Reconnaissance des coups joués

Le programme de reconnaissance visuelle des coups joués s’articule autour de plusieurs étapes : tout d’abord une prise de vue de l’échiquier. Puis la photo est analysée pour y détecter l’échiquier. Les différentes cases de l’échiquier sont ensuite traitées pour définir leur état d’occupation (vide, pièce blanche, pièce noire). Enfin, la configuration de l’échiquier est comparée à la précédente pour définir quel coup vient d’être joué.

Le programme à exécuter se situe dans le fichier par2.py. Toute la partie peut alors se dérouler.

Contenu

[Prise de vue 3](#_Toc484644482)

[Détection de l’échiquier 4](#_Toc484644483)

[Détection du contour de l’échiquier 4](#_Toc484644484)

[Recherche du centre de l’échiquier 5](#_Toc484644485)

[Calcul de la position des cases de l’échiquier 6](#_Toc484644486)

[Analyse des cases 8](#_Toc484644487)

[Définition de la couleur moyenne des cases 8](#_Toc484644488)

[Détermination de l’état d’occupation des cases 8](#_Toc484644489)

[Rotation de la matrice d’occupation 9](#_Toc484644490)

[Détection du coup joué 11](#_Toc484644491)

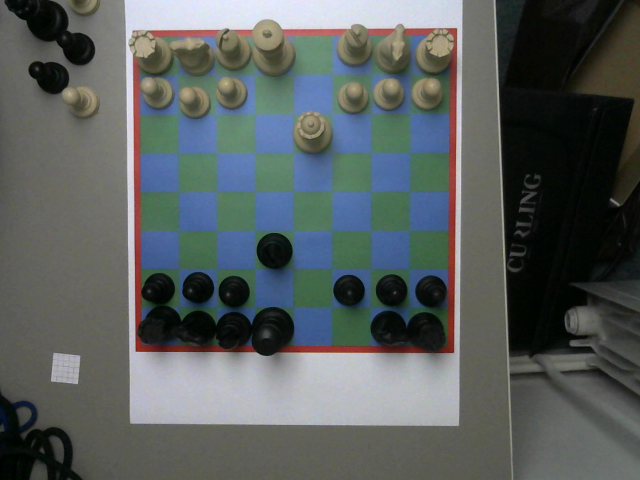
[Coup classique 11](#_Toc484644492)

[Roque 11](#_Toc484644493)

[Prise en passant 11](#_Toc484644494)

# Prise de vue

Cette partie est assurée par le fichier prisephoto.py en utilisant OpenCV. La fonction photo() affiche dans une nouvelle fenêtre la capture vidéo prise par une caméra branchée sur un port USB. Dès que l’utilisateur appuie sur une touche du clavier, la dernière photo prise par la caméra est sauvegardée et retournée en tant que « frame ». Il s’agit d’un tableau tridimensionnel. Ainsi, frame est un tableau dont chaque élément est encore un tableau représentant une ligne de pixels de l’image. Les éléments d’une ligne sont encore un tableau représentant un pixel donné. Enfin, un pixel est représenté par un tableau donnant ses valeurs de couleur : bleu puis vert et rouge. Chaque valeur de couleur est un entier compris entre 0 et 255. Par simplicité, on notera par la suite r la valeur de la composante rouge d’un pixel, v sa valeur de vert et b sa valeur de bleu. La photo suivante est un exemple de prise de vue sauvegardée à cette étape du code. Dans ce cas, d’après la résolution de la caméra utilisée, la taille de « frame » est 480\*640\*3.

