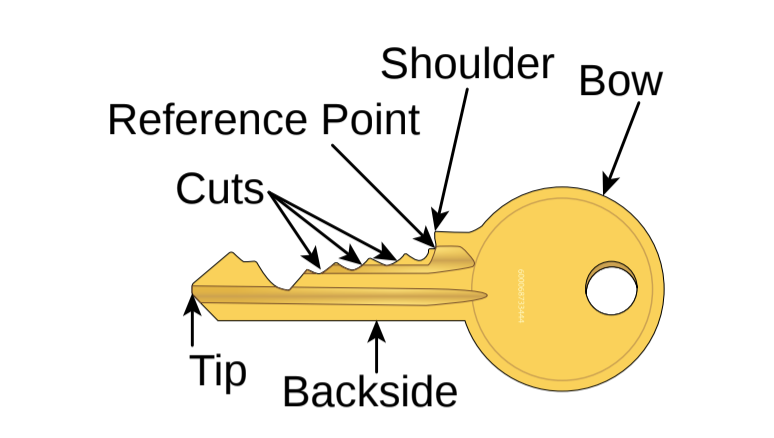
**Rapport de projet “Reconnaissance de clés”**

Période : P4 (Décembre 2019 - Janvier 2020)



Étudiants : LEROY Yanis / MAES Sébastien (Promo FI21)

Encadrant : MENNESSON José

Sommaire

[**Introduction**](#_4hh0rjlge1ch) **2**

[**1. Recherche documentaire et étude du code existant**](#_skybo8xrrv1m) **3**

[**2. Réalisation d’une base de données d’images**](#_5dbc8gmsjhc) **4**

[**3. Recherche de solutions**](#_ta9615bbu5om) **5**

[a. Étude de l’outil GrabCut](#_bij52h7qxnkt) 5

[b. Étude du “Template Matching”](#_n1mdl53y6aar) 5

[c. Étude du “Principal Component Analysis” (PCA)](#_n1mdl53y6aar) 6

[**4. Solutions adoptées**](#_c6jwm3oxejq) **7**

[a. Mise à la verticale de la clé](#_w7j8dy1lwb5x) 7

[b. Segmentation de la clé](#_vmry264qk6) 8

[c. Réorientation et segmentation des crans](#_fw1taby4yvau) 9

[d. Recherche des contours de la clé](#_ohwcyokwiuy3) 10

[e. Recherche de l’épaule de la clé et du premier cran](#_je3grzok4h8l) 11

[f. Détermination automatique de la position des crans](#_qaita9l52djt) 12

[g. Mesure de la profondeur des crans](#_tlidiursvksz) 13

[h. Obtention du code de la clé](#_fegkeqtojblw) 13

[**5. Résultats**](#_pzr8pwro1pc6) **14**

[a. Comparaison des mesures empiriques et logicielles](#_iw75ecewlzp8) 14

[b. Comparaison des mesures sur plusieurs prises de vue](#_po7yejol1coi) 15

[**6. Bilan**](#_kor5j50wn9v) **19**

[**Bibliographie**](#_dkt3qcrygimz) **22**

# Introduction

L’objectif de ce projet est de développer un logiciel en python permettant, **à partir d’une simple photo** prise avec un smartphone, **d’obtenir le taillage d’une clé** c’est-à-dire acquérir les mesures des différentes encoches qui composent la clé dans le but de pouvoir **obtenir le code correspondant** et permettre la fabrication de répliques. Ainsi, il sera possible de copier une clé plus rapidement et facilement mais aussi de façon moins onéreuse qu’avec du matériel professionnel dédié.

Notre objectif est d’apporter un programme qui fonctionne avec **plusieurs types de clé**, et des conditions différentes, dans la majorité des cas. Même si les résultats ne sont pas toujours optimaux, le **but est d’explorer, tester et mesurer l’efficacité de diverses méthodes** possibles sans recourir à des méthodes plus avancées et complexes telles que l’apprentissage automatique.

