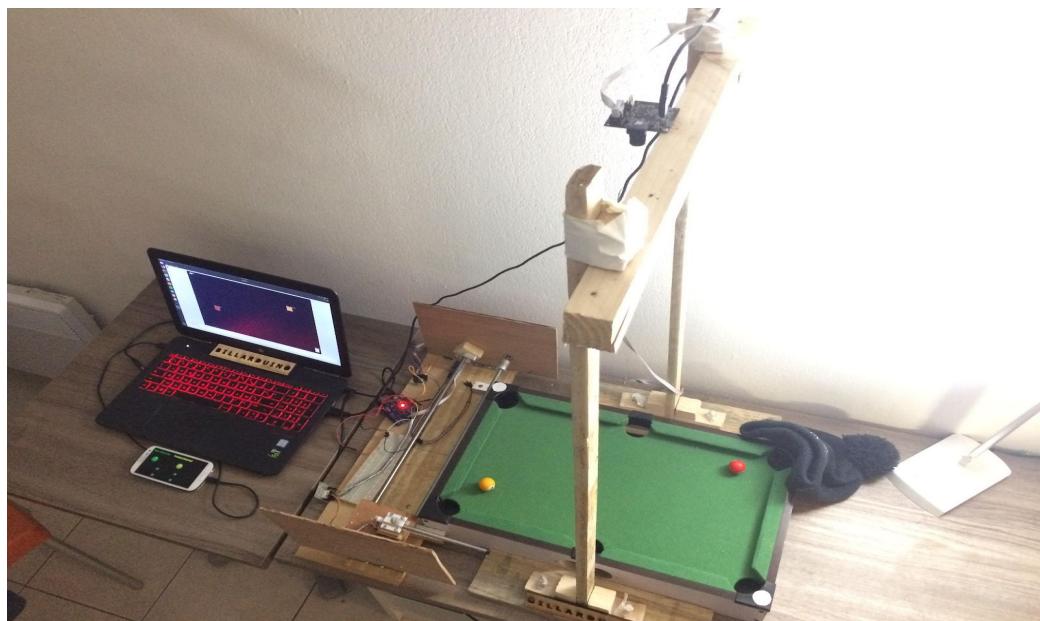




Rapport du projet Billarduino



POLYTECH[®]
NICE-SOPHIA



Université
Nice
Sophia Antipolis

Membre de UNIVERSITÉ CÔTE D'AZUR

Sommaire

Chapitre I: Objectif premier et matériel

Chapitre II: Assemblages des différents objets et composants

Chapitre III: Explications théoriques et mathématiques

Chapitre IV: Partie informatique et acquisition vidéo

Chapitre V: Etat de fin de projet

Chapitre I: Objectif premier et matériel

Notre projet est celui d'un billard autonome. Le principe est simple ; en posant une bille de tir (bille blanche) à une distance proche du module de tir, et une bille visée (bille objet), une queue de billard viendra se placer dans l'axe de la bille blanche pour rentrer la bille objet dans le trou le plus adapté.

Pour cela, nous avons besoin de construire un environnement de communication entre un composant responsable de l'acquisition vidéo, l'ordinateur, et les différents moteurs qui, nous le verrons plus tard, réalisent rotation et translation.

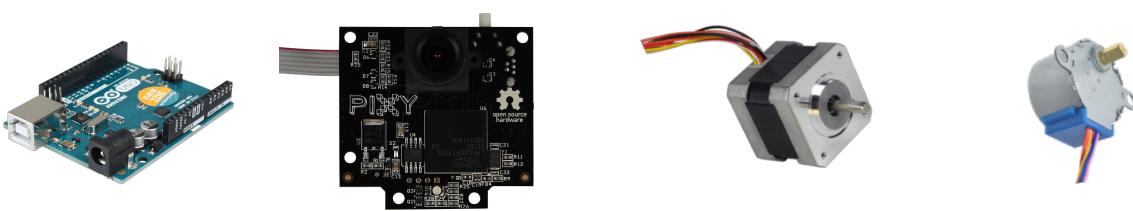
L'objectif de mi-projet fixé était de bien faire fonctionner le système d'acquisition et de s'affranchir définitivement de la partie Hardware pour pouvoir se focaliser sur le code (et donc l'IA).

