République Tunisienne
Ministère de l'Enseignement Supérieur
et de la Recherche Scientifique

Université de Sfax

Ecole Nationale d'Electronique et des
Télécommunications de Sfax

Génie
Informatique Industrielle

Projet de fin d'année

PROJET DE FIN D'ANNEE

Présenté à

L'Ecole Nationale d'Electronique et des Télécommunications de Sfax

Génie Informatique Industrielle

Par

Nawres Barhoumi

Réalisation d'un module IHM (Interface Homme-Machine) pour la commande d'un système de transmission vidéo embarqué d'un train

Soutenu le 24 mai 2024, devant la commission

M.	**Moncef TRIKI et Mohamed Wassim JMEL**	***Encadrant***
M.	**Mohamed ben SLIMA**	***Examinateur***

Année Universitaire : 2023-2024

Abstract :

This project aims to develop a Human-Machine Interface (HMI) integrating the STM32NUCLEO board, the OV7670 camera, as well as ultrasonic and DHT11 sensors. The HMI will enable user-friendly interaction between the user and the embedded system.

The integration of these hardware components into a coherent and functional system will require effective programming of the STM32NUCLEO microcontroller, as well as proper management of data from the camera and sensors. The HMI, developed with a user-friendly and intuitive approach, will allow the user to visualize the information.

This project is part of an initiative to enhance connectivity, monitoring, and control across various fields of application, particularly in transportation.

Keywords :

IHM, OV7670 Camera, STM32NUCLEO, Ultrasonic sensor, DHT11 sensor.

Abstrait :

Ce projet vise à développer une Interface Homme-Machine (IHM) intégrant la carte STM32NUCLEO, la caméra OV7670, ainsi que les capteurs ultrason et DHT11. L'IHM permettra une interaction conviviale entre l'utilisateur et le système embarqué.

L'intégration de ces composants matériels dans un système cohérent et fonctionnel nécessitera une programmation efficace du microcontrôleur STM32NUCLEO, ainsi qu'une gestion appropriée des données provenant de la caméra et des capteurs, L'IHM, développée avec une approche conviviale et intuitive, permettra à l'utilisateur de visualiser les informations.

Ce projet s'inscrit dans une démarche d'amélioration de la connectivité, de la surveillance et du contrôle dans divers domaines d'application, notamment le transport.

Mot-clé :

IHM, Caméra OV7670, STM32NUCLEO, Capteur ultrason, capteur DHT11.

Remerciement

Au terme de ce projet, je tiens à exprimer mes sincères remerciements à Monsieur Moncef Triki, ainsi qu'à Monsieur Wassim Jmal, pour leur soutien indéfectible et leurs conseils avisés tout au long de ce travail.

Leur généreuse contribution en termes de matériel a joué un rôle déterminant dans la réalisation de ce projet.

Enfin, ma plus profonde gratitude va aux estimés membres du jury qui ont aimablement consacré leur temps à l'examen et à l'évaluation de mon travail. Leur évaluation a été cruciale pour affiner et améliorer mon projet.

Table des matières

Listes des figures :

Introduction Générale :

Dans le domaine du transport ferroviaire, la sécurité, l'efficacité opérationnelle et la satisfaction des passagers sont des préoccupations primordiales. L'intégration de systèmes avancés de communication et de commande joue un rôle crucial dans la réalisation de ces objectifs.

Dans ce contexte, notre projet s'est concentré sur le développement d'un module d'Interface Homme-Machine (IHM) destiné à la commande d'un système de transmission vidéo embarqué à bord des trains.

Ce rapport présente une vue d'ensemble détaillée du processus de conception, de développement et de mise en œuvre de ce module IHM.

Je commence par définir le contexte et les objectifs du projet, puis je vais discuter les technologies et les méthodologies utilisées pour atteindre ces objectifs.

Ensuite, je fournis une analyse des fonctionnalités clés de l'IHM et des étapes que j'ai suivies au cours du processus de développement. Enfin, je conclus par un aperçu des résultats obtenus

Ce projet représente une étape importante dans l'amélioration de la connectivité et de la gestion des données à bord des trains, contribuant ainsi à renforcer la sécurité, l'efficacité opérationnelle et l'expérience globale des passagers.

Chapitre 1 : Mise en place de l'environnement de travail

I. Spécification :

1. Le titre de projet :

Réalisation d'un module IHM (Interface Homme-Machine) pour la commande d'un système de transmission vidéo embarqué d'un train.

2. Description et Architecture du projet :

Ce projet présente une solution professionnelle visant à mettre en œuvre une interface graphique homme-machine (IHM) pour permettre à l'agent de conduite d'un train de contrôler efficacement la montée et la descente des voyageurs. Cette IHM est conçue pour intégrer une caméra afin de surveiller les mouvements des passagers. Un capteur ultrasonique est utilisé pour détecter la présence d'objets, tandis que la caméra capture des images correspondantes.

La liaison entre la caméra et la carte STM32 est établie via le protocole I2C. Les données ainsi collectées sont transmises au PC via le protocole UART. Sur le PC, les données sont lues à partir du port série et converties en images à l'aide de l'environnement de développement intégré QT Creator.

Dans le cadre de ce projet, le capteur DHT11 est également intégré. Ce dernier est déployé pour surveiller la température à bord du train, permettant ainsi de prendre en compte les variations saisonnières significatives entre l'été et l'hiver. Une considération majeure est d'évaluer si le nombre de passagers excédera la capacité de places disponibles en fonction de la température ambiante détectée.

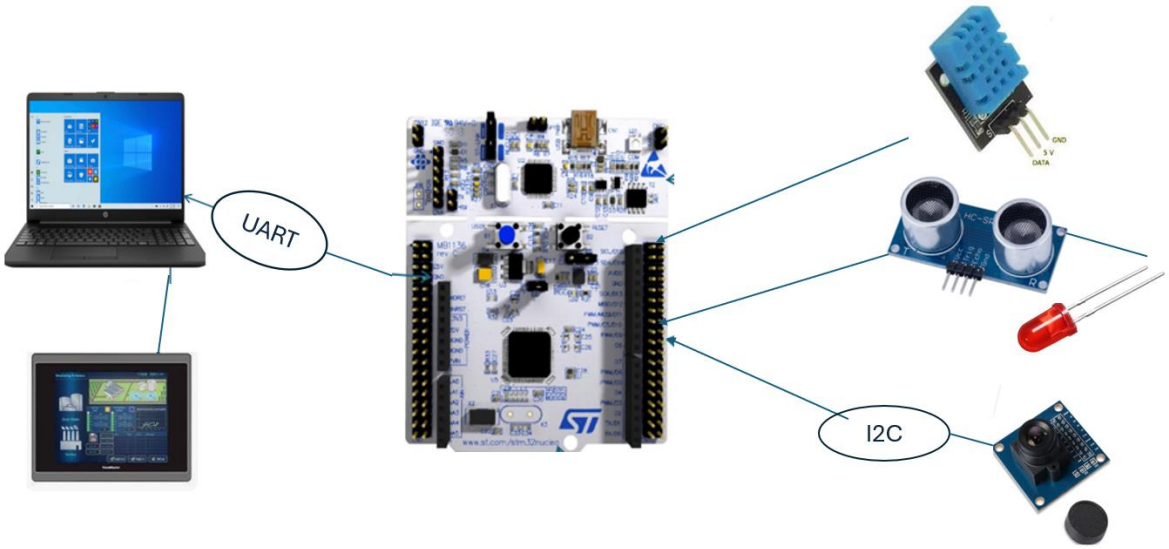


Figure 1:Architecture du projet

3. Méthodologie et objectifs :

- Les étapes à effectuer sont les suivantes :

-installer STM32cubeIDE.

