System Design

Mechatronics & Software Engineering

Team 3 - SmartRead: Instant Translation with Wearable Device

Ding (Chris) Hao
Yimin (Jane) Pang
Yucheng Yao
Wenyu (Winnie) Yin
Taoming Yu
Xiang (Shawn) Zhang

Date	**Developers**	**Change**
Jan 9, 2025	Group 3	The first version of the document

Table 1. Revision History

Table of Contents

Introduction

The SmartRead Project is an innovative wearable device designed to simplify language translation by providing quick and efficient text interpretation in a portable and user-friendly format. The system captures text, processes it for translation, and displays the result on a connected interface. SmartRead aims to assist users in overcoming language barriers in an easier way. By integrating this device into their routines, users can enhance their experiences in diverse environments, such as travel, work, entertainment and education.

Purpose

The purpose of this documentation is to provide a comprehensive system design framework for the SmartRead project based on the outlined system design requirements and hazard analysis. It serves as a reference for understanding the structure, behaviour, and interactions of the system's components, ensuring alignment with defined requirements, safety standards, and performance goals throughout all development phases.

Scope

The scope of this document gives the detailed system design of the SmartRead project, focusing on the integration and interaction of its core components: the wearable device and the smartphone application. It provides an overview of system boundaries using a context diagram and elaborates on the subsystems based on their inputs, outputs, and behavioral interactions. This document also includes monitored and controlled variables, constants, and timing constraints as outlined in the System Requirements and Hazard Analysis documents. Additionally, it addresses initialization procedures, normal operational behavior, and strategies for handling undesired events, offering a comprehensive framework to guide the project's design and development.

Assumptions

1. Users follow safety guidelines, including handling the device properly to avoid damage or misuse and operating it within the recommended environmental conditions.
2. The system operates in an environment from 0 ℃ to 45 ℃ [1], and the device does not support operations in extreme environmental conditions.
3. Users will ensure a stable and reliable Wi-Fi connection to support the system's data transmission requirements, such as uploading captured images.

4. Users will only use the system to translate tangible text entries.
5. It is assumed that the system will be operated in a stable and well-lit environment, ensuring optimal conditions for capturing clear images.
6. Given that all clients share the same backend server and the server will queue the tasks, this document assumes that the queuing process would have a negligible impact on overall system performance.
7. At the initialization stage, it is assumed that all hardware components are pre-installed and securely connected together.

Monitored and Controlled Variables

Monitored Variables

Variable Name	**Unit**	**Description**
M_BUTTON	Category: pressed, not pressed	Captured images when the button is pressed
M_LIGHTING	Lumen	Represents the ambient lighting levels, which may affect image quality and text extraction accuracy
M_DISTANCE	Meters	The distance between the camera module and the object being imaged
M_BATTERY_VOLTAGE	Voltage	Represents battery status
M_TEMP	Celsius	Represents the operating temperature of hardware components
M_DETECT_LANGUAGE	Category: English, French, Chinese, etc.	User's selection of which language they want the device to detect
M_TRANSLATE_LANGUAGE	Category: English, French, Chinese, etc.	User's selection on which language they want the device to translate to

Table 2. Monitored Variables

Controlled Variables

Variable Name	Unit	Explanation
C_CAPTURED_IMAGE	Pixels	The images captured by the camera module
C_DETECTED_TEXT	Category: English, French, Chinese, etc.	Legible text extracted from the captured images that meet quality and accuracy thresholds
C_TRANSLATED_TEXT	Category: English, French, Chinese, etc.	The translated text shown on the user interface is controlled to ensure readability and accuracy.
C_DEVICE_STATUS	Category: Idle, Regular operation	When the capture image button is not pressed for a long time, the device will go idle mode to save the battery.
C_LED	Category: off, on, blinking	The LED light is used to represent battery status. LED is off when the battery is out, on when the battery does have electricity, and blinking when the battery is low.
C_WARNING	Category: send, not send	When the battery is low or dead, a warning will be sent to the application display.

Table 3. Controlled Variables

Constants with Nominal Values

The calculated variables for system parameters are based on theoretical data from datasheets and general assumptions. However, actual values may vary due to factors such as component tolerance, environmental conditions, voltage regulator inefficiency and variations in current draw during different operational states.

Constant	Nominal value / Range
CONST_MAX_TOTAL_WEIGHT	225 grams

CONST_LIGHTING_RANGE [2]	300 ~ 750 lux
CONST_MAX_POWER_CONSUMPTION	1.55 W
CONST_WIFI_SPEED_RANGE	802.11 b/g/n
CONST_MAX_MICROCONTROLLER_DIM	27 * 40.5 * 4.5 (± 0.2) mm
CONST_MIN_RESOLUTION [3]	40 × 30
CONST_MAX_RESOLUTION [3]	1600 × 1200
CONST_BATTERY_CAPACITY_RANGE [1]	1000 - 2000 mAh
CONST_BATTERY_DURABILITY_STD	5.8 ~ 10 hours
CONST_BATTERY_DURABILITY_IDLE	5.8 ~ 300 hours
CONST_MAX_OVERALL_TIME	2.8 seconds
CONST_MAX_IMAGE_CAPTURE_TIME	935 ms
CONST_MAX_PROCESSING_TIME	1.4 seconds
CONST_MAX_TRANSMISSION_TIME	32 ~ 436 ms
CONST_MAX_APPLICATION_STORAGE [4]	50 Mb
CONST_MIN_TRANSLATION_ACCURACY	78%

Table 4. Constants with Nominal Values

Behaviour Overview

Upon powering on, the SmartRead wearable device initializes in an idle, low-power state, ensuring energy efficiency while monitoring input variables such as M_BATTERY_VOLTAGE and M_TEMP to confirm a proper operational condition. The microcontroller actively awaits the

M_BUTTON input to transition the system to an active state. Once activated, the device captures images through the camera module, influenced by variables such as M_LIGHTING and M_DISTANCE, to ensure optimal quality. The microcontroller processes and transmits the captured image (C_CAPTURED_IMAGE) to the backend system, which performs text extraction and translation based on user-configured variables like M_DETECT_LANGUAGE and M_TRANSLATE_LANGUAGE, and returns the translated text (C_TRANSLATED_TEXT) for display on the smartphone app. This coordinated interaction between hardware and software subsystems ensures the proper functionality and user experience.

Context Diagram Showing Boundaries

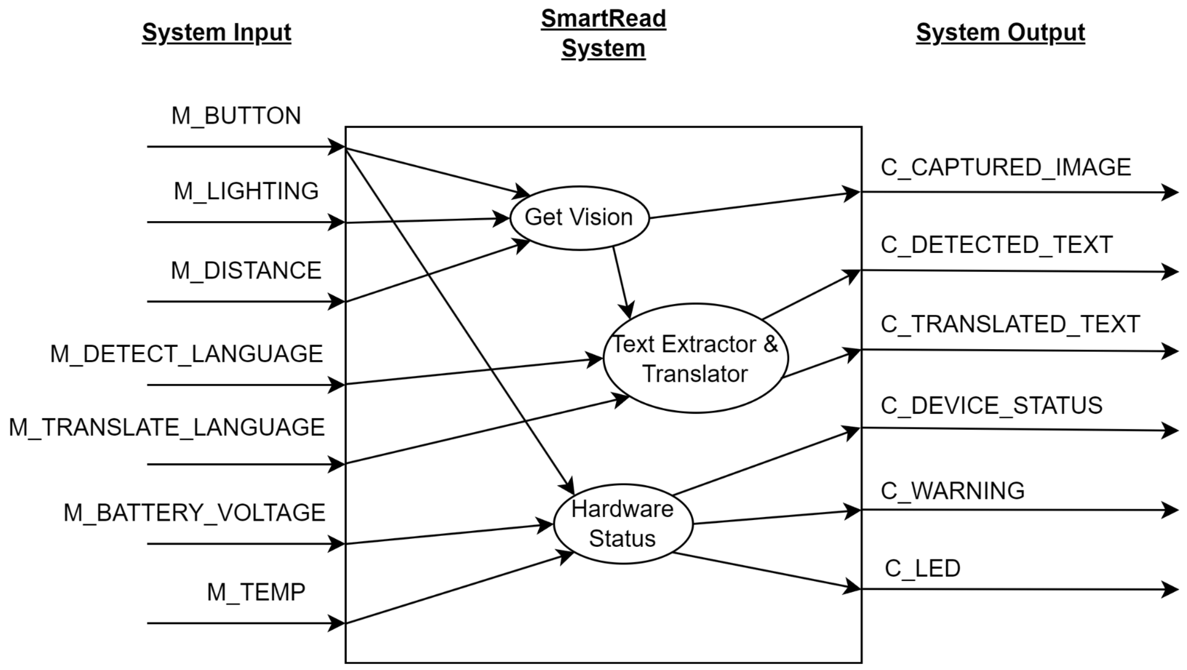


Fig 1. System Boundary Diagram

The context diagram illustrates the boundaries of the SmartRead system, showing the flow of inputs and outputs between the system and its environment. It highlights how the system processes various inputs to perform its core functions, such as capturing, extracting, and translating text, as well as managing hardware status. The diagram provides a high-level view of the system's interactions, outlining how inputs trigger processes within the system to produce corresponding outputs that fulfill the system's objectives.

