**Documentation pierre-feuille-ciseaux**

# But

Cette station permet à un utilisateur de jouer au pierre-feuille-ciseaux contre un robot, qui réagit en fonction de ce qu’à fait l’utilisateur. Ainsi, le robot ne peut pas perdre.

# Utilisation

Pour que le programme fonctionne bien, il faut d’abord bien régler la caméra. Cela se fait via le script “configuration.py”, qui permet de modifier les paramètres avec un clavier. Les boutons sont les suivants :

* a/q pour modifier les fps
* z/s pour modifier le temps d’exposition (il y a un petit lag lors du changement)
* e/d pour modifier la luminosité
* r/f pour modifier le gain
* t/g pour modifier le gamma

Pour un réglage optimal, il faut que le fond de l’image soit saturé, et qu’un objet au premier plan ne le soit pas (voir image ci-dessous).

Une fois les réglages faits, il faut appuyer sur la touche “Entrée” pour valider cette configuration. Si on souhaite quitter sans sauvegarder ces changements, il faut appuyer sur la touche “Echap”.

Une fois que tout est bien configuré, le programme “recognition.py” devrait fonctionner sans soucis. L’objet au premier plan est reconnu puis analysé pour donner une réponse au robot.

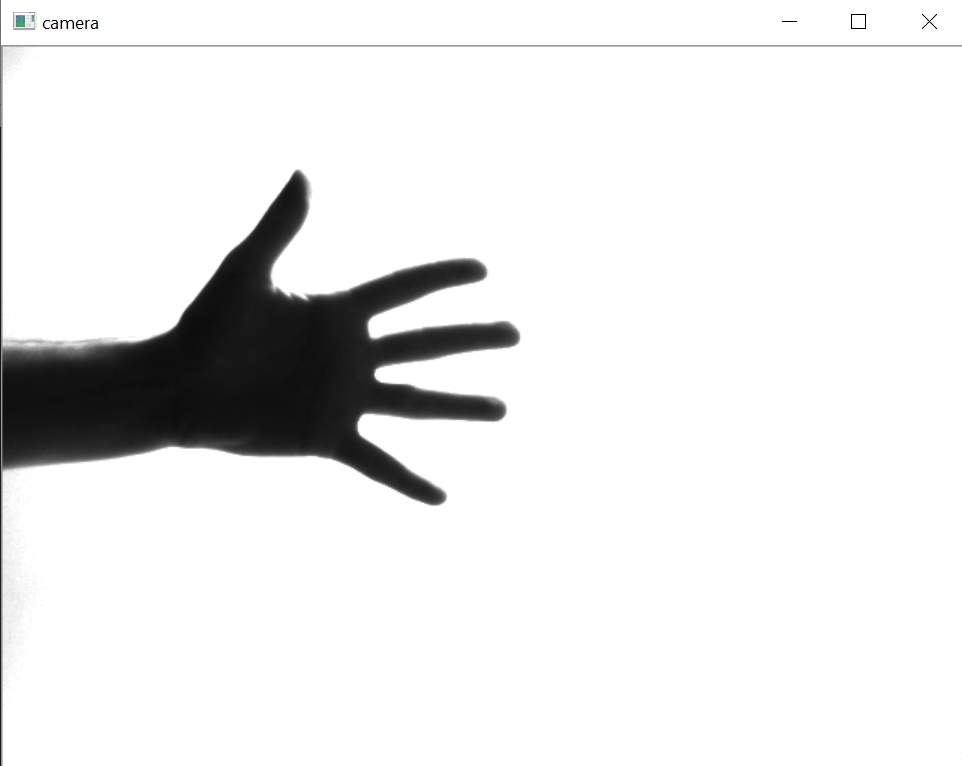


Image d’une main avec un fond saturé

# Fonctionnement

De manière général, le fonctionnement de cette station est inspiré de ce qui est décrit dans l’article scientifique suivant:

K. Ito, T. Sueishi, Y. Yamakawa and M. Ishikawa, "Tracking and recognition of a human hand in dynamic motion for Janken (rock-paper-scissors) robot," *2016 IEEE International Conference on Automation Science and Engineering (CASE)*, Fort Worth, TX, 2016, pp. 891-896, doi: 10.1109/COASE.2016.7743496.

Le fonctionnement général est le suivant : une caméra capte les images dans la zone de jeu, et lorsqu’un objet entre dans cette zone, sa forme est analysée afin de prédire quelle est la forme de la main, et donc transmettre au robot les mouvements qu’il doit faire.

On va ici s’intéresser à l’analyse de l’image. Elle se découpe en plusieurs parties :

* capture de l’image
* extraction de la forme de l’objet
* analyse de la forme de l’objet

Tout le code est fait en python, et en utilisant la bibliothèque OpenCv, qui permet de manipuler des flux vidéos très facilement.

## Capture de l’image

Pour récupérer une image avec OpenCv, il suffit d’avoir une caméra branchée sur l’ordinateur, avec les bons drivers d’installés. Dans notre cas, la caméra utilisée est une “zwo asi”, et il faut installer le driver “ASI Camera” **ET** le driver “DirectShow”.