Pour la fin de projet, on veillera à bien remplir l'objectif et donc pour deux positions de bille données, mettre le module dans les bonnes conditions de tir.

Notre objectif est de tout automatiser ; c'est à dire d'automatiser le déplacement du module de tire comme le tir en lui-même.

Pour mener à bien ce projet, nous avons besoin de différents éléments statiques ou électroniques divers :

Une tige filetée, une tige coulissante (responsable du tir), deux écrous (sur la tige fileté), une prise à roulement (pour la tige filetée), deux fixations (pour la tige coulissante), deux coulisseaux (pour la tige coulissante et pour la translation du module), un moteur pas à pas 12V Nema 17 et son coupleur, un moteur 28-byj-48 (pour la rotation), un billard miniature, une PixyCam (pour l'acquisition vidéo), trois morceaux de bois (deux pieds et un support pour tenir la PixyCam avec pour les maintenir des équerres).



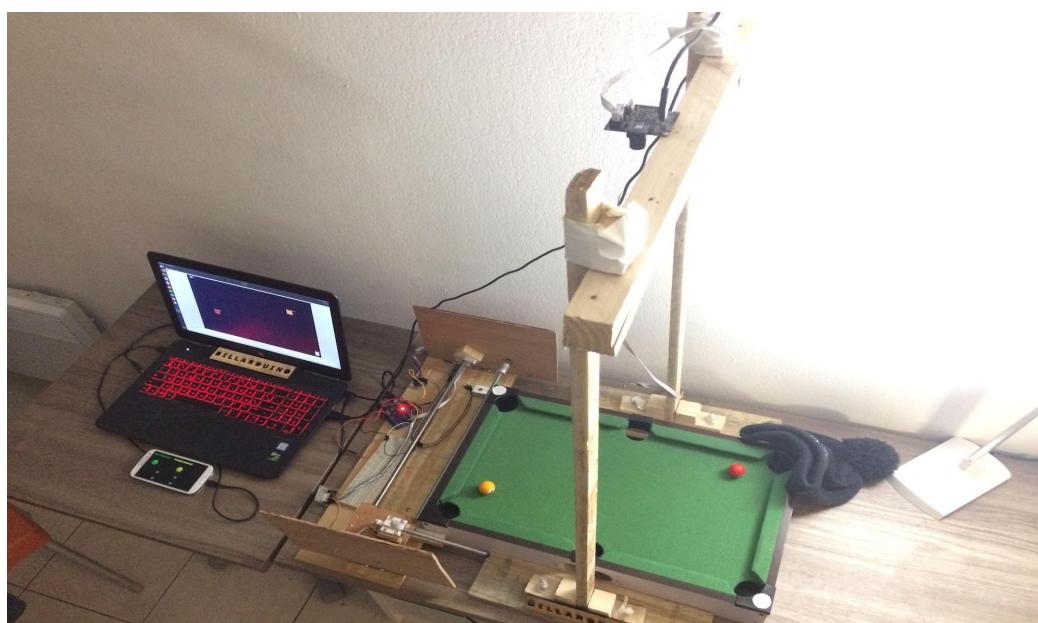
Chapitre II: Assemblages des différents éléments et composants

La structure du billard est arrangée de façon à ce que tous les composants électroniques soient en communication avec la carte Arduino. Cette carte Arduino sera alimenté par un ordinateur et fournira à son tour une alimentation aux différents composants. Notre billard sera accroché toujours à la même position au rail principale grâce à deux cales de bois condamnables. La perche, elle, se réglera au moment de l'installation.

Pour la réalisation du projet, voici le planning théorique qui permet de se représenter les différentes phases de conception :

Tâches \ Séances	03/12/18	10/12/18	17/12/18	07/01/19	14/01/2019	21/01/19	06/02/19	13/02/19	27/02/19	06/03/19	13/04/2019
Test PixyCam											
Rédaction du programme (avec portions de code test)											
Construction du module mobile											
Construction de la PixyPole (perche PixyCam)											
Construction du rail du module											
Finalisation du programme											
Tests et réglages d'appoint											

Voici une photographie d'une vue d'ensemble du billard:



Tout d'abord, le billard doit être posé sur une surface plane, parallèle au sol puisque l'objectif est avant tout de pouvoir mettre en place un système de coordonnées cartésiennes.

