Département des sciences et technologies
Master en sciences de l'ingénieur industriel
Orientation électromagnétique
Finalisation automatique

Année académique 2021/2022

Projet de recherche et développement

DEADPOOL

Mulnard Théo & Talbi Sara

Projet encadré par
Vachaudez J.
Estievenart W.

Table des matières

Introduction

Dans le cadre du cursus académique de 1re année de master en sciences de l'ingénieur industriel, un projet de recherche et développement doit être mené. Ce projet, ainsi que les personnes choisies pour le réaliser, est imposé par la Haute école Louvain en Hainaut. Avec une précision, le sujet est imposé parmi une sélection faite par l'étudiant·e dans une liste de projets disponibles. Ce fonctionnement est très différent des projets de bachelier où le choix était total. Dans ce cas-ci, la démarche se rapproche nettement plus d'une démarche réelle d'entreprise ou de bureau d'études.

Le projet DEADPOOL a été confié à Mulnard Théo et Talbi Sara, étudiants à la HELHa en 1re année de master 1. Ce projet est initié par M Vachaudez et M Estievenart, chercheurs au CEReF. Ce projet fut motivé par le visionnage d'une vidéo de YouTube (Stuff Made Here, 2021) sur une queue de billard automatique. Dans cette vidéo, le système calcule le meilleur coup possible et avec une très grande précision pour viser automatiquement la bille. Ce système utilise une caméra et un projecteur afin d'analyser le jeu et d'afficher des informations.

L'idée fut donc de reprendre ce système de caméra et de projecteur, mais en retirant la visée automatique avec une queue de billard pour offrir un système d'entrainement autonome. Cette idée est également motivée par l'extrême popularité du billard, mais où il peut être fastidieux de s'entrainer de soi-même.

Le but de ce projet est de fournir un dossier comportant le code du projet ainsi qu'un document fournissant toutes les informations nécessaires à une bonne installation et utilisation du projet. Ces documents et ce programme doivent évidemment être open source.

Ce projet est un projet essentiellement de recherche. Il comporte très peu de pratique. Il a d'ailleurs fallu atteindre 2 mois avant la fin du développement du projet pour avoir besoin du montage pour effectuer les premiers tests.

La plus grande partie du travail et du rapport sera consacrée à la reconnaissance d'image et des billes. Il s'agit là du plus gros obstacle de ce projet. Le second gros obstacle est la correction de perspective de l'image, que ce soit pour la caméra ou pour l'image projetée.

1. Analyse fonctionnelle externe

1.1. Caractérisation du besoin

Saisir le besoin

Le billard est un sport très répandu partout dans le monde. Malgré son apparente simplicité, c'est un sport qui peut se révéler être extrêmement tactique et compliqué à maitriser. L'amateur ne va essayer d'empocher qu'une seule bille, alors que le joueur professionnel va réfléchir plusieurs tours à l'avance. Il va aller chercher à se repositionner correctement, à déplacer d'autres de ses billes ou bloquer son adversaire. C'est finalement un jeu très technique qui demande une très bonne vision du jeu dans son ensemble, mais également d'énormément d'entrainements.

Or l'entrainement autonome peut parfois être compliqué et fastidieux. L'objectif du projet est de faciliter cet entrainement en apportant une aide. Cette aide sera à la fois tactique et visuelle. Il proposera des entrainements dynamiques qui seront projetés directement sur la table de billard pour que le joueur puisse avoir une expérience interactive et ludique.

Diagramme bête à cornes

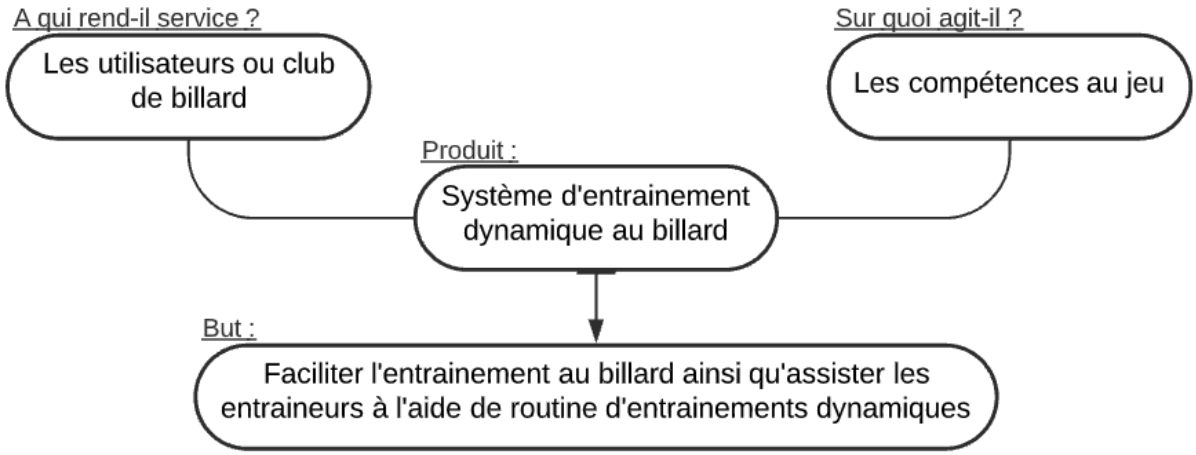


Figure 1.1diagramme bête à cornes

Valider le besoin

Raisons de l'existence du besoin :

- Projet de recherche impliqué dans les études en vue de l'obtention d'un diplôme
- Faciliter l'entrainement des joueurs ou débutants
- Faciliter l'approche au sport pour les non-joueurs
- Fournir un projet open source sur internet pour les personnes à travers le monde

Facteurs pouvant faire disparaitre ou évoluer le besoin

- Alternative commerciale disponible à bas prix et complet
- Évolution du niveau des joueurs
- Ajout de différent type ou mode de jeu
- Modification du code open source selon les besoins de l'utilisateur

Conclusion

De cette courte étude, il est clair que le besoin pour ce projet est bien réel. Le billard est un sport très intéressant, mais qui peut parfois causer de l'appréhension par les nouveaux joueurs à la vue de sa complexité. De plus, à la vue de la faible probabilité des besoins visant à faire disparaitre ce projet, et au contraire le nombre de facteurs pouvant le faire évoluer, ce projet tend à subsister.

1.2. Analyse du milieu extérieur

1.2.1. Situations de vie et d'usages du projet

Situations de vie de l'objet

Pour ce projet, trois situations de vie sont envisagées : développement, fonctionnement et libre accès. Ces trois situations sont choisies, car elles représentent les trois situations les plus globales

Situations d'usage de l'objet.

Développement : - Recherche & Développement (R&D)
Fonctionnement : - Entrainement utilisateur
Libre accès : - Montage

1.2.2. Identification des éléments extérieurs de l'environnement du produit

Développement → Recherche & Développement

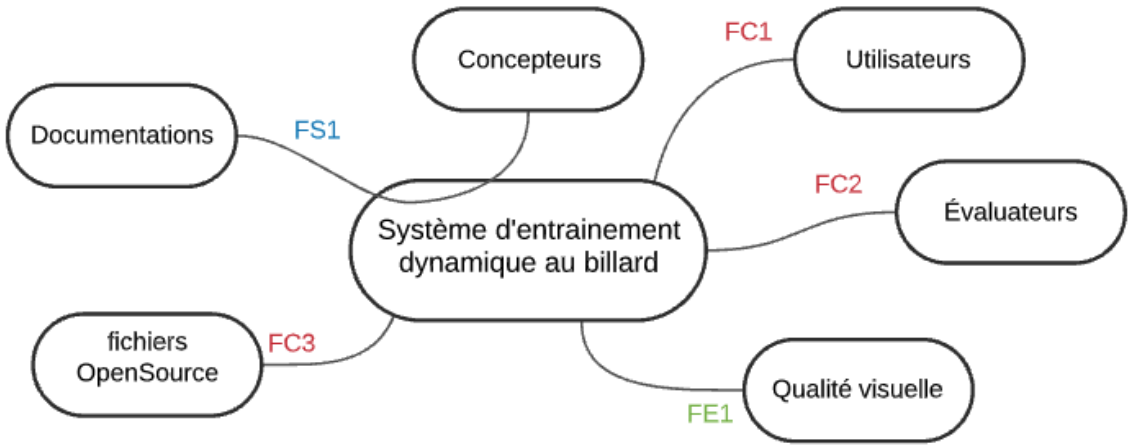


Figure 1.2 diagramme pieuvre des éléments extérieurs pour la phase développement – R&D

FS1 - Choisir la solution la plus simple et efficace pour le projet par les concepteurs.
FE1 - Assurer une qualité visuelle du projet.
FC1 - Assurer une utilisation et un entrainement efficaces par le projet pour le futur utilisateur.
FC2 - Répondre aux critères d'exigences des évaluateurs pour le projet.
FC3 - Fournir un projet final open source.

Fonctionnement → Entrainement utilisateur

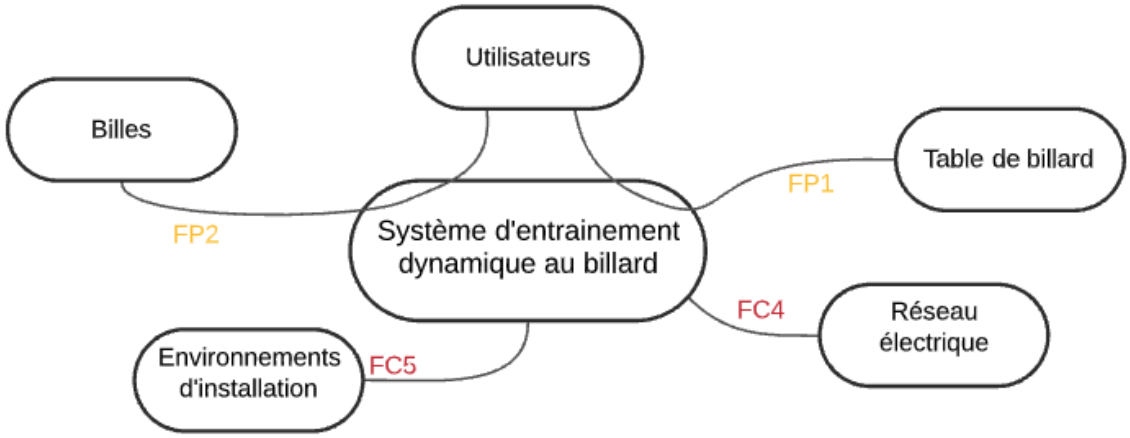


Figure 1.3 Diagramme des éléments extérieurs pour le fonctionnement – entrainement utilisateur

FP1 - Afficher un entrainement dynamique sur la table de billard pour l'utilisateur.
FP2 - Détecter les billes pour le fonctionnement du système d'entrainement dynamique au billard pour l'utilisateur
FC4 - Alimenter le projet sur le réseau électrique.
FC5 - Assurer un bon fonctionnement, peu importe l'environnement d'installation

Libre accès → Montage

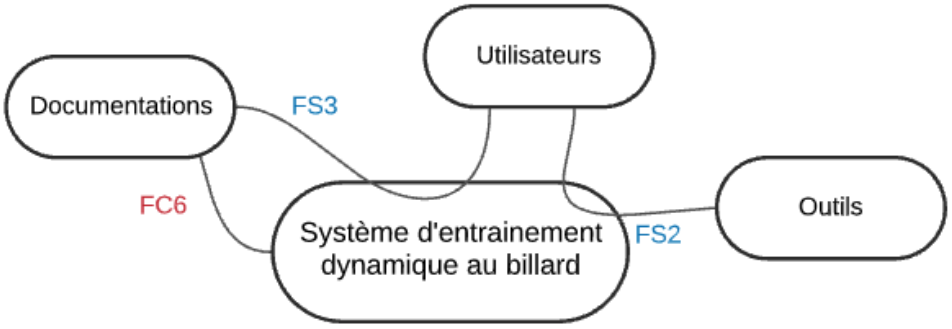


Figure 1.4 Diagramme pieuvre des éléments extérieurs pour la phase développement – R&D

FS2 - Faciliter le montage du projet par l'utilisateur à l'aide de la documentation
FS3 - Assembler le projet à l'aide d'outils par l'utilisateur
FC6 - Assurer une bonne documentation en ligne, gratuite et entièrement libre d'accès

1.3. Analyse et caractérisation des fonctions de la SU principale

Pour la partie analyse et caractérisation des fonctions, seule la situation de vie « Fonctionnement » sera étudiée. La situation de vie « Montage » ne porte pas réellement lieu à une analyse par critère. La situation de vie « Recherche et développement » est précisément ce qui constitue le projet, il n'y a donc pas vraiment de raisons et de logique à caractériser ces fonctions. Au contraire, ce sont les caractérisations apportées qui vont permettre de complètement comprendre cette phase de recherche et développement.

FP1 : afficher un entrainement dynamique sur la table de billard pour l'utilisateur

Tableau 1.1 caractérisation de la FP1

Critères	Niveau du critère	Flexibilité du niveau	Limite d'acceptation	Taux d'échange
Durée affichage	Continu	N/A	N/A	N/A
Fiabilité	99%	± 1%	Min → 95%	Meilleurs composants et développement supplémentaires
Aspect	Sobre	N/A	N/A	N/A

FP2 : détecter les billes pour le fonctionnement du système d'entrainement dynamique au billard pour l'utilisateur

Tableau 1.2 caractérisation de la FP1

Critères	Niveau du critère	Flexibilité du niveau	Limite d'acceptation	Taux d'échange
Temps réponse	2 s	± 1 s	Max → 5 s	Meilleure puissance de calcul
Précision	99%	± 1%	Min → 95%	Meilleurs composants et développement supplémentaires

FC4 : alimenter le projet sur le réseau électrique

Tableau 1.3 Caractérisation de la FC4

Critères	Niveau du critère	Flexibilité du niveau	Limite d'acceptation	Taux d'échange
Alimentation	Suivre les critères d'alimentation pour chaque composant			
Réseau électrique	Suivre la norme relative à l'appareillage électrique à basse tension CEI - 60364			

FC5 : assurer un bon fonctionnement, peu importe l'environnement d'installation

Tableau 1.4 caractérisation de la FC5

Critères[1]	Niveau du critère	Flexibilité du niveau	Limite d'acceptation	Taux d'échange
Distance	2 m	± 0,5 m	Max → 3 m	Caméra avec meilleure résolution
Angle caméra	5°	± 5°	Max → 15° Min → 0°	Code de correction d'image plus performant et robuste
Posit. caméra	Centre de la table	± 10 cm	Max → 15 cm Min → 5 cm	Système de fixation (impression 3D) pour l'ensemble
Angle projecteur	0°	± 2°	N/A	Code de correction d'image plus performant et robuste
Posit. projecteur	Centre de la table	± 2 cm	Max → 5 cm Min → 5 cm	Système de fixation (impression 3D) pour l'ensemble

[1] L'angle est donné entre une verticale et la direction donnée par les focales de la caméra / du projecteur. Les valeurs positives (distance et angle) sont données par rapport au / en direction du centre de la table.

1.4. Classification sur base de l'importance des fonctions

	FP2	FS1	FS2	FS3	FE1	FC1	FC2	FC3	FC4	FC5	Points	%
FP1	2 FP1	4 FP1	5 FP1	5 FP1	4 FP1	2 FP1	4 FC2	4 FC3	3 FP1	3 FP1	28	16,5%
FP2	4 FP2	5 FP2	5 FP2	4 FP2	2 FP2	4 FC2	4 FC3	3 FP2	3 FP2	26	15,3%
FS1	2 FS2	4 FS1	3 FS1	3 FC1	5 FC2	4 FC3	2 FC4	1 FC5	7	4,1%
FS2	3 FS2	2 FS2	4 FC1	5 FC2	4 FC3	2 FC3	3 FS2	8	4,7%
FS3	3 FE1	4 FC1	5 FC2	4 FC3	3 FC3	1 FS3	1	0,6%
FE1	3 FC1	4 FC2	4 FC3	1 FE1	2 FE1	6	3,5%
FC1	2 FC1	2 FC3	2 FC1	3 FC1	21	12,4%
FC2	1 FC2	1 FC4	4 FC2	32	18,8%
FC3	4 FC4	4 FC3	30	17,6%
FC4	3 FC5	7	4,1%
FC5	4	2,4%
TOTAL	170	100,0%

Après cette classification, il en ressort cinq fonctions ressortant du lot. Ces cinq fonctions vont permettre d'évaluer quels sont les points les plus importants et d'écrire le cahier des charges au point suivant. Les fonctions sont classées dans l'ordre d'importance :

FC1 - Assurer une utilisation et un entrainement efficaces par le projet pour le futur utilisateur.
FC2 - Répondre aux critères d'exigences des évaluateurs pour le projet.
FC3 - Fournir un projet final open source.