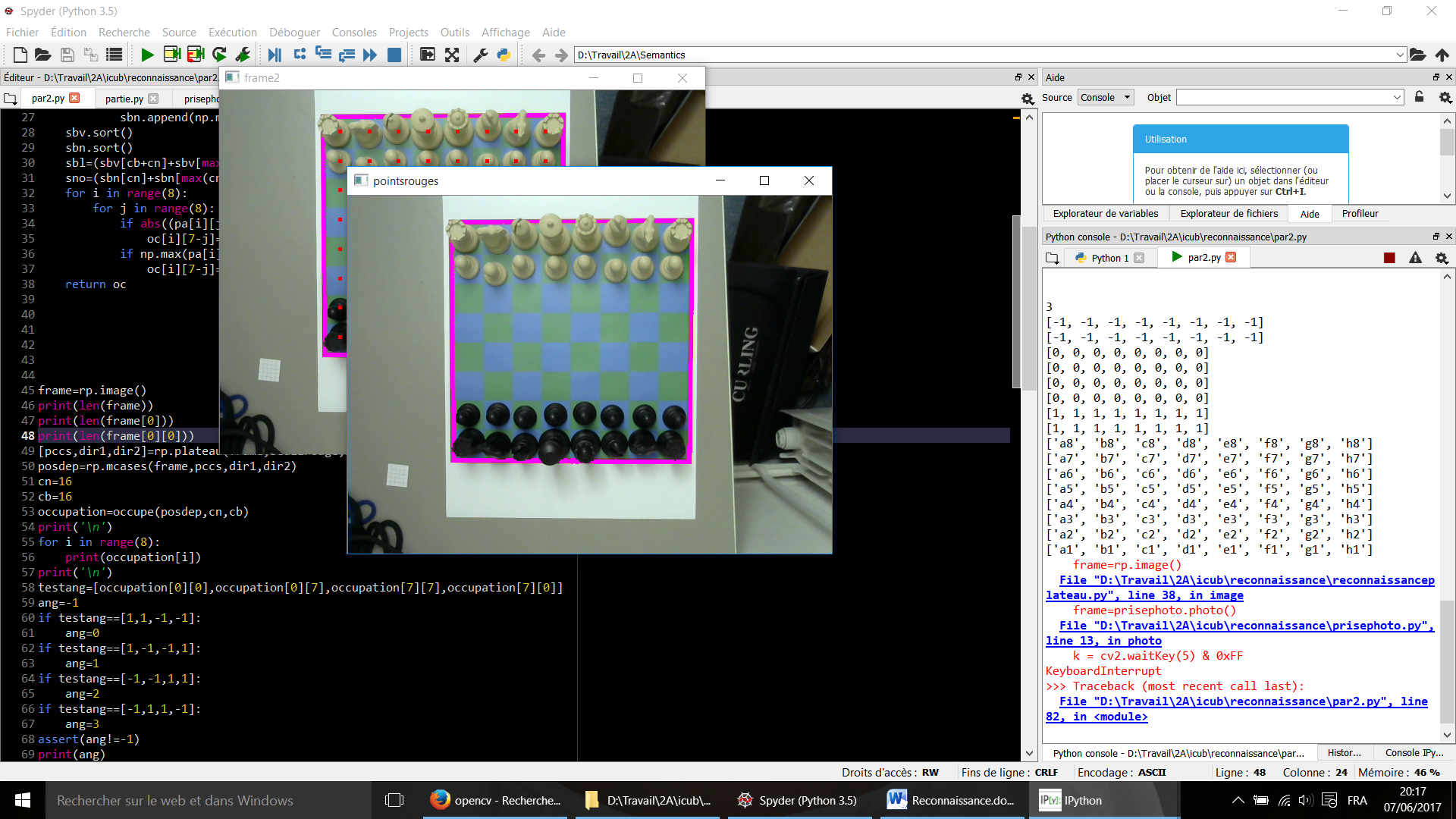


# Détection de l’échiquier

La détection de l’échiquier se fait par la fonction plateau(frame,seuilrouge,seuilrb,seuilrv) du fichier reconnaissanceplateau.py. Elle prend comme argument « frame », un tableau tridimensionnel représentant la photo prise précédemment, ainsi que trois nombres servant de seuils pour détecter les pixels du tour de l’échiquier. Et elle retourne « pccs », « dir1 », et « dir2 » qui sont respectivement un tableau de taille 8\*8\*2 contenant pour chaque case de l’échiquier les coordonnées calculées du pixel situé au centre de la case, et les deux vecteurs utilisés dans l’image pour définir le carré situant l’échiquier dans la photo. Le principe est celui-ci : l’échiquier, dont les cases sont vertes et bleues pour aider la reconnaissance des pièces, est encadré de rouge. Pour définir la position de l’échiquier, on cherche les pixels rouges de la photo puis on cherche à en extraire un carré qui définit le contour de l’échiquier. Sachant que l’échiquier est un carré de 8 cases de côté, les différentes cases sont situées en fonction du carré ainsi trouvé.

