Quentin FORESTIER

CFPT – Informatique

TPI 2018 – 2019

Pascal Bonvin

23/05/2019

RedBallBot

Documentation Technique

Résumé du rapport

Situation de départ

Le but du projet est de déplacer un robot afin que celui-ci se colle à une balle rouge. Il faudra en plus développer une télécommande sur une page WEB pour contrôler le robot. Le programme devra réaliser quelques tâches supplémentaires, telles que le calcul de distance entre le centre de l’image et le centre de la balle, l’affiche d’un cadre autour de la balle ou encore la possibilité de voir le travail du robot au travers d’une page web disponible par un connecteur TCP/IP.

Mise en œuvre

Pour cela, j’ai utilisé un serveur Flask afin de pouvoir avoir un accès TCP/IP. Le serveur renvoie une page html qui actualise le flux vidéo de la caméra dès que possible. Cette page html contient différents boutons et slider pour télécommander au mieux le robot.

Pour l’analyse d’image, j’ai décidé d’utiliser Scitkit-image et de traiter l’image sur un autre ordinateur. L’ordinateur envoie les informations de la balle au serveur, qui va ajouter un cadre vert autour de la balle grâce à la librairie PIL avec l’objet DrawImage.

La classe AlphaBot2 est disponible sur le site du fabricant, j’ai donc utilisé cette ressource pour créer ma classe Robot qui permet de faire bouger le robot.

Résultats

Le robot arrive à être télécommandé depuis une page WEB en mode manuel. Le mode automatique marche dans certaines conditions. S’il n’y a pas trop de contraste et que l’environnement n’est pas rouge, l’analyse marchera bien. Cependant, si l’environnement n’est pas adéquat, la détection risque de ne pas marcher à 100%.

Les tests que j’ai effectués ont été réalisés avec une balle rouge de 4cm de diamètre et la distance maximale de détection était de 41,5cm.

Table des matières

[1 Introduction 5](#_Toc9508976)

[2 Rappel du cahier des charges 6](#_Toc9508977)

[2.1 Objectifs 6](#_Toc9508978)

[2.2 Spécifications 6](#_Toc9508979)

[2.3 Points importants à réaliser 6](#_Toc9508980)

[2.4 Environnement 6](#_Toc9508981)

[2.5 Organisation 6](#_Toc9508982)

[2.6 Livrables 7](#_Toc9508983)

[3 Analyse de l’existant 7](#_Toc9508984)

[4 Analyse fonctionnelle 7](#_Toc9508985)

[4.1 Fonctionnalités 7](#_Toc9508986)

[4.1.1 Serveur Web 7](#_Toc9508987)

[4.1.2 Streaming vidéo 7](#_Toc9508988)

[4.1.3 Analyse d’image 8](#_Toc9508989)

[4.1.4 Déplacement du robot 8](#_Toc9508990)

[4.1.5 Manette de contrôle 8](#_Toc9508991)

[4.2 Interfaces 8](#_Toc9508992)

[4.2.1 Prévision 8](#_Toc9508993)

[4.2.2 Effectif 9](#_Toc9508994)

[4.2.3 Raison des changements 9](#_Toc9508995)

[5 Analyse organique 10](#_Toc9508996)

[5.1 Architecture du projet 10](#_Toc9508997)

[5.1.1 Générale 10](#_Toc9508998)

[5.1.2 Détaillée 11](#_Toc9508999)

[5.2 Architecture du code 11](#_Toc9509000)

[5.2.1 Arborescence des fichiers 11](#_Toc9509001)

[5.2.2 Diagrammes de classes 12](#_Toc9509002)

[5.3 Fonction principale 13](#_Toc9509003)

[5.3.1 Détection de cercles 13](#_Toc9509004)

[5.3.2 Détection de rouge 13](#_Toc9509005)

[5.3.3 Déplacement du robot 14](#_Toc9509006)

[5.3.4 Flux vidéo 14](#_Toc9509007)

[5.4 Dimensions temporelles 15](#_Toc9509008)

[5.4.1 Résultats 15](#_Toc9509009)

[5.4.2 Explications 15](#_Toc9509010)

[6 Problème rencontrés 15](#_Toc9509011)

[6.1 Problème matériel 15](#_Toc9509012)

[6.2 Problème d’environnement 16](#_Toc9509013)

[6.3 Problème de performances du Raspberry Pi 16](#_Toc9509014)

[7 Élaboration du projet et réflexion 17](#_Toc9509015)

[7.1 Personnalisation du robot 17](#_Toc9509016)

[7.2 Serveur WEB 17](#_Toc9509017)

[7.2.1 Apache 17](#_Toc9509018)

[7.2.2 Flask 17](#_Toc9509019)

[7.2.3 Séparation en 2 Serveur WEB 18](#_Toc9509020)

[7.2.4 Réunification des serveurs WEB 18](#_Toc9509021)

[7.3 Flux vidéo 18](#_Toc9509022)

[7.3.1 L’objet StreamingOutput 18](#_Toc9509023)

[7.3.2 Ajout d’un générateur Python 19](#_Toc9509024)

[7.4 Analyse d’image 19](#_Toc9509025)

[7.4.1 Choix de la librairie 19](#_Toc9509026)

[7.4.2 Détection de ronds 19](#_Toc9509027)

[7.4.3 Détection de rouge v1 20](#_Toc9509028)

[7.4.4 Détection de rouge v2 20](#_Toc9509029)

[7.4.5 Choix du meilleur rond rouge 21](#_Toc9509030)

[7.5 Déplacement du robot 21](#_Toc9509031)

[7.5.1 Recherche de balle 21](#_Toc9509032)

[7.5.2 Déplacement en direction de la balle 21](#_Toc9509033)

[7.6 La télécommande WEB 23](#_Toc9509034)

[8 Tests 23](#_Toc9509035)

[8.1 Conditions de tests 23](#_Toc9509036)

[8.2 Plan de tests 24](#_Toc9509037)

[8.3 Rapport des tests 25](#_Toc9509038)

[9 Conclusion 26](#_Toc9509039)

[9.1 Résultat obtenu 26](#_Toc9509040)

[9.2 Les 7 points techniques 26](#_Toc9509041)

[9.3 Retour sur la planification 26](#_Toc9509042)

[9.4 Améliorations possibles 27](#_Toc9509043)

[9.5 Bilan Personnel 27](#_Toc9509044)

[10 Planning 28](#_Toc9509045)

[10.1 Planning prévisionnel 28](#_Toc9509046)

[10.2 Planning effectif 28](#_Toc9509047)

[11 Glossaire 29](#_Toc9509048)

[12 Bibliographie 30](#_Toc9509049)

[12.1 Codes repris 30](#_Toc9509050)

[12.2 Sites utilisés 30](#_Toc9509051)

[12.3 Aides reçues 31](#_Toc9509052)

[13 Table des illustrations 31](#_Toc9509053)

[13.1 Figures 31](#_Toc9509054)

[13.2 Tableaux 31](#_Toc9509055)

[14 Annexe 32](#_Toc9509056)

# Introduction

Cette documentation permet d’obtenir des détails sur le projet « RedBallBot », réalisé dans le cadre du TPI (Travail Pratique Individuel).

L’objectif est de déplacer un robot pour que celui-ci se colle à une balle rouge. L’utilisateur pourra cependant prendre la main et le déplacer à l’aide de boutons disponibles sur une page Web. Le mode automatique permettra au robot d’analyser son environnement afin de trouver la balle rouge et de se déplacer vers elle.

# Rappel du cahier des charges

## Objectifs

Le but du projet est de réaliser une application Python / Web qui permet de télécommander un robot et de le voir chercher une balle rouge en utilisant uniquement sa caméra embarquée à travers une page / application Web.