Ce système cartésien nous permettra le repérage sur un quadrillage de pixel la position des éléments en 2D. Si le billard est un peu penché, les calculs seront disproportionnés par rapport à la réalité. C'est la PixyCam, véritable actrice de l'acquisition vidéo qui nous fournit de manière brut les éléments de ce repérage. Cette dernière agit comme l'oeil du projet. La PixyCam devra être fixé relativement haut par soucis de focale de l'objectif.

La PixyCam tient grâce à deux perches en bois tenues par des pieds fixés et une barre en bois transversale sur laquelle elle repose.

Nous avons également construit une structure en bois sur laquelle se repose la tige filetée accompagnée d'une tige coulissante (en acier) pour déplacer le module de translation proprement. La tige fileté a été fixé fermement entre les deux planches de la structure en bois (moteur d'un côté, roulement de l'autre), afin que la rotation du moteur n'entraîne pas de mobilités non souhaitées. La tige filetée peut tourner grâce à un moteur Nema 17 piloté par le driver qui transmet le mouvement de rotation à la tige grâce au coupleur.

Afin que le module en bois (sur lequel se trouve la tige de tir) puisse bien se déplacer, il est posé sur une tige coulissante qui elle est aussi fixé entre les deux planches de la structure en bois, parallèlement à la tige fileté. Le module possède un support en bois et en dessous un coulissoir pour permettre de contrôler la translation, comme sur l'image ci-dessous:



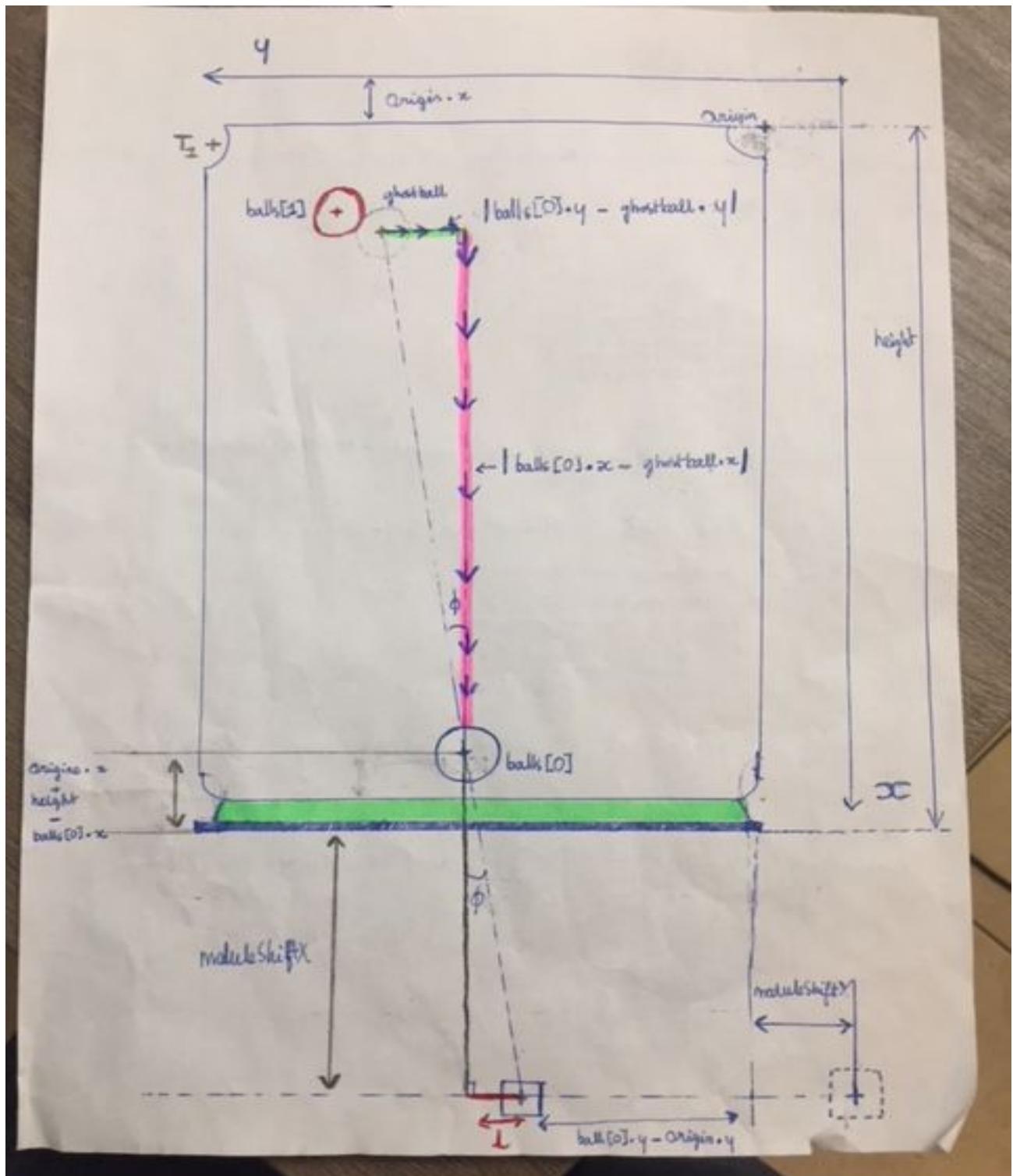
Un écrou (deux en fin de projet) est fixé sur ce support en bois et est vissé sur la tige fileté.

