Page de garde

# Table des matières

[Table des matières 2](#_Toc122472384)

[Introduction 4](#_Toc122472385)

[Description du projet 4](#_Toc122472386)

[Description du système 5](#_Toc122472387)

[Projecteur utilisé 5](#_Toc122472388)

[Raspberry Pi 5](#_Toc122472389)

[Caméra 5](#_Toc122472390)

[… 5](#_Toc122472391)

[Travail réalisé par le groupe de l’année précédente 6](#_Toc122472392)

[Objectifs pour la fin du Q1 7](#_Toc122472393)

[Problématiques rencontrées 9](#_Toc122472394)

[Détection des couleurs 9](#_Toc122472395)

[Détection des billes (cercle) 11](#_Toc122472396)

[Détection des trous (cercle) 13](#_Toc122472397)

[Détection des limites du billard (ligne) 13](#_Toc122472398)

[Détection des couleurs et des billes en même temps 13](#_Toc122472399)

[Calibration de la caméra 14](#_Toc122472400)

[Raspberry Pi pas suffisante 🡪 Solutions possibles 14](#_Toc122472401)

[Utilisation de Linux pour plus d’audience 14](#_Toc122472402)

[Pistes d’amélioration futures 15](#_Toc122472403)

[Explication de la détermination du programme d’entraînement 16](#_Toc122472404)

[Choix de plusieurs difficultés 16](#_Toc122472405)

[Choix du type d’exercices 16](#_Toc122472406)

[Collé bande 16](#_Toc122472407)

[Bille cachée 16](#_Toc122472408)

[Défense 16](#_Toc122472409)

[Line-Up 16](#_Toc122472410)

[Carré 16](#_Toc122472411)

[Replacements 16](#_Toc122472412)

[Contrôle de la force 16](#_Toc122472413)

[Angles d’attaque 16](#_Toc122472414)

[Rétro 16](#_Toc122472415)

[Coulé 16](#_Toc122472416)

[Carreaux 16](#_Toc122472417)

[… 16](#_Toc122472418)

[Conclusion 17](#_Toc122472419)

# Introduction

## Description du projet

# Description du système

## Projecteur utilisé

## Raspberry Pi

## Caméra

## …

# Travail réalisé par le groupe de l’année précédente

# Objectifs pour la fin du Q1

Pour commencer, plusieurs ??? nous ont été imposés en ce début de projet. Nous devions réaliser un programme d’entraînement en temps réel pour les joueurs de billard voulant s’entraîner efficacement ou en s’amusant, à l’aide de mini-jeux. Pour arriver à cela, il nous est demandé impérativement de coder un programme à l’aide du langage Python. De ce fait, nous avons dû apprendre par nous-mêmes ce langage de programmation afin d’arriver aux fins de ce projet. Nous devions aussi utiliser le système d’exploitation Linux afin que ce projet soit « Open source », c’est-à-dire que le titulaire du droit d'auteur accorde aux utilisateurs le droit d'utiliser, d'étudier, de modifier et de distribuer le logiciel et son code source à quiconque et à n'importe quelle fin. Cela permet donc à n’importe quelle personne voulant utiliser ce programme, de voir le matériel nécessaire pour son utilisation et els étapes à suivre pour y arriver.

La première étape de ce projet a été de découvrir le travail effectué par le groupe d’étudiants de l’année précédente. Cela nous a permis de voir les éléments qui y fonctionnaient correctement, pouvant donc être repris directement afin de gagner du temps dans la réalisation du projet, mais aussi les parties n’ayant pas donné de résultats concluants, qui devront donc être améliorées, voire changées totalement.