## Spécifications

L’application permet de :

* Télécommander un AlphaBot2 au travers d’une page / application Web
* Obtenir le flux vidéo de la caméra sur une page Web
* Savoir où le robot a analysé la balle grâce à un cadre vert
* Connaitre la distance entre le centre de l’image et le centre de la balle
* Diriger automatiquement le robot vers la balle rouge
* Modifier la vitesse du robot

## Points importants à réaliser

* Le flux vidéo de la caméra est accessible par un connecteur TCP/IP
* L’analyseur d’images trace un cadre vert autour d’une balle rouge
* Les coordonnées du cadre vert et la distance par rapport au centre de l’image sont calculées
* Une classe permet au robot de se déplacer
* Un service WEB tcp (http) permet de visualiser le travail du robot
* Le robot est télécommandable par une page WEB (formulaire web + script python)
* Les dimensions temporelles du système sont expliquées (vitesse de traitement des images, vitesse de déplacement du robot, limites)

## Environnement

* PC sous Windows 10
* PyCharm 2018.3.5
* AlphaBot2 version Raspberry Pi
* Raspberry Pi modèle 3 B+ sous Raspbian 9.8
* Git (GitHub et GitKraken) et Google Drive
* Méthodologie agile avec GitHub Project
* Outil bureautique (Word, Excel)
* Logiciel pour créer des diagrammes ou maquettes (Pencil, Draw.io, UMLetino)
* Logiciel de correction orthographique (Antidote)

## Organisation

Élève :

* Quentin Forestier, [quentin.frstr@eduge.ch](mailto:quentin.frstr@eduge.ch)

Maitre d’apprentissage :

* Pascal Bonvin, [pascal.bonvin@edu.ge.ch](mailto:pascal.bonvin@edu.ge.ch)

Experts :

* Arnold Rullo, [Arnold.Rullo@etat.ge.ch](mailto:Arnold.Rullo@etat.ge.ch)
* Jean Sottas, [jean.sottas@ville-ge.ch](mailto:jean.sottas@ville-ge.ch)

## Livrables

Plusieurs documents seront disponibles :

* La documentation technique
* Le manuel utilisateur
* Le code source
* Un journal de bord
* Un répertoire GitHub[[1]](#footnote-1)

# Analyse de l’existant

Il existe déjà des robots recherchant des objets. Cependant, il y a très peu d’informations sur les AlphaBot2.

Les constructeurs d’AlphaBot2 ont, eux aussi, créé une page web permettant de contrôler le robot. Le code est disponible sur le site, mais il commence à être obsolète et très peu documenté.

# Analyse fonctionnelle

## Fonctionnalités

### Serveur Web

Un des points importants du projet est de pouvoir accéder au flux vidéo via une page web. Pour cela, il faut donc un serveur WEB capable d’afficher une page HTML selon une URL.

### Streaming vidéo

Le flux vidéo sera disponible par le biais d’un connecteur TCP/IP. La seule restriction est d’être connecté au même réseau que le robot. Un carré vert sera ajouté autour de la balle lorsque celle-ci sera analysée.

### Analyse d’image

L’analyse d’image se fera, en premier lieu, sur le robot. Cependant, il sera possible de lancer le script d’analyse d’image sur un ordinateur externe afin que le processus soit plus rapide.

### Déplacement du robot

Le robot a 2 modes de fonctionnement. Le mode manuel, qui permet à l’utilisateur de contrôler le robot par l’intermédiaire d’une page WEB. Le mode automatique, qui se dirige vers une balle rouge en fonction de l’image analysée, soit par le robot, soit par un ordinateur distant.

### Manette de contrôle

La manette de contrôle se présentera sous forme de page WEB contenant des entrées qui permettent à l’utilisateur de déplacer le robot, de changer la vitesse, ou de changer son mode de déplacement.