Pour le tir, nous avons utilisé un coulisseau, dans lequel est mis la tige en acier, qui fonctionne comme une arbalète avec un élastique. Pour pouvoir effectuer un mouvement de rotation, un moteur pas à pas a été fixé de manière à supporter une plaquette de bois sur laquelle repose le coulisseau de tir :



Chapitre III: Explications théoriques et mathématiques

Voici sur la feuille de modélisation mathématique de notre projet:



Ici est tracé un repère cartésien dont l'origine se trouve sur la partie supérieure droite de la feuille. Ceci correspond à un coin de la PixyCam.

Nous avons principalement trouvé les équations mathématiques qui nous permettront de calculer précisément la position de l'impact. Le principe est simple: à partir des coordonnées de la bille objet et du trou (en deux dimension, coordonnées cartésiennes), nous pouvons calculer les coordonnées de "tir", c'est à dire où exactement nous devons propulser la bille blanche.

Cette position, que nous avons appelé "ghostball" se résume au prolongement de la bille objet par rapport au trou, à une distance d'un diamètre de bille. On précisera que la bille objet est appelée `balls[1]` au sein du code. Bien sûr, le trou est préalablement choisi grâce à la position en `x` des deux billes (le trou le plus commode en fonction de la situation).

Les deux trous sont déterminés par la PixyCam durant la phase d'acquisition après établissement de l'échelle.

Pour la rotation du module, elle se fait grâce à la position de la ghostball. L'angle est calculé grâce à une règle de trigonométrie qui fait intervenir les deux distances verte et rose sur le schéma (distance entre la ghostball 1 et la bille blanche en coordonnées `x` et `y`).

Sur le schéma, nous pouvons voir les différentes constantes du billard, qui sont considérés comme fixe (cela apporte sûrement des petites imprécisions). Ces constantes sont:

- `ModuleShiftX` et `ModuleShiftY` qui sont respectivement les distances du centre du module en `X` et en `Y` à l'angle du billard le plus proche.
- La `width` (largeur) et `height` (longueur) du billard.

La longueur `origin.x` qui correspond à la coordonnée en `x` de l'origine du billard (coin haut droit du billard).

A l'aide de toutes ces distances et de l'angle nous pouvons calculer la distance parcourue par le module de translation à savoir aller jusqu'à la bille blanche en soustrayant ou additionnant le décalage relatif à l'angle en fonction de la positivité de l'angle

Voici donc le détail de tous nos calculs:

Calculs :

l'arc pour les deux bras

$$\text{ghostball_x} = \text{aimetole_x} + (\text{ballz}(+1_n - \text{aimetole_x})^2 + (\text{centToCent}(\text{ballz}))^2) / \text{aimetole_distambi(balls)}$$

ghostball_y = Same thing with y

Dans le cas de Tz :

$$\phi = \text{atan}\left(\frac{|\text{ghostball_y} - \text{ballz}[0]_y|}{|\text{ghostball_x} - \text{ballz}[0]_x|}\right)$$

$$l = \tan(\phi) * \text{CartToCent}(\text{origine_x} + \text{CentToCart}(\text{height}) - \text{ballz}[0]_x + \text{CentToCent}(\text{moduleShiftX}))$$