-Envoyer un message par le protocole UART (pour vérifier la connexion entre la carte et le pc).

-Connecter le capteur ultrason et le capteur DHT11 à la carte STM32f030NUCLEO.

-Envoyer la distance via le protocole UART sur l'émulateur PuTTY.

-Envoyer la température (capteur DHT11) via le protocole UART sur l'émulateur PuTTY.

-Connecter la caméra OV7670 à la carte STM32f030NUCLEO en utilisant le protocole I2C.

-Installer QT Creator version 5.12.2.

-Utiliser le module (Serial Port) et trouver les identifiants du port série.

-Lire les données de l'image via le port série en utilisant QT Creator, puis de les convertir en une représentation visuelle pour affichage sur une interface homme-machine (IHM).

II. Environnement de travail :

1. Introduction :

Ce chapitre sera consacré dans un premier temps à présenter l'étendue de l'environnement de travail utilisé.

Qui sera composé d'une carte STM32 NUCLEO comme unité de traitement principale.

Le système sera programmé en utilisant les langages de programmation C et C++.

2. Problématique :

Dans l'univers complexe des transports ferroviaires, plusieurs défis peuvent compromettre la sécurité, la fiabilité et l'efficacité des opérations à bord des trains. Des incidents de sécurité, tels que les comportements suspects des passagers ou les obstacles sur les voies, peuvent potentiellement mettre en danger la vie des passagers et perturber le bon déroulement des trajets.

Ou les situations d'urgence médicale, peuvent se produire à bord d'un train.

3. Objectif :

Réalisation d'une interface graphique IHM qui permet à l'agent de conduite d'un train de contrôler la montée /descente des voyageurs par une caméra.

4. Hardware :

a. STM32f030nucleo :

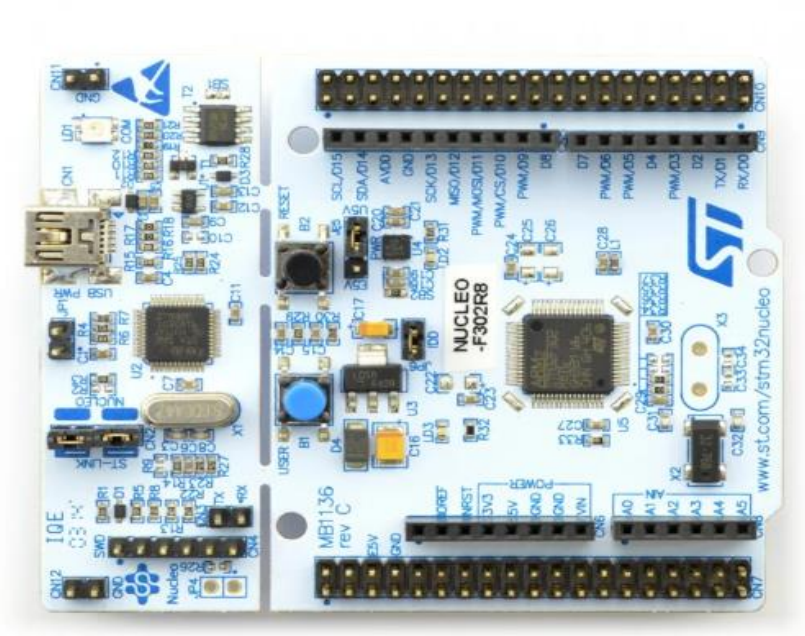


Figure 2:Carte STM32NUCLEO

Avantages :

-Le microcontrôleur STM32F030 est basé sur l'architecture ARM Cortex-M0, offrant une puissance de traitement suffisante pour de nombreuses applications embarquées.

- La carte STM32 Nucleo-64 ne nécessite aucune sonde séparée car elle intègre le débogueur/programmeur ST-LINK.

-De plus uart2 est intégré avec le ST-LINK donc on n'a pas besoin d'utiliser un convertisseur UART-USB.

Désavantages :

-Il est moins puissant que certains autres microcontrôleurs de la famille STM32.

-Il n'intégré pas l'interface DCMI (Digital camera interface), conçue pour faciliter la connexion et la communication avec des capteurs d'image ou des caméras numérique.

b. Caméra OV7670 :

Le CAMERACHIPTM OV7670 est un capteur d'image CMOS basse tension qui Fournit toutes les fonctionnalités d'une caméra VGA monopuce et d'un processeur d'image dans un petit paquet d'empreinte.

Figure 3:Caméra ov7670

Description des pins :

3.3V: 3.3V Approvisionnement

GND : Ground

SDIOC : Horloge SSCB (I2C)

Données SDIOD : SSCB (I2C)

VSYNC : Synchronisation verticale

HREF : synchronisation horizontale

PCLK : Horloge Pixel

XCLK : horloge système

D0 - D7 : sortie de données Pixel

RESET : réinitialiser

PWDN : mode veille

c. Le capteur ultrason :

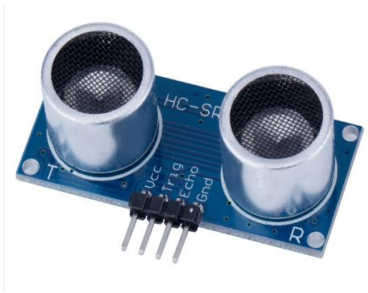


Figure 4:capteur ultrason

Les capteurs à ultrasons sont particulièrement polyvalents. Ils permettent aussi bien la détection de position, la mesure de distance que la détection de supports solides ou liquide, le tout sans contact.

d. Le capteur DHT11 :

Le capteur DHT11 mesure la température et l'humidité, Le capteur DHT11 a 4 broches, mais il est souvent vendu sur une carte support qui possède 3 broches.

Figure 5:capteur DHT11

Avantages :

-Le DHT11 est généralement peu coûteux et facile à intégrer dans des projets électroniques.

Désavantages :

-Le DHT11 n'est pas aussi précis que d'autres capteurs de température et d'humidité.

- La plage de mesure du DHT11 est limitée en comparaison avec d'autres capteurs plus avancés.