## Interfaces

### Prévision

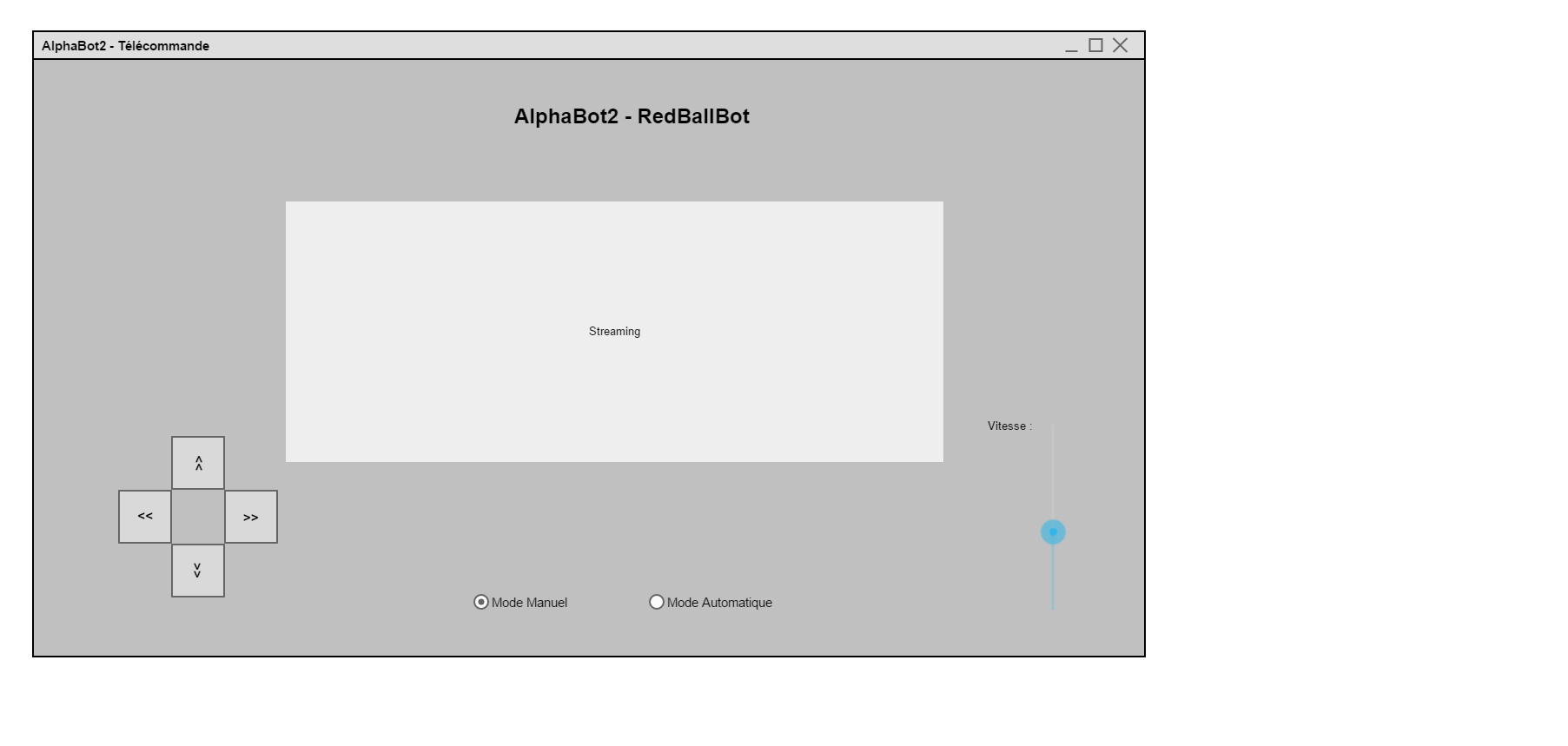


Figure 1 : Design prévisionnel de la page principale

### Effectif

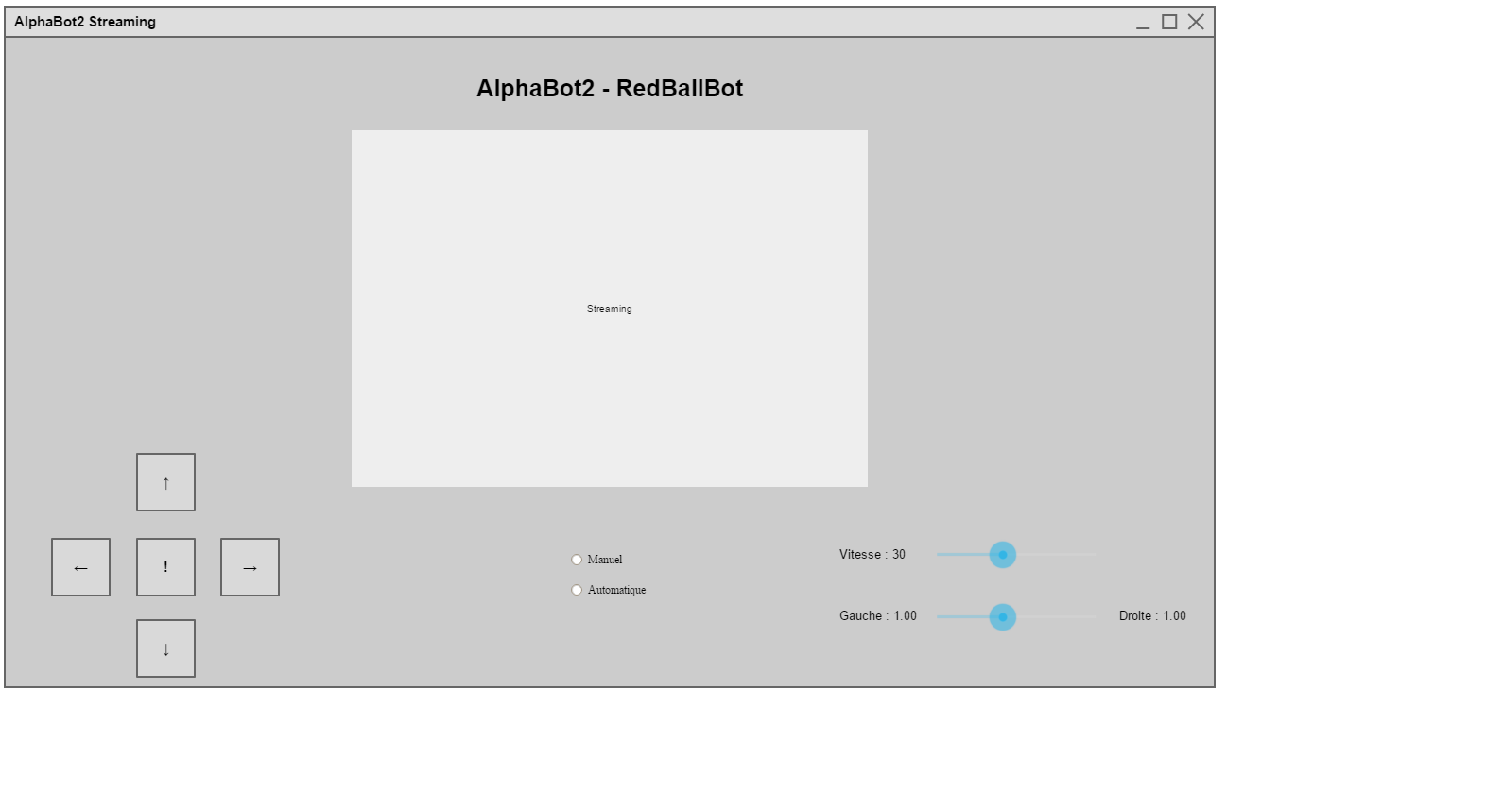


Figure 2 : Design effectif de la page principale

### Raison des changements

#### Les différences mécaniques

J’ai décidé d’ajouter un « Range Slider » pour ajuster la vitesse des moteurs afin que ceux-ci puissent aller à la même vitesse, malgré une différence du matériel.

#### Bouton stop

Je pensais tout d’abord arrêter le robot lorsqu’aucun bouton n’était appuyé, cependant je me suis fait la réflexion qu’il serait peut-être préférable de ne pas avoir à tenir un bouton enfoncé et que le robot continue d’avancer. Pour cela, j’ai donc ajouté un bouton stop sur la page web.

# Analyse organique

## Architecture du projet

### Générale

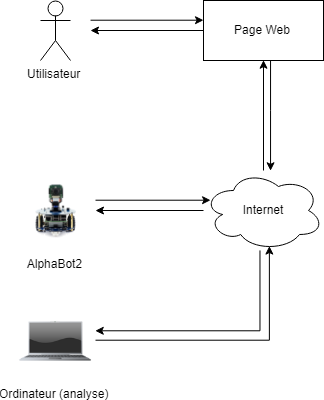


Figure 3 : Diagramme de projet général

### Détaillée



Figure 4 : Diagramme de projet détaillé

## Architecture du code

### Arborescence des fichiers

L’arborescence des fichiers de mon projet se présente comme ceci :

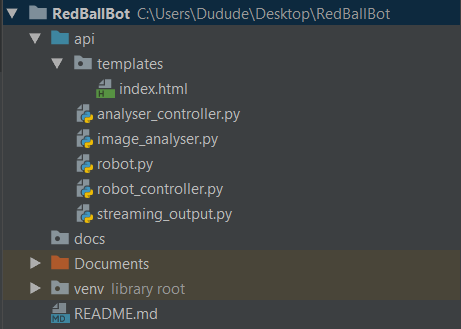


Figure 5 : Arborescence du projet

### C:\Users\Dudude\Desktop\RedBallBot\Documents\Diagrammes et Images\redballbot_diagram.pngDiagrammes de classes

Figure 6 : Diagramme de classes

## Fonction principale

### Détection de cercles

Cette fonction permet de trouver tous les ronds présents dans une image. Pour cela, j’ai utilisé Scikit-image, qui permet d’utiliser différentes méthodes, telles que :

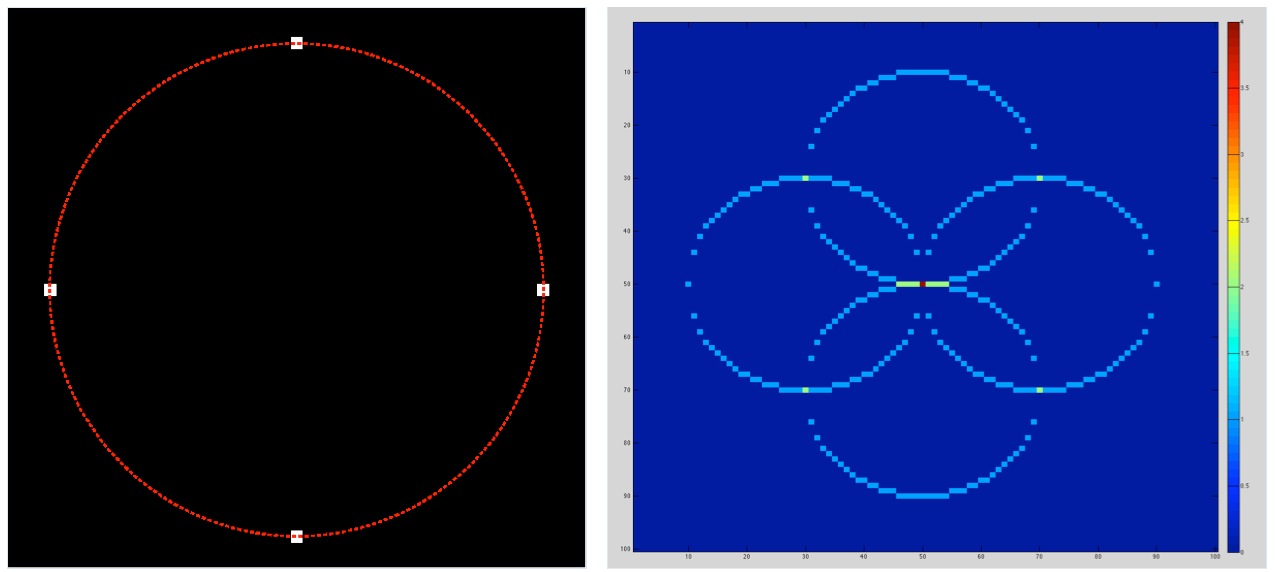
* Canny
  + Détection de bord grâce à un dérivé de la fonction Gaussienne
    - Paramètre sigma permet de réduire les effets du bruit
  + Bord potentiel réduit à 1 pixel et enlève les pixels trop loin des axes du dégradé
  + Vérifie avec un seuil d’hystérèse pour éviter les pixels limites
* Hough\_circle[[2]](#footnote-2)
  + À savoir : (x-a) ² + (y-b) ² = r²
    - (a ;b) : centre du rond
    - (x ;y) : cordonné pixel rond
    - r = rayon
  + « Dessine » un rond autour de chaque pixel (x ; y) de taille r
    - Si beaucoup d’intersections au même endroit, potentiel rond
* Hough\_circle\_peaks
  + Cherche meilleurs cercles dans les espaces de Hough
    - Espace de Hough : Transforme coordonné en vecteur forme polaire et crée une sinusoïde unique pour chaque valeur de normes et angles

Figure 7 : Méthode de Hough pour trouver un cercle

### Détection de rouge

* Détecte la rouge grâce à un masque de couleur appliqué sur l’image
* Les pixels correspondant au masque sont transformés en vert
* Les ronds obtenus lors de la détection de rond sont comparés avec le rouge trouvé afin de voir si un rond contient du rouge

### Déplacement du robot

#### Mode automatique

Le mode automatique consiste simplement à se déplacer en fonction des informations envoyées par l’analyseur d’image. Le robot adapte sa vitesse de rotation en fonction de la position en X de la balle.

Si aucune balle n’est présente sur l’image, l’analyseur renvoie « None », ce qui veut dire au robot qu’il doit tourner sur lui-même afin d’essayer de mettre la balle dans son champ de vision.

#### Mode manuel

Le mode automatique exécute simplement les informations envoyées par la page WEB.

Les boutons de déplacement sur la page WEB appellent une App Route Flask grâce à un objet XMLHttpRequest qui permet d’appeler en arrière-plan une URL.

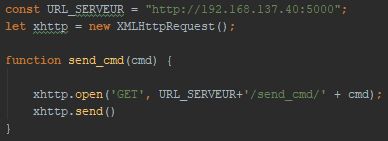


Figure 8 : Utilisation de XMLHttpRequest

### Flux vidéo

Le flux vidéo est récupéré grâce à un générateur python qui s’exécute lorsque le buffer de la caméra est disponible. Il demande la nouvelle image au serveur et, si celle-ci est disponible, répond.