Translation : moduleShiftY + CartToCent(ballz[0].y - Origin.y) - l

Dans le cas de Tx :

$$\phi = -\text{atan}\left(\frac{|\text{ghostball_y} - \text{ballz}[0]_y|}{|\text{ghostball_x} - \text{ballz}[0]_x|}\right)$$

$$l = \tan(-\phi) * \text{CartToCent}(\text{origine_x} + \text{centToCart}(\text{height}) - \text{ballz}[0]_x + \text{centToCart}(\text{moduleShiftY}))$$

Translation : moduleShiftY + CartToCent(ballz[0].y - Origin.y) + l

Chapitre IV: Partie software et automatisation

Maintenant que nous avons compris le fonctionnement global du projet, nous allons nous pencher sur le cœur de son l'intelligence ; à savoir la partie informatique du projet.

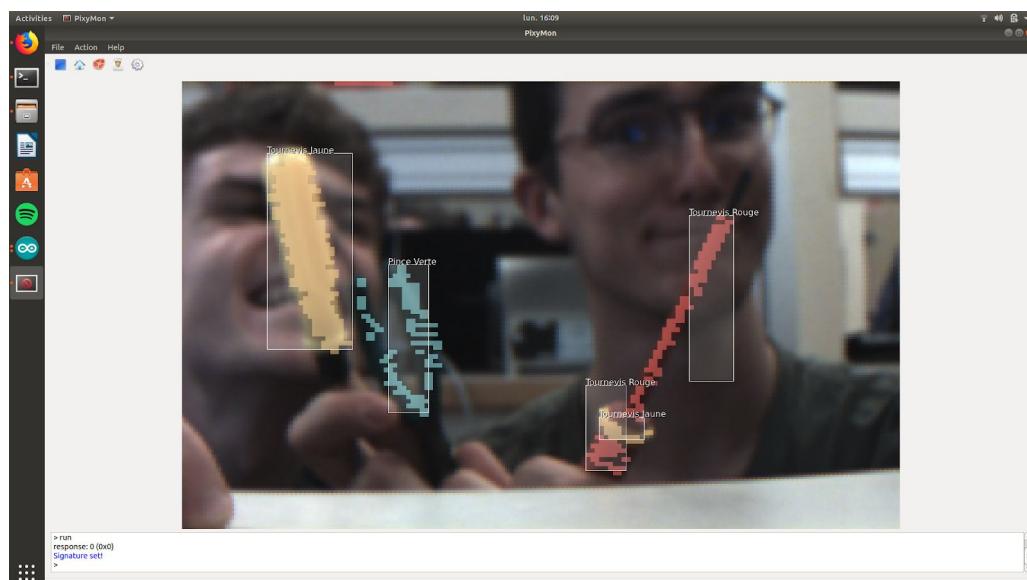
L'acquisition vidéo est réalisée par l'intermédiaire de la PixyCam. Cette petite caméra permet de détecter des objets de couleurs bien discernables. Chaque objet préalablement enregistré par la PixyCam en phase d'acquisition possède des informations précise exploitable dans un deuxième temps. Ces informations sont détenues grâce à la création de la signature de l'objet.informations sont détenues grâce à la création de la signature de l'objet.

Pour ce qui est de la manière de définir une signature, il faut tout d'abord brancher la PixyCam à l'ordinateur. On laisse ensuite son doigt appuyé sur le seul bouton de la PixyCam pour faire défiler les couleurs qui correspondent aux emplacements de signatures.

Ainsi, on a un cycle de couleurs qui nous permet de choisir et de définir un total de 8 signatures. Voici donc les signatures que l'on définit :

- Une **bille jaune** qui jouera le rôle de la bille blanche.
- Une **bille rouge**, qui jouera le rôle de la bille objet qu'il faut à tout prix mettre dans le trou.
- **Deux gommettes** (à côté des trous), situés au niveau de deux coins diagonaux. Ceci pour établir une échelle la plus précise possible.