# 1. Recherche documentaire et étude du code existant

Dans un premier temps, nous avons effectué diverses recherches sur les **grands types et normes de clés** afin de mieux comprendre comment le taillage d’une clé est effectué et comment le code correspondant est établi. Nous avons alors étudié différents sites web (voir bibliographie) mais aussi différents catalogues constructeurs listant les types de clés les plus courants.

*Clé à billes Clé plate*

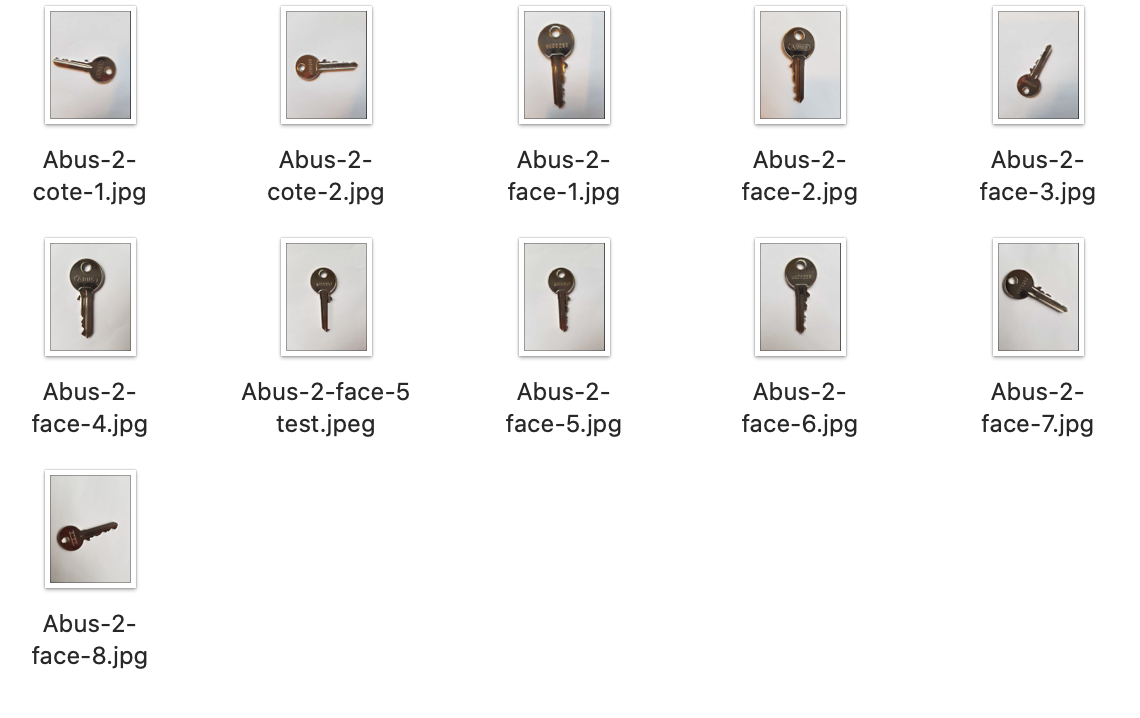
Cependant, dans notre cas, les types de clés nous intéressant sont les **clés “plates”** et non pas les clés à billes par exemple. En effet, la profondeur des “perforations” de ces dernières ne sont pas mesurables avec une simple photo.

Dans un second temps, nous avons travaillé sur la **librairie OpenCV** dédiée au traitement d'images et qui nous sera très utile dans la réalisation du projet. Nous avons alors analysé les fonctions pouvant nous servir dans le traitement de la photo, la recherche de la clé dans celle-ci et ensuite dans la mesure des crans.

Après ceci, nous avons étudié le **rapport et le code réalisés** par l’étudiant auparavant afin de mieux le comprendre et donc de trouver ses points forts et faibles. Selon nous, les points à améliorer étaient donc les suivants : pouvoir effectuer des **mesures sur davantage de types de clés** et dans **davantage de conditions** (dents vers le haut, dents sur la gauche ou la droite, clé sur le côté, etc...), mais aussi **améliorer la précision** des mesures puisque celles-ci étaient parfois très éloignées de la réalité.