#### Cadre vert

L’ajout du cadre vert se fait directement sur le flux, lorsque celui-ci est envoyé à la page WEB. L’image du buffer, en binaire, est transformé en image PIL, et il est alors possible de dessiner sur l’image. Le carré est calculé à partir du centre X, Y de la balle et son rayon.

Par la suite, j’ai trouvé une librairie, pdoc3, permettant d’extraire la documentation de mon code en fichier html. Pour cette raison, la documentation des classes de mon programme se trouve ici :

* [Robot](../docs/api/robot.html)
* [AnalyserController](../docs/api/analyser_controller.html)
* [ImageAnalyser](../docs/api/image_analyser.html)
* [StreamingOutput](../docs/api/streaming_output.html)

## Dimensions temporelles

J’ai pu apercevoir que le temps d’analyse d’image changeait énormément en fonction de la puissance de la machine. C’est pourquoi j’ai décidé de mesurer les différences de performances entre une analyse sur mon ordinateur portable et sur le Raspberry Pi.

### Résultats

Les résultats qui suivent comprennent la récupération de l’image depuis le serveur, l’analyse de l’image et l’envoi des nouvelles informations.

Ordinateur potable :

* ~ 0,2 seconde de traitement pour chaque image
* Distance minimum à 13cm pour une balle de 4cm de diamètres
* Distance maximum à 41,5cm pour une balle de 4cm de diamètres

Raspberry Pi :

* ~ 1,2 secondes de traitement pour chaque image
* Distance minimum à 13cm pour une balle de 4cm de diamètre
* Distance maximum à 41,5cm pour une balle de 4cm de diamètres

### Explications

La différence du temps de traitement s’explique facilement en raison de la puissance des processeurs. Le Raspberry Pi a un processeur de quatre cœurs à 1,4Ghz contre un processeur à 6 cœurs à 2,20Ghz. On peut observer que le Raspberry Pi est 12x moins rapide que le portable.

# Problème rencontrés

## Problème matériel

Au cours de l’élaboration du projet, j’ai rencontré un problème purement matériel. En effet, les moteurs des robots n’étant pas toujours parfaits, il est possible qu’il y ait des différences. Dans le cadre de mon projet, cela handicape le déplacement jusqu’à la balle car un des 2 moteurs va plus vite que l’autre. Pour pallier à ce problème, j’ai ajouté un « Range Slider » qui permet à l’utilisateur de réguler cette différence.

De plus, lorsque le robot tourne, il n’est pas toujours égal. Ceci est très perturbant lorsque le robot doit détecter l’image et qu’il accélère juste avant la balle rouge. Se faisant, l’analyse d’image manque la balle et le robot repart pour un nouveau tour.

## Problème d’environnement

Le plus gros problème que j’ai eu est survenu au niveau de l’environnement. Il m’a été difficile d’analyser la balle rouge dans des conditions où la luminosité n’était pas bonne, que la balle soit posée sur un fond qui tire vers le rouge ou encore que la caméra soit à contre-jour.

J’ai donc essayé au mieux de « limiter les dégâts » de l’environnement en augmentant la saturation de l’image afin que le rouge ressorte mieux, ce faisant, la balle est aussi plus contrastée et donc plus facile à détecter. Cette manière de faire n’est clairement pas parfaite étant donné que l’augmentation de la saturation augmente aussi pour l’entourage. Il est donc très difficile de trouver la balle rouge sur un plancher.

## Problème de performances du Raspberry Pi

J’ai vite remarqué que le Raspberry mettait trop de temps à analyser une image (1,2 secondes). J’ai donc opté pour que la détection se fasse sur un autre ordinateur plus puissant. Cette méthode m’a permis d’améliorer la réactivité du robot de 6fois car l’ordinateur peut analyser l’image en ~0,2secondes.

# Élaboration du projet et réflexion

## Personnalisation du robot

Afin de réussir au mieux le travail, j’ai tout d’abord décidé de modifier le robot afin que la caméra soit à l’avant. De plus, le robot ne pouvant pas être alimenté avec les piles prévues à cette effet, j’ai ajouté une pièce à l’arrière du robot afin que je puisse brancher une batterie.[[3]](#footnote-3)

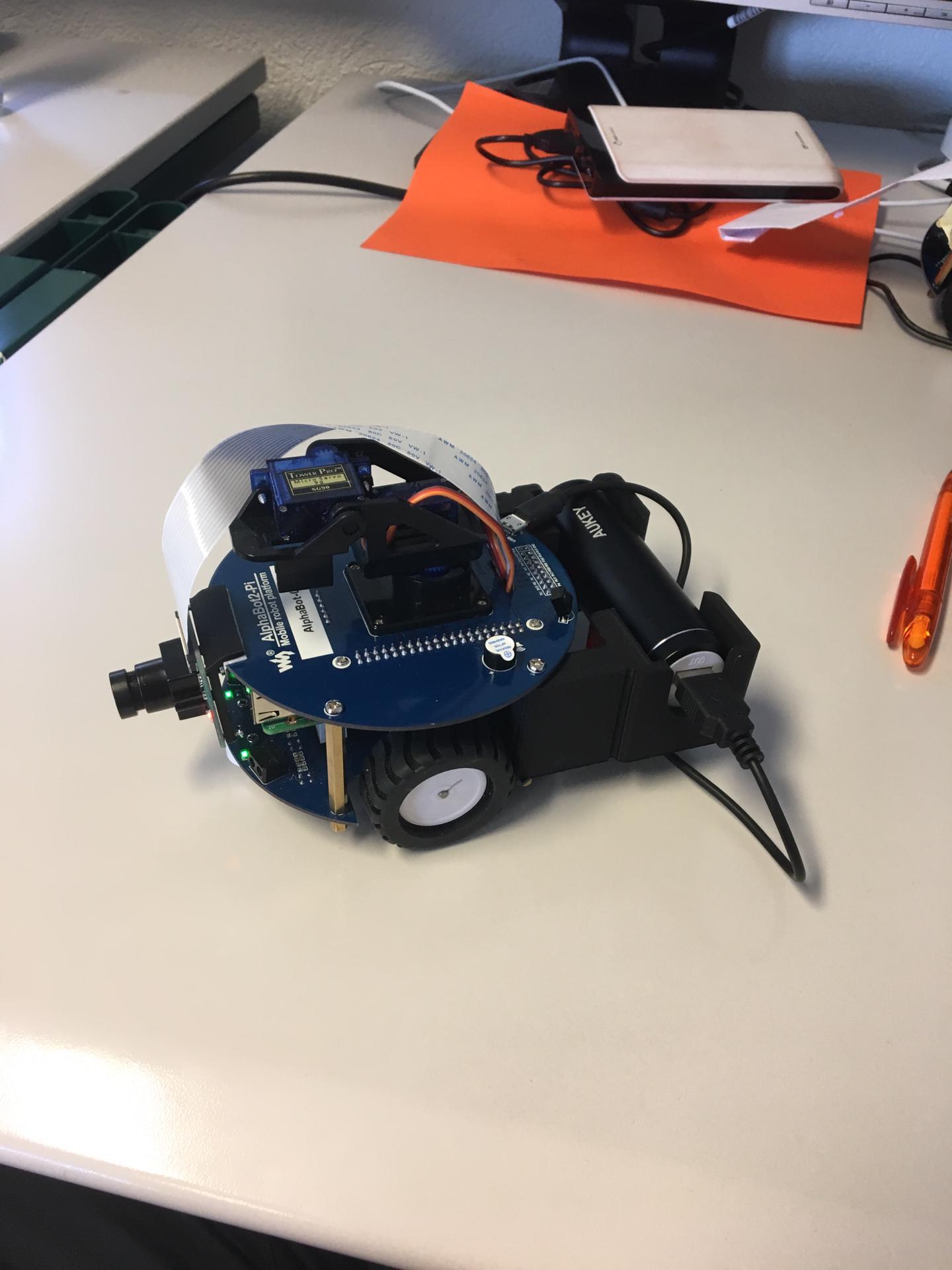


Figure 9 : Personnalisation de l'AlphaBot2

## Serveur WEB

Tout d’abord, il a fallu que je prenne connaissance du cahier des charges. Suite à cela, je me suis documenté sur les différentes possibilités de serveur web.

### Apache

Tout d’abord, mon but était d’utiliser Apache et de combiner du PHP et du Python. J’ai vite compris qu’il était compliqué d’avoir accès à du script Python depuis PHP et j’ai donc décidé de changer de méthode.