## Détection du contour de l’échiquier

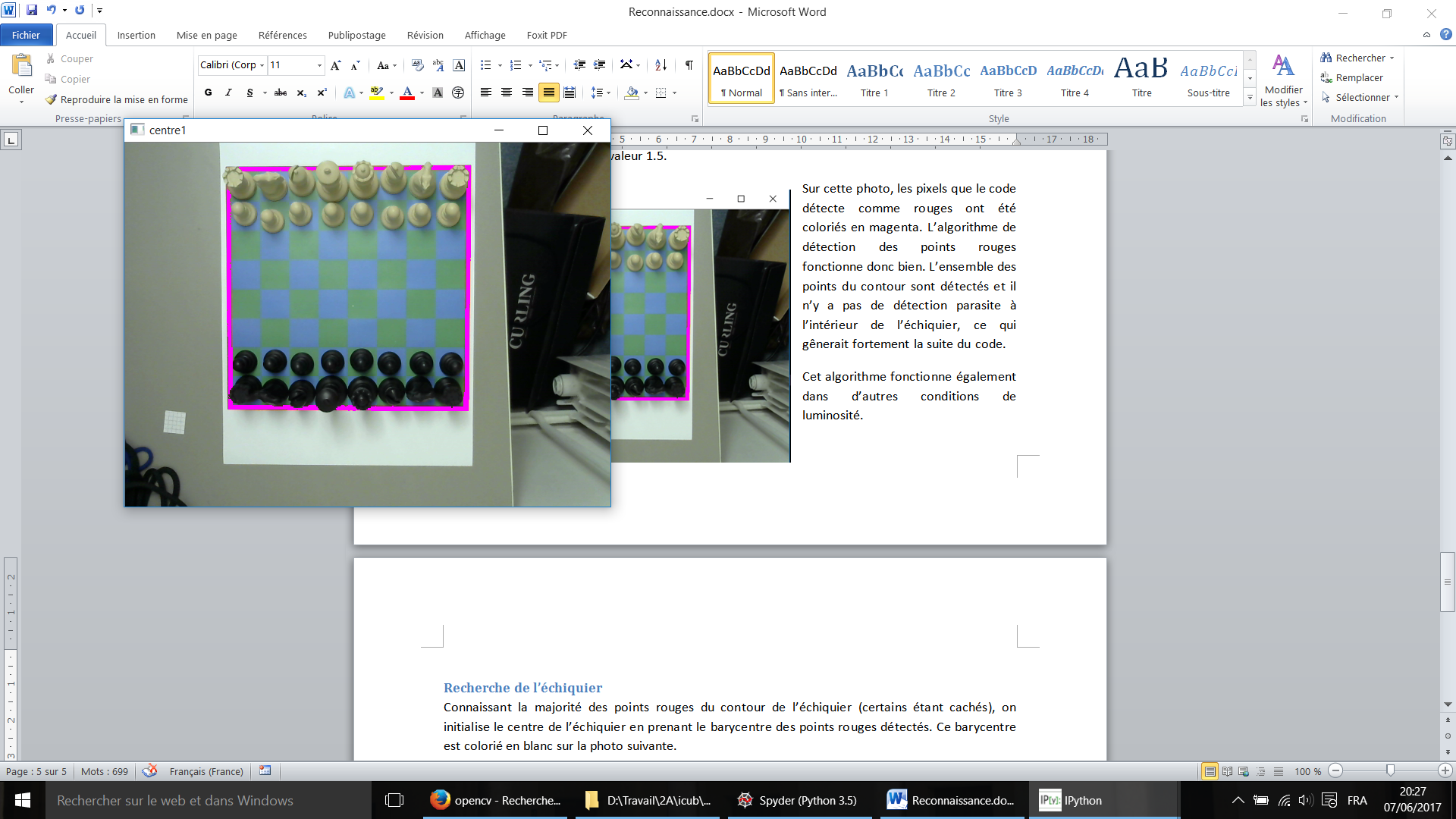
L’échiquier étant encadré de rouge, on cherche les points rouges dans l’image. Parmi tous les pixels de l’image, on commence par tester si r>seuilrouge (seuilrouge étant un paramètre de la fonction « plateau »). Cette valeur de seuil est en pratique utilisée dans le fichier par2.py en prenant la valeur 75. Cette étape permet de s’assurer que les pixels noirs ne soient pas retenus. En effet, les deux tests suivants sont des tests relatifs entre les différentes composantes du pixel. La valeur minimale de 75 de rouge permet donc d’assurer que le pixel considéré n’est pas trop sombre, et il s’agit de la seule composante utilisable pour un tel seuil puisque les pixels à détecter peuvent n’avoir ni vert ni bleu. Parmi les pixels ayant r>75, on vérifie ensuite que r>v\*seuilrv et r>b\*seuilrb avec seuilrv et seuilrb paramètres de la fonction « plateau ». En pratique, ces deux valeurs de seuils prennent la valeur 1.5.

Sur cette photo, les pixels que le code détecte comme rouges ont été coloriés en magenta. L’algorithme de détection des points rouges fonctionne donc bien. Une grande majorité des points du contour sont détectés et il n’y a pas de détection parasite à l’intérieur de l’échiquier, ce qui gênerait fortement la suite du code.

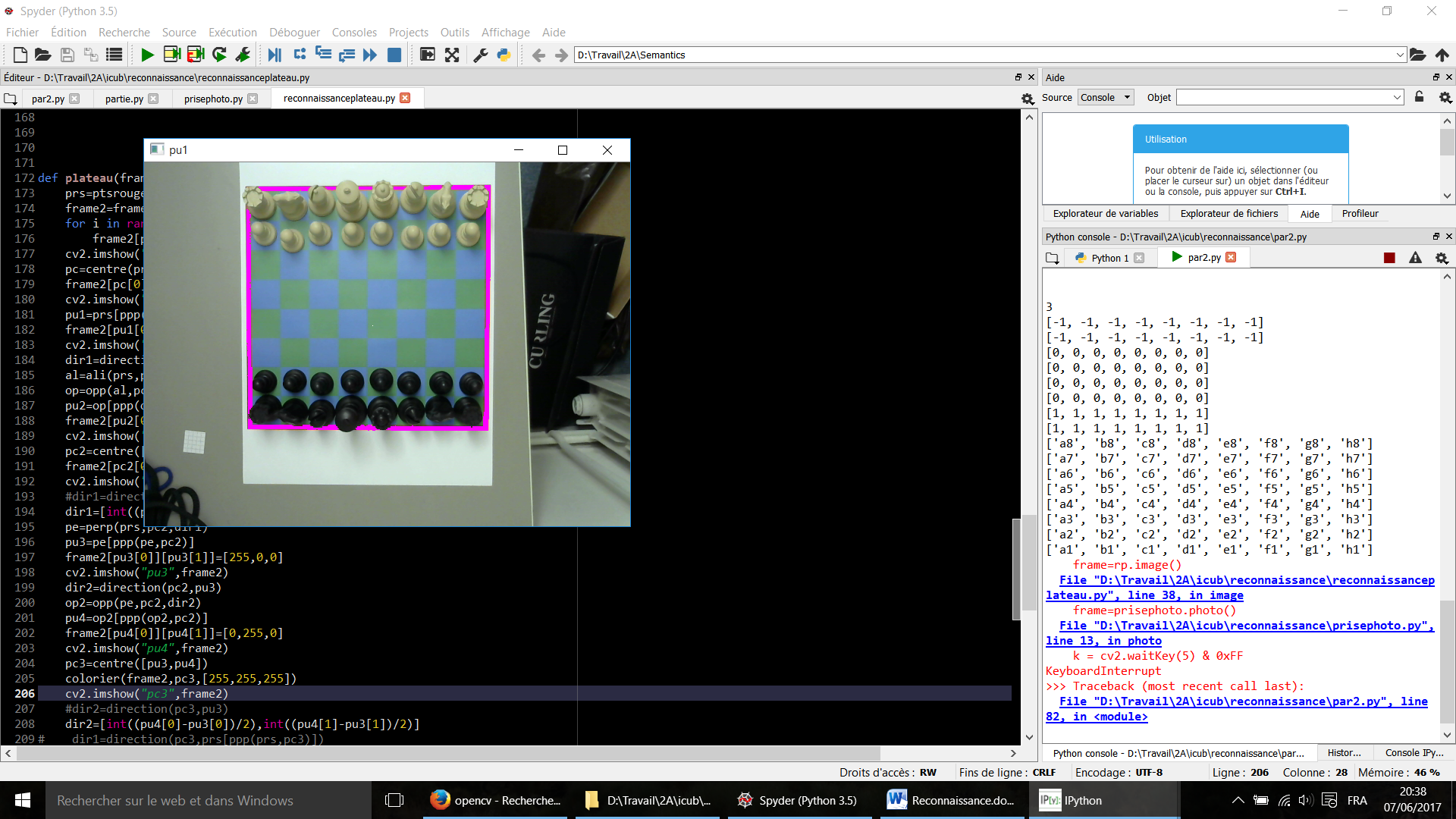
Cet algorithme fonctionne également dans d’autres conditions de luminosité.

