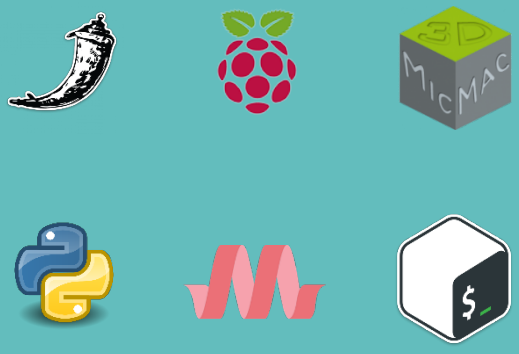
Rapport

Projet Développement

# RaspiModel : Application génératrice de modèles en trois dimensions avec une Raspberry Pi




ENSG Géomatique

ÉCOLE NATIONALE DES SCIENCES GÉOGRAPHIQUES

UNIVERSITÉ PARIS-EST MARNE-LA-VALLÉE

Marion CHARPENTIER

Master 1 Géomatique - ENSG / UPEM


30/05/2017

# TABLE DES MATIERES

# INTRODUCTION

Depuis le début de l'informatique, les ordinateurs tendent à devenir de plus en plus miniaturisés et de plus en plus rapides. Par la suite des nano-ordinateurs monocarte à processeur ARM ont vu le jour comme les Raspberry Pi, accompagnés de leurs accessoires tous aussi réduit les uns que les autres. Par ailleurs l'engouement pour la confection de modèles en trois dimensions est grandissant.

Ainsi, le projet que j'ai réalisé porte sur ces thématiques, miniaturiser la confection de modèles en trois dimensions.

Le but de ce projet est de modéliser en trois dimensions un objet en utilisant une Raspberry Pi équipée d'une mini caméra, d'un robot biaxial et d'une plateforme mouvante.

Le défi de ce projet est de faire communiquer tous les appareils entre eux et de prendre en main des technologies que je n'ai encore que peu étudiées, tout en mettant à profit mes connaissances en imagerie optique.

# I – ORGANISATION, CHOIX DES LANGAGES, TECHNOLOGIES ET MATERIELS UTILISES

## A – ORGANISATION

Ce projet de développement d'application a pour durée initiale 200h, où une salle nous est mis à disposition. De surcroît en plus de travailler à l'école, une grande partie de mon projet a été réalisé sur mon temps libre le soir et le week-end. Cela m'a permis de faire plus de choses et de mieux comprendre certains points qui m'étaient encore inconnus au début du projet. Ce temps en plus m'a également permis de créer des pièces sur mesure tels que la voiture télécommandée avec sa remorque et le bloc pour les piles du robot.

Ce programme ayant besoin d'un routeur pour faire la connexion entre les appareils et d'une certaine liberté géographique, j'ai choisi d'utiliser mon smartphone avec la connexion 4G. Cela m'a permis de travailler aussi bien à l'école qu'à mon domicile.

Pendant toute cette durée j'ai eu l'aide de mes professeurs, ainsi que de Guillaume Hérault qui m'a aidé pour m'expliquer certains points que je n'avais pas compris.

## B – LANGAGES, TECHNOLOGIES

Pour la réalisation de ce projet, les langages de programmation sont assez nombreux bien qu'ils soient logiquement utilisés pour communiquer entre eux. En effet la Raspberry Pi a beaucoup de petits programmes déjà créés exploitables en python, de plus le Bash/Shell a été beaucoup utilisé car Raspbian est le système d'exploitation.

J'ai utilisé pour les calculs de reconstruction en 3D le logiciel Micmac de l'IGN, c'est un logiciel que je connais depuis plusieurs années et qui a l'avantage d'être scriptable. Pour le faire fonctionner, des commandes ont été lancées via des scripts en Shell. Ce langage étant assez simple d'utilisation et extrêmement rapide par rapport à Python, Shell a souvent été utilisé lors des fonctions qui « jouaient la montre ».

Pour l'interface graphique, un premier jet a été fait en QT, finalement c'est avec le micro Framework Flask que l'interface a été façonnée. Ainsi le langage web classique (html, css et js) a été utilisé en plus de python via Flask et Jinja.

Tous les appareils ont pu communiquer entre eux avec un système Wifi via un smartphone, l'application est donc faite pour être dans le futur totalement sur Internet et non plus simplement en semi-local comme cela est le cas ici. En effet les systèmes SSH et FTP sont utilisés par Python et Shell, mais les ordinateurs (serveur web, raspberry pi et serveur de calculs) sont en local, en d'autres termes ils ne sont pas accessibles depuis l'extérieur du réseau wifi du smartphone. En revanche, ils ont accès à une connexion internet pour recevoir et envoyer des données (type librairies Python et viewer en ligne pour afficher les nuages de points).

### C – MATERIEL

Pour réaliser ce projet un certain nombre d'appareils ont dû être utilisés et/ou fabriqués, en voici la liste (les éléments soulignés ont été fournis par l'école) :

- Raspberry pi 2B
- Carte micro SD 8Go de classe 10 (écriture rapide)
- Caméra RVB Rev 1.3 (compatible Raspberry pi) caractéristiques disponibles en annexe 1.
- Asus T100HAN sous Windows 10 (Ordinateur / Tablette de liaison)
- HP Pavilion G7 2346-sf sous Ubuntu (Ordinateur de calculs)
- Connexion internet (via Wifi avec un smartphone)
- Alimentation Raspberry (batterie 5V smartphone)
- Clé wifi
- Voiture télécommandée + télécommande (jouet pour enfant)
- Robot biaxial
- Création d'un bloc de piles pour le robot (soudures nécessaires + câbles + cosses)
- 4 piles pour le robot + 3 piles pour la voiture télécommandée

## II – METHODE ET REALISATIONS

La réalisation de ce projet a été dans une grande partie dans le même ordre que le plan qui va suivre, le travail a débuté sur la Raspberry Pi puis sur le serveur de calculs pour terminer sur l'important travail de l'interface graphique.

### A – INSTALLATION DE LA RASPBERRY PI

Pour commencer, il a fallu apprendre comment fonctionne une Raspberry Pi. Pour cela de très bonnes explications sont disponibles sur internet. La Raspberry pi était vide, elle ne possédait pas de système d'exploitation, j'ai donc choisi d'y installer « *Raspbian Jessie with PIXEL* », la version du 11 janvier 2017 (voir ici).

Pour l'installation j'ai suivi ce protocole :

- Utilisation de Win32DiskImager pour créer une carte SD bootable (téléchargeable ici)
- Mise en route de la Raspberry pi après avoir brancher le clavier, la souris, l'écran et la carte micro SD.
- Entrer l'identifiant et le mot de passe lors de la première utilisation (ainsi que pour le reste du projet) :
  - ID : *pi*
  - Passe : *raspberry*
    Attention le clavier risque d'être en mode Qwerty au premier démarrage, il faudra alors taper : *rqspberry*.
- Pour configurer la raspberry pi lors du premier démarrage, suivre le tutoriel qui se trouve ici. Penser de temps en temps à mettre à jour et redémarrer la raspberry pi : « *sudo apt-get update -y && sudo apt-get upgrade -y && sudo reboot* »

Ensuite il faut installer la caméra de la raspberry pi, bien faire attention que la raspberry soit débranchée et mettre la fiche dans le bon sens (coté bleu vers les ports USB). Pour l'activation j'ai suivi deux tutoriels dont un est la documentation de la caméra :

1. Pour activer la caméra, retourner dans « *sudo raspi-config* » et suivre ce tutoriel.
2. Installation de Python PiCamera : ici.

La dernière étape pour rendre la raspberry opérationnelle est de lui activer le SSH pour y accéder depuis un autre ordinateur (ici ce sera la tablette Asus). Les informations nécessaires à cette manipulation sont décrites sur ce site.

Ces différents liens décrivent de façon détaillé les étapes à suivre, mais ne sont pas utiles dans ce rapport si la raspberry pi est déjà configurée.

## B – EXPLICATION DES FONCTIONNALITES

Pour avoir une organisation plus simple, l'écriture de l'application est découpée en plusieurs scripts (le diagramme de déploiement est disponible en annexe 2, et le diagramme d'utilisation en annexe 3). Ainsi plusieurs scripts plus ou moins longs ont été écrits, cela permet de cibler les éventuelles erreurs et permet une meilleure lecture du code de l'application. Cela permet aussi de plus facilement améliorer les fonctionnalités.

