

Programmation – projet 2018 (r1)

Introduction

Analyse de séquences vidéo pour déterminer le nombre de impacts entre les pucks et la table de air hockey ainsi que le nombre de chocs entre les pucks. Un fichier PDF « score » avec l'analyse de la partie sera produit.

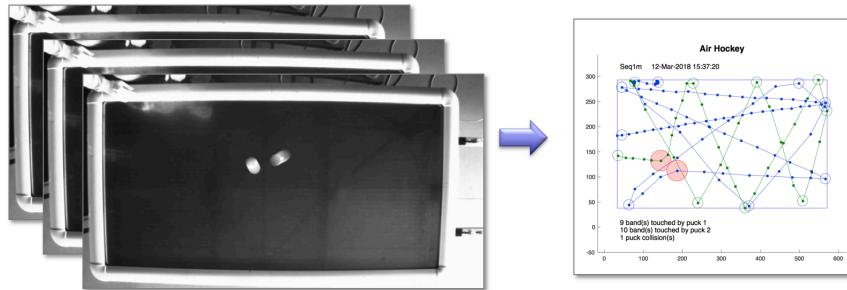


Fig. 1. Analyse des traces des pucks

NB. Plusieurs simplifications seront faites par rapport à la description ci-dessus.

Ce projet a pour but de vous familiariser avec les trois environnements étudiés au cours. Il vous permettra également de mettre en œuvre les concepts présentés durant le cours au travers d'un exercice de plus longue haleine. Chaque environnement générera une étape du projet. Le projet est découpé en plusieurs parties schématisées par le graphique ci-dessous:

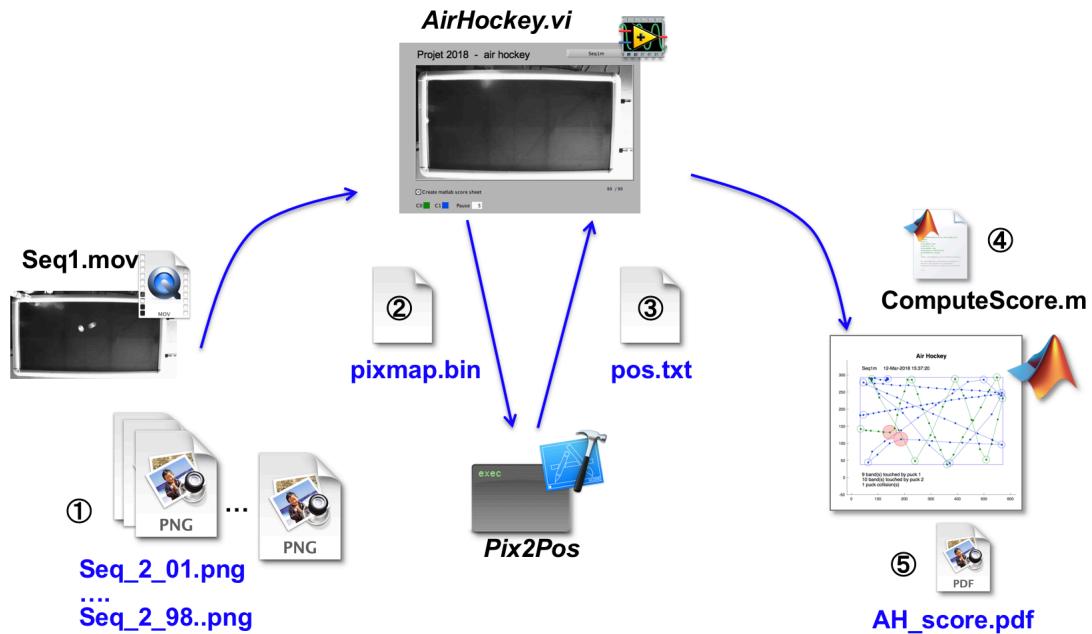


Figure 2. Les étapes du projet. La première étape (1) est la lecture d'un fichier PNG contenant une image (frame) de la séquence vidéo. Cette image est sauvegardée dans un format compressé qu'il faudra décoder et sauver dans un nouveau format (2). Les

coordonnées des pucks seront extraites pour être sauvées dans un autre fichier (3). Les différentes positions sont sauvées dans des vecteurs qui seront analysés pour déterminer les points d'impacts. Le résultat de l'analyse sera sauvé dans un fichier PDF (5).