## Recherche du centre de l’échiquier

Connaissant la majorité des points rouges du contour de l’échiquier (certains étant cachés), on initialise le centre de l’échiquier en prenant le barycentre des points rouges détectés. On appellera ce point PC1 (comme point central). Il est colorié en blanc sur la photo suivante, au centre du cercle rouge.

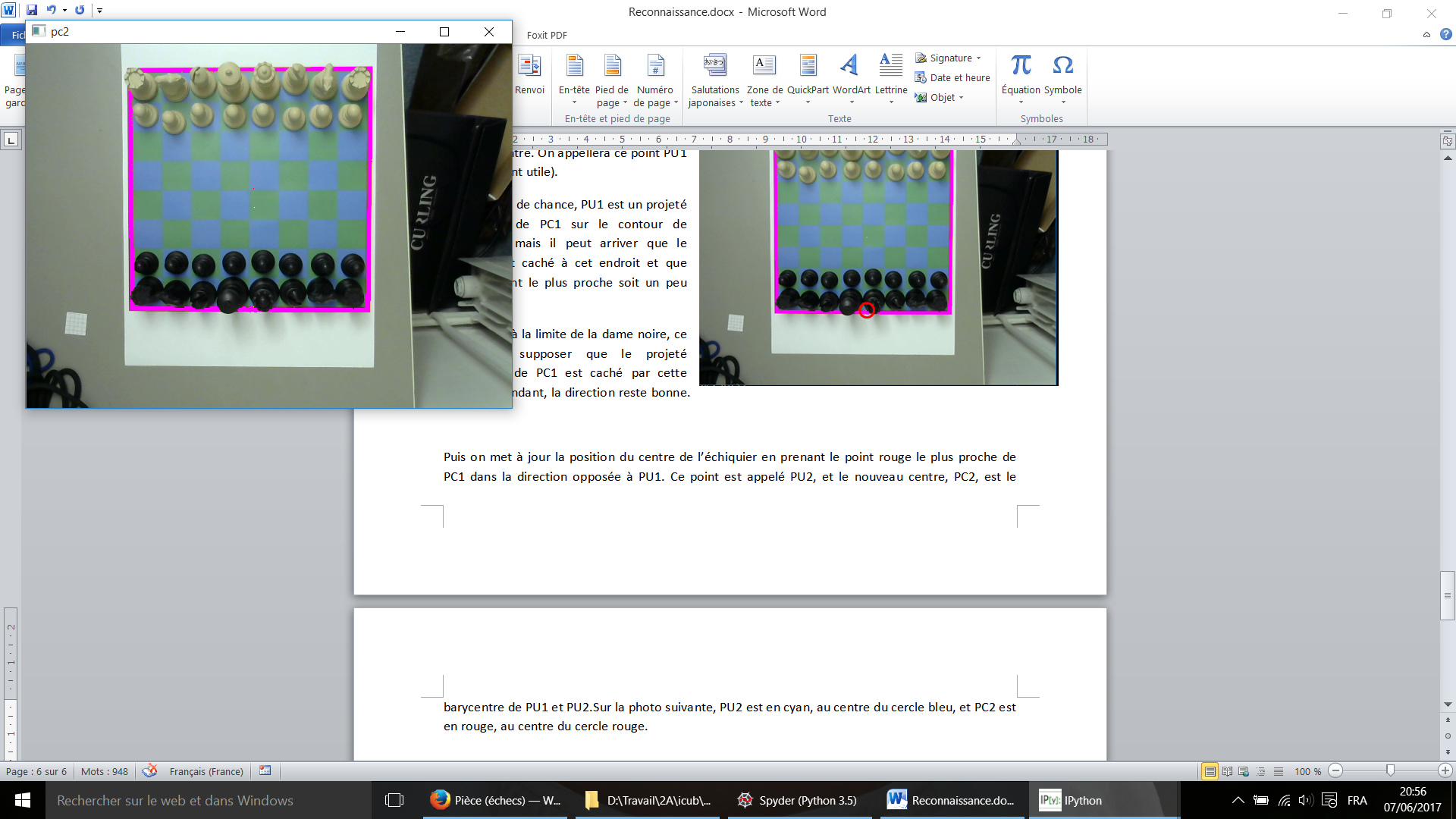
On peut constater que le contour de l’échiquier est moins caché du côté noir que du côté blanc, d’où le décalage de PC1 par rapport au centre de l’échiquier vers les pièces noires.

