SCANPARKING

Gestion des autorisations d’accès à des parkings privés

Lycée Vauvenargues, Aix-en-Provence

BTS SN2

Résumé

Ce système permet de vérifier que les véhicules garés dans les parkings privés du lycée Vauvenargues sont autorisés à stationner via un scan des plaques d’immatriculations.

Table des matières

Présentation du projet

Présentation du contexte

Cadre d'utilisation du système

Le projet ScanParking a pour objectif de fournir au personnel de sécurité un système de contrôle des véhicules autorisés à stationner sur les parkings privés du lycée Vauvenargues.

Sans ce système, un agent de sécurité aurait dû contrôler chaque véhicule entrant dans l'établissement afin de vérifier l'identités des individus et ainsi s'assurer que ces personnes soient bien autorisées à pénétrer dans l'établissement.

En plus de la complexité de la tâche, et dans un souci de maintien de la fluidité de la circulation, il est impossible d'y avoir recours.

Le projet consiste donc à développer un terminal portable permettant de contrôler les véhicules stationnés dans les trois parkings du lycée.

Ce terminal se présentera sous la forme d'un pistolet intégrant un scanner et capable de vérifier en très peu de temps si la plaque d'immatriculation appartient à un véhicule autorisé à stationner dans le parking.

Intérêts du système

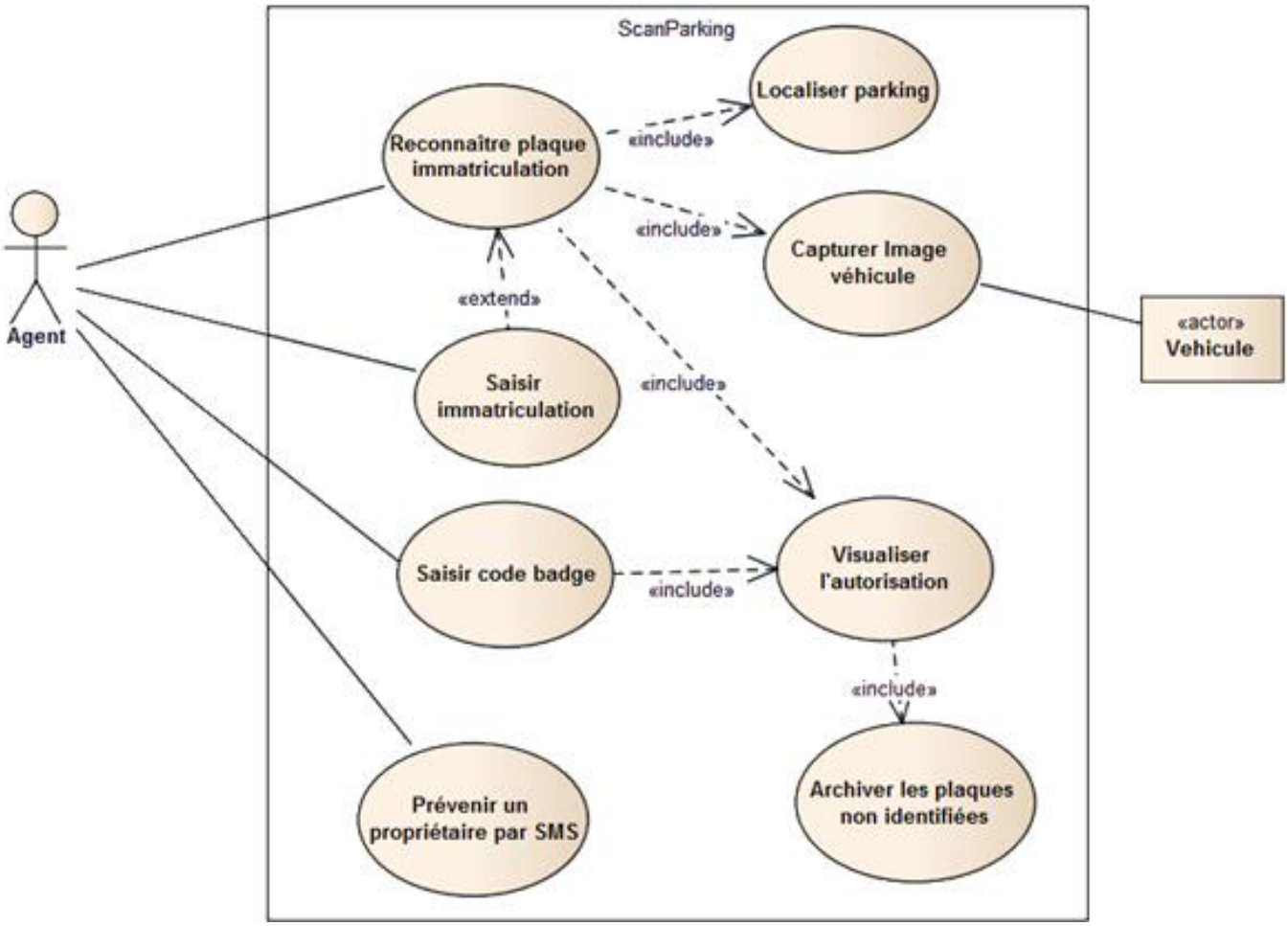
Le système présente la capacité de scanner les plaques d'immatriculations souhaitées tout en fonctionnant rapidement et avec ergonomie. L'agent n'a qu'à viser les plaques, appuyer sur la gâchette, et le système effectue de lui-même la photographie, une analyse pour identifier le numéro de plaque, une géolocalisation du parking, et dans la base de données des plaques enregistrées. Toutes ces actions sont faites en arrière-plan, sans sollicitation de l'agent de sécurité, cependant un écran LCD tactile permet tout même de visualiser les actions du système et d'agir dessus en cas de besoin.

De plus, l'agent a la possibilité de prévenir par SMS le propriétaire d'un véhicule grâce à un bouton de l'interface disponible via l'écran tactile, s'il détecte un problème particulier.

L'agent possède alors un total contrôle sur son outil, ce qui lui permet d'inscrire manuellement les numéros de plaques par exemple, tout en bénéficiant de la vitesse et du confort d'un système automatisé.

Proposition de solution

Cas d'utilisation

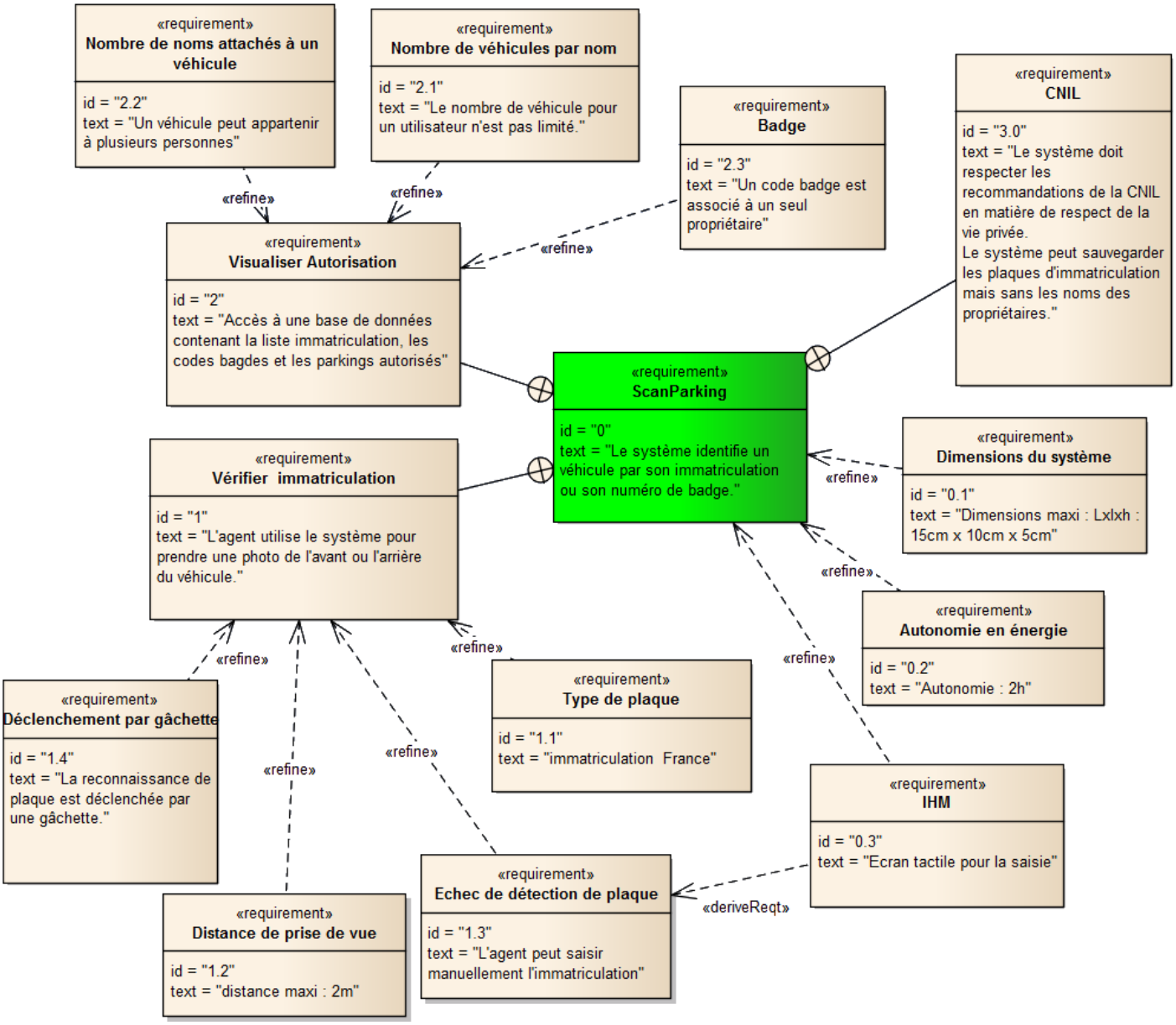


« Reconnaître plaque immatriculation » est le scénario nominal : après avoir appuyé sur la gâchette, le terminal est géolocalisé, la photo est prise, et l'image analysée pour y extraire le numéro de plaque qui sera comparé à la base de données.

Le numéro de plaque ou de badge peuvent aussi être écrits manuellement, ce qui n'implique alors pas la photographie et l'analyse d'image.

Dans le cas où le numéro de plaque ne correspond pas à l'un de ceux existant dans la table des plaques enregistrées, il est géolocalisé, horodaté, et enregistré dans une autre table.

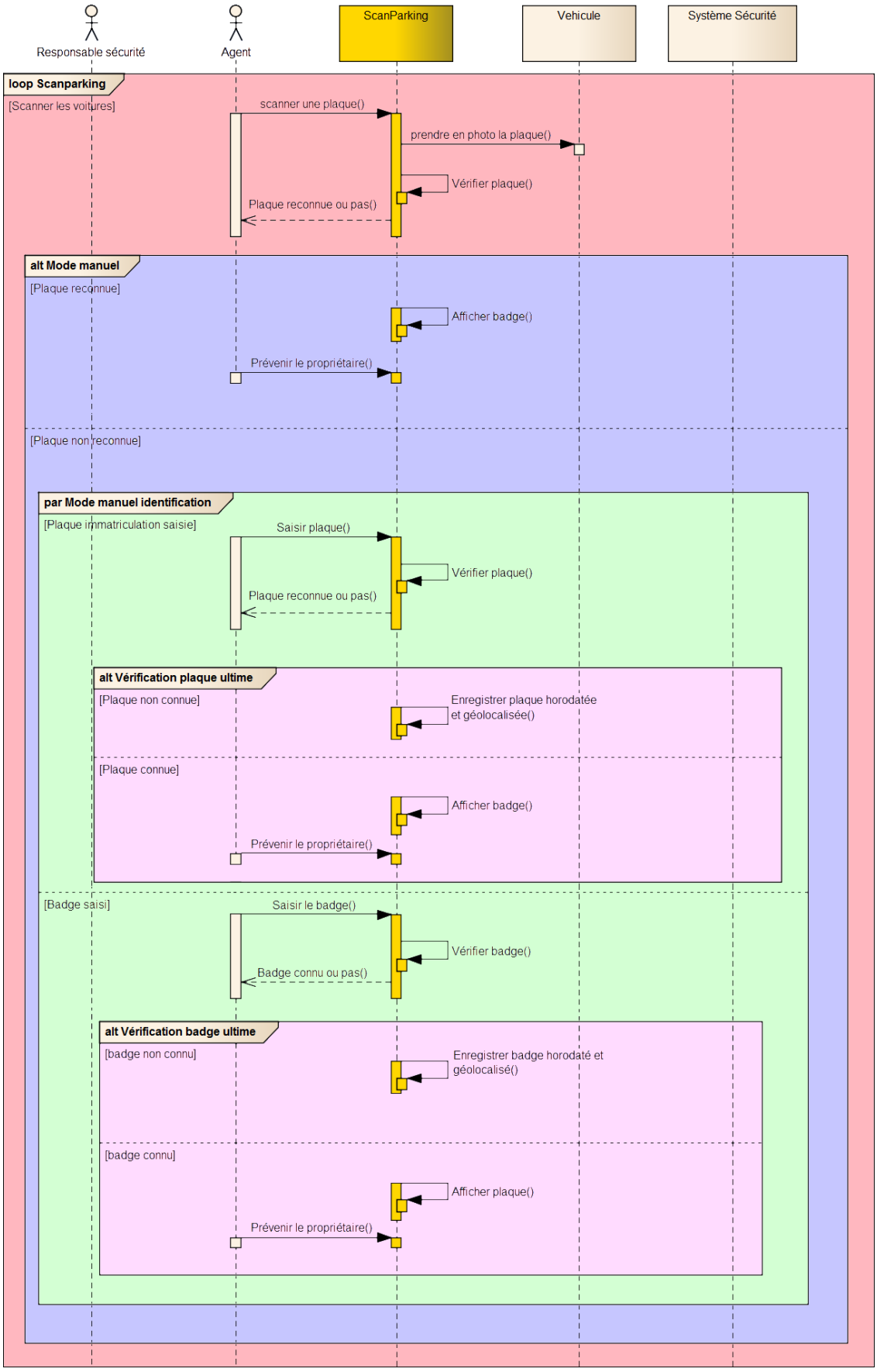
Exigences



Il est nécessaire que la base de données permette à une seule personne de posséder plusieurs véhicules, et inversement. De plus, dans le cadre du respect de la vie privée, le système doit respecter les recommandations de la CNIL, aucun nom ne doit donc apparaitre dans la base de données. A la place, chaque automobiliste doit posséder à un numéro de badge et c'est ce numéro qui sera enregistré.

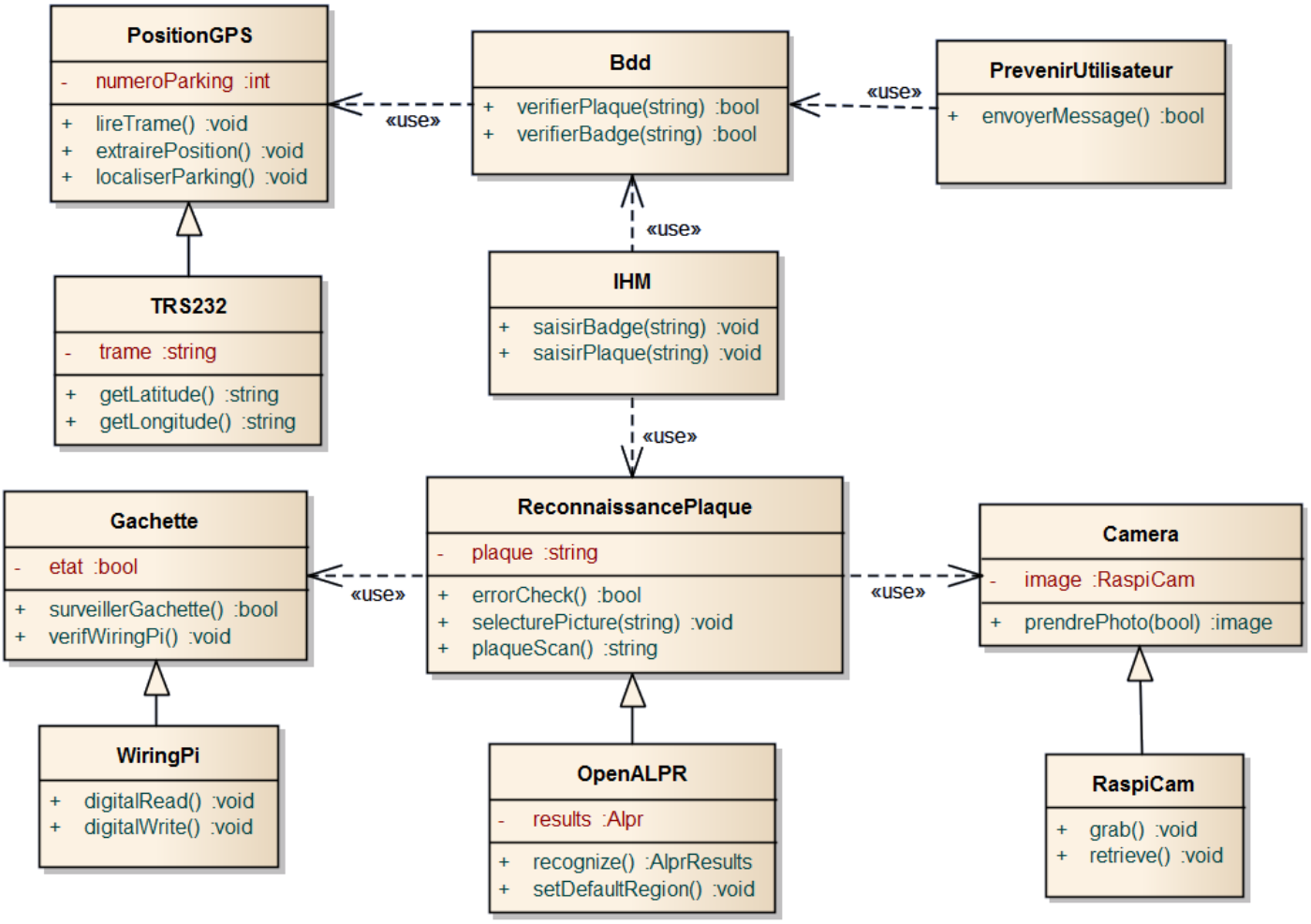
L'agent de sécurité doit aussi avoir accès à une IHM pour pouvoir saisir manuellement l'immatriculation ou le badge.

Fonctionnement global du système



Architecture

Architecture logicielle



Le programme « ScanParking » exécuté sur le Raspberry a été écrit en C++.

Il a été développé en trois parties :

- Capture et analyse de plaques d'immatriculations

  Cette partie est composée des classes « Camera », « Gachette » et « ReconnaissancePlaque »

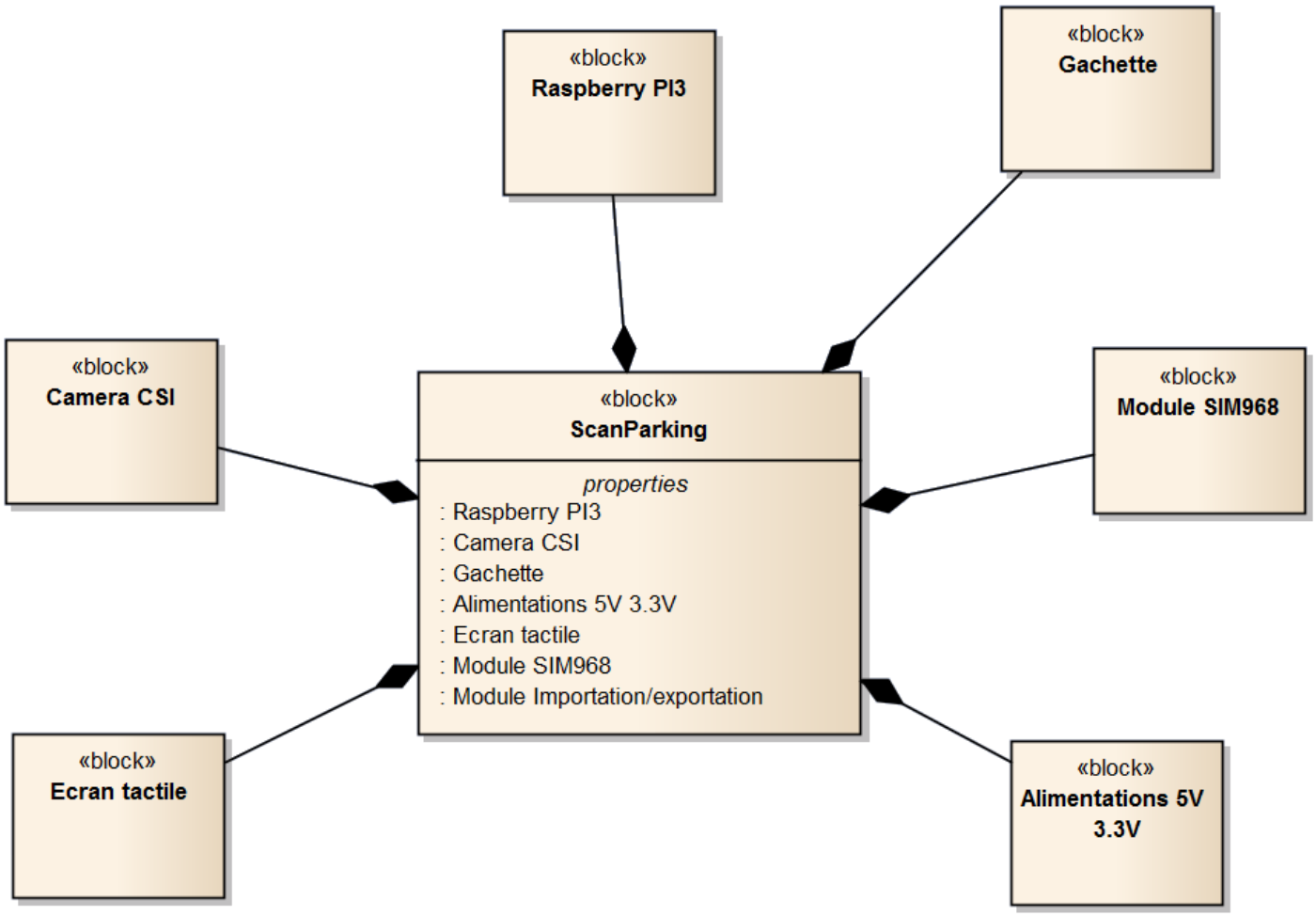
- IHM, connexion et requêtes à la base de données

  Cette partie est composée des classes « IHM » et « Bdd »

- Localisation et avertissement du propriétaire

  Cette partie est composée des classes « PositionGPS » et « PrévenirUtilisateur »

Architecture matérielle



Le système est développé autour d'un Raspberry Pi 3 sur lequel est exécuté le programme qui surveille l'état de la gâchette, envoie l'ordre à la camera CSI de prendre des photographies et les analyse pour y trouver un numéro de plaque, effectue les requêtes SQL pour vérifier les autorisations au stationnement, traite les trames envoyées par le module SIM968 (module GPS et GSM), déclenche l'envoi de SMS, et gère l'IHM, avec laquelle on peut interagir grâce à l'écran tactile LCD.

L'intégralité du système est portable grâce à l'alimentation par une batterie LiPo de 7.5V, dont la tension est abaissée à 5V par un transformateur.

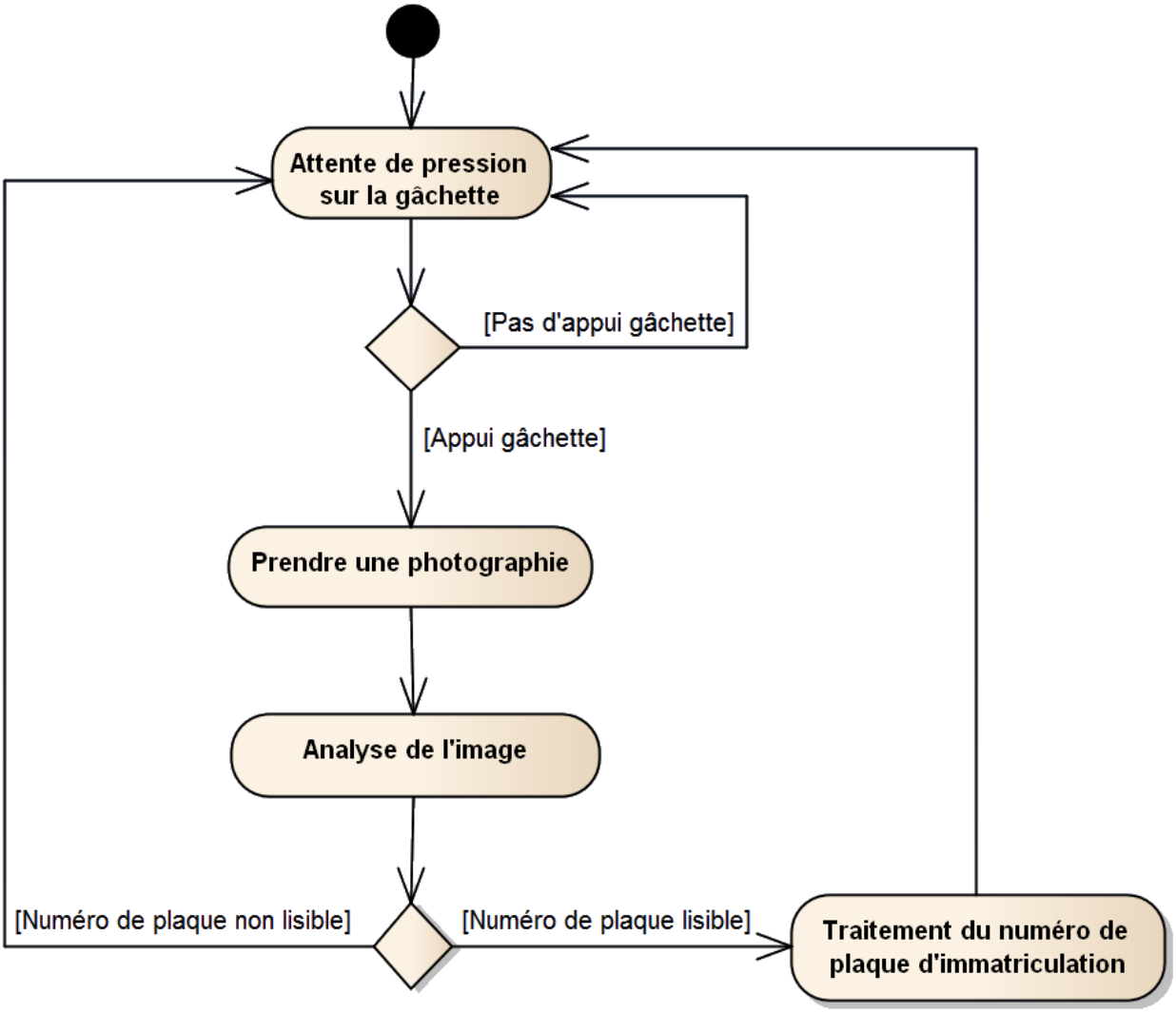
Conception détaillée

Capture et analyse de plaque d'immatriculation

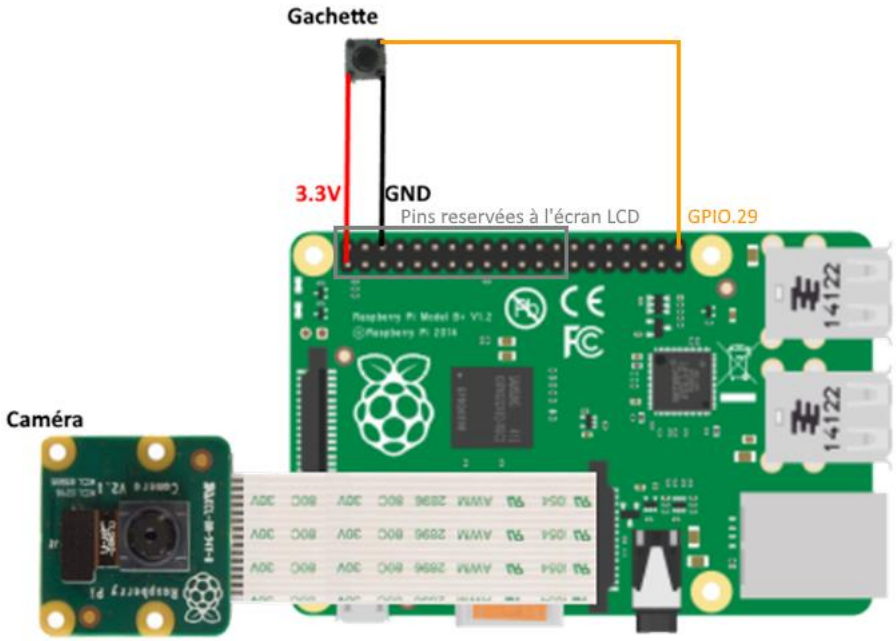
Présentation générale de la partie

La partie « Capture et analyse de plaque d'immatriculation » consiste à vérifier l'état de la gâchette pour détecter les pressions de l'utilisateur. Une fois un appui détecté, la caméra reçoit l'ordre de prendre une prise de vue. L'image est alors analysée par un logiciel de reconnaissance de plaque afin d'en extraire une chaine de caractères correspondant au numéro de la plaque photographiée.

Cette chaine de caractères sera ensuite exploitée pour une requête dans la base de données (voir plus dans la partie « IHM et base de données »). Cette partie a été développée sur Raspbian stretch installé sur un Raspberry Pi 3.



Matériel



La partie « Capture et analyse de plaque d'immatriculation » du logiciel s'occupe de la caméra et de la gâchette (remplacée par un bouton poussoir dans le schéma).

Les pins 3.3V et GND sont déjà utilisées par l'écran LCD, elles sont donc utilisées en dérivation pour le circuit de la gâchette.

La caméra utilisée est une Raspberry Pi Camera Rev 1.3 dont voici les caractéristiques :

Type	Module appareil photo
Nombre de voies	1
Interfaces bus supportées	CSI-2
Résolution maximum supportée	3280 x 2464
Cadence maximum capture	30fps
Dimensions	23.86 x 25 x 9mm
Hauteur	9mm
Largeur	25mm
Longueur	23.86mm
Température de fonctionnement maximum	+60°C
Température de fonctionnement minimum	-20°C

En ce qui concerne la gâchette, la méthode *surveillerGachette()* utilise les méthodes de la librairie WiringPi pour configurer un port GPIO afin de détecter un éventuel signal sur le port GPIO29, généré lors d'un appui sur la gâchette.

Les librairies utilisées

WiringPi :

WiringPi est une librairie écrite en C permettant de faciliter l'utilisation des pins GPIO de toutes les versions de Raspberry Pi. Cette librairie permet de configurer en C ces ports d'entrées et sorties de manière similaire à la façon dont on les configure sur une carte Arduino.

OpenCV :

OpenCV est une librairie graphique open source développée par Intel. Elle permet de traiter des images ou un flux vidéo en temps réel.

RaspiCam :

RaspiCam est une librairie développée par « Aplicaciones de la Visión Artificial » (AVA), permettant de contrôler la caméra Raspberry Pi Camera.

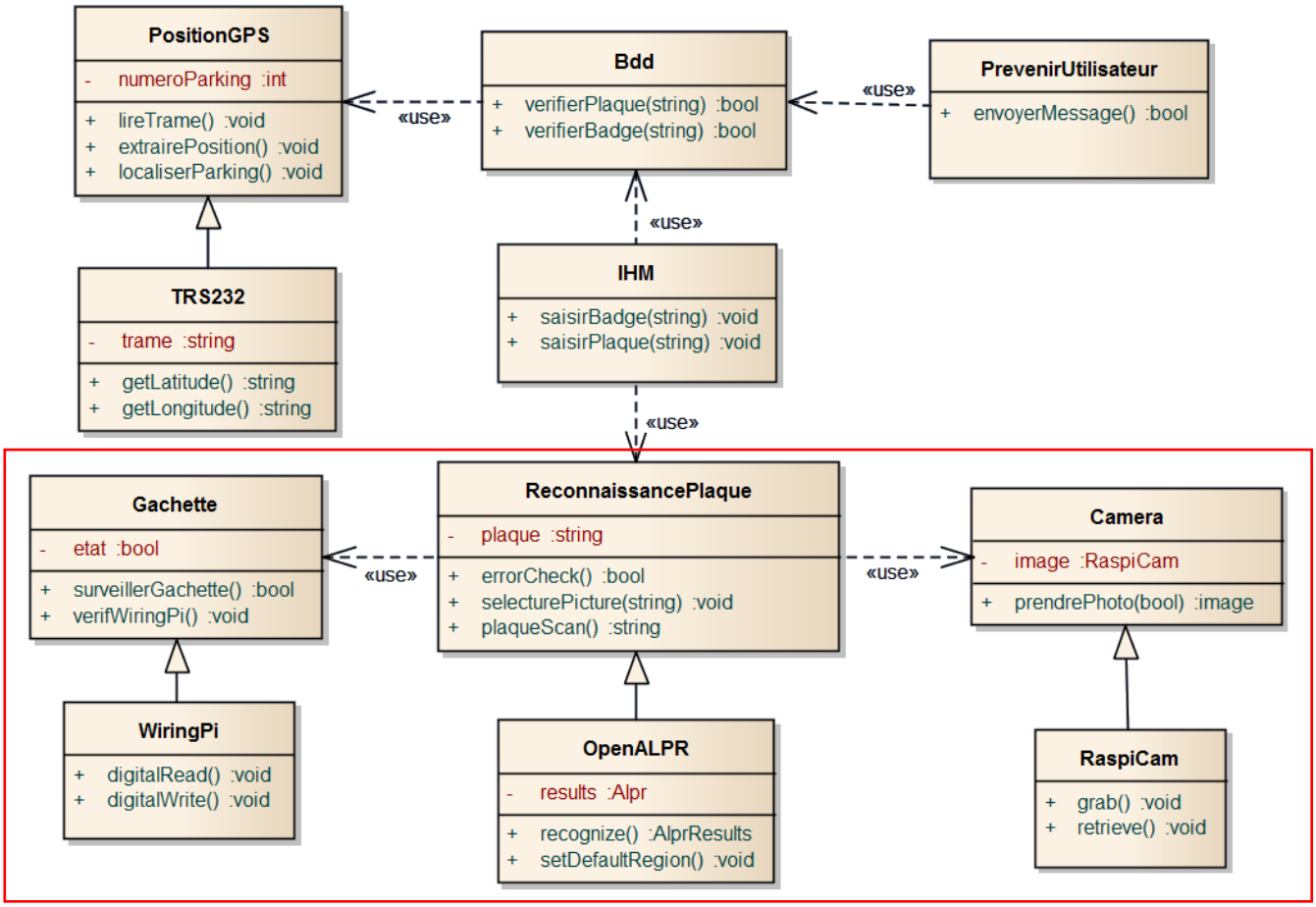
Cette librairie a été choisie car elle répond au besoin du programme et propose une API en C++.

OpenALPR :

OpenALPR (Automatic License Plate Recognition) est une librairie open source écrite en C++. La librairie analyse les images ou flux vidéo afin d'identifier les plaques d'immatriculation, et d'en extraire une chaine de caractères correspondant au numéro des plaques scannées. OpenALPR a été développé à l'aide des librairies OpenCV et Tesseract, un logiciel de reconnaissance optique de caractères (OCR) développé par Google.

Cette librairie a été choisie car elle correspond parfaitement à la tâche que doit effectuer le programme, avec une très bonne fiabilité vis-à-vis de la reconnaissance des caractères.

Structure logicielle



La partie « Capture et analyse de plaque d'immatriculation » du logiciel est composée de trois classes.

La classe « Gâchette » surveille l'état de la gâchette pour détecter les appuis sur cette dernière. Cette classe hérite de la classe « WiringPi » pour configurer le port GPIO29.

La classe « Caméra » possède une méthode *prendrePhoto()* qui donne l'ordre à la caméra de prendre la photographie après appui sur la gâchette.

La classe « ReconnaissancePlaque » hérite de la classe « OpenALPR ». Elle permet de sélectionner l'image prise par la caméra pour l'analyser, et en extraire un une chaine de caractères correspondant au numéro de plaque d'immatriculation présent sur la photographie.

Liste des fichiers : - « Gachette.h » « Camera.h » « ReconnaissancePlaque.h »

- « Gachette.cpp » « Camera.cpp » « ReconnaissancePlaque.cpp »

L'intégralité du programme a été compilé grâce au compilateur g++.

La classe « Gachette » :

Gachette.h :

```
#pragma once
#include <iostream>
#include "wiringpi.h"

class Gachette
{
public:
    Gachette();
    ~Gachette();
    void setup();
    bool surveillerGachette();
};
```

Apres avoir vérifié que la librairie WiringPi est opérationnelle, l'objet « gâchette » appelle les méthodes *setup()* pour configurer le port GPIO et *surveillerGachette()* pour surveiller l'état de la gâchette :

Extrait de Gachette.cpp :

Les méthodes *setup()* et *surveillerGachette() :*

```
void Gachette::setup() {

    pinMode(etatGachette, INPUT);
}

void Gachette::surveillerGachette() {

    if (digitalRead(etatGachette) == HIGH && gachettePression == 0)
    {
        //ACTIVER CAMERA
        gachettePression = 1;
    }
    if (digitalRead(etatGachette) == LOW)
    {
        gachettePression = 0;
    }

}
```

La classe « Camera » :

Extrait de la méthode *PrendrePhoto()* :

```
raspicam::RaspiCam_Cv Camera;
cv::Mat image;

//set camera parameters
Camera.set(CV_CAP_PROP_FORMAT, CV_8UC1); //Format of the Mat objects returned by retrieve()

//Open camera
if (!Camera.open())
{
    cerr << "Error opening the camera" << endl;
    return -1;
}

//Start capture
Camera.grab();              // Grabs the next frame from capturing device
Camera.retrieve(image);     // Save the grabbed frame
Camera.release();           // Close capturing device
```

La capture est effectuée lors de l'appel de la méthode *grab()* qui suit l'appui sur la gâchette. Elle est stockée dans la variable *Image* grâce à la méthode *retrieve()*. L'image peut aussi être convertie en fichier « .jpg » par la méthode *imwrite* d'OpenCv.

La classe « ReconnaissancePlaque » :

ReconnaissancePlaque.h :

```
#pragma once
#include <alpr.h>
#include <iostream>
#include <string>


class ReconnaissancePlaque
{
private:

    std::string plaque;
    alpr::Alpr openalpr;
    alpr::AlprPlateResult plate;
    alpr::AlprPlate candidate;
    alpr::AlprResults results;

public:

    ReconnaissancePlaque();
    ~ReconnaissancePlaque();
    bool errorCheck();
    void selectPicture(std::string &picture_path);
    void plaqueScan();
    void setPlaque(std::string &numeroPlaque);
    std::string getPlaque();

};
```

Extrait de *ReconnaissancePlaque.cpp* :

Méthode *errorCheck()* :

```
bool ReconnaissancePlaque::errorCheck()
{
    if (openalpr.isLoaded() == false)
    {
        std::cerr << "Error loading OpenALPR" << std::endl;
        return 1;
    }
    else
    {
        return 0;
    }
}
```

La méthode *errorCheck()* permet de vérifier que OpenALPR est prêt à être utilisé par le programme, et affiche un message en cas d'erreur, par exemple si le fichier « openalpr.conf » n'est pas trouvable.

Méthode *augmenterLuminosite()* :

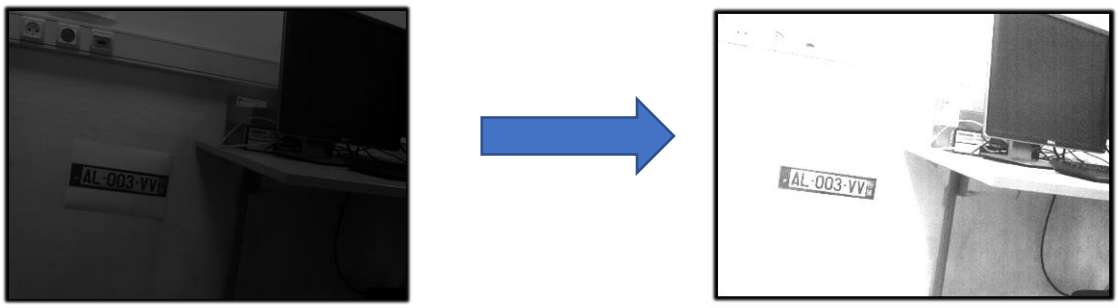
```
double modifContraste = 10; // contrast
int modifLuminosite = 50;  // brightness

// Read the image file
Mat image = imread("raspicam_cv_image.jpg");

// Check for failure
if (image.empty())
{
    cout << "Could not open or find the image" << endl;
    cin.get(); //wait for any key press
    return -1;
}

Mat imageBrighnessHigh100;
image.convertTo(imageBrighnessHigh100, -1, modifContraste, modifLuminosite);
```

La méthode *augmenterLuminosité()* permet d'augmenter le contraste et la luminosité de l'image avant de la scanner, grâce à la méthode *ConverTo() d'*OpenCv. Cela permet de faire ressortir les caractères et de pallier le problème du faible temps d'exposition de la caméra et donc de la faible luminosité de l'image.



Méthode *PlaqueScan()* :

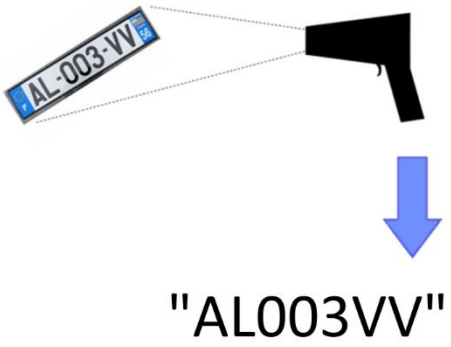
```
alpr::AlprResults results = openalpr.recognize("Images/Plaque_fr.jpg");

alpr::AlprPlateResult plate = results.plates[0];
alpr::AlprPlate candidate = plate.bestPlate;
std::cout << candidate.characters << std::endl;
```

La méthode *recognize()* permet à OpenALPR d'analyser les plaques d'immatriculation de la photographie passée en paramètre. Après cette analyse, *results* est composée de toutes les plaques d'immatriculation trouvées sur l'image, classées de la plaque la plus en haut de l'image à celle la plus en bas, ainsi que pour chacune d'entre elles, des numéros de plaque possible et des indices de confiance associés (toujours en majuscule et sans espace).

Ensuite *plate* prend comme valeur la liste de numéros de plaque possible pour une plaque donnée (ici la première, si on considère qu'il n'y a qu'une plaque sur la photographie).

Enfin, *candidate* prend comme valeur la chaine de caractères associé au plus haut indice de confiance, c'est cette chaine de caractères qui sera convertie en *string* et utilisée par la partie « IHM et base de données » du projet.



Tests unitaires

Le développement du programme a été segmenté, et chaque classe testée indépendamment.

Programme de surveillance de l'état de la gâchette :

Un affichage (std::cout) a été inséré à la place de la méthode qui ordonne la prise de vue de la caméra.

On s'attend donc à voir cet affichage apparaitre quand on appuie sur la gâchette, et une seule fois par pression.

```
pi@raspberryGuillaume:~/testPIN $ ./testPIN
Appui !
Appui !
Appui !
```

La gâchette est bien détectée par le programme et l'affichage apparait exactement quand on s'y attend.

Programme d'utilisation de la caméra :

L'exécution du programme simule l'appui sur la gâchette.

Après l'exécution, on s'attend à ce que la caméra prenne une photographie de qualité suffisante pour être scanner par OpenALPR.

```
pi@raspberryGuillaume:~/RaspiCamTest $ ./test
Opening Camera...
pi@raspberryGuillaume:~/RaspiCamTest $
```

Le programme ordonne à la caméra de prendre la photographie avec succès, et on accède à cette dernière sans problème.

Cependant, il existe un cas de figure pour lequel la photographie n'est pas satisfaisante. En effet, le faible temps d'exposition de la prise de vue implique une faible luminosité de l'image. Dans le cas où la luminosité ambiante est trop faible, l'image ne peut être scanner par OpenALPR car trop sombre.

Pour corriger au mieux ce problème, on a le choix entre augmenter le temps d'exposition pour augmenter la luminosité de l'image, ou augmenter cette dernière artificiellement après avoir pris la photographie.

Cette deuxième option a été choisie car augmenter le temps d'exposition aurait demandé à l'agent de rester immobile un moment afin de ne pas rendre flou l'image, ce qui aurait posé un nouveau problème. On applique donc un post-traitement à l'image pour augmenter fortement sa luminosité et son contraste grâce à la méthode convertTo() de la classe OpenCv, dans le but de rendre les caractères plus lisibles pour OpenALPR.

Programme de reconnaissance de plaque :

On ordonne au programme de scanner une image précédemment prise par la caméra et on s'attend à ce que le programme affiche le numéro de plaque présent sur l'image.

```
pi@raspberryGuillaume:~/ScanParking_backup $ ./ScanParking
AL003VV
```

Le programme réagi comme convenu, et affiche la plaque d'immatriculation présente sur l'image.

Néanmoins, il est nécessaire que le contraste de l'image permette aux caractères de se détacher du fond.

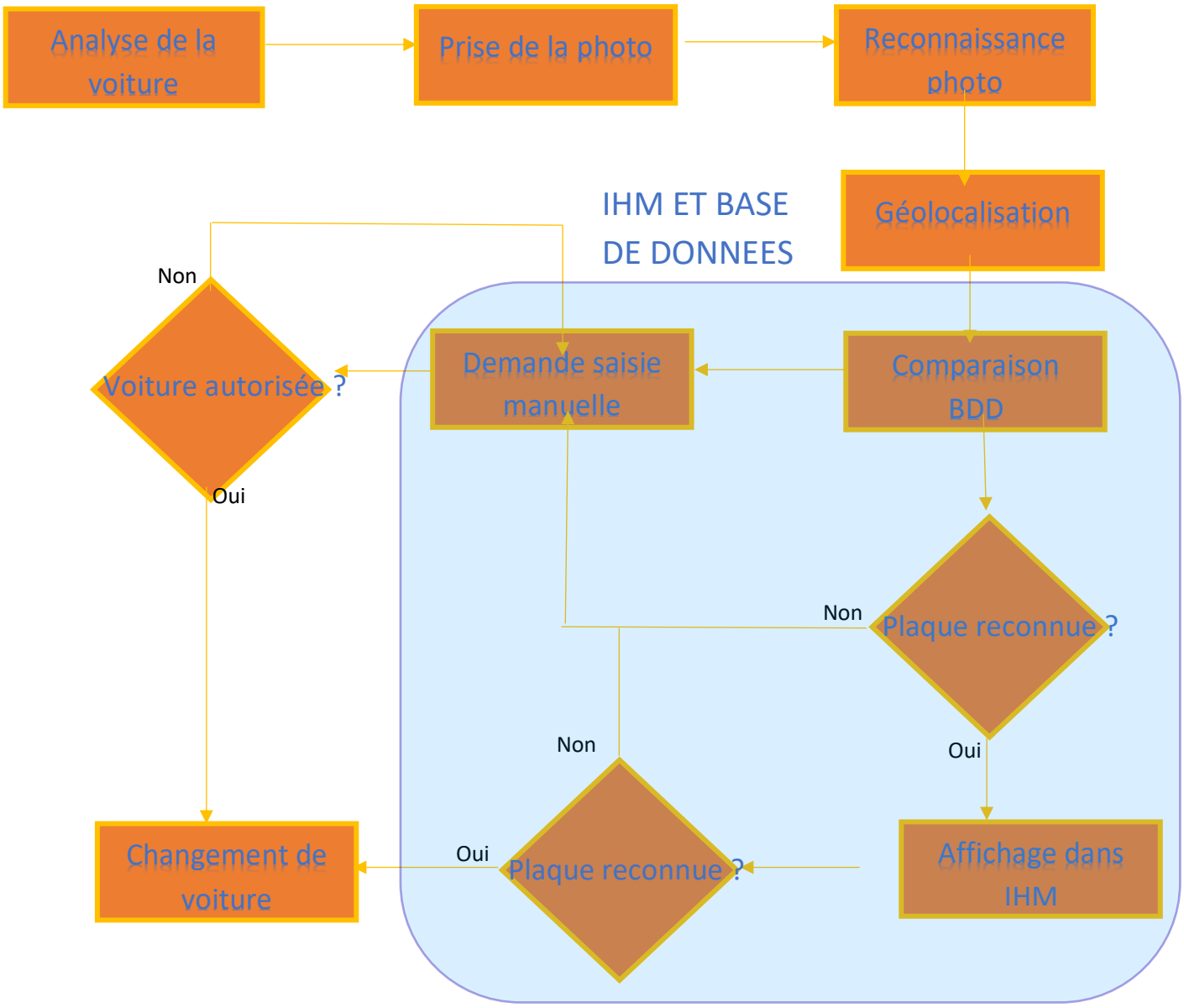
Une image trop sombre pourrait donc ne pas convenir à OpenALPR.

IHM et base de données

Objectifs

Le but de cette partie est de créer une interface Homme-machine visible sur l'écran de notre boitier portable ainsi qu'une base de données interne au boitier permettant au programme du micro-ordinateur Raspberry pi 3 de comparer les plaques d'immatriculation des voitures des trois parkings du lycée avec les données enregistrées dans la base de données afin de savoir si les voitures sont autorisées à stationner. Les informations pourront être entrées par la reconnaissance photo conjointement à la géolocalisation ainsi que manuellement par l'agent de sécurité.

Je vais pour réaliser ma partie personnelle utiliser un microcontrôleur Raspberry pi 3, un écran LCD de 5 '' pour la partie matérielle car il convient parfaitement à notre cahier des charges et pourrait également s'insérer dans un boîtier. Ainsi que le langage de programmation C++ et MySQL pour la base de données en ce qui concerne la partie virtuelle.



Outils et matériel

Dans le cadre de mon projet, le code servant de liaison entre l'interface graphique réalisée avec Qt Creator, présenté ci-dessous et la base de données par les actions de l'utilisateur, qu'elles soient manuelles comme automatique à l'aide du capteur photographique s'exécutent en C++. J'ai utilisé le C++ car c'est un langage polyvalent demandé dans le cahier des charges et un langage que je manipule. Les actions, une fois captées par les entrées de l'interface graphique de Qt ou une fois la gâchette servant à la prise de photo appuyée seront lues par le code et analysées selon le cas pour décider quelles informations seront renvoyées à destination de l'utilisateur.

Qt a été le support de l'interface graphique de l'écran présentée plus loin dans le rapport et Qt a également servit à la création de mon code en C++ pour lier la base de données ainsi que le traitement des informations venant de cette dernière conjointement avec les informations provenant des voitures pour définir la légitimité qu'ont les voitures à stationner dans un des parkings du lycée. Qt est muni de widgets permettant de récupérer des données entrées à l'aide d'un clavier virtuel dans notre cas ou bien d'afficher des valeurs ou réponses pour transmettre à l'utilisateur le résultat du code. J'ai utilisé Qt car c'est un framework que nous avons utilisé au long de notre formation et que les interfaces graphiques qu'il est possible de créer s'adaptent à notre besoin.

.

VNC Viewer était une facilité afin de travailler directement sur le Raspberry avec une interface graphique sur mon système d'exploitation par transmission Wi-Fi sans connexion physique. N'ayant au début pas d'écran, il a été possible de tester la résolution de l'écran par rapport à la taille de l'interface avant la réception de ce dernier.

MySQL a été indispensable pour créer ma base de données de test ainsi que pour réaliser les requêtes appelant la base de données à l'intérieur du code. Une base de données préexistante répertoriant toutes les plaques d'immatriculations, badges, parkings autorisés ainsi que les numéros de téléphone de l'intégralité du personnel est fournie par l'agent de sécurité.

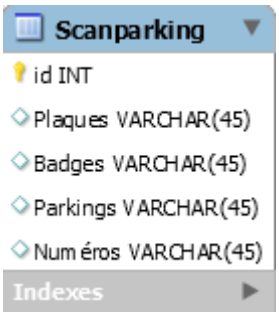
Le Raspberry est notre support de programmation et contient également la base de données, on lancera au démarrage de ce dernier le programme ScanParking lancé en plein écran sur notre écran LCD Display 5".

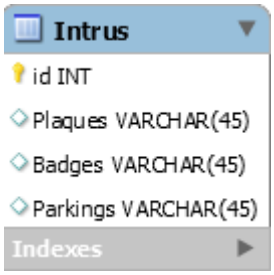
L'écran tactile **LCD Display 5" Touchscreen** branché sur le Raspberry via ses ports GPIO et transmettant le flux vidéo en HDMI affiche notre IHM au lancement de ce dernier. Nous avons choisi d'utiliser cet écran car il convient aux dimensions et est tactile

Réalisation

Afin de réaliser les différentes tâches qui m'incombaient, j'ai tout d'abord installé les librairies libmysql, libmysqlclient et mysql-server pour pouvoir utiliser MySQL dans mon code. N'ayant pas la base de données réelle contenant les données de tous les professeurs du lycée, j'ai dû créer une version de test en localhost comme présenté ici :

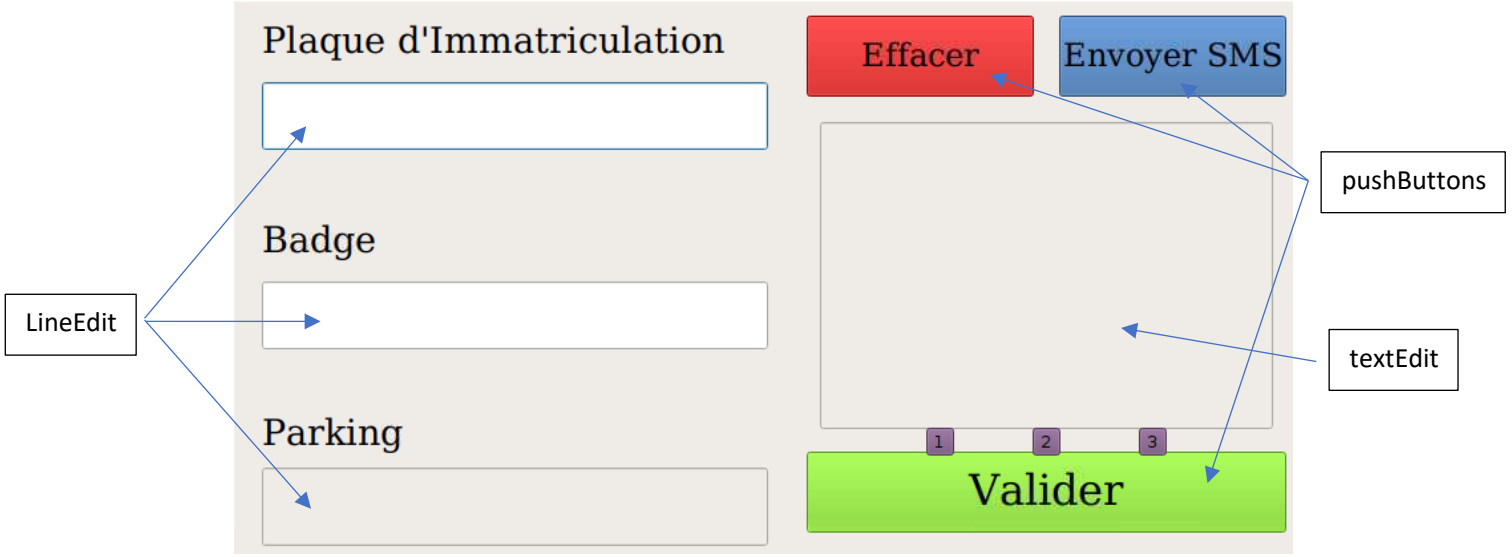
ID	Plaques	Badges	Parking	Numéro
1	AA123AA	A52	1	0618394884
2	BB456BB	B85	1	0694621758
3	CC789CC	C102	1	0621987463
4	DD123DD	D91	2	0665987432
5	EE456EE	E50	2	0694738512
6	FF789FF	F12	2	0697513495
7	GG123GG	G51	3	0721948360
8	HH456HH	H108	3	0687446831
9	II789II	I149	3	0697843218
10	JJ123JJ	A52	1	0678916345





Voici la représentation des deux tables de notre base de données, Scanparking qui contient toutes les données des utilisateurs de voitures du lycée.

Ayant maintenant une base de données comme support, j'ai donc pu commencer à créer le code sur l'IDE QtCreator ayant pour but de comparer les informations entrées dans le code qui pour le moment seront entrées manuellement avec la base de données locale. Il me fallut alors faire appel aux librairies présentées plus haut puis créer l'IHM que voici :



Ici, je transmets les valeurs contenues dans les lineEdit de Qt représentant les trois variables définissant un véhicule, leur plaque d'immatriculation, leur badge et leur parking à ma classe Bdd via un système de setter.

Puis en fonction du cas d'utilisation, à savoir si la Plaque d'immatriculation ou le badge sont connus, on effectuera la requête correspondante. Les résultats du traitement des requêtes détaillées dans la classe BDD expliquée plus bas seront renvoyés dans la structure maStruct avec la plaque, le badge et le parking en paramètres. Ces résultats seront par la suite affichés dans les lineEdit correspondants grâce à la méthode setText().

```
void MainWindow::on_pbValider_clicked()
{

    m_bdd.setter(ui->lePlaque->text(), ui->leBadge->text(), ui->leParking->text());

    if (ui->lePlaque->text().size() != 0 && ui->leBadge->text().size() == 0){
        maStruct Struct2 = m_bdd.Select("SELECT * from Scanparking WHERE Plaques='" + ui-
        ui->leBadge->setText(Struct2.badge);
        ui->leParking->setText(Struct2.parking);
        ui->leMessages->setText("Plaque valide");
    }

    else if (ui->leBadge->text().size() != 0 && ui->lePlaque->text().size() == 0){
        maStruct Struct2 = m_bdd.Select("SELECT * from Scanparking WHERE Badges='" + ui->
        ui->lePlaque->setText(Struct2.plaque);
        ui->leParking->setText(Struct2.parking);
        ui->leMessages->setText("Badge valide");
    }
    else
    ui->leMessages->setText("Voiture non autorisée");

}
```

Afin de pouvoir utiliser la syntaxe de MySQL dans ma classe BDD et vérifier que notre code a bien effectué la connexion à cette dernière, j'effectue la connexion puis un test d'état.

```
Bdd::Bdd()
{
    db = QSqlDatabase::addDatabase("QMYSQL");
    db.setHostName("localhost");
    db.setUserName("root");
    db.setPassword("scanparking");
    db.setDatabaseName("Scanparking");

    if(db.open())
    {
        qDebug() << "Vous êtes maintenant connecté à BDD";
        db.close();
    }
    else
    {
        qDebug() << "Vous n'avez pas réussi à vous connecter a la" << "BDD";
    }

}
```

Une fois certains que la base de données est bien connectée à notre code, il faut préparer la requête transmise depuis la classe mainWindow gérant la partie interface homme machine. A l'aide de la méthode prepare ainsi que next, les informations que nous recherchons provenant des colonnes respectives de notre table peuvent être enregistrées dans différentes variables en convertissant les valeurs de la requête en variable de type string

```
maStruct Bdd::Select(const QString & str_query)
{
    if(!db.open())
    {
        qDebug() << "not opened";
    }

    QSqlQuery q;
    q.prepare(str_query);
    //q.bindValue(":valeur", m_linePlaque );
    q.exec(str_query);
    q.next();

    qDebug() << str_query;

    maStruct l_Struct;
    l_Struct.plaque = q.value(1).toString();
    l_Struct.badge = q.value(2).toString();
    l_Struct.parking = q.value(3).toString();
    l_Struct.numero = q.value(4).toString();
    return l_Struct;

}

void Bdd::setter(QString plaque, QString badge, QString parking)
{
    m_linePlaque = plaque;
    m_lineBadge = badge;
    m_lineParking = parking;

}
```

Problèmes rencontrés

Au cours de mon projet, j'ai été confronté à de nombreux problèmes, réfléchir, m'aider des forums sur internet ou appeler une personne ayant trouvé la solution ou ayant des pistes voir même appeler un professeur ont été les moyens de les résoudre.

- Tout d'abord, afin de compiler mon programme sous Qt, une erreur de compilation car le compilateur n'est pas reconnu :

```
Error while building/deploying project test (kit: Desktop)
When executing step "qmake"
```

Afin de la résoudre, j'ai dû ajouter manuellement le compilateur en C et C++ sur Qt, et ce problème étant récurrent à chaque redémarrage de Qt, lancer l'exécutable de mon programme au lancement du Raspberry permet d'éviter cette contrainte.

- Lors de l'intégration des librairies MySQL, la compilation était impossible du fait de cette erreur récurrente :

```
QSqlDatabase: QMYSQL driver not loaded
QSqlDatabase: available drivers: QSQLITE
```

J'ai pu après de longues recherches et plusieurs essais infructueux finir par comprendre qu'il s'agissait d'un problème de drivers n'étant pas indiqués sur l'installation de MySQL, celui nommé libqt5sql5-mysql.

- Notre programme étant réalisé avec Qt, au lancement du Raspberry il fallait aller chercher le dossier contenant le fichier exécutable pour le lancer. L'enjeu de ce problème résidait dans le fait de lancer le programme au démarrage du Rasperry. J'ai donc créé un fichier de service dans /lib/systemd/system allant chercher au démarrage du micro-ordinateur le chemin de l'exécutable pour l'ouvrir sans avoir aucune manipulation à effectuer. Une fois ce fichier de service activé, le reste d'effectuera à chaque démarrage.

Tests fonctionnels

Test Fonctionnel : Test de connexion à la base de données	
Objectif :	Vérifier la connexion à la base de données
Eléments à tester :	■ Base de données
Pré requis :	■ Base de données sur le Rasperry nommée Scanparking ■ Code développé en C++

Id	Démarche	Comportement attendu	OK ?
1	Se connecter à la base de données via Qt	Connexion effective à la BDD	OK

Fonctionnalité :

V Excellente
❑ Bonne
❑ Moyenne
❑ Faible

Test Fonctionnel : Test de requête à la base de données	
Objectif :	Vérifier si les données transmises au code par l'IHM sont en adéquation avec la base de données
Eléments à tester :	■ Requête MySQL depuis de code
Pré requis :	■ Base de données active ■ Code développé en C++

Id	Démarche	Comportement attendu	OK ?
1	Renvoyer les valeurs après comparaison avec la BDD	Valeurs identiques	OK

Fonctionnalité :

V Excellente
❑ Bonne
❑ Moyenne
❑ Faible

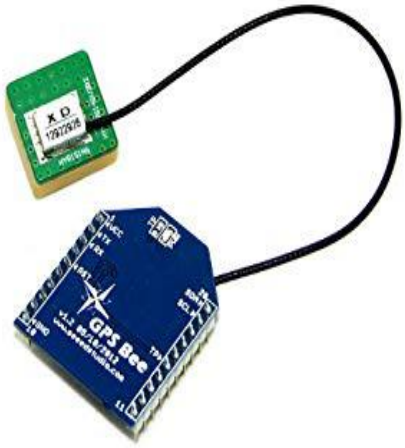
Localisation du parking et avertissement du propriétaire

Résumé

La partie « localisation du parking et avertissement du propriétaire » consiste dans un premier temps à localiser le parking dans lequel l'agent de sécurité effectue une vérification de plaque d'immatriculation et ensuite de proposer à l'agent de sécurité de contacter le propriétaire du véhicule en cas de besoin (Par exemple si le véhicule a une roue crevée).