L'échange d'informations entre les trois logiciels se fait par l'intermédiaire des fichiers **Seq_Y_XX.png**, **pixmap.bin**, **pos.txt**, **ComputeScore.m** et **AH_Score.pdf**.

Lors de la première étape, le fichier PNG (niveau de gris sur 8 bits) contenant une image (frame) de la séquence vidéo sera décompressé (fonction fournie) en un vecteur de points (entier) représentant les pixels de l'image (pixmap). Ce vecteur sera analysé pour en extraire les coordonnées des différentes pucks. La taille de l'image, les couleurs des pucks ainsi que les pixels seront sauvés au format binaire dans le fichier **pixmap.bin**. Chaque puck est représenté par une carré de 5x5 pixels de même couleur.



Figure 3. Exemple de pixmap avec 2 pucks

La deuxième étape du projet consiste à lire le fichier **pixmap.bin** contenant les pixels et à y trouver les patterns de pixels de couleur correspondant aux différents pucks. Les coordonnées de chaque puck seront sauvegardées dans le fichier de sortie **pos.txt**. Cette opération sera répétée pour chaque image (frame) de la séquence vidéo.

Au cours de la troisième étape du projet, il s'agit de lire les fichiers **pos.txt** contenant les coordonnées de chaque puck. Les coordonnées seront sauvegardées dans un vecteur dans le programme appelant **AirHockey.vi**.

Dans la quatrième étape, les coordonnées successives seront analysées pour déterminer les points d'impact. Elles seront affichées selon des paramètres spécifiés dans le fichier **ComputeScore.m** puis cette figure sera sauvegardée au format PDF dans le fichier **AH_Score.pdf**.

Mise en œuvre

1^{re} partie en C++ : le programme **Pix2Pos cherche des patterns de pixels de couleur correspondant aux pucks.**

Spécifications

Le programme C++ **Pix2Pos** va lire le fichier **pixmap.bin** contenant une suite de valeurs représentant les pixels de l'image. Les valeurs lues seront analysées pour détecter les pixels de couleur représentant les pucks.

Afin de simplifier l'analyse de l'image, vous pouvez vous baser sur les hypothèses simplificatrices suivantes :

- Les valeurs des couleurs des 2 pucks sont données
- Les dimensions du pixmap sont données
- Les pucks sont représentés par des patterns de 5x5 pixels de même couleur
- Toutes les valeurs du fichier binaire sont encodées au format unsigned int, mais pour les pixels de l'image les valeurs possibles sont comprises entre 0 et 255
- Les pucks peuvent être présents ou non, il y en a au maximum 2

Le fichier **pixmap.bin** correspondant est une suite de valeurs au format **binaire** structuré comme ci-dessous. Chaque valeur est codée sous la forme d'un entier non signé **uint** et les valeurs excédentaires sont à ignorer :

```
Largeur de l'image en pixels
Hauteur de l'image en pixels
Couleur1
Couleur2
pixel 0
pixel 1
...
last pixel (il y a largeur x hauteur pixels dans le fichier)
```

Organisation des pixels

Les pixels sont organisés en ligne, le pixel 0 correspond au pixel en haut à gauche de l'image. Les lignes sont mises bout à bout dans le fichier. Il n'y a pas de retour de ligne (ou équivalent) dans le fichier.