### Flask

Après quelques recherches et demandes à certains professeurs, j’en suis arrivé à la conclusion que Flask pouvait être une bonne alternative. Il n’est pas recommandé d’utiliser Flask en production, mais je me suis dit que pour de « simple » déplacement de robot, il serait suffisant.

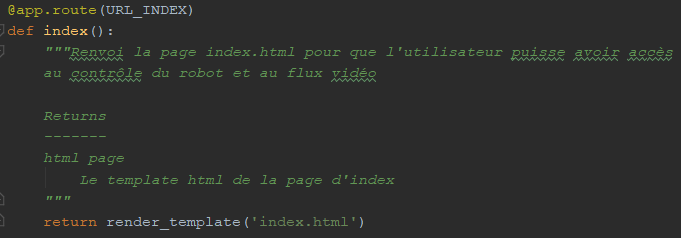
J’ai donc commencé à implémenter Flask et comprendre son utilisation, qui est plutôt facile. Il suffit d’ajouter une décoration avec l’URL souhaité en paramètre, nommée « App Route » au-dessus d’une fonction Python. Cette décoration a pour but de pouvoir lancer une fonction Python lorsqu’un navigateur appelle l’URL de l’App Route.

Figure 10 : Décoration Flask

### Séparation en 2 Serveur WEB

Lors d’un atelier, j’avais déjà eu l’occasion d’utiliser Flask et l’AlphaBot2, et j’avais essayé de changer l’angle dans la caméra, par le biais des servos moteurs, en appelant une URL. Cependant, j’avais eu le souci, qu’après une série d’appel d’URL trop rapide, le Raspberry PI redémarrait complétement. C’est pour cette raison, qu’à la base, j’avais opté pour séparer les tâches « Streaming » et « Contrôle du robot », me posant la question si les crashs allaient recommencer.

### Réunification des serveurs WEB

Étant donné que je n’utilisais pas les servos moteurs dans ce projet, j’ai par la suite décidé de les réunir afin de voir si des crashs allaient survenir. Fort heureusement, aucun crash n’est arrivé depuis et j’en ai donc déduit que c’était la cohabitation de Flask et de l’utilisation des servos moteurs qui ne fonctionnait pas.

## Flux vidéo

Comme l’un des points de mon travail était de pouvoir avoir accès au flux vidéo par le biais d’un connecteur TCP/IP, j’ai dû rechercher un moyen de streamer depuis le Raspberry Pi. J’ai notamment trouvé quelques librairies qui permettent d’envoyer le flux vidéo, tel que Motion, ou encore envoyer le flux grâce à VLC. Cependant, avec l’envoi du flux qui n’était pas directement dans mon script Python, il m’était difficile de rajouter le carré vert, ou encore d’avoir accès facilement à l’image afin que celle-ci soit analysée.

### L’objet StreamingOutput

J’ai trouvé sur internet un objet permettant de faciliter le streaming depuis le Raspberry. L’objet réceptionne les données de la caméra et, grâce à l’objet « Condition » de la librairie « Threading », permet de faire de l’asynchrones. L’objet permet simplement d’envoyer une sorte de notification au client lorsqu’une nouvelle image est disponible.

### Ajout d’un générateur Python

Le flux vidéo fonctionne déjà, cependant, il ne permet pas d’avoir un nombre d’images par seconde suffisant. J’ai donc essayé de combiner une autre manière de streamer le code que j’avais et pour cela, j’ai donc rajouté un objet générateur. Il permet qu’une fonction renvoie une partie de code à exécuter à la prochaine itération, de ne pas surcharger le processeur du Raspberry Pi et donc qu’il n’envoie que le nécessaire.

## Analyse d’image

Lors de ma réflexion par rapport au cahier des charges, j’ai pensé qu’analyser l’image pourrait être fait sur une machine distante. J’ai donc imaginé ma solution en m’assurant qu’il était possible d’analyser depuis une autre machine et depuis le Raspberry PI.

### Choix de la librairie

Cette partie du travail étant un des plus gros points du travail, il a fallu faire les bons choix. Je me suis donc documenté sur les différents possibilités d’analyse d’images, et 2 choix sont ressortis :

* OpenCV
* Scikit-Image

La première étape a été de voir les différences entre OpenCV et Scikit-image. J’ai donc trouvé qu’OpenCV était plus rapide à l’exécution et qu’il analysait le flux vidéo en temps réel, mais qu’il était plus complexe à comprendre et utiliser. Scikit-Image a l’avantage d’être plus facile à comprendre. Cependant, Scikit-image est plutôt axé sur l’analyse d’images et non de flux vidéo. Scikit-Image est apparemment plus puissant dans l’analyse d’images et donc plus juste, car il est basé sur du « Machin Learning ».

Avec mes connaissances en mathématiques et en physiques, j’avais l’impression que j’allais utiliser OpenCV tel une « Black Box » et je n’ai pas voulu continuer sur ce chemin. Scitkit-Image m’a demandé beaucoup de temps de compréhension, cependant j’ai pu comprendre et appliquer au mieux les méthodes.

### Détection de ronds

J’avais donc vu que Scikit-image avait la possibilité de trouver des ronds dans une image et j’ai tout d’abord repris le code, comme dans le tutoriel qui est disponible ici.

Il y a 3 méthodes importantes à retenir :

* Canny / Détection de bord
* Hough\_circle
* Hough\_circle\_peaks

Ces 3 méthodes ont des paramètres qui peuvent préciser l’analyse et ce fut un des points les plus difficiles à comprendre. Par exemple, pour la méthode Canny qui utilise un dérivé de la fonction Gaussienne, il est possible de modifier le sigma et donc de réduire ou d’augmenter les effets du bruit de l’image. [[4]](#footnote-4)

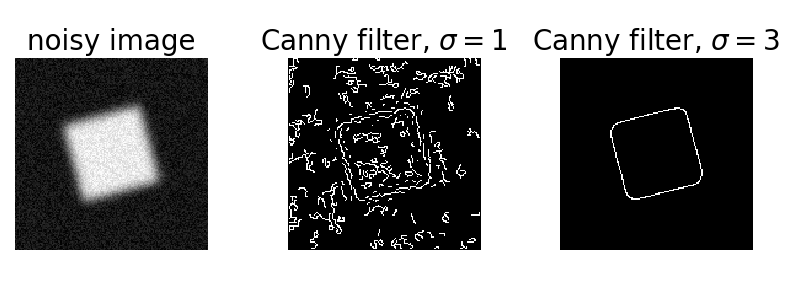


Figure 11 : Exemple de la fonction Canny

### Détection de rouge v1

Pour la détection de rouge, j’ai choisi de vérifier si la moyenne des couleurs dans un rond est rouge. Plus simplement, je regarde chaque pixel de chaque rond, je fais une moyenne de la couleur pour chaque rond et je regarde si celui-ci est rouge avec une condition que j’ai inventée en observant les valeurs que je considère comme rouge d’une palette de couleurs.

Je n’ai pas remarqué tout de suite, mais cette méthode ne marchait pas très bien et dépendait énormément de la luminosité. Pour mettre des chiffres sur ces propos, en cas de luminosité adéquate, la détection marchait 8 fois sur dix, alors que dans des conditions moins bonnes, l’analyse de rouge ne marchait pas.

Le résultat n’étant pas concluant, j’ai décidé de modifier la manière de détecter le rouge.

### Détection de rouge v2

Pour cette 2ème méthode, j’ai pensé à analyser l’image pour trouver du rouge, sans prendre en compte les ronds. Pour cela, j’ai appliqué un masque de couleur à l’image, ce qui veut dire que tous les pixels correspondant au masque seront transformés en vert « pure ».

J’ai ensuite vérifié que les ronds analysés contiennent au moins une cinquantaine de pixels rouges, devenus verts.