Cependant, ce décalage n’est pas gênant tant que PC1 reste à l’intérieur de l’échiquier (ce qui n’est pas garanti puisque d’autres points rouges pourraient être détectés dans l’environnement).

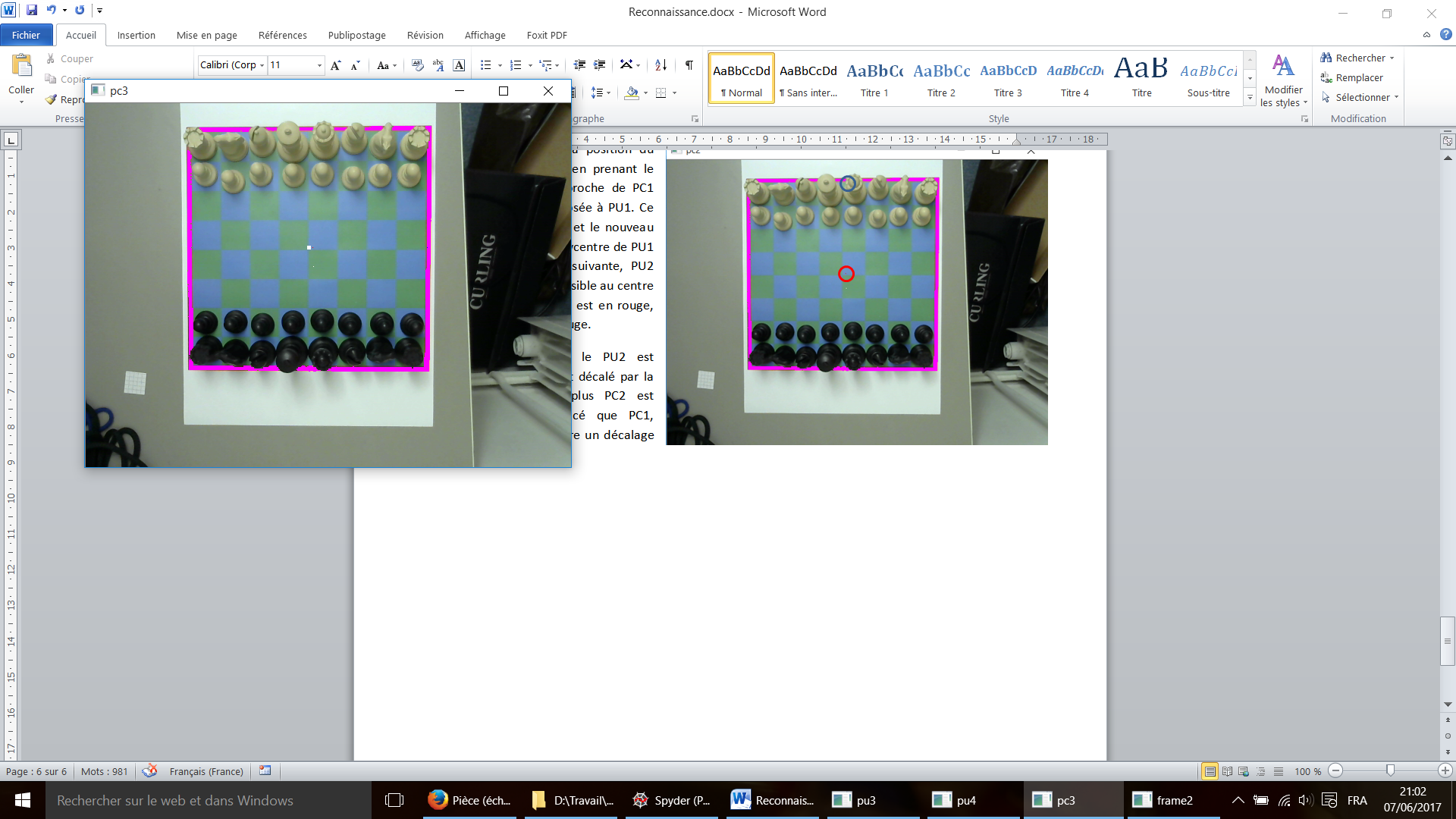
Puis on recherche parmi les points rouges celui qui est le plus proche de PC1, pas rrès visible en jaune au centre du cercle sur la photo ci-contre. On appellera ce point PU1 (comme point utile).