### 1 – RASPBERRY PI

Dans ce projet la Raspberry Pi a comme seules vocations de réaliser les acquisitions et de faire bouger le robot. Ainsi le premier travail à réaliser est de tester le fonctionnement de la caméra pour acquérir des vidéos ou des images. Après plusieurs itérations il s'est révélé que l'utilisation d'images est plus pertinente, car les images sont moins lourdes qu'une vidéo et peuvent être traitées en quasi-direct pour faire un retour vidéo image/image. Cela va fortement influencer l'affichage des images en fonction du wifi.

Ces fichiers qui vont suivre sont placés dans le dossier « */home/pi/Videos/scripts_python* » de la Raspberry Pi.

#### INITIALISATION – PYTHON

Ce script a été écrit en python, il utilise les librairies *os* et *shutil* (initialisation.py). Il permet de vérifier si le dossier d'acquisition est déjà présent avec *os.path.exist*(). S'il est déjà existant, il sera supprimé avec *shutil.rmtree*(), cette commande permet de supprimer un dossier récursivement, puis le recrée avec *os.mkdir*().

C'est un script très simple mais qui permet de remplir le dossier sans risquer de mal écraser les fichiers existants.

#### ACQUISITION – BASH

Pour des raisons pratiques, notamment pour de petits calculs, ce script a été écrit en Bash (raspistillAcqui.bash). En effet le Shell n'étant pas prévu pour faire des calculs, il n'est pas adapté dans notre cas. Le Bash ressemble énormément au Shell et est capable de faire des opérations simples, c'est pour cela qu'il est utilisé ici.

La première action réalisée par ce script est d'éteindre le programme Raspistill qui est le module d'acquisition de photos. Cette action permet de sécuriser et ne pas multiplier les activités sur la caméra, au cas où Raspistill ne serait pas éteint.

Ce script prend en entrée la durée de l'acquisition en nombre de secondes, lorsqu'il sera appelé il faudra utiliser cette syntaxe pour un exemple de 5 secondes : « *bash raspistillAcqui.bash 5* ». Cette valeur est par la suite multipliée par 1000 pour avoir le nombre de millisecondes, puis sera incorporé à la ligne de commande lançant la prise de vue.

L'acquisition est lancée par la commande « *raspistill -vf -hf --nopreview -q 5 -o /home/pi/Videos/donnees_acqui/img_%1d.jpg -tl 500 -t $temps* », ainsi Raspistill va prendre une photo toutes les demi secondes (-tl 500) pendant le nombre de secondes rentré en paramètres et multiplié par 1000 (*-t $temps*). Chaque image sera enregistrée dans le dossier des acquisitions créé avec le fichier « initialisation.py » et auront un nom avec comme suffixe « img_ », un fichier incrémenté de 1 à chaque enregistrement, le tout au format JPG (exemple : img_1.jpg, img_2.jpg …) . Comme la caméra est positionnée à l'envers sur son support il faut retourner l'image à l'aide de « *-vf -hf ». Le paramètre « -q 5 » correspond à la qualité de l'image à enregistrer sur une échelle de 0 à 100.*

Lorsque raspistill a écoulé le nombre de secondes d'acquisition, le script s'arrête automatiquement.

### RETOUR DES IMAGES APRES L'ACQUISITION – PYTHON

Cette fonction a pour vocation d'être activée après l'acquisition car c'est avec ces images que le diaporama se crée. Pour l'écrire, les librairies suivantes ont été utilisées : *os* pour lire un dossier, *time* pour utiliser *sleep*() et *Image* pour ouvrir des images, les modifier et les enregistrer.

Une boucle va tourner tant que la variable « *arret* » n'est pas vraie (*True*). Durant cette boucle le script va regarder si parmi les fichiers du dossier « */home/pi/Videos/* » un se nomme « arret.txt », si oui alors le script va retourner *True* et arrêter le script, par conséquent l'affichage du diaporama aussi. Si au contraire le script ne rencontre pas ce fichier, il va récupérer les images précédemment acquises une à une pour réduire leur taille et les enregistrer sous le nom de pic.jpg dans le dossier « */tmp/stream* » afin que le serveur de streaming puisse les lire automatiquement. Les images vont défiler par intervalle de 5 secondes, le temps prend en compte le retard pris par python pour convertir les images en faisant une soustraction du temps pris par le temps total de passage c'est-à-dire 5 secondes. Puis la boucle va recommencer et ce jusqu'à ce que le fichier « *arret.txt* » apparaisse dans le dossier d'acquisition, traduisant la fin des calculs.

### ACTIVATION DU ROBOT – PYTHON

Il y a deux fichiers python pour activer les robots, un pour le servo supérieur et un autre pour le servo inférieur. Ils sont construits de la même manière, c'est pour cela qu'un seul fichier sera expliqué. Les robots doivent se brancher sur les ports GPIO de la raspberry pi (voir le schéma en annexe 4). Pour connecter les câbles dit de « commande » le servo moteur supérieur est branché est le GPIO 18 (pin 12) et l'inférieur sur le GPIO 17 (pin 11). L'explication détaillée des branchements effectués se trouve dans la partie C - Création et montage du matériel.

Ainsi les librairies suivantes ont été nécessaires à ce script : *time* pour sa fonction *sleep*(), *sys* pour récupérer l'argument passé lors du lancement de la fonction et *RPi.GPIO* pour communiquer avec le servo moteur utilisé dans le fichier. Le servo est initialisé et s'il peut bouger, celui-ci bouge d'autant qu'on lui a demandé en paramètre en entrée. Une pause, même minime, est nécessaire pour laisser le temps au moteur de bouger, cela évite des à-coups. Puis le servo est éteint.

## 2 – SERVEUR DE CALCULS

C'est sur cette machine qu'est installé Micmac. Au début du projet, une version ARM avait été installée sur la Raspberry Pi mais les calculs étant très lourds et long, il a été décidé d'utiliser un ordinateur distant pour gagner du temps. C'est donc un HP Pavilion G7 2346-sf équipé de SSH et FTP qui a été utilisé comme serveur de calculs.

## CALCULS MICMAC – SHELL

Afin d'améliorer les temps de calculs qui sont déjà assez long, l'algorithme de détection de points d'intérêts SIFT présent par défaut dans Micmac a été remplacé par Digeo. Ce dernier est moins robuste que SIFT du fait qu'il a moins été testé mais il a l'avantage d'être plus rapide.

Ainsi le script est écrit en Shell, cela est beaucoup plus simple pour se mouvoir dans les dossiers et lancer les commandes Micmac, mais aussi parce que Shell est beaucoup plus rapide que Python.

Ce script est composé d'une fonction « testErreur » qui lorsqu'on l'appelle va vérifier si le fichier qui sert de rapport d'erreur est présent dans le dossier de travail. Si les dossiers et fichiers ne sont pas présents, cela va afficher une erreur dans le terminal mais cela n'influe en rien sur le déroulement du script, l'erreur est affichée à titre informatif par la commande *rm*. Cette fonction est appelée après chaque commande micmac (mm3d) pour vérifier si l'étape s'est bien déroulée. Si une erreur apparait alors le script micmac est arrêté.

Tout d'abord le dossier des images et les fichiers de fin et d'erreur potentiellement présents sont supprimés par sécurité puis les images de l'acquisition sont récupérées depuis de ftp de la raspberry pi avec les commandes : *sshpass* qui sert à écrire dans le script le mot de passe et *scp* qui va copier des fichiers depuis le ftp d'une machine distante.

Puis en se plaçant dans le dossier des images, le calcul micmac peut commencer avec le *Tapioca* qui va rechercher dans toutes les paires d'images possibles les points homologues, c'est-à-dire des points d'intérêts présents sur plusieurs images. Ainsi la commande suivante est lancée : « *mm3d Tapioca All ".*jpg" -1 Detect=Digeo > Tapioca.txt* ». Les arguments *All* et *-1* indiquent que les images ne sont pas sous échantillonnées lors de la recherche. Le rapport du calcul est enregistré dans le fichier Tapioca.txt .

S'en suit avec le *Tapas* qui va mettre en place les images les unes avec les autres en calculant le sommet de prise de vue de chaque image par rapport aux autres, c'est-à-dire de savoir d'où ont été prises chacune des images. La commande suivante est donc lancée : *« mm3d Tapas RadialBasic ".*jpg" Out=MEP > Tapas.txt* ». Ici le modèle de calibration de la caméra est *RadialBasic*, il a été choisi du fait qu'il se rapproche le plus de la calibration d'une Pi Caméra. La mise en place est exportée dans la variable MEP et le rapport du calcul est enregistré dans le fichier Tapas.txt tout comme pour le Tapioca.