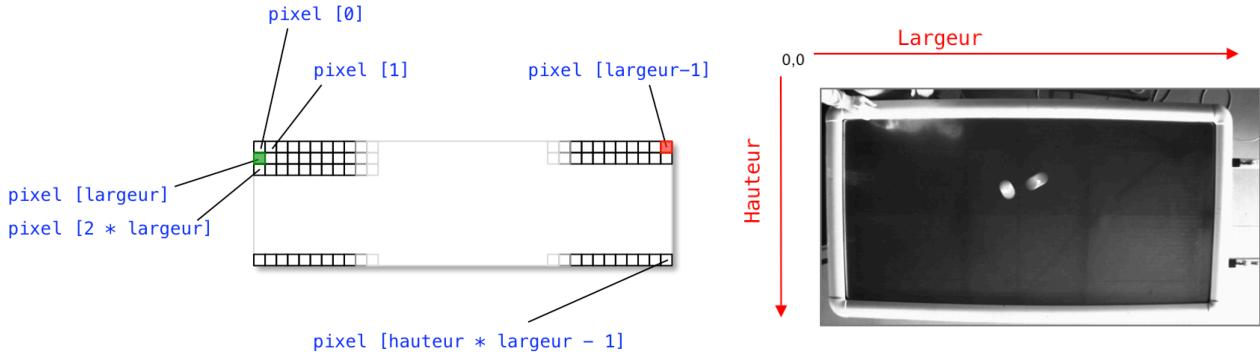


Figure 4a. Organisation des pixels en mémoire



Figure 4b. Organisation des pixels dans le fichier

Exemple du contenu du fichier [pixmap.bin](#) correspondant à l'image Fig. 4, ci-dessus :

```

640 // largeur en pixels
340 // hauteur en pixels
0 // couleur 1
1 // couleur 2
20 // valeur pixel 0
243 // valeur pixel 1
...
0 // un pixel de couleur 1
...
1 // un pixel de couleur 2)
...
56 // valeur du dernier pixel

```

Détection des pucks

Vous devez chercher dans le pixmap en mémoire, une pattern de 5x5 pixels de couleurs identiques C1 (ou C2), comme sur le zoom Fig 4.c. D'autres pixels de couleurs peuvent exister dans l'image, mais pas sous forme de pattern 5x5. Il y a au maximum une fois la pattern d'une couleur donnée, il peut également ne pas y en avoir. Les coordonnées du puck correspondent au point central de la pattern 5x5.

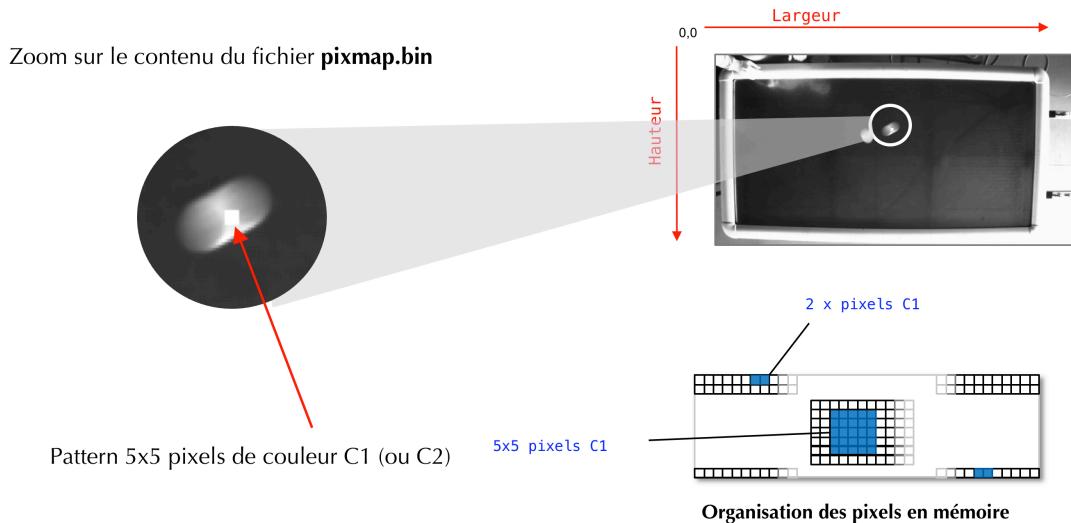


Figure 4c. Zoom sur le pixmap et pattern 5x5

Les couleurs des 2 pucks sont :

Couleur 1: 0
Couleur 2: 1

Dans une premier temps le programme **Pix2Pos** doit lire les valeurs du fichier **Pixmap.bin** une à une, les valider et les stocker dans un tableau dynamique **Pix** (vector of vector). Puis, il s'agira de comparer chaque valeur de **Pix** avec les couleurs des pucks. En cas de concordance, il faut vérifier que les pixels adjacents formant la pattern 5x5 sont de même couleur, dans l'affirmative la position sera mémorisée. Puis les pixels suivants seront évalués. Une fois tous les pixels visités, les positions des pucks seront sauveées dans le fichier **pos.txt**. Il faut cependant tenir compte de l'origine (0,0) des coordonnées à chaque étape. Pour l'image PNG le (0,0) se trouve en **haut** à gauche, pour l'affichage dans Matlab, l'origine est en **bas** à gauche.