### Choix du meilleur rond rouge

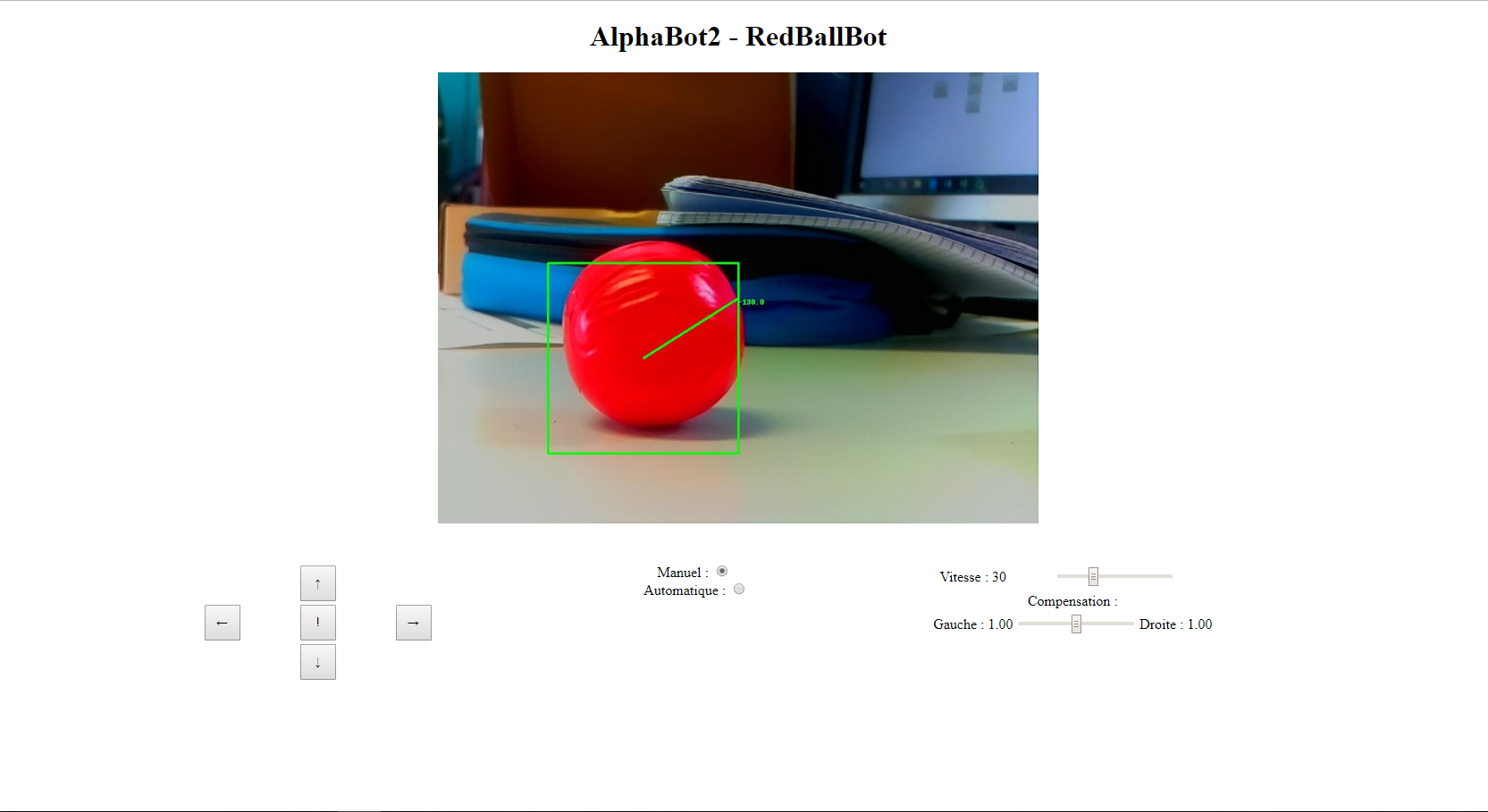
Cette dernière étape de l’analyse est la plus simple. Je choisis ensuite le rond avec le plus grand rayon, ce qui correspond la plupart du temps au rond recherché.

Figure 12 : Flux vidéo avec balle trouvée

## Déplacement du robot

L’AlphaBot2 ayant pour moteur des brush motors sans capteur, il est difficile de savoir précisément la distance parcourue.

Les servos de la caméra qui sont, quant à eux, des step motors, il est possible de les régler à une position définie.

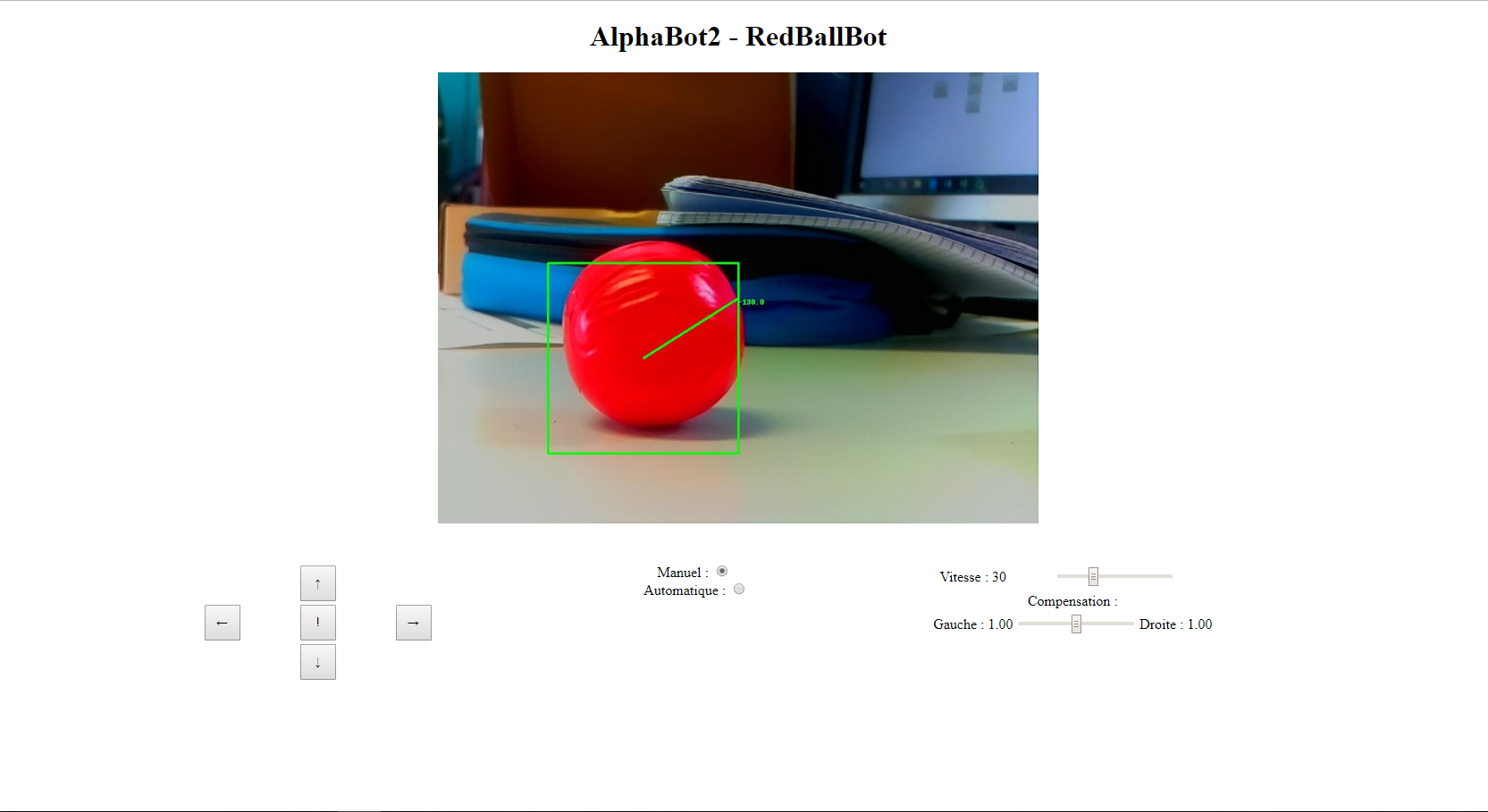
Cependant, les servos de la caméra permettent seulement une rotation de 180°. N’étant pas mentionné dans le cahier des charges que la balle doit absolument se trouver devant le robot, j’ai opté pour la méthode qui utilise seulement les roues du robot.

### Recherche de balle

Le principe est simple, car le robot ne fait que de tourner sur lui-même. Il y a un problème avec cette méthode quand la vitesse est trop élevée, car le robot risque de manquer la balle.

### Déplacement en direction de la balle

Tout d’abord, j’ai délimité l’image en 3 parties. La première partie de la gauche de l’image jusqu’à son centre, la deuxième partie est au centre alors que la dernière est à droite.

J’ai décidé d’ajouter une marge à la partie centrale, sinon il serait quasiment impossible d’être pile en face de la balle. Les parties latérales font environ 300 pixels de large, alors que la zone centrale en fait 160.

Droite en

Gauche

Avancer

Figure 13 : Séparation de l'image pour le déplacement

Cependant, le fait que le robot fasse des à-coups ne favorise pas le bon fonctionnement de l’analyse d’images, car cette dernière risque d’être floue.

J’ai donc opté pour une seconde méthode qui consiste à ajuster la vitesse en fonction de la position X de la balle.

Cette méthode marche plutôt bien, avec quelques petits soucis de temps en temps, mais qui sont liés au matériel qui n’est pas identique.