Puis vient le *C3DC* qui va calculer le model 3D à partir de la mise en place MEP calculée avec le *Tapas*. Voici la ligne lancée : « mm3d C3DC QuickMac ".*jpg" MEP ZoomF=4 > C3DC.txt », tout comme les deux précédentes commandes le rapport de calcul est enregistré dans un fichier texte.

Si tout s'est bien passé côté micmac, le nuage de points généré est au format .ply, il est converti avec PotreeConverter pour créer la page html de visualisation du nuage de points colorisé. Tous les dossiers nécessaires à la visualisation sont envoyés sur le dépôt loué chez OVH avec mon nom de domaine. Ici les fichiers sont envoyés sur *raspberry.marion-charpentier.fr*. Le nuage de points au format ply est aussi envoyé sur le serveur pour qu'on puisse le télécharger ultérieurement. Seulement, comme les navigateurs internet interprètent mal les fichiers ply, il est nécessaire d'ajouter .xyz à la fin du fichier, l'utilisateur devra donc retirer cette surcouche au nom du nuage de points pour le lire avec Meshlab ou CloudCompare par exemple.
Ces envois sur mon ftp personnel ont été réalisés à l'aide de *wput* avec l'option *-u* qui permet d'écraser les fichiers s'ils existent déjà.

### VERIFICATION DES ERREURS MICMAC – PYTHON

Les erreurs générées par micmac sont facilement détectables lorsque l'on possède les rapports de calculs, ce qui est notre cas. Il suffit de chercher la chaine de caractère « FATAL ERROR » dans les fichiers texte.

Cette fonction est une boucle qui ne s'arrête pas avant la fin du calcul ou avant d'avoir une erreur dans micmac et va recommencer toutes les 10 secondes. En effet, elle va chercher grâce en partie à la librairie *os* dans les fichiers Tapioca.txt, Tapas.txt et C3DC.txt lorsqu'il existe s'ils possèdent une chaine de caractères correspondant à notre erreur. Si cela arrive, le fichier *fin.txt* et *erreur.txt* sont créés (cela aide à la détection de la fin du calcul et d'erreur dans la suite du programme). De plus Micmac est stoppé par la commande « *subprocess.call(['pkill', '-f', 'mm3d'])* ». Puis le script a fini son travail et s'arrête avec *break*.

## 3 – SERVEUR WEB FLASK

Le serveur web a été mis sur la tablette Transformer Asus en localhost. Ceci devait être totalement en ligne à la fin du projet, mais par manque de temps et de connaissances en sécurité et en réseau, j'ai pris la décision de laisser l'application en locale via mon smartphone.

Afin de pouvoir communiquer avec les autres appareils avec Python, le Framework Flask a été utilisé. De ce fait nous disposons de quatre fichiers à placer dans le www du serveur web local, ici c'est dans le dossier « *www* » de EasyPhp. Ces fichiers sont le main.py, le accueil.html, le style.css et le init.css. Pour l'interface le Framework visuel Materializecss a été utilisé, de ce fait des fichiers supplémentaires sont présents. Voici l'organisation de ces fichiers :

- Main.py
- Static
  - Css
    - materialize.css
    - materialize.min.css
    - style.css
  - Fonts
    - Roboto …
  - Js
    - init.js
    - materialize.js
    - materialize.min.js
- Templates
  - Accueil.html

Les fichiers vont communiquer de la manière suivante :

- Lors du chargement de l'application :

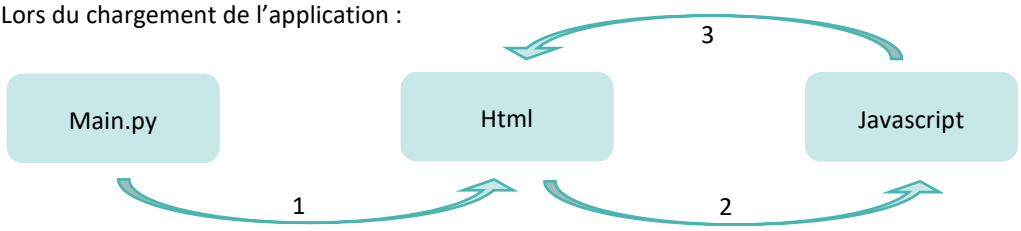



Figure 1 : Communication entre les fichiers lors du chargement de l'application.

- Lors des actions suivantes :


Figure 2 : Communication entre les fichiers lors des actions dans la page.

L'application est donc accessible via la localhost à charger dans le navigateur internet. Comme c'est une version en locale, l'accès se fait avec le port 5000 : « *localhost:5000* ».

## FONCTION PRINCIPALE – MAIN.PY

### a - BASES DE FLASK

Ce fichier python a été le premier à être écrit pour l'interface. Pour le tester il faut interagir avec l'utilisateur, il faut donc rapidement du contenu html. C'est pour cela que tous les fichiers de cette partie ont été rédigés en simultané.

Le code de ce fichier est organisé sous forme de fonction, cela permet une meilleure lecture et surtout une réutilisation du code.

Flask permet d'utiliser le code de deux manières différentes :

- Sous forme de fonctions simple.
  Exemple :
  ```
  def maFonction() :
      …
  ```
- Sous forme de fonctions qui peuvent être appelées via le javascript (voir la partir sur le javascript).
  Exemple :
  ```
  @app.route('/lienVersMaFonction')
  Def maFonction() :
      …
  ```

Ainsi pour générer la page html lors du chargement de la page, une fonction chargeant le *template* html est lancée avec les lignes :

```
@app.route('/')
def maFonction() :
    …
    return render_template('accueil.html', titre="RaspiModel)
…
#if __name__ == '__main__':
app.run(debug=True)
```

Avec ce code, l'application va charger au démarrage la page *accueil.html* en mettant dans la balise Jinja « *titre* » le mot « *RaspiModel* ».

### b - COMMUNICATION

C'est avec ce programme Python que les connections via SSH aux autres ordinateurs (G7 et Raspberry pi) sont faites avec la librairie *Paramiko*. La vérification de l'état des connexions SSH est opérée dès le chargement de la page, si un des deux ordinateurs distants n'est pas connecté ceci est détecté avec la ligne : *except (paramiko.SSHException, TimeoutError):* et va renvoyer l'état des connexion SSH dans le fichier html pour en informer l'utilisateur.

Quelques-unes unes des fonctions du main.py envoient des requêtes en SSH sur la raspberry pi ou le serveur de calculs pour lancer le fichier python ou Shell correspondant.

De plus certaines des fonctions sont lancées via des fonctions javascript, l'utilisation de JSoniFy depuis le main.py était donc nécessaire pour renvoyer une valeur au javascript.

### c -ROBOT

C'est aussi dans ce fichier que sont activés les deux axes du robot. Les robots sont d'abords « activés » au chargement de l'application pour initialiser leur position. Les maximums et minimums de chaque servo ne sont pas connu d'origine, c'est pour cela que j'ai réalisé des tests pour en déterminer leur centre, minimum et maximum en fonction d l'utilisation que je vais en faire. Voici les valeurs pour chaque servo moteur (les unités sont des rapports cycliques) :

- Servo moteur supérieur branché sur le Pin 12 (GPIO 18) :
  - Centre : 5
  - Minimum : 3.5
  - Maximum : 5.5
- Servo moteur inférieur branché sur le Pin 11 (GPIO 17):
  - Centre : 6.5
  - Minimum : 2.5
  - Maximum : 11

Lorsque l'utilisateur va actionner les boutons de déplacement de la caméra, le fichier main.py va envoyer l'ordre à la Raspberry Pi de lancer les fonctions correspondantes comme bouger vers le haut, le bas, la gauche et la droite. De plus le fichier main.py gère la rapidité de déplacement des axes du robot en récupérant la valeur de rapidité de déplacement du html puis en envoyant un coefficient multiplicateur de vitesse aux fonctions de mouvement. Ces fonctions sont activées à chaque fois que l'on va appuyer sur le bouton correspondant au mouvement, les moteurs vont donc s'allumer et s'éteindre à chaque mouvement. Ceci est nécessaire pour éviter l'effet « grelot » des moteurs qui tremblent lorsqu'ils ne bougent pas mais qui sont allumés, cela est dû au fait que la Raspberry Pi ne consacre pas 100% de son temps au python.