Voici une image d'une de nos premières acquisitions (de ce que l'on avait sous la main) :



Il est intéressant de remarquer qu'au niveau du Tournevis Rouge, la PixyCam croit détecter un "Tournevis Jaune" au niveau de l'anneau jaune alors que logiquement il fait partie du Tournevis Rouge.

On s'attend donc par déduction à ne pas choisir des objets de couleurs voisines ou des objets multicolors, dans le potentiel risque d'avoir des problèmes lors de la phase d'acquisition.

Le programme, lui, met en relation toutes les parties du projet ; il réalise l'acquisition vidéo et tant que les critères requis ne sont pas validés, il bouclera sur cette étape. A partir de l'acquisition, on établit l'échelle à partir des deux gommettes, et on effectue toutes la suite de calcul évoqués précédemment pour arriver au résultat de l'angle et la distance voulu, qui sont convertis en nombre de pas à effectuer pour chaque moteur.

Le programme est adapté aux complexités d'un repère cartésien puisque nous avons créé une classe Point qui permet facilement le repérage des différents éléments et le calcul des distances qui peuvent les séparer.

Ces distances se convertissent très facilement du repère cartésien à la réalité, ou l'inverse grâce aux fonctions de passage `centToCart(d)` ou `cartToCent(d)` (respectivement de Centimètre à Cartésien et de Cartésien à Centimètre)

The screenshot shows a Linux desktop environment with several open windows. At the top, there's a system tray icon for PixyMon. The main window is titled "PixyMon" and displays a live video feed from a camera. Two small orange rectangles are overlaid on the video, representing detected objects. To the right of the video feed is a terminal window titled "/dev/ttyACM1 (Arduino Uno WiFi Rev2)" showing a continuous stream of sensor detection logs. Below the video feed, the terminal window has a scroll bar indicating it's displaying multiple lines of text. The bottom of the screen features a dock with various application icons, including a file manager, a browser, and system tools. The overall interface is typical of a Linux desktop with a dark theme.

```
block 1: sig: 2 x: 196 y: 59 width: 11 height: 14
Detected 2:
block 0: sig: 1 x: 34 y: 35 width: 15 height: 13
block 1: sig: 2 x: 198 y: 54 width: 11 height: 12
Detected 2:
block 0: sig: 1 x: 34 y: 36 width: 13 height: 14
block 1: sig: 2 x: 198 y: 54 width: 13 height: 13
Detected 2:
block 0: sig: 1 x: 35 y: 36 width: 15 height: 14
block 1: sig: 2 x: 198 y: 55 width: 11 height: 14
Detected 2:
block 0: sig: 1 x: 34 y: 36 width: 13 height: 14
block 1: sig: 2 x: 198 y: 55 width: 13 height: 14
Detected 2:
block 0: sig: 1 x: 34 y: 36 width: 13 height: 14
block 1: sig: 2 x: 198 y: 55 width: 14 height: 12
Detected 2:
block 0: sig: 1 x: 34 y: 36 width: 13 height: 14
block 1: sig: 2 x: 197 y: 54 width: 13 height: 13
Detected 2:
block 0: sig: 1 x: 34 y: 36 width: 13 height: 12
block 1: sig: 2 x: 199 y: 55 width: 12 height: 14
Detected 2:
block 0: sig: 1 x: 34 y: 36 width: 14 height: 14
block 1: sig: 2 x: 198 y: 54 width: 13 height: 13
Detected 2:
block 0: sig: 1 x: 34 y: 36 width: 14 height: 14
block 1: sig: 2 x: 200 y: 54 width: 14 height: 14
Detected 2:
block 0: sig: 1 x: 35 y: 36 width: 14 height: 14
block 1: sig: 2 x: 199 y: 55 width: 14 height: 14
Detected 2:
block 0: sig: 1 x: 34 y: 36 width: 14 height: 14
block 1: sig: 2 x: 198 y: 54 width: 10 height: 14
Detected 2:
block 0: sig: 1 x: 34 y: 36 width: 13 height: 14
block 1: sig: 2 x: 199 y: 54 width: 13 height: 14
Detected 2:
block 0: sig: 1 x: 34 y: 36 width: 13 height: 14
block 1: sig: 2 x: 198 y: 54 width: 12 height: 14
Detected 2:
block 0: sig: 1 x: 34 y: 36 width: 14 height: 14
block 1: sig: 2 x: 198 y: 54 width: 12 height: 14
Detected 2:
block 0: sig: 1 x: 34 y: 36 width: 15 height: 14
block 1: sig: 2 x: 198 y: 54 width: 11 height: 11
Detected 2:
block 0: sig: 1 x: 34 y: 36 width: 14 height: 14
block 1: sig: 2 x: 198 y: 54 width: 11 height: 14
Detected 2:
block 0: sig: 1 x: 34 y: 36 width: 13 height: 14
block 1: sig: 2 x: 199 y: 54 width: 12 height: 15
```