Subsystem Overview

Based on the context diagram and the features supported by SmartRead, the system has been decomposed into four subsystems to ensure efficient development, maintainability, and scalability. This decomposition follows the principle of separation of concerns, ensuring that changes in one subsystem have minimal impact on others.

At the top level, the SmartRead system is divided into hardware and software components, emphasizing a clear distinction between physical device functionality and computational or online services.

The hardware component is further divided into:

1. Wearable headband subsystem
2. Electronics subsystem

The software component is divided into:

1. Smartphone application system
2. Backend server system

This decomposition enables modularity, allowing each subsystem to be developed and maintained independently while supporting the overall functionality of the system.

The following diagram illustrates how the SmartRead system is decomposed and interconnected.

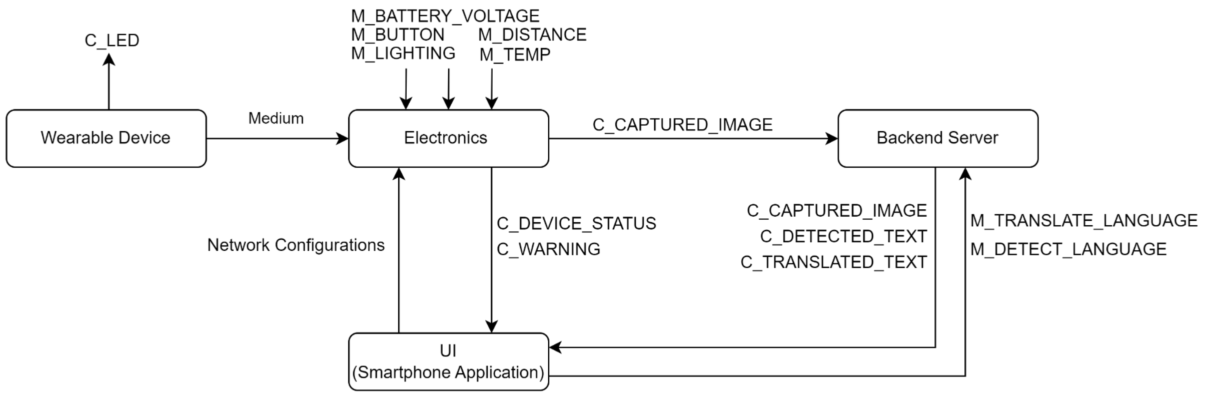


Fig 2. System Decomposition Diagram

The following table elaborates on how the subsystems are connected.

From	**To**	**Interface**[1]
Wearable Device	Electronics	N/A. The wearable device acts as a mechanical holder and is used to hold the electronics securely on the user's head.
Electronics	Backend Server	HTTPS Requests
Electronics	UI	WebSocket
Backend Server	UI	Firebase Push Notification
UI	Backend Server	HTTPS Requests
UI	Electronics	Wi-Fi Provisioning

Table 5. Subsystem Connection

Wearable Headband Subsystem

The wearable headband sub-system for the ESP32-CAM board is designed to seamlessly integrate the Electronics Subsystem into a lightweight and ergonomic headband. The primary objectives are to ensure user comfort during prolonged use and easy accessibility to device controls and functionality.

Design Philosophy:

1. Comfort First: The headband will be constructed using lightweight, breathable, and adjustable materials to ensure comfort during extended wear. (HSR-7)
2. Ease of Access: Buttons, ports, and indicators will be strategically placed for quick and effortless access.
3. Balanced Weight Distribution: The ESP32-CAM module and associated electronics will be positioned to minimize pressure points on the user's head.
4. Environment-Friendly: The 3D-printed part of the Wearable Headband Subsystem will be designed to consume a reasonable amount of material and will be printed with PLA, which is a recyclable material. (ESR-1, ESR-2, ESR-4)

Electronics Subsystem

The electronics subsystem serves as the core hardware management unit of the SmartRead system, bridging the integration of physical components with the software layers. Its primary purpose is to ensure the operation of the hardware, providing the foundation for

[1] The behavior and constraints of each interface is introduced in corresponding subsystems.

capturing images, monitoring the system's internal state, and enabling reliable communication with other subsystems.

This subsystem detects user input through the button press, activating the camera module to capture images of the target text for translation. It continuously monitors critical hardware parameters, such as battery voltage and device temperature, to ensure safe and stable operation. Additionally, it optimizes power usage by transitioning to an idle mode when the capture button remains inactive for a specified period.

Beyond these core functions, the Electronics Subsystem handles data communication by transferring captured images and hardware status updates to the smartphone application for further processing and feedback. It also provides visual feedback to the user via an LED indicator, signalling system states such as a low battery or operational errors. As the backbone of hardware operations, this subsystem ensures smooth functionality and effective interaction with other subsystems.

Smartphone Application Subsystem

The primary purpose of the smartphone application subsystem is to provide a straightforward interface for users to interact with the SmartRead system, aligned with requirement R-7, which offers a clear and intuitive display of the device's status. This subsystem acts as a bridge between the embedded system and the backend server for real-time user interactions. It communicates with users by providing an interface on a smartphone that helps users interact with the SmartRead system. There are three major responsibilities of this subsystem:

1. Guiding users to connect the microcontroller to a network.
2. Prompting users for source and target languages.
3. Displaying translated text in a proper format.

This document will focus on the software implementation of this subsystem (i.e. the UI application). With the presumed platform for hosting the UI being common smartphones, the hardware component of this subsystem has already been well-implemented and will not be discussed in this document.

Backend Server Subsystem

The backend server subsystem is considered a centralized processing unit for requests from all end-users, and it takes care of tasks that require significant computation resources. Reserving a dedicated server for such resource-intensive jobs ensures lightweight software for users to install, minimizing the performance requirement on user's devices.

This subsystem puts an emphasis on two major responsibilities. The first is performing machine learning predictions, such as page segmentation and optical character recognition (OCR), to process images and extract text with high accuracy. The outputs will then be shared with the smartphone application, where users can interpret them. The second is handling smartphone application requests, such as user authentication and session management, which enables secure and seamless interaction.

The backend server subsystem, together with the smartphone application subsystem, achieves most of the software requirements mentioned in the system requirement document. For example, with optimized processing algorithms and a comprehensible way of displaying information on the UI, system requirement R-2 can be achieved. Similarly, the machine learning models and image processing techniques adopted by the server are the key to satisfying general requirements R-5, R-10, and R-11. The backend server also incorporates security measures like data encryption to align with data security requirements DSR-1 and DSR-2. A steady connection between the UI and the electronic system guarantees R-7 by ensuring reliable communication and seamless data exchange. The server-client architecture mitigates the storage issue for the UI application by hosting the machine learning models on the server and thus supports R-8 [9].

Wearable Device Subsystem

Input

No notable input. The subsystem is designed to be worn by the user.

Output

No notable output. This subsystem is used to hold the electronic components.

Description of Behavior

Our primary objective is to provide comfort during extended use while ensuring easy access to the device's controls. We positioned the camera at eye level to capture clearer images. The input button is located on the top, and the charging port is on the right side, making it more convenient to take photos and charge the device. Finally, the elastic strap and foam padding inside the forehead support help provide a secure yet comfortable fit suitable for prolonged wear.

Components

This subsystem consists of three main parts: the forehead support, the protective enclosure for the electronics, and the elastic strap. The enclosure is designed with openings for the camera, the input button, and charging ports.

Below are images of the prototype Wearable Device Subsystem. For simplicity, the elastic strap has not been shown. The gray component represents the protective enclosure for the electronics, while the lighter gray component is the forehead support. In the final design, the elastic strap will attach to both sides of the headband to securely hold the device on the user's head.