5. Software :

a. STM32cubeIDE :

Figure 6:STM32CubeIDE

STM32CubeIDE est un environnement de développement intégré (IDE) avancé destiner à la programmation et au développement de logiciels embarqués pour les microcontrôleurs STM32 de STMicroelectronics.

b. PuTTY :

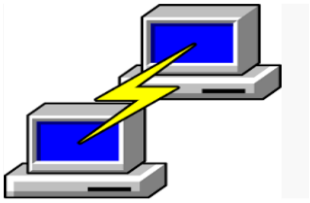
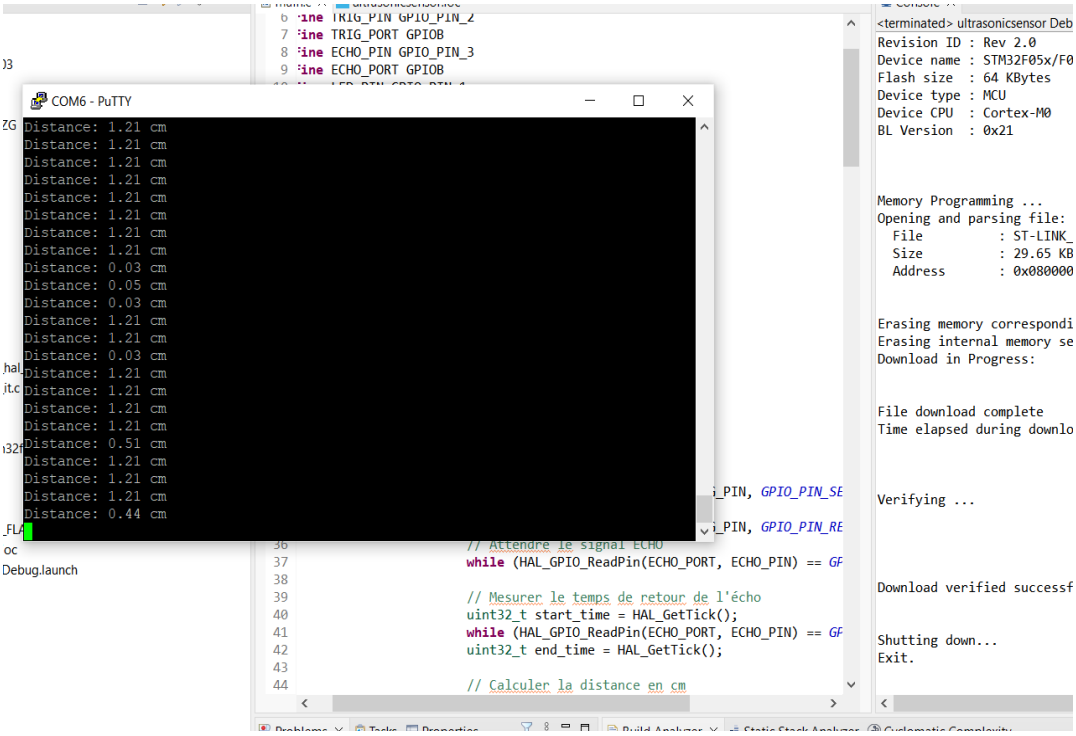


Figure 7:PuTTY logo

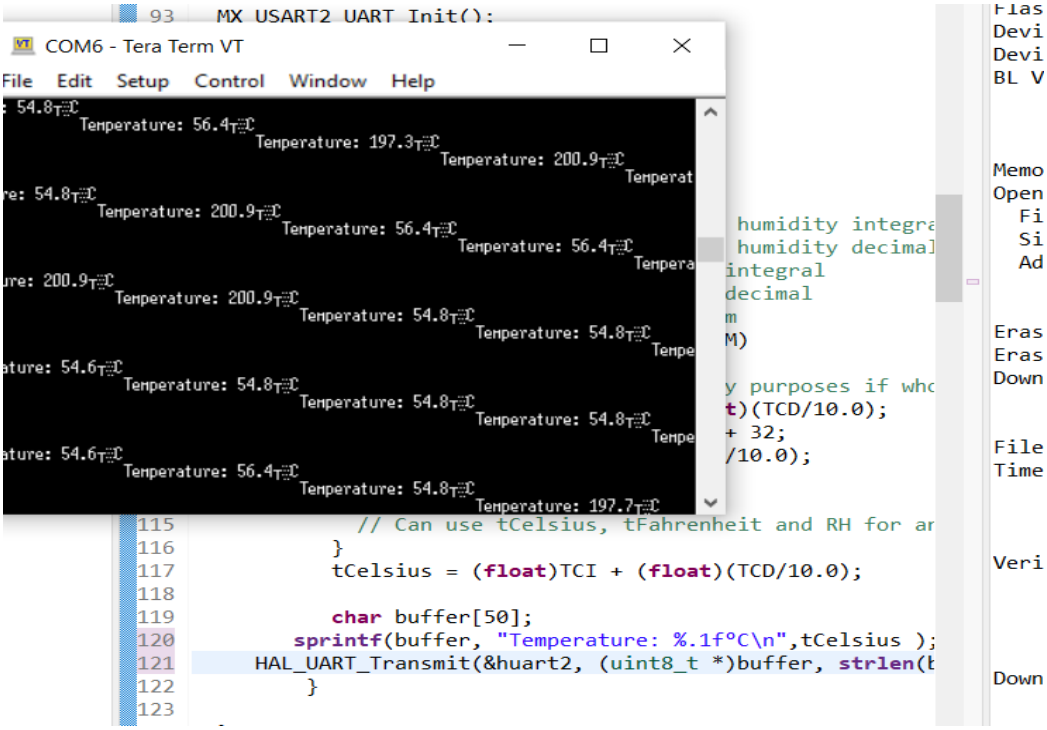
PuTTY est un émulateur de terminal pour Windows permettant la connexion à une machine distante par protocole ssh.

Dans le cadre de mon projet, j'ai utilisé PuTTY pour afficher la distance détectée par le capteur ultrasonique via le protocole UART. (L'utilisation de l'émulateur PuTTY c'est juste de vérifier que les composants électroniques fonctionnent).

Donc je vais sélectionner le port de communication (COM6) et définir la vitesse à 115200 bauds.



Et aussi j'ai utilisé PUTTY pour afficher la température qui détecté par le capteur DHT11 via le protocole UART.



NB : L'utilisation du protocole UART est envisagée pour la visualisation des données relatives à la distance et à la température et les données de la caméra OV7670, sans recourir au convertisseur UART-USB, en raison de la liaison du UART2 avec le STLINK.

- Pour valider le bon fonctionnement de la caméra et la transmission des données via le port série, j'ai également employé l'émulateur PuTTY.

c. *QT Creator :*

Qt Creator est un environnement de développement intégré multiplateforme faisant partie du Framework Qt. Il est donc orienté pour la programmation en C++.

Figure 8:QT Creator logo

6. Les langages de programmation :

a. *Langage C :*

Figure 9:Langage c logo

Le langage c est parfaitement adapté pour l'implémentation des différents algorithmes sur des microcontrôleurs dans des équipements divers tels que les objets connectés, l'électronique médicale de dernière génération, la robotique, la géolocalisation, etc.

b. *Langage C++ :*

Figure 10:C++ logo

Le C++ est un langage de programmation : il permet d'écrire des programmes informatiques, pour créer des applications mobiles ou des jeux vidéo, par exemple. C++ est créé à partir du langage C, dont il étend les fonctionnalités : C++ permet notamment de faire de la programmation orientée objet (POO).