# 2. Réalisation d’une base de données d’images

Dans le but de pouvoir de tester le code réalisé par l’étudiant auparavant et ensuite de tester le nôtre, nous avons réalisé en début de projet une **base de données des différentes clés fournies**. Pour chaque type de clé, celle-ci contient donc une multitude de photos avec différentes prises de vue (clé de face, sur le côté, avec un peu d’ombres, avec quelques reflets...etc).



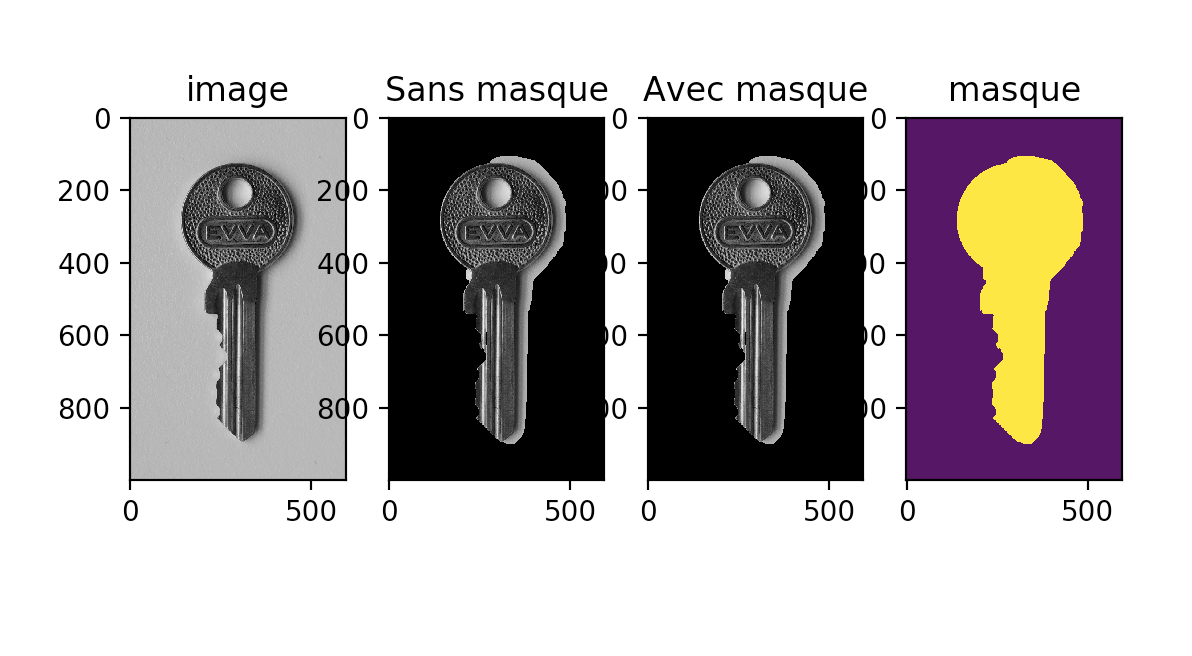
# 3. Recherche de solutions

## a. Étude de l’outil GrabCut

Étant donné que l’algorithme Canny utilisé pour faire de la détection de bords et de contours peut être perturbé par des éléments en arrière-plan. Nous avons donc essayé de récupérer uniquement le premier plan c’est-à-dire la clé dans son intégralité, avec ses crans, grâce à un outil qui s’appelle ***GrabCut***.

Tout d’abord, il convient de définir un rectangle autour de ce que l’on souhaite extraire. Cet algorithme distingue 4 types de zone : une première zone qui est certaine d’être le premier plan, une autre certaine d’être l’arrière-plan (en dehors du rectangle), mais aussi deux autres zones qui sont peut-être l’un des deux (à l’intérieur du rectangle).