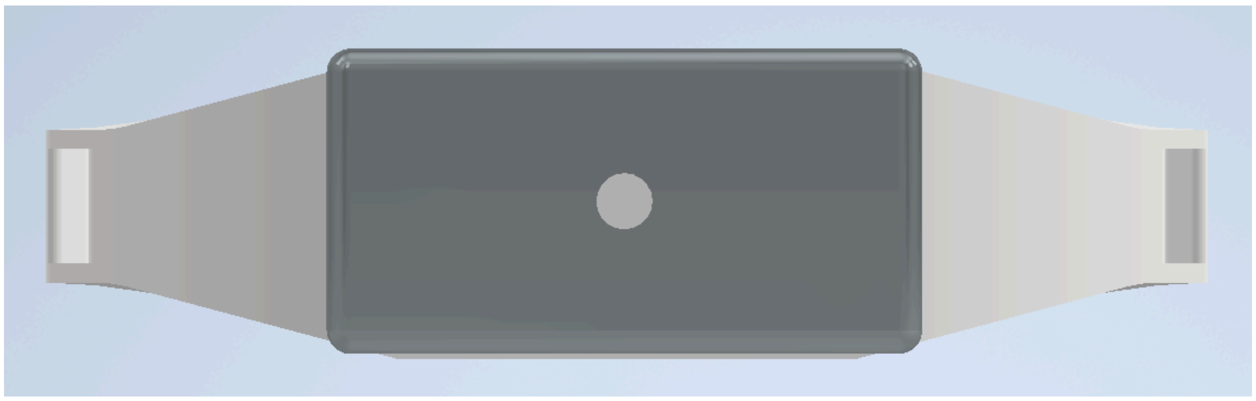


Fig 3. Wearable Device Subsystem Model - Front (with hole for the camera)

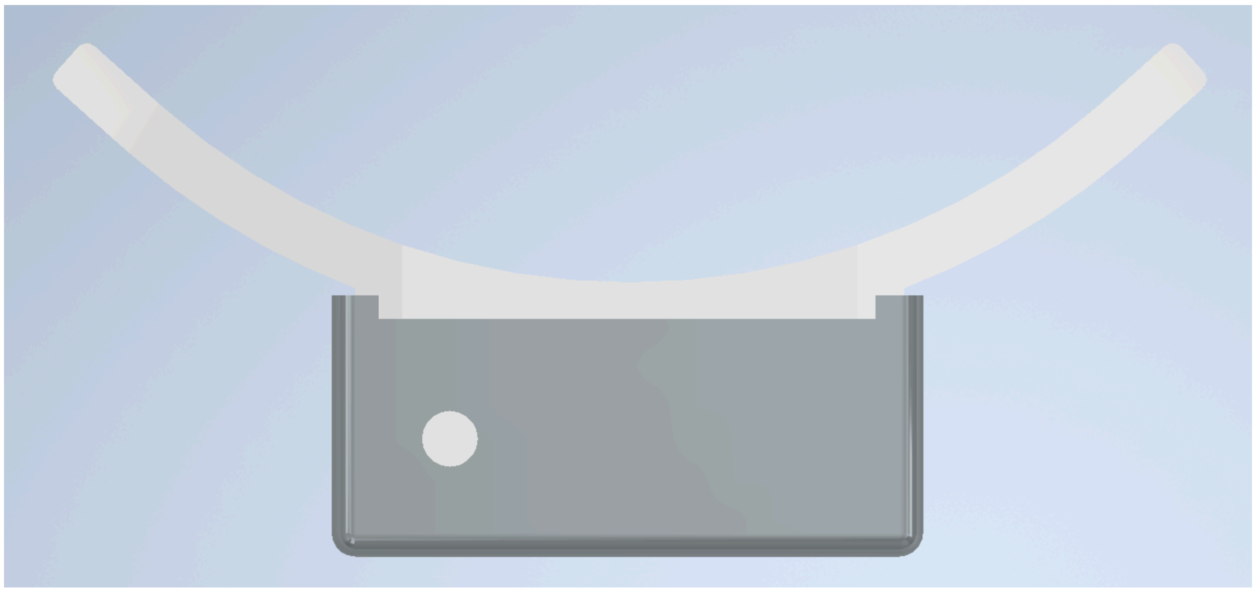


Fig 4. Wearable Device Subsystem Model - Top (with hole for the button)

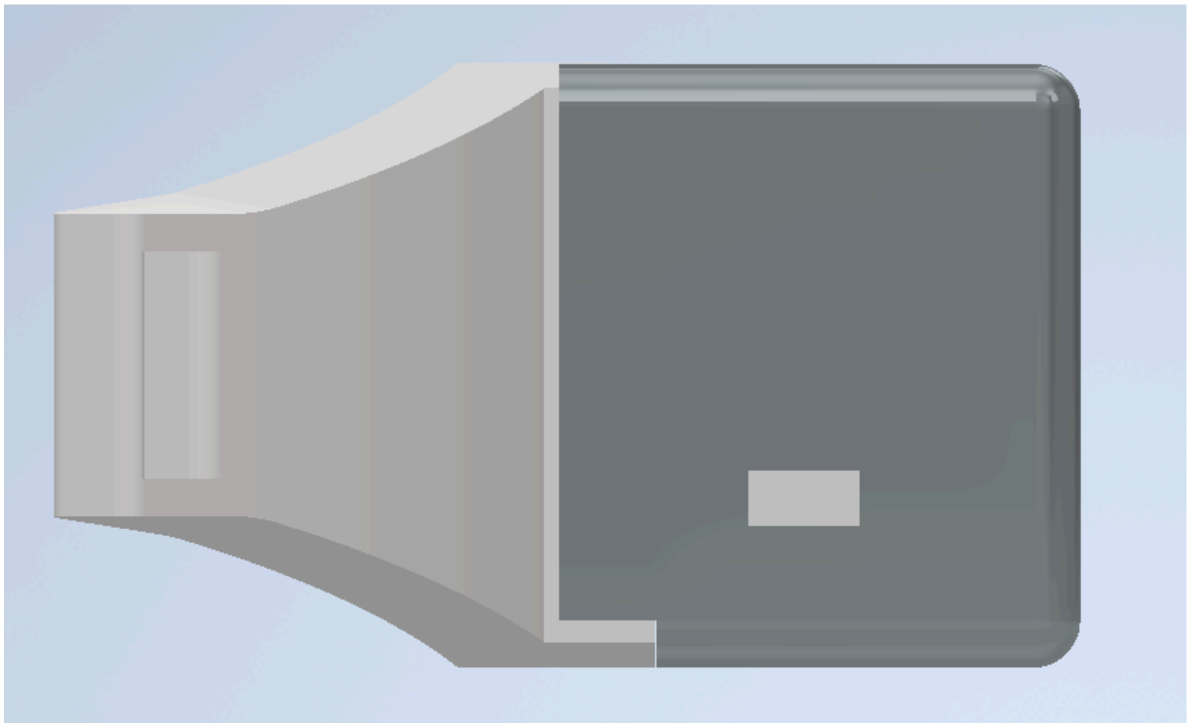


Fig 5. Wearable Device Subsystem Model - Left (with charging slot)