response: 0 (0x0)
> runprogArg 8 1
response: 0 (0x0)
error: Pixy has stopped working.
Pixy off
> runprogArg 8 1
response: 0 (0x0)
> Signature set!
> runprog 0
response: 0 (0x0)
> runprog 0
response: 0 (0x0)
sync
> runprog 8
response: 0 (0x0)
> runprog 0
response: 0 (0x0)
> runprogArg 8 1
response: 0 (0x0)

Autoscroll Show timestamp No line ending 9600 baud Clear output

Chapitre V: Etat de fin de projet

Au final, nous avons réussi à atteindre notre objectif principal :

- *A partir des positions quelconques des deux billes (blanche et objet), on arrive automatiquement à amener la queue de billard au bon endroit et avec le bon angle.*

Cependant, le tir doit être effectué manuellement à l'aide d'un élastique, ce que l'on aurait aimé idéalement automatiser. Cette fonctionnalité aurait eu de sérieuse répercussions sur la précision du module, dû au jeu existant au niveau du pan du moteur de rotation. De plus, nous avons réalisé que les défauts de précision sur certains tirs étaient uniquement dû à la fiabilité des autres composants. En effet, l'acquisition approximative de la PixyCam (dont la résolution est de 200x320) couplée aux imprécisions de mesure réelles mènent à de certaines imprécisions au niveau du tir. Beaucoup d'aspects qui au final négligent les différentes corrections que l'on peut apporter au sein du code.

Autre point à souligner, c'est l'impossibilité à cause de la carte dont nous disposons (UNO) de faire fonctionner la partie Bluetooth que nous avions développé. Celle ci prévoyait par son comportement de rajouter plus de flexibilité au programme pour valider les différentes phases et ainsi ne pas être pris de cours, ce qui est actuellement régi par des `delay(s)`.

Si nous avions encore plus de séance/temps à consacrer voici ce que l'on pourrait rajouter :

- 1: Plusieurs billes à viser, et un calcul permettant de déterminer l'enchaînement des tirs pour rentrer les billes dans le bon ordre sans occasionner de gène !
- 2: A condition de changer de carte Arduino, la possibilité de déplacer nous même le module par Bluetooth et faire une sorte de compétition contre l'IA !
- 3: Sans le soucis de la précision qui fut notre combat jusqu'au dernier jour, on pourrait travailler le design du projet, ce dernier étant actuellement aménagé

pour être purement fonctionnel (mise en place d'un boîtier, estompage des formes voire peinture)

Pour conclure, nous sommes très fier que le déroulement du programme que ce soit les calculs et le déplacement se réalisent correctement, et qu'on arrive la plupart du temps à rentrer la bille du premier coup ! Il a été difficile jusqu'aux derniers jours précédant la présentation d'obtenir un placement cohérent quelque soit la disposition des billes. La plupart des soucis techniques ont été surmontables de notre côté comme en témoigne le résultat final. Ce projet fut pour nous l'occasion de nous former sur les points suivants :

- Travailler en équipe sur un projet pluri-disciplinaire
- Communication interne/externe avec l'Arduino
- Développement d'un code orienté objet pour une organisation optimale
- Structuration d'un programme multiphasé
- Analyse de données entrante par un composant tiers
- Mise en application de formules mathématiques théoriques sur un système réel

Remerciements :

A Pascal Masson, pour le matériel et les conseils techniques (et aussi le billard !)

A Lyonel Dahan, pour son aide et son FabLab' maison pour la réalisation du hardware.

Maxime et Antoine.