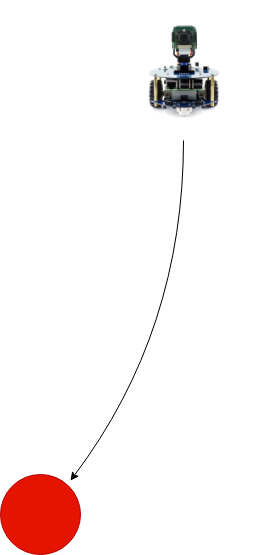


Figure 15 : Nouveau mode de déplacement

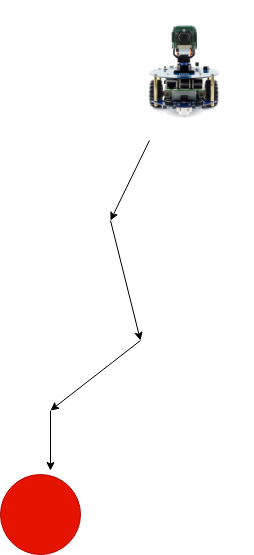


Figure 15 : Ancien mode de déplacement

## La télécommande WEB

Mon cahier des charges mentionnait aussi la possibilité de télécommander l’AlphaBot2. J’ai donc créé une simple page WEB, comme sur la maquette disponible dans le document. Comme il me suffisait d’appeler une URL afin d’envoyer une commande au robot, j’ai trouvé un objet JavaScript qui permet de faire des appels en arrière-plan. Cet objet se nomme XMLHttpRequest.

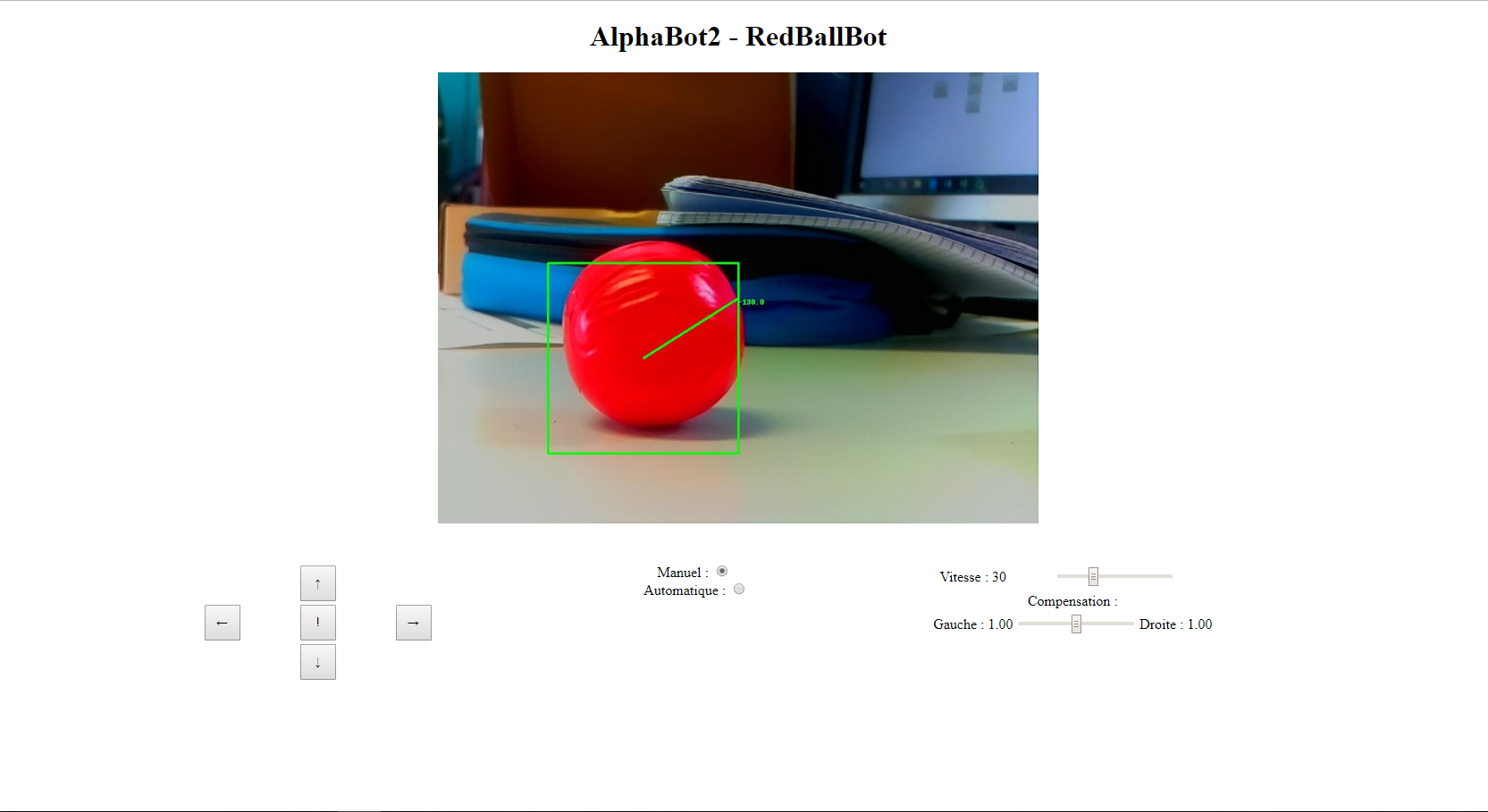


Figure 16 : Télécommande du robot

# Tests

## Conditions de tests

Tous les tests ont été effectués sur une table blanche dans la salle de classe. Toutes les lumières étaient allumées. Le robot se situait dans un endroit sans reflet du soleil.

Le Raspberry Pi était connecté au réseau de mon ordinateur portable.

Les tests ont été faits avec une balle de 4cm de diamètre et à une distance de ~35cm.

## Plan de tests



Tableau 1 : Plan de tests

## Rapport des tests



Tableau 2 : Rapport de tests

# Conclusion

## Résultat obtenu

Le cahier des charges a été rempli entièrement. Cependant, des améliorations peuvent être effectuées, telles qu’une meilleure analyse d’images (le carré vert s’affiche quelques fois en décalé par rapport à la balle).

## Les 7 points techniques

* Le flux vidéo de la caméra est accessible par un connecteur TCP/IP
  + Il est possible d’accéder au flux en entrant l’adresse IP du Raspberry Pi sur un navigateur WEB. Il faut juste être connecté au même réseau afin d’avoir accès à l’IP
* L’analyseur d’images trace un cadre vert autour d’une balle rouge
  + Un cadre vert s’affiche autour de la balle directement sur le flux vidéo. Cependant, il se peut que le cadre n’entoure pas parfaitement la balle car un souci de luminosité peut perturber l’analyse.
* Les coordonnées du cadre vert et la distance par rapport au centre de l’image est calculé
  + J’affiche une ligne sur le flux entre le point central de l’image et le centre du rond
* Une classe permet au robot de se déplacer
  + Le serveur Flask a une classe Robot. Cette classe contient des fonctions qui permettent au robot de se déplacer
* Un service WEB tcp (http) permet de visualiser le travail du robot
  + On peut voir l’avancée du robot depuis le flux vidéo grâce au cadre vert qui se met à jour rapidement
* Le robot est télécommandable par une page web (formulaire web + script python)
  + Sur la page d’index disponible à l’adresse IP du serveur, des boutons permettent de faire exécuter des commandes au robot.
* Les dimensions temporelles du système sont expliquées (vitesse de traitement des images, vitesse de déplacement du robot, limites)
  + Voir point [Dimensions temporelles](#_Dimensions_temporelles)

## Retour sur la planification

On peut voir sur le planning prévisionnel que j’avais prévu de programmer le robot après l’analyse d’images. Cependant, je me suis vite aperçu qu’en conditions « réelles », la différence de luminosité, d’environnement et de contraste entravaient énormément l’analyse. C’est pourquoi j’ai décidé de programmer le robot rapidement afin de pouvoir faire les tests directement dans des conditions moins parfaites.

Une autre différence assez marquante est le temps pour la création de la télécommande depuis le site WEB. N’ayant encore que peu d’expérience avec l’union de Python avec un serveur, je ne savais pas comment envoyer un ordre depuis un bouton HTML à une fonction Python. J’ai eu de la chance de trouver rapidement ce que je cherchais, c’est-à-dire un objet qui peut appeler des URLs en arrière-plan, afin de ne pas embêter l’utilisateur et le temps de réalisation a donc considérablement été réduit.

En règle générale, j’ai réussi à avoir de l’avance sur la partie programmation. Une avance qui a été compensée par le retard que j’ai pu avoir sur la documentation tout au long du travail.