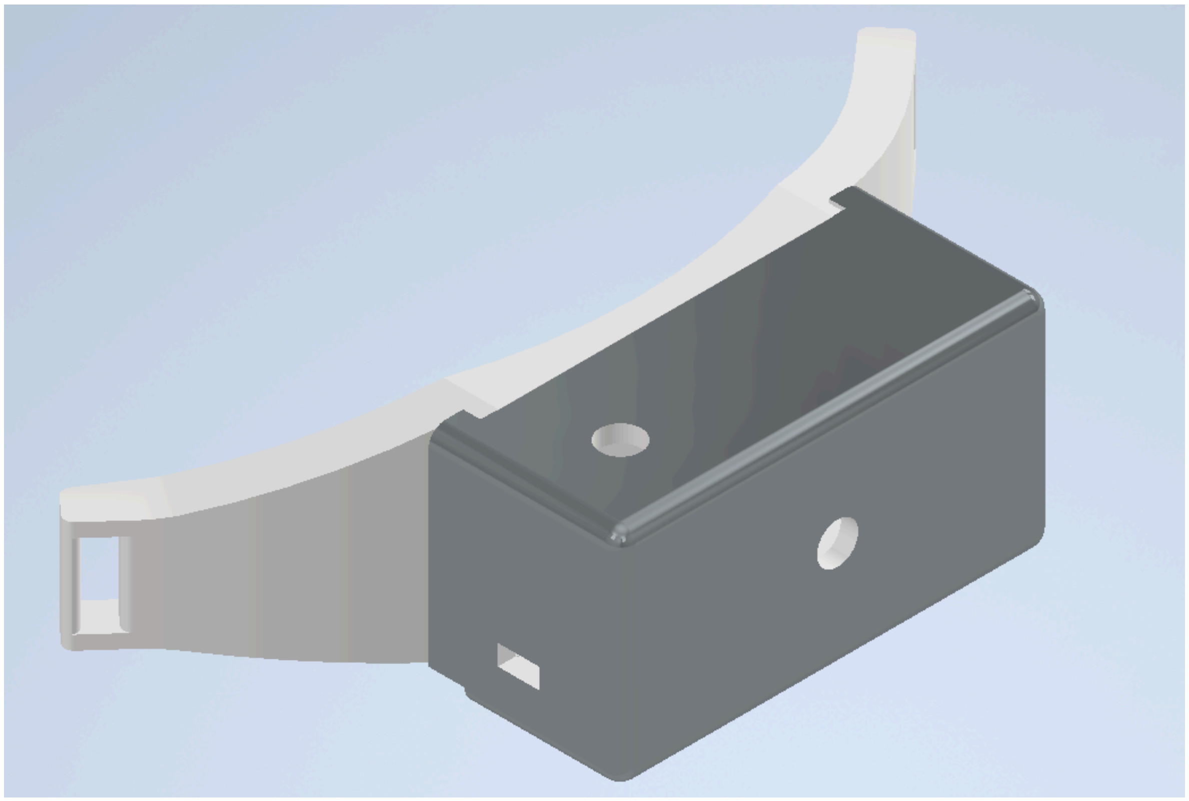


Fig 6. Wearable Device Subsystem Model - Overall

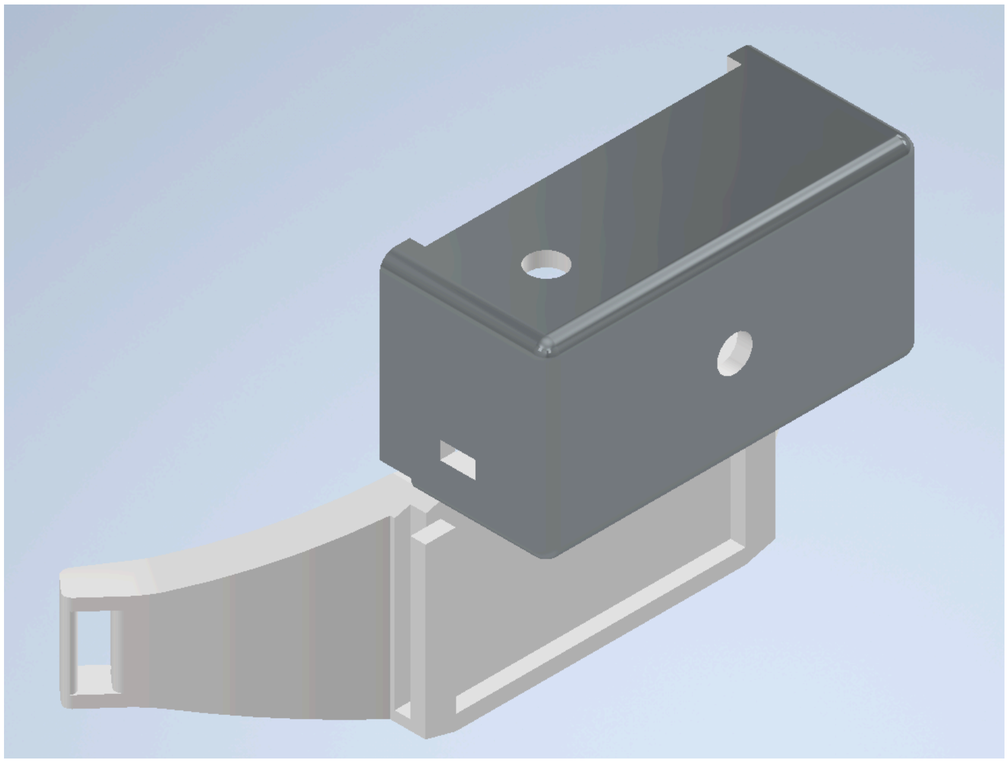


Fig 7. Wearable Device Subsystem Model - Disassemble1

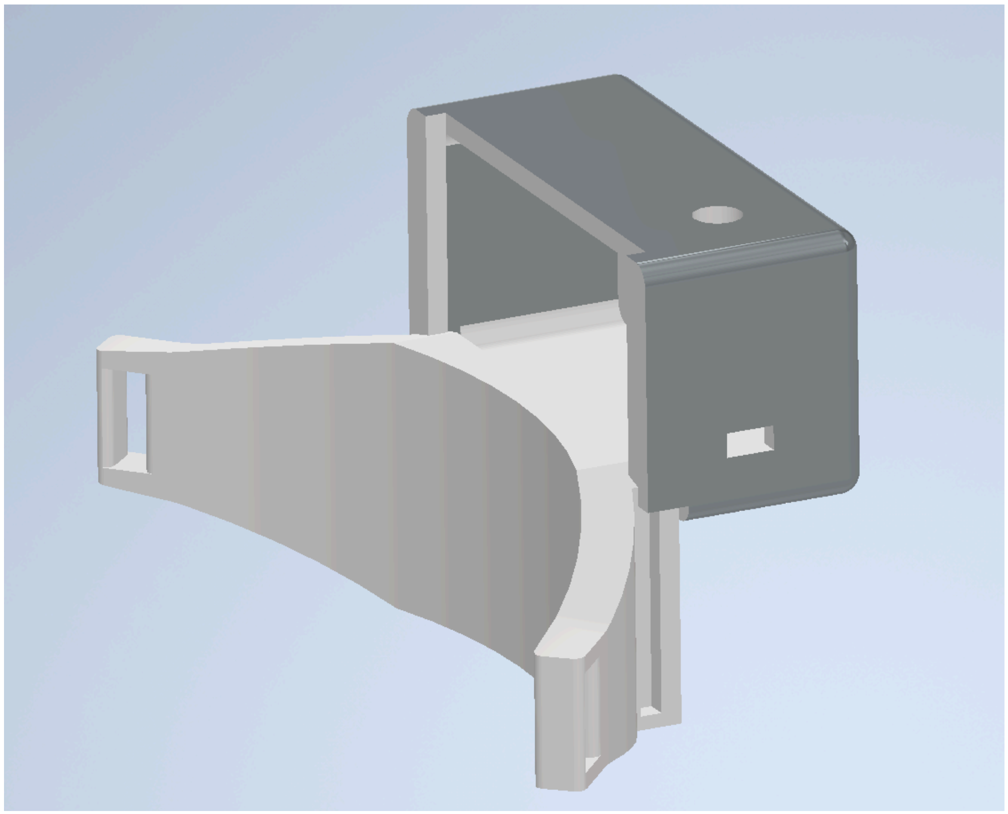


Fig 8. Wearable Device Subsystem Model - Disassemble2

Electronics Subsystem

Input

1. Button press (M_BUTTON)
   - Input from the physical button, which triggers the image capture process
2. Lighting condition (M_LIGHTING)
   - Input from the environment, which represents the lighting level.
3. Distance (M_DISTANCE)
   - Input from the environment, the distance between the camera and the object being imaged.
4. Power supply
   - Battery input to provide energy for all subsystem operations

Output

1. Captured image (C_CAPTURED_IMAGE)
   - The processed image data generated by the camera module and sent to the smartphone application for further use
2. Hardware status (C_DEVICE_STATUS)
   - Indicates the current operational state of the device. If the button is not pressed for a prolonged period, the device transitions to idle mode to conserve battery power.
3. Warning signals (C_WARNING)
   - Alerts are generated when low battery or overheating occurs.
4. LED signal (C_LED)
   - Provides visual feedback to the user, such as indicating a low battery or confirming operational status.

Description of Behavior

1. The system detects user input through the button press, with the M_BUTTON variable monitored by the subsystem. The input device is the physical button, and pressing it activates the camera module, which serves as the output device. This triggers the subsystem to capture an image, which is then processed and sent to the smartphone application. The captured image is represented by the C_CAPTURED_IMAGE controlled variable.
2. To conserve battery power, the subsystem monitors the M_BUTTON variable through the physical button as the input device. If the button is not pressed for a prolonged period, the microcontroller acts as the output device, transitioning the system to an idle state. This idle state is communicated as the C_DEVICE_STATUS controlled variable.
3. The subsystem monitors the internal temperature of the device using a temperature sensor as the input device, with the M_TEMP variable representing the monitored data. The

microcontroller is the output device, generating a warning signal if overheating occurs. This warning is communicated as the C_WARNING controlled variable.

4. The LED indicator provides visual feedback on the system's operational status. The LED indicator is the output device, which is controlled based on the system's battery state. It blinks when the voltage is low, and its status is represented by the C_LED controlled variable. There is no external input device for this process, as it relies on internal system monitoring.

Hardware Component

1. ESP32-CAM
    - Includes microcontroller, camera module, and wireless communication module.
    - Handles image capture, processing, and communication with the smartphone application.
2. Button
    - A physical input connected to the ESP32-CAM to trigger the image capture process
3. Temperature Sensor
    - An internal sensor connected to the ESP32-CAM to monitor the device's temperature and ensure safe operation.
4. LED Indicator
    - Connected to the ESP32-CAM to provide visual feedback on system status (e.g., idle, active, or warnings).
5. Battery
    - Powers the ESP32-CAM and all other connected components.