Le fichier texte de sortie **pos.txt** contient l'index de la couleur ainsi que les coordonnées X et Y de chaque puck. En cas d'absence de puck, les coordonnées suivantes seront utilisées : -1,-1. La syntaxe employée est la suivante :

```
IDcouleur1, Xcouleur1, Ycouleur1, retour à la ligne
IDcouleur2, Xcouleur2, Ycouleur2, retour à la ligne
```

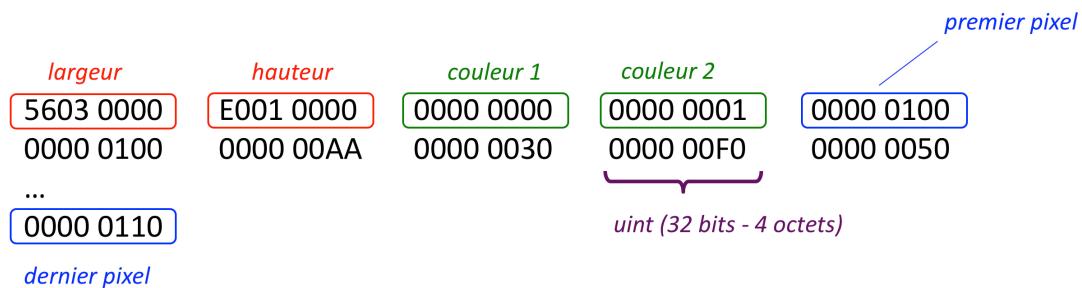
Exemples :

<code>0,37,173</code>	coordonnée du puck de couleur 1
<code>1,18,15</code>	coordonnée du puck de couleur 2
<code>0,-1,-1</code>	puck de couleur 1 non trouvé
<code>1,18,15</code>	coordonnée du puck de couleur 2

Code C++

La première partie de votre code consiste à lire le fichier d'entrée **Pixmap.bin**. Après avoir lu la taille de l'image et les couleurs des 2 pucks, vous devez lire les pixels au format **binaire**. Inspirez-vous du slide 35 *Exemple lecture d'un fichier binaire* de la semaine 4. Vous devez comparer chaque pixel avec les 2 couleurs des pucks. Dans le cas où il y a concordance, vous devez tester que les pixels adjacents formant la pattern 5x5 soit de la même couleur que la couleur du puck. A la fin, sauver les coordonnées des 2 pucks dans le fichier **Pos.txt**. Utilisez des fonctions à l'intérieur de votre programme, par exemple pour valider les pixels de la pattern 5x5.

Le fichier **binaire** est sous la forme suivante :



Big et LittleEndian: organisation des octets dans un fichier.

Il y a deux manières d'organiser les octets dans un fichier : soit l'octet de poids le plus fort est enregistré à l'adresse mémoire la plus petite, soit c'est l'inverse. Un octet est représenté par 2 caractères hexadécimaux (0..9, A..F), un entier non signé comprend 4 octets, soit 8 caractères hexadécimaux (ou 32 bits).

Exemple :



Par exemple pour une largeur d'image de 854_{10} pixels. Si l'on représente cette valeur en hexadécimal, cela donne 356_{hex} . Cette valeur est sauvee dans un entier non signé (unsigned int ou uint) qui utilise 4 octets (32 bits). Cette valeur en mémoire sera de 00 00 03 56.

L'octet de droite (56) est l'octet de poids faible et celui de gauche (00) l'octet de poids fort. Lorsque ce nombre est transféré par exemple dans un fichier sur le disque, il sera lu octet par octet, ceci soit de droite à gauche, i.e. *little endian* (poids faible en premier), soit de gauche à droite, i.e. *big endian* (poids fort en premier).

Dans votre code, vous devrez veiller à avoir choisi la même convention pour l'écriture (LabVIEW) et la lecture (C++) du fichier.