Concernant le matériel utilisé, certains composants ne correspondaient pas aux performances attendues. Ce fut le cas du projecteur et de la Raspberry Pi. Le premier d’entre eux n’était pas assez puissant. De ce fait, l’image projetée n’était pas assez lisible. Une solution simple était de prendre un projecteur d’une gamme supérieure afin que les performances souhaitées soient atteintes. Pour le second, le modèle de Raspberry Pi utilisé manquait de puissance. Suite à cela, le programme permettant le traitement d’image et les calculs du programme d’entraînement créé mettait un certain temps avant d’envoyer une réponse. L’expérience de l’utilisateur n’était donc pas optimal suite aux moments d’attente paraissant incessants. Pour ce problème, plusieurs solutions étaient possibles. La marque Raspberry Pi proposant plusieurs gammes de leur produit, ayant des performances plus ou moins grandes, il aurait été possible de se tourner vers un modèle plus adapté aux résultats attendus.

Concernant la programmation en elle-même, nous avons fait une liste de tous les logiciels à développer qui pourraient nous être utiles pour la suite du projet. Parmi ceux-ci, nous pouvons retrouver les suivants :

* Programme 1 : Détection des billes grâce à leur couleur

Ce premier programme a été élaboré dans le but que, durant les séances d’entraînement, les billes puissent être détectées en fonction de leur couleur (jaune, rouge ou blanche). Cela permettra de savoir si la bonne bille a été rentrée, et non une bille de mauvaise couleur.

* Programme 2 : Détection des billes grâce à leur forme circulaire

Ce second programme permet de déterminer la position des objets circulaires présents sur une image. Grâce à cela, il est possible de trouver la localisation de chacune des billes, ainsi que des poches du billard. Cela pourra nous être grandement utile pour la suite du programme d’entraînement lorsqu’il faudra calculer le point d’impact sur la bille à jouer pour la faire tomber dans un trou du billard.

* Programme 3 : Détection des billes à l’aide des deux méthodes précédentes (cercle + couleur)

Celui-ci permet de faire un mixte entre les deux premiers programmes réalisés. Ce troisième programme permet de se baser sur les masques de couleur sortant du programme 1. En effet, ayant eu quelques problèmes au niveau du paramétrage du programme 2, qui détectait des cercles à des endroits vides de la table, ce programme 3 permettrait d’éliminer ce genre de parasites.

* Programme 4 : Suppression du fond de l’image

L’idée de ce programme était de supprimer de l’image principale (le billard **avec** les billes) l’image de fond (le billard **sans** aucune bille dessus). Le logiciel vient remplacer chaque pixel identique se trouvant sur les deux images par un pixel noir. Il n’en ressort donc que les pixels des billes. Le problème de ce programme est que, pour les billes foncées telles que la noire et les mauves, leur détection reste assez compliquée. En effet, le fait de remplacer l’image de fond pour des pixels noirs ne permet pas de faire la différence entre les pixels noirs du fond et ceux de la bille noire, par exemple.

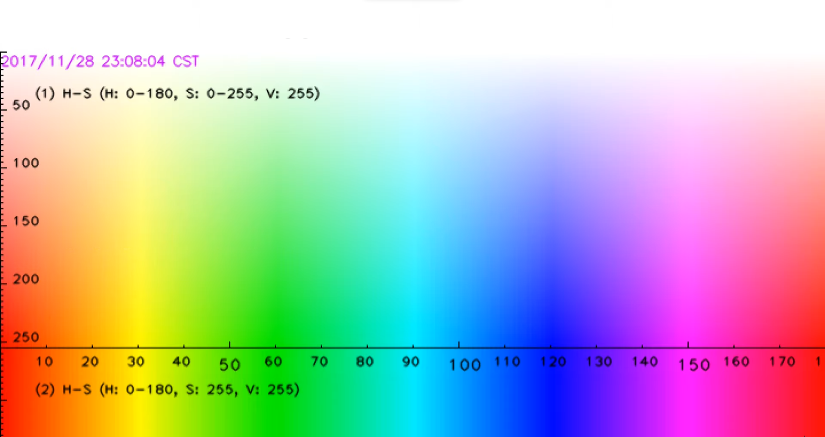
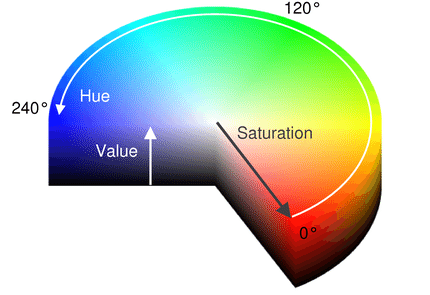
* Programme 5 : Détection de lignes pour déterminer la surface de jeu