Software Design

The software design for the electronics subsystem focuses on managing hardware components, processing inputs, and ensuring reliable communication with other subsystems. While this subsystem is not a software-intensive system, as most functionality depends on hardware components and direct user interaction, the software plays a critical role in coordinating these operations. The microcontroller software handles all core hardware operations, including processing button presses, controlling the camera, and monitoring the system's internal temperature. It also manages the logic for transitioning between active and idle modes to conserve battery life while ensuring the coordinated operation of hardware components.

The communication protocol is implemented using HTTP to transmit captured images from the electronics subsystem to the backend server subsystem. Additionally, the ESP32 operates in Access Point (AP) mode to establish a direct connection with the user's phone, allowing the transfer of Wi-Fi configuration details. This dual-mode operation ensures efficient data transmission while enabling seamless user interaction for network setup. The system is designed with lightweight, modular, and Arduino-compatible embedded software to provide an effective and user-friendly control mechanism tailored for this primarily hardware-driven application.

Smartphone Application Subsystem

Input

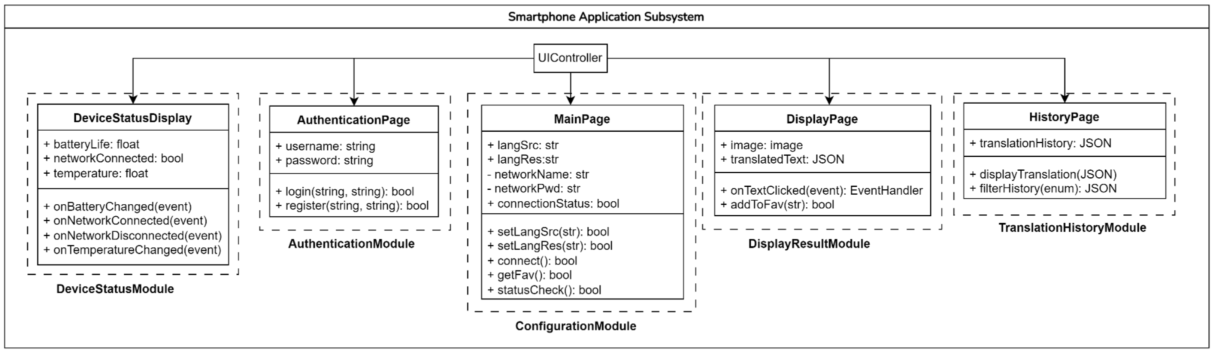
1. Translated text (C_TRANSLATED_TEXT)
    - A list of text extracted from the captured image and translated into the target language
2. Captured image (C_CAPTURED_IMAGE)
    - Original image captured by the user
3. Source language (M_DETECT_LANGUAGE)
    - Text language in the captured image
4. Target language (M_TRANSLATE_LANGUAGE)
    - The language that the user wants to translate into

Output

1. A visual combination of translated text and the original image to be displayed on a smartphone
2. Name and password of the Wi-fi that the smartphone is currently connected to

Modules

The smartphone application subsystem is further divided into modules based on their distinct functionalities. In such a UI-based application, each page corresponds to an individual module. Hence, this subsystem comprises three modules: DeviceStatusModule, AuthenticationModule, ConfigurationModule, and DisplayResultModule. The following image illustrates the structure and organization of these modules.



[2]

Fig 9. Smartphone Application Subsystem Decomposition

[2] UIController is the central module that handles page switches and app management. This is typically handled by the UI development toolkit (e.g. React Native) so it is not considered as a critical module in SmartRead.

DeviceStatusModule

Service

1. Receive the status of the device, including battery life, network connection, and device temperature.
2. Display the device's status on the UI and notify users when one or more critical statuses require attention.

Secret

1. A WebSocket connection from the device to the UI.

Module Interface Specification

Purpose

The device status module is responsible for monitoring the status of the device and alerting users when some statuses are out of the standard range. This module serves as an addition to the LEDs on the device. It helps users to grasp a better understanding of the current condition of the device.

Public Attributes

1. BatteryLife: float
2. NetworkConnected: bool
3. Temperature: float

Events

1. OnBatteryChanged(event) → EventHandler
2. OnNetworkConnected(event) → EventHandler
3. OnNetworkDisconnected(event) → EventHandler
4. OnTemperatureChanged(event) → EventHandler
    - Methods to handle information sent from the device.
    - Will update the UI to represent any changes in the device's status.

Exceptions/Errors

1. Device disconnected
    - Raised when the device cannot be connected to the UI due to expected conditions

AuthenticationModule

Service

3. Allow users to log in with an existing account that is defined by a combination of a username and a password.
4. Allow users to register an account if they have not done so.

Secret

2. A connection to a database that keeps track of user account information

Module Interface Specification

Purpose

The authentication module ensures secure access to the software by verifying user credentials, managing login and logout processes, and safeguarding user accounts against unauthorized access.

Private Attributes

4. Username: string
5. Password: string

Methods

5. Login(str, str) → bool
    - Method to authenticate users
6. Register(str, str) → bool
    - Method to create a new account for a user

Exceptions/Errors

2. Username does not exist
    - Raised if the user provides an invalid username
3. Invalid password
    - Raised if the password the user provides does not match the username

ConfigurationModule

Service

1. Allow users to specify a source language and a target language that will be used in text extraction and translation.
2. Allow users to connect the microcontroller to a network.

Secret

1. Connection credentials to the wearable device

Module Interface Specification

Purpose

The configuration module manages the core application setup, including configuring language preferences, establishing a connection to the backend server, and monitoring the connection status. Setting up the system on the configuration page is a prerequisite for properly operating the system and observing translation results.

Public Attributes

1. SourceLang: string
    - The language on the original image
2. TargetLang: string
    - The language that will be translated into
3. ConnectionStatus: Enum
    - Represent if the microcontroller is successfully connected to a network

Private Attributes

1. NetworkName: string
    - Name of an available network
2. NetworkPassword: string
    - Password for the network that NetworkName specifies

Methods

1. SetSourceLang(string) → bool
2. SetTargetLang(string) → bool
3. Connect(string, string) → bool
    - Connect the microcontroller to a network using a name and a password
4. CheckConnection() → bool
    - Determine if the microcontroller is properly connected to a network

Exceptions/Errors

1. Cannot connect to a network
    - Raised when the microcontroller fails to connect to a network

DisplayResultModule

Service

1. Display the translation result in a comprehensible manner.
2. Support additional features like "Favorite Phrase Collection" and others.

Secret

Not Applicable

Module Interface Specification

Purpose

The display result module receives translation results and other necessary content that will be displayed on the screen from the backend software server. It organizes all the elements and presents them to the user in a clear and user-friendly manner. It also facilitates additional features like saving results, marking favorites, or triggering subsequent operations based on the displayed content.

Public Attributes

1. Image: image
    - The original image captured by the user
2. TranslatedText: JSON
    - All information relevant to what has been translated and how it should rendered on the screen

Methods

1. AddToFav(string) → bool
    - Add a phrase to the favourite phrases collection

Events

1. OnTextClicked
    - Display a popup that contains the original text and other relevant information/features when the user clicks on a text segment

Exceptions/Errors

1. Translation not received
    - Raised when the user captures an image but the translation result is not received by the smartphone application due to unexpected conditions.

TranslationHistoryModule

Service

1. Display all translations that the user has performed in the past
2. Allow to filter the translation history by their timeframes or other criteria

Secret

Not Applicable

Module Interface Specification

Purpose

The translation history module is responsible for offering a comprehensive view of the user's translation history and allows users to view any translation that they performed in the past. This module helps users revisit previous translations without needing to reprocess the same text or image, saving time and effort.