Lecture de valeurs binaires

L'opérateur ">>" fonctionne uniquement pour les fichiers au format texte. Dans le cas de fichiers binaires, vous devez employer la méthode `read()`, au format suivant :

```
istream& read (char* s, streamsize n);
```

Avec

s: un pointeur sur un tableau de caractères (buffer)
n: un entier indiquant le nombre de byte(s) à lire

La méthode `read()` va lire les **n** caractères suivants dans le fichier et va les mettre dans le buffer **s** passé en paramètre. Le buffer **s** contient des caractères, si nous passons un `unsigned int` comme buffer, le compilateur va nous indiquer que le format n'est pas correct (`unsigned int ≠ char*`). Il faut indiquer au compilateur qu'il doit interpréter l'espace mémoire occupé par le `unsigned int` (4 bytes) comme étant un tableau de caractères de la même taille (4 bytes). Pour cela il faut employer l'instruction `reinterpret_cast<>`

Ex.

```
unsigned int tmp;  
f_in.read(reinterpret_cast<char *>(&tmp), sizeof(tmp));
```

avec

`&tmp` retourne l'adresse mémoire de `tmp`
`sizeof(tmp)` retourne la taille mémoire occupée par `tmp`, (4 bytes)

Gestion des erreurs

Il est primordial de gérer les erreurs dans votre code. Vous devez donc tester les fonctions que vous appelez, typiquement avec `.fail()` pour les fonctions sur les *stream* (fichier, string) ou une autre convention si vous appelez l'une des fonctions que vous avez vous-même définies.

En cas d'erreur, il faut afficher un message dans la console d'erreur **cerr** en procédant de la même manière que pour **cout**. Ensuite, vous devez sortir de votre fonction ou de votre programme.

Ex :

```
f_in.read(reinterpret_cast<char *>(&Largeur), sizeof(Largeur));
if (f_in.fail()) {
    cerr << "erreur lors de la lecture de la largeur";
}
else if (Largeur <=0) {
    cerr << "Largeur doit etre > 0";
}
else ...
```

Vos fonctions devront également retourner une erreur (si cela fait sens). Cette erreur peut être un **bool** ou un **int**.

Validation

Une fois votre code compilé, vous validez votre programme en effectuant des tests avec différents fichiers. Effectuez d'abord des tests avec des valeurs plausibles et valides, et ensuite avec des valeurs fausses ou impossibles. Vous contrôlerez par ailleurs que votre programme suit les spécifications définies précédemment.

Exemples de tests de validation:

- Largeur ou hauteur dans les bornes [10..1000]
- Le nombre de pixels ne correspond pas à largeur x hauteur
- Problème lors de la lecture ou de la création des fichiers

Evaluation de votre code

Lors de la soumission, votre code C++ sera testé automatiquement avec en premier des valeurs plausibles et ensuite avec des erreurs. Les erreurs possibles sont par exemple :

- Largeur et/ou hauteur en dehors de bornes [10..1000]
- Trop de pixels
- Pas assez de pixels
- Fichier d'entrée manquant (pixmap.bin)
- Impossible de lire dans le fichier d'entrée (pixmap.bin)
- Impossible d'écrire dans le fichier de sortie (pos.txt)
- Plusieurs pucks de la même couleur

2^e partie : le programme LabVIEW

Le programme LabVIEW est l'acteur principal du projet. C'est lui qui va faire le lien avec les deux autres programmes (C++ et matlab).

Gestion de l'interface utilisateur et génération des paramètres

La possibilité de générer facilement les interfaces graphiques est l'une des forces de LabVIEW. Il suffit de quelques clics de souris pour placer les éléments graphiques de votre interface.

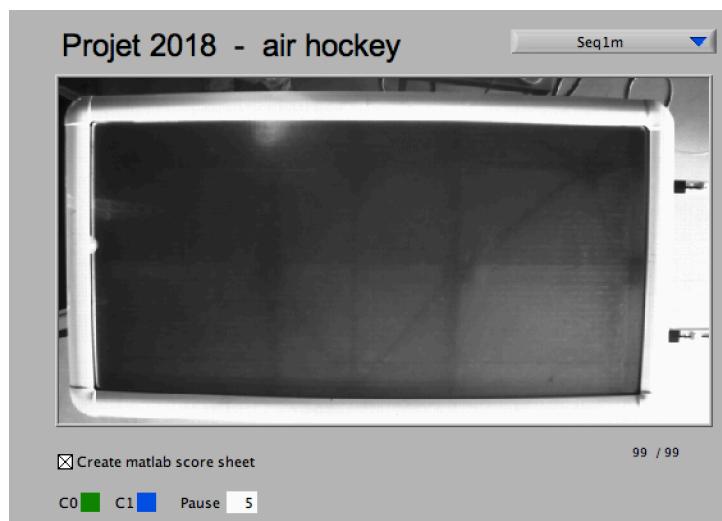


Figure 5. Interface graphique LabVIEW

Votre interface devra permettre de sélectionner la séquence à convertir et d'afficher son contenu (image/frame). Elle doit aussi permettre de définir les paramètres d'affichage des courbes dans la partie matlab, ceci à l'aide de boutons, de sliders ou d'autres éléments de votre choix.