### d - VIDEO

Pour le retour vidéo, trois fonctions en python ont été créées : allumer la caméra, allumer le serveur et éteindre la caméra. Pour allumer la caméra il faut d'abord créer un dossier pour entreposer l'image qui sera lue par le serveur de streaming. Les images vont être générées par Raspistill le module de prise de photo de la PiCamera. Ces images vont être en réalité qu'une seule image du fait que toutes les 0.5 seconde, une image viendra écraser image précédente. Ces images seront donc stockées dans le dossier « */tmp/stream* » de la raspberry pi sous le nom de *pic.png* comme cela est visible dans la commande envoyée via SSH : « *raspistill -vf -hf --nopreview -w 266 -h 200 -q 5 -o /tmp/stream/pic.jpg -tl 500 -t 9999999* ». Grace à cette commande, il est possible de préciser la taille des images à enregistrer (*-w* et *-h*), leur qualité (*-q*) , ainsi que leur orientation. Ici comme la caméra est sur un support, elle est donc positionnée à l'envers c'est pour cela que « *-vh -hf* » sont nécessaires.

La seconde fonction va allumer le serveur de streaming de la Raspberry Pi qui va envoyer via le port 8080 du localhost un retour visuel sur l'image « *pic.jpg* » présente dans le dossier « */tmp/stream* » de la Raspberry Pi

précédemment acquise. Ceci est réalisé avec l'aide de « *mjpg_streamer* », c'est une application simple d'utilisation et relativement compétente pour le streaming en direct. C'est donc avec la commande suivante que le serveur de streaming se met en route via SSH : « *mjpg_streamer -i "/usr/local/lib/input_file.so -f /tmp/stream -n pic.jpg" -o "/usr/local/lib/output_http.so -w /usr/local/www"* ». Cette image est donc lue en boucle et affichée en temps réel à l'adresse suivante : « *http://IP_RaspPi:8080/?action=snapshot* ». Il n'est pas nécessaire de modifier les ports écoutés d'Apache. Il m'a été assez simple de mettre cette solution en marche grâce aux explications présentes sur ce site internet (voir option 4).

Il sera utile d'éteindre la caméra à certains moments, une commande est lancée pour éviter tout dédoublement du programme raspistill : « *pkill raspistill* ». Ainsi le mot clé *pkill* va arrêter tous les processus du nom de *raspistill* sans demander de mot de passe, contrairement à un simple *kill*. Cette commande est lancée à l'aide d'une commande SSH en Python à l'intérieur d'une fonction déclenchable par javascript.

Ces trois fonction python ne renvoient rien. Le JsoniFy a tout de même besoin d'une réponse, c'est pour cela que les trois fonctions renvoient une chaine de caractères vide.

### e - ACQUISITION

La méthode d'acquisition est découpée en deux parties. Ne pouvant pas mettre de paramètres dans les fonctions appelées par javascript, plusieurs fonctions de « redirection » ont été écrites, une pour chaque temps d'acquisition proposé : *acquisition5*, *acquisition10*, *acquisition15*. Chacune de ces fonctions n'ont qu'un seul but : lancer la fonction acquisition principale avec comme paramètre le temps choisi par l'utilisateur. Voici un schéma d'exemple :

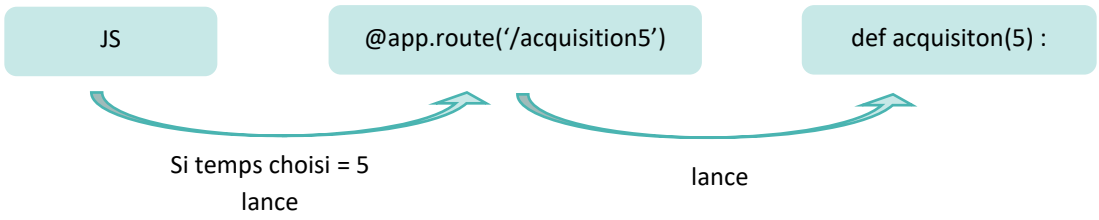



Figure 3 : Schéma d'exemple de passage de paramètre entre javascript e le main.py

La fonction acquisition va lancer en SSH sur la Raspberry Pi le fichier *raspistillAcqui.bash* avec en paramètre $1 le temps choisi par l'utilisateur. Cette fonction ne renvoi rien.

### f - CALCULS MICMAC

Pendant les calculs les images précédemment acquises sont affichées en boucle afin que l'utilisateur puisse les visualiser pendant que les calculs s'exécutent. Ainsi le retour vidéo qui se trouve après l'acquisition est lancé sur la Raspberry Pi via SSH. En même temps, les calculs micmacs sont opérés en exécutant, toujours par SSH, le script micmac se situant sur le serveur de calculs. Au même moment est lancé le script python de vérification de micmac permettant de révéler les erreurs survenues dans micmac. Afin de détecter la fin des calculs ou si les calculs n'ont pas fonctionné, la fonction python « *finCalculs* » est lancée. Celle-ci va vérifier si le fichier fin.txt ou Erreur.txt sont visibles dans le dossier de travail du serveur de calculs. Cette fonction est composée d'une boucle *while* qui va vérifier toutes les 5 secondes la présence de ces fichiers, si d'un des deux vient à survenir, la fonction s'arrête et renvoie un rapport d'erreur à la fonction parente sous forme de booléen *True* ou *False* (False = pas d'erreur, True = une erreur est survenue). Cette réponse est ensuite transmise au javascript par l'intermédiaire de la fonction de calculs pour modifier le html en conséquence.

### g - EXTINCTION DE LA RASPBERRY PI

Une option de dernière minute a été ajoutée à l'application, afin de mieux éteindre la raspberry pi, un bouton a été créé à cet effet dans le html. Son clic va donc enclancher la fonction « *eteindreRasp* » (via le javascript) qui va lancer une commande SSH vers la raspberry pi pour l'éteindre avec la commande « *sudo shutdown now* ». Bien évidement il faut mettre un mot de passe avec cette fonction, c'est pour cela que « *shutdown* » a été ajouté à la liste des commandes qui n'ont pas besoin de mot de passe (mais qui nécessite sudo tout de même). Ceci a été réalisé via la commande « *sudo visudo* » sur la Raspberry Pi et en ajoutant la ligne « *%admin ALL=NOPASSWD: /sbin/shutdown* ». Ces instructions m'ont été fournies sur ce site web.

## FICHIERS WEB CLASSIQUES – HTML, CSS, JS

Ces fichiers sont la base graphique de l'application. Avant de se lancer dans l'écriture de l'interface, une maquette a été réalisée sur Power Point avec des boites pour visualiser l'emplacement des blocs, leurs actions et leurs affichages en fonction des situations.

Le html a été écrit avec la logique de Materializecss qui permet de faire une mise en page design. Le css était donc en partie déjà créé, ce qui m'a permis de gagner beaucoup de temps sur les aspects visuels, j'ai seulement dû ajouter quelques classes au css pour le personnaliser. De plus grâce à ce Framework web, l'application est responsive, c'est-à-dire qu'elle s'adapte à la taille de l'écran du navigateur et fonctionne par conséquent sur smartphone (hormis que cela n'est pas possible pour le moment du fait que l'application est en locale sur la tablette).

Flask n'étant pas dynamique, par choix je voulais que tout se passe sur une seule et même page, la technologie javascript a été utilisée pour « cacher » les éléments en fonction des actions en cours. Voici un schéma explicatif de l'organisation des divisions dans le html:

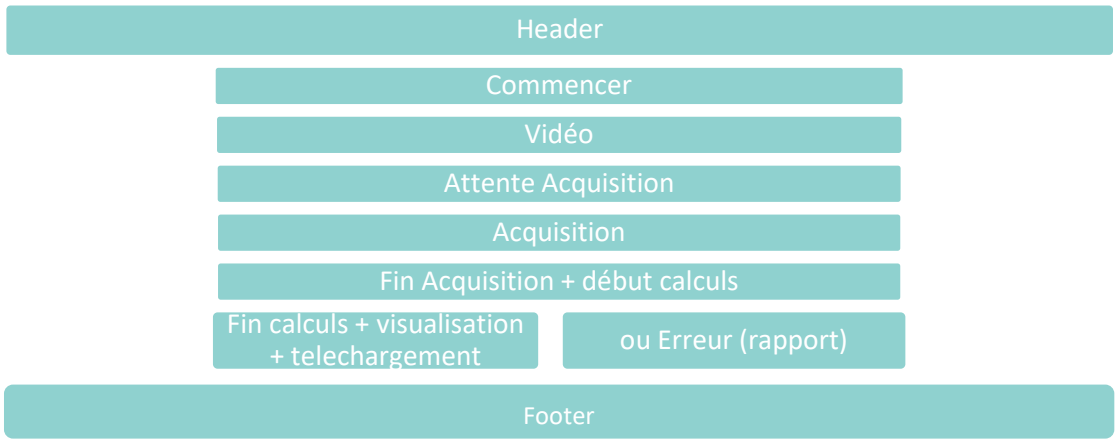



Figure 4 : schéma explicatif de l'organisation des divisions dans le html

Le fichier html sert donc de *template* à Flask au chargement de la page, ce *template* est par la suite modifié par le javascript pour y ajouter des éléments ou en modifier, en afficher et en camoufler.