Ce dernier programme permet de détecter les lignes présentes sur l’image, telle que les bandes du billard. Le but final derrière cela serait de détecter les bandes du billard pour en déterminer la surface de jeu, pour ensuite calibrer l’image reçue afin d’avoir un rectangle parfait (dans le cas où la caméra ne serait pas placée parfaitement au-dessus du billard et que l’image donnée n’est pas rectangulaire). Cela nous aurait permis de ne pas devoir utiliser les ArUco tags et d’opter pour une calibration différente de la précédente, étant donné que celle-ci ne convenait pas.

# Problématiques rencontrées

## Détection des couleurs

L’objectif de pouvoir détecter les différentes couleurs de billes présentes sur le billard est de permettre un programme d’entraînement plus complet et une détection des billes plus facile. Le problème était alors qu’il est nécessaire de trouver pour chacune des couleurs voulues les plages de valeurs pour les trois paramètres : teinte (= hue), saturation (= saturation) et contraste (= value). Ces valeurs sont comprises respectivement entre 0-180, 0-255 et 0-255.



Pour faciliter les limites inférieure et supérieure pour chacune de ces valeurs, nous avons utilisé un programme Python permettant de faire varier chacun des valeurs en voyant le résultat instantanément. Grâce à cela, nous avons pu éviter la détection de la bille blanche (qui est en réalité un blanc cassé/beige très clair) lorsque nous voulions détecter uniquement les billes de couleur jaune.

Une image contenant texte

Description générée automatiquement

Tout d’abord, expliquons de quelle manière ce logiciel travaille. Les lignes présentes dans la partie supérieure de la fenêtre, ayant chacune un curseur déplaçable, permettent de déterminer les valeurs du seuil inférieur (pour les trois premières lignes) pour la teinte, la saturation et le contraste, et les valeurs du seuil supérieur (pour les trois dernières). En ce qui concerne les images situées dans la partie inférieure de la fenêtre, celles-ci montre en temps réel les changements effectués. Celle du milieu affiche l’image originale traitée. Celle présente sur la gauche montre à l’aide de la couleur blanche les pixels étant présents dans les limites choisies. Dans ce cas-ci, les limites allant toutes les trois de leur plus petite valeur à leur plus grande, tous les pixels de l’image sont choisis, d’où l’image totalement blanche reçue à gauche de la fenêtre. L’image de droite montre quant à elle les pixels choisis directement par rapport à l’image originale, et non à l’aide d’une couleur blanche cette fois-ci. Encore une fois, dans ce cas-ci, tous les pixels sont sélectionnés aux vues des limites déterminées. De ce fait, toute l’image originale est retranscrite à la droite de la fenêtre.

Afin de déterminer les valeurs de limite pour les trois paramètres vus précédemment, nous avons fait des tests par essai-erreur. De cette manière, nous avons trouvé les valeurs permettant la meilleure détection possible pour les couleurs jaune, rouge, blanche et noire. Cette dernière a été utilisée pour trouver l’emplacement des poches, plus que pour déterminer la position de la bille noire présente sur le billard. En effet, celle-ci étant en réalité une bille rayée noire et blanche, la détection n’est pas optimale. De ce fait, pour la suite du projet, une bille totalement peinte en noir sera utilisée par facilité.

Une image contenant texte

Description générée automatiquement

Voici en exemple les valeurs trouvées pour la détection de la couleur rouge. Nous avons donc bien au milieu l’image d’origine, à gauche les pixels de couleur rouge étant compris dans les limites choisies à l’aide des curseurs, et à droite ces mêmes pixels repris de l’image d’origine afin de montrer les éléments détectés.

Cette première étape nous aura permis de trouver avec facilité et exactitude les valeurs de limite pour les paramètres utiles pour notre futur logiciel. Cela nous a permis de créer un programme permettant la détection de chacune des couleurs nécessaires avec plus de facilité. Voici ci-dessous la partie du code permettant la détermination de la couleur rouge sur l’image choisie.