Avec un peu de chance, PU1 est un projeté orthogonal de PC1 sur le contour de l’échiquier, mais il peut arriver que le contour soit caché à cet endroit et que donc le point le plus proche soit un peu plus loin.

Ici, PU1 est à la limite de la dame noire, ce qui laisse supposer que le projeté orthogonal de PC1 est caché par cette dame. Cependant, la direction reste bonne.

Puis on met à jour la position du centre de l’échiquier en prenant le point rouge le plus proche de PC1 dans la direction opposée à PU1. Ce point est appelé PU2, et le nouveau centre, PC2, est le barycentre de PU1 et PU2.Sur la photo suivante, PU2 est en cyan, pas très visible au centre du cercle bleu, et PC2 est en rouge, au centre du cercle rouge.

On peut noter que le PU2 est également légèrement décalé par la dame blanche. De plus PC2 est beaucoup mieux placé que PC1, même s’il existe encore un décalage horizontal.

Enfin, les points PU3 et PU4 sont les points rouges les plus proches de PC2 de chaque côté, dans la direction perpendiculaire à celle donnée par PU1 et PU2. PU3 et PU4 sont en bleu et en vert, au centre des cercles bleus sur cette photo.

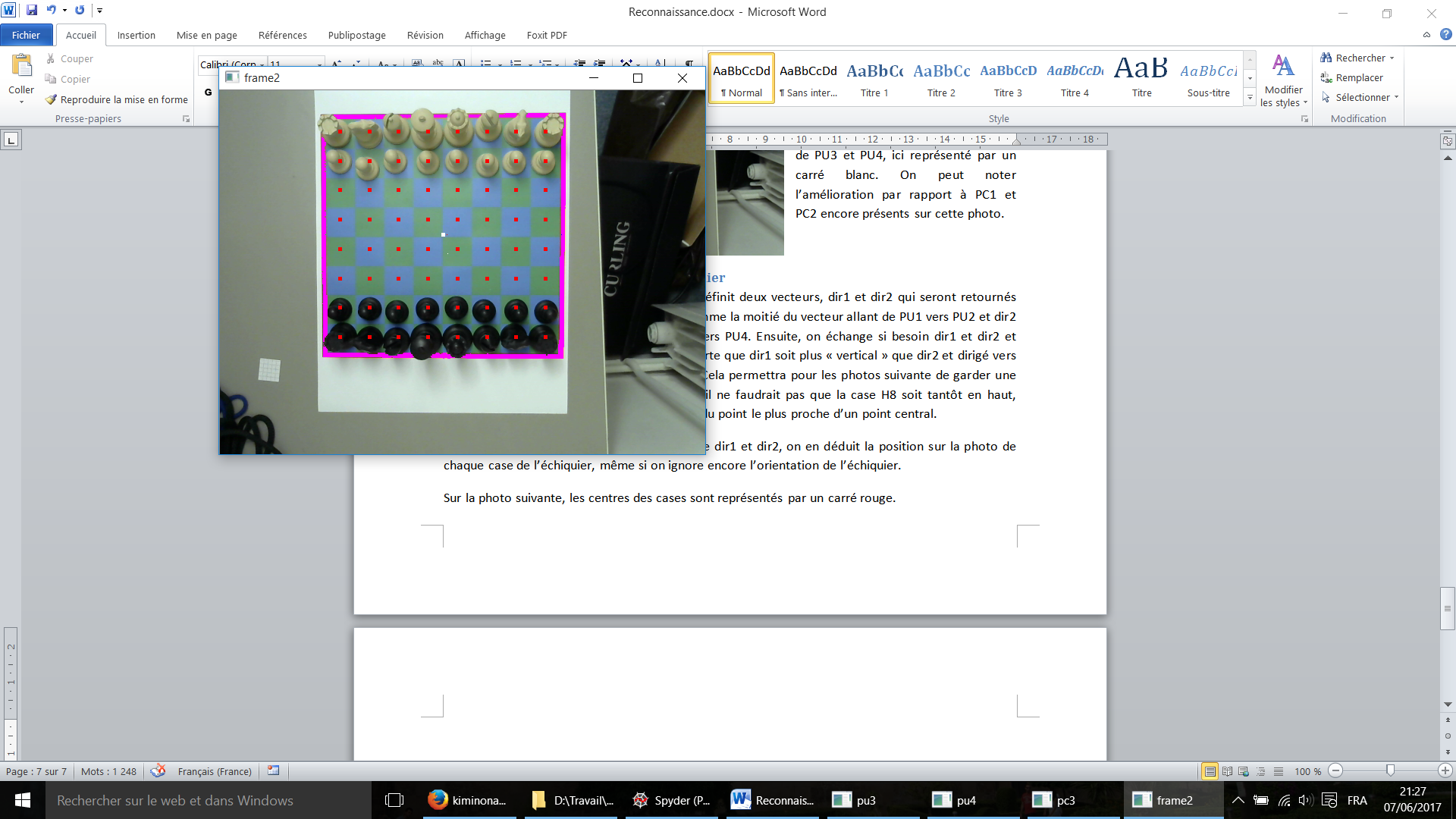
Le centre final, PC3 est le barycentre de PU3 et PU4, ici représenté par un carré blanc. On peut noter l’amélioration par rapport à PC1 et PC2 encore présents sur cette photo.