Ainsi lors du chargement de l'application, le javascript (JS dans le reste du rapport) va lire les statuts de connexion de la raspberry pi et du serveur de calcul. Si tout est connecté, JS ne va rien modifier ; or si un des deux appareils n'est pas connecté, le bouton *Commencer* va se désactiver et le « OK » de l'appareil non connecté va se transformer en « NO » rouge. Cette vérification n'est effectuée qu'au chargement de la page, c'est-à-dire que si un des appareils se déconnecte pendant l'utilisation de l'application, son changement de statut ne sera pas visible.

Pour plus de visibilité dans le JS, les objets ont été renommés sous forme d'alias.

Une grande partie des fonctions du main.py sont ordonnées par le JS via des événements. C'est ainsi que le bouton *Commencer* possède un événement dans le JS qui va s'activer lors du clic. Va alors se déclencher une série de fonctions JS qui vont s'appeler entre elles, ce bouton va enclancher la fonction « *commencer*() » qui va cacher les éléments qui ne sont plus nécessaire à l'affichage, centrer la caméra sur la page et afficher la div de compte à rebours avant l'acquisition.
Cette fonction va ensuite activer « *attente*() », ici on regarde si la case présente au démarrage précisant si on voulait conserver les secondes d'attente avant l'acquisition est restée cochée ou non. Si oui, alors un décompte s'affiche grâce à une suite de lignes retardées de x secondes, chacune affichant le temps restant avant de lancer à la fin du décompte la fonction « *debutAcqui*() ». Si la case est décochée, le décompte se ne fait pas et passe directement à la fonction suivante c'est-à-dire « *debutAcqui*() ».
Tout comme la fonction « *commencer*() », « *debutAcqui*() » modifie l'affichage pour n'y avoir que le décompte avant la fin de l'acquisition. Le retour vidéo n'est pas présent ici du fait que la caméra ne peut faire qu'une seule chose à la fois, par conséquent elle est occupée à acquérir les images pour les traitements à ce moment. Pour éviter tout problème de double utilisation de la caméra, elle est arrêtée via la fonction « *finCamera*() » et relancée lors du lancement de la fonction d'acquisition en python correspondante au temps choisi par l'utilisateur. Puis elle va lancer de compte à rebours indiquant la fin de l'acquisition avec la fonction « *attenteFin*() ». Celle-ci va simplement afficher un compte à rebours et à la fin va lancer la fonction « calculs() ». Cette fonction est une étape intermédiaire, car elle va modifier encore une fois l'affichage pour n'afficher que les éléments pour l'attente de la fin du calcul et lancer la fonction des calculs avec micmac : « *calculsMicmac*() ». Cette fonction va demander au main.py de lancer la fonction qui se trouve à « */calculs* » puis à la fin va récupérer le rapport d'erreur généré par la fonction calculs du mains.py (booléen pour rappel). JavaScript va donc regarder s'il y a une erreur avec *True* et afficher un message d'erreur si c'est le cas. Si le calcul s'est bien passé, alors le rapport affiche *False*, la dernière étape de l'application va donc s'afficher : un message de réussite avec la possibilité de télécharger et visualiser le nuage de points. Cette *div* est simplement réactivée et les autres cachées.

C'est grâce à la classe « *hide* » du css qu'il est possible de jouer avec l'affichage du html et le rendre un peu plus dynamique qu'avec un simple temple html sans JS. L'aperçu de la page est disponible en annexe 7.

En plus des actions principales, l'application possède des boutons sur la page d'accueil pour faire pivoter le robot. Un évènement est créé sur chacun des quatre boutons de direction et fait déplacer la caméra via le main.py en envoyant la fonction correspondante au mouvement choisi. Cette fonction gère aussi le fait que le robot soit en buté, dans ce cas la fonction python renvoie « no » et le bouton de la direction en buté est grisé. Le bouton se réactive dès que le robot n'est plus en buté.

L'option de vitesse de déplacement est aussi gérée par JS qui va prendre en compte la vitesse choisie et l'envoyer à python pour en modifier la vitesse de déplacement du robot lorsque l'on clique sur les flèches. Ici l'évènement est directement placé dans le html avec un *onchange*.

La dernière tache effectuée par le JS est d'écouter le bouton pour éteindre la raspberry pi. Celui-ci va envoyer l'ordre au main.py d'éteindre la raspberry pi. Le Js va aussi modifier l'affichage en grisant le bouton *Commencer* (s'il est visible) et de changer l'état de connexion de la Raspberry pi en « NO » rouge.

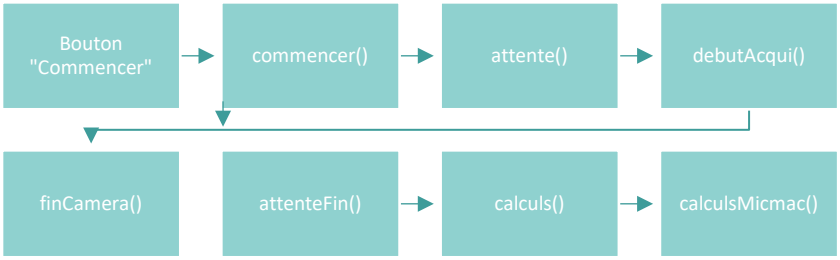



Figure 5 : Schéma explicatif des fonctions enclenchées par l'évènement du bouton « Commencer »

### VISUALISATEUR WEB POTREE

Potree est une solution de visualisation de nuages de points en ligne. Il est hébergé sur un dépôt en ligne privé, ici sur mon site web personnel chez OVH avec le nom de sous domaine *raspberry.marion-charpentier.fr*. Il se présente sous forme de dossier avec des librairies, le html de la page à afficher et les données 3D du nuage de points qui seront lus dans le html de visualisation.

Pour réaliser cela il m'a fallu installer PotreeConverter sur le serveur de calculs car c'est après les calculs micmac que le fichier .ply de sortie de micmac est converti et est envoyé sur le dépôt. Etant un exécutable, il a fallu que je compile l'application de PotreeConverter, il s'est avéré que deux fichiers C++ étaient erronés :

- Stuff.cpp : il faut remplacer le code lui ressemblant par cela :

```
#if BOOST_OS_WINDOWS
#include <Windows.h>
#elif BOOST_OS_LINUX
#include <linux/limits.h>
#elif BOOST_OS_MACOS
#elif BOOST_OS_BSD
#endif
```

- PotreeConverter.cpp : la variable *path* prend en compte le nom du binaire ce qui crée une erreur pour aller chercher le fichier Template pour la génération du html de visualisation. Il faut donc remplacer la variable *exePath* par son emplacement en dur. Dans le futur et pour une installation plus propre il faudrait modifier seulement la variable *path* pour que le changement de dossier soit reconnu automatiquement.

Moyennant l'ensemble des fonctions de ces fichiers, l'application est rendue simple d'emploi pour l'utilisateur et facilement modifiable par le grand découpage des fonctions.

## C – CREATION ET MONTAGE DU MATERIEL

Pour transporter tout le matériel l'idée retenue fut d'utiliser une petite voiture télécommandée (jouet). Comme les batteries étaient trop imposantes, il a fallu y ajouter une remorque (avec les moyens du bord pour créer la boule d'attelage). Cet ensemble nécessite trois piles AA 1,2V et d'une pile 9V pour la télécommande.

De plus les servo moteurs étant trop gourmands en énergie, il est nécessaire de séparer leur alimentation de celle de la Raspberry Pi. Pour les faire fonctionner il faut entre 4,8V et 5V, c'est pourquoi un bloc pile fut créé afin d'y héberger 4 piles AA de 1,2V. N'ayant pas de bloc neuf sous la main, celui-ci a été découpé d'une ancienne télécommande de micro-hélicoptère. De nouvelles soudures ont été réalisées accompagnés de nouveaux câbles dans le but de sécuriser l'alimentation. Des cosses ont été ajoutés à la sortie du bloc piles de sorte qu'il puisse être détachable du reste du montage Raspberry Pi – Servo moteur – Bloc piles. De plus une cosse a dû être crée pour s'enficher dans un des GPIO de la Raspberry Pi.