Malheureusement, **nous avons rencontré certains problèmes** avec cette méthode. La forme de la clé empêche elle-même une extraction optimale à cause notamment de pertes partielles de détails au niveau des crans, voire même totales, rendant ainsi impossible une mesure précise de leur position et de leur profondeur. De plus, la photo et le modèle, pouvant servir de masque pour améliorer la qualité de l’extraction, sont trop éloignés pour que cela fonctionne correctement.



## b. Étude du “Template Matching”

Étant donné que l’extraction de la clé via l’outil GrabCut n’était pas concluant, nous avons adopté une autre approche qui est de **rogner l’image au plus près de la clé.**

Nous avons alors testé l’outil **“Template Matching”** d’OpenCV. Ce dernier consiste en effet à prendre une photo et une image modèle et ensuite de **rechercher des zones correspondantes dans l’image.** L’algorithme compare **pixel par pixel** les deux images afin de trouver ceux en communs.

Nous n’avons pas gardé cette solution car elle présentait un **nombre trop importants d’inconvénients.** D’abord, dans notre cas, le modèle utilisé est parfois très différent de la clé elle-même. La comparaison par pixel fonctionnerait donc difficilement. De plus, le modèle et l’image doivent être parfaitement alignés. Or même si une fonction analyse l’angle de rotation de la clé, il est difficile d’aligner parfaitement le modèle et la photo.

Nous avons donc choisi un autre outil, qui sera présenté dans la partie des solutions adoptées.

## c. Étude du “Principal Component Analysis” (PCA)

Nous avions tenté d’améliorer la rotation de la clé en mettant à l’épreuve une autre méthode qui s’appelle l’**Analyse en Composantes Principales** (ACP) ou **Principal Component Analysis** (PCA) en anglais.

Cette méthode consiste à détecter les contours de potentiels objets ou caractéristiques dans l’image puis déterminer le centre du contour à partir duquel on établi un repère. Pour trouver l’orientation de l’objet, on calcule l’angle entre l’un des axes du repère avec l’axe des abscisses de l’image.

L’idée semblait séduisante mais **cette méthode manque cruellement de précision**. De plus, elle détecte facilement d’autres contours d’objets dans l’image en plus de celle de la clé ce qui signifie que l’on récupérait une liste d’angle sans savoir à quel objet chacun correspondait. Il est en effet facilement trompé par l’arrière-plan ou les détails sur la clé. En plus du manque de précision, il est également **fort probable que l’angle de rotation ne correspondait pas forcément à celui de l’objet voulu, la clé**.

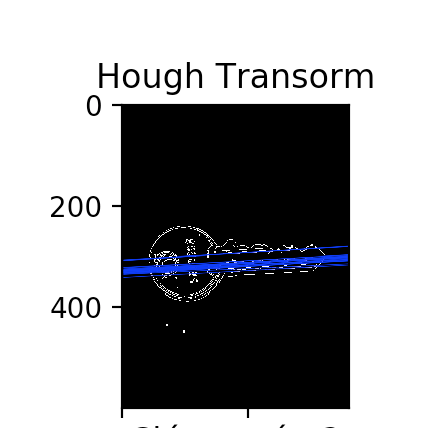
Cet algorithme est aussi inutilement compliqué comparé à la transformée de Hough et n’apporte rien de plus. Nous avons donc opté pour complémenter cette dernière en gérant notamment le basculement de la clé que nous détaillerons dans une partie ultérieure.

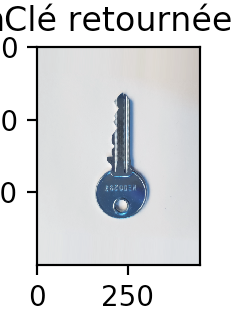
# 4. Solutions adoptées

## a. Mise à la verticale de la clé

Pour pouvoir effectuer une **mesure de profondeur** des crans d’une clé, il est important que celle-ci soit **bien droite** sur la photo. Si celle-ci est légèrement orientée sur le côté, les mesures seront faussées. Dans notre code, la première étape est donc la **mesure de l’orientation de la clé, puis sa rotation si besoin.**