7. Conclusion :

En conclusion, la sélection du bon environnement matériel et logiciel est une étape cruciale dans développer un projet réussi. Pour la réalisation d'une IHM (interface homme machine) j'ai choisi la carte STM32 NUCLEO comme plate-forme matérielle En raison de son prix abordable, de sa flexibilité et de sa fiabilité. De plus, j’ai sélectionné la caméra OV7670 module comme matériel de caméra pour capturer les images.et le capteur ultrason pour détecter la présence des objets (les personnes) et un capteur de température DHT11.

Chapitre 2 : Implémentation et Résultats

I. Introduction :

Dans ce chapitre, je discuterais de l'approche étape par étape adoptée pour construire le système, Détaillant comment chaque composant a été intégré pour fournir une solution complète.

Je vais analyser également les résultats obtenus en testant le système, en mettant l'accent sur les techniques pour réaliser une IHM et Je vais présenter les protocoles que j'ai utilisé pour rendre cette solution possible entre les composant électronique.

II. Les protocoles de communication :

1. Le protocole I2C :

J'ai implémenté le protocole I2C afin de faciliter la communication entre la carte STM32 NUCLEO et la caméra OV7670.

a. Définition :

Le protocole I2C est un protocole maitre-esclave ce qui signifie qu'il y'a un seul dispositif maitre qui contrôle les transferts de données avec un ou plusieurs dispositifs esclave, Le dispositif maître initial et contrôle la communication avec les dispositifs esclaves.

b. Les lignes de communication :

SDA (Serial Data Line) : C'est la ligne de données bidirectionnelle utilisée pour transférer les informations entre les dispositifs connectés.

SCL (Serial Clock Line) : C'est la ligne de signal d'horloge qui synchronise les transferts de données entre les dispositifs.

c. Caractéristiques :

- série
- Half-duplex
- les données sont encapsulé
-Vitesse : entre 100kbps et 3.4Mbps ou 5Mbps(unidirectionnel).

d. Architecture :

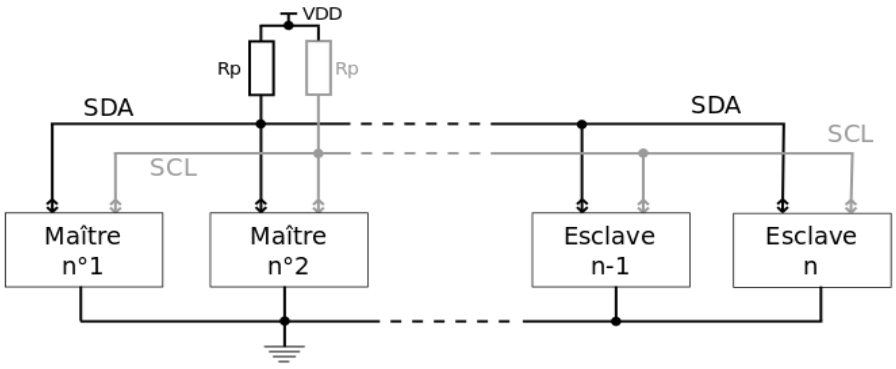


Figure 11:Architecture du I2C

e. La trame I2C :

La trame de I2C est constituée des bits suivants :

Au repos SDA et SCL sont au niveau 1, aucun circuit n'émet.

Début de la Trame (Start Condition) : La trame commence par un signal de démarrage (Start) lorsque la ligne SDA passe de haut (1) à bas (0) alors que la ligne SCL est à l'état haut (1).

Adresse de l'Esclave : Après le signal de démarrage, l'émetteur envoie l'adresse de l'esclave qu'il souhaite communiquer. Cette adresse est généralement composée de 7 bits (parfois 10 bits dans le cas de l'I2C étendu) suivis d'un bit de lecture/écriture (R/W), indiquant si l'émetteur souhaite lire (R=1) ou écrire (W=0) des données à l'esclave.

Bit d'acquittement (ACK) : Après avoir reçu l'adresse de l'esclave, l'esclave envoie un bit d'acquittement (ACK) en mettant la ligne SDA à 0 pour confirmer qu'il a été correctement formulé.

Donnée (Data) : Ensuite, les données sont envoyées, généralement sous forme d'octets, du maître à l'esclave ou vice versa. Chaque octet est suivi d'un bit d'acquittement (ACK) pour indiquer si la transmission a été réussie.

Bit de Stop (Stop Condition) : La trame se termine par un signal d'arrêt (Stop) lorsque la ligne SDA passe de bas (0) à haut (1) alors que la ligne SCL est à l'état haut (1) . Cela indique la fin de la transmission.

Recommencement (Repeated Start) : Si le maître souhaite communiquer avec un autre esclave immédiatement après une transmission, il peut envoyer un signal de redémarrage (Repeated Start) au lieu d'un signal de démarrage.

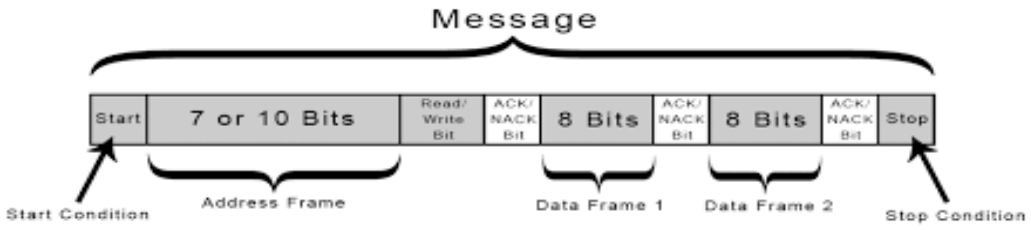


Figure 12:Trame du I2C

***Dans le cadre de ce projet les adresses de la caméra sont :**

```
#define writeAddressSCCB 0x42
#define readAddressSCCB 0x43
```

2. Le protocole UART :

J'ai opté pour l'utilisation du protocole UART afin d'établir la transmission des données entre la carte STM32 NUCLEO et l'ordinateur personnel.

a. Définition :

Le protocole UART (Universal Asynchronous Receiver-Transmitter) est un protocole de communication asynchrone largement utilisé pour transférer des données séries entre des équipements électroniques tels que des microcontrôleurs, des modules RF, des capteurs, des périphériques de communication, etc. la transmission de données bit par bit sur une seule ligne de communication, ce qui le rend simple et efficace pour les communications à courte distance.

b. Les lignes de communication :

TX (Transmit - Émission) : Cette ligne est utilisée par le dispositif émetteur pour envoyer des données au dispositif récepteur. Les données à transmettre sont sérialisées (converties en une séquence de bits) et envoyées sur cette ligne.