Ensuite, il faut configurer les paramètres de la caméra, afin d’avoir une image où l’objet en premier plan ressort le plus du fond. Après plusieurs tests, la méthode la plus efficace et la plus rapide consiste à avoir un fond qui sature l’image.

Pour modifier les paramètres de capture de la caméra, il faut utiliser le script “configuration.py”, qui permet de modifier en direct ces paramètres, tout en visualisant le rendu de la caméra.

Une fois le script lancé, une fenêtre nous montrant la capture de la caméra s’ouvre. On peut alors modifier les paramètres avec le clavier :

* a/q pour modifier les fps
* z/s pour modifier le temps d’exposition (il y a un petit lag lors du changement)
* e/d pour modifier la luminosité
* r/f pour modifier le gain
* t/g pour modifier le gamma

Une fois les modifications faites, il faut appuyer sur “Entrée” pour valider les paramètres. Si on veut annuler les modifications pour garder les anciens paramètres, il faut appuyer sur “Echap”.

Un autre paramètre théoriquement modifiable est la résolution de la fenêtre de capture. Elle est réglée à 640x480, mais peut être changée directement dans le code source si besoin. Attention : il faut changer ces paramètres dans les 2 fichiers “configuration.py” et “recognition.py”.

La configuration de la caméra est enregistrée dans un fichier csv (“properties.csv”). Ce fichier est ensuite chargé lors du lancement du script “recognition.py”.

## Extraction de la forme

Si la caméra est bien configurée, lorsqu’un objet est dans la zone de jeu, celui-ci apparaît à la caméra, alors que le fond est lui saturé.

Pour commencer, on floute légèrement l’image. Cela permet de supprimer l'éventuel bruit de l’image.

Ensuite, on effectue une binarisation de l’image : cela signifie que les pixels de l’image sont soit noirs (le fond), soit blancs (l’objet). Pour cela, on utilise simplement un seuillage, d’une valeur assez haute puisque le fond est saturé. Le seuil doit être situé entre 0 (noir) et 255 (blanc). Il vaut pour l’instant 220, mais est modifiable directement dans le code source si la binarisation est mauvaise (variable “threshold\_value” dans les hyperparamètres, en début de code).

Une fois la binarisation faite, on applique une fermeture morphologique. Cela permet de de combler d’éventuels trous dans l’objet. C’est une opération qui peut être optionnelle, mais elle est extrêmement rapide, donc la supprimer ne fait pas gagner de temps.

Dans une première version du projet, il y avait un premier prétraitement de l’image, qui consistait à soustraire de l’image actuelle l’image du fond. C’est une opération un peu coûteuse, et très utile quand le fond n’est pas très uniforme, mais inutile dans notre cas, où le fond est de toute façon saturé.