Public Attributes

1. TranslationHistory: JSON
    - A record of the user's past translation results retrieved from the backend software server

Methods

1. DisplayTranslation(JSON) → None
    - A view similar to the DisplayPage should be presented to the user when they select one of the past translations
2. FilterHistory(enum) → JSON

Exceptions/Errors

1. Cannot retrieve translation history
    - Raised when the app fails to retrieve the translation history from the backend server for unexpected conditions

Backend Server Subsystem

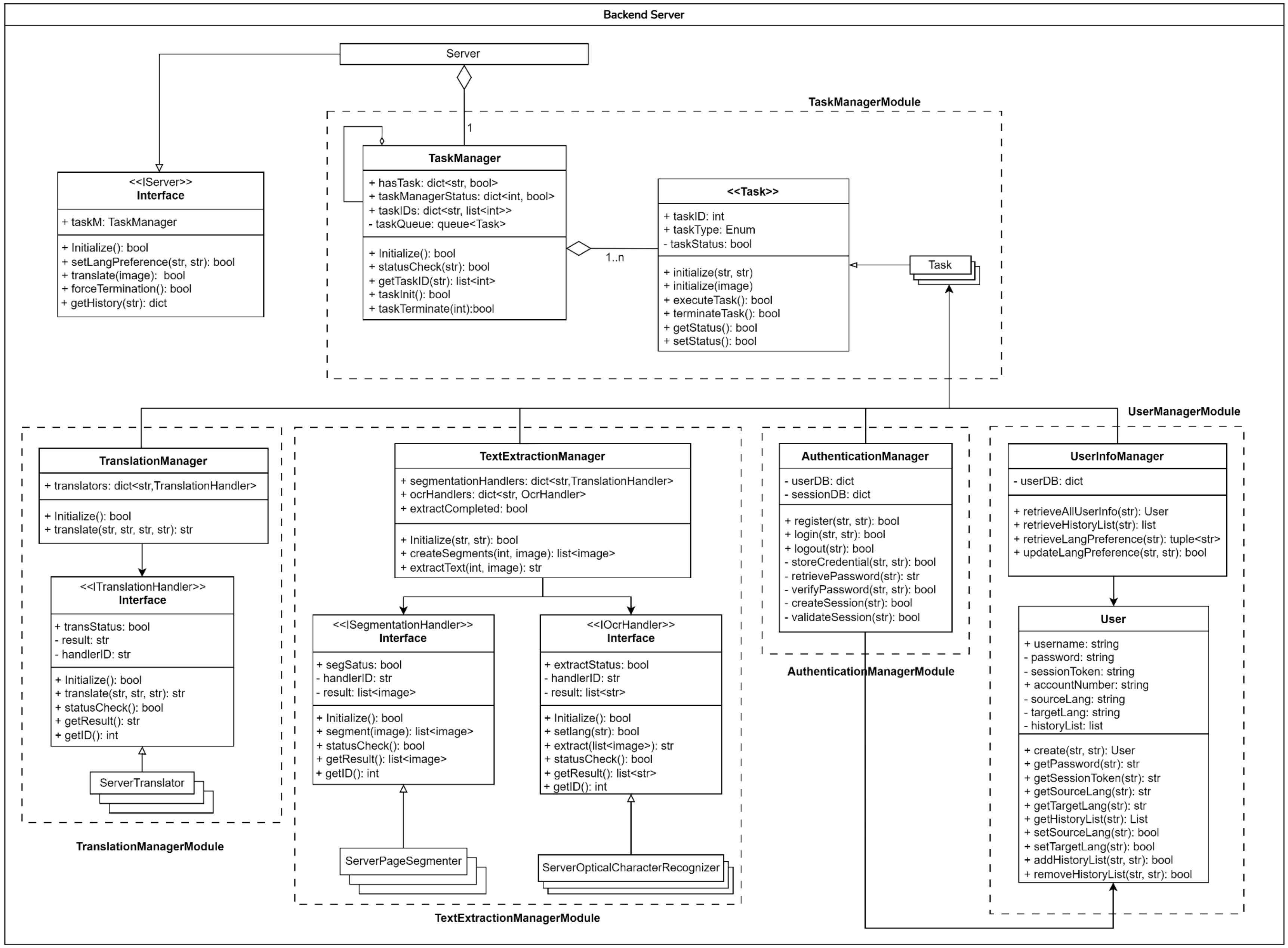


Fig 10. Backend Server Subsystem Decomposition

Input

1. Captured image (C_CAPTURED_IMAGE)
    - Original image captured by the user
2. Source language (M_DETECT_LANGUAGE)
    - Text language in the captured image
3. Target language (M_TRANSLATE_LANGUAGE)
    - The language that the user wants to translate into

Output

1. Translated text (C_TRANSLATED_TEXT)
    a. A list of text extracted from the captured image and translated into the target language
2. Favourite translation: A list of previous translation results those has been added to the favourite list.

Modules

ServerModule

Service

1. Handle incoming requests from either the user interface or the microcontroller
2. Initialize task manager who is responsible for allocating computational resources
3. Terminate any progressing tasks upon requests

Secret

1. Server URL

Module Interface Specification

Purpose

The server interface is an abstraction layer that facilitates interaction between the client and the server's core functionality, which is represented by the TaskManager. Its goal is to provide high-level methods for initializing the server and managing task-related operations such as setting language preferences. By delegating these responsibilities to the TaskManager, the interface ensures modularity, enabling seamless communication and efficient handling of server-side operations.

Public Attributes

1. TaskM: TaskManager
    - A singleton instance of the TaskManager that allocates resources for all tasks

Methods

1. Initialize() → bool
    - Initialize the server, including starting the server and creating the instance of TaskManager
2. SetLangPreference(string, string) → bool
    - High-level method to store the language preference in the database
3. Translate(image) → bool
    - High-level method to extract text from the image and translate it into the target language
4. GetHistory() → bool
    - High-level method to retrieve a user's past translation results
5. ForceTermination() → bool
    - Terminate an ongoing operation when receiving such a request

Exceptions/Errors

1. The user does not exist
    - Raised when a request cannot be completed due to invalid user credentials
2. Task not found
    - Raised when attempting to terminate a task, but the task does not exist

TranslationManagerModule

Service

1. Manage translator instances.
2. Do translation by available instances.

Secret

1. API key used for translation.

Module Interface Specification

Purpose

The TranslationManagerModule is responsible for managing and coordinating translation tasks within the system. It initializes and allocates translator instances to handle untranslated inputs, ensuring efficient processing. The module interacts with translator instances via the TranslationHandler interface to perform translations, check statuses, retrieve results, and manage instance identifiers.

Public Attributes

TranslationHandlers

1. TransStatus: bool

    - Status of the ongoing translation (e.g., whether it is complete or still processing).

Private Attributes

TranslationHandlers

1. Result: string
    - The translated result text after processing.
2. HandlerID: int
    - A unique identifier for each translator instance.

TranslationManager

1. Translators: dict<str, TranslationHandler>
    - Reference to the active translator instance implementing the TranslationHandler interface with its ID.

Methods

TranslationManager

1. Initialize() → bool
    - Initialize the translation manager and create necessary translator instances.
2. doTranslate(string, string, string, string) → string
    - Receive untranslated text, assign a translator instance by ID, and return the translated output.

TranslationHandler Interface

(These methods are implemented by classes such as ServerTranslator that conform to the TranslationHandler interface)

1. translate(string, string, string) → string
    - Translate the input text into the desired language, if the source language can be detected, then input sourceLanguage (string) is not necessary.
2. statusCheck() → bool
    - Check whether the translation process is complete.
3. getResult() → string
    - Retrieve the result of the translation.
4. getID() → integer
    - Retrieves the unique identifier of the translator instance.

Exceptions/Errors

1. NoAvailableTranslatorError
    - Raised when no translator instance is available to handle an incoming task.
2. TranslationFailureError
    - Raised if a translation task fails due to unexpected conditions, such as network errors or internal processing issues.

TextExtractionManagerModule

Service

1. Manage segmentation and OCR handlers to process images for text extraction.
2. Create segmented image regions for targeted processing.
3. Perform text extraction using OCR from segmented regions or the entire image.

Secret

1. Trained machine learning models to perform page segmentation and text extraction

Module Interface Specification

Purpose

The TextExtractionManagerModule is responsible for processing images to extract text. It coordinates between segmentation and OCR handlers to divide the input image into segments (always necessary) and extract textual information. This module manages segmentation tasks and OCR operations, ensuring that the pipeline runs efficiently and that extracted content is accurate and accessible.

Public Attributes

TextExtractionManager

1. SegmentationHandlers: SegmentationHandler
    - Reference to the active segmentation handler implementing the SegmentationHandler interface.
2. OcrHandlers: OcrHandler
    - Reference to the active OCR handler implementing the OcrHandler interface.
3. ExtractCompleted: bool
    - Status indicating whether the text extraction process is completed (specific to the TextExtractionManager).

SegmentationHandler

1. SegStatus: bool
    - Status indicating whether the segmentation process is complete (specific to segmentation handlers).

OcrHandler

1. ExtractStatus: bool
    - Status indicating whether the OCR process is complete (specific to OCR handlers).

Private Attributes

TextExtractionManager

No private attribute for this class.

SegmentationHandler

1. HandlerID: int
    - A unique identifier for each handler instance (for segmentation or OCR).
2. Result: list<image>
    - The result of the segmentation process, which contains the images for extracting.

OcrHandler

1. HandlerID: int
    - A unique identifier for each handler instance (for segmentation or OCR).
2. Result: list<string>
    - The result of the extraction process, which contains the texts.

Methods

TextExtractionManager Methods:

1. Initialize(string, string) → bool
    - Initialize the manager with specific settings and inputs for segmentation and OCR.
2. CreateSegments(int, image) → list<image>
    - Divide the input image into smaller segments for targeted OCR.
3. ExtractText(int, image) → string
    - Extract text from the given image using the OCR handler.