Le câblage nécessite de la précision, étant donné que les câbles doivent suivre une logique lors de leur branchement. Chaque GPIO à une fonction différente (terre, 5V, commande …), de ce fait il est nécessaire de respecter le schéma de notre branchement en annexe 5 et pour toute option supplémentaire ou changement, suivre le schéma en annexe 4.

Les images de ce montage et de l'équipement roulant sont disponibles en annexe 6.

## III – DEROULEMENT DE L'APPLICATION

Avant de charger dans le navigateur l'application, il faut vérifier que l'ordinateur de calculs, la raspberry pi, et le serveur Flask (client web par la même occasion) soient bien allumés et connectés par wifi au smartphone. Puis il faut lancer le serveur Flask en lançant la commande dans le terminal de la tablette « *python main.py* » à l'emplacement du fichier *main*. A ce moment le serveur se met en marche et permet une connexion au *localhost:5000*.

Lorsque l'utilisateur charge cette page, s'affiche :

- Le retour vidéo de la caméra
- Les quatre commandes pour déplacer la caméra ainsi que le choix de vitesse de déplacement de la caméra
- L'état connexion de la raspberry pi et de l'ordinateur de calculs
- Le choix du temps d'attente de 10 secondes avant l'acquisition
- Le choix du temps d'acquisition avec le bouton « Commencer »
- Le bouton permettant d'éteindre la raspberry pi « correctement » c'est-à-dire devoir arracher le câble d'alimentation.

L'utilisateur, choisi son angle de prise de vue et le temps d'acquisition puis clique sur « Commencer ».

Si le temps d'attente est resté coché, un compte à rebours de 10 secondes d'affiche, toujours avec le retour vidéo. Lorsque le compte à rebours est terminé ou s'il n'a pas été coché, l'application lance l'acquisition et affiche un compte à rebours du temps d'acquisition sélectionné à la première étape.

A la fin de l'acquisition, les calculs se lancent automatiquement et l'application en informe l'utilisateur en affichant un message et un symbole de chargement.

Une fois les calculs terminés il peut y avoir deux affichages différents :

- Les calculs ont échoué : cela peut être dû à une mauvaise prise de vue, alors s'affiche un message d'erreur et propose de recommencer avec un icone – lien vers la page d'accueil. Si l'utilisateur veut éteindre la raspberry pi, il en a la possibilité.

- Les calculs ont réussi : s'affiche alors le dernier volet de l'application avec un message de réussite, un bouton pour visualiser dans un autre onglet le nuage de point avec Potree. Ce bouton s'accompagne d'un second prévu pour télécharger le nuage de points au format *.ply*, ce fichier devra se voir retirer son extension *.xyz* pour être lu dans CloudCompare ou Meshlab par exemple.

Dans tous les cas, il y a la possibilité de recommencer en cliquant sur le bouton avec deux flèches en cercle. L'application va se réinitialiser et l'utilisateur pourra recommencer une acquisition.

Comme le serveur du visualisateur est hébergé chez OVH, il se peut que le nuage de points dans Potree ne soit pas celui précédemment créé, pour cela il faut supprimer les cookies générés par OVH sur le navigateur, pour plus d'informations il faut aller voir ce site.

Il faut prendre en compte que l'application n'enregistre aucun fichier de points, il est donc impératif que l'utilisateur le télécharge lorsque cela est proposé s'il veut le conserver.

# CONCLUSION, LIMITES ET AMELIORATIONS

Cette application a pour destinée à être utilisée occasionnellement afin de générer des objets en 3D. L'utilisateur de l'application n'a pas besoin d'avoir de grandes connaissances en photogrammétrie, il faut simplement que l'objet soit visible sur au moins trois images non floues et qu'il ne brille pas. De plus l'interface est simple d'utilisation et les erreurs sont gérées dans la mesure du possible. L'application est donc nommée au terme de ce projet *RaspiModel*, en référence à la Raspberry pi et au modèle 3D qui est généré. Celle-ci est du plus possible automatisée afin d'éviter un nombre trop important de clics par l'utilisateur.

De par la fragilité du matériel, cette application est dédiée aux personnes adultes et soigneuses car un choc important sur le capteur ou le robot pourrait endommager le matériel.

RaspiModel n'est pas prévue pour les utilisateurs professionnels étant donné que la précision et le géoréférencement ne sont pas gérés. Il serait bon dans le futur de proposer plusieurs modes Micmac fin de personnaliser un peu plus l'application pour les utilisateurs compétents en photogrammétrie.

Pour les utilisateurs novices ou pour ceux qui ne comprennent pas certaines choses dans l'application, il serait pratique de créer une forme de manuel utilisateur visible dans l'interface via un bouton **i** par exemple.

Actuellement, le fait de recommencer une acquisition pendant un cycle déjà en cours n'est pas pris en compte. Dans une éventuelle suite du projet certaines choses dont celle-ci seraient à ajouter. De plus les statuts de connexion sont mis à jour seulement lorsque la page est rechargée ou quand on clique sur le bouton *Recommencer*, il faudrait gérer le changement de statut pendant l'utilisation de l'application et non pas seulement d'au démarrage.

Pour le moment RaspiModel fonctionne en local. Lorsqu'il sera porté en ligne, un plus gros travail sur l'affichage responsive devra être réalisé. L'interface ne pose pas de problème en soit, hormis le retour vidéo qui est encore en taille fixe et qui pose des soucis lors de son utilisation sur smartphone.

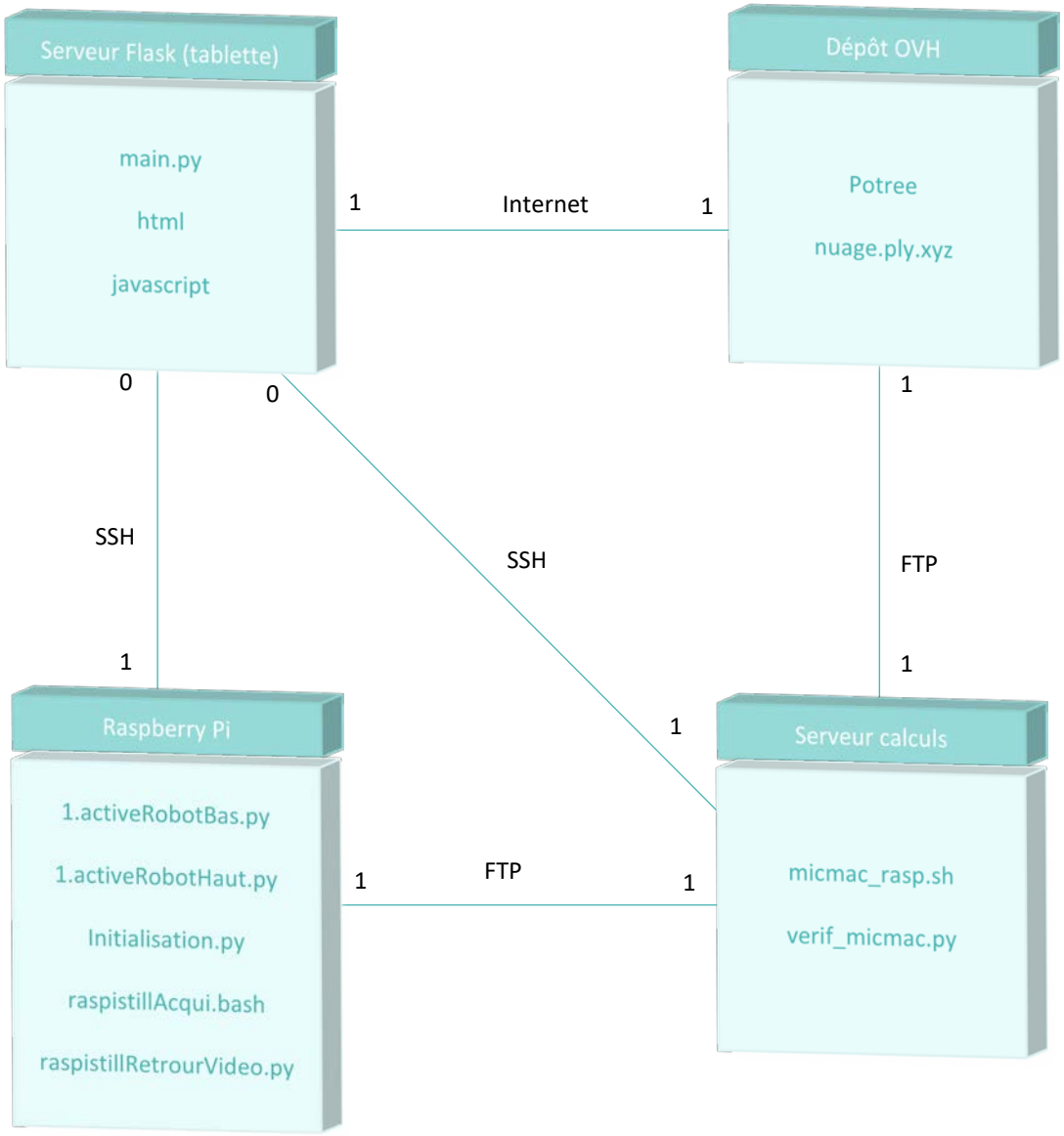
Pour conclure, ce projet a été une grande partie de mon année de master 1 Géomatique, j'ai pu découvrir les technologies que renferme les Raspberry Pi, apprendre plus en profondeur la puissance et les limites de Python. Ce projet met tient d'autant plus à cœur qu'il comporte une partie photogrammétrie que j'affectionne particulièrement. L'application nécessite plusieurs machines, mais cela reste accessible et peut-être reproduit dans le cadre privé pour l'auto apprentissage ou pour simplement se faire plaisir en bricolant.