## Calcul de la position des cases de l’échiquier

Une fois le centre de l’échiquier trouvé, on définit deux vecteurs, dir1 et dir2 qui seront retournés par la fonction « plateau ». On définit dir1 comme la moitié du vecteur allant de PU1 vers PU2 et dir2 comme la moitié du vecteur allant de PU3 vers PU4. Ensuite, on échange si besoin dir1 et dir2 et inverse leur direction si nécessaire, de telle sorte que dir1 soit plus « vertical » que dir2 et dirigé vers le haut, et que dir2 soit dirigé vers la droite. Cela permettra pour les photos suivante de garder une unité dans le calcul des positions des cases, il ne faudrait pas que la case H8 soit tantôt en haut, tantôt en bas de l’échiquier, selon la position du point le plus proche d’un point central.

A partir de PC3 et par combinaison linéaire de dir1 et dir2, on en déduit la position sur la photo de chaque case de l’échiquier, même si on ignore encore l’orientation de l’échiquier. Les positions des cases sont enregistrées dans un tableau de taille 8\*8\*2, « pccs », que l’on retourne dans la fonction « plateau » avec les vecteurs dir1 et dir2.

Sur la photo suivante, les centres des cases sont représentés par un carré rouge.



On peut noter que l’échiquier est considéré comme carré, ce qui n’est pas le cas sur la photo lorsque celle-ci est prise avec un angle de vue trop éloigné de la verticale. S’il y a trop de perspective, les carrés rouges seront disposés en carré, mais ne coïncideront plus avec les centres de chaque case. Cette amélioration du code serait nécessaire pour reconnaître l’échiquier à partir des caméras de l’iCub, plutôt qu’à partir d’une caméra située au-dessus de l’échiquier. Cependant, cela serait également gênant au niveau des points rouges cachés par les pièces, et la détection des pièces serait aussi plus difficile car elles déborderaient sur les cases alentour. L’utilisation d’une caméra externe peut être dommage, mais elle simplifie grandement le problème de reconnaissance visuelle.

# Analyse des cases

## Définition de la couleur moyenne des cases

Connaissant le centre d’une case (donné dans pccs) et les vecteurs dir1 et dir2, il est possible de déterminer un carré constitué de pixels autours du centre de la case couvrant environ 4/9 de la case (soit un côté d’environ 2/3 du côté de la case). Cela permet de s’assurer que toute pièce sur cette case sera bien détectée, mais que celles qui pourraient déborder sur la case analysée auront une influence moindre. La fonction « mcases » du fichier reconnaissanceplateau.py utilise ces carrés pour calculer une couleur moyenne sur la case dont les composantes r,v et b sont des moyennes sur tous les pixels du carré centré sur cette case.

Les couleurs moyennes d’un échiquier dans sa position de départ sont rangées dans le graphe suivant. Chaque groupe de huit couleurs représente une colonne de l’échiquier dont les deux premières cases sont occupées par une pièce blanche et les deux dernières par une pièce noire.

## Détermination de l’état d’occupation des cases

On peut noter les grands écarts concernant la composante rouge entre les pièces blanches, les cases vides et les pièces noires. Chaque domaine étant bien séparé des autres, cette valeur semble appropriée pour définir l’état d’une case.

On définit un seuil pour séparer les cases occupées par une pièce blanche des autres cases, et un autre seuil pour séparer les cases occupées par une pièce noire des autres cases. Ces deux seuils sont calculés à chaque photo que l’on prend après chaque coup joué.

Pour cela, on initialise deux variables cn et cb à 16. Ces deux variables représentent respectivement le nombre de pièces noires et le nombre de pièces blanches présentes sur l’échiquier. Après chaque coup, on mettra à jour ces variables pour pouvoir calculer les seuils au coup suivant.

On remplit un tableau avec l’ensemble des valeurs moyennes de rouge de toutes les cases, puis on trie ce tableau et on choisit le seuil comme étant la moyenne entre la cn-1ème valeur et la cn+1ème valeur pour détecter les pièces noires. La cn+1ème valeur correspond à la plus petite valeur dont on est sûr qu’elle correspond à une case vide, alors que la cn-1ème valeur correspond à la plus grande dont on est sûr qu’elle correspond à une pièce noire. La cnème valeur pouvant être soit une pièce noire, soit une case vide, selon que le coup venant d’être joué et que l’on essaie de détecter a été une prise d’une pièce noire ou non. Le seuil moyen entre les valeurs sûres extrêmes doit pouvoir distinguer ces deux cas. Pour détecter les pièces blanches, on effectue la même opération, mais en triant le tableau par ordre décroissant, et en prenant cb à la place de cn.

On obtient ainsi un tableau ressemblant à celui-ci :

[-1, -1, 0, 0, 0, 0, 1, 1]

[-1, -1, 0, 0, 0, 0, 1, 1]

[-1, -1, 0, 0, 0, 0, 1, 1]

[-1, -1, 0, 0, 0, 0, 1, 1]

[-1, -1, 0, 0, 0, 0, 1, 1]

[-1, -1, 0, 0, 0, 0, 1, 1]

[-1, -1, 0, 0, 0, 0, 1, 1]

[-1, -1, 0, 0, 0, 0, 1, 1]

Les -1 représentent les cases occupées par une pièce noire, tandis que les 1 représentent les cases occupées par une pièce blanche.