SegmentationHandler Interface Methods

1. Segment(image) → list<image>
    - Segment the input image into regions of interest.
2. StatusCheck() → bool
    - Check whether the segmentation process is complete.
3. GetResult() → list<image>
    - Retrieve the segmented images.
4. GetID() → int
    - Retrieve the unique identifier of the segmentation handler.

OcrHandler Interface Methods

1. SetLang(string) → bool
    - Set the language for OCR processing.
2. StatusCheck() → bool
    - Check whether the OCR process is complete.
3. GetResult() → string
    - Retrieve the text extracted by OCR.
4. GetID() → int
    - Retrieve the unique identifier of the OCR handler.

Exceptions/Errors

1. SegmentationFailureError
    - Raised when the segmentation handler fails to divide the input image into meaningful segments.
2. OcrFailureError
    - Raised when the OCR handler fails to extract text from the image.
3. NoHandlerAvailableError
    - Raised when there are no available handlers (Segmentation or OCR) to process the task.

TaskManagerModule

Service

1. Allow tasks initialization and termination.
2. Manage and coordinate task queue for text extraction, translation, and other related backend processes.

Secret

1. Internal queue management for scheduling tasks.

Module Interface Specification

Purpose

The TaskManagerModule ensures efficient task execution in the backend server. It manages any backend requests and schedules different users' tasks for modules like TextExtractionManager and TranslationManager.

Public Attributes

TaskManager Class

1. HasTask: dict<string, bool>
    - A dictionary indicates if a user has active task processes in the queue
2. TaskIDs: dict<string, list<integer>>
    - A dictionary stores each user's active task processes' ID
3. TaskManagerStatus: dict<integer, bool>
    - A dictionary stores the status of each task queued in the manager

Task Class

1. TaskID: integer
    - A unique identifier indicating a specific task
2. TaskType: enumeration
    - Indicating task type (translation, text extraction, user management)

Private Attributes

TaskManager Class

1. TaskQueue: Queue<Task>
   - All tasks that are queued based on arrival time

Task Class

1. TaskStatus: bool
   - A boolean value indicating the status of the task (success/fail)

Methods

TaskManager

1. Initialize() → bool
   - Initialize the task manager and assign initial values to necessary variables
2. StatusCheck(string) → bool
   - Check the status of all tasks requested by a user
3. GetTaskID(string) → list<integer>
   - Obtain the IDs of tasks requested by a user

Task Class

1. Initialize(string, string) → bool
   - Initialize a task instance using two strings representing language preferences
2. Initialize(image) → bool
   - Initialize a task instance using an image to be processed
3. ExecuteTask() → bool
   - Execute the task and set the status to true
4. TerminateTask() → bool
   - Terminate the task and set the status to false
5. GetStatus() → bool
   - Get the status of the task
6. SetStatus() → bool
   - Set the status of the task

Exceptions/Errors

1. TaskQueueEmptyError
   - Raised when an operation (e.g., retrieving the next task) is attempted on an empty TaskQueue.
2. QueueOverflowError
   - Raised when the TaskQueue exceeds its maximum allowed capacity due to excessive tasks being queued simultaneously.
3. TaskInitializationError

    - Raised when a task fails to initialize due to invalid parameters or missing data during the initialize(str, str) method call.
4. TaskExecutionError
    - Raised when a task encounters an error during execution in the executeTask() method. Say lack of image input or invalid segmentation during processing.

AuthenticationManagerModule

Service

1. Handle user authentication and validation processes
2. Manage user login, registration, and session verification for secure interactions between the application and backend server

Secret

1. User credentials and session tokens stored in the backend database

Module Interface Specification

Purpose

The AuthenticationManagerModule is responsible for securely managing user authentication tasks, including login, logout (session validation), registration, and credentials verification. It guarantees that only authorized users have access to the system.

Private Attributes

1. UserDB: dict
    - A dictionary used for storing user credentials and session tokens
2. SessionDB: dict
    - A dictionary used for storing active session tokens

Methods

1. Login(string, string) → bool
    - Validate the user credentials and create a new session token
2. Register(string, string) → bool
    - Register a new user and store the user credential in the database
3. Logout(string) → bool
    - Terminate the session
4. StoreUserCredentials(string, string) → bool
    - Store user credentials (username, password, session tokens) in the database
5. RetrievePassword(string) → string
    - Retrieve the password associated with the given username from the database
6. VerifyPassword(string, string) → bool

    - Compare and check if the entered password is the same as the password stored in the database
7. CreateSession(string) → bool
    - Create a new active session with a unique token
8. ValidateSession(string) → bool
    - Validate the session by checking the token's existence and timestamp

Exceptions/Errors

1. Invalid user credentials
    - Raised when the provided username or password is incorrect
2. Unauthorized access
    - Raised when attempting to access the system without a valid session/session is expired

UserManagerModule

Service

1. Store information for all end-users
2. Retrieve information for users upon valid requests

Secret

1. Access token for the database that stores all user information

Module Interface Specification

Purpose

The user manager module plays an essential role in handling requests for user information. This module ensures efficient user account management by providing functionalities to create user profiles, set language preferences, and validate user credentials. Its aims at centralizing and streamlining user information handling while supporting secure and scalable user interactions within the system.

Public Attributes

User Class

1. Username: string
    - The username of the user attempting to access the system
2. SessionToken: string
    - A unique identifier indicating an authenticated user's active session
3. SourceLang: string
    - The user preferred language on the original image
4. TargetLang: string
    - The user preferred language that will be translated into

5. HistoryList: list
    - A list of previous translated results

Private Attributes

UserInfoManager Class

1. UserDB: dict
    - A dictionary used for storing user credentials and session tokens

User Class

1. Password: string
    - The password given by the user for accessing the system
2. AccountNumber: string
    - A unique identifier for the user account

Methods

UserInfoManager Class

1. RetrieveAllUserInfo(string) → User
    - When a request arrives, use the account number to obtain all information related to the user, including username, password, language preferences, and translation history
2. RetrieveHistoryList(string) → list
    - Based on an account number, retrieve all the previous translations that the user has performed
3. RetrieveLangPreferences(string) → Tuple<string>
    - Based on an account number, retrieve the user's language preference
4. UpdateLangPreferences(string) → bool
    - Update the language preference of a certain user

User Class

1. Create(string, string) → User
    - Create a user object with username and password
2. GetPassword(string) → string
3. GetSessionToken(string) → string
4. GetSourceLang(string) → string
5. GetTargetLang(string) → string
6. GetHistoryList(string) → list
    - All get methods are for retrieving user private attributes
7. SetSourceLang(string) → bool
8. SetTargetLang(string) → bool
    - All set methods are for setting user language preference
9. AddHistoryList(string, string) → bool
    - Add a new entry of translation result to the existing history

10. RemoveHistoryList(string, string) → bool
    - Remove an existing entry of translation history given an ID

Exceptions/Errors

1. User not found
    - Raised when there is no matched user for the provided account number
2. Fail to update user information
    - Raised when updating a field in user information is unsuccessful for unexpected conditions

Time Constraints

Referring to Performance Requirements introduced in System Requirements & Hazard Analysis, time constraints of the system are introduced in the following table [9].

Requirement No.	**Description**	**Likelihood of Change**
R-2	The system shall display the final translated text within *CONST_MAX_OVERALL_TIME* seconds starting from receiving a button press input (*M_BUTTON*) from the user.	Unlikely to change process time with current technology.
R-4	The system should have a battery that provides a minimum of *CONST_BATTERY_DURABILITY_STD* hours of active use under standard usage conditions or *CONST_BATTERY_DURABILITY_IDLE* hours on standby.	Likely to change the battery life data after actual experimentation.

Table 6. Time Constraints

Initialization

Assuming all the hardware components are working well and are connected securely, the LED will be on to indicate that the wearable device is power-on successfully. To get ready to use the device, first is to download and install the software application SmartRead on a smartphone. Launch the app, create an account, and log in. Assuming the user is using an available WiFi (*WiFi_A*) in their area for internet connection. Pair the app with the microcontroller by hitting the 'Connect' button on the main page of the app to connect the smartphone to the ESP32's WiFi. Upon successful connection, users can connect the microcontroller to the backend server by inputting the username and password of *WiFi_A*. Users should switch back to *WiFi_A* after receiving a response message. The user is required to make sure that the device and their smartphone are using the same internet connection. To prepare for image capturing, users should configure the preferences, including display language and translation settings, on the app's home page.

Test the system by pressing the camera module's button to capture an image. The translated text should be displayed on the smartphone. If issues arise, consult the troubleshooting section in the user manual. Once initialized, SmartRead is ready for regular use, providing efficient text translation with each capture.