# ANNEXES

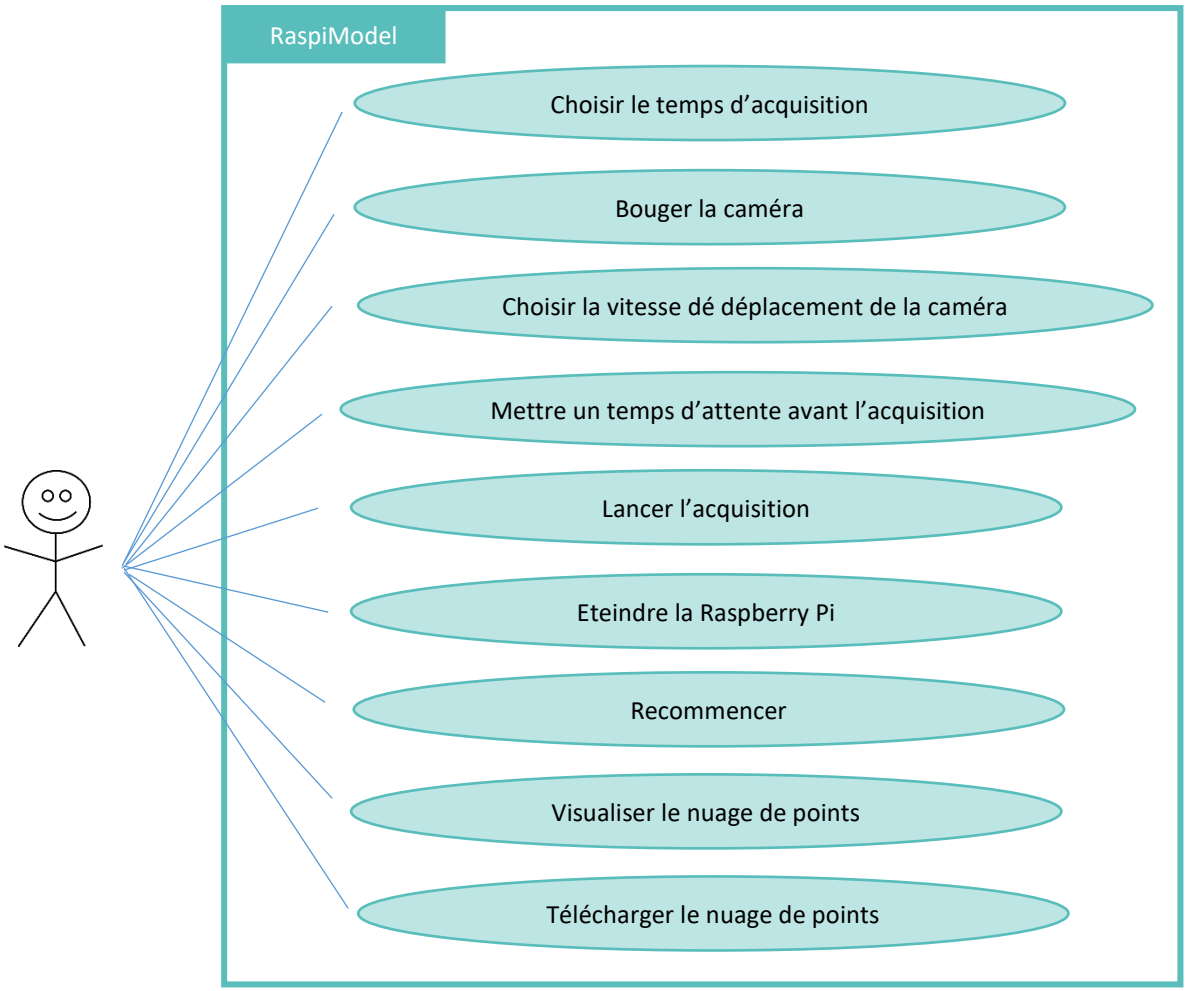
## ANNEXE 1 : CARACTERISTIQUES DE LA CAMERA

| | |
|---|---|
| **Capteur** | Omnivision 5647 – CMOS |
| **Focale** | Fixe de 3,6 mm |
| **Taille d'un pixel** | 1,4 µm |
| **Taille du capteur** | En µm 3673,6 x 2738,4 |
| | En mm 3,67 x 2,74 |
| **Taille image** | En pixel 2592 x 1944 |
| **Encodage** | 8 bits, RVB |
| **Filtre Bayer** | |