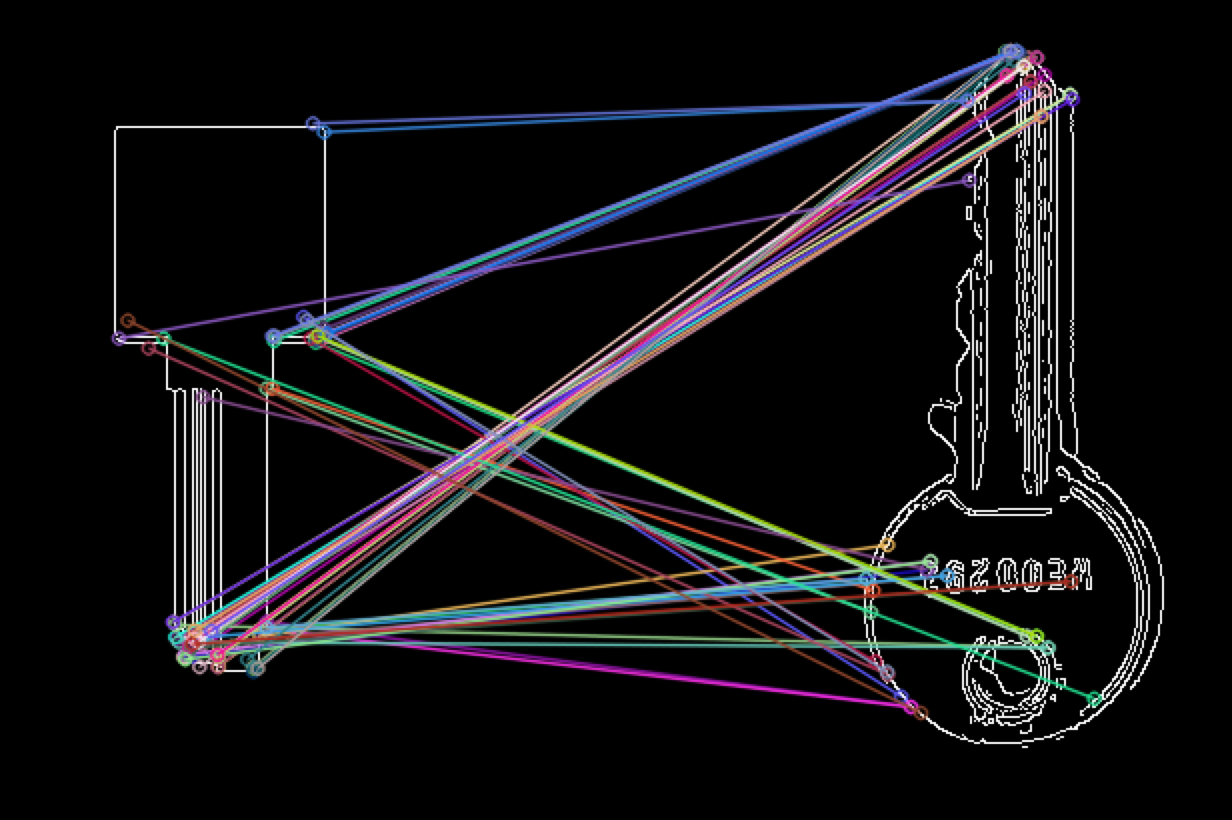
Pour cela, nous utilisons la technique de la **transformée de Hough** qui consiste à extraire des lignes à partir des contours d’une image (via l’outil Canny Edge Detector d’OpenCV). Nous mesurons ensuite l’angle de chaque ligne et enfin nous calculons la médiane de l’ensemble de ces angles pour avoir une estimation de l’orientation de la clé assez fidèle de la réalité.

Cette technique **fonctionne bien** dans notre car les **clés présentent suffisamment de lignes** sur leur face pouvant être facilement retrouvées par la transformée de Hough. 

