still functions and no major problems happen for Google Translate's translation quality, the translated texts appearing on the screen will always be accurate enough to help the user navigate through foreign language environments.

Image 11 Translation Result Screenshot

As Image 11 has shown, the overall translation of the texts at the back of the phone regarding its company and model are accurate and clear enough for the user to understand.

3.3 . *Database Usage*

In this study, we utilized a database to store and manage our data.

Image 12 shows the establishment of a connection with the database and the display of the results on the console at the beginning of the program execution. This connection was achieved by using the appropriate database connection string and the relevant database driver.

Image 13 illustrates the results displayed on the console for each database insertion operation. These insertions were performed by executing the corresponding SQL statements, and we used the appropriate database insertion syntax to insert the data into the database.

Image 14 describes a single record inserted into the database. This record includes specific data fields and their corresponding values, which were correctly inserted into the respective table of the database.

By utilizing a database, we were able to efficiently store and manage our data, and perform retrieval and manipulation of the data through executing appropriate database operations. This played a crucial role in the smooth execution of our research project.

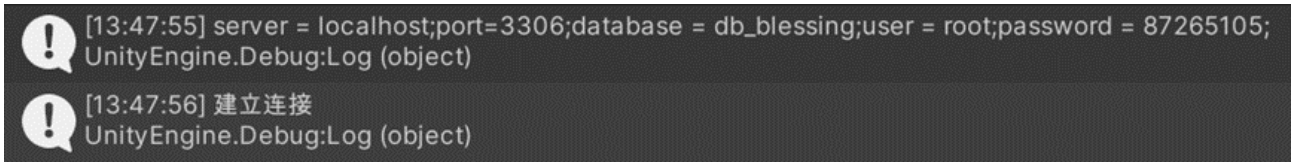


Image 12 Establishing Connection with Database

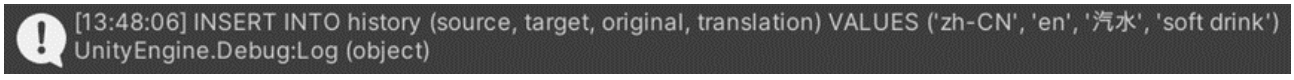
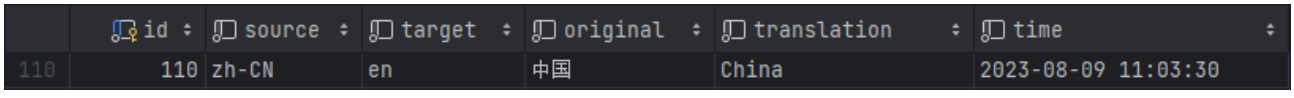


Image 13 Execution of Insert Statement



	id	source	target	original	translation	time
110	110	zh-CN	en	中国	China	2023-08-09 11:03:30

Image 14 A Record in Database

3.4 . *Application Performance*

The discussion of the application performance can be divided into translation time and character recognition performance. For the translation time, it is highly dependent on the application's connection with Google Cloud Vision and Google Translate API, therefore whenever the application is run outside of the service range of the two APIs mentioned, the overall translation speed is reduced significantly. This is a problem that cannot be resolved by the team. On the other hand, the result of character recognition is already demonstrated in Image 3. The translated text "Use 20" is the translation of part of the words 进网试用 and the number 20 to the right of the words. This is caused by the shaking of the camera when trying to recognize the characters in the camera capture range.

4. Discussion

Currently, the application is at a stage where all the basic functionalities are working. However, there are still lots of potential improvements the team can make to the application so that the user will have a better experience. This section will be focusing on the current problems of the application and trying to come up with appropriate solutions.

4.1 . *Translation Performance*

The application currently has the whole operation of receiving results and sending pre-translated results running at a local level. This means that all scripts are written internally within the Unity engine that the application is running on. In the future, the team could move all the script operations into a cloud service, thus reducing the cost of running by a huge amount and improve the overall translation performance of the application. This can be done by running a local translation library on applications such as Docker and connecting the application to Docker with API. The application will then receive the translation result regardless of the stability of the connection with Google server. The user from outside the Google service range will then be able to use the application much more easily.

4.2 . *Translated Texts*

The team had the idea of spawning translated texts in camera as 3D objects so that these translated texts will cover the original texts regardless of the angle that the

user takes when looking at the pre-translated texts with the camera. This can be done by designing a system that can help pinpoint the exact locations of the pre-translated texts in the current space and spawning 3D objects with those locations.

4.3 . *More Languages Supports*

At the moment, the application can only recognize and translate CJK languages. It will be a huge improvement if the application can recognize and translate all languages that are listed inside the Google Translate language list. This can be done by re-organizing the code base so that the application will be able to turn all languages into unicode to recognize them correctly.

4.4 *Running speed*

Due to our use of both the Google Vision and Google Translate APIs, there were instances when the connection to Google's APIs was not optimal, particularly when used within mainland China. This led to significant delays, impeding the attainment of real-time translation, especially during scenarios with rapidly changing scenes such as high-speed movements. In order to address this issue, we are considering performing text recognition or translation processes locally in subsequent improvements. Given the need to handle a variety of outdoor scenes, artistic fonts, and handwritten text, we are still considering using mature pre-trained datasets for text recognition.

4.5 *More application context*

The main focus of this project was on solving line-by-line translation, which is more suitable for outdoor translation of slogans, billboards, etc. However, there are limitations in dealing with large coherent blocks of text, and errors might occur when dealing with multi-line text. For future enhancements, we hope to introduce other methods, such as extracting punctuation positions, to improve the accuracy of translation for multi-line texts.

4.6 *Improving Storage*

The application can be improved by using cloud storage. Migrating a local database system to the cloud offers several advantages. Firstly, it eliminates the need for expensive hardware equipment and dedicated personnel for maintenance, reducing costs. Additionally, cloud service providers have robust data backup and recovery mechanisms, ensuring data security and availability. Cloud databases also enable cross-device and cross-location data access, improving work efficiency and flexibility. Overall, cloud databases offer cost savings, convenience, and enhanced data security compared to local database systems.

5. Acknowledgement

Youding Yin, Guanzheng Liu, and Shiqi Zhang contributed equally to this work and should be considered co-first authors.

References

[1] Google Translate Help. (2023). https://support.google.com/translate/answer/6142483

[2] Alan B. Craig, Understanding Augmented Reality: Concepts and Applications, Morgan Kaufmann Publishers, 2013.