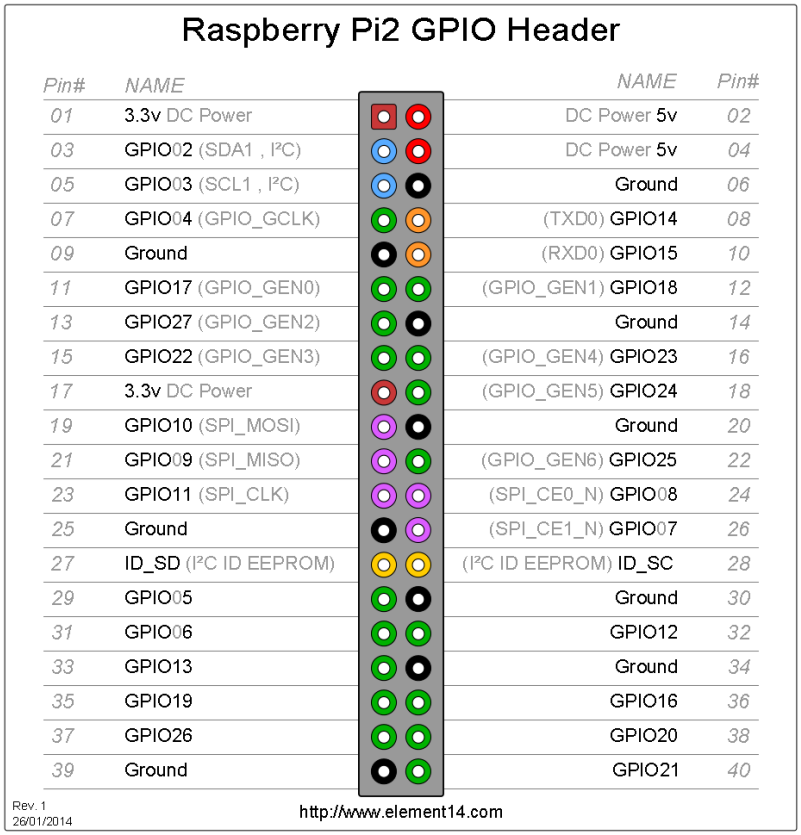
## ANNEXE 2 : DIGRAMME DE DEPLOIEMENT



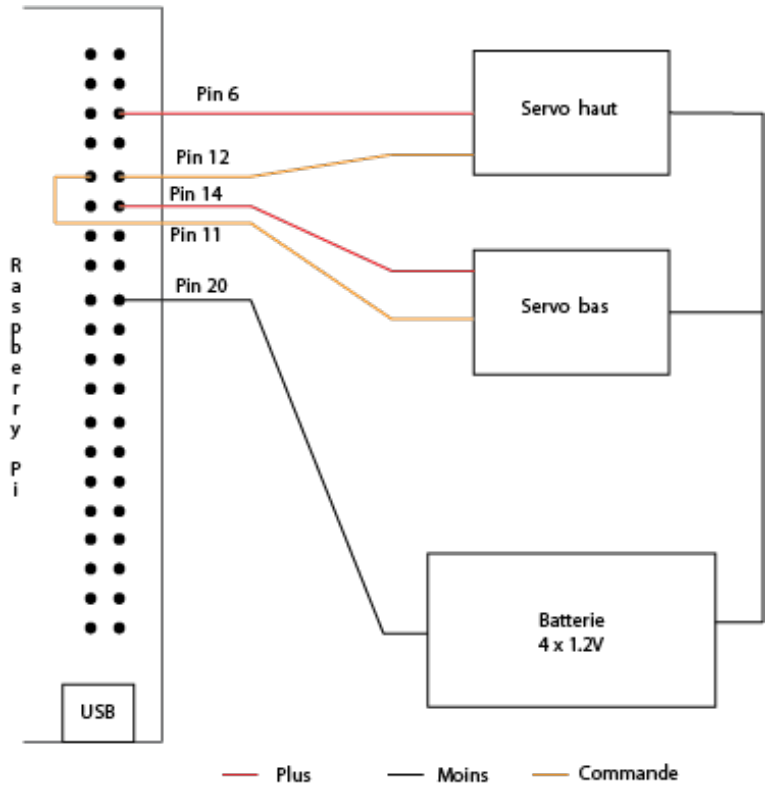
## ANNEXE 3 : DIAGRAMME D'UTILISATION



## ANNEXE 4 : SCHEMA DES TETES GPIO DE LA RASPBERRY PI 2



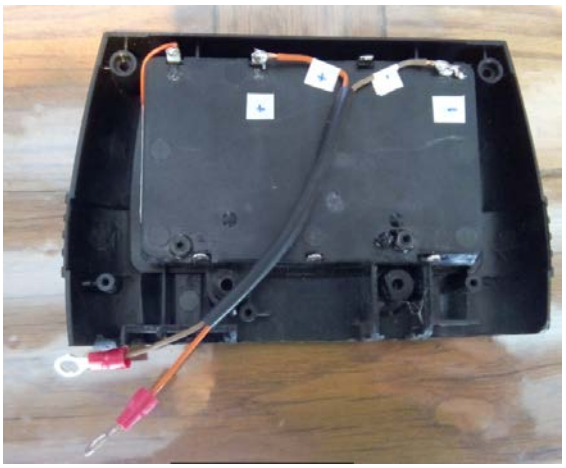

## ANNEXE 5 : SCHEMA DU CABLAGE DE LA RASPBERRY PI 2, SES ROBOTS ET LEUR BLOC PILES



## ANNEXE 6 : IMAGES DE LA VOITURE TELECOMMANDEE, DU BLOC PILES ET DU CABLAGE

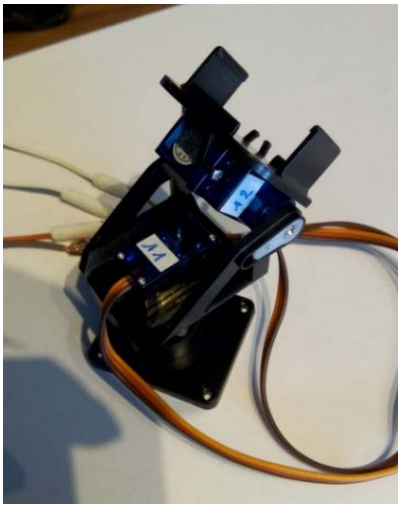
Voiture télécommandée

Remorque



Dessous du bloc piles avec nouvelles soudures

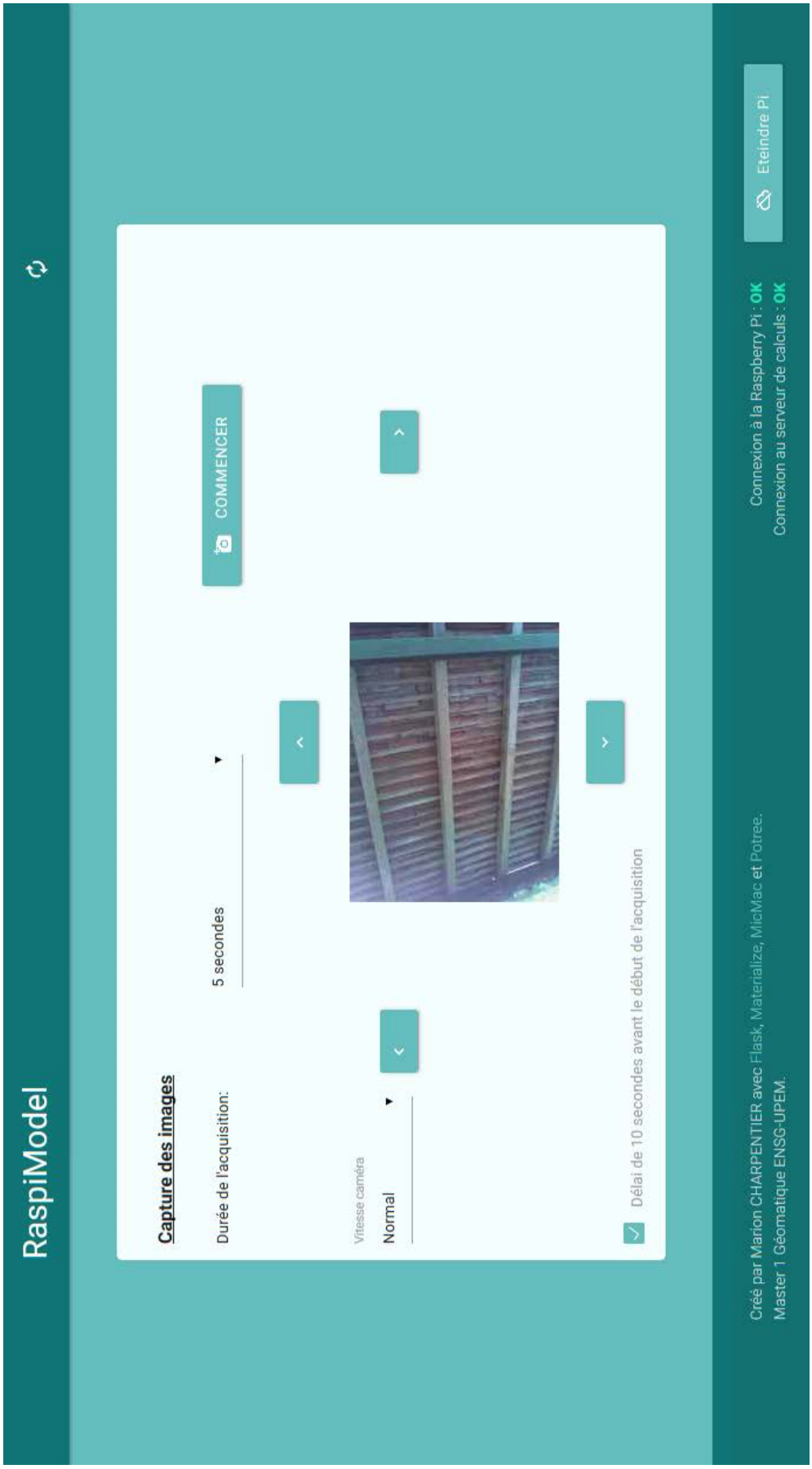

Dessus du bloc piles



Robot biaxial avec ses deux servo moteurs (pin 11 et 12)


Cosse incurvée

## ANNEXE 7 : VISUEL DE LA PAGE D'ACCUEIL DE L'INTERFACE



## SITOGRAPHIE COMPLEMENTAIRE

**Visionnés et utilisés (en plus des liens cités dans le rapport) :**

- Aides pour Flask : http://opentechschool.github.io/python-flask/core/form-submission.html
- Installation robot : http://www.toptechboy.com/raspberry-pi/raspberry-pi-lesson-28-controlling-a-servo-on-raspberry-pi-with-python/
- Installation robot (suite) : http://electroniqueamateur.blogspot.fr/2015/11/controler-un-servomoteur-en-python-avec.html

**Utiles pour une potentielle suite du projet :**

- Installer flask sur ovh https://gist.github.com/pyguerder/37ade154837e550646b0
- Installer un serveur apache avec wsgi pour flask : http://www.bogotobogo.com/python/Flask/Python_Flask_HelloWorld_App_with_Apache_WSGI_Ubuntu14.php
- Commandes de base Ubuntu : http://juliend.github.io/linux-cheatsheet/