Une image contenant texte

Description générée automatiquement

En remettant directement les valeurs trouvées à l’aide du premier programme pour les limites inférieures (= lowerred) et supérieures (= upperred), les billes de couleur rouge sont détectées et affichées sur un masque noir et blanc. La couleur blanche est utilisée ici pour montrer les pixels rouges. Un traitement est aussi effectué sur l’image d’origine car l’outil utilisé pour ce programme fonctionne mieux lorsque l’image d’entrée est en nuance de couleur « HSV », et non en « BGR » comme elle l’était de base.

Une image contenant texte, billard, vert, salle de billard

Description générée automatiquement

Nous voyons donc sur la fenêtre de gauche l’image originale devant être traitée, qui est donc elle en « BGR ». L’image du milieu est l’apparence obtenue de celle-ci en nuance de couleur « HSV ». Tout à droite, nous retrouvons en blanc les pixels ayant été détectés comme étant dans les limites choisies pour la couleur rouge. En vérifiant sur la première image, nous remarquons que ce sont bien les billes rouges qui ont été détectées, et aucune autre. Notre programme est donc fonctionnel et assez précis pour son usage ultérieur.

A la fin de ce programme et après avoir détecté toutes les couleurs voulues séparément (jaune, rouge, blanche et noire pour les trous), nous avons rassemblé tous les masques obtenus en un seul masque.

Une image contenant texte, intérieur, capture d’écran

Description générée automatiquement

Nous pouvons remarquer sur les images ci-dessus les observations suivantes :

* En haut à gauche : image originale
* En haut au milieu : le masque montrant les billes jaunes
* En haut à droite : le masque montrant les billes rouges
* En bas à gauche : le masque montrant la bille blanche
* En bas au milieu : le masque montrant la couleur noire (les poches)
* En bas à droite : les masques reprenant les quatre masques précédents.

## Détection des billes (cercle)

Le programme permettant la détection de cercles présents sur une image contient beaucoup de paramètres très importants pour le bon fonctionnement du logiciel. Il a donc fallu améliorer petit à petit ce dernier à l’aide d’essais erreurs afin d’arriver à un programme fonctionnel, qui détectait correctement les billes présentes et rien d’autre.

Pour la réalisation de ce programme, nous avons utilisé l’outil présent sur OpenCV « HoughCircles ». Cette méthode est caractérisée par de nombreux paramètres. Il faut donc trouver le paramétrage le plus adéquat pour que le programme détecte les éléments que nous souhaitons mettre en évidence. Pour expliquer le paramétrage utilisé, nous allons prendre la ligne de code permettant de détecter les cercles présents sur l’image donnée.



Les différents paramètres relevés ici sont les suivants :