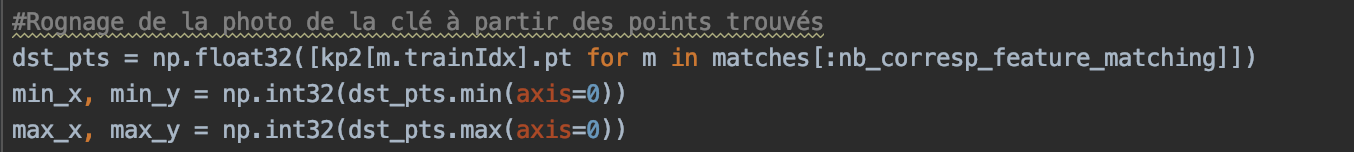


## b. Segmentation de la clé

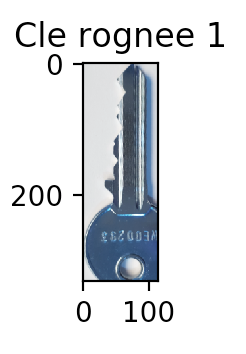
À partir de l’image de la clé “réorientée”, nous souhaitons désormais **rogner l’image** pour ne garder que la clé et **supprimer au mieux ce qui l’entoure**. Pour cela, nous utilisons une méthode de **“Feature Matching”** d’OpenCV. Cette méthode permet de rechercher de points clés dans les contours (via l’outil Canny Edge Detector d’OpenCV) de deux images (ici la photo et un modèle de clé) puis de mettre en correspondance ces points.



Dans notre cas, on remarque que la **mise en correspondance des points** est **loin d’être parfaite** puisque la clé et son modèle sont trop différents. L’algorithme met donc généralement en correspondance les “coins” de la clé.

Cependant, ce résultat ne représente pas un problème pour le résultat final souhaité. En effet, puisque les correspondances se font sur les “coins” de la clé, en **recherchant les minimums et maximums des coordonnées X et Y** des points présents dans la liste des correspondances, nous obtenons les coordonnées du haut, du bas, et des côtés gauche et droite de la clé. 

À partir de ces coordonnées, nous pouvons donc rogner l’image au plus près de la clé. Pour avoir le meilleur “cadrage” possible, nous ne gardons cependant qu’un certain nombre de correspondances. Nous avons estimé que garder uniquement les **70 meilleures correspondances était suffisant.**



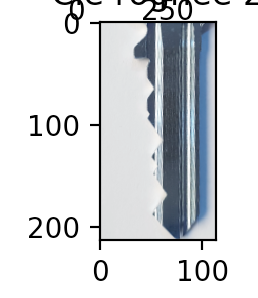
## c. Réorientation et segmentation des crans

La prochaine étape du code est donc de **définir si la clé est orientée vers le haut ou le bas**, ce qui n’a pas été possible avec la technique de la transformée de Hough. Étant donné qu’une majorité des clés plates présente une **partie supérieure circulaire**, notre approche était donc de **rechercher un cercle dans l’image** grâce à l’outil “Hough Circles” d’OpenCV.



Dans un premier temps, nous étudions les **coordonnées du centre du cercle** obtenu. Si la position en hauteur (coordonnée Y) est inférieure à la moitié de la hauteur de l’image, cela indique que la clé est dans le mauvais sens, c’est-à-dire les crans vers le haut. Dans ce cas, il nous suffit donc de retourner l’image.

Dans un second temps, nous **finalisons le rognage de l’image** pour ne **garder que les dents** sur l’image. Pour cela, nous “supprimons” la partie partant du haut de l’image jusqu’en bas du cercle.



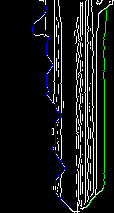
Cette approche que nous avons adoptée présente cependant quelques points faibles. En effet, même si elle fonctionne dans la majorité des cas, l’algorithme peut être facilement trompé si des formes arrondies sont présentes sur la clé, comme par exemple un logo de la marque de la clé. Dans certains cas, **l’algorithme peut ne pas trouver de cercle dans l’image ou en trouver un ne correspondant pas à ce que nous attendons**.