## Rotation de la matrice d’occupation

Une rotation de cette matrice d’occupation peut être nécessaire afin que les cases correspondent bien à leur nom qui est stocké dans un autre tableau sous cette forme :

['a8', 'b8', 'c8', 'd8', 'e8', 'f8', 'g8', 'h8']

['a7', 'b7', 'c7', 'd7', 'e7', 'f7', 'g7', 'h7']

['a6', 'b6', 'c6', 'd6', 'e6', 'f6', 'g6', 'h6']

['a5', 'b5', 'c5', 'd5', 'e5', 'f5', 'g5', 'h5']

['a4', 'b4', 'c4', 'd4', 'e4', 'f4', 'g4', 'h4']

['a3', 'b3', 'c3', 'd3', 'e3', 'f3', 'g3', 'h3']

['a2', 'b2', 'c2', 'd2', 'e2', 'f2', 'g2', 'h2']

['a1', 'b1', 'c1', 'd1', 'e1', 'f1', 'g1', 'h1']

En effet, selon l’orientation de la caméra, les pièces blanches peuvent être à gauche, en bas, à droite ou bien en haut sur la photo, ce qui influe directement sur la matrice d’occupation définie plus haut.

Pour effectuer cette rotation, une variable appelée « angle » est définie ici. Elle n’est définie qu’une seule fois pour toute la partie, au début, lorsque l’on connait la position de chaque pièce et que cela suffit pour connaître l’orientation de l’échiquier. On regarde donc l’occupation des quatre coins de l’échiquier, et en fonction, on définit une variable « angle » et on applique la rotation de la matrice d’occupation nécessaire afin que les tours blanches soient situées en A1 et H1, et les tours noires en A8 et H8.

Après rotation, la matrice d’occupation ressemble donc à ceci :

[-1, -1, -1, -1, -1, -1, -1, -1]

[-1, -1, -1, -1, -1, -1, -1, -1]

[0, 0, 0, 0, 0, 0, 0, 0]

[0, 0, 0, 0, 0, 0, 0, 0]

[0, 0, 0, 0, 0, 0, 0, 0]

[0, 0, 0, 0, 0, 0, 0, 0]

[1, 1, 1, 1, 1, 1, 1, 1]

[1, 1, 1, 1, 1, 1, 1, 1]

Notons que cet « angle » n’est défini qu’une fois par partie. Il faut donc s’assurer que l’échiquier est pris en photo avec une orientation semblable à chaque coup. La caméra ne doit pas nécessairement être immobile, mais si son angle varie trop, la détection du coup joué va échouer. Théoriquement, si l’échiquier était bien orienté par rapport à la photo au début de la partie, la caméra devrait pouvoir tourner autour d’un axe vertical d’un angle compris entre -45° et 45° sans que cela n’ait d’impact sur l’orientation de la matrice d’occupation. En revanche, la définition des couleurs moyennes serait moins robuste au dépassement des pièces situées sur les cases alentour, car le petit carré définit pour calculer la couleur moyenne de la case est orienté selon les pixels de l’image, et non pas selon l’orientation de l’échiquier.

# Détection du coup joué

Le programme doit renvoyer à chaque coup joué une chaîne de caractères qui sera ensuite envoyée à stockfish, l’AI utilisée via un autre programme. Cette chaîne de caractères, pour être utilisée par stockfish doit être de la forme « position startpos moves e2e4 e7e5 … » où « e2e4 e7e5 … » constitue la liste de tous les coups joués depuis le début de la partie. On crée donc une chaîne de caractères type, à laquelle est concaténée à chaque itération la chaîne correspondant au coup joué.

## Coup classique

Pour un coup classique, seules deux cases de l’échiquier voient leur état d’occupation changer : celle dont vient la pièce ayant bougé et celle où elle s’est arrêtée. La case de départ passe donc d’un état occupé (1 ou -1) à un état vide (0). La case d’arrivée passe d’un état quelconque à un état occupé, mais si l’état était occupé, il le reste tout en changeant de couleur (-1 -> 1 ou 1 -> -1).

Le coup à retourner est donc une chaîne de caractères contenant la case de départ suivie de la case d’arrivée, par exemple « e2e4 ».

On peut noter que le code ne prend pas en compte les déplacements possibles de chaque pièce. Ainsi, n’importe quelle pièce peut se déplacer sur n’importe quelle case de l’échiquier à chaque coup. Le risque est de retourner un mouvement non règlementaire, comme « A1G8 » qui n’est permis pour aucune pièce. Cependant, ce risque est mineur dans la mesure où les coups que l’on joue sont censés être réglementaires.

## Roque

Pour le roque, quatre cases changent d’état d’occupation. Deux passent dans un état vide et deux dans un état occupé. Le code vérifie simplement que la position de ces quatre cases est compatible avec un roque, puis il retourne le coup sous forme d’une chaîne de caractères représentant le déplacement du roi.

Ici aussi, certains coups joués non réglementaires pourraient retourner un coup.

## Prise en passant

Pour la prise en passant, deux cases passent à l’état vide et une seule passe dans un état occupé. Le code ne vérifie que la cohérence des positions de ces trois cases. Le coup retourné correspond au déplacement du pion effectuant la prise en passant.