* circles : cela va nous permettre d’obtenir un vecteur de sortie ayant comme caractéristiques (x centre , y centre , rayon) pour chaque cercle détecté.
* cv2.HoughCircles() : c’est la fonction permettant de trouver des cercles dans des images étant en nuances de gris à l’aide de la transformée de Hough. Cette dernière est une méthode permettant de détecter diverses formes sur des images, que ce soient des courbes, ou des droites. Nous verrons d’ailleurs par la suite un programme utilisant ce même principe pour détecter les lignes présentes sur l’image donnée.
* gray : cet élément est l’image sur laquelle nous allons rechercher les formes circulaires. Pour arriver à cette dernière, nous avons tout d’abord donné l’image originale, que nous avons traitée afin qu’elle ressorte en nuances de gris, d’où son appellation « gray ». Cela a été effectué à l’aide de l’outil d’OpenCV « cvtColor(ImageOriginale, cv2.COLOR\_BGR2GRAY) ».
* cv2.HOUGH\_GRADIENT : c’est la méthode faisant partie de la fonction générale « HoughCircles » qui va déterminer de quelle manière vont être détectés les cercles présents sur l’image.
* dp=1 : le « dp » est le rapport inverse de la résolution de l'accumulateur à la résolution de l'image. En prenant un ratio égal à 1, l’accumulateur a la même résolution que l’image d’entrée. En mettant cette valeur à 2, il aura une largeur et une hauteur deux fois moins grandes.
* minDist=10 : ce paramètre détermine la distance minimale entre les centres des cercles détectés. Si la valeur est trop faible, plusieurs cercles se chevaucheront, ce qui n’est pas admissible dans le cas de détection de billes. A contrario, si celle-ci est trop élevée, des cercles pourraient être manqués. Dans notre situation, les cercles ne doivent pas se superposés étant donné qu’il n’est pas possible d’avoir deux billes l’une sur l’autre. Nous avons choisi de mettre le double du rayon minimum des cercles détectés (minRadius expliqué par après).
* param1=100 : ce premier paramètre spécifique de la méthode HoughCircles montre le seuil le plus élevé des deux seuils transmis au détecteur de bords de Canny (le plus bas est deux fois plus petit).
* param2=10 : ce second paramètre spécifique de la méthode utilisée permet de trouver le seuil d'accumulation des centres de cercle à l'étape de la détection. Plus il est petit, plus de faux cercles peuvent être détectés. Les cercles correspondant aux plus grandes valeurs d'accumulateur seront retournés en premier.
* minRadius=5 : il permet de déterminer la valeur minimale du rayon des cercles détectés. En mettant une valeur inférieure à 5, les billes présentes sur le billard ne sont pas détectées. Au-dessus de celle-ci, des cercles non-désirés sont détectés.
* maxRadius = 15 : il permet de déterminer la valeur maximale du rayon des cercles détectés. En mettant une valeur inférieure à 15, les poches du billard ne sont pas détectées. Au-dessus de celle-ci, des cercles non-désirés sont détectés.

Après avoir trouvé un paramétrage le plus adéquat à notre problématique (détection des billes et des poches sur un billard), nous obtenons les résultats suivants.

Une image contenant intérieur, table de travail

Description générée automatiquement

L’image située à gauche est l’image d’origine sur laquelle nous voulons détecter les billes de billard et les poches. L’image du centre est le résultat de la première, qui a été traitée afin d’obtenir une image en nuance de gris, ainsi que légèrement floutée afin de rendre la détection des cercles plus facile. La troisième image est le résultat de l’originale sur laquelle le logiciel vient dessiner les cercles détectés afin de les mettre en évidence. Nous obtenons donc des cercles verts avec des points rouges en leur centre.

## Détection des trous (cercle)

Le but de ce programme était de déterminer l’emplacement des poches du billard. Avec l’outil utilisé, le centre du cercle détecté (et donc de la poche) est aussi affiché. C’est exactement ce dont nous avons besoin pour calculer le point d’impact sur la bille à jouer pour la faire tomber dans un trou du billard. Le paramétrage pour déterminer correctement l’emplacement des poches est expliqué dans le point précédent en grande partie dans le point précédent. Cependant, des modifications ont été effectuées afin d’améliorer la détection des trous uniquement.



Nous voyons que le « minDist » passe de 10 à 30. En effet, en laissant cette valeur égale à 10, des cercles non-désirés apparaissaient aux environs des poches du billard. Le fait d’augmenter ce nombre a permis de les faire disparaître. Aussi, le « minRadius » valant précédemment 5 et est remplacé par 10. Cela permet de ne plus avoir des cercles de plus petit rayon que ceux des poches. Grâce à ces deux modifications, le programme permet de détecter uniquement les poches du billard, sans aucune interférence. La figure présente ci-dessous montre le résultat final de ce programme Python.

Une image contenant intérieur

Description générée automatiquement

Les trois images présentes ci-dessus montrent donc l’efficacité et la précision de ce programme. La première image étant celle d’origine, celle du milieu l’image traitée afin d’obtenir une nuance de gris, et celle de droite montrant les poches détectées.