## d. Recherche des contours de la clé

L’étape suivante est donc de **rechercher les contours de la clé** pour ensuite pouvoir rechercher la position des crans et mesurer leur profondeur. Pour cela, nous utilisons d’abord la détection de bords (*“edge detection”*) via l’algorithme **Canny**, implémenté dans OpenCV.

Cet outil a déjà été utilisé auparavant dans la transformée de Hough et le “Feature Matching” mais nous ne reprenons pas les contours obtenus dans ces étapes car l’image a été modifiée plusieurs fois (retournée, rognée) et en réitérant l’opération avec l’image finale, c’est-à-dire seulement les dents, nous estimons que nous pouvons avoir de meilleurs résultats qu’avec une clé “entière”.

Pour améliorer la détection de contours, nous avons également réalisé une **fonction calculant automatiquement les seuils de détection** nécessaires pour l’utilisation du “Canny Edge Detector”. Ainsi, il n’est pas nécessaire de définir manuellement ces paramètres et ceux-ci sont définis de façon plus précise.



Ensuite, nous nous sommes inspirés de la fonction “Polygon Detection” réalisée l’étudiant auparavant. Nous l’avons modifié pour que celle-ci enregistre dans **deux tableaux** distincts les coordonnées (X, Y) des **contours à droite et à gauche de la clé.**

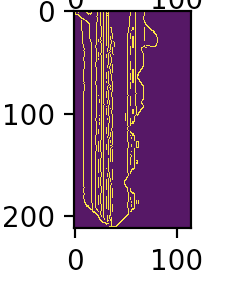
Nous calculons alors la **variance de chacun des tableaux** et nous les comparons. Étant donné que le côté de la clé sans les crans est principalement “lisse”, le tableau présentant la **variance la plus élevée** sera celui avec les **crans**. À partir de là, nous pouvons donc retourner ou non l’image afin de toujours avoir les dents sur le côté droit de l’image. L’ensemble des étapes suivantes utilisent alors le tableau des contours droit de l’image.

Cependant, cette technique présente quelques limites, notamment si les contours sont mal définis ou que le rognage précédent a été mal réalisé. Dans certains cas, la comparaison des variances peut donc ne pas fonctionner.

## e. Recherche de l’épaule de la clé et du premier cran

L’épaule (ou *“****shoulder****”*) de la clé est la **partie du contour le plus à droite de l’image.** Il nous suffit donc de chercher dans le tableau des “contours droit” le point ayant la coordonnée X la plus élevée. La position en largeur (coordonnée X) de l’épaule permet de définir une référence pour mesurer la profondeur de chaque cran.

Pour trouver le premier cran, nous prenons une partie du tableau des contours droit, partant de la position en hauteur (coordonnée Y) de l’épaule et allant jusqu’au 50ème pixel après celle-ci. Ainsi, le pixel du **contour le plus à gauche donnera la position du premier cran.** L’espace entre les crans étant constant, en connaissant la position du centre du premier, on déduit aisément la position des autres.

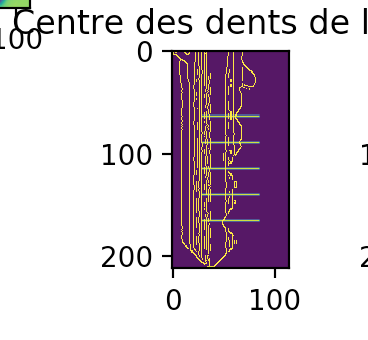


## f. Détermination automatique de la position des crans

Pour pouvoir mesurer la profondeur de chacun des crans, nous devons au préalable **trouver leur coordonnée en “hauteur”** (coordonnée Y). Pour cela, nous avons étudié **deux solutions.**

La première consiste à exploiter le fait que **l’écart entre chaque dent est constant**. En partant de la première, il est donc possible de trouver une position plutôt précise de la position des autres dents. Les deux principaux inconvénients sont suivants :

* Il est **nécessaire de connaître et de définir manuellement l’écart** entre les dents.
* La **position de la première dent doit être suffisamment précise**. Dans le cas contraire, les positions de toutes les autres dents seront erronées.