FP1 - Afficher un entrainement dynamique sur la table de billard pour l'utilisateur.
FP2 - Détecter les billes pour le fonctionnement du système d'entrainement dynamique au billard pour l'utilisateur

Sans surprise, ce sont les fonctions contraintes qui arrivent en premières positions. Elles correspondent aux énoncés du projet et à l'attente des superviseurs. La caractéristique open source est également importante. Les deux fonctions principales sont également présentes, car ce sont elles qui vont valider ou invalider la suite du projet.

1.5. Cahier des charges

Dans cette analyse fonctionnelle, la partie interne n'est pas réalisée. Comme il s'agit d'un projet de recherche et développement, ce projet sera constitué d'essais et erreurs afin d'arriver au résultat final. Il n'est donc pas logique de faire réaliser cette analyse interne. Néanmoins, cette analyse fonctionnelle permet de mettre en place un cahier des charges. Ce cahier des charges concorde évidemment avec les attentes des superviseurs du projet.

D'un point de vue utilisateur, le cahier des charges finales donne les points suivants :

1. Mise en place d'un système d'entrainement au coup par coup
2. Mis en place d'un système d'entrainement de jeu avec points.

D'un point de vue du fonctionnement logiciel, les points imposés sont :

3. Calibrage logiciel automatique de la caméra pour contrer l'effet de perspective
4. Calibrage logiciel automatique du projecteur pour l'affichage
5. Reconnaissance d'image des boules et encodage de leur position

Pour les contraintes, imposées par les superviseurs ou autres, celles-ci sont :

6. Répondre aux critères d'exigences des évaluateurs
7. Assurer un bon fonctionnement, peu importe les conditions d'installations (dans la limite du raisonnable, se référer à la future documentation)
8. Réaliser le projet sous python et open source
9. Réaliser une documentation en ligne du projet, open source et complète.

2. Recherches préliminaires et environnement de travail

2.1. Choix du single board controller

Aucune documentation n'est disponible sur les spécifications minimums pour qu'openCV fonctionne pleinement. Cependant, sur le forum de questions-réponses du site internet, il est indiqué que le logiciel peut tourner sur à peu près tout tant que la plateforme supporte Windows ou Linux.

> *« OpenCV will run on anything. […]. As long as it can run Windows or Linux, you're mostly OK. Then, everything depends on your processing needs, and you didn't tell what you want to do. There are some very fast algorithms, but other need lots of computing power. »*
>
> (Anonym, 2017)

Le choix doit donc se faire au cas par cas. Tout va dépendre du code qui doit tourner derrière, des besoins de la RAM pour le traitement d'image, etc. La seule chose sûre est que le GPU n'est pas utile pour ce type d'application. Celui-ci n'est important que dans des applications plus poussées comme le traitement d'image par exemple. Il faut donc déterminer, ou estimer quelle est la configuration minimum.

En toute logique, le *single board controller* (SBC) devrait être le plus puissant possible pour être capable de traiter l'image le plus rapidement possible et de poursuivre dans le code. Néanmoins, le fonctionnement (prendre une photo, l'analyser, en sortir des conclusions) n'est pas probablement pas une demande extrême à notre époque. Le système ne fonctionnera pas avec une vidéo donc la puissance nécessaire ne devrait pas être excessive. En contre-argument, avoir un micro-ordinateur puissant permettrait d'effectuer ces traitements plus rapidement, donc de prendre des photos plus souvent et de détecter les changements plus souvent. Un compromis est donc à trouver.

Pour ce qui est des sorties périphériques, il faut une sortie HDMI pour pouvoir utiliser le projecteur. Il faut au minimum 2 ports USB, pour pouvoir brancher un clavier et une souris. Il faut également un port microsd pour le stockage. Pour le fonctionnement, une connexion internet n'est pas nécessaire, et donc une clé wifi peut être utilisée pour la configuration du logiciel si nécessaire avant la mise en fonctionnement.

Figure 2.1 Raspberry Pi 3 modèle B

Le choix final se porte sur un Raspberry Pi 3 modèle B. Ce modèle possède 1gb de RAM, 4 ports USB, un port HDMI et un port micro-USB pour l'alimentation, un port Ethernet, le Bluetooth 4.1 ainsi que du wifi embarqué. Ce choix est justifié par les caractéristiques qui devraient être suffisantes. De plus celui-ci était déjà disponible et il ne faut pas oublier que lors de ce projet, le monde est encore plongé dans une pénurie globale de composants électroniques, rendant assez compliqué l'achat de micro-ordinateur tel que des Raspberry.

2.2. Choix de la caméra d'acquisition

La caméra est l'élément clé de ce projet. Il est important de bien la choisir. Les caractéristiques principales pour une caméra sont sa résolution ainsi que son optique. Les deux sont importants pour que le capteur photo puisse recevoir suffisamment de lumière et que le résultat final est une bonne définition. La résolution reste néanmoins l'élément principal de comparaison.

Pour des raisons de simplicité, le choix se limitera aux caméras les plus répandues et à celles pouvant s'intégrer directement à l'interface caméra des Raspberry PI. Des caméras pouvant fonctionner en USB ou avec des cartes d'acquisition pourraient être utilisées pour obtenir une meilleure qualité, mais cela compliquerait le système. De plus, il y a fort à parier que des images trop grandes viennent compliquer le travail du Raspberry (ce point sera vérifié par la suite). Voici un tableau récapitulant les caractéristiques de différentes caméras.

Tableau 2.1 comparaisons des caméras

	Cam. Raspberry Pi LABISTS B01	**Cam. Fish-eye pour Raspberry**	**Cam. Raspberry Pi V2.1**	**Cam. LABISTS Raspberry Pi V2**
Résolution	5 MP | 2592 x 1944 px		8 MP | 3280 x 2464 px	
Type de capteur	Omnivision 5647	/	Sony IMX219-8	Sony IMX219-8
Résolution vidéo max	1080p @30fp | 720p @60fps | 480p @90fps			
Prix	± 13 €	± 26 €	± 29 €	± 30 €

Afin de mettre toutes les chances du côté de la réussite, c'est la caméra Raspberry Pi V2.1 qui est choisie pour sa résolution d'affichage plus grande. C'est cette caméra était également disponible dès le début du projet.

Figure 2.2 caméra Raspberry Pi V2.1

Cependant, et pour anticiper le dernier chapitre sur les tests en conditions réelles, il y est montré que le Raspberry PI 3 b ne supporte pas une telle résolution d'image. Et même s'il la supportait, le temps de traitement est extrêmement long avec ce modèle de SBC. Avec ce modèle spécifique, une caméra de première génération pourrait alors suffire. Cependant, le choix final reste la 2.1, car elle offre plus de possibilités.

2.3. Choix du projecteur

Le projecteur est également un composant très important du projet. Ce dernier sert de lien entre le système d'entrainement, le code, et l'utilisateur. Si celui-ci n'est pas bon, les informations qui seront fournies à l'utilisateur et au code ne seront pas correctes.

Les critères importants pour le choix d'un projecteur sont la résolution, la luminosité, le contraste, le poids et enfin les entrées disponibles. La résolution se définit par la netteté et la qualité d'une image. Plus la luminosité dans la pièce sera élevée, plus le projecteur doit avoir une luminosité grande afin d'afficher un contenu visible. Il faut néanmoins faire attention, car plus le projecteur sera loin, plus la luminosité devra être importante. Le contraste est exprimé comme étant le rapport entre le point le plus sombre et le plus clair que le vidéoprojecteur pourra afficher.

Pour ce projet, un projecteur (MEDION md22900) était déjà disponible. La première chose à faire est donc de vérifier les caractéristiques de ce produit pour savoir s'il pourrait correspondre avant d'acheter un nouveau projecteur.

Figure 2.3 projecteur médion md22900

Tableau 2.2 caractéristiques du projecteur (MEDION)

Caractéristiques	Valeurs
Dimensions	105x31x105 mm
Poids	0,27 kg
Résolution	854x480 pixels (16 : 9) | 800 x 600 pixels (4 : 3)
Puissance lumineuse	Jusqu'à 55 lumens
Rapport de contraste	1000 : 1
Taille max d'image projetée	0,3 à 3,05 m
Écart avec la surface de projection	0,5 à 5 m

Premièrement, le point bonus de ce projecteur est qu'il possède un filetage à sa base comme sur les appareils photo, ce qui est utile pour le futur montage. La résolution n'est pas très grande. Ce n'est pas extrêmement important, car les informations qui devront être affichées seront grandes et la précision ne se fera pas au pixel. Le contraste n'est pas très élevé, mais devrait, en théorie, être suffisant.

En théorie, ce projecteur pourrait convenir à cette utilisation. Ces capacités seront vérifiées dans le chapitre 9- Tests en conditions réelles. Dans tous les cas, il convient pour la première phase d'un projet de recherche et développement.

2.4. Détermination de l'environnement de travail

Pour un tel projet de recherche et développement, essentiellement basé sur de la programmation, il est important de définir l'environnement de travail utilisé pour le codage ainsi que pour les tests sur le SBC. Pour des raisons de facilités et de rapidité de tests, le projet est entièrement codé sur les ordinateurs personnels puis est porté vers le SBC pour les tests en conditions réelles.

Ordinateur de programmation

Le projet est développé sous Python 3.10 et avec openCV 4.5.4 sur un MacBook pro de 2017 (Intel I5 2,3 GHz et 8 Go RAM) tournant sous macOS 11.6. Le logiciel de programmation utilisé est PyCharm community Edition (Build #PC-212.5284.44). Il s'agit d'un environnement de développement intégré, couramment appelé IDE, très puissant et avec beaucoup de fonctionnalités, de détection d'erreurs, d'aides de simplification de syntaxes, etc. Comparée à la version professionnelle, la version *community* est gratuite, mais ne permet de ne coder qu'en python. Ce n'est donc pas un problème ici.

Environnement du SBC

Pour le Raspberry, comme dit plus haut, celui-ci est un Raspberry Pi 3 modèle B de 1gb de RAM. L'OS est la toute dernière version, soit Raspberry PI OS (32-bit), sorti le 28 janvier 2022. Il a été flashé sur une carte SD de 16gb à l'aide du logiciel Raspberry Pi Image v1.7.1. La version de python utilisé est la 3.9.2 et la dernière version d'openCV. Sur le Raspberry, l'IDE utilisé est celui intégré avec l'OS, à savoir Thonny. C'est un logiciel simple, mais très léger, permettant des petites modifications ou des corrections d'erreurs en cas de besoin.

Pour le contrôle du Raspberry, lors des tests à domiciles, ce dernier est connecté à un écran externe via sa sortie HDMI et contrôlé avec un clavier Bluetooth Logitech K400+. Lors des tests en conditions réelles, le contrôle se fait directement depuis l'ordinateur personnel via une retranscription de l'écran du Raspberry Pi via Ethernet et le logiciel VNC Viewer. Un contrôle est également possible utilisant le clavier cité ci-dessus et en regardant l'image projetée sur la table de billard.

3. Traitement d'images

3.1. Recadrage de l'image

Le recadrage de l'image est une étape extrêmement importante dans ce projet. Puisqu'à la fois le projecteur et la caméra doivent être positionnés au-dessus de la table de billard, les deux ne peuvent pas se trouver parfaitement au centre en même temps. De plus, toujours en imaginant que ce projet pourrait être utilisé et installé par n'importe qui, il faut imaginer un projet assez robuste. Apporter une correction qui permettrait de corriger les éventuelles erreurs d'installation, dans la mesure du raisonnable. Afin de développer et d'expliquer les points suivants, un exemple sera pris (Figure 3.1).

Figure 3.1 exemple billard - image de base

3.1.1. Détection des tags ArUco

Afin de détecter les extrémités de la table en vue de la modification de la perspective, ce sont les tags ArUco qui sont utilisés. Ces tags sont composés d'une bordure noire et d'une matrice interne composée de carré blanc et noir correspondant à un identifiant bien spécifique. La détection et la gestion de ces tags sont directement intégrées à openCV, ce qui en fait un candidat idéal pour la correction des points.

> *« The ArUco module is based on the ArUco library, a popular library based for detection of square fiducial markers developed by Rafael Munoz and Sergio Garrido »*
>
> (OpenCV, 2021)

Plusieurs tailles de tags peuvent être utilisées. Leur taille correspond au nombre de bits dont ils sont composés. Pour ce projet, les formats utilisés sont 5X5_100. Ce qui correspond à des tags de 100 mm de côtés et 25 bits. Il y a 7 carrés et non 5 à cause des bandes noires. Ce type a été choisi de manière plus ou moins arbitraire, car il n'y a pas vraiment de mauvaises réponses. Il fallait simplement veiller à ne pas utiliser des types trop grands en dimensions pour que les tags puissent se positionner sur le côté de la table. Également, la taille (ici du 5x5) ne devait pas être trop grande pour être sûre que, avec la distance de la caméra et sa résolution, les tags soient correctement détectés.

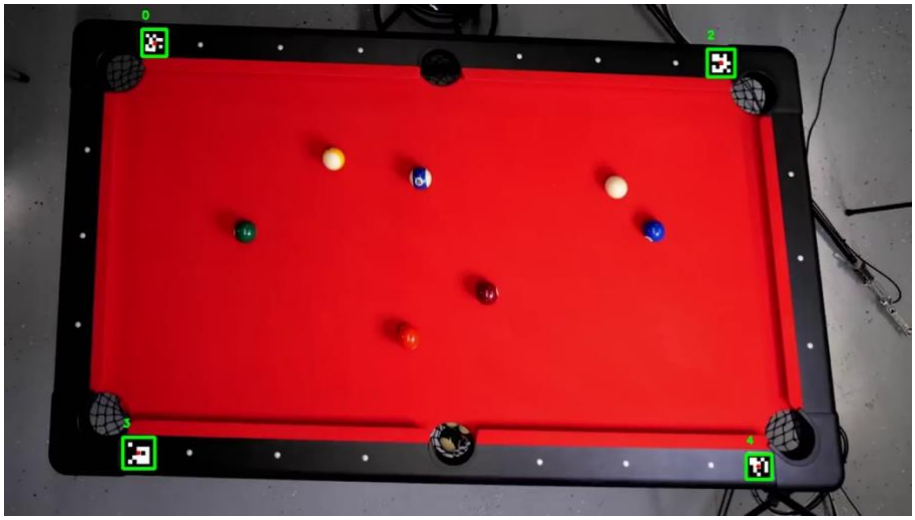


Figure 3.2 exemple billard – tags ArUco détectés

Pour ce qui est du code, celui se base sur un tutoriel (Rosebrock, 2020) avec un certain nombre de modifications et d'améliorations. Cette partie se base sur la librairie cv2.aruco qui est une librairie toute faite permettant de détecter ces tags. Le reste du code consiste à tracer des carrés autour de ces tags en fonction des données récupérées, de calculer et tracer le centre et enfin d'indiquer l'identifiant du tag.

Ce programme a été placé dans une fonction et écrit de manière générale afin de pouvoir la réutiliser notamment pour la correction plus tard de l'affichage avec projecteur.

3.1.2. Correction de la perspective de l'image

Maintenant que les tags, et plus important encore, que leurs centres sont trouvés, il ne reste plus qu'à modifier la perspective de l'image. Pour ce faire, la première étape consiste à savoir quelle est la position relative de chaque tag dans l'image. En d'autres termes, lesquelles sont à gauche / à droite et en haut / en bas.

Une solution serait d'utiliser les identifiants afin de définir à quelle position chacun correspond. Mais cette solution créée beaucoup de problèmes. Que faire si les tags ne sont pas placés dans le bon ordre ? Que faire si la caméra est à l'envers ? Pour contourner ces problèmes, il faut trier les tags en fonction de leurs coordonnées. Explication :

```
1.  tagList = []
2.  tempList = []
3.  tagCenters = {}
4.
5.  for i in range(0, 4):
6.      tempList.append((tagList[i][0] + tagList[i][1], i))
7.  tempList.sort()
8.  tagCenters["TOP_R"] = tagList[tempList[0][1]]
9.  tagCenters["BOT_R"] = tagList[tempList[1][1]]
10. tagCenters["TOP_L"] = tagList[tempList[2][1]]
11. tagCenters["BOT_L"] = tagList[tempList[3][1]]
```

Lors de la détection, les coordonnées du centre de chaque tag sont enregistrées dans une liste (*tagList*). L'algorithme de tri consiste simplement à additionner les valeurs en x et en y et à placer ces valeurs dans une nouvelle liste *tempList*. Comme il s'agit d'une image rectangulaire, les 4 résultats sont différents. De plus, l'image sera toujours au format paysage,

car aligné avec le projecteur sur la table de billard. Ainsi, ces sommes peuvent être triées en utilisant templist.sort() et correspondent chacun à une position précise dans l'image. Ces positions sont fixées par le système de coordonnées d'openCV qui est constant.