Normal Behavior

1. The user selects the desired source and target languages for translation in the smartphone application.
2. The user presses the physical button on the wearable device, triggering the microcontroller to capture an image.
3. The microcontroller transmits the image to the backend server via the Wi-Fi module.
4. The backend server processes the image, extracting the desired text and translating it into the selected target language.
5. The server sends the original image, extracted text, and translated text to the smartphone application.
6. The smartphone application displays the received information on its screen.
7. The user reviews the displayed information to understand the meaning of the foreign text.

Undesired Event & Handling

Event No.	**Description**	**Resolution**	**Rationale**
E-1	The software application (app) is suddenly shut down during processing.	The user needs to reopen the app and re-press the shutter to redo the translation.	If the smartphone application connection is lost, the current translation process will be interrupted.
E-2	The user attempts to capture images repetitively in a short time.	If the user wants to translate multiple items, they need to do it separately with at least CONST_MAX_OVERALL_TIME between each button press.	The system has an input limitation to prevent high-frequency pressing. The system will only capture images at the first press, ignoring the later ones in a certain period.
E-3	The camera module runs out of battery during device usage.	The user needs to charge the battery to allow the device to work again. If an image is already sent before the hardware dies, the software can process it.	There is no power for hardware, so no image will be captured when the button is pressed, but this would not affect software.

E-4	The connection between the wearable device and the user’s interface device is cut off.	The user needs to re-establish the connection by checking the WiFi condition or asking the customer service team.	The interruption of hardware and software connection will cause the user to lose access to ongoing and future translations.
E-5	The server is offline during or before processing.	Although images can be captured, translation functionality is unavailable. A warning message will be shown on the smartphone application. The user needs to wait until the server is back.	The server is the key to extracting and translating text and working between hardware and software UI.
E-6	The lens becomes dirty or obstructed.	The user will need to clean the lens of the camera to keep it functional.	There is no sensor to detect lens cleanness. The user will notice the lens need to be cleaned by looking at the returned degraded result.

Table 7. Undesired Event & Handling [9]

References

[1] LiPo_Battery_LP652631_3.7V_480mAh.pdf - LiPol Battery, https://www.lipolbattery.com/LiPo-Battery-Datahseet/LiPo_Battery_LP652631_3.7V_480mAh.pdf (accessed Nov. 22, 2024).

[2] “Lighting tips: Understanding the recommended lux levels in workspaces,” Lux Level Lighting Tips for Workspaces | Wipro Lighting, https://www.wiprolighting.com/blog/lighting-tips-understanding-the-recommended-lux-levels-in-workspaces (accessed Nov. 22, 2024).

[3] Omni ision®, https://www.uctronics.com/download/OV2640_DS.pdf (accessed Nov. 23, 2024).

[4] By: Brendon Boshell February 21, “Average app file size: Data for Android and IOS Mobile apps,” Sweet Pricing Blog, https://sweetpricing.com/blog/2017/02/average-app-file-size/ (accessed Nov. 23, 2024).

[5] COMPONENTS101, https://components101.com/sites/default/files/component_datasheet/ESP32-CAM_datasheet.pdf (accessed Nov. 21, 2024).

[6] “What affects data extraction accuracy,” Veryfi, Inc. Help Center, https://faq.veryfi.com/en/articles/8048089-what-affects-data-extraction-accuracy. (accessed: Nov. 24, 2024).

[7] “How Accurate is Google Translate?,” Weglot Blog, https://www.weglot.com/blog/how-accurate-is-google-translate. (accessed: Nov. 24, 2024).

[8] “How to Implement a Software-Based Debounce Algorithm for Button Inputs on a Microcontroller,” DigiKey, https://www.digikey.cn/zh/maker/tutorials/2024/how-to-implement-a-software-based-debounce-algorithm-for-button-inputs-on-a-microcontroller. (accessed: Nov. 21, 2024).

[9] “System Requirement And Hazard Analysis,” https://docs.google.com/document/d/1JTKtlGqitk2mzPC-x7vKvZ6OcfRZPXXa/edit#heading=h.ca8cpe3ce1xl (accessed: Jan. 05 2025).

Appendix A - Constant Calculation

For maximum total weight, below is the calculation

Assume there are only four components for the hardware system.

- Microcontroller module and camera module [5]
    - Around 15 grams
- Battery module
    - Around 40 - 60 grams [1]
- System frame
    - 30 ~ 150 grams, depending on the material.

Maximum weight = $15 + 60 + 150 = 225$ grams.

For transmission time, below is the calculation [5]

Assume a 2MP image is compressed to approximately 300 KB.

- Data transfer rate [5]
    - Wifi 802.11 b: Up to 11 Mbps (Theoretical max)
    - Wifi 802.11 g: Up to 54 Mbps (Theoretical max)
    - Wifi 802.11 n: Up to 150 Mbps (Theoretical max)
- Convert image size to bits
    - $300\,KB = 300 \times 8 = 2400\,Kb$
- Calculate effective data rate (real-world application)
    - For Wi-Fi 802.11 g (assuming 50% efficiency)
    - Effective rate $54\,Mbps \times 0.5 = 27\,Mbps$
- Transmission time (T) given by
    - $T = \frac{Data\ size}{Effective\ rate}$
    - For 802.11 b (5.5 Mbps effective)
        - $T = \frac{2400}{5.5} \approx 0.436\ seconds\ (436\ ms)$
    - For 802.11 n (75 Mbps effective)
        - $T = \frac{2400}{75} \approx 0.032\ seconds\ (32\ ms)$ System Boundary Diagram

---

For maximum power consumption and battery durability, below is the calculation [1] [5]

Assume 90% efficiency after using a voltage regulator.

$Power = Voltage \times Current$

$Runtime \ = \ \frac{Battery\ Capacity}{Current\ Draw} \times \ Efficiency$

- Turn off the flash lamp mode
    - $Power \ = \ 180mA \ \times \ 5V \ = 0.9\,W$
    - $Runtime \ = \ \frac{2000}{180} \times \ 0.9 \ = \ 10\ hours$
- Turn on the flash lamp (max brightness) mode
    - $Power \ = 310mA \ \times \ 5V \ = \ 1.55\,W$
    - $Runtime \ = \ \frac{2000}{310} \times \ 0.9 \ = \ 5.8\ \ hours$
- Deep sleep
    - $Power \ = 6mA \ \times \ 5V \ = \ 0.03W$
    - $Runtime \ = \ \frac{2000}{6} \times \ 0.9 \ = \ 300\ \ hours$
- Modern sleep
    - $Power \ = 20mA \ \times \ 5V \ = \ 0.1W$
    - $Runtime \ = \ \frac{2000}{20} \times \ 0.9 \ = \ 90\ \ hours$
- Light sleep
    - $Power \ = 6.7mA \ \times \ 5V \ = \ 0.0335W$
    - $Runtime \ = \ \frac{2000}{6.7} \times \ 0.9 \ = \ 268.7\ \ hours$

For minimum translation accuracy, below is the calculation

General Accuracy = Text Extraction Accuracy × Translation Accuracy [6] [7]

= 95% × 82.5%

= 78%

This means that 78% of the text the user intends to translate will be translated accurately.

For minimum image processing time, below is the calculation

Assume there are two steps for image processing.

- Text Extraction
    - Around 1.2 seconds
- Text Translation
    - Around 0.2 seconds

Minimum Image Processing Time = $1.2 + 0.2 = 1.4$ seconds

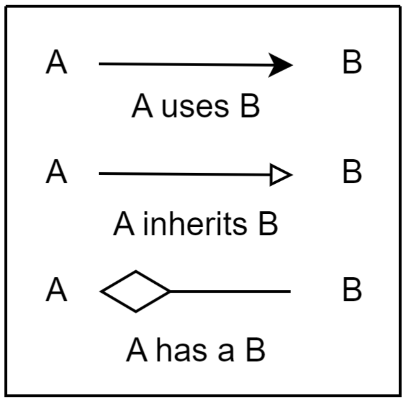
For maximum image capture time, below is the calculation

- Image capture time includes button debouncing, camera initialization and image capture time.
- The standard debounce times for hardware buttons range from 10 to 50 ms [8], depending on the switch type and the desired stability.
- Camera initialization often takes 100 ~ 300 ms depending on resolution and setup.
- Image capture time often takes 200 to 500 ms depending on the image resolution and compression method.
- We can also multiple the time by 1.1 for real-time delay and additional latency that might happen.
- $Max\ time = (50 + 300 + 500) * 1.1 = 935\ ms$

For maximum overall time, below is the calculation

- Assume the maximum overall time is the combination between button pressed, image taken, image processing time, as well as transmission time.
- $Max\ time = 935 + 1400 + 436 \approx 2.77$ seconds

Appendix B - UML Diagram Legend


A
B
A uses B
A
B
A inherits B
A
B
A has a B