RX (Receive - Réception) : Cette ligne est utilisée par le dispositif récepteur pour recevoir les données envoyées par le dispositif émetteur. Les données reçues sont désérialisées (converties de la séquence de bits à leur forme d'origine) et traitées par le dispositif récepteur

NB : Pour garantir une transmission optimale des trames, j'ai opté pour une vitesse uniforme de 115200 bauds pour toutes les applications.

c. Architecture :

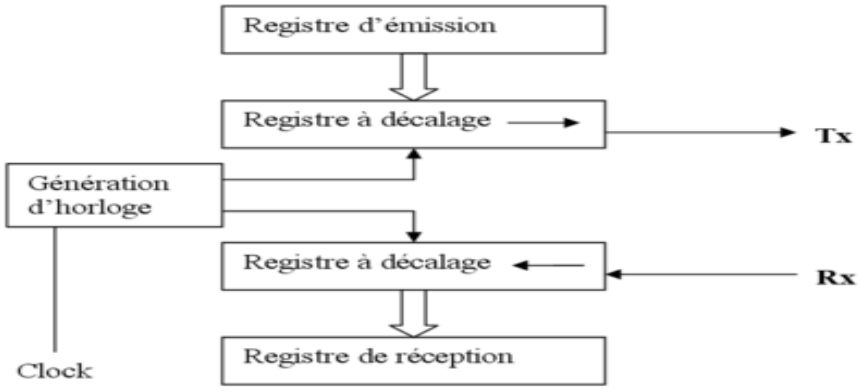


Figure 13:Architecture UART

d. La trame du UART :

Une trame de données dans le protocole UART (Universal Asynchronous Receiver/Transmitter) est la structure qui permet de transmettre un octet (8 bits de données) ou plus de manière asynchrone entre un émetteur et un récepteur.

Début de trame : La trame commence par un bit de départ (start bit) à 0 (basse tension). Le début de la trame est marqué par cette transition de la haute tension (bit d'arrêt précédent) à la basse tension.

Données : Après le bit de départ, les bits de données sont transmis du moins significatif (LSB - Least Significant Bit) au plus significatif (MSB - Most Significant Bit). Le nombre de bits de données peut varier, mais il est généralement de 8 bits.

Ces bits contiennent la charge utile de la trame, telle que des caractères ASCII ou des données binaires.

Bit de parité (optionnel) : Si la parité est activée, un bit de parité est inclus après les bits de données. Ce bit est utilisé pour vérifier l'intégrité des données et peut être pair (even), impair (odd), ou aucun (no parity). La parité est utilisée pour détecter les erreurs de transmission.

Bits de stop : Après les bits de données (et le bit de parité), un ou plusieurs bits de stop sont inclus. Les bits de stop sont toujours à 1 (haute tension). Ils indiquent la fin de la trame et permettent au récepteur de se synchroniser pour la trame suivante. Le nombre de bits de stop est généralement configuré à 1 ou 2.

Fin de trame : La trame se termine par la transition d'un bit de stop (haute tension) au bit de départ suivant (basse tension), marquant ainsi la fin de la trame.

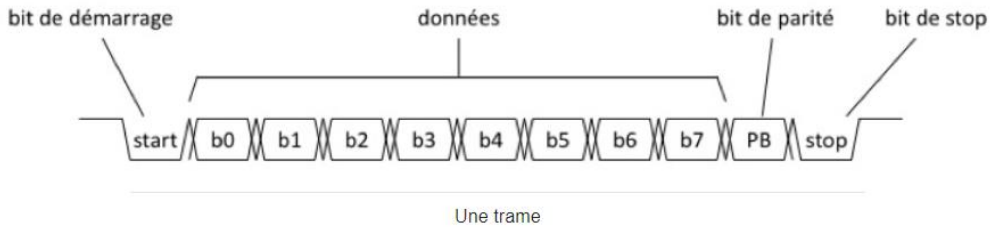


Figure 14:Trame UART

III. Code et résultat :

1. Code partie STM32cubeIDE :

a. Code du capteur ultrason :

Quand le capteur détecte un objet :

```
AL_Init();
ystemClock_Config();
X_GPIO_Init();
X_USART2_UART_Init();
hile (1)
    // Déclencher le signal ultrasonique
                        HAL_GPIO_WritePin(TRIG_PORT, TRIG_PIN, GPIO_PIN_SET);
                        HAL_Delay(10);
                        HAL_GPIO_WritePin(TRIG_PORT, TRIG_PIN, GPIO_PIN_RESET);
                        // Attendre le signal ECHO
                        while (HAL_GPIO_ReadPin(ECHO_PORT, ECHO_PIN) == GPIO_PIN_RESET);

                        // Mesurer le temps de retour de l'écho
                        uint32_t start_time = HAL_GetTick();
                        while (HAL_GPIO_ReadPin(ECHO_PORT, ECHO_PIN) == GPIO_PIN_SET);
                        uint32_t end_time = HAL_GetTick();

                        // Calculer la distance en cm
                        float pulse_width = (float)(end_time - start_time);
                        float distance = pulse_width * 0.034 / 2;

                        //Vérifier la distance et allumer la LED si néces-saire
                          if (distance < 10) {
                             HAL_GPIO_WritePin(LED_PORT, LED_PIN, GPIO_PIN_SET); // Allumer la LED
                           } else {
                             HAL_GPIO_WritePin(LED_PORT, LED_PIN, GPIO_PIN_RESET); // Éteindre la LED
                           }
                    // Envoyer la distance via UART
                        char buffer[50];
                        sprintf(buffer, "Distance: %.2f cm\r\n", distance);
                        HAL_UART_Transmit(&huart2, (uint8_t *)buffer, strlen(buffer), HAL_MAX_DELAY);
```

b. *Code du capteur DHT11 utilisé :*

Ce capteur est déployé pour surveiller la température à bord du train, afin de prendre en compte les variations saisonnières significatives entre l'été et l'hiver. Une considération majeure est d'évaluer si le nombre de passagers excédera la capacité de places disponibles en fonction de la température ambiante.

```
while (1)
{
     if(DHT11_Start())
         {
           RHI = DHT11_Read(); // Relative humidity integral
           RHD = DHT11_Read(); // Relative humidity decimal
           TCI = DHT11_Read(); // Celsius integral
           TCD = DHT11_Read(); // Celsius decimal
           SUM = DHT11_Read(); // Check sum
           if (RHI + RHD + TCI + TCD == SUM)
           {
             // Can use RHI and TCI for any purposes if whole number only needed
             tCelsius = (float)TCI + (float)(TCD/10.0);
             tFahrenheit = tCelsius * 9/5 + 32;
             RH = (float)RHI + (float)(RHD/10.0);
             // Can use tCelsius, tFahrenheit and RH for any purposes
           }
           tCelsius = (float)TCI + (float)(TCD/10.0);

           char buffer[50];
       sprintf(buffer, "%.1f,",tCelsius );
    HAL_UART_Transmit(&huart2, (uint8_t *)buffer, strlen(buffer), 100);
         }
}
```