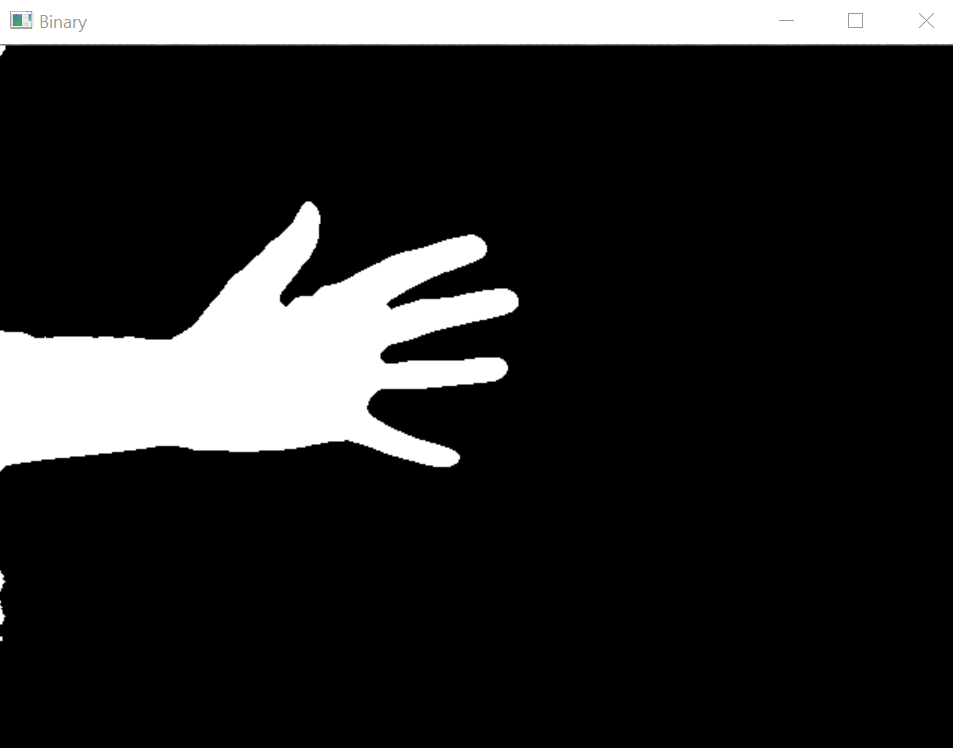


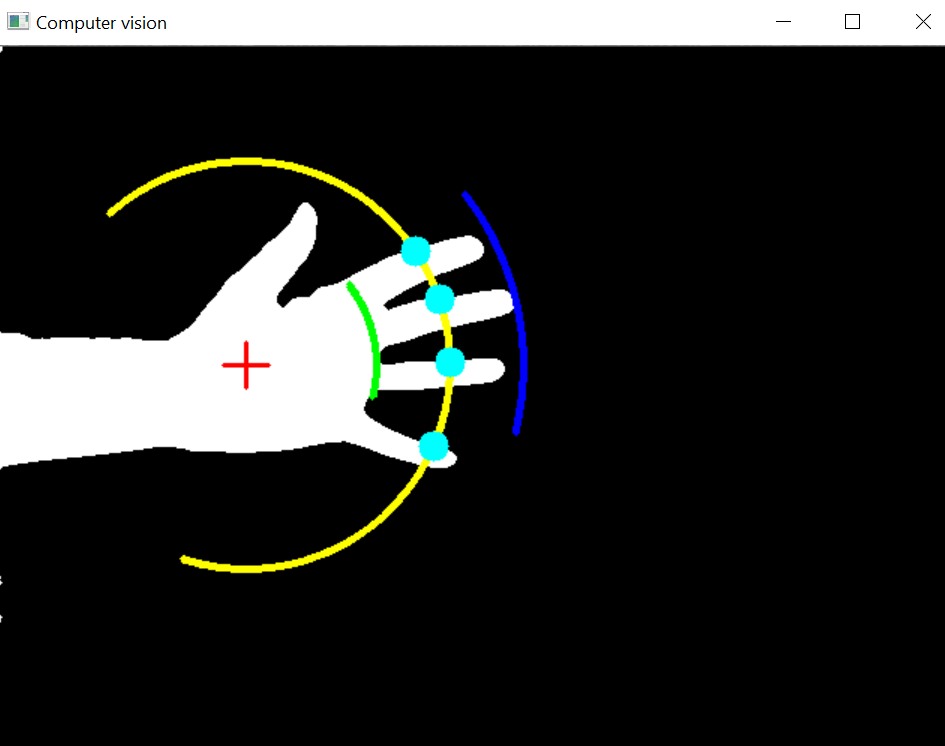
Image binarisée de la main : l’objet est extrait du fond

## Analyse de la forme

Pour l’analyse de la forme, on utilise la méthode décrite dans l’article cité au début du document.

Pour la simplicité de compréhension du code, les fonctions mathématiques créées pour ce programme ont été regroupées dans le fichier “tools.py”.

On va rechercher différentes caractéristiques de l’objet, puis les analyser pour trouver un résultat. Les caractéristiques trouvées sont affichées sur une fenêtre, permettant de mieux comprendre ce que fait le programme (image ci-dessous).



Vision de l’ordinateur

Tout d’abord, on vérifie s’il y a bien un objet dans l’image. Pour cela, on compte simplement le nombre de pixels blancs, et si la proportion est assez importante, on considère qu’un objet est présent.

Ensuite, on trouve le centre de l’objet, ainsi que son axe principal. Le centre est représenté en rouge sur l’image, et les arcs de cercle créés plus tard ont ce même centre, et sont orientés selon l’axe principal.

Dans la suite, on cherche 2 longueurs particulières, nommées Ltip et Lmin dans le code et dans l’article. Ltip correspond à la distance entre le centre et l'extrémité des doigts, et Lmin à la distance entre le centre et la base des doigts. Ltip est en bleu et Lmin en vert sur l’image.

De manière plus détaillée, pour trouver Ltip, on va générer un arc de cercle à une certaine distance, puis regarder si celui-ci a des pixels blancs. Si c’est le cas, on va augmenter le rayon, et inversement on le diminue si tous les pixels sont noirs. On met ainsi en place une recherche dichotomique de Ltip. Le raisonnement est le même pour Lmin, où on regarde s’il y a des pixels noirs.

Une fois Ltip et Lmin déterminés, on va pouvoir les comparer. Si Ltip/Lmin est proche de 1, cela signifie que les 2 distances sont proches, donc que le poing est fermé. On sait donc que le joueur a joué pierre. Sinon, on considère que la main est ouverte, et donc que le joueur a fait soit feuille, soit ciseaux. Il faut maintenant les distinguer.

Pour cela, on crée un 3ème arc de cercle (en jaune), à une distance Ljudge, qui dans ce programme est la moyenne de Ltip et Lmin. Cet arc de cercle passe logiquement par tous les doigts ouverts de la main, et il suffit alors de les compter pour prendre une décision. Sur l’image, chaque point cyan représente le début d’un nouveau doigt rencontré par l’arc de cercle. Si on compte 3 doigts ou moins, cela signifie que le joueur a fait ciseaux. Sinon, il a joué feuille.

On peut alors envoyer au robot le résultat de l’analyse de l’image.