Une fois la séquence sélectionnée et les paramètres d'affichage définis, le VI est lancé. La première étape consiste à lister les images contenues dans le dossier sélectionné. Le VI **List Folder** retourne la liste des fichiers et dossiers contenus dans le dossier passé en paramètre. Il est possible de restreindre les fichiers retournés en spécifiant un critère de recherche, par ex. « *.png ». L'étape suivante consiste à lire un à un les fichiers PNG, à les afficher et à trouver les coordonnées des 2 pucks. Le VI **Read PNG File.vi** lit un fichier .png, décode l'image au format .png et stocke dans un *cluster* les informations concernant l'image. A l'aide de **Draw Flatten Pixmap.vi**, vous pouvez afficher dans un *picture indicator* l'image décodée contenue dans le *cluster*.



List Folder

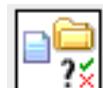


Read PNG File.vi



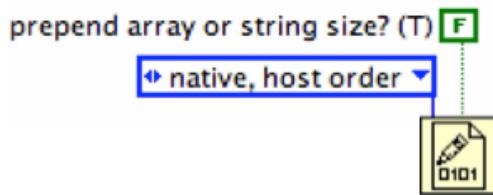
Draw Flatten Pixmap.vi

A vous de gérer les erreurs potentielles. Vous devez par exemple tester la présence du fichier à l'aide du VI **Check if File ou Folder Exists.vi** dans la palette file/advanced, ou créer votre propre test.



Check if File ou Folder Exists.vi

A l'aide du VI **Write to Binary File**, vous devez ensuite sauver les données collectées dans le fichier **pixmap.bin** en respectant le format défini dans la partie C++. Veillez à spécifier l'agencement des octets dans le fichier binaire.



Write to Binary File

Afin de ne pas ajouter la taille du tableau au début du fichier mettez **prepend array or string size? (T)** à faux (F). De même vous devez ajuster *l'endianess* des données sauvee sur votre disque dur (**little-endian** ou **native, host order**), cette option dépend de votre OS.

Une fois le fichier sauvé, exécutez votre programme **Pix2Pos** en employant le VI **System Exec**.



System Exec

Après l'exécution de votre programme C++, le fichier **pos.txt** contenant les coordonnées des 2 pucks devrait être présent sur le disque dur.

N'oubliez pas de traiter les erreurs retournées par **System Exec**. Vous devrez répéter les opérations ci-dessus pour chaque image de la séquence en mémorisant les positions intermédiaires que vous intégrerez au Script matlab.

Génération du script Matlab

Le fichier **Analyse.m** contiendra les coordonnées successives des 2 pucks ainsi que le script pour analyser et afficher la feuille de résultat (Score sheet). Les trajectoires des 2 pucks peuvent être affichées de différentes manières, en fonction des choix de l'utilisateur. Ces choix sont paramétrés dans l'interface LabVIEW avant d'être convertis en une suite de commandes Matlab (script). Les spécifications complètes du script Matlab sont décrites dans la troisième partie.

Depuis LabVIEW, l'utilisateur doit pouvoir choisir au minimum les options suivantes :

- Paramètres (couleur, etc.) des courbes
- Temps d'attente pour l'affichage dans matlab

Un fois le script généré, il faut le sauver sur le disque dur puis l'appeler à l'aide du VI fourni **MP_LaunchMatlabScript.vi**



MP_LaunchMatlabScript.vi

3^e partie : le programme Matlab

Lecture, analyse, affichage et sauvegarde du graphique représentant les traces des pucks

Vous devez écrire le script **Analyse.m** Matlab qui effectue les opérations suivantes :

- Chargement des 2 courbes/traces représentant les positions successives de chaque pucks
- Affichage des 2 courbes
- Affichage des 4 bandes du billard
- Affichage des points d'impact des pucks avec les différentes bandes
- Affichage des points d'impact entre les pucks
- Affichage du nom de la séquence
- Affichage de la date (et heure) courante
- Affichage du nombre de bandes touchées par chaque puck
- Affichage du nombre de chocs entre pucks
- Sauvegarde de la figure générée au format PDF. Le nom du fichier est **AH_Sheet.pdf**