## Améliorations possibles

Pour ce projet, il existe un grand nombre d’améliorations. En voici quelques exemples :

* Un meilleur filtre de rouge
* Une analyse des cercles plus précise
* Un déplacement du robot plus rapide
* Un design plus complet pour la page WEB afin que l’utilisateur soit aidé
  + Je pense notamment à une désactivation des boutons lorsque le robot est en mode automatique
* Le robot puisse détecter à plus que 41,5cm la balle de 4cm
* Le robot puisse vraiment se coller à la balle
  + Ici, le robot s’arrête quand le rayon de la balle est plus grand que 190pixels.
* L’utilisation des servos moteurs avec un laser monté dessus et qui pointe la balle
* Un gestion « automatique » des analyseurs
  + Lorsqu’aucun analyseur ne détecte de balle pour le robot, il utilise un analyseur en local
* L’utilisation des capteurs pour empêcher le robot de foncer dans un mur ou de tomber d’une table.

## Bilan Personnel

Je suis très content de l’avancée du projet pendant ces 11 jours. Mon plus grand regret a été de devoir arrêter d’améliorer le programme car la fin du TPI arrivait. Il faut dire qu’au début, il m’était difficile d’arrêter de programmer pour faire de la documentation étant donné que voir le robot atteindre finalement son but était très motivant. Ce travail a été pour moi bénéfique pour la gestion du temps et pour l’élaboration complète d’une documentation.

De plus, je suis content d’avoir pu utiliser les compétences acquises durant ma formation. Cela me montre les progrès que j’ai faits depuis mon arrivé dans l’école jusqu’à aujourd’hui.

# Planning



Tableau 3 : Planning prévisionnel

# Glossaire

A

asynchrones

Désigne le caractère de ce qui ne se passe pas à la même vitesse, dans le temps ou dans la vitesse., 19

B

brush motors

Moteur éléctrique à commutation interne., 22

F

Flask

Framework open-source de développement web en Python., 15

fonction Gaussienne

Fonction en exponentielle de l'opposé du carré de l'abscisse. Elle a une forme caractéristique de courbe en cloche., 14

G

générateur python

Permet de créer plus facilement des itérateurs., 15

M

Machin Learning

Intelligence artificielle qui se fonde sur des approches statistiques, 20

R

Range Slider

Élement de contrôle graphique permettant d'ajuster une valeur grâce à un indicateur, 10

S

sigma

Représente l'écart type de la fonction Gaussienne, 14

step motors

Permet de transformer une impulsion électrique en un mouvement angulaire, 22

T

tcp

Protocole de contrôle de transmissions, 6

TCP/IP

Protocoles utilisés pour le transfert de données sur Internet, 2

X

XMLHttpRequest

Objet JavaScript développé par Microsoft. Il permet de récupérer facilement des données via HTTP, 15

# Bibliographie

## Codes repris

* Fonction de détection de cercles prise et adaptée du site :  
  <https://scikit-image.org/docs/dev/auto_examples/edges/plot_circular_elliptical_hough_transform.html#sphx-glr-auto-examples-edges-plot-circular-elliptical-hough-transform-py>
* Classe Robot prise et modifiée du site :  
  <https://www.waveshare.com/wiki/File:AlphaBot2-Demo.7z>
* Classe StreamingOutput prise et adaptée du site :  
  <https://randomnerdtutorials.com/video-streaming-with-raspberry-pi-camera/>
* Utilisation des générateurs Python repris de :  
  <https://blog.miguelgrinberg.com/post/video-streaming-with-flask>

## Sites utilisés

Serveur WEB Flask :

<http://flask.pocoo.org/docs/1.0/quickstart/>

Vidéo Stream :

<https://blog.miguelgrinberg.com/post/video-streaming-with-flask>

<https://randomnerdtutorials.com/video-streaming-with-raspberry-pi-camera/>

<https://www.ntfs.com/jpeg-signature-format.htm>

<http://sametmax.com/comment-utiliser-yield-et-les-generateurs-en-python/>

<https://docs.python.org/3/library/threading.html>

Détection de rond :

<https://scikit-image.org/docs/dev/auto_examples/edges/plot_canny.html>

<https://en.wikipedia.org/wiki/Circle_Hough_Transform>

<https://scikit-image.org/>

<https://scikit-image.org/docs/dev/auto_examples/edges/plot_circular_elliptical_hough_transform.html#sphx-glr-auto-examples-edges-plot-circular-elliptical-hough-transform-py>

Détection du rouge :

<https://codereview.stackexchange.com/questions/184044/processing-an-image-to-extract-green-screen-mask>

## Aides reçues

* M. Pascal Bonvin, pour la globalité du projet
* M. François Oscar, pour le Serveur Flask
* M. Guillaume Forestier, pour les pièces supplémentaires
* M. Enzo Roy, pour le partage d’idée
* Mme Nathalie Forestier, pour les relectures

# Table des illustrations

## Figures

[Figure 1 : Design prévisionnel de la page principale 9](file:///C:\Users\Dudude\Desktop\RedBallBot\Documents\Documentation%20Technique.docx#_Toc9501376)

[Figure 2 : Design effectif de la page principale 10](file:///C:\Users\Dudude\Desktop\RedBallBot\Documents\Documentation%20Technique.docx#_Toc9501377)

[Figure 3 : Diagramme de projet général 11](file:///C:\Users\Dudude\Desktop\RedBallBot\Documents\Documentation%20Technique.docx#_Toc9501378)

[Figure 4 : Diagramme de projet détaillé 12](file:///C:\Users\Dudude\Desktop\RedBallBot\Documents\Documentation%20Technique.docx#_Toc9501379)

[Figure 5 : Arborescence du projet 12](file:///C:\Users\Dudude\Desktop\RedBallBot\Documents\Documentation%20Technique.docx#_Toc9501380)

[Figure 6 : Diagramme de classes 13](file:///C:\Users\Dudude\Desktop\RedBallBot\Documents\Documentation%20Technique.docx#_Toc9501381)

[Figure 7 : Méthode de Hough pour trouver un cercle 14](file:///C:\Users\Dudude\Desktop\RedBallBot\Documents\Documentation%20Technique.docx#_Toc9501382)

[Figure 8 : Utilisation de XMLHttpRequest 15](file:///C:\Users\Dudude\Desktop\RedBallBot\Documents\Documentation%20Technique.docx#_Toc9501383)

[Figure 9 : Personnalisation de l'AlphaBot2 18](file:///C:\Users\Dudude\Desktop\RedBallBot\Documents\Documentation%20Technique.docx#_Toc9501384)

[Figure 10 : Décoration Flask 19](file:///C:\Users\Dudude\Desktop\RedBallBot\Documents\Documentation%20Technique.docx#_Toc9501385)

[Figure 11 : Exemple de la fonction Canny 21](file:///C:\Users\Dudude\Desktop\RedBallBot\Documents\Documentation%20Technique.docx#_Toc9501386)

[Figure 12 : Flux vidéo avec balle trouvée 22](file:///C:\Users\Dudude\Desktop\RedBallBot\Documents\Documentation%20Technique.docx#_Toc9501387)

[Figure 13 : Séparation de l'image pour le déplacement 23](file:///C:\Users\Dudude\Desktop\RedBallBot\Documents\Documentation%20Technique.docx#_Toc9501388)

[Figure 15 : Nouveau mode de déplacement 24](file:///C:\Users\Dudude\Desktop\RedBallBot\Documents\Documentation%20Technique.docx#_Toc9501389)

[Figure 15 : Ancien mode de déplacement 24](file:///C:\Users\Dudude\Desktop\RedBallBot\Documents\Documentation%20Technique.docx#_Toc9501390)

[Figure 16 : Télécommande du robot 24](file:///C:\Users\Dudude\Desktop\RedBallBot\Documents\Documentation%20Technique.docx#_Toc9501391)

## Tableaux

[Tableau 1 : Plan de tests 25](#_Toc9501372)

[Tableau 2 : Rapport de tests 26](#_Toc9501373)

[Tableau 3 : Planning prévisionnel 29](#_Toc9501374)

[Tableau 4 : Planning effectif 29](#_Toc9501375)

# Annexe

* Modèles 3D des pièces
* Journal de Bord
* Manuel Utilisateur
* Code source
* Documentation des classes du code source
* Plannings et la liste des tâches

1. Disponible à l’adresse : <https://github.com/Quentinfrstr/RedBallBot> [↑](#footnote-ref-1)
2. Image méthode de Hough : <https://en.wikipedia.org/wiki/Circle_Hough_Transform> [↑](#footnote-ref-2)
3. Les modèles des pièces 3D sont disponible dans les annexes. [↑](#footnote-ref-3)
4. <https://scikit-image.org/docs/dev/auto_examples/edges/plot_canny.html> [↑](#footnote-ref-4)