Pour remédier à ces inconvénients, nous avons cherché à réaliser une **fonction cherchant automatiquement** les positions des dents. Celle-ci fonctionne de la manière suivante :

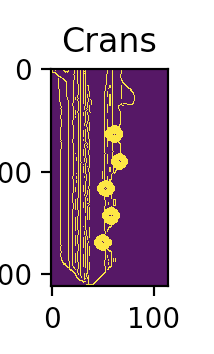
1. **Calcul des dérivées** “en X” des contours droits obtenus
2. Recherche des “coordonnées Y” où il y a plus de 2 points stagnants **(dérivée nulle)**
3. **Séparation des points trouvés** : création d’ensembles contenant les coordonnées Y des points séparés de 1 px (correspond à une dent)
4. **Calcul de la médiane** de ces ensembles. Celle-ci correspondra à la coordonnée Y approximative de chaque cran.

Cependant, **cette solution n’a pas pu être exploitée.** En effet, **l’image étant de moyenne qualité** (ombres, reflets, défauts dûs à la lentille du smartphone…), les contours ne sont pas parfaits. Les calculs de dérivées sont donc faussés par ces contours imparfaits et les positions des dents se trouvent mal définies.

Nous avons donc finalement conservé la première solution même s’il est toujours possible, via un paramètre présent dans le “main” du code, d’utiliser la seconde, dans le cas où une image de bien meilleure qualité serait disponible.

## g. Mesure de la profondeur des crans

Une fois les coordonnées Y des dents obtenues, il est facile à partir du tableau des “contours droit”, d’obtenir les coordonnées X correspondantes : il suffit d’aller **chercher dans le tableau, le couple (X,Y) avec Y connu.** Pour connaître la profondeur, en pixel, de chaque cran, il nous reste à faire la **différence entre la coordonnée X de l’épaule et celle du cran.**

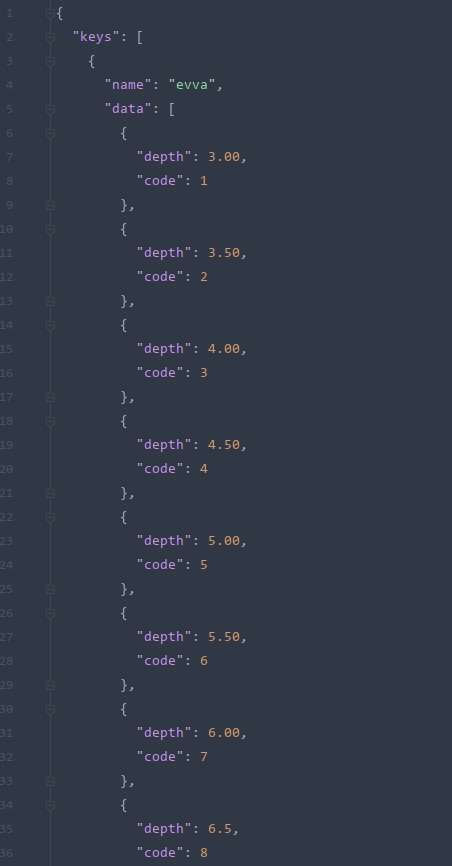


Pour faire la **conversion pixels-millimètres** de la profondeur, il nous faut ensuite un “repère” sur l’image où nous connaissons déjà la distance en millimètre. Pour cela, nous calculons donc la **distance en pixels entre l’épaule et le bas de l’image**. Nous avons établi qu’en moyenne, cette distance est de **27mm**. Ensuite, avec une simple règle de trois, nous pouvons savoir à combien de millimètres correspond un pixel et donc faire la conversion.

## h. Obtention du code de la clé

Les codes associés à la profondeur des crans, pour chaque clé connue, sont stockés dans un **fichier JSON**. Cette approche comporte **plusieurs avantages** à savoir **la facilitation de l