Mais le problème maintenant est de pouvoir lier ce tri aux coordonnées d'origines. Pour régler ce souci, un numéro est ajouté à chaque somme dans *tempList*. C'est cet identifiant qui va permettre d'aller rechercher le couple de coordonnées dans la liste d'origine. Autrement dit, l'association se fait avec la liste *tagCenters* à la position templist[X][1].

Ces données sont sauvegardées dans un dictionnaire python. Cela permet d'aller chercher d'utiliser de manière claire et simple les coordonnées de chaque tag. Cet algorithme est intégré directement à la détection des tags. C'est un algorithme assez simple, mais très robuste et efficace.

La suite de la correction de perspective consiste à créer deux matrices : l'une avec les coordonnées de l'image d'origine et une avec les coordonnés de la nouvelle image. Une fonction dans openCV permet ensuite de créer une matrice permettant de passer d'une matrice à l'autre. Enfin, une seconde fonction se charge d'appliquer cette matrice de transformations sur l'image.

Figure 3.3 exemple billard – correction de perspective

Il faut noter que dans cet exemple-ci, un *offset* ou décalage a été ajouté sur les coordonnées des centres dans la largeur afin de pouvoir voir la table de billard en entier. Cette option est paramétrable dans la fonction liée à la correction de perspective. Il est donc tout à fait possible de demander à l'utilisateur de placer les tags aux coins, et sinon d'activer l'option d'*offset*.

3.2. Suppression de l'arrière-plan

3.2.1. Besoin de la suppression

À la suite d'essais préliminaires, il apparait que, peu importe la méthode de détection utilisée, il est préférable de complètement supprimer l'arrière-plan de l'image (les méthodes sont présentées au point suivant). Dans le cas présent, celui-ci est de couleur assez unie et sa suppression devrait donc être assez simple. Supprimer l'arrière-plan va ainsi empêcher les fausses détections, car le fond ne sera constitué que de pixels noirs.

Pour supprimer l'arrière-plan, la technique est assez similaire à ce qui serait utilisé dans un système de retouche photo. Premièrement, un masque est appliqué sur la photo afin de ne sélectionner que la couleur souhaitée. Pour sélectionner la couleur, il faut fournir un seuil de détection entre une valeur max et une valeur min (ce point sera repris par la suite). Ensuite au travers de plusieurs manipulations le masque est appliqué sur l'image. Cette partie dans le code est inspirée d'un utilisateur sur internet sur le site stackoverflow (StackOverflow, 2019).

3.2.2. Détermination de la couleur d'arrière-plan

Revenons sur le seuil de détection de la couleur. Afin de pouvoir adapter ce système à toutes les situations, le programme doit être capable de détecter la couleur d'arrière-plan. Les tables de billard qui existent sont principalement de couleur rouge, verte ou bleue. Il faut donc un programme permettant de détecter la couleur principale dans l'image, déterminer de quelle couleur il s'agit et enfin d'appliquer un seuil spécifique en fonction de la valeur. La partie détermination de la couleur est elle aussi inspirée d'un utilisateur sur le site stackoverflow (StackOverflow, 2018).

Pour la partie détermination de la couleur de la table, une simple comparaison est faite entre les trois valeurs RGB. Par chance, ces trois couleurs primaires correspondent aux trois couleurs possibles que peut avoir une table, il y aura donc toujours obligatoirement une valeur qui se démarquera. Les seuils appliqués pour chaque couleur sont assez sensibles, il est donc nécessaire de correctement les configurer au moyen de tests.

```
1.   if (gVal > bVal) & (gVal > rVal):  # Pool table is green
2.           if terminalOutput:
3.               print("   table is green")
4.           lower = np.array([0, 150, 0])
5.           upper = np.array([120, 255, 120])
6.       elif (bVal > gVal) & (bVal > rVal):  # Pool table is blue
7.           if terminalOutput:
8.               print("   table is blue")
9.           lower = np.array([150, 0, 0])
10.          upper = np.array([255, 150, 150])
11.      elif (rVal > gVal) & (rVal > bVal):  # Pool table is red
12.          if terminalOutput:
13.              print("   table is red")
14.          lower = np.array([0, 0, 150])
15.          upper = np.array([60, 60, 255])
```

Figure 3.4 suppression de l'arrière-plan sans améliorations

La suppression fonctionne correctement. Évidemment la bille rouge ne ressort pas bien, mais il s'agit d'un problème inhérent à la méthode de détection. Il suffira de préciser, si des billes de couleurs sont utilisées, laquelle utiliser en fonction de la couleur de la table de billard afin d'avoir un bon contraste. Le seul problème pour l'instant est que les bords de l'image ne sont pas supprimés. Or ceux-ci causent également des problèmes lors de la détection des billes et entrainent de faux résultats. Notamment à cause de la forme ronde des trous. Pour résoudre ce problème, des solutions simples sont démontrées au paragraphe suivant.

3.2.3. Suppression des bords de l'image

Afin de supprimer les bords de la table de billard, des rectangles noirs sont simplement dessinés sur l'image dans les quatre côtés. Cette étape, bien qu'elle puisse créer quelques soucis dans les cas où la bille se trouverait contre un bord, permet d'avoir un fond de couleur pratiquement unie. De plus, comme les dimensions de l'image sont constantes, quelle que soit la situation grâce à la correction de la perspective, il est possible de trouver une valeur « d'offset » ne mordant pas trop sur les bords de la table.

Figure 3.5 suppression de l'arrière-plan avec améliorations visibles

Pour les coins de l'image, autrement les 4 filets aux extrémités, la logique est la même qu'au-dessus avec cette fois des cercles dessinés dans les 4 coins. Pour les deux filets centraux, ceux-ci ne sont pas corrigés, car ils n'ont provoqué aucun souci lors des tests préliminaires. De plus, ils sont déjà à moitié recouverts par les rectangles et que leur profondeur dans la table peut varier d'une table à l'autre (en fonction du type).

La figure ci-dessus montre visiblement les formes ajoutées afin de supprimer les bords de l'image. Une comparaison avec la figure d'avant (Figure 3.4) montre bien que la table en elle-même est conservée dans sa quasi-totalité, à quelques pixels près. Les recouvrements des coins ne sont pas forcément complets, mais ils permettent d'éviter de faux résultats par les méthodes de détection et sont donc suffisants. Sur l'image, la suppression de l'arrière-plan est pratiquement complète.

Figure 3.6 suppression complète de l'arrière-plan

4. Détections des billes

4.1. Méthodes envisagées

4.1.1. Méthode 1 : utilisation de la méthode ‘template matching’

Explication de la méthode

La première idée qui vient en tête lorsque l’on parle de détection d’objets est le concept de « machine learning ». Il s'agit cependant d'un procédé assez complexe et qui demande un certain nombre de ressources pour l’ordinateur. Dans ce projet, la détection semble assez simple : les objets à détecter sont de couleur unie sur un fond uni également.

La méthode envisagée est cv2.matchTemplate(‘source’, ‘template’, ‘method’). Cette méthode prend le gabarit (‘template’) et va le glisser sur l’image source pixel par pixel. Ensuite elle compare le gabarit et ce qui se trouve en dessous de l’image. Au total, 6 méthodes de comparaison existent, d’où l’argument ‘method’ afin de préciser laquelle est utilisée. En sortie, la méthode cv2.template fournit une image en nuance de gris où chaque pixel représente comment chaque pixel voisin correspond ou non à l’image source. Quelques manipulations supplémentaires permettent de sortir le rectangle de détection de l’objet. (OpenCV, 2021)

Figure 4.1 gabarit utilisé pour la méthode

Test de la méthode

Pour tester cette méthode, l’image source utilisée est l’image qui a été obtenue en initialisant la correction de perspective de l’image (Figure 3.3). Pour l’image gabarit, c’est normalement une image provenant directement de l’image source qui doit être utilisée. Dans ce projet, le système doit s’adapter à chaque situation. Il faut donc un gabarit le plus simple possible (Figure 4.1) .

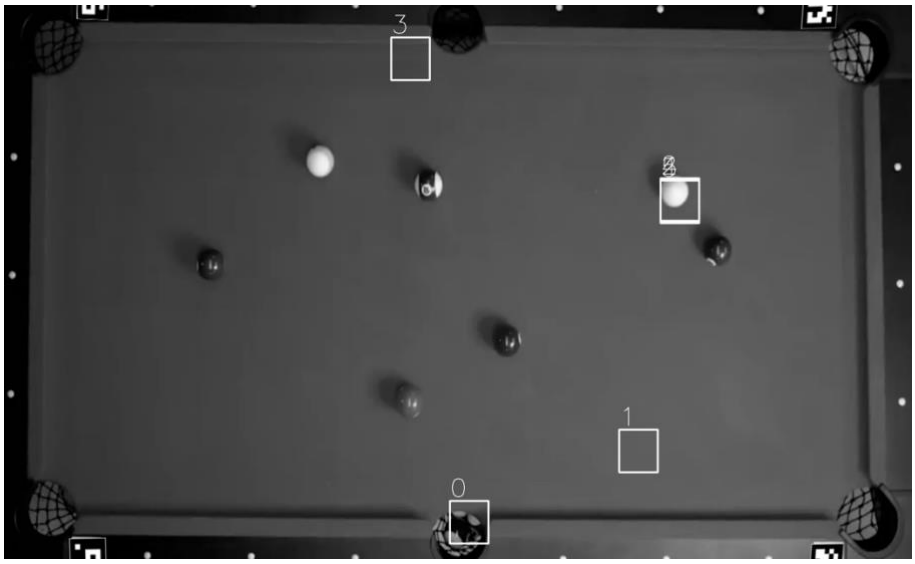


Figure 4.2 reconnaissance d’objets avec cv2.matchTemplate et arrière-plan

Ici deux méthodes de tests provoquent des résultats aberrants. Ces résultats ne peuvent pas exister, car il n'est pas dit que, en choisissant une méthode qui fonctionne dans ce cas, elle fonctionnera à chaque fois. C'est pourquoi c'est l'image sans arrière-plan qui est utilisée.

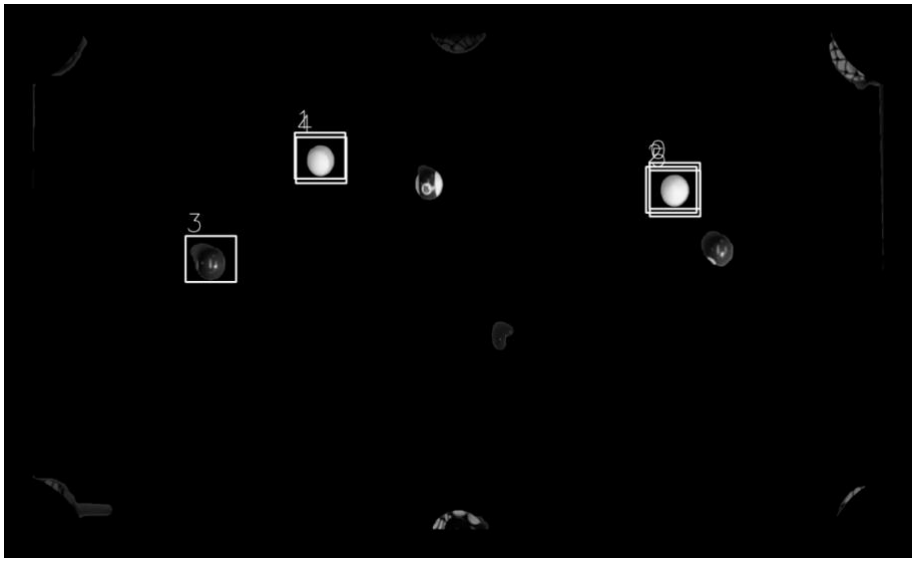


Figure 4.3 reconnaissance d'objet avec cv2.matchTemplate et sans arrière-plan

Conclusion sur la méthode

Le premier problème qui saute aux yeux et que cette méthode travaille avec une image en nuance de gris. De ce fait, la reconnaissance de billes, même pour l'œil humain, est déjà plus compliquée qu'en temps normal. En utilisant le gabarit d'une bille blanche, certaines méthodes détectent de manières erronées la bille verte et la bille rayée jaune et blanche. Ce problème est lié à la méthode de détection en elle-même. Pour résoudre ce problème, il faudrait isoler les billes détectées, déterminer la / les couleur(s) dominante(s) pour déterminer de quelle bille il s'agit. Cette solution est néanmoins assez fastidieuse. De plus, les couleurs vont être fortement influencées par l'éclairage, ce qui rend la chose encore plus complexe.

Le second problème est que la détection n'est pas des plus précises. Les détections ne sont jamais centrées sur l'objet, ce qui n'est pas idéal pour un projet lié à de l'entrainement, et donc demandant de la précision. Sur ce test, l'écart semble constant et pourrait donc être corrigé dans le code, mais rien n'indique cependant que ce sera le cas pour toutes les situations de jeux possibles.

4.1.2. Méthode 2 : utilisation de la méthode ‘houghCircle’

Explication de la méthode

La seconde technique envisagée est complètement différente de celle vue précédemment. Le but ici est de se servir de la méthode de détection des cercles fournie avec openCV afin de détecter les billes. Cette méthode devrait, en théorie, être beaucoup plus précise que la méthode cv2.matchTemplate. Cependant, si celle-ci n’a pas été envisagée en premier, c'est parce qu’elle ne permet pas par elle-même de détecter de quelle bille il s’agit. Contrairement, en théorie, à la méthode avec le gabarit. De plus, il y a une correction de perspective dans cette image, ce qui fait que les billes pourraient avoir tendance à ne pas être parfaitement rondes et donc avoir de faux résultats.

La méthode envisagée est cv2.houghCircles(‘source’, ‘method’, ‘dp’, ‘minDist’). Cette méthode utilise elle aussi une image en nuances de gris pour détecter les cercles. Les méthodes servent à la détection. La méthode utilisée ici est HOUGH_GRADIENT. La variable ‘dp’ est le rapport inverse de la résolution de l’accumulateur de l’image source. La valeur recommandée est 1.5, mais celle-ci peut être augmentée si de petits cercles doivent être trouvés. La variable ‘minDist’ correspond à la distance minimale entre deux centres de cercles détectés. Si le paramètre est trop grand, certains cercles peuvent ne pas être détectés, mais s’il est trop petit, de faux résultats peuvent apparaitre. En sortie, cette méthode donne une liste des centres (x, y) des cercles ainsi que leurs rayons. Quelques manipulations supplémentaires permettent d’afficher le cercle et le centre de détection (OpenCV, 2021).

Test de la méthode

Pour tester cette méthode, l’image source utilisée est l’image dont l’arrière-plan a été supprimé (Figure 3.6 suppression complète de l'arrière-plan). Des tests préliminaires ont montré que pour que cette détection fonctionne, il faut que la variable ‘dp’ soit assez grande. Ce qui provoque un très grand nombre de fausses détections (voir Figure 4.4), en utilisant une image avec l’arrière-plan. Bien que la détection se fasse sur une image en nuance de gris, les résultats sont affichés sur l’image en couleur.

Figure 4.4 reconnaissance d’objets avec cv2.houghCircles et arrière-plan

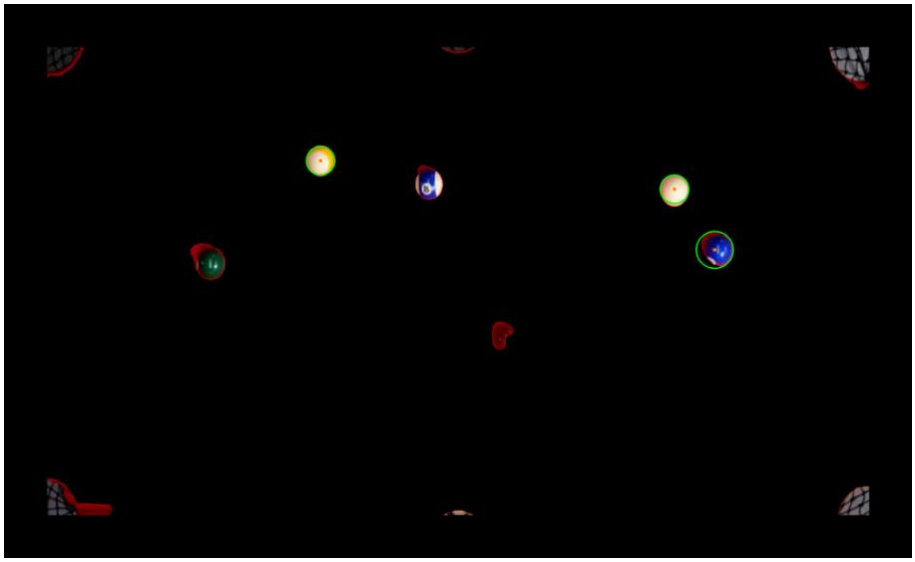


Figure 4.5 reconnaissance d'objets avec cv2.houghCircles sans arrière-plan

Conclusion sur la méthode

La première remarque est l'importance de retirer l'arrière-plan. Sur la première image, énormément de fausses détections ont lieu, bien que la bille blanche et la bille blanche/jaune soient correctement détectées. Ce test remet en avant le travail fait en amont sur la modification de l'image, notamment sur les filets où les détections sont nombreuses, dû à leur forme relativement ronde.

Un avantage évident de cette méthode est la précision des détections. Celles-ci sont beaucoup plus précises qu'avec la méthode précédente. En bonus, avec seulement une méthode, il est possible de détecter plusieurs billes. Contrairement à auparavant où plusieurs gabarits devaient être utilisés afin d'espérer détecter une bille spécifique.

De manière générale cette méthode semble être favorable à la méthode précédente. Cependant l'inconvénient avec cette méthode est qu'ici la détermination de la couleur des billes devient obligatoire dès que plusieurs billes veulent être utilisées. Il n'est ici plus possible de détecter une seule bille à la fois, ou du moins de ne détecter qu'une seule bille parmi des billes de contrastes assez différents.

4.2. Test de détection d'un lot d'images avec les méthodes

Toutes les méthodes vues précédemment sont assez sensibles à la casse. La modification légère d'un paramètre peut donner des résultats qui peuvent être très bons ou très mauvais. Cela a déjà été répété assez souvent, mais ce projet ne doit pas fonctionner pour un seul projet, mais dans tous les cas. C'est pourquoi un essai en lot avec des situations variées doit être réalisé.

Cet essai va utiliser les mêmes fonctions que lors du traitement d'une seule image. Cependant une boucle va permettre de faire varier l'image de base ainsi que les chemins de fichier pour venir sauvegarder les différents résultats. Le code affiché en dessous n'est pas complet (les fonctions n'ont pas été indiquées), mais il est présent, car la boucle avec la variation pour aller chercher les différents fichiers reste assez intéressante.

```
1.  print("[info] starting batch testing")
2.
3.  for i in range(vMin,vMax):
4.      print("> pool-" + str(i) + ".png")
5.      inputImg = "assets/test_batch/test-" + str(i) + ".png"
6.      resultPathNoBack = "assets/test_batch/r_noback/resultNB-" + str(i) + ".png"
7.      resultPathTempDtct = "assets/test_batch/r_tempdtct/resultTmpDt-" + str(i) + ".png"
8.      resultPathCircDtct = "assets/test_batch/r_circdtct/resultCirDt-" + str(i) + ".png"
9.
10.     # code to warp an image by detecting ArUCo tags
11.     # code to remove background
12.     # code to the white ball and printing the center
13.     # code to the white ball using circle detection
14.
15.     print("[info] image number [" + inputImg + "successfully tested")
16.
17. print("[info] ending batch testing")
```

Plus de 16 images différentes ont été utilisées. Il est évident que les 16 résultats ne vont pas être fournis. Néanmoins ces essais ont permis de montrer un gros problème avec la méthode cv2.houghCircles. Avec certaines images, le test prenait plusieurs dizaines de secondes pour faire les opérations nécessaires. Le problème venant de la méthode susmentionnée, où celle-ci détectait des cercles sur pratiquement tous les pixels de l'image.

Ce problème a été résolu en ajustant les paramètres de la méthode. Le 'dp' a été grandement augmenté afin d'augmenter la tolérance de détection. En contrepartie, un rayon maximum et un rayon minimum ont été ajoutés. Ces valeurs n'ont pas une tolérance précise, et devraient donc être adaptées pour chaque situation. Étant donné que le rapport entre la taille de la table et la taille d'une bille devrait rester plus ou moins constant. Afin d'éviter de détecter des billes trop proches, la distance minimale a été définie à une valeur entre les deux valeurs de rayons. Cela permet de détecter deux billes collées, mais d'éviter d'avoir deux détections sur la même bille.

```
1.  # ancien appel de la méthode
2.  circles = cv2.HoughCircles(grayImg, cv2.HOUGH_GRADIENT, 2.5, 10)
3.
4.  # nouvel appel de la méthode
5.  circles = cv2.HoughCircles(grayImg, cv2.HOUGH_GRADIENT, 3.4, 90, minRadius=60,
    maxRadius=100)
```

Exemple de résultats

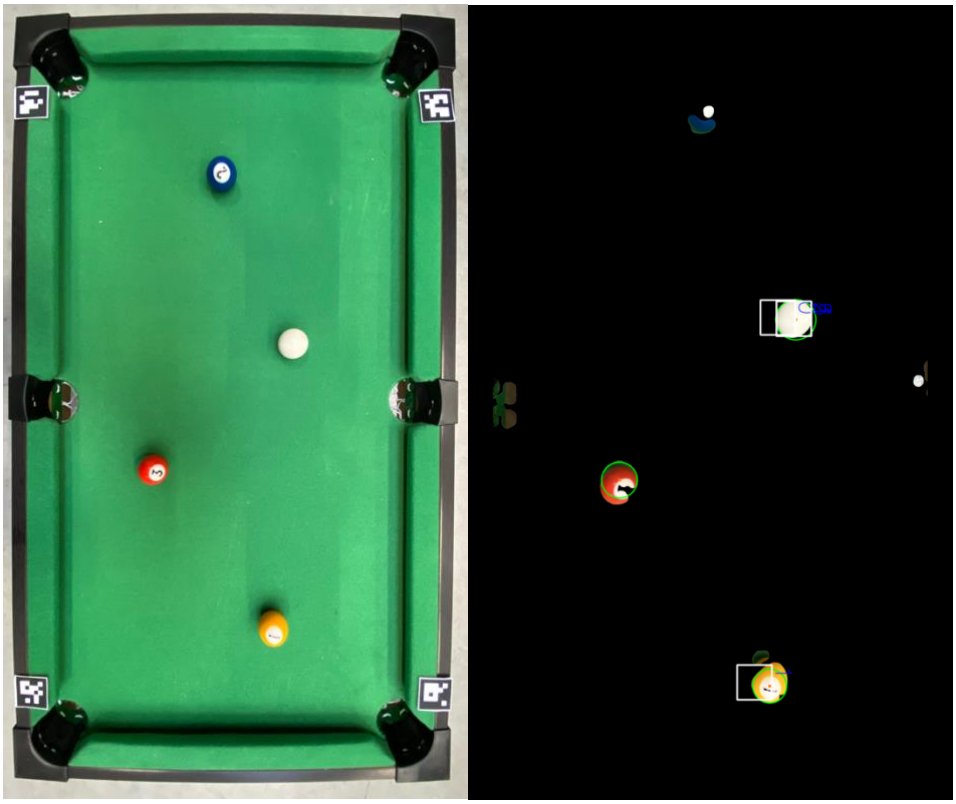


Figure 4.6 résultats du test en lot (exemple 1)

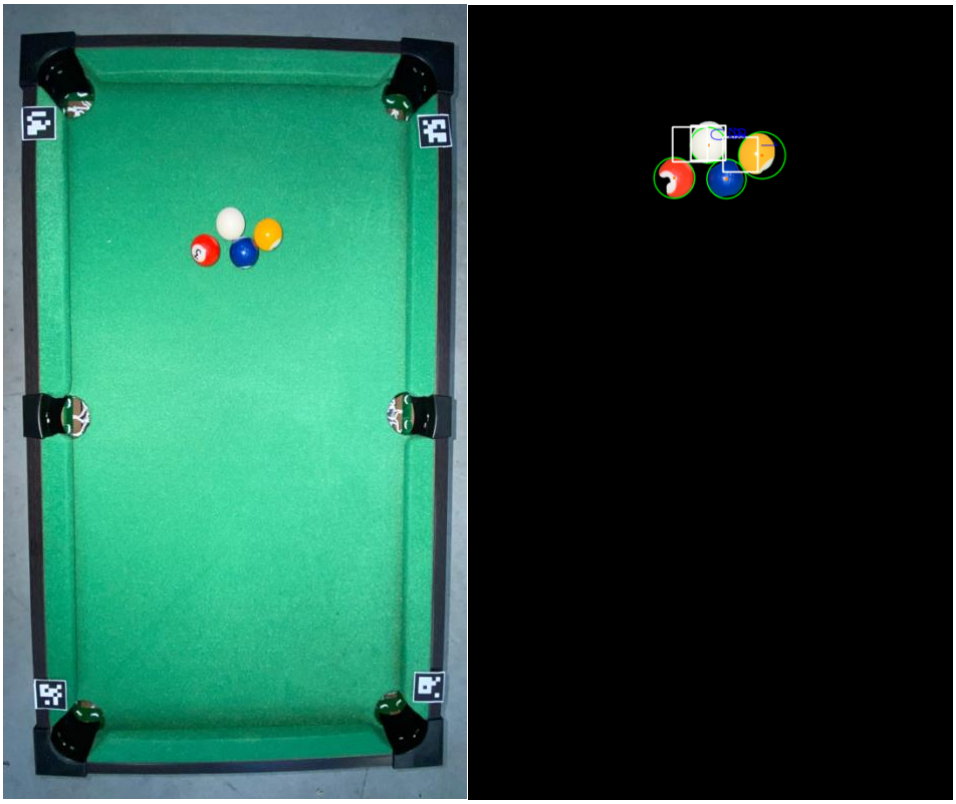


Figure 4.7 résultats du test en lot (exemple 2)

4.3. Conclusion sur la détection des billes

4.3.1. Méthode finale utilisée

En conclusion, ce qui ressort de ces tests, c'est que la méthode de détection avec les cercles et beaucoup plus fiable que la méthode avec le gabarit. Comme dit plus haut, cette méthode permet de détecter toutes les billes en une seule fois puis de définir les couleurs de chacune. Cela évite d'utiliser des gabarits différents pour chaque bille. La méthode de détection des couleurs sera expliquée dans le point suivant.

Cependant, la méthode avec le gabarit ne doit pas être complètement supprimée. Celle-ci deviendrait assez fiable s'il n'y a qu'une seule bille utilisée. Il peut être envisagé d'utiliser cette fonction lorsque la méthode d'entrainement ne demande que la bille blanche. La détection devrait être théoriquement plus rapide et donc fournir une meilleure expérience. Cette option peut être gardée en tête, même si c'est la méthode par cercle qui est maintenue.

Dernière remarque, il ne faut pas oublier que dans les tests faits actuellement, un certain nombre de paramètres sont modifiés afin d'obtenir les meilleurs résultats. Les paramètres notables sont le 'dp' de la méthode cv2.hougcircles ainsi que l'intervalle de valeurs pour la suppression de l'arrière-plan. Il faudrait donc penser à pouvoir changer facilement ces paramètres pour l'utilisateur final. Mais une fois que ceux-ci sont correctement configurés, les résultats obtenus sont très bons.

4.3.2. Détection de la couleur des billes

Comme dit au-dessus, la détection de la couleur des billes est maintenant une étape primordiale au bon fonctionnement. Pour ce faire, la fonction de détection des cercles a été modifiée pour directement détecter la couleur de chaque bille. En sortie de cette fonction, c'est donc une image et un dictionnaire Python (index = couleur de la bille et donnée = coordonnées (x, y) [le rayon n'est pas nécessaire]) qui sont fournis.

```
1.  def __colorInRange(listBGR, colorMIN, colorMAX):
2.      inColorRange = True
3.      for i in range(0, 3):
4.          if colorMIN[i] > listBGR[i] or colorMAX[i] < listBGR[i]:
5.              inColorRange = False
6.      return inColorRange
7.
8.  def circleDetection(imageInput)
9.      [...]
10.     for (x, y, r) in circles:
11.             [...]
12.             offset = 0.9
13.             imgTemp = image[(y - int(r * offset)):(y + int(r * offset)), (x - int(r *
    offset)):(x + int(r * offset))]
14.
15.             # detecting the most dominant color then check for every color
16.             listBGR = imageProcessing.dominantcolor(imgTemp)
17.             listCircles = {}
18.             if (listBGR[0] != 0) and (listBGR[1] != 0) and (listBGR[2] != 0):
19.                 # check if WHITE
20.                 if __colorInRange(listBGR, colWHITEMin, colWHITEMax):
21.                     listCircles["WHITE"] = (x, y)
22.
23.                 # repeat for YELLOW, RED, BLUE
```

Pour déterminer si oui ou non la couleur se trouve dans l'intervalle, la première option fut de vérifier chaque paramètre à l'aide de deux conditions « if » et des « et » logiques. Cette solution est cependant fastidieuse. Le choix final s'est porté sur l'utilisation d'une boucle "for" pour parcourir les 3 valeurs de couleurs et, dès qu'une valeur sort de l'intervalle, alors une condition empêche de déterminer la couleur. Ces étapes sont répétées pour chaque couleur dans une fonction afin d'éviter la répétition.

Un grand nombre de tests ont été nécessaires afin de déterminer correctement les valeurs limites de détection des couleurs. Certains paramètres peuvent fausser les résultats. La détection est assez bonne, mais la détermination de la couleur primaire peut parfois laisser à désirer. Deux causes principales ont été déduites : la couleur détectée étant trop loin de l'intervalle ou les reflets sur la bille. Pour régler le premier problème, la détermination se fait non pas sur toute la détection, c'est-à-dire 2x le rayon, mais sur une zone légèrement plus petite. Pour le second problème, il n'y a pas de réelle solution. Il faudra simplement vérifier que les détections sont correctes. Comme pour les intervalles détection de la couleur de la table, il faudrait permettre à l'utilisateur de faire des tests et modifier ces valeurs selon sa situation spécifique.

Figure 4.8 résultats de détection des couleurs

5. Projection sur la table

5.1. Identification du problème

Comme pour la caméra, il y aura un problème d'alignement entre le projecteur et la table de billard. Les deux ne pouvant se trouver centré au-dessus de la table. Il est évident que le but reste d'aligner le plus efficacement le projecteur afin d'éviter des problèmes de perspective. Il faut tout de même rappeler que certains projecteurs possèdent d'eux-mêmes des corrections d'alignement vertical et/ou horizontal, le « keystone ». Ces alignements peuvent être numériques, ou même optiques sur les projecteurs les plus chers et sont plus avancés. Ils sont donc à privilégier en premier par rapport à la correction dans ce projet.

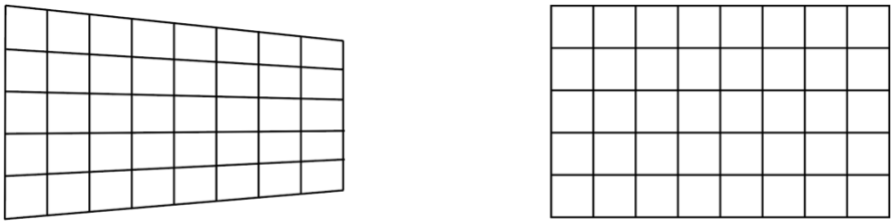


Figure 5.1 correction de la projection

5.2. Correction du problème

Pour la correction de la perspective, le fonctionnement sera pratiquement similaire à la correction faite pour la caméra. La théorie est donc de projeter une image sur la table de billard contenant quatre tags ArUco. Ces tags doivent être de type différent pour pouvoir différencier ceux provenant des coins de la table et ceux provenant de l'image projetée. Une fois tous les tags détectés, il suffit d'utiliser la même logique qu'auparavant en venant « étirer » les centres des tags de l'image sur ceux de la table. Bien que les fonctions soient similaires, il a été décidé de ne pas réutiliser la fonction de correction de perspective. Elles ne sont cependant pas exactement identiques et généraliser la fonction pour les deux cas n'est pas intéressant. Cela rendrait la compréhension plus complexe que nécessaire.

```
1.  # detecting and sorting the tags to get their positions + output of an image
2.      tagListTAB, image = tagDetect(imgIN, p.tagType)
3.      cv2.imwrite(p.kstTagged, image)
4.      tagListPRJ, image = tagDetect(image, p.tagTypePRJ)
5.      cv2.imwrite(p.kstTagged, image)
6.
7.      # tags on the pool table, target positions
8.      original = np.float32([[tagListPRJ["TOP_R"][0], tagListPRJ["TOP_R"][1]],
9.                             [tagListPRJ["TOP_L"][0], tagListPRJ["TOP_L"][1]],
10.                            [tagListPRJ["BOT_L"][0], tagListPRJ["BOT_L"][1]],
11.                            [tagListPRJ["BOT_R"][0], tagListPRJ["BOT_R"][1]]])
12.
13.     # tags on the projected image, initial positions
14.     unwarped = np.float32([[tagListTAB["TOP_R"][0], tagListTAB["TOP_R"][1]],
15.                            [tagListTAB["TOP_L"][0], tagListTAB["TOP_L"][1]],
16.                            [tagListTAB["BOT_L"][0], tagListTAB["BOT_L"][1]],
17.                            [tagListTAB["BOT_R"][0], tagListTAB["BOT_R"][1]]])
```

La première chose intéressante à noter est que la fonction de détection des tags est appelée à deux reprises, mais avec des paramètres différents. Cela sert à détecter dans un premier temps les coins de la table de billard et ensuite les tags posés sur la table. Ensuite deux matrices sont créées en allant chercher les valeurs des centres des tags dans leur dictionnaire python associé. Ce sont là les deux plus grandes différences avec la correction de perspective de la caméra. Dans ce cas, la fonction de détection n'est appelée qu'une fois et la seconde matrice possède des coins fixes, à savoir (0, 0) et (largeur_image, hauteur_image). Il est donc aisé de comprendre qu'il serait peu productif de venir encombrer la fonction principale avec des conditions booléennes.

Pour pouvoir tester si ce code fonctionne, il est nécessaire, pour la première fois, de faire des tests sur le système réel. Jusqu'à maintenant, les tests pouvaient être faits avec des images tests sur des ordinateurs externes. Le besoin du projecteur change cela et donc ces tests ont été réalisés pour la première fois avec la table de billard. (voir chapitre 9 - Tests en conditions réelles)

Avec la projection de l'image, un premier problème est apparu : la couleur de la table rend la détection des tags impossible (voir Figure 5.2). Le système de détection recherche des tags en noir et blanc. De ce fait le vert de la table ne permet pas leur détection. La première solution fut alors de placer des feuilles blanches au moment de la projection pour faire ressortir les tags. Mais là encore, la détection ne se fait pas (voir …). Il est probable que cette solution fonctionnerait avec un projecteur ayant une forte luminosité et avec beaucoup de contraste ou dans une pièce plus sombre.

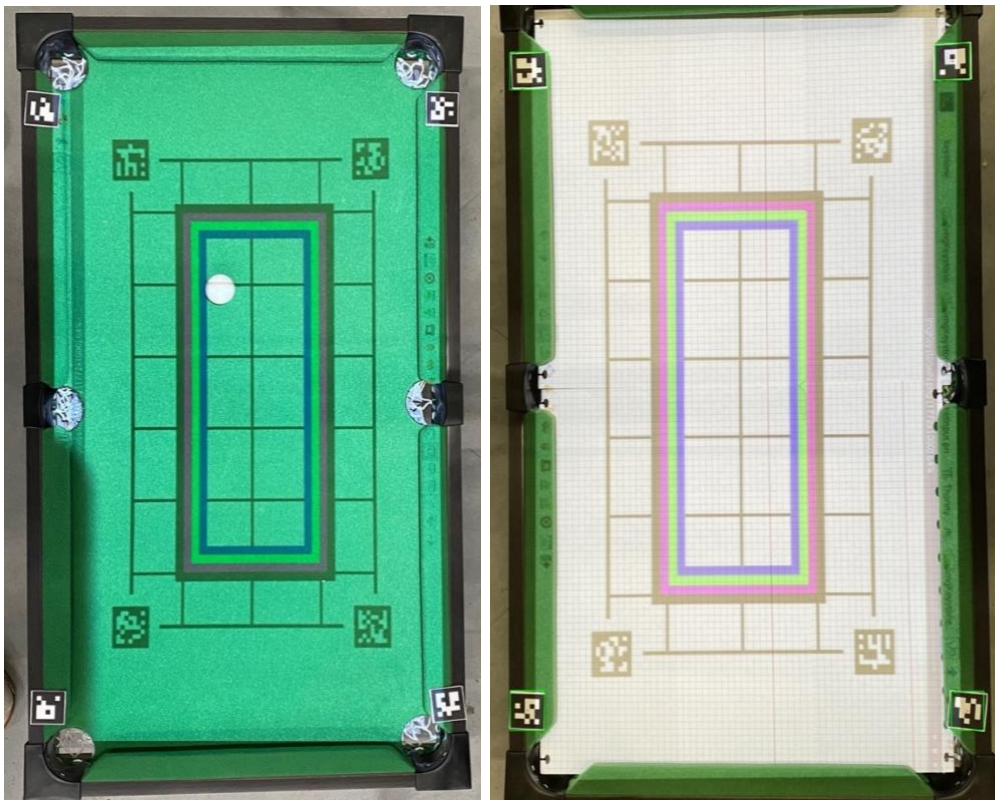


Figure 5.2 correction de la projection : tags sur fond vert et sur fond blanc

La solution finale est de venir placer des tags imprimés et découpés, comme sur la table de billard, aux bons emplacements sur l'image projetée. Cette solution n'est pas la plus élégante, mais fonctionne parfaitement. De plus, cela ne pose pas de problème puisque la correction de la projection ne doit se faire qu'une seule fois : les tags sur la zone de jeu peuvent donc être retirés directement.

Le résultat est une image étirée correctement sur les quatre coins de la table (voir …). Il est important de noter que ces étapes se font avec la caméra et avec une image non corrigée. C'est-à-dire que la perspective de l'image prise par la caméra n'est pas corrigée avant d'effectuer la correction sur la projection.

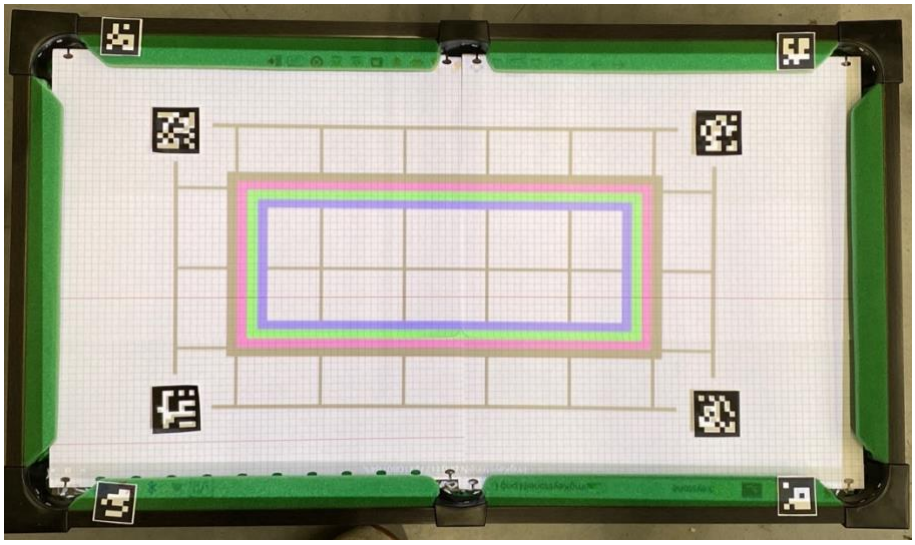


Figure 5.3 correction de la projection : tags imprimés

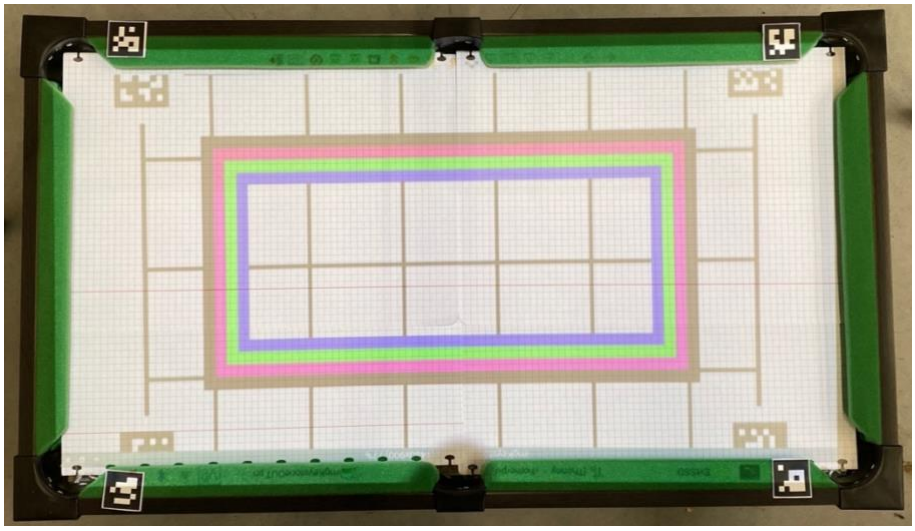


Figure 5.4 correction de la projection : résultats

6. Routines d'entrainements

6.1. Systèmes envisagés

Après avoir réglé tous les paramètres et réalisé toutes les fonctions nécessaires, il est enfin temps de s'attarder sur le cœur de ce projet : les routines d'entrainements. Pour rappel, le but étant de donner à l'utilisateur le choix de plusieurs jeux avec différents cas pour pouvoir s'entrainer. Pour réaliser ceci, deux options ont été envisagées.

Dessin automatique des zones

La première option envisagée est une option purement en programmation. L'idée serait d'avoir une image unie qui serait sauvegardée dans le dossier du projet. De là, le programme va dessiner des cercles sur l'image pour représenter la (les) cible(s) de départ et d'arrivée. Cette méthode est pratique, car les jeux pourraient être facilement modulables, mais la modification de ces jeux ne serait pas très intuitive.

Image spécifique à chaque jeu

La seconde option envisagée est d'avoir une image avec des zones préétablies pour chaque jeu. De là, le système de détection des billes, ici des cercles, est utilisé afin de détecter chaque centre. Seule la zone d'arrivée autour du point final doit être dessinée par le programme, car son rayon peut varier. Cette méthode est modulable, car il suffit de changer les images et le programme fait le reste, et est la plus intuitive pour l'utilisateur.

Comparaison des deux méthodes

Tableau 6.1 Résumé des avantages et inconvénients de la méthode de programmation des jeux

	Avantages	**Inconvénients**
Ajout des cibles par programmation	• 1 seule image • Modulable (dans le code) • Rapide	• Possibles problèmes de résolution • Possibles problèmes de coordonnées avec la correction de perspective • Programmation plus complexe
Images avec cibles et détection des zones	• Modulable (changer les images) • Création de nouveau jeu simple pour l'utilisateur • Programmation facile	• Plus d'images stockées • Possibles problèmes de détection dans l'image

Finalement le choix se porte sur la deuxième option, car elle offre une manière plus intuitive de pouvoir modifier les jeux. Le partage, si une communauté se forme, serait également beaucoup plus simple. En bonus le système devient moins fastidieux dans sa programmation.

6.2. Fonctionnement des jeux

6.2.1. Présentation des trois modes

Le programme n'est pas réellement pensé avec 3 modes de jeu, mais plutôt avec 3 types de configurations de zones et de billes. Ce sont, pour l'instant, au maximum deux billes qui peuvent être utilisées en même temps, à savoir : la bille blanche (principale) et la bille jaune (secondaire) et deux zones d'arrivées maximum. Ces deux billes ont été choisies, car ce sont celles qui offrent les meilleurs résultats en fiabilité de détection.

La zone de validation autour de la zone des zones d'arrivées des billes peut directement être modifiée depuis le menu principal, permettant d'adapter le jeu au niveau de chaque joueur.

Même si seulement deux billes sont utilisées, 4 ont été enregistrées dans la programmation et peuvent être identifiées. Seulement, ça veut donc dire que si quelqu'un souhaite aller modifier le code en rajoutant un jeu, cela peut être fait facilement, soit en changeant les couleurs des billes, soit en réutilisant une des cibles comme nouvelle zone. Ceci est évidemment réservé aux personnes à l'aise avec le code et n'est pas prévu pour être fait par n'importe qui. Mais cela signifie tout de même qu'il n'est pas nécessaire d'aller modifier des parties potentiellement plus sensibles comme le script de traitement d'image.

Sur les images, les cercles jaune et blanc correspondent aux zones de départ des billes de même nom, et les cercles marron et cyan correspondent respectivement à la zone d'arrivée de la balle principale et de la balle secondaire. Sur les figures ci-dessous, les flèches représentent le chemin théorique des billes qui ne sont pas présentes réellement.

Mode 1 : ligne droite

Ce mode de jeu est le mode le plus simple. Seule la bille principale est utilisée. Il y a une zone de départ et une zone d'arrivée. Le but étant de faire parvenir la bille blanche dans la zone d'arrivée le plus précisément possible. Il est laissé libre à l'utilisateur de faire un rebond ou non et de spécifier la taille de la zone d'arrivée.

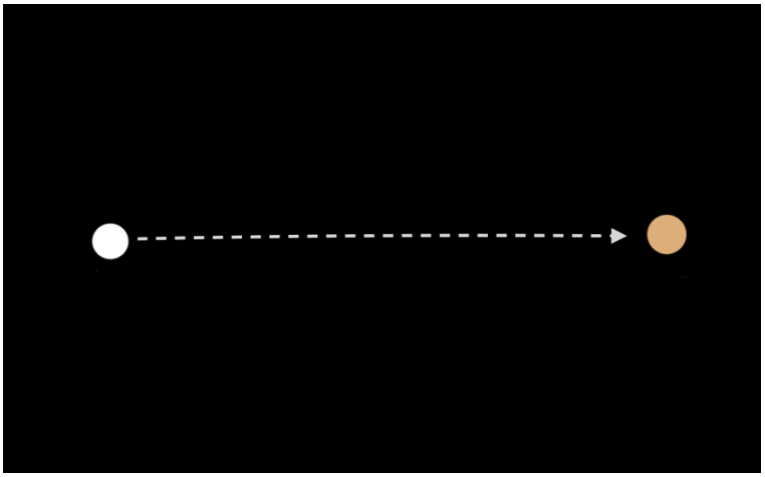


Figure 6.1 Mode de jeu n°1 : ligne droite

Mode 2 : balle immobile

Ce mode ressemble assez au premier cependant cette fois-ci les deux billes sont utilisées. La bille principale doit toujours atteindre une cible, mais la bille seconde, la bille jaune, ne peut pas bouger de son emplacement. Le mode actuel prévoit de placer la bille secondaire entre son point initial et final afin de créer un obstacle, mais là encore, les possibilités sont grandes.

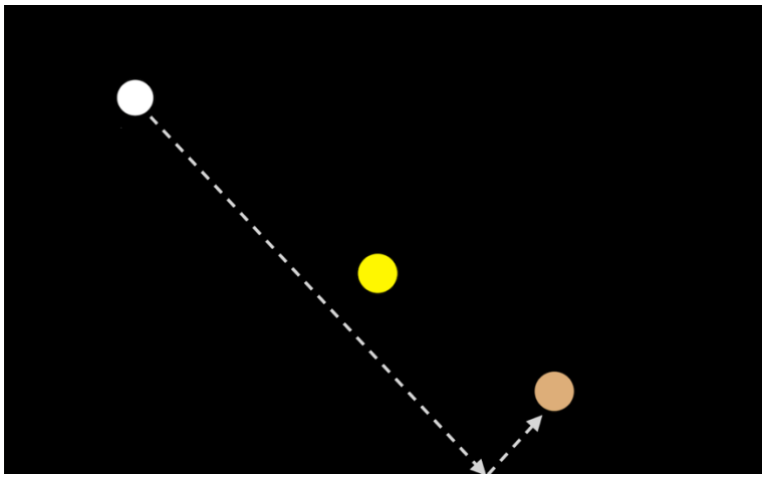


Figure 6.2 Mode de jeu n°2 : balle immobile

Mode 3 : double zone

Ce mode de jeu est le plus complexe et peut-être aussi le plus complet. Il prévoit l'utilisation des deux billes et de deux zones d'arrivées. Actuellement le jeu est pensé pour venir taper la bille jaune avec la balle blanche pour que la première arrive dans la zone cyan et la seconde dans la zone brune.

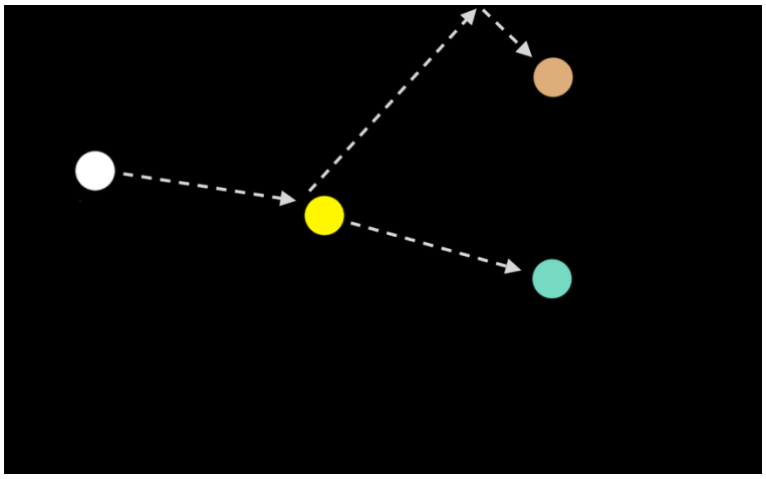


Figure 6.3 Mode de jeu N°3 Double zone

6.2.2. Fonctionnement du système

Étant donné qu'il s'agit du fonctionnement du projet en lui-même, ce pour quoi le projet a été pensé, le code est disponible ci-dessous. Il faut tout de fois noter que, pour des raisons de lisibilité, ce dernier a été raccourci en supprimant par exemple les affichages dans la console et les différentes possibilités en cas d'erreurs de placement ou autre.

Le code ci-dessous se trouve dans la fonction du jeu n°3. Les fonctions des trois jeux sont néanmoins très semblables. Une explication plus détaillée est disponible page suivante.

```
1.  image = cv2.imread(imgGamePath)
2.  image = cv2.resize(image, (p.width, p.height))
3.
4.  # Vérification si la correction de projection a été faite ou non
5.  try:
6.      matrixTransform = loadtxt(p.kstData, delimiter=",")
7.  except IOError:
8.      return 0, -1, "ERROR"
9.
10. # Correction de la perspective
11. image = cv2.warpPerspective(image, matrixTransform, (p.width, p.height))
12. image, listTargets = imgProcess.circleDetection(image, dp=7, minDist=100, minRadius=20,
    maxRadius=80,imgDisplayOut=False)
13.
14. try:
15.     # Dessin des zones autour des cibles finales
16.     cv2.circle(image, listTargets["BROWN"], targetRadius, p.gmColBROWNMax, 4)
17.     cv2.circle(image, listTargets["CYAN"], targetRadius, p.gmColCYANMax, 4)
18.     cv2.imwrite(p.gmToDisplay, image
19.
20.     # Affichage de l'image pour placement initiale des billes
21.     correctPlacement = False
22.     # Temps que le placement n'est pas correct, le programme boucle
23.     while not correctPlacement:
24.         imgShow(p.gmToDisplay)
25.         imgTake(p.pathCamIN)
26.         listBalls = detectBall(p.pathCamIN)
27.
28.         if __inZone(listBalls["WHITE"], listTargets["WHITE"], placementRadius):
29.             if __inZone(listBalls["YELLOW"], listTargets["YELLOW"], placementRadius):
30.                 correctPlacement = True
31.
32.     # Affichage de l'image pour afficher les cibles finales
33.     imgShow(p.gmToDisplay)
34.     imgTake(p.pathCamIN)
35.     listBalls = detectBall(p.pathCamIN)
36.
37.     # Obtention des scores
38.     score = __getScore(listBalls["WHITE"], listTargets["BROWN"])
39.     score += __getScore(listBalls["YELLOW"], listTargets["CYAN"])
40.     score /= 2
41.     gameStat = 0
42.     if __inZone(listBalls["WHITE"], listTargets["BROWN"], placementRadius):
43.         if __inZone(listBalls["YELLOW"], listTargets["CYAN"], targetRadius):
44.             displayInfo = "WIN ! You are the best"
45.             gameStat = 0
46.
47.     return score, gameStat, displayInfo
48.
49. except KeyError:
50.     return 0, -1, "ERROR"
```

Cette fonction ne prend que deux paramètres en entrée, à savoir le chemin d'accès de l'image à afficher et le rayon des zones finales. La première étape qui est faite est de lire cette image est de venir la redimensionner aux dimensions générales de travail du projet. Cette étape est très importante, car juste après, l'image est soumise à la matrice de transformation de perspective. Si les images étaient de tailles différentes, la transformation ne pourrait se faire.

Ensuite, avec les zones identifiées grâce à la fonction 'circleDetection', des zones sont placées autour des cibles finales. L'image est ensuite sauvegardée en tant que fichier « .png » pour pouvoir être affichée par le programme juste après.

Ces actions de lecture et écriture d'image ainsi que le traçage des cercles sont nécessaires à cause du choix qui a été fait de coder le système de jeu. Développer le système alternatif expliqué plus haut pourrait être intéressant.

Une fois l'image obtenue, la fonction affiche l'image en plein écran sur la table de billard et l'utilisateur peut placer ses billes. Dès que la touche 'Enter' est enfoncée (touche par défaut pour quitter une image avec cv2), le programme continue et une image est prise avec la caméra. De là il faut vérifier que le placement des billes est correct. Si oui le programme peut continuer, sinon les opérations précédentes sont relancées.

À ce moment, l'image n'est plus affichée, car il est impossible d'afficher une image et de faire tourner du code en même temps (voir 9.3 Évaluation du fonctionnement). Il faut donc réafficher l'image et le joueur peut alors tirer. Une fois fait, le programme va détecter les billes et leurs centres et les comparer pour obtenir un score. D'autres informations sont également renvoyées pour l'affichage du score (voir chapitre suivant sur le score).

Deux 'try…except' sont utilisés dans cette fonction pour contenir des erreurs bien spécifiques. Le premier sert à vérifier que le fichier de correction de perspective est présent. Sinon, il considère qu'il n'a pas été fait et empêche de faire le jeu. Le second sert au cas où la détection et/ou l'assignation d'une couleur à une bille ne se passeraient pas bien. Cela évite que le programme plante complètement et ne crée pas de problèmes pour le joueur qui pourrait perdre sa partie actuelle.

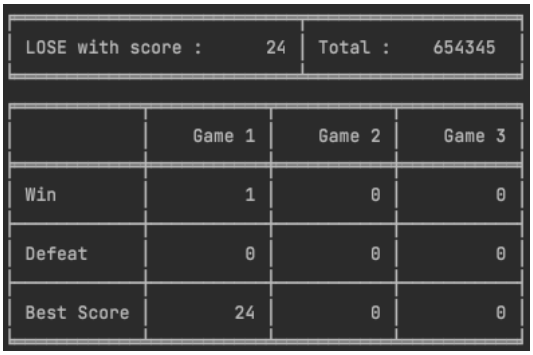
Il pourrait être simple et réducteur de penser, en voyant le morceau de code ci-dessus, qu'il ne s'agit de pas grand-chose. Mais il ne faut pas oublier qu'énormément de codes et de fonctions sont cachés derrière. Cependant, cette partie du code peut être tout à fait comprise par quelqu'un qui n'aurait pas accès aux autres fonctions. Cela signifie, ou en tout cas c'est un signe, que la programmation est claire, modulable et bonne de manière générale.

6.3. Score et suivi d'entrainement.

Pour avoir un système du plus simple, le score est calculé de façon que le meilleur score corresponde au score le plus bas possible. En réalité, le calcul se fait en comparant les coordonnées de la bille par rapport aux coordonnées de la cible. Les différences des valeurs en X et en Y sont calculées et additionnées pour obtenir un score final.

Afin de tout de même avoir plus d'informations sur l'entrainement, ce n'est pas la seule chose qui est faite pour donner une indication de l'évolution / un suivi de l'entrainement à l'utilisateur. La première chose évidente est que chaque partie gagnée ou perdue est enregistrée et sauvegardée jusqu'à une réinitialisation par l'utilisateur ou un arrêt du programme. Le programme enregistre le score total, le nombre de parties gagnées / perdues par type de jeu ainsi que le meilleur score pour chaque jeu.

Cette approche apporte un compte-rendu à l'utilisateur et peu ainsi avoir un suivi sur sa partie. Pour faciliter l'affichage du tableau, la bibliothèque « tabulate » est utilisée. C'est une bibliothèque populaire dans la communauté Python qui permet d'afficher des tableaux avec une vaste sélection de mise en page. L'aperçu est visible sur la figure ci-dessous. Le tableau supérieur est un tableau qui est codé à la main et qui permet de rajouter des informations supplémentaires au tableau comme le score total et l'information si la partie a été gagnée ou non.

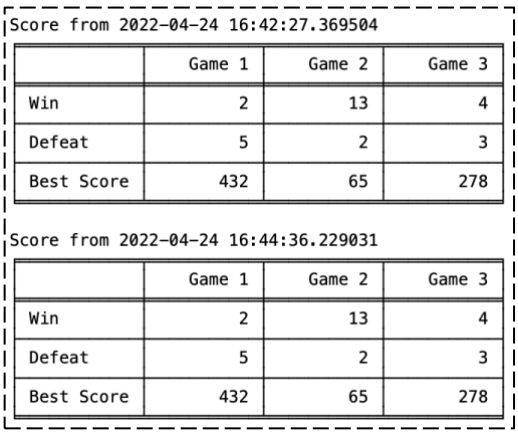


LOSE with score : 24	Total : 654345

	Game 1	Game 2	Game 3
Win	1	0	0
Defeat	0	0	0
Best Score	24	0	0

Figure 6.4 Impression des scores dans la console

Les données de ce tableau peuvent évidemment être réinitialisées depuis le menu principal sans avoir besoin de fermer et relancer le projet. De plus, le tableau récapitulatif peut également être sauvegardé dans un fichier texte. Ce fichier contient chaque tableau enregistré (si le fichier n'est pas supprimé ou déplacé, sinon un nouveau fichier sera automatiquement créé) avec une information de la date et l'heure à laquelle a été jouée la partie.



Score from 2022-04-24 16:42:27.369504

	Game 1	Game 2	Game 3
Win	2	13	4
Defeat	5	2	3
Best Score	432	65	278

Score from 2022-04-24 16:44:36.229031

	Game 1	Game 2	Game 3
Win	2	13	4
Defeat	5	2	3
Best Score	432	65	278

Figure 6.5 Exemple d'affichage des tableaux dans un fichier texte

7. Interface et commande utilisateur

7.1. Souhait original et limitations

La conception d'une interface pratique provient de la façon dont doit être monté le projet en lui-même. Celui-ci sera positionné en hauteur, au-dessus d'une table de billard : l'accès y est donc compliqué. Il ne serait pas pratique d'y attacher un clavier et une souris.

Au commencement du projet, l'idée originale pour l'interface était d'utiliser l'image projetée par le projecteur et la table de billard comme interface et d'utiliser la bille blanche comme pointeur. Ainsi un menu et différentes possibilités auraient été affichés sur la table. L'utilisateur n'aurait alors plus qu'à poser la boule blanche à l'endroit voulu. Le programme aurait alors pu connaitre le choix et agir en fonction.

Cette approche aurait permis de se passer complètement d'un ensemble clavier-souris et d'avoir une interface intuitive, graphique et sympathique pour l'utilisateur.

Seulement plusieurs problèmes se sont posés pendant le développement du projet rendant cette interface impossible à réaliser à l'heure actuelle. Ces problèmes seront survolés ici, mais plus de détails sont disponibles dans le chapitre 9. Ces problèmes sont une impossibilité d'afficher une image et de faire fonctionner du code en même temps et une très grande lenteur pour la vitesse de calcul sur le Raspberry Pi.

À l'heure actuelle, le projet ne tourne que sur un script python principal qui va chercher des fonctions dans des scripts enfants. Et avec cv2, l'affichage d'une image bloque l'avancement du programme jusqu'à ce que l'image soit fermée. Il faudrait alors deux programmes tournant en parallèle.

Il sera également nécessaire d'avoir un ordinateur plus puissant, le traitement d'une seule image prenant actuellement 20 secondes en moyenne. Ce temps rendrait une interface comme expliquée plus haut frustrante voir inutilisable.

C'est pourquoi le choix s'est tourné vers une interface purement console, nécessitant un clavier et une souris. De là il y a deux possibilités pour afficher / contrôler le système. Comme l'unique sortie vidéo[2] est prise par le projecteur, la solution la plus simple est de connecter un clavier et une souris sans fil au micro-ordinateur et de regarder l'écran sur la table de billard. Cette solution est la plus simple, mais pas forcément la plus ergonomique. La seconde solution est d'utiliser le port Ethernet du Raspberry et de le contrôler à l'aide de « VNC connect ». Ainsi, un autre ordinateur peut être connecté à l'autre bout du câble Ethernet et l'écran, la souris et le clavier de l'ordinateur peuvent être utilisés pour contrôler le Raspberry. Cette configuration est plus ergonomique et permet de voir plus clairement les informations.

[2] Il existe une sortie « écran », mais pour des écrans de basses résolutions et faibles dimensions

7.2. Exemples de l'interface console

Les interfaces consoles peuvent être peu intuitives et faire peur aux non-initiés. Il y a donc tout intérêt à essayer de concevoir une interface la plus simple possible et la plus sympathique possible malgré les limitations. De plus, comme le système est assez lent, il faut régulièrement informer l'utilisateur de ce qu'il se passe afin qu'il ne pense pas que le système soit bloqué ou en panne.

Figure 7.1 Exemple d'interface du projet

Le résultat final est une interface pratique, lisible et qui fournit des informations à l'utilisateur, et ce malgré les limitations de l'interface console.

7.3. Modifications des paramètres du projet

Comme il est visible sur la figure ci-dessous et comme il a déjà été évoqué dans ce projet, le paramétrage est très important. Pour donner un exemple concret, l'éclairage où va se trouver l'installation va fortement influencer les couleurs captées par la caméra, donc les valeurs RGB des billes sur l'image. Cela pourrait poser des problèmes à la détection avec les valeurs hautes et basses prédéfinies pour l'association de couleurs. Voilà pourquoi le paramétrage est important.

Pour sauvegarder les valeurs encodées par l'utilisateur, 2 fichiers au format « json » sont utilisés. L'un contient tous les paramètres et l'autre contient les valeurs par défaut des paramètres. C'est dans le premier fichier que le programme lit les valeurs au démarrage et dans ce fichier-là également que l'utilisateur va, à l'aide de l'interface, aller modifier les valeurs. Le second fichier ne sera lu que si l'utilisateur entre le mot clef « default » au moment de définir la valeur du nouveau paramètre.

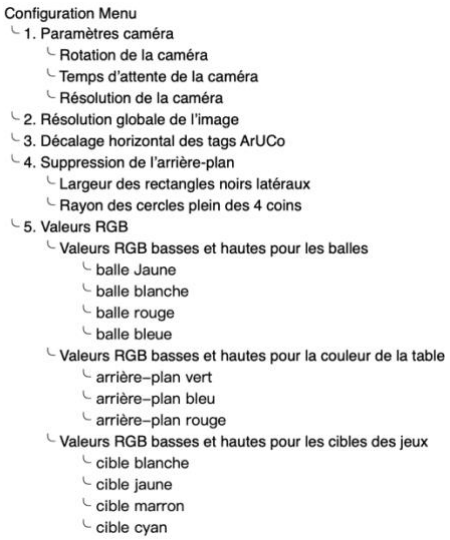


Figure 7.2 Arborescence des paramètres

Le nombre de paramètres est grand et varié : il y a un menu principal, avec plusieurs sous-menus à chaque fois. Il était donc important de trouver des fonctions généralisant le plus possible, en évitant la répétition dans le code et en gardant un aspect graphique clair pour l'utilisateur. Pour cela, une seule et unique fonction a le rôle de modifier les valeurs. La partie graphique est prise en main par des fonctions spécifiques à chaque type de paramètres.

```
1.  def __setInteger(paramValue, paramFile, condValues, strInfo): [...]
2.  def __setResolution(paramValues, paramNames): [...]
3.  def __setRGBThreshold(paramValues, paramNames): [...]
4.
5.  def __modifyValues(paramFile, nbrValues, condValues, strInfo):
6.      inputOK = False
7.      newValues = []
8.
9.      while not inputOK:
10.         strValues = input(strInfo).split()
11.
12.         if not strValues:
13.             inputOK = True
14.             with open("scripts/parameters.json", "r") as jsonFile:
15.                 file = json.load(jsonFile)
16.                 newValues = file[paramFile]
```

```
17.         elif strValues[0] == "default":
18.             inputOK = True
19.             with open("assets/defaultParameters.json", "r") as jsonFile:
20.                 defaultFile = json.load(jsonFile)
21.                 newValues = defaultFile[paramFile]
22.         elif len(strValues) == nbrValues:
23.             for i in range(0, len(strValues)):
24.                 if strValues[i] not in condValues:
25.                     inputOK = False
26.                 else:
27.                     inputOK = True
28.                     newValues.append(int(strValues[i]))
29.
30.     with open("scripts/parameters.json", "r") as jsonFile:
31.         newData = json.load(jsonFile)
32.
33.     newData[paramFile] = newValues
34.
35.     with open("scripts/parameters.json", "w") as jsonFile:
36.         json.dump(newData, jsonFile)
37.
38.     return newValues
```

La fonction prend comme paramètre le nom du paramètre dans le fichier, le nombre de valeurs à modifier, une liste de chaine de caractères avec toutes les valeurs acceptées et la chaine de caractères à afficher dans la console lors de l'acquisition de la valeur.

La fonction va d'abord vérifier que l'entrée de l'utilisateur est correcte : ces options sont la touche 'Enter', le mot clef 'default' ou encore une valeur correcte spécifiée. De là, le programme va effectuer différentes actions en fonction du choix de l'utilisateur. Un point intéressant à noter est le « .split() ». Il est utilisé lors de l'acquisition du changement de la résolution ou d'une valeur RGB : l'utilisateur doit séparer les différentes valeurs avec un espace. C'est donc ici qu'intervient l'utilité du paramètre « nbrValues », car le nombre de valeurs attendues peut changer.

Le fichier « json » est d'abord accessible en lecture afin d'aller retrouver ou enregistrer la nouvelle valeur. Une fois le changement fait, le fichier est de nouveau accédé, mais cette fois-ci en écriture afin de le réécrire complètement. Cela permet de s'assurer que les anciennes données sont bien écrasées et qu'il n'y a pas de doublons.

Après avoir effectué ces différentes actions et avoir assigné la valeur au paramètre, la fonction renvoie la valeur sélectionnée dans la fonction appelante afin de confirmer et d'afficher dans la console la nouvelle valeur.

8. Support de matériel

8.1. Souhait original

Le projet de base est d'utiliser ce système sur une table de billard de grande taille. Il y a cependant beaucoup de matériel à venir positionner au-dessus de la table : le projecteur, la caméra et le Raspberry. En plus de ce qui devait donc être fourni de base pour ce projet, il a été décidé de fournir des plans pour imprimer en 3D des supports pour tous ces composants. L'impression 3D étant de plus en plus répandue, cela permettrait à chacun de facilement installer ce projet. Mais il faut cependant faire une étude des matériaux à utiliser pour choisir le matériau dans lequel le support sera fait. En effet, différentes forces vont agir sur ce support. Et il faudrait que dans le temps le support ne flanche/tombe pas. Le deuxième facteur, est la température, il faudrait un matériau qui dissiperait la chaleur, et qui ne se détériorerait pas à une certaine température. Le poids est également un facteur à prendre en considération, en effet, quel serait le poids d'un projecteur à utiliser pour une table de billard d'une taille standard.

Ce support permet également de correctement positionner la caméra et le projecteur. Ceux-ci doivent nécessairement se trouver le plus possible au centre de la table. Avoir un support commun permet de s'assurer de l'alignement et de coller le plus possible les deux éléments. La nécessité de monter le projecteur à la verticale peut également poser problème : les supports pour le plafond vendu dans le commerce sont généralement faits pour projeter l'image sur un mur. Le support en 3D résoudrait aussi ce problème.

Évidemment, les dimensions de ces plans seraient spécifiques au matériel actuel. Il n'est cependant pas très complexe de modifier des plans si ceux-ci sont correctement fait. Un utilisateur pourrait alors modifier l'installation pour correspondre à ces besoins. Il posséderait néanmoins une base ou une idée à suivre.

L'idée est d'utiliser le filetage typique au caméra présent sur le projecteur utilisait comme élément principal. Cet élément est très commun et l'adaptation pour d'autres projecteur serait donc aisé. Le projecteur serait ainsi mis verticalement, contre une boite. Cette boîte contiendrait le Raspberry et la caméra, réduisant l'encombrement au maximum. L'attache au plafond serait modulaire, pour pouvoir fixer le module sur une surface plane, accrocher à une poutre, etc.

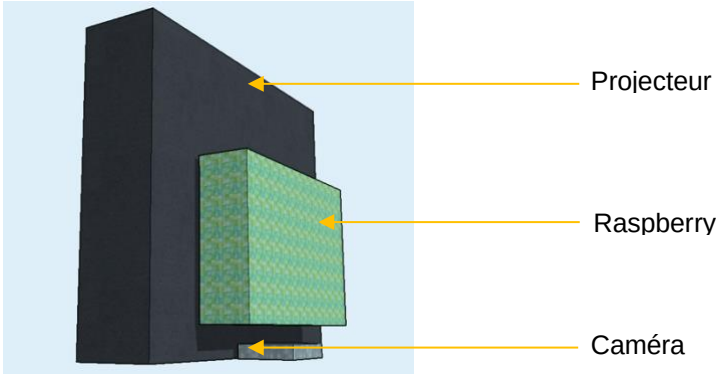


Figure 8.1 visualisation de l'idée originale

La figure ci-dessus représente sommairement l'idée et le placement des composants dans le support.

8.2. Explications du manquement des plans

Malheureusement aucun plan n'a été réalisé. Bien que l'idée de ce support soit une bonne idée, son stade n'a pas dépassé l'état de plan de concept. Le but orignal était tout de même de proposer un prototype fonctionnel et imprimé. Plusieurs facteurs expliquent que ce support n'a pas dépassé le stade de concept.

Le premier est que ce système a été pensée à l'origine pour être utiliser lors des tests en conditions réels. Mais ces tests sont arrivés plus tôt que prévu. Et comme il est indiqué dans le chapitre suivant, une solution de fixation a été trouvée sans avoir besoin d'imprimer un tel support 3D.

Le second est que ces tests ont également montré que le matériel actuel n'était finalement pas adapté pour ce projet. Modéliser des supports et autres prototypes pour du matériel qui devrait de toute façon être changé est tout de suite devenu moins pertinent.

Enfin, l'arrivée de ces tests et la solution trouvée ont fait penser à la personne chargée de cette partie que les plans ne seraient de plus aucune utilité. Et comme cette partie n'avait pas de rapport direct avec le reste de l'avancement du projet, la décision d'abandonner les plans n' a pas été transmise au reste de l'équipe. Un manque de communication est donc également responsable de l'absence de ces plans.

9. Tests en conditions réelles

9.1. Présentation du montage

Présentation générale

Comme expliqué au point précédent, le contrôle se fait à partir d'un ordinateur externe reliée au Raspberry par un câble Ethernet. Tout le système tient sur un trépied d'appareil photo auquel a été montée une structure verticale pour le projecteur et le Raspberry. Comme le projecteur est plus lourd et qu'il est plus loin par rapport au point de fixation du trépied que le micro-ordinateur, un contrepoids est placé sous ce dernier. Une rallonge est fixée sur le trépied afin de ne pouvoir organiser plus facilement les câbles.

Pour faciliter le développement, ce n'est pas une table de billard à taille réelle qui est utilisée, mais une table miniature. Cela facilite à la fois l'acquisition de ladite table, le rangement, l'affichage (à cause de la faible puissance du projecteur) et le développement.

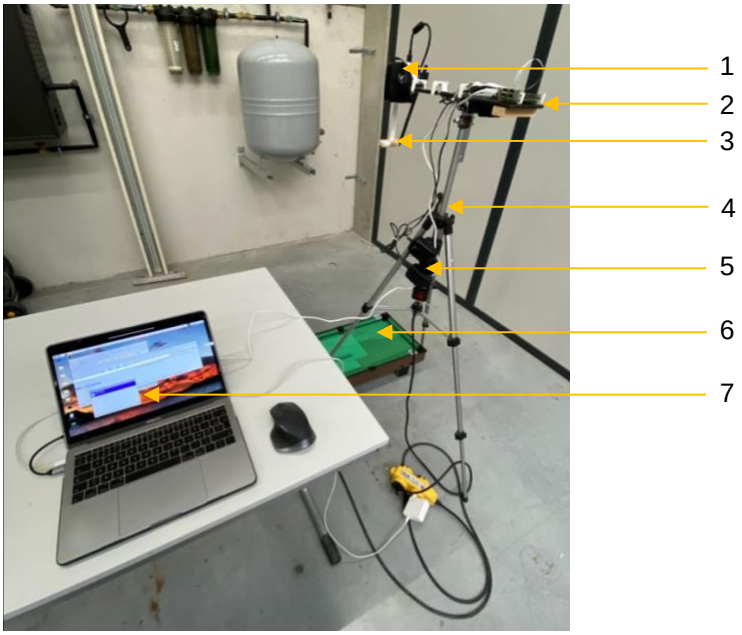


Figure 9.1 Présentation du montage – vue d'ensemble

Tableau 9.1 Identification des éléments du montage – vue d'ensemble

	Identification
01	Projecteur – Médion
02	SBC – Raspberry Pi 3
03	Caméra – Pi Caméra V2.1
04	Trépied d'appareil photo
05	Rallonge électrique
06	Table de billard miniature
07	Ordinateur de contrôle

Le montage ainsi que tous les tests, ont été réalisés dans le bâtiment du CEReF sur le site de l'école d'ingénieur de la HELHa campus Mons. L'éclairage était donc, plus ou moins, contrôlé et assez fort de manière générale.

Raspberry, projecteur et caméra

Le support noir est un support entièrement customisé fait avec des matériaux de récupération. Il n'a donc rien couté. Il sert d'un côté au support pour le Raspberry, et de l'autre à maintenir le projecteur à la verticale et à créer un moyen de centrer et placer la caméra sous le projecteur. C'est également sur ce support que vient se visser le trépied.

Le Raspberry est maintenu en place à l'aide de support imprimé en 3D (jwags55, 2014). Il devait être facilement détachable pour pouvoir être programmé à domicile sans devoir ramener toute l'installation. Une fois le Raspberry attaché, tous ces câbles peuvent être branchés, à savoir le câble HDMI, le câble Ethernet, l'alimentation et enfin la connectique pour la caméra. Cette connectique doit passer de l'avant à l'arrière de l'installation et est assez fragile. Donc pour réduire le risque de problèmes , des supports en 3D créés pour l'occasion ont été réalisés afin de s'adapter sur le corps du montage.

L'attache de la caméra est la moins propre et la moins finie, car ce fut le dernier point. De plus, il aurait été nécessaire d'avoir quelque chose de modulaire pour faire les tests. Il a donc été jugé plus judicieux de se concentrer sur le fonctionnement du projet que sur l'esthétique du montage de test.

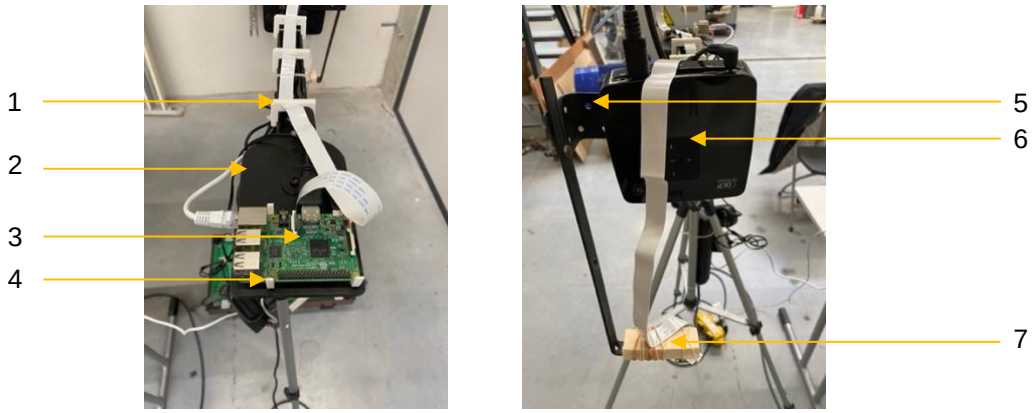


Figure 9.2 Présentation du montage – composants

Tableau 9.2 Identification des éléments du montage – vue d'ensemble

	Identification
01	Support de connectique caméra imprimé en 3D
02	Squelette / Structure du montage (support Raspberry)
03	SBC – Raspberry Pi 3
04	Support pour Raspberry Pi 3 imprimé en 3D
05	Squelette / Structure du montage (support vertical projecteur et caméra)
06	Projecteur – Médion
07	Caméra – Pi Caméra V2.1

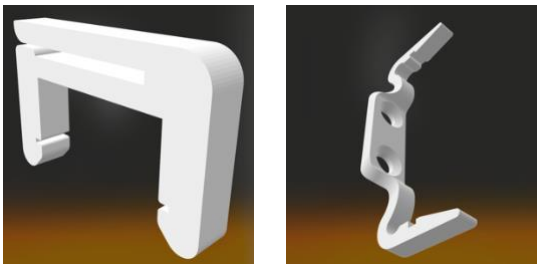


Figure 9.3 Modèles 3D du support de câble (à gauche) et du support pour Raspberry (à droite)

9.2. Évaluation du matériel

Single board controller

Les tests ont montré que le choix du Raspberry Pi 3 ne fut probablement pas la meilleure idée. Celui-ci possède finalement une puissance de calculs très lente pour le traitement d'image. Pour expliquer ces propos, voici deux exemples parlants.

Le premier est au niveau de la prise de photo. Premièrement celui-ci n'accepte pas la résolution maximale de la caméra, à savoir 3280x2464 pixels. Il faut donc la baisser. Mais également le temps d'encodage de l'image est très long et peut prendre entre 3 et 5 secondes simplement pour enregistrer l'image. Et ce, sans devoir ajouter un retard dans le programme permettant à la caméra de s'acclimater en cas d'environnement dynamique. Il est donc conseillé de réduire au maximum la résolution de la caméra, sans détériorer la qualité du programme, pour améliorer la rapidité. Ce n'est clairement pas idéal.

Le second problème est simplement au niveau de la suppression d'arrière-plan. Ici, une boucle de 15 tests complets (méthode similaire à celle utilisée au point 4.2). La suppression de l'arrière-plan se fait en allant vérifier chaque pixel de l'image pour voir si oui ou non il se trouve dans l'intervalle de couleur à supprimer ou non. Cette méthode est obligatoire pour le bon déroulement de la détection de billes et il est compliqué de l'optimiser. La solution la plus simple pour l'accélérer et de diminuer la taille de l'image. Mais alors dans ce cas, les risques de mauvaise détection des billes augmentent. Les temps ci-dessous sont pour une image de traitement de 1450x900 pixels.

```
[info] ending batch testing
– total time  : 0:04:52.701857
– average time : 0:00:19.513457
– t max (N°11)  : 0:00:29.404029

>>>
```

Figure 9.4 temps de suppression de l'arrière-plan

Caméra d'acquisition

La caméra est plus que correcte. La résolution maximale de la caméra fait que les images peuvent être extrêmement nettes (Figure 9.5 Exemple de qualité d'image possible avec la caméra). Même si, comme dit plus haut, le Raspberry actuel ne supporte pas une résolution aussi haute. Il faut également rappeler que les tests ont été réalisés dans une pièce très bien éclairée. Il faudrait faire des tests complémentaires pour savoir si cette caméra pourrait être utilisée à plus faible luminosité. Mais ceci est peu probable vu la taille du capteur.

Un autre point est survenu sur la non-détection des tags. Comparativement aux photos prises avec le téléphone lors des tests en condition réelle, le système n'arrivait pratiquement jamais à détecter les tags de la table ou de la zone de jeu. Il a été nécessaire d'agrandir le contour blanc des images afin que la caméra puisse bien voir le contraste et détecter les images. Ce défaut est vraisemblablement dû à la qualité nécessairement basse de la caméra. Mais ce n'est, en soi, pas un problème grave ou important.

Projecteur

Un autre point négatif au niveau matériel pour ce projet et le projecteur. Ce n'est clairement pas un projecteur de bonne qualité. Même en considérant que la surface de projection est verte et donc loin d'être idéale, le contraste, la résolution et la luminosité ne sont pas bons. Pour les couleurs ce n'est pas un problème, car la surface de projection ne permet de toute façon pas un bon affichage. Néanmoins, les autres paramètres rendent, à l'heure actuelle, la navigation dans le projet inutilisable sans un écran externe.

Figure 9.5 Exemple de qualité d'image possible avec la caméra

Conclusion sur le matériel

En conclusion de cette évaluation de matériel, il faut premièrement retenir qu'il s'agit de matériel ayant été récupéré sur d'autres projets ou acquis par le(s) superviseur(s). Le budget matériel pour ce projet est donc proche de zéro, ce qui est un atout considérable dans un projet essentiellement basé sur la recherche comme c'est le cas ici.

Il s'agit ici de matériel correct pour la première phase de développement. Mais bien qu'utilisable, le système pourrait être grandement amélioré en changeant quelques composants comme le Raspberry Pi et le projecteur.

Le remplacement de ces deux éléments augmenterait l'efficacité et la fluidité du projet sans réellement toucher au code. Cela permettrait également de nouvelles fonctions comme une interface plus graphique, une meilleure détection des balles, un programme plus rapide, etc. Le jeu et le projet deviendraient alors beaucoup plus agréables à jouer.

9.3. Évaluation du fonctionnement

Temps de détection

Le second point qui rejoint l'évaluation du matériel et le temps nécessaire global pour la détection des billes. Le temps actuel est excessivement long et rend le jeu assez désagréable à jouer. L'attente de 20 secondes pour savoir simplement si les billes sont bien placées ou non n'est pas des plus optimales. Comme dit plus haut, ce problème vient notamment du Raspberry Pi 3B+ utilisé. Cependant, il est fort probable que ce temps soit causé par la fonction qui détecte la couleur dominante de la table et qui supprime l'arrière-plan. Dans le code donc.

Cette fonction utilise la méthode des k-means pour détecter la couleur dominante de la table. Ensuite, en fonction de la couleur de la table, le système va vérifier la couleur de chaque pixel de l'image et la comparer à une valeur interne. Si le pixel est dans cette plaque de couleur, celui-ci devient noir. Cette fonction a initialement été imaginée de la sorte afin d'éviter à l'utilisateur d'encoder la couleur de la table. Après avoir fait ces tests, il parait évident qu'il n'est pas nécessaire de réévaluer ce paramètre entre chaque détection. Puisque la couleur de la table ne va pas changer. De plus, si le système est déplacé, la correction du projecteur doit de toute façon être refaite. Encoder de nouveau la couleur de la table à chaque remise à zéro, et non à chaque lancement du programme, serait plus que convenable.

Pour vérifier ces dires, un test de détection sur 18 images (méthode similaire à celle utilisée au point 4.2) a été fait en supprimant la vérification de la couleur.

Tableau 9.3 Influence du k-means sur le temps de détection

	Avec détection de la couleur		**Sans détection de la couleur**	
	T. total [s]	T. moy. [s]	T. total [s]	T. moy. [s]
Essai 1	24,24	1,35	22,7	1,26
Essai 2	23,41	1,30	22,8	1,27
Essai 3	27,2	1,51	23,82	1,32
Moyenne	24,95	1,39	23,11	1,28

Malheureusement, ces tests n'ont pu être effectués sur le Raspberry Pi par manque de temps. Ils sont donc faits sur l'ordinateur de développement. Malgré la faible différence, il y aurait donc un gain de temps réel à gagner en modifiant cette fonction. Ce gain serait sans doute d'autant plus important sur le Raspberry vu sa faible puissance. Il serait aussi intéressant de jouer sur les paramètres du k-means. Il faut néanmoins remettre les choses dans leur contexte : le point précédent a montré qu'il faudrait de toute façon changer le micro-ordinateur pour d'autres raisons, et la détection des couleurs ne peut être entièrement supprimée, car elle est utilisée pour la détection de la couleur des billes.

Détection des couleurs

Le point qui pose encore le plus de problèmes est la détection des couleurs. Environ entre 20 et 10% du temps, le système ne marche pas, car la détection des couleurs ne se fait pas correctement. Les billes elles sont correctement détectées. Pour preuve, voici une image de la détection d'une bille au bord de la table de billard :

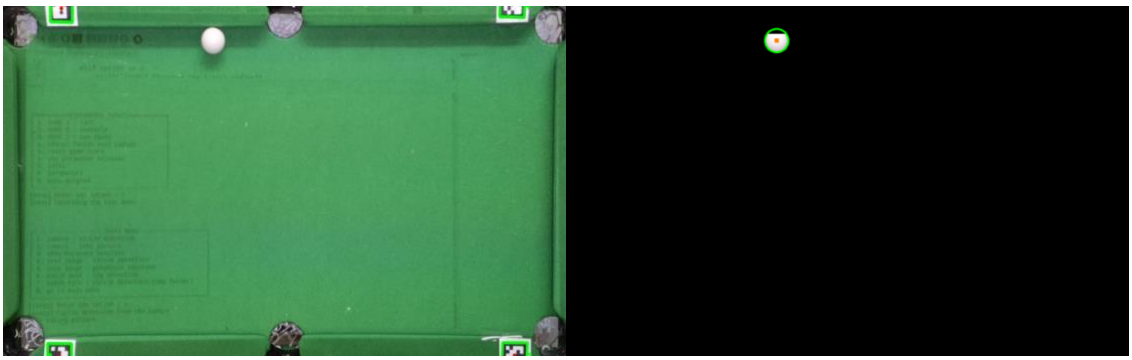


Figure 9.6 Détections sur le bord de la table

Le cas ci-dessus a été obtenu en essayant un des jeux. Le système n'a pas détecté et a donc stoppé le jeu pour revenir au menu principal. Pourtant la détection se fait bien ,mais l'association des couleurs pose de sérieux soucis. Lors de détections, les valeurs RGB détectées pouvaient dans certains cas varier de plus de 30. Pourtant, la méthode k-means effectue plusieurs itérations en donnant la meilleure valeur. Les plages de couleurs de détections sont également très grandes. La solution simple serait de les agrandir à chaque problème, mais les chances alors que les couleurs se superposent augmentent.

Ces problèmes de détections sont à la fois valables pour les billes, mais aussi pour les images des jeux. Les arguments de changement de luminosité ou autres facteurs pourraient être retenus pour la couleur dominante des billes, autant ce n'est pas le cas pour les images de jeux. Celles-ci n'ont aucune raison d'évoluer.

BILLE BLANCHE	BILLE JAUNE	ZONE BLANCHE	ZONE JAUNE	ZONE BRUNE	ZONE CYAN
200 ,200 ,200	150, 130, 0	200 ,200 ,200	230, 230, 0	170, 130, 80	60, 140, 130
255, 255, 255	255, 255, 100	255, 255, 255	255, 255, 25	225, 180, 130	225, 180, 130

Figure 9.7 Plages de couleurs utilisées

La figure ci-dessus montre les plages de couleurs pour chaque couleur utilisée. Même avec des plages assez grandes, des erreurs peuvent tout de même apparaitre. Malgré cela, comme dit plus haut, le système fonctionne plus souvent qu'inversement. Si jamais il y a une erreur, celles-ci sont contrôlées par le code, ce qui ne plante pas le programme et permet à l'utilisateur de continuer sa partie. Donc ce système, bien que fonctionnel, pourrait être révisé.

10. Améliorations futures

Comme pour tout projet de recherche, plusieurs itérations sont à faire avant d'avoir un premier vrai système fonctionnel. Et même ce système final peut être remplacé par des versions antérieures, plus complètes. Mais quel et le bilan de ce projet et quels sont les points à améliorer dans la version actuelle ?

La première chose est au niveau du matériel. Comme dit dans le point 9.2, un matériel de meilleure qualité permettrait grandement de rendre ce projet plus viable. Les deux éléments les plus intéressants à changer sont le projecteur et le Raspberry. Il est important d'avoir un meilleur contraste pour le projecteur, et un Raspberry plus puissant permettrait d'utiliser la caméra dans sa meilleure plage de fonctionnalité. Pour le Raspberry, étant donné la charge de calcul qui lui ait demandé et, en supposant sur de longues sessions d'entrainement, il pourrait également être nécessaire de le refroidir activement avec un ventilateur et un dissipateur de chaleur. En effet, à partir de 85 °C, la fréquence du processeur est limitée afin de se protéger. Il faudrait également vérifier et/ou adapter le système pour une taille réelle. Mais ce point est lié à l'amélioration des composants.

Le deuxième point est au niveau du jeu. Actuellement, il est impossible d'afficher une image sur la table de billard et de faire tourner le script en même temps, ce qui casse un peu l'immersion. Il faudrait alors décomposer le projet en deux scripts parallèles, voir en deux micro-ordinateurs communiquant. L'un pour l'affichage et l'autre pour les calculs. Une telle chose rendrait le système beaucoup plus agréable à jouer.

Ceci rejoint le point suivant sur l'interface graphique. Avoir une interface beaucoup plus graphique plutôt que des lignes de codes , ça serait la prochaine étape. Même si l'idée n'est pas poussée aussi loin que d'utiliser la bille blanche comme pointeur et qu'un ensemble clavier-souris est nécessaire, pouvoir utiliser que la surface projetée serait beaucoup plus simple.

La troisième chose est au niveau du fonctionnement du code. Pour la correction de la perspective des images prises et des images projetées, il faudrait pouvoir réussir à calibrer et faire une correction sans simplement étirer l'image. Cela augmenterait grandement la précision et donc la qualité des entrainements. Évidemment, cette étape ne servirait pas si la caméra et l'ordinateur ne permettent pas une telle précision.

La dernière chose serait de revoir et terminer le support en 3D. Il s'agit d'un réel plus dans le développement de ce projet vers quelque chose de transportable et facilement exploitable par n'importe qui. Il faudrait certainement revoir le concept en créant quelque chose d'encore plus modulable ou adapté au nouveau matériel.

Enfin, d'autres fonctionnalités pourraient être rajoutées comme la possibilité de s'entrainer à plusieurs personnes à distances, des modes de jeux différents, la capacité de changer le style de billes utilisées, etc.

Conclusion

Voici la conclusion de ce rapport de projet de recherche et développement réalisé dans le cadre du cursus académique de 1re année de master en sciences de l'ingénieur industriel. Pour rappel, le projet DEADPOOL fut initié par M Vachaudez et M Estivenart et confié à Mulnard Théo et Talbi Sara. Il s'agit essentiellement d'un projet de recherche, ayant pour but un système d'entrainement autonome basé sur une caméra et un projecteur.

Après ces multiples mois de travail, le projet est accompli et fonctionnel. Le résultat est bien un système composé d'une caméra, d'un projecteur et d'un micro-ordinateur. Plusieurs modes de jeux sont disponibles et un suivi du score est présent. Le code, ainsi que la documentation permettant de réaliser soi-même ce projet, est totalement open source. Le but principal de ce projet a été atteint, même s'il a fallu revoir à la baisse certaines attentes.

En effet des problèmes de matériel ont forcé le développement à être revu à une échelle plus basse. Le point le plus visible a été de passer de l'envie d'un système sur une table taille réelle à un système miniature. Ces changements ont été causés par un matériel trop peu puissant et une non-praticité du développement s'il avait fallu utiliser une table de taille réelle. Néanmoins, ce n'est absolument pas un problème pour un projet de recherche. Les résultats obtenus montrent que le système fonctionne et est viable. Une adaptation du projet pour une table plus grande peut donc tout à fait être envisagée.

Comme il était prévu au départ, c'est bien la partie détection des billes qui a pris le plus de temps. Cette partie est la plus complexe et il s'agit de celle ayant pris le plus de temps en développement : à savoir le dernier trimestre de 2021 et des ajustements pendant les premiers mois de 2022. Et encore à l'heure actuelle, ce système n'est pas parfait. Il faudrait effectuer beaucoup de tests et jouer énormément sur les réglages pour obtenir un taux de réussite de 100%. Le système actuel n'est cependant pas pénalisé, car les erreurs sont gérées et cela ne pose pas de problèmes au potentiel utilisateur.

La correction de perspective de l'image est également un point intéressant dans ce projet. Celle-ci est pleinement suffisante pour l'instant, mais pourrait être adaptée pour une plus grande précision.

Sur les trois objectifs fixés au mois de janvier, seul deux ont pu être réalisés : à savoir l'affichage d'une image avec la perspective corrigée sur la table et la création de plusieurs modes de jeu. Seule la création du support pour l'ensemble caméra, Raspberry et projecteur n'a pas être créé. Ces objectives ont été fixés après les premiers mois de travail car il devait permettre de se rendre du travail à accomplir et donc de se fixer des objectifs atteignables. Et pourtant le troisième objectif n'a pu être réalisé.

Bien que ce soit le moins important des trois, son absence reste un problème. L'explication a déjà été donnée dans le chapitre dédié à ce support, mais l'explication vient principalement du matériel. Les premiers tests ayant montrés que ceux-ci n'étaient pas adapté pour le futur, il a semblé moins pertinent de pousser la création de modèle 3D. Ces modèles auraient de toute façon été modifiés voir complètement refait lors d'une réitération de ce projet. De plus, ces supports n'ont pas été nécessaire pour effectuer les tests.

En conclusion, dire que ce projet est fini serait faux, dire que cette première version est finie est plus juste. Il est certain que des améliorations peuvent être apportées, mais cette version est une base de travail fonctionnelle pour quiconque voudrait un système similaire. De plus, comme il s'agit d'un code open source, les intéressés pourront apporter des modifications afin de l'améliorer ou de le modifier selon les convenances. Il est possible d'imaginer qu'une communauté se formerait autour de ce projet.

Et d'un point de vue personnel, ce projet a permis d'en apprendre, voire d'apprendre, le langage de programmation python, d'utiliser le traitement et la manipulation d'image, d'utiliser la lecture et la sauvegarde de fichiers, etc. Ce fut un projet très intéressant et instructif à la fois d'un point de vue du thème en lui-même et de l'apprentissage.

Bibliographie

Anonym, F. Q.-R. (2017, décembre). *Laptop : minimum system requirement for OpenCV*. Consulté le novembre 2021, sur answers.opencv.org: https://answers.opencv.org/question/179923/laptop-minimum-system-requirement-for-opencv/

jwags55. (2014, 10 24). *Raspberry Pi Mount.* Récupéré sur Thingiverse: https://www.thingiverse.com/thing:513629

MEDION. (s.d.). *Mini projecteur à led - mode d'emploi.* Récupéré sur Mini projecteur à LED - MEDION LIFE: https://download2.medion.com/downloads/anleitungen/bda_md22900_be_(fr).pdf

OpenCV. (2021, Décembre). *Detection of ArUco markers*. Récupéré sur docs.opencv.org: https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html

OpenCV. (2021, décembre). *Feature Detection - HoughCircles().* Récupéré sur docs.opencv.org: https://docs.opencv.org/4.x/dd/d1a/group__imgproc__feature.html#ga47849c3be0d0406ad3ca45db65a25d2d

OpenCV. (2021, Décembre). *Template Matching*. Récupéré sur docs.opencv.org: https://docs.opencv.org/4.x/d4/dc6/tutorial_py_template_matching.html

Rosebrock, A. (2020, Décembre). *Detecting ArUco markers with openCV and Python*. Consulté le Novembre 2021, sur pyImagesearch: https://www.pyimagesearch.com/2020/12/21/detecting-aruco-markers-with-opencv-and-python/

StackOverflow. (2018, juin). *Most dominant color in RGB image - OpenCV / NumPy / Python - response by zindarod.* Consulté le décembre 2021, sur stackoverflow: https://stackoverflow.com/a/50900494

StackOverflow. (2019, Juillet). *How can I remove blue background color on image using OpenCV ? - response from nathancy.* Consulté le décembre 2021, sur stackoverflow: https://stackoverflow.com/a/56878194

Stuff Made Here. (2021, février 15). *Automatic pool stick vs. strangers*. Récupéré sur Youtube: https://www.youtube.com/watch?v=vsTTXYxydOE