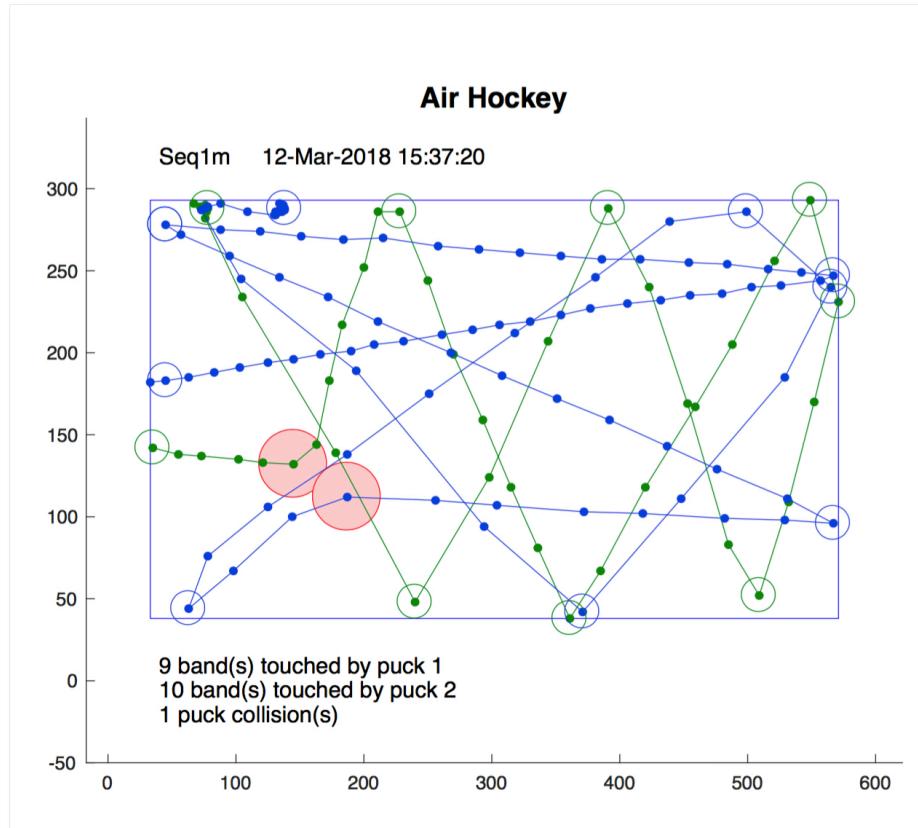


Figure 6. Le fichier PDF contenant le résultat (AH_Sheet.pdf).

Il existe plusieurs options pour passer les paramètres d'affichage de LabVIEW à Matlab. L'option la plus simple consiste à générer le fichier **Analyse.m** à chaque exécution. Autre option possible: utiliser un fichier supplémentaire contenant uniquement les paramètres et les points des courbes, le fichier d'analyse restant identique.

Voici quelques suggestions pour mener à bien votre analyse de la partie.

- 1) Un puck touche une bande lorsqu'une de ses coordonnées est proches d'une des coordonnées du rectangle représentant le billard.
- 2) Un puck touche l'autre puck lorsqu'ils ont une position proche, au même moment (i.e. même image) lorsque la direction des pucks changent et que ce n'est pas un ricochet sur une bande.
- 3) Un puck fait environ 15 pixels de diamètre
- 4) La numérisation des positions successives de pucks a été faite manuellement, ce qui veut dire qu'il y a une marge d'erreur (d'environ ± 5 pixels) pour chaque coordonnée.

L'hypothèse suivante peut être faite: toutes les bandes du billard sont touchées par les pucks.

Pour afficher du texte, utilisez la fonction **text(xx,yy,txt,...)**

Pour sauver la figure générée, utilisez la fonction **print('-dpdf',...)**

L'appel de votre script Matlab **Analyse.m** se fait depuis LabVIEW à l'aide du VI fourni **MP_LaunchMatlabScript.vi**

Points importants

- **Tous les fichiers doivent se trouver dans le même répertoire.**
- Merci de respecter les noms de fichiers, vos programmes seront testés par des outils automatiques