## Détection des limites du billard (ligne)

L’objectif de ce programme est de délimiter la surface de jeu du billard afin de calibrer par la suite l’image envoyée par la caméra. En effet, cette dernière ne pouvant pas toujours être placée parfaitement au-dessus du billard et bien centrée, l’image reçue sur le logiciel est de biais. Le billard apparait donc à l’écran comme étant un trapèze au lieu d’être rectangulaire, et les billes sont légèrement ovales à la place d’être parfaitement circulaire. Cela peut poser des problèmes pour la suite des opérations, par exemple, lorsqu’il faudra trouver le point d’impact exact de la bille à empocher par le joueur. Dans le monde du billard, un décalage de quelques millimètre par rapport au point de visée peut avoir comme conséquence de ne pas réussir à empocher la bille voulue. Il faut donc une calibration des plus précises afin d’éviter tout soucis par la suite.

Le système trouvé par le groupe de l’année passée ne semblait pas satisfaisant à ce niveau, nous avons donc décidé d’essayer une autre méthode pour voir si celle-ci permettait une meilleure précision ou non. Pour rappel, les élèves de l’année précédente étaient partis sur une calibration faite à l’aide d’ArUco Tags.

Une image contenant texte, fichier, carré, commode

Description générée automatiquement

Notre programme permet de détecter les lignes présentes sur l’image donnée au logiciel. Les bandes d’un billard étant droites, celles-ci sont détectées comme étant des lignes. Cela nous permet donc d’obtenir ce que l’on cherchait en créer ce programme. Le problème persistant encore actuellement est que le paramétrage de cette méthode n’est pas optimal. De ce fait, d’autres lignes sont aussi détectées et il n’est donc pas encore possible de réaliser un calibrage de cette manière. Nous poursuivons les recherches à ce niveau afin d’améliorer ce programme, dans le but d’avoir une calibration correcte pour la suite des événements. Comme nous pouvons le voir sur la figure présente ci-dessus, un traitement d’image afin de passer l’image d’origine en nuance de gris est appliqué. Cela permet une meilleure réponse de l’outil utilisé (HoughLines).

## Détection des couleurs et des billes en même temps

Le programme permettant la détection des billes à l’aide de leur forme circulaire ayant quelques problèmes d’interférence (détection de cercle là où il n’y en a pas), nous avons pensé à fusionner deux de nos programmes. En effet, notre premier logiciel sortait un masque en noir et blanc des billes et des poches détectées à l’aide de leur couleur, que nous pouvons d’ailleurs voir sur l’image centrale ci-dessous. Nous avons donc pensé qu’en utilisant ce masque, cela pourrait réduire les pixels parasites présents sur l’image de base.

Une image contenant texte, moniteur

Description générée automatiquement

Nous avons implémenté à la suite de celui-ci le programme utilisant l’outil « HoughCircles ». Comme nous le remarquons sur la figure ci-dessus, cela nous a permis d’obtenir des résultats très satisfaisants.

Une image contenant texte, extérieur, botte, parking

Description générée automatiquement

En effet, comme nous pouvons l’observer sur la comparaison entre les deux programmes, présente ci-dessus, certains cercles détectés sur les bandes du billard disparaissent dans le deuxième cas. L’amélioration est donc nette visible. L’unique défaut de ce nouveau programme est la non-détection de la poche inférieure gauche du billard. Il pourrait donc être utile d’effectuer en premier lieu la détection des trous avec une image de billard sans bille, suivi de la détection des billes par couleur et finir par la détection des billes à l’aide de leur forme à partir du masque de couleurs.

## Calibration de la caméra

## Raspberry Pi pas suffisante 🡪 Solutions possibles

## Utilisation de Linux pour plus d’audience

# Pistes d’amélioration futures

# Explication de la détermination du programme d’entraînement

## Choix de plusieurs difficultés

## Choix du type d’exercices

### Collé bande

### Bille cachée

### Défense

### Line-Up

### Carré

### Replacements

### Contrôle de la force

### Angles d’attaque

### Rétro

### Coulé

### Carreaux

### …

# Conclusion

* Objectifs du Q1 remplis
* Détermination des objectifs du Q2
* Ressenti par rapport à l’avancée actuelle
* …