c. *Code de la caméra OV7670 :*

- En ce qui concerne la caméra OV7670, veuillez trouver ci-dessous un extrait du code associé :

```
#include "ov7670.h"
#include "stm32f0xx_hal.h"
#include "main.h"

extern I2C_HandleTypeDef hi2c2;
//extern SPI_HandleTypeDef hspi1;
extern UART_HandleTypeDef huart2;
extern TIM_HandleTypeDef htim6;
extern TIM_HandleTypeDef htim14;

//This settings includes RGB565 format,QQVGA format and color optimization settings.
//You can find "OV7670 Implementation " document as pdf by OmniVision Company

const uint8_t OV7670_reg[][2] = {
 {REG_COM7, 0x80},
 {REG_CLKRC, 0x91},
 {REG_COM11, 0x0A},
 {REG_TSLB, 0x04},
 {REG_TSLB, 0x04},
 {REG_COM7, 0x04},

 {REG_RGB444, 0x00},
 {REG_COM15, 0xD0},

 {REG_HSTART, 0x16},
 {REG_HSTOP, 0x04},
 {REG_HREF, 0x24},
 {REG_VSTART, 0x02},
 {REG_VSTOP, 0x7a},
 {REG_VREF, 0x0a},
 {REG_COM10, 0x02},
 {REG_COM3, 0x04},
 {REG_MVFP, 0x3f},
 // 3 consecutive lines of code are QQVGA format settings.
 {REG_COM14, 0x1a},
```

```
213
214     //You need to set of Bit[7] for 0x12 Register Address
215     if( HAL_I2C_Mem_Write(&hi2c2, writeAddressSCCB, REG_COM7, I2C_MEMADD_SIZE_8BIT, pData, 1, 10) != HAL_OK)
216     {
217
218         return;
219     }
220 }
221
222 void WriteOperationOVCam(uint16_t memADdress, uint8_t  pData)
223 {
224     if( HAL_I2C_Mem_Write(&hi2c2, writeAddressSCCB, memADdress, I2C_MEMADD_SIZE_8BIT, &pData, 1, 10) != HAL_OK)
225     {
226
227         return;
228     }
229 }
230
231 void ReadOperationOVCam(uint16_t memAddress, uint8_t* buffer)
232 {
233
234     if(HAL_I2C_Master_Transmit(&hi2c2, writeAddressSCCB, (uint8_t*)&memAddress, 1, 10) != HAL_OK)
235     {
236         return ;
237     }
238    if(HAL_I2C_Master_Receive(&hi2c2, readAddressSCCB, buffer, 1, 10) != HAL_OK)
239    {
240        return;
```

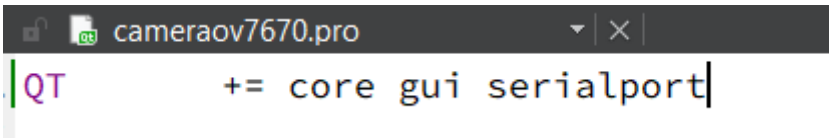
2. Code partie QT Creator :

Module (Qt Serial Port) :

Le port série Qt fournit des fonctionnalités de base pour la configuration, les opérations d'E/S et l'obtention et le réglage des signaux de contrôle des brochages RS-232.

Utilisation du module :(avec Cmake ou qmake)

Dans le cas de mon projet je vais utiliser qmake :



Pour déterminer le nombre de ports connectés à l'ordinateur.

```
qDebug() << "Number of available ports: " << QSerialPortInfo::availablePorts().length();
```

Résultat :

```
01:55:24: Démarrage de C:\Users\nawre\Desktop\pfa\build-cameraov7670-Desktop_Qt_5_15_2_MinGW_64_bit-Debug\d
Number of available ports:  1
```

-Ce code est une boucle foreach utilisée pour parcourir les informations sur les ports série disponibles sur l'ordinateur. Il utilise la classe QSerialPortInfo de la bibliothèque Qt pour obtenir ces informations.

```
foreach(const QSerialPortInfo &serialPortInfo, QSerialPortInfo::availablePorts()){
    qDebug() << "Has vendor ID: " << serialPortInfo.hasVendorIdentifier();
    if(serialPortInfo.hasVendorIdentifier()){
        qDebug() << "Vendor ID: " << serialPortInfo.vendorIdentifier();
    }
    qDebug() << "Has Product ID: " << serialPortInfo.hasProductIdentifier();
    if(serialPortInfo.hasProductIdentifier()){
        qDebug() << "Product ID: " << serialPortInfo.productIdentifier();
    }
```

```
Has vendor ID:  true
Vendor ID:  1155
Has Product ID:  true
Product ID:  14162
```

-Et après, je vais fixer les identifiants, puis je vais mettre le code en commentaire.

```
static const quint16 STM32F030R8T6_vendor_id = 1155;
static const quint16 STM32F030R8T6_product_id = 14162;
```

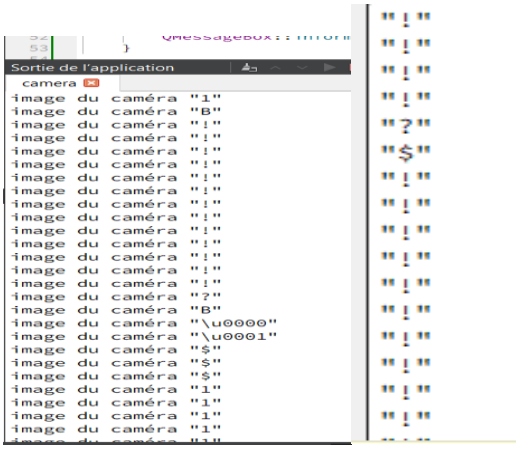
-Configuration du port série :

```
if(STM32F030R8T6_is_available){
    // open and configure the serialport
    STM32F030R8T6->setPortName(STM32F030R8T6_port_name);
    STM32F030R8T6->open(QSerialPort::ReadOnly);
    STM32F030R8T6->setBaudRate(QSerialPort::Baud115200);
    STM32F030R8T6->setDataBits(QSerialPort::Data8);
    STM32F030R8T6->setParity(QSerialPort::NoParity);
    STM32F030R8T6->setStopBits(QSerialPort::OneStop);
    STM32F030R8T6->setFlowControl(QSerialPort::NoFlowControl);
}else{
    // give error message if not available
    QMessageBox::warning(this, "Port error", "Couldn't find the STM32!");
}
```

-Lire les données de l'image à partir du port série par QT Creator et les affiche dans le Widget console :

```
}
void MainWindow::readSerial()
{
   // qDebug()<<"serial port WORKS ";
    QByteArray serialData=STM32F030R8T6->readAll();
   QString image =QString::fromStdString(serialData.toStdString());
    qDebug()<<image;

}
```



-Ce code a été élaboré dans le dessein de convertir les données d'une image provenant du port série en une représentation visuelle affichable au sein d'une Interface Homme-Machine (IHM), rendue possible grâce à QT Creator (C++).

```
   // qDebug()<<"serial port WORKS ";
QByteArray serialData=STM32F030R8T6->readAll();
   QString imageData =QString::fromStdString(serialData.toStdString());
qDebug()<<imageData;
  // QByteArray imageData = STM32F030R8T6->readAll(); // Lire les données de l'image depuis le port série
   //qDebug()<<serialData;//pour afficher les données dans le widget console
   // Convertir les données RGB565 en QImage
   QImage image(reinterpret_cast<const uchar*>(serialData.data()), 320, 120, QImage::Format_RGB16);

   // Afficher l'image sur la QLabel
   ui->imageLabel->setPixmap(QPixmap::fromImage(image));

}
```

- Ce code implémente la fonctionnalité permettant à l'utilisateur d'appuyer sur le bouton "Connecter" afin d'afficher l'image dans l'interface, tandis que l'appui sur le bouton "Déconnecter" permet de faire disparaître l'image.

```
void MainWindow::on_connecter_clicked()
{
    qDebug()<<"user est connecté";
    // Afficher l'image sur la QLabel
    ui->imageLabel->setVisible(true); // Rendre l'image visible
}

void MainWindow::on_disconnecter_clicked()
{
    qDebug()<<"user est disconnecté";
    // Cacher l'image sur la QLabel
    ui->imageLabel->setVisible(false); // Rendre l'image invisible
}
```

3. Câblage et Résultat :

Câblage et la configuration :

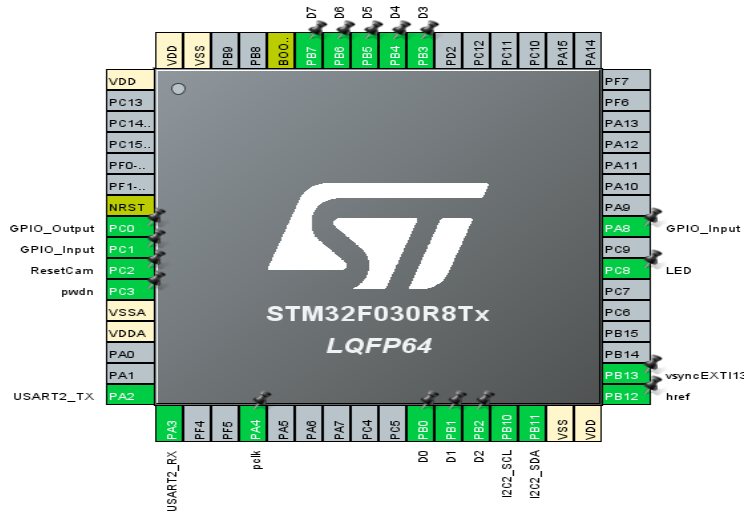
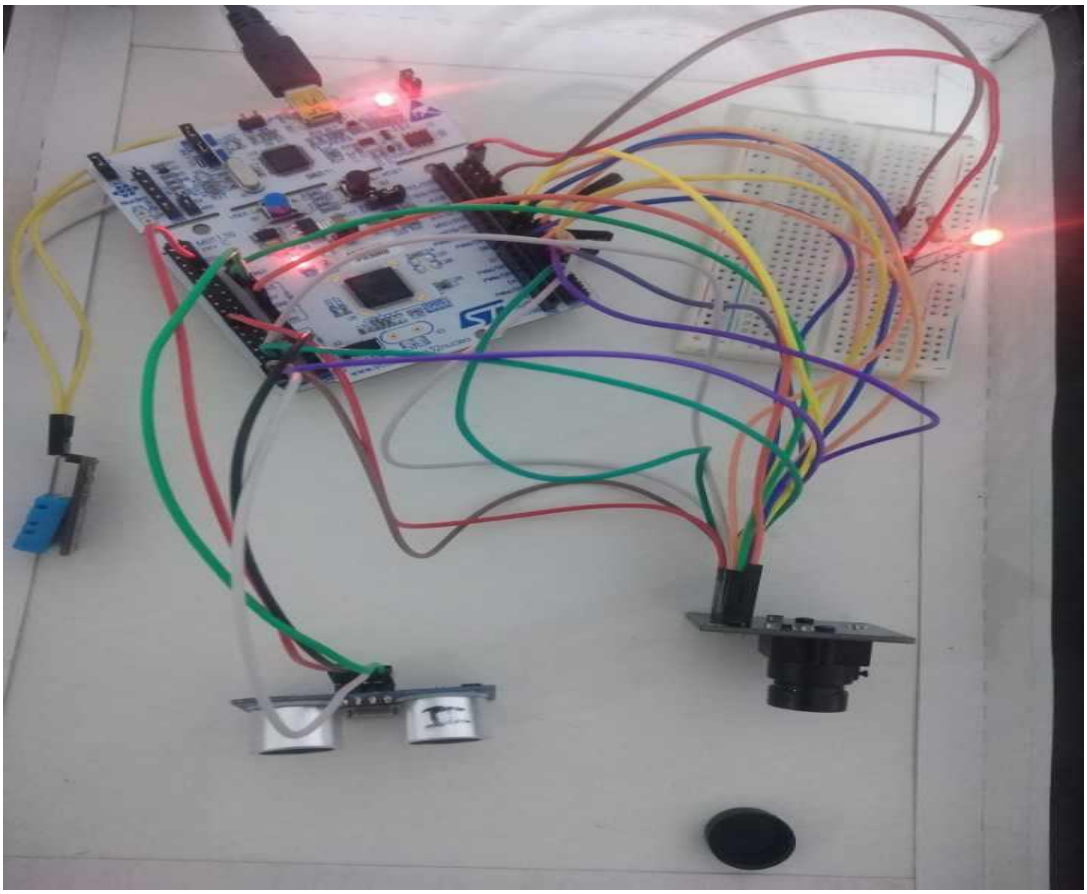
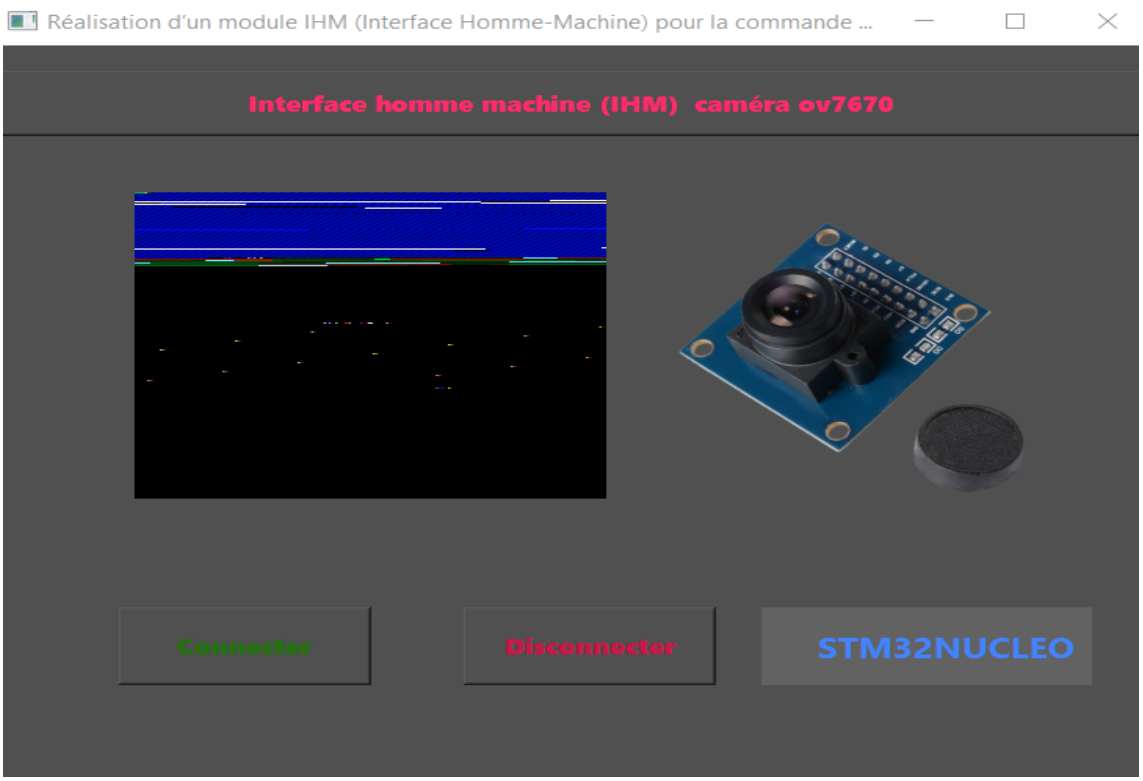
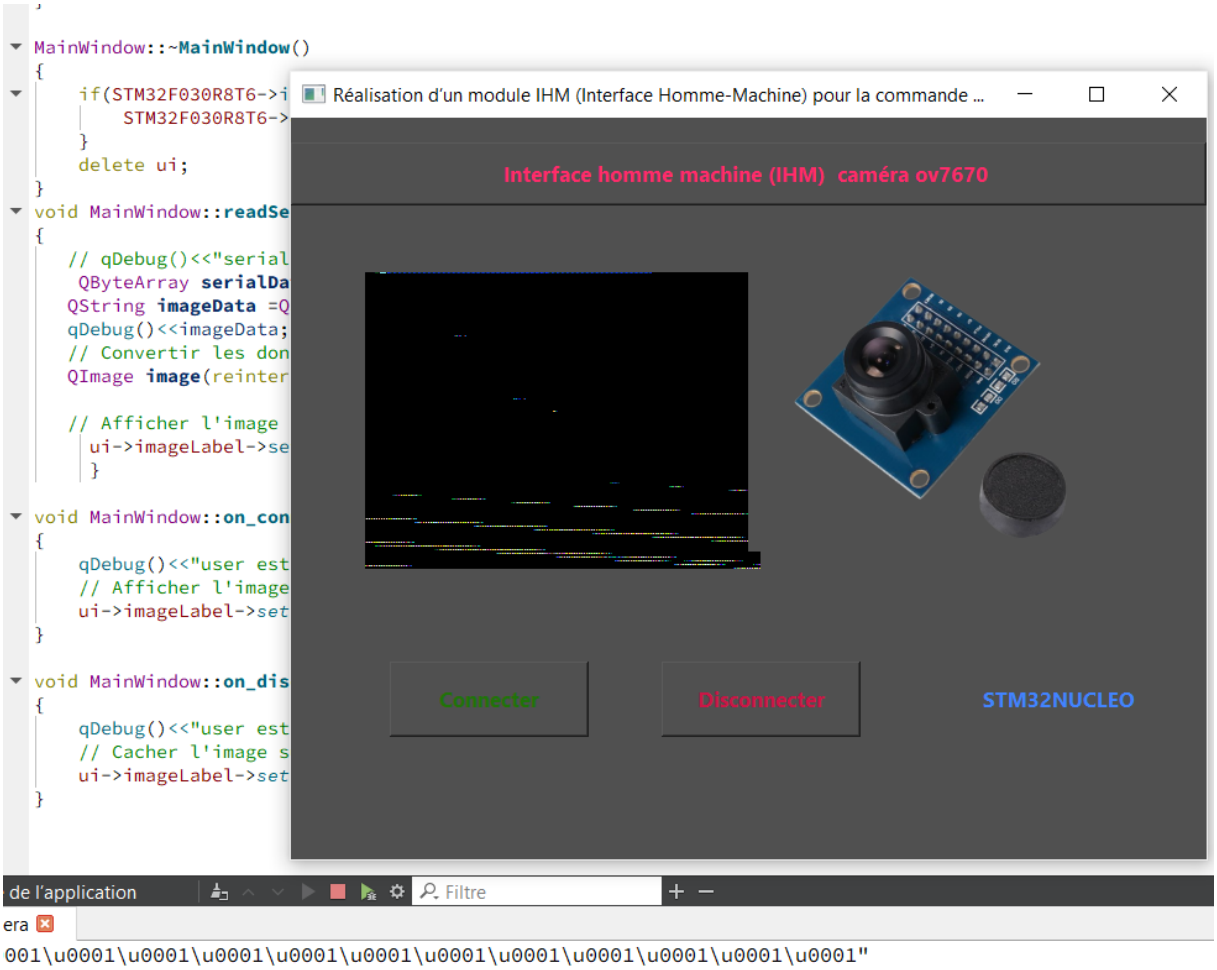


Figure 15:Configuration de la carte STM32NUCLEO



Résultat :


Réalisation d'un module IHM (Interface Homme-Machine) pour la commande ...
Interface homme machine (IHM) caméra ov7670
Connecter
Disconnecter
STM32NUCLEO



4. Conclusion :

Ce chapitre a mis en lumière l'essentiel du processus de développement logiciel et matériel, en se concentrant sur l'intégration et la communication entre les différents composants du système. À travers l'utilisation de STM32CubeIDE pour la programmation du microcontrôleur STM32 NUCLEO et de QT Creator pour le développement de l'IHM, j'ai réalisé une fusion réussie entre le monde matériel et logiciel.

Conclusion générale :

La réalisation de ce projet a permis de concevoir et de développer avec succès un module d'Interface Homme-Machine (IHM) pour la commande d'un système de transmission vidéo embarqué à bord des trains. En utilisant une combinaison de matériel comprenant la carte STM32 NUCLEO, la caméra OV7670, le capteur ultrason et le capteur DHT11, ainsi que des logiciels tels que QT Creator et STM32CubeIDE, et en programmant en langages C et C++, nous avons pu créer une solution innovante répondant aux besoins spécifiques du secteur ferroviaire.

Ce module IHM offre une interface conviviale et intuitive, permettant au personnel ferroviaire de contrôler efficacement le système.

Au cours du processus de développement, plusieurs défis ont été rencontrés et surmontés, notamment l'intégration de différents capteurs et la gestion des protocoles de communication.

En conclusion, ce projet représente une contribution significative à l'amélioration de la connectivité, de la sécurité et de l'efficacité opérationnelle dans le secteur ferroviaire. Il ouvre également la voie à de futures possibilités d'innovation et d'amélioration continue dans le domaine de l'Interface Homme-Machine et des systèmes de contrôle embarqués.

Référence bibliographique :

https://www.qt.io/product/development-tools

https://github.com/mmtcoder/OV7670-Stm32F070Rb/blob/main/Src/ov7670.c

https://doc.qt.io/qt-6/qpixmap.html#details

https://www.technologuepro.com/cours-systemes-embarques/cours-systemes-embarques-Bus-I2C.htm

https://www.st.com/content/st_com/en.html

https://chat.openai.com/

https://www.microsonic.de/fr/support/capteurs-%C3%A0-ultrasons/principe.htm

https://web.mit.edu/6.111/www/f2016/tools/OV7670_2006.pdf

https://www.robot-maker.com/shop/img/cms/datasheet-capteur-ultrasons-hc-sr04.pdf

https://www.mouser.com/datasheet/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf