INSA RENNES
4_SRC
2014

TABLEAU DE BORD AUDIO

Rapport de la réalisation du projet électronique

Nirenjanee Seenevassenpillay
Ndeye Aminata Ndiaye
Christophe Hallet
Elias Hami

Lipu Zhong
Saifedine Ben Malek
Mouad Abouelmejd
Ang Li

SOMMAIRE

I. Introduction : Rappel du principe

Dans le cadre de notre projet électronique de 4ème année, nous allons réaliser un prototype de tableau de bord audio d'une moto. Ce prototype a pour but d'informer le conducteur de sa vitesse et éventuellement les témoins d'alertes. Voici le rappel du cahier des charges sous forme de schéma :

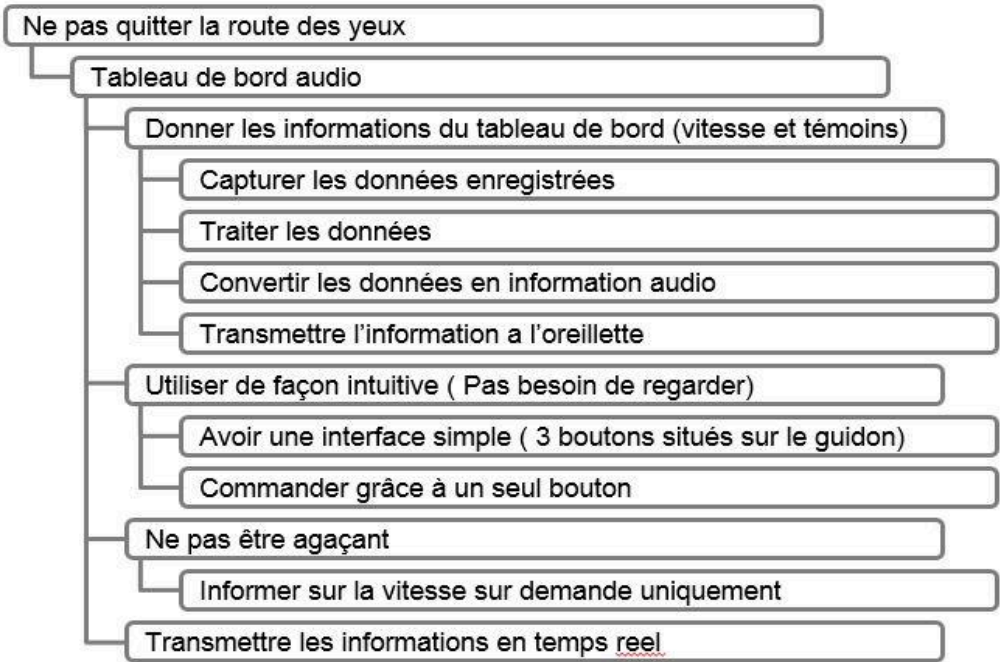


Figure I.1 : Cahier des charges

Pour réaliser notre projet, nous avons distingué trois grandes parties :

- ❖ acquisition des données par analyse d'image;
- ❖ conversion audio des informations;
- ❖ émission bluetooth.

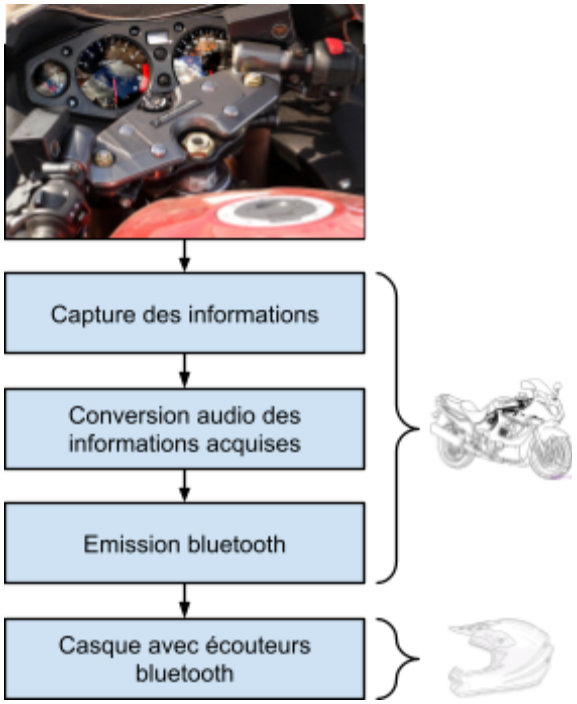


Figure I.2 : Blocs complémentaires constituant le projet

Pour rappel, voici le schéma global du prototype constitué de la partie analyse d'image (carte raspberry pi, caméra raspberry pi, bouton poussoir), la partie conversion audio (PIC16F690, SOMO 14D) et de la partie émission bluetooth(émetteur bluetooth).

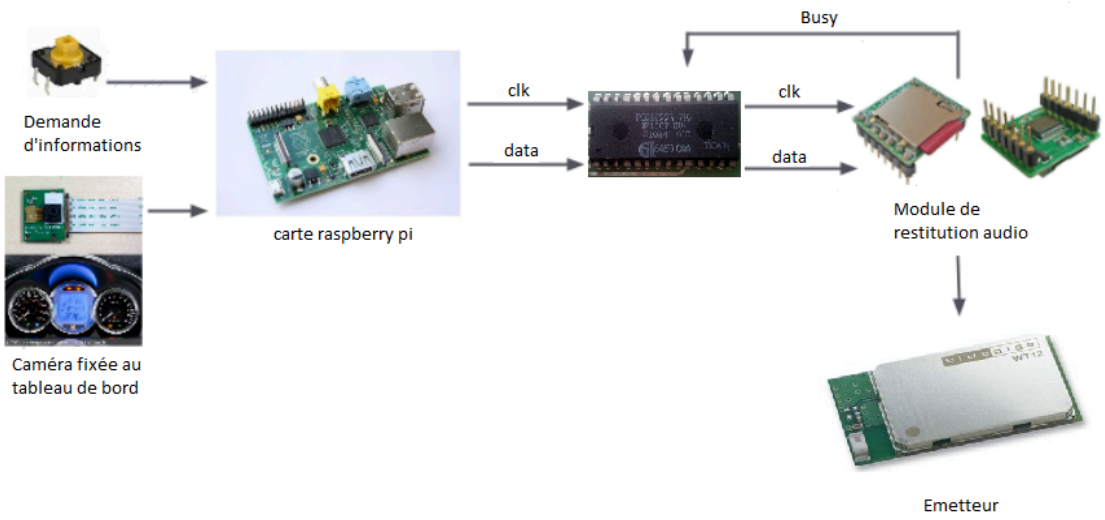


Figure I.3 : Schéma global de la conception du prototype

II. Module de traitement d'images : Capture des informations

II.1. Présentation du module

L'acquisition des données telles que la vitesse et les indicateurs (niveau d'huile, batterie, frein,…) du tableau de bord a été réalisée par un traitement et une analyse d'images.

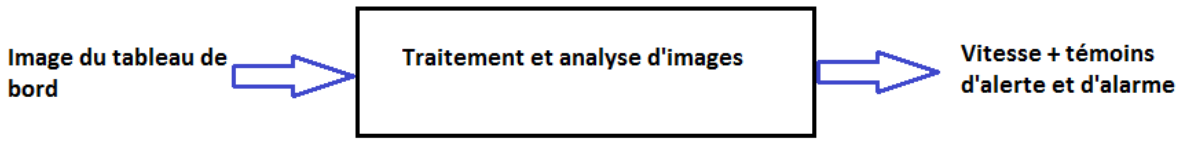


Figure II.1.1 : Présentation du module

En effet une caméra placée au niveau du tableau de bord prend des photos de façon régulière et/ou à la demande de l'utilisateur grâce à un bouton poussoir; ensuite les photos sont envoyées à un microcontrôleur pour y subir un traitement et une analyse d'image afin d'extraire les informations du tableau de bord. Le choix du microcontrôleur s'est porté sur la carte raspberry pi et la programmation a été réalisée en langage C++ couplé de la bibliothèque OpenCV, spécialisée dans le traitement d'images en temps réel.

Dans ce module, il a été question, non seulement, d'écrire les programmes de traitement et d'analyse d'images et de les implémenter sur la carte raspberry pi afin qu'ils y soient exécutés en boucle mais aussi d'écrire un script sous raspbian qui gère le fonctionnement du module.

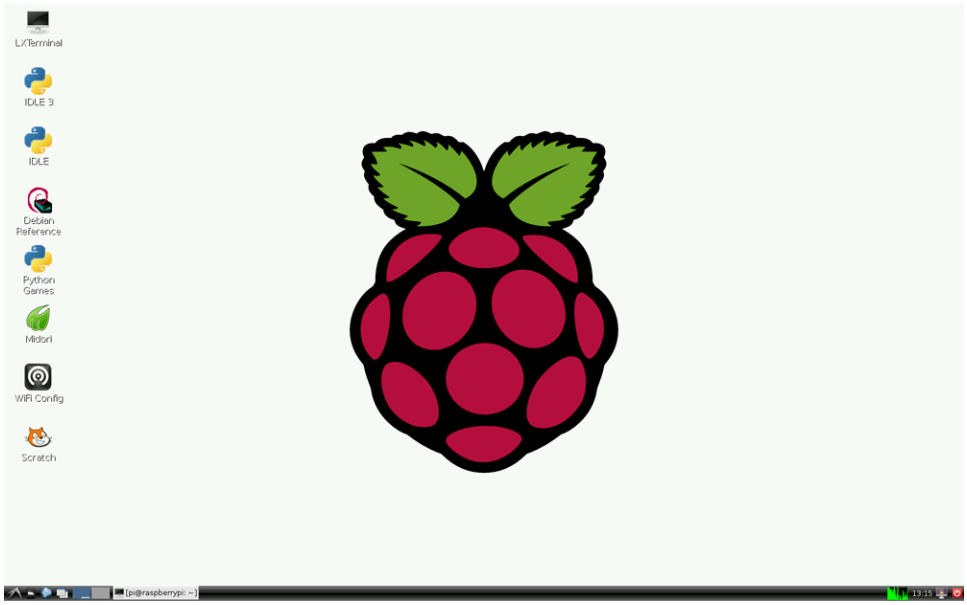
Pour la réalisation, ce module a été divisé en 3 phases à savoir:

- Installation des outils (logiciels & matériels) nécessaires à la réalisation du module
- Acquisition des informations fournies par le tableau de bord (traitement et analyse d'images)
- Implémentation des programmes au niveau de la carte raspberry pi et création d'un script pour la gestion du microcontroleur

II.2. Outils nécessaires au module (caméra et carte raspberry)

- Installation du système d'exploitation de la carte raspberry pi

Pour utiliser la carte Raspberry Pi, il faut donc d'abord installer le système d'exploitation. On a choisi Raspbian, qui est un système d'exploitation libre basé sur Debian optimisée pour le matériel Raspberry Pi. Celui-ci demande une carte SD pour le stockage. Après la configuration du système (tutoriel cf lien en bibiographie), on obtient donc l'interface de l'OS Raspbian :



- Installation de la bibliothèque opencv

Après avoir installé le système sur la carte, il faut maintenant installer l'environnement pour nos codes. La bibliothèque opencv est nécessaire. En suivant les instructions sur le site d'opencv et un tutoriel (cf liens en bibliographie) , nous sommes parvenus à bien installer la bibliothèque.

//image test à mettre d'ici mercredi

- Module Caméra Raspberry pi

Comme la caméra RaspiCam est compatible avec la carte Raspberry pi, on peut l'activer directement dans la configuration système :

Pour l'utiliser, la commande "***raspistill -o image.jpg***" est lancée depuis un terminal pour prendre une photo. Cependant, pour répondre à notre cahier de charges, il il faut que des images soient prises régulièrement. On utilise la commande avec l'option "-t" qui permet de définir l'intervalle de temps entre les prises et donc pour qu'une image soit prise toutes les 15 secondes:

"rapistill -o image.jpg -t 15000"

II.3. Acquisition de la vitesse

Afin d'acquérir l'information de la vitesse, nous avons réalisé un FAST (Function Analysis System Technique) nous donnant ainsi une ligne de travail à suivre; voici ce que nous obtenons :

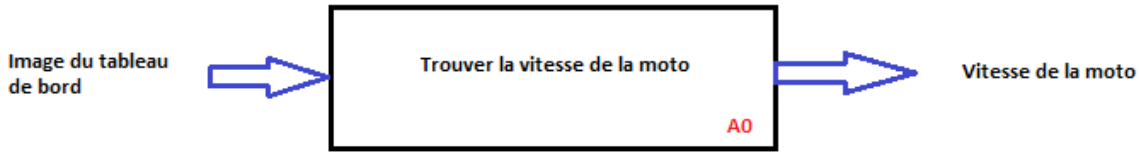


Image II.3.1 : Etape A0

L'étape principale A0 se décompose en trois sous-étapes complémentaires :

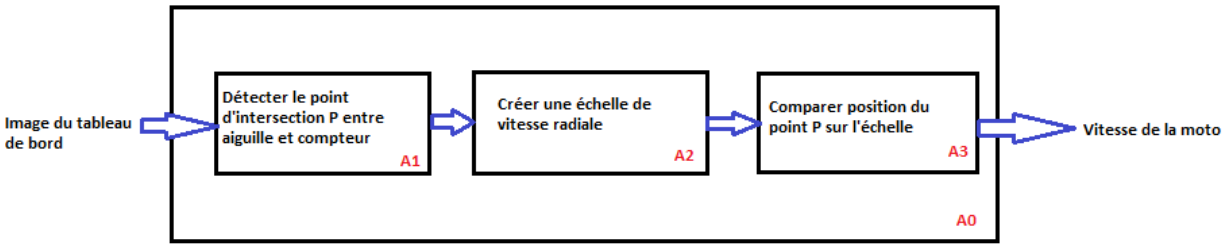


Image II.3.2 : Trois étapes complémentaires: A1 - A2 - A3

Dans chaque sous-étape complémentaires se distinguent d'autres étapes pour résoudre le problème, voici les sous-étapes les plus importantes du FAST détaillé :

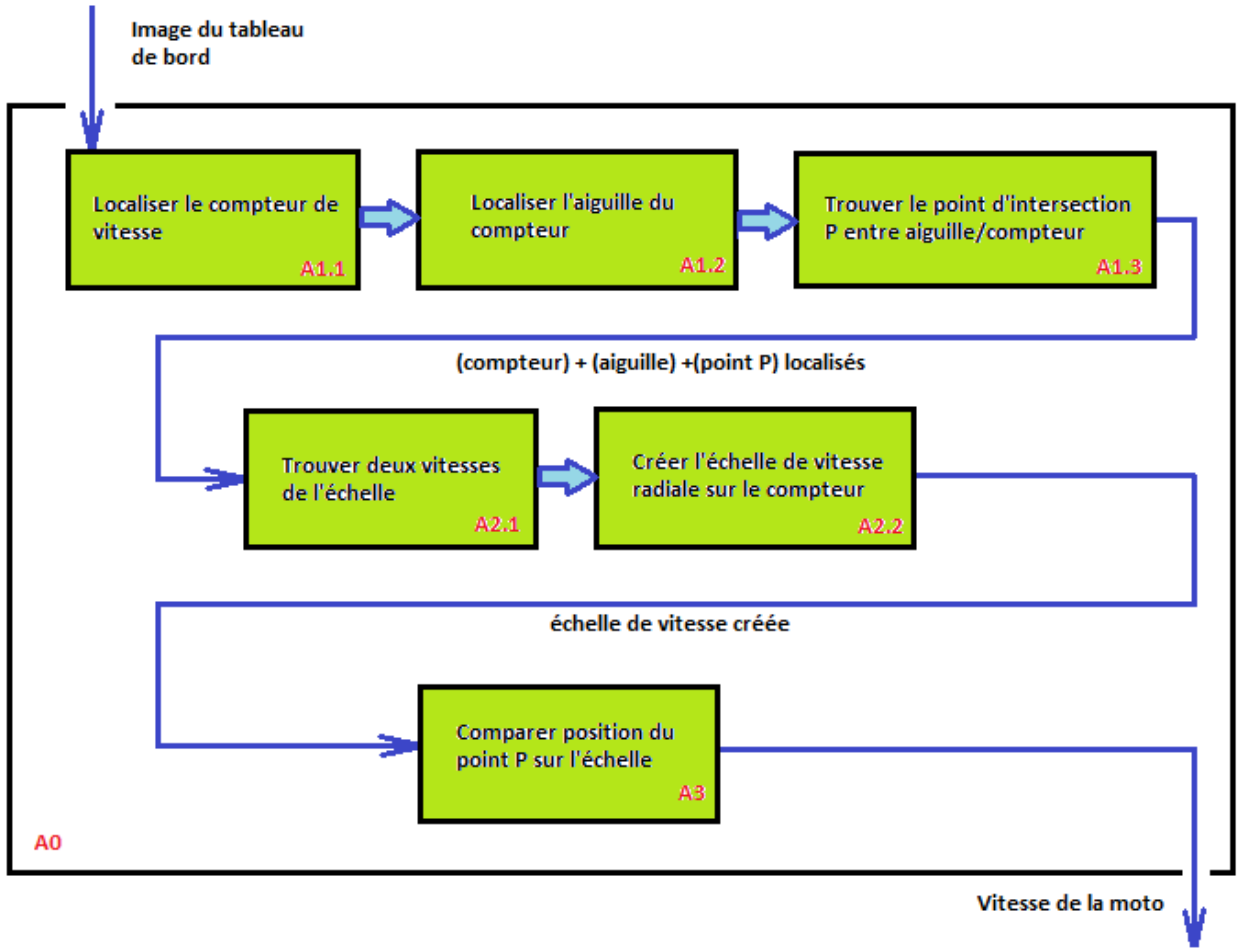


Image II.3.3 : Schéma du FAST détaillé

II.3.1. Localisation du compteur vitesse dans l'image (étape A1.1)

II.3.1.1) Détection des deux compteurs : vitesse et accélération

L'objectif de cette étape est de localiser les deux compteurs de vitesse et d'accélération sur le tableau de bord. Pour cela, nous avons appliquer une transformée de Hough pour les cercles de la bibliothèque opencv : houhgcircle. Cette fonction permet de localiser des cercles en ayant 2 paramètres intéressants en fixant un rayon minimal et maximal que l'on souhaite trouver. Notre étude s'est réalisée sur l'image du tableau de bord en **Figure 1** ci contre:

Figure II.3.1 : image du tableau de bord

Sur la **Figure 2**, nous pouvons voir que le code a bien réalisé ce que nous cherchons : deux ensembles de cercles bleus avec leurs centres respectifs en vert représentant l'un le compteur de vitesse, l'autre le compteur d'accélération.

Figure II.3.2 : Localisation des deux compteurs de vitesse (gauche) et d'accélération (droite)

II.3.1.2) Critère de sélection

Une fois les deux compteurs localisés dans l'image, il faut trouver un critère de sélection, c'est à dire trouver une ou plusieurs caractéristiques qui distinguent chacun des compteurs.

L'utilisation de l'outil OCR (Optical Character Recognition) est possible pour déterminer les nombres par exemple mais trop coûteuse en ressources.

En se basant donc sur l'idée des chiffres, on remarque que les chiffres créent des rondeurs et donc des portions de cercles. Par conséquent, en utilisant une autre transformée de Hough pour les cercles nous obtenons la **Figure 3** :

Figure II.3.3 : Cercles en bleu de centre vert symbolisant majoritairement les chiffres

Ainsi, en appliquant le critère du nombre le plus élevé de cercles en bleu se situant dans l'un des deux compteurs de vitesse ou d'accélération, nous pouvons donc choisir le compteur de vitesse puisqu'il possède d'avantage de minis cercles.

Figure II.3.4 : Repérage du compteur de vitesse dans l'image du tableau de bord

II.3.1.3) Redimensionnement de l'image

Nous avons réussi jusqu'à présent à localiser précisément la position du compteur de vitesse dans l'image du tableau de bord. Par la suite, dans l'optique d'améliorer la rapidité du programme, il est judicieux de redimensionner l'image du tableau de bord au niveau du compteur de vitesse et ainsi d'éviter tout calcul pouvant s'effectuer en dehors du compteur.

Figure II.3.5 : Image du tableau de bord redimensionnée au niveau du compteur de vitesse

II.3.2. Déterminer la position du point image vitesse

II.3.2.1) Détection de la position de l'aiguille (étape A1.2)

L'étape suivante est la détection de l'aiguille. Pour se faire, il est donc indispensable de localiser des droites dans une image. Pour se faire, nous avons utilisé une autre transformée de Hough mais pour les droites cette fois ci la fonction houghlines possédant un paramètre qui détecte des droites à partir d'une certaine longueur de droite. Le résultat est le suivant sur la **Figure 6**:

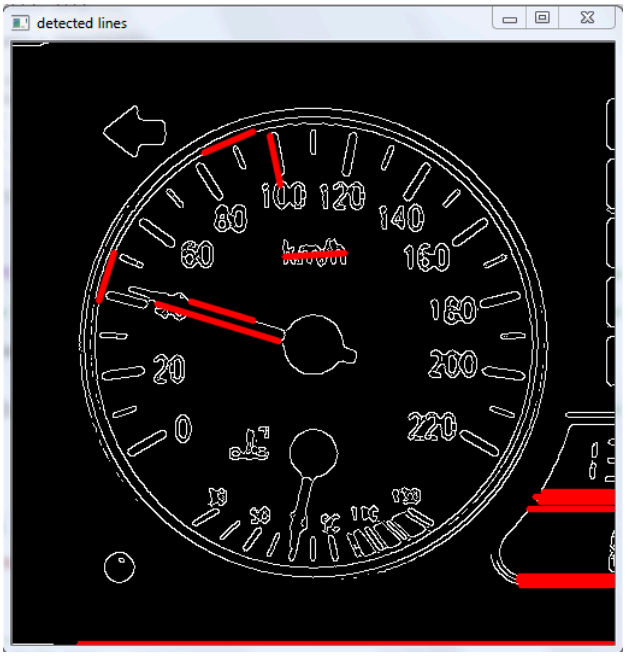


Figure II.3.6 : Détection de droites en rouge

A partir de là, nous devons trouver un autre critère de sélection ne choisissant que les droites appartenant à celle de l'aiguille. Pour se faire, nous avons créée une zone rectangulaire centrée sur le centre du compteur, mise dans une boucle "while" jusqu'à temps de trouver au moins un point de droite parmi toutes les droites trouvées (une droite est définie par deux points, ses extrémités).

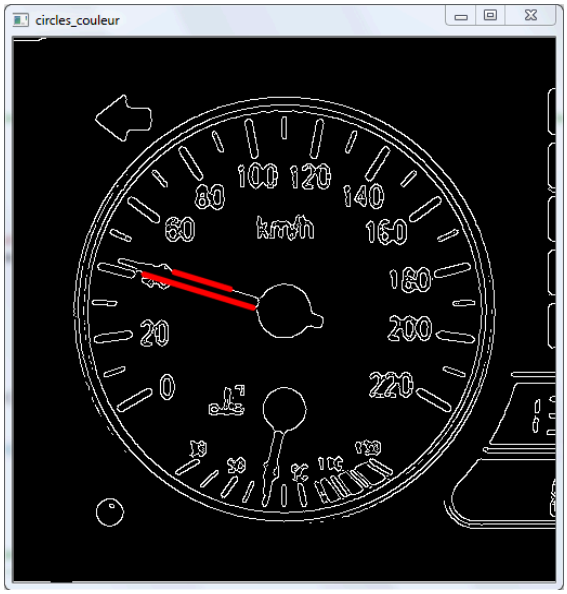


Figure II.3.7 : Localisation de l'aiguille

II.3.2.2) Obtention de la projection radiale de la pointe de l'aiguille sur le compteur (étape A1.3)

A ce niveau là, nous avons le cercle moyen (compteur de vitesse) et une ou plusieurs droites symbolisant l'aiguille. A partir de là, grâce à quelques opérations géométriques telles les allongements de droites pour trouver la pointe de l'aiguille, et opérations d'intersections entre droites et cercle, nous avons réussi le projeté radial de la pointe de l'aiguille sur le compteur :

Figure II.3.8 : Projeté radial de la pointe de l'aiguille sur le compteur

II.3.3. Etablissement d'une échelle angulaire de vitesses

II.3.3.1) Localisation de deux valeurs de vitesses (étape A2.1)

L'objectif dans cette partie est de repérer deux valeurs de vitesse dans le compteur pour ainsi créer une échelle angulaire de vitesse à partir de ces deux points là par extrapolation. En effet, il est possible d'utiliser l'outil OCR (Optical Character Recognition) mais comme mentionné ci dessus, cet outil demande des ressources et peut être difficile à l'insérer dans un système embarqué qui est dans notre cas la carte raspberry pi.

Pour cela, nous avons réfléchi à une autre méthode en utilisant la fonction hougcircle. Il s'agit de repérer les chiffres 8 dans l'image pour trouver la valeur de la vitesse 80 et 180.
Ceci est une méthode basique pouvant largement être améliorée, mais voici le résultat :

Figure II.3.9 : Deux Chiffres 8 détectés

Ainsi, à partir de ces chiffres 8 et grâce nombreuses là aussi à de nombreuses opérations géométriques, nous avons pu projeter radialement les vitesses 80 et 180 sur le compteur :

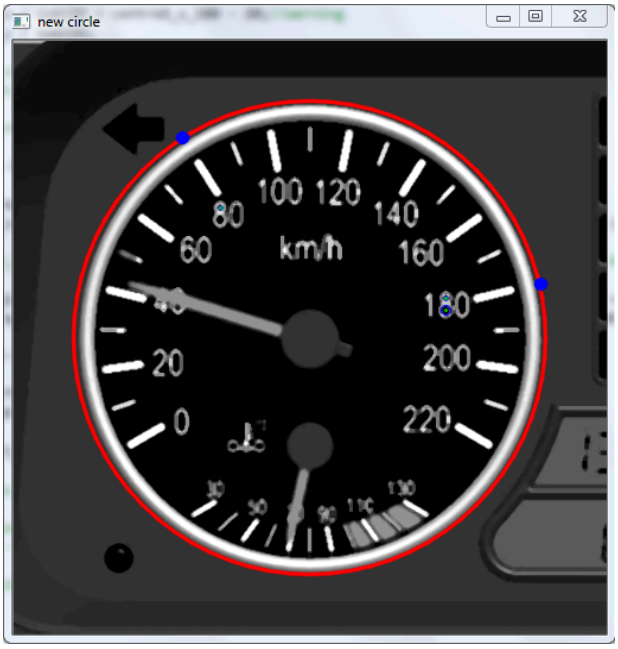


Figure II.3.10 : Projetés de façon radiale des nombres 80 et 180 sur le compteur de vitesse

II.3.3.2) Création de l'échelle angulaire de vitesses (étape A2.2)

A partir de là, comme nous avons réussi à trouver deux points sur le cercle moyen associés à deux valeurs de vitesse 80 et 180, il est désormais facile de créer une échelle angulaire de vitesse par le biais d'opérations géométriques.

Nous avons limité le pas de l'échelle à 1 km/h, voilà ce que nous obtenons :

Figure II.3.11 : Echelle angulaire de vitesses (chaque cercle en rouge représente une valeur de vitesse)

II.3.4. Obtention de la vitesse (étape A3)

Nous récoltons ici les fruits de notre travail. Nous avons à notre disposition une échelle de points associés à des vitesses (partie 3.2) créée sur le cercle moyen, ainsi que le projeté de la pointe de l'aiguille sur le cercle moyen aussi (partie 2.2). Il suffit donc de comparer la position du point projeté et de le comparer aux différents points de l'échelle et d'en déduire la vitesse associée à ce point projeté :

Figure II.3.12 : Résultat final

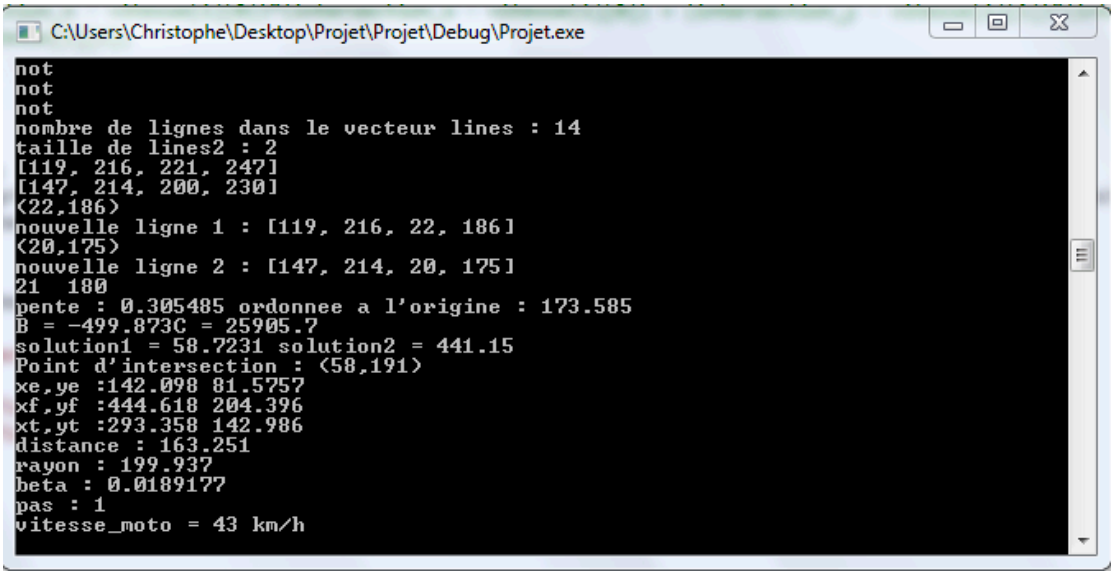


Figure II.3.13 : Obtention de la vitesse de la moto

II.3.5. Amélioration du programme

Pour l'instant, nous obtenons la vitesse de la moto à partir d'une simple image. Cependant, l'image n'est pas prise par la caméra raspberry pi, ce n'est donc pas une image réelle. En réalité, lorsqu'on acquiert une image par la caméra et que nous lançons le programme, nous nous heurtons à de nombreux problèmes :

- taille de l'image : modifiant certains paramètres qui étaient fixes
- rajouter une partie traitement de l'image pour la rendre plus nette
- la qualité de l'image : un peu floue car la caméra ne peut pas focaliser
- aiguille non localisée : niveau de seuil de gris non adapté
- les 8 non détectés : difficulté à trouver les cercles intérieurs des 8 car l'image n'est pas nette.

Figure II.3.14 : Image réelle du tableau de bord

Le problème le plus compliqué à résoudre est de trouver deux points de l'échelle. Une autre méthode a été trouvée rendant le programme plus robuste et plus performant que l'ancien programme avec les 8.

II.3.5.1. Trouver deux points de l'échelle

Pour créer l'échelle, il nous faut toujours au moins deux points. Le point le plus intéressant à trouver est le "0" car il est au bout de l'échelle et il ne risque pas d'être recouvert par l'aiguille. La **Figure 15** montre deux traits du compteur en rouge pointant vers le zéro et le vingt, il suffit donc pour trouver le zéro de choisir le trait le plus bas :

Figure II.3.15 : La vitesse 0 trouvé

Ensuite, la deuxième étape consiste à trouver le pas entre les chiffres, c'est à dire la distance qui sépare deux traits consécutifs (associés aux chiffres). Il faut tout d'abord trouver une grande partie des traits qui nous intéresse (qui se dirigent tous vers les chiffres du compteur).

Figure II.3.16 : Traits du compteur trouvés

Et après, à partir de ces traits stockés dans un tableau, pour trouver le pas, il suffit de trouver la distance entre deux traits du tableau dans la couronne contenant les traits et de ne retenir que la distance qui revient le plus souvent.

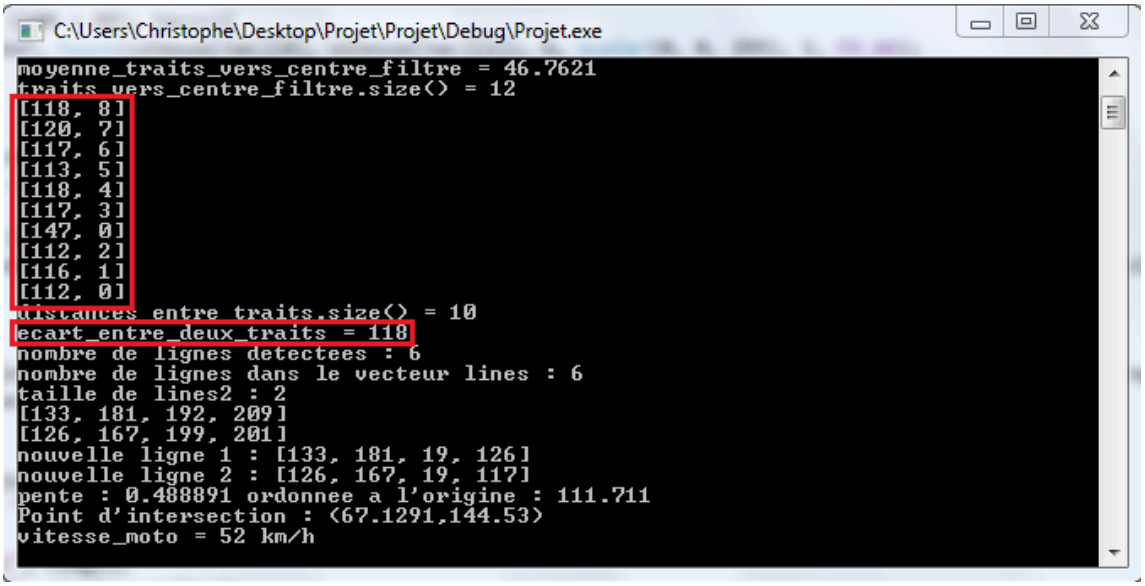


Figure II.3.17 : [Valeur Va de distances entre deux traits du tableau, valeur d'itération qui augmente de + 1 si Va est commune (à +/- quelques %) aux autres valeurs du tableau]

II.3.5.2. Résultats sur l'image réelle

Voilà ce que nous obtenons sur l'image réelle prise par la caméra raspberry pi, photographiant la vidéo visionnée sur une tablette :

Figure II.3.18 : Compteur localisé sur l'image réelle

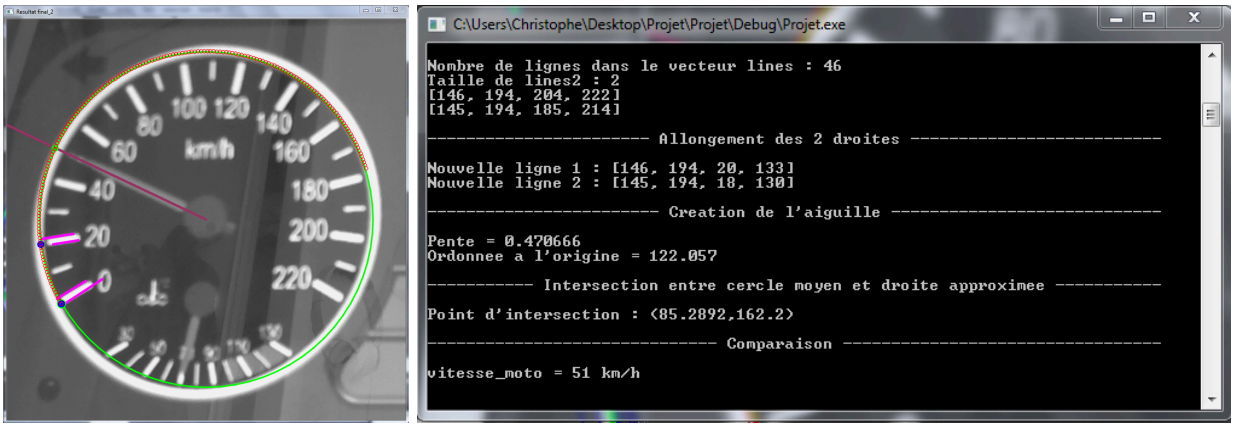


Figure II.3.19 : Résultat final => vitesse de 51 km/h

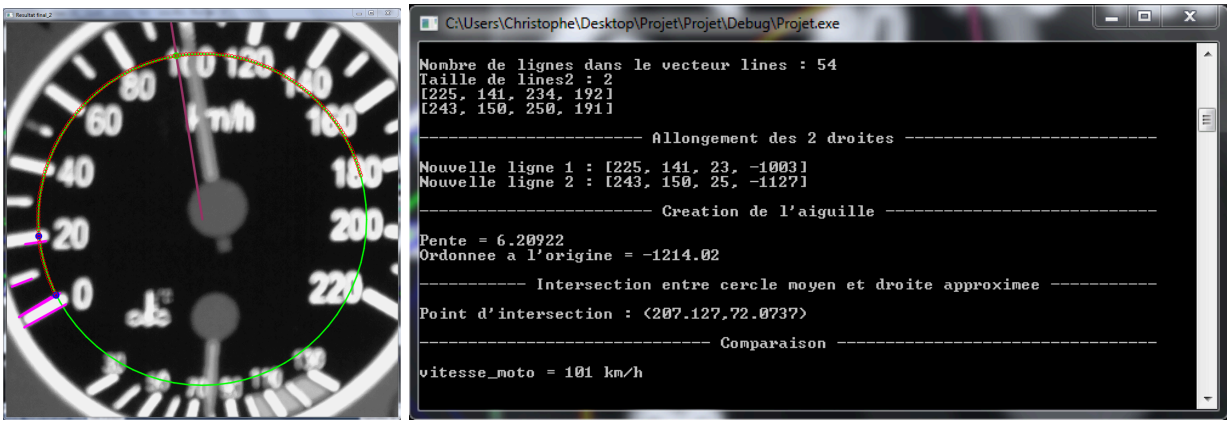


Figure II.3.20 : Résultat final => vitesse de 101 km/h

II.4. Acquisition des informations fournies par les témoins d'alerte et d'alarme

Comme lors de l'acquisition de la vitesse, un FAST a été également réalisé, toujours dans le but de nous donner une ligne de travail à suivre; l'étape principale étant ici d'extraire les informations fournies par les témoins d'alerte et d'alarme.

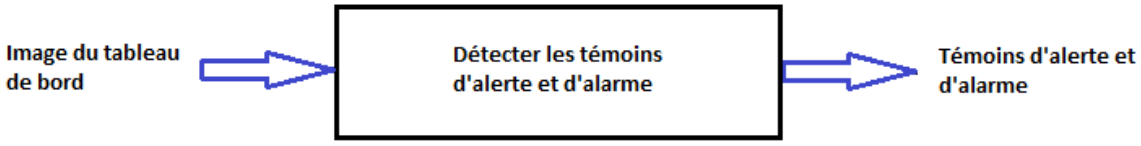


Figure II.4.1 : Principale étape

L'étape principale se décompose en sous-étapes:

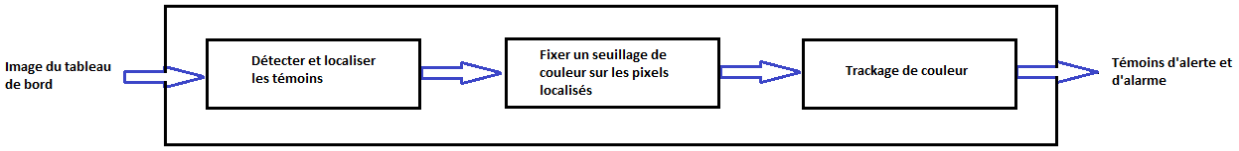


Figure II.4.2 : Trois sous-étapes pour l'acquisition des témoins

II.4.1. Détection et localisation des témoins par la méthode “Matching Template”

La méthode du “matching template” est une technique de recherche de zones d'une image qui correspond à une image modèle. C’est une méthode efficace et rapide pour localiser un objet unique. Pour ainsi identifier la zone de correspondance, il a fallu comparer l'image modèle sur l'image source en la faisant glisser. En glissant, nous entendons déplacer le modèle d'un pixel à la fois (de gauche à droite, de haut en bas) sur toute l’image. À chaque endroit, une métrique est calculée de sorte qu'il représente la bonne correspondance à cet endroit ou pas. Toutes les valeurs de métriques sont stockées dans un vecteur, et pour trouver la bonne correspondance, il faut juste localiser la valeur la plus élevée des métriques.

Test de notre programme avec une image telle que:

Figure II.4.3 : Image test

Appliqué à chaque témoin, voilà ce que nous obtenons pour 4, entre autres, la bonne correspondance étant représentée par un rectangle vert.

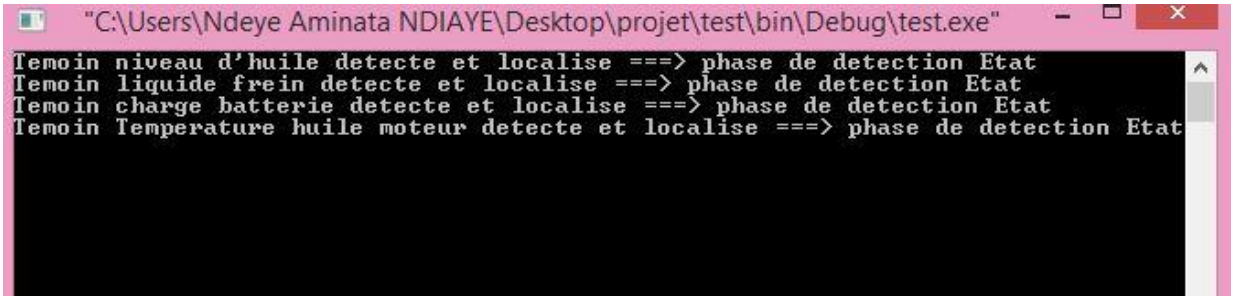


Figure II.4.4 : Vu sur le terminal

Figure II.4.5 : Détection et localisation du témoin de niveau d'huile

Figure II.4.6 : Détection et localisation du témoin de la température de l'huile moteur

Figure II.4.7 : Détection et localisation du témoin de batterie

Figure II.4.8 : Détection et localisation du témoin du liquide frein

Toutefois il faut noter que cette technique de localisation ne renseigne pas sur l'état des témoins, car elle ne tient pas compte des couleurs. Il a fallu donc récupérer les pixels localisés et y faire une analyse d'images.

II.4.2. Détection de l'état des témoins

Une fois les témoins localisés, il est maintenant question de détecter l'état des témoins qui est soit allumé ou éteint, et dans quel cas il faudrait faire une alerte au motocycliste. La détection de l'état des témoins regroupe les 2 dernières sous-étapes mentionnées dans le FAST.

II.4.2.1) Application d'un seuillage de couleur

Pour la plupart des tableaux de bord, les témoins d'alerte et d'alarme se caractérisent soit par une lumière verte, soit bleue ou rouge. Il n'est pas évident de faire un trackage sur ces couleurs parce qu'il existe des tonnes de nuance pour ces couleurs. C'est pour celà nous fixons un seuillage direct de couleurs sur les composantes RVB (ou RGB en anglais) obtenues après localisation des témoins.

Le seuillage consiste à séparer les objets en les considérant comme des formes de couleurs différentes. Ce terme désigne plus précisément la définition d'un seuil au-dessus ou en dessous duquel on va garder certaines valeurs de niveaux de gris. L'étape de seuillage nous fournira des images binaires dont les sections prendront le plus haut niveau de gris soit 1 (blanc) alors que les autres sections prendront le plus bas niveau de gris soit 0 (noir). Le trackage se fera donc sur le blanc par la suite.

Les figures ci-dessous montrent le résultat obtenu après seuillage:

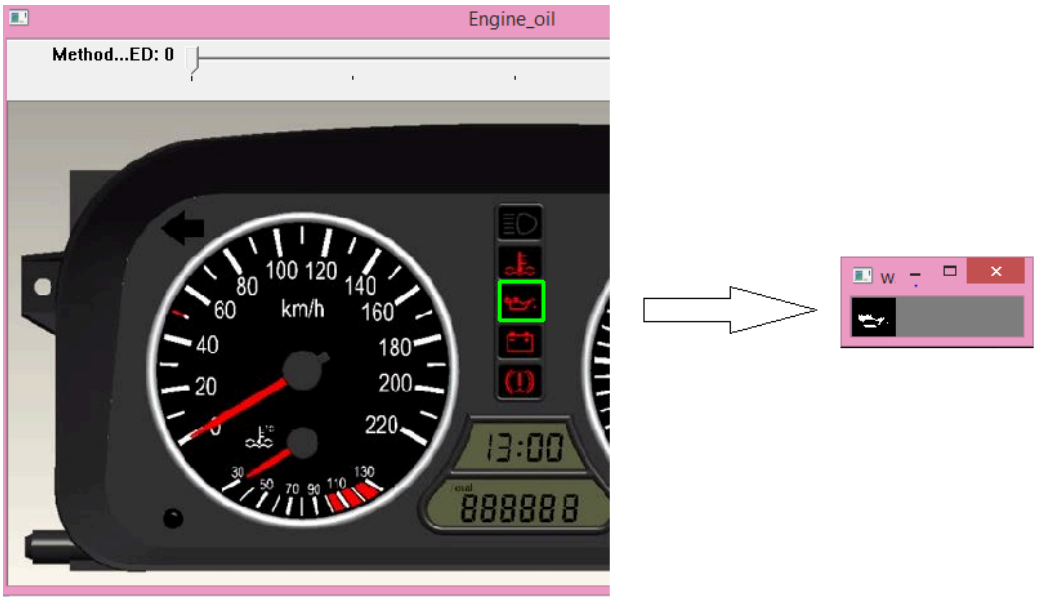


Figure II.4.9 : Seuillage appliqué sur une image où le témoin est allumé

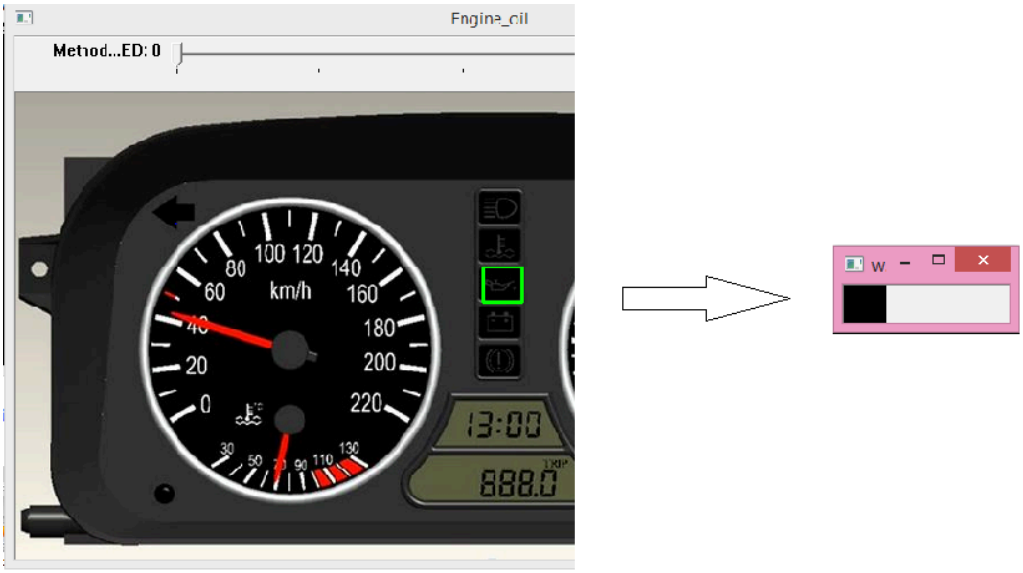


Figure II.4.10 : Seuillage appliqué sur une image où le témoin est allumé

Sur ces deux figures, on voit clairement que nous arrivons à distinguer un témoin allumé et un témoin éteint. Il suffira juste de faire un test de blanc, la présence de blanc ou pas, pour enfin détecter l'état d'un témoin.
NB : cette analyse est appliquée à tous les témoins

II.4.2.2) Trackage de couleur

Pour le trackage de blanc, nous parcourons l'image obtenue après seuillage et comparons chaque pixel au blanc dont les corrdonnées RVB sont (R=255, V=255, B=255). Dès que le blanc est détecté, cela veut dire que le témoin est allumé et donc, une alerte peut se faire.

Voici les résultats obtenus après trackage de blanc:

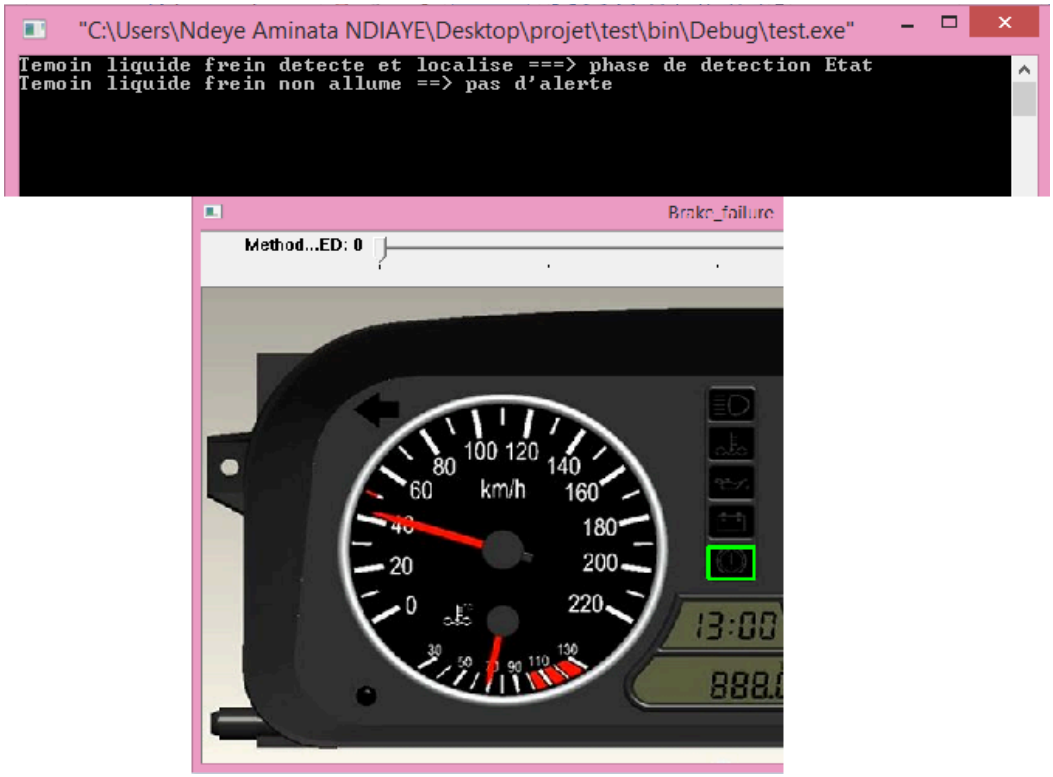


Figure II.4.11 : Cas où le témoin est éteint

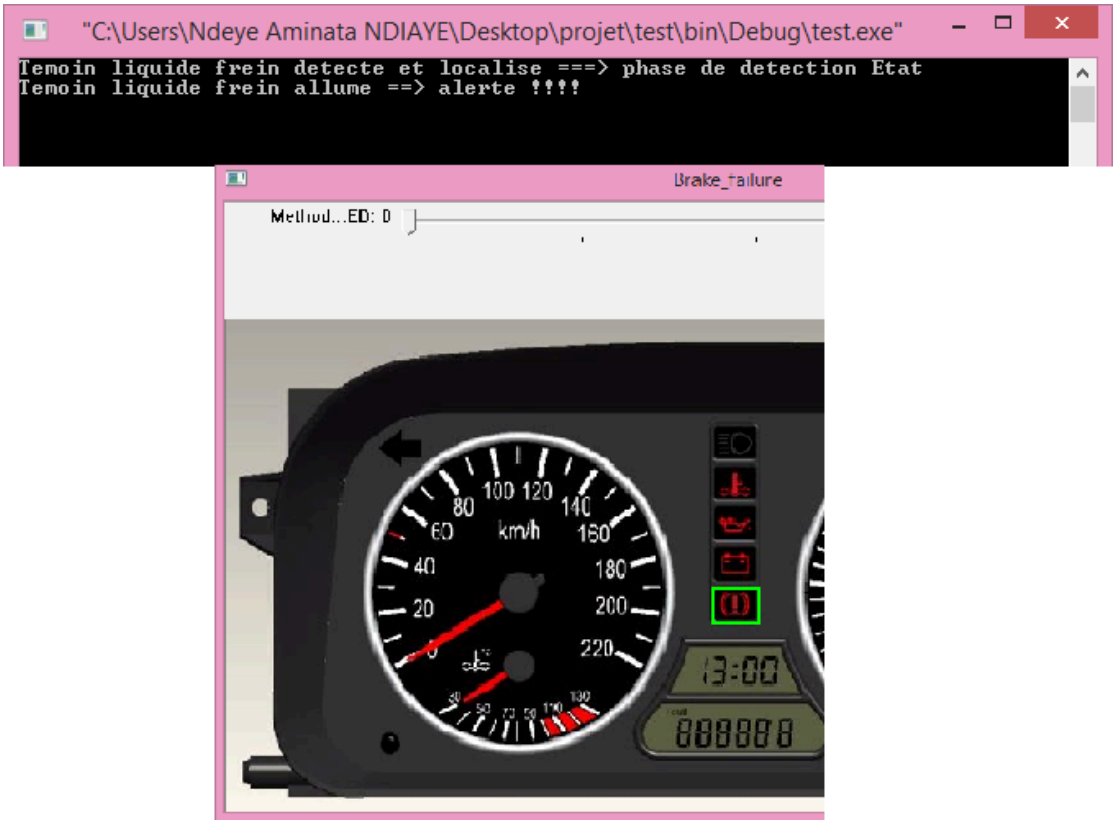


Figure II.4.12 : Cas où le témoin est allumé

(NB : cette analyse est appliquée également à tous les témoins)

II.4.3. Application à une image réelle

Comme pour l'acquisition de la vitesse, l'utilisation d'une image réelle engendre de nouveaux problèmes dont les principaux ici sont la taille et la netteté. Pour y remédier, avant toute analyse, on effectue un redimensionnement de l'image et ensuite on applique un filtre gaussien qui est utilisé comme constituant du masque flou qui améliore la netteté apparente des photographies numériques.

Voilà ce que nous obtenons sur l'image réelle prise par la caméra raspberry pi, photographiant la vidéo visionnée sur une tablette:

Figure II.4.13 : image rèelle

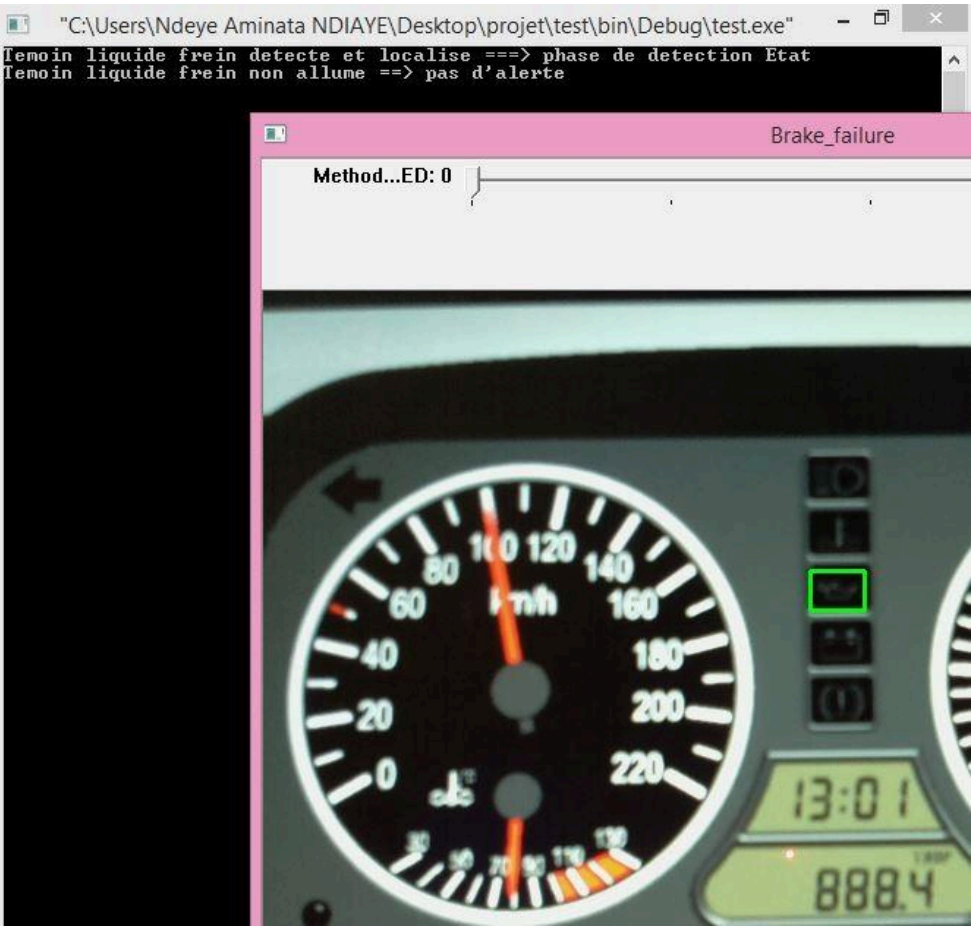


Figure II.4.14 : Acquisition témoin sur une image réelle

II.5. Résultats finaux (implémentation des programmes sur la carte, commande du module) (Lipu, Christophe et Aminata)

Le programme ainsi réalisé sous nos machines, a ensuite été implémenté sur la carte raspberry pi. Voici les conditions de notre expérimentation :

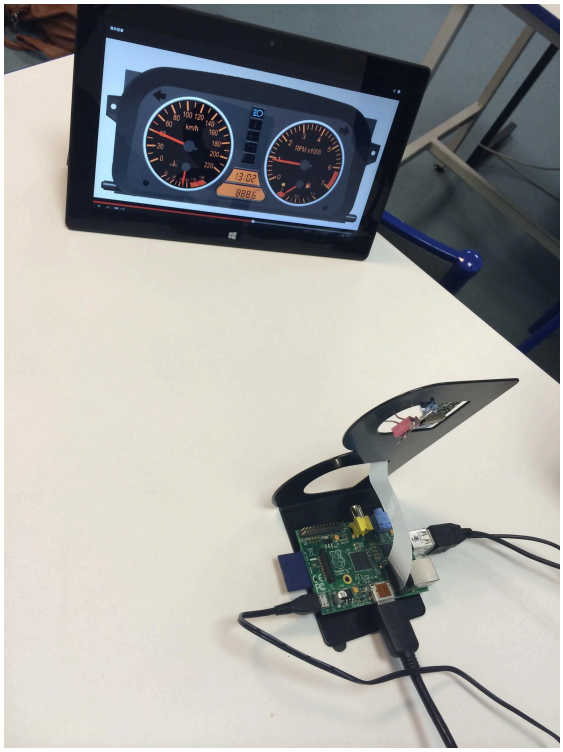


Figure II.5.1 : Photo contenant la carte raspberry pi et la tablette simulant le tableau de bord d'une moto

Ainsi, pour le test, nous avons lancé le programme sur la carte avec une image prise sur la tablette affichant la vidéo d'un compteur.

Par ailleurs, nous avons créé un script sous raspbian permettant de lancer en boucle deux commandes : l'acquisition d'une image et notre programme.

Voici le contenu du script :

```
while [1];
do
    raspistill -o  image_reelle2.jpeg;
    ./testons;
    sleep 15s;
    pkill testons;
done;
```

Figure II.5.2 : Script sous raspbian

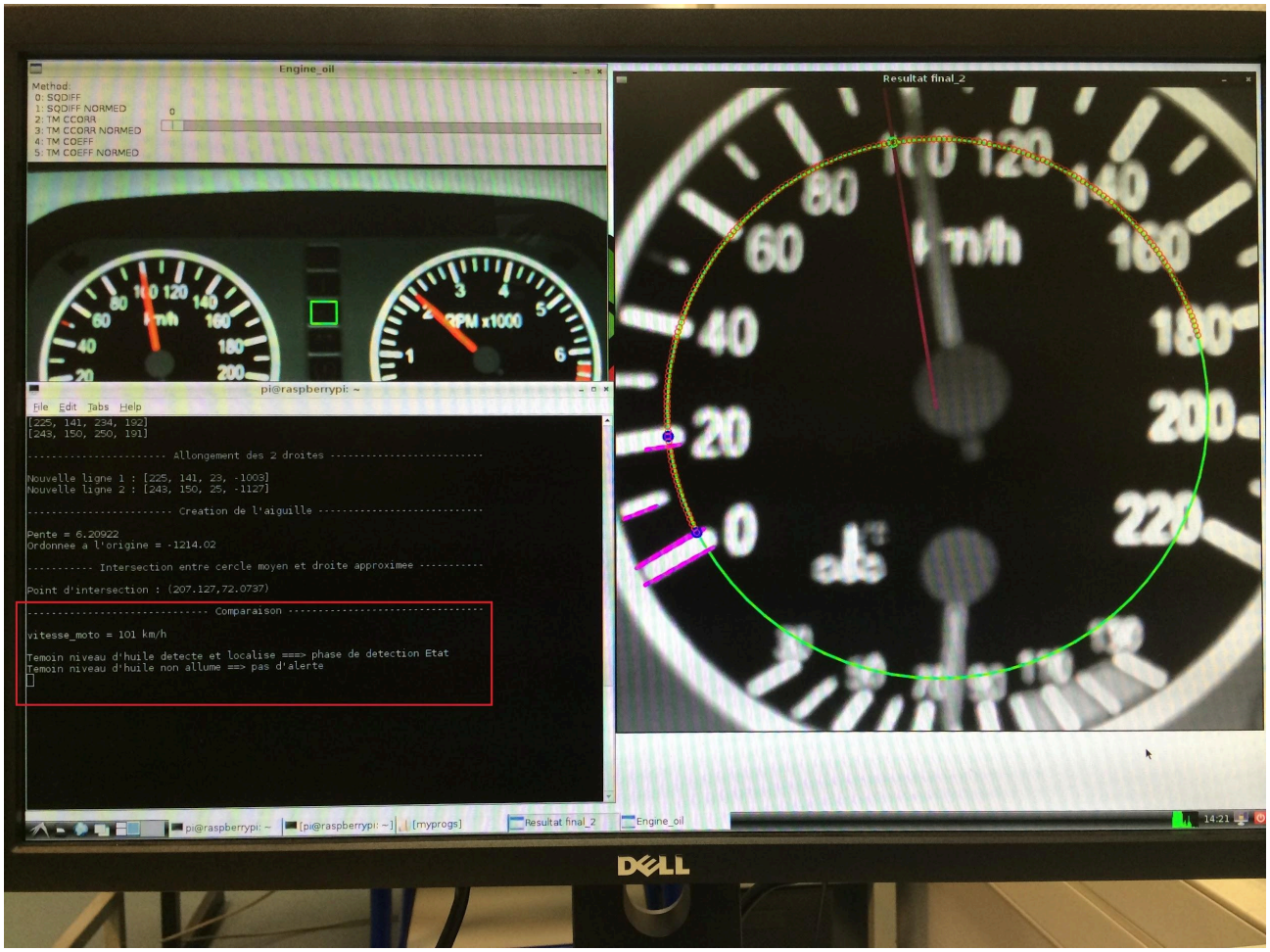


Figure II.5.2 : Photo prise sous linux montrant le résultat du programme

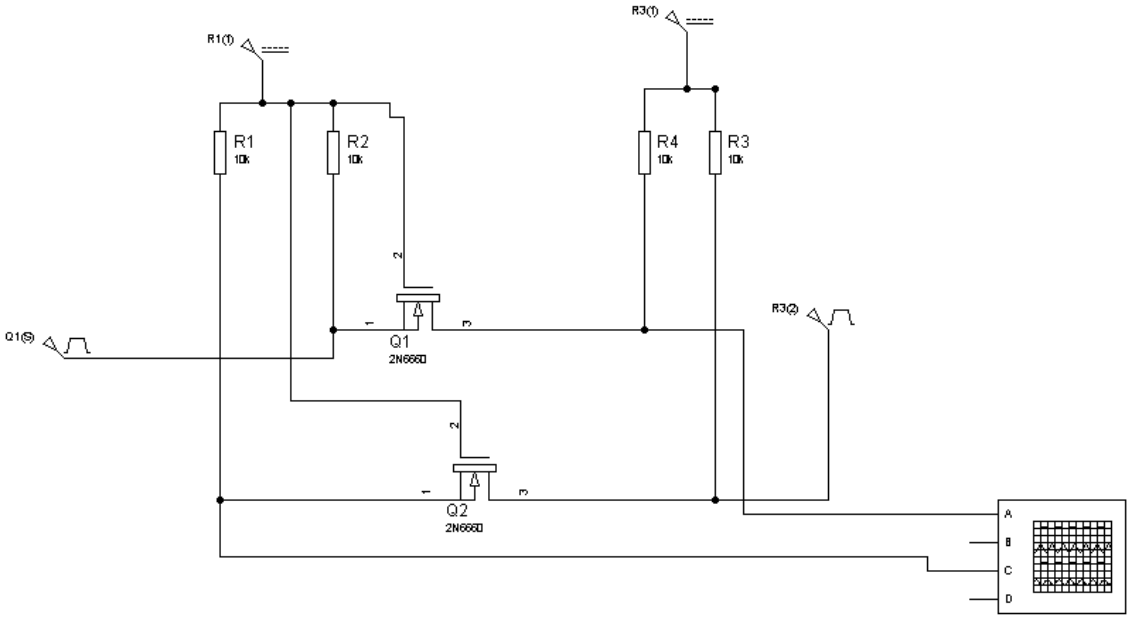
Nous avons donc réussi l'analyse d'image à partir d'une image de tableau de bord, à détecter l'information de la vitesse et les témoins d'alerte et d'alarme et à lancer le programme sur la carte raspberry pi.

Néanmoins, la partie du bouton poussoir n'a pas été traitée et la durée d'éxécution du programme est supérieure à une seconde comme qui était prévu dans le cahier des charges (aux alentours de 10 secondes).

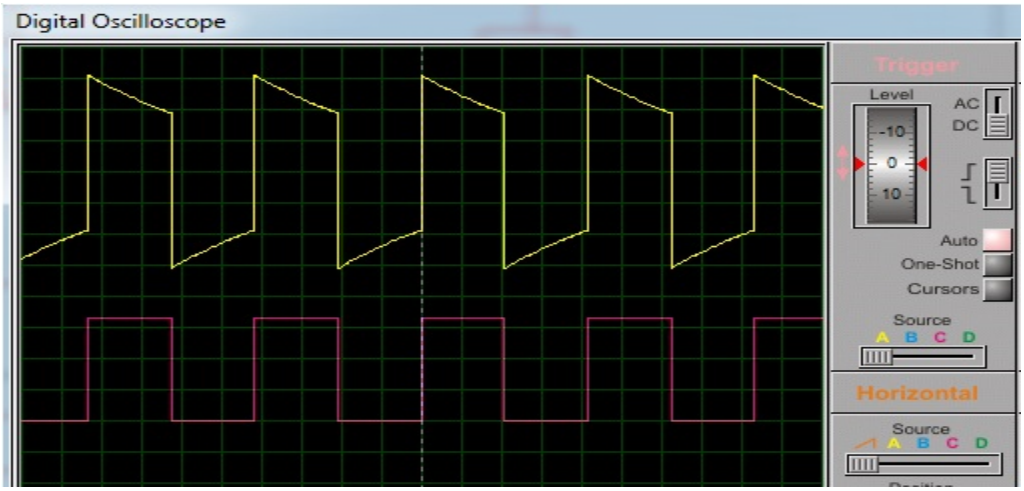
III. Transmission inter modules

III.1. Transmission Raspberry pi et PIC

Le Raspberry Pi fonctionne avec la tension 3.3V alors que le PIC fonctionne avec une tension de 5V, pour établir la liaison entre les deux on a ajouté un circuit de conversion de tension.



Simulation du circuit sur ISIS :



On a un signal de 3.3 v à l'entrée et on a pu recupérer un signal de 5 V à la sortie.

la liaison entre le PIC et le Raspberry pi est une liaison série et pour la transmission on a choisi d'utiliser le protocole UART.

L'UART (Universal Asynchronous Receiver Transmitter) est un protocole de communication série bidirectionnelle qui necessite deux lignes une pour l'emission TX et l'autre pour la reception RX.

- **Liaison uart du Raspberry pi**

Le Raspberry pi propose plusieurs protocoles de communication :

➜I2C
➜SPI
➜UART

Nous avons choisi d'utiliser l'UART parce que les protocoles I2C et SPI ne permettent pas une communication asynchrone.

Pour configurer l'UART sur le Raspberry pi, il suffit de configurer le fichier ttyAMA0 qui se trouve dans le répertoire /dev/.

Configuration de l'UART sur le Raspberry pi :
Pour configurer l'UART on doit éditer sur le terminal du Raspbian les fichiers suivants:

/boot/cmdline.txt : Pour supprimer l'utilisation de l'UART au demarrage.
/etc/inittab : Pour désactiver le terminal au demarrage.

Enfin on configure la vitesse de la liaison série à 19200 bauds sur le fichier /dev/ttyAMA0.

Commande :

stty -F /dev/ttyAMA0 19200 cs8 -cstopb -onlcr -echo -echoe -echok -opost

Les données qu'on veut transmettre au PIC seront écrites sur ce fichier pour cela on a utilisé un programme en C++ qui permet d'ouvrir et écrire sur le fichier /dev/ttyAMA0 les informations acquises (vitesse et témoins).

- **Liaison UART du PIC**

Pour la configuration de l'UART sur le PIC16F690, on a utilisé le compilateur CC5X pour la compilation du code C et afin de générer un code assembleur qui sera utilisé dans la suite du programme.La vitesse de transmission est la même que dans le Raspberry pi 19200 bauds.

Le code C est constitué de trois fichiers :

UART.C le fichier de définition de fonction

Interrupt.C pour la gestion de l'interruption
main.C le programme principal

Dans le main on a définit les constantes fosc et baud dans l'entête, aprés on a utilisé les fonctions suivantes :
setup_UART() : pour l'initialisation de l'UART du PIC en fonction des paramètres définies.
getch() : Pour tester le buffer de reception et pour retourner le caractère reçu.

Finalement le caractère retourné par la fonction getch() sera utilisé dans la suite du programme pour piloter le module de restitution audio.

IV. Transmission Bluetooth

IV.1. Introduction

Pour la transmission des données de la moto vers le casque nous utilisons l'emetteur WT12-A-AI 2.1. Cet émetteur utilise la technologie Bluetooth 2.0+EDR (Enhanced Data Rates) qui est trois fois plus rapide que le Bluetooth 1.2 tout en consommant moins d'énergie.

Comme on peut le voir sur le diagramme en bloc le WT12 possède un microcontroleur BlueCore04, une antenne, une memoire Flash et un oscillateur à 26Mhz. Il possède également différentes interfaces lui permettant de communiquer avec d'autres périphériques. Pour notre projet nous aurons besoin des interfaces USB, PIO et PCM

Pour commencer il faut configurer le WT12, pour cela nous le connectons à un PC à l'aide des interfaces USB et PIO. Nous avons réalisé le montage expliqué dans la partie suivante cependant celui ci n'a pas fonctionné en effet le WT12 n'envoyait aucune données vers le PC et vice versa. Nous n'avons pas réussi à comprendre l'origine du problème cependant cela n'a pas impacté les autres modules puisque le module bluetooth se situe en bout de chaine du projet.

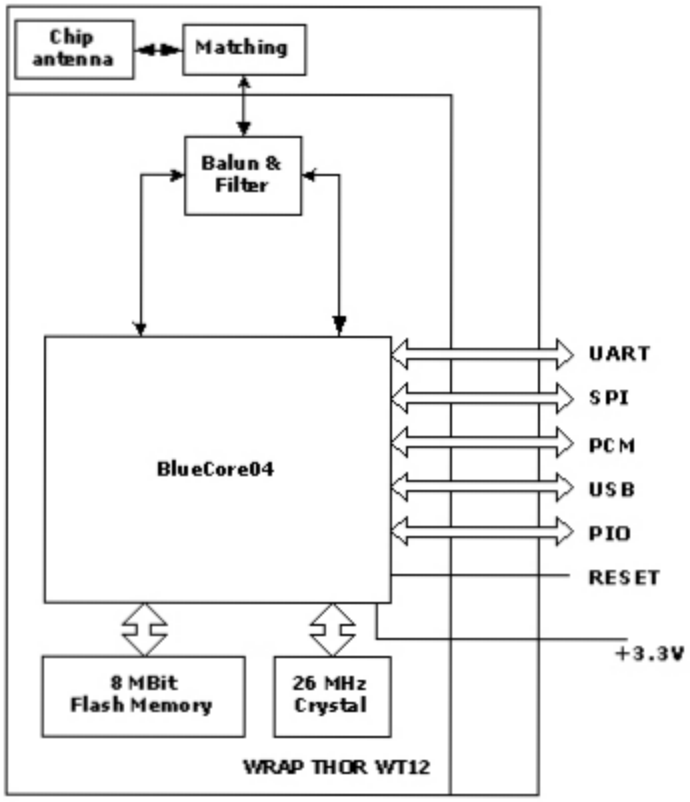


Figure : Block Diagram of WT12

IV.2. Interface USB

Pour adapter l'impédance du WT12 à celle du cable USB on ajoute des résistances entre les pins D+ et D- du WT12 et du cable. Celles ci doivent avoir une valeur variant de 0 à 20Ω avec une valeur nominale 10Ω et une tolérance de 1%.

On ajoute une résistance de 1.5kΩ entre le pin PIO2 et USD_D+ comme indiqué dans la Datasheet. Ensuite on connecte un diviseur de tension d'un coté sur l'entrée USB_ON et de l'autre sur le VBUS(5V) du cable USB. Il doit permettre d'abaisser la tension issue du PC à 3.3V afin que le WT12 puisse détecter si le câble USB est branché au PC ou non.

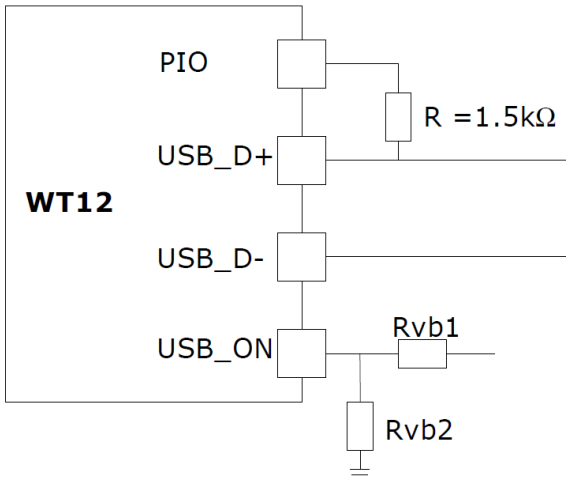


Figure : USB interface

Une fois le montage terminé on installe les drivers ainsi que le logiciel BlueSuite tools permettant de configurer le WT12.

Pendant la configuration on modifie directement le registre PSKEY_PCM_CONFIG32 afin de choisir le mode de fonctionnement du PCM. Pour cela on utilise un fichier excel fourni par Bluegiga qui permet de calculer la valeur du registre en fonction du mode de fonctionnement choisi.

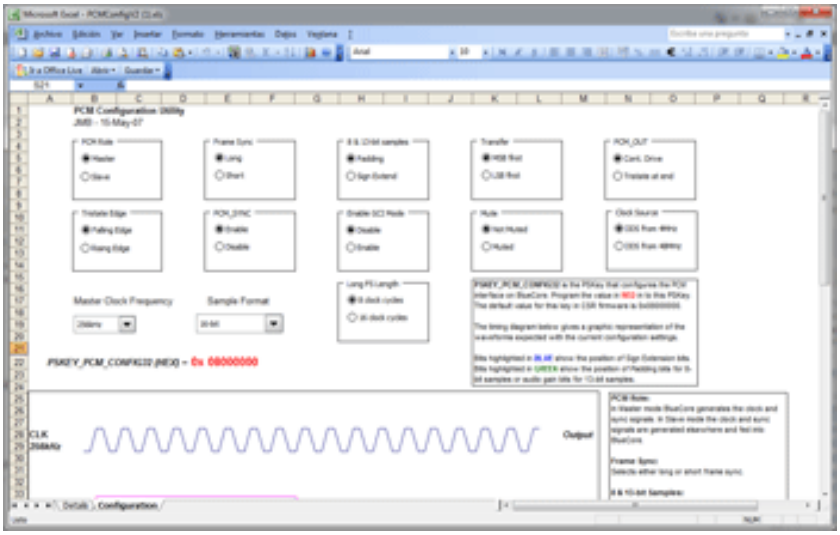


Figure : PCMConfigV2

IV.3. Interface PCM

PCM : **P**ulse **C**ode **M**odulation ou modulation impulsionnelle codée est une représentation numerique d'un signal analogique via son échantillonnage. Elle permet la continuelle émission et réception de données audio non compressée via Bluetooth.

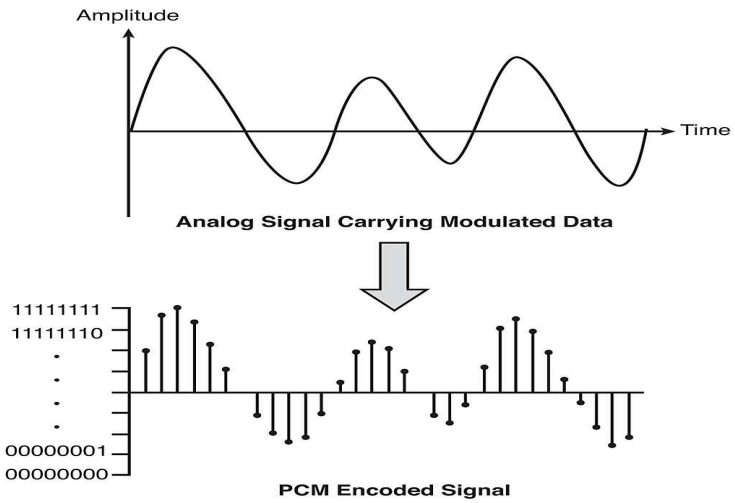
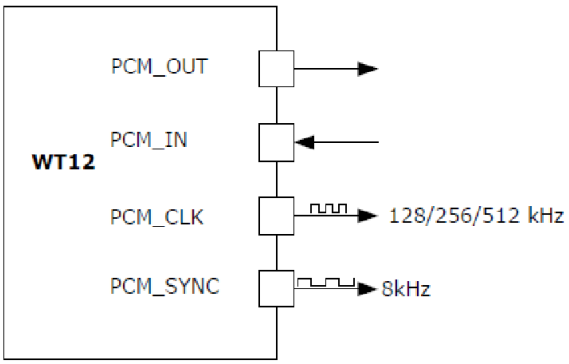


Figure : PCM

L'interface PCM peut fonctionner suivant deux modes, Master ou Slave. En Master le Wt12 génère une horloge de 128, 256 ou 512kHz suivant la configuration qu'on lui a donné. En Slave l'horloge doit être fournie au Wt12 avec une fréquence maximale admissible de 2048kHz.



WT12 as PCM master

On choisit d'utiliser le WT12 en Master. L'émetteur possède une sortie PCM_CLK permettant éventuelement de le synchroniser avec d'autres périphériques ainsi qu'une entrée PCM_IN recevant les données à transmettre. L'entrée PCM_SYNC indique le début de chaque nouveau mot binaire et possède une fréquence de 8kHz. A chaque front montant de l'horloge PCM_CLK les données disponibles sur PCM_IN sont émises. Les données sur PCM_IN étant en 16 bits la fréquence de PCM_CLK doit être configurée à 128kHz.

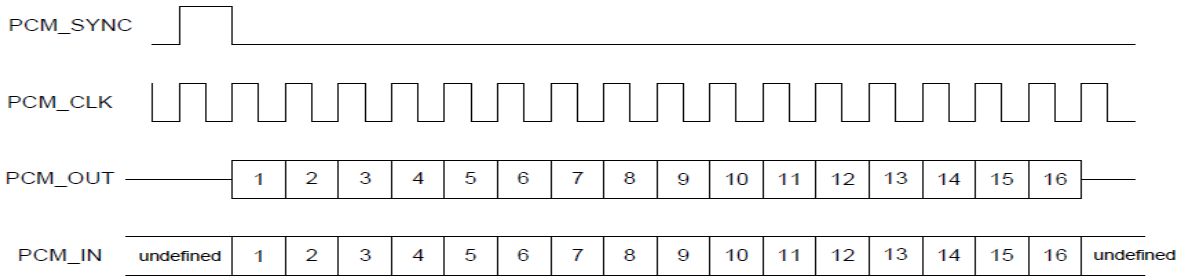


Figure : Short frame sync (shown with 16-bit Companded Sample)

IV.4. L'oreillette Bluetooth

Le récepteur est une oreillette bluetooth CyberBlue qui peut s'apareiller avec n'importe quel émetteur bluetooth de version antérieur à la V3.0. Il possède une autonomie de 3 heures en marche et 50 heures en stand by.

V. Diagramme de Gant

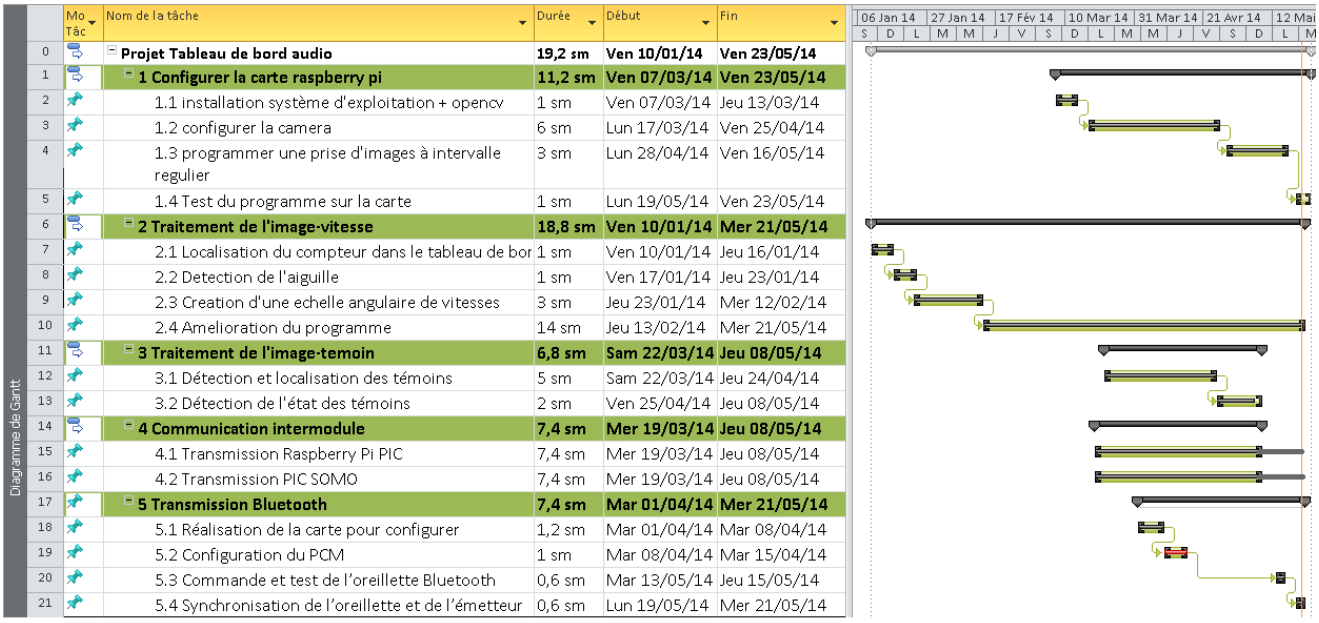


Diagramme de Gantt

	Mo Tâc	Nom de la tâche	Durée	Début	Fin
0		**Projet Tableau de bord audio**	**19,2 sm**	**Ven 10/01/14**	**Ven 23/05/14**
1		**1 Configurer la carte raspberry pi**	**11,2 sm**	**Ven 07/03/14**	**Ven 23/05/14**
2		1.1 installation système d'exploitation + opencv	1 sm	Ven 07/03/14	Jeu 13/03/14
3		1.2 configurer la camera	6 sm	Lun 17/03/14	Ven 25/04/14
4		1.3 programmer une prise d'images à intervalle regulier	3 sm	Lun 28/04/14	Ven 16/05/14
5		1.4 Test du programme sur la carte	1 sm	Lun 19/05/14	Ven 23/05/14
6		**2 Traitement de l'image-vitesse**	**18,8 sm**	**Ven 10/01/14**	**Mer 21/05/14**
7		2.1 Localisation du compteur dans le tableau de bor	1 sm	Ven 10/01/14	Jeu 16/01/14
8		2.2 Detection de l'aiguille	1 sm	Ven 17/01/14	Jeu 23/01/14
9		2.3 Creation d'une echelle angulaire de vitesses	3 sm	Jeu 23/01/14	Mer 12/02/14
10		2.4 Amelioration du programme	14 sm	Jeu 13/02/14	Mer 21/05/14
11		**3 Traitement de l'image-temoin**	**6,8 sm**	**Sam 22/03/14**	**Jeu 08/05/14**
12		3.1 Détection et localisation des témoins	5 sm	Sam 22/03/14	Jeu 24/04/14
13		3.2 Détection de l'état des témoins	2 sm	Ven 25/04/14	Jeu 08/05/14
14		**4 Communication intermodule**	**7,4 sm**	**Mer 19/03/14**	**Jeu 08/05/14**
15		4.1 Transmission Raspberry Pi PIC	7,4 sm	Mer 19/03/14	Jeu 08/05/14
16		4.2 Transmission PIC SOMO	7,4 sm	Mer 19/03/14	Jeu 08/05/14
17		**5 Transmission Bluetooth**	**7,4 sm**	**Mar 01/04/14**	**Mer 21/05/14**
18		5.1 Réalisation de la carte pour configurer	1,2 sm	Mar 01/04/14	Mar 08/04/14
19		5.2 Configuration du PCM	1 sm	Mar 08/04/14	Mar 15/04/14
20		5.3 Commande et test de l'oreillette Bluetooth	0,6 sm	Mar 13/05/14	Jeu 15/05/14
21		5.4 Synchronisation de l'oreillette et de l'émetteur	0,6 sm	Lun 19/05/14	Mer 21/05/14

Conclusion

Le projet de quatrième année nous a permis non seulement d'apprendre à rédiger un cahier des charges, de découvrir de nouvelles technologies notamment le

traitement d'images, la transmission de bluetooth; mais aussi à travailler en groupe de façon organisée et efficace, nous donnant ainsi un aperçu de ce qui se passe en entreprise.

Cependant il ne marche pas aussi bien que nous l'avions souhaité. Notre principale lacune a été de mal évaluer le niveau de difficulté des différentes tâches. La gestion du projet a été un défi qui a été dur à relever et nous nous sommes rendu compte de nos lacunes en terme d'organisation des tâches.

Pour la démonstration finale, nous avons bien séparé les différentes parties afin de pouvoir isoler les modules au cas où certains modules ne fonctionneraient pas comme prévu.

Annexes

Bibliographie

- l'installation de raspbian : https://www.benji1000.net/tutoriel-pour-debuter-avec-le-raspberry-pi/
- installation opencv : http://indranilsinharoy.com/2012/11/01/installing-opencv-on-linux/
- Analyse d'image : http://fr.wikipedia.org/wiki/Raspberry_Pi http://docs.opencv.org/ http://www.cplusplus.com/reference/vector/vector/erase/ http://www.youtube.com/watch?v=VBachuL6phQ&app=desktop&hd=1
- Transmission Bluetooth : https://www.bluegiga.com/en-US/
- Datasheet SOMO https://www.sparkfun.com/datasheets/BreakoutBoards/SOMO-14D-DS-rev3.pdf
- Datasheet PIC http://www.microchip.com.tw/Data_CD/Datasheet/8-Bits/41262C.pdf
- Datasheet LM555 (Horloge Externe) http://www.fairchildsemi.com/ds/LM/LM555.pdf
- Transmission Raspberry pi : http://www.poivron-robotique.fr/