Matériel

GPS bee kit module



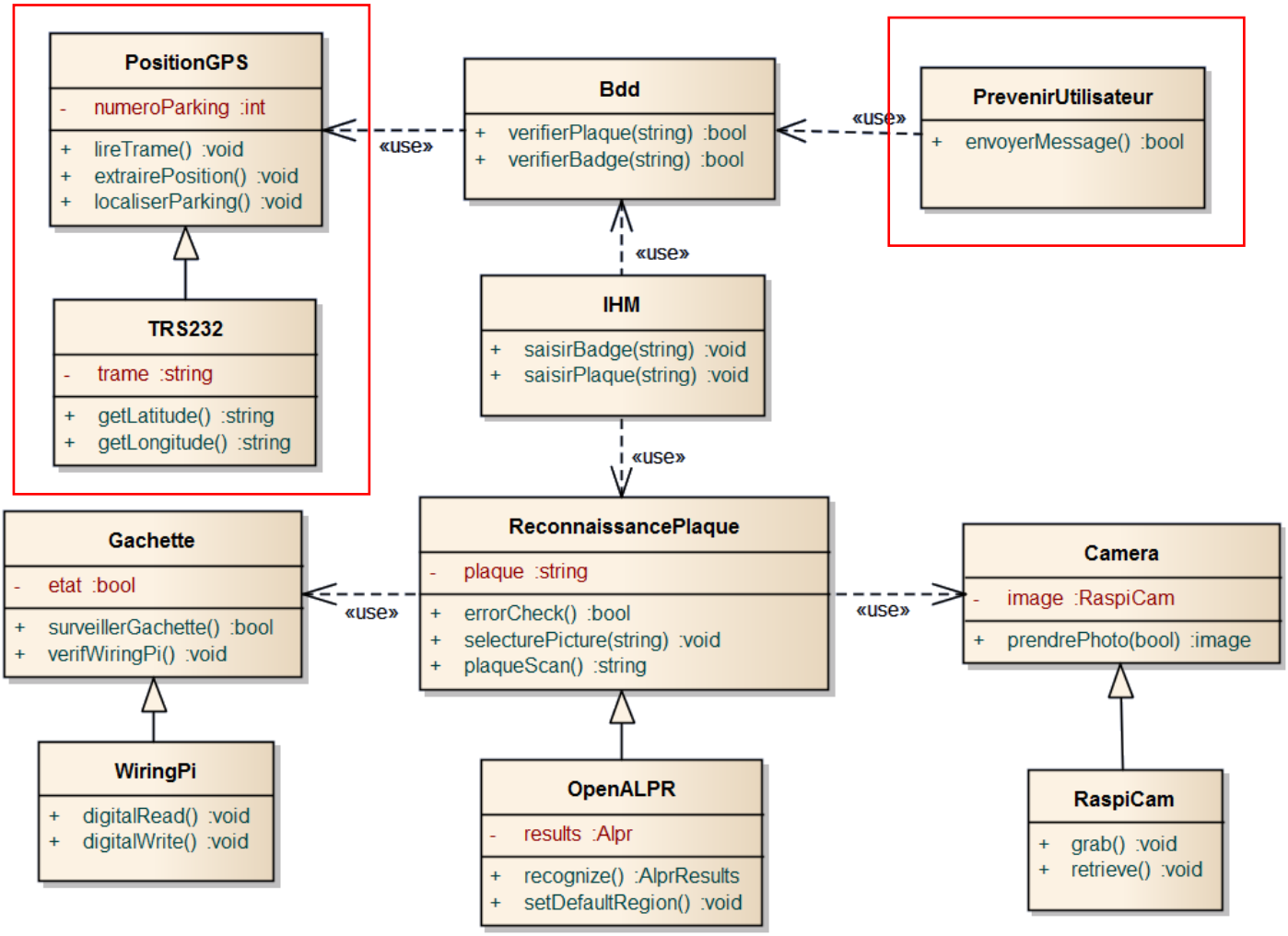
Le GPS bee kit est un module gps permettant de recevoir des trames NMEA (National Marine Electronics Association), et notamment les trames GPGGA qui nous permettent de connaître la position ainsi que la date

GSM M590E

Le GSM M590E est un module de communication téléphonique permettant via une carte SIM et des commandes AT d'effectuer des actions telles que déverrouiller la carte SIM installée dans le module, ou envoyer un message

Les deux modules utilisés dans ma partie de ce projet communiquent avec un Raspberry pi via liaison RS232.

Structure logicielle



La classe tRs232 est une classe permettant d'effectuer une liaison entre le Raspberry pi et les deux modules communiquant en RS232. La classe nous donne la possibilité de fixer par défaut un port de communication ainsi que sa vitesse de communication en Bauds, la longueur d'un caractère (exprimée en nombre de bits), la parité ainsi que le nombre de bits de stop.

```
class tRs232
    {
    private:
        termios Config;
        char MessageConfigurer[30];

        int Configurer(char* pPort,int Vitesse,int NbBits,int Parite,int NbStop);
    protected:
        int fd; //Descripteur de fichier
   //-----------------------------
    public:
        tRs232(void);
        tRs232(char* pPort,int Vitesse,int NbBits,int Parite,int NbStop);
        ~tRs232();
        void AfficherMessageConfigurer();

        // Envoyer une chaine de caractï¿½e
        int Envoyer(char* pChaine);

        // Recevoir Nb caractï¿½es
        int Recevoir(int Nb,char* pChaine);

        // Recevoir une chaï¿½e se terminant par un caractï¿½donne

        // avec ou sans time out
        int Recevoir(char Fin, char* pChaine, int Mode);
        int Recevoir(char Fin, char* pChaine,int Mode, int Attente);

        // Recevoir une sï¿½ie de caractï¿½es avec time out
        int RecevoirCaractere(char* Caractere, int Attente);

        // Recevoir un fichier
        int RecevoirFichier(char *NomFichier, int AttenteDebut);

   } ;
```

La classe nmea permet le parsing des trames reçus, c'est à dire que nous pouvons rechercher une donnée en fonction de sa position dans la trame. Par exemple nous recherchons la latitude et la longitude, on effectue donc un test dans un premier temps pour vérifier que la trame traitée est bien la bonne et ensuite on récupère la chaine de caractères présente en position 3 et 4 pour la latitude puis 5 et 6 pour la longitude.

```
nmea::nmea(){

}

string nmea::latitude() {

    if ( parse(1) == "$GPGGA" )
        return  parse(3) + parse(4);

    return "Erreur GPGGA";

}

string nmea::longitude() {

    if ( parse(1) == "$GPGGA" )
        return  parse(5) + parse(6);

    return "Erreur GPGGA";

}

string nmea::parse(int n) {

    if ( trame.length() <= 10) return "Erreur trame";

    int debut = -1;
    int fin = 0;
    string tmp = trame;

    while (n--) {
        debut += fin + 1;
        fin = tmp.find(",");
        tmp = tmp.substr(fin + 1, tmp.npos);
    }
    return trame.substr(debut   ,  fin );
}

string nmea::typeTrame(){

    return parse(1);

}

void nmea::setTrame(string trame) {
    this->trame = trame;
}
```

Pour le module GSM, il est nécessaire d'utiliser les commandes AT pour communiquer avec celui-ci et ainsi permettre de déverrouiller la carte sim (voir fig 1) et de communiquer le numéro de téléphone et le message au module GSM (voir fig 2).

Description	**This command is to check PIN status and input PIN code.**
Format	● AT+CPIN=<pin>[,<newpin>] ● AT+CPIN?
Syntax	<pin>: <newpin> is a string type value.

(fig 1)

Description	**This command is to send message from the module to network, the network will response reference value <mr> to the module after sending successfully.**
Format	● Command syntax(text mode): AT+CMGS=<da><CR><text><ctrl-Z/ESC> ● Command syntax (PDU mode): AT +CMGS=<length><CR><pdu><ctrl-Z/ESC>

Syntax	<da>:
	  Send message to target number in text mode
	<text>:
	  Message content in text mode
	<length>:
	  The length of message content digits in PDU mode.
	<mr>:
	  Storage location
	<CR> :
	  End character.
	<ctrl-Z> :
	  Indicate the end of the message input.
	<ESC> :
	  Give up to input message.

(fig 2)

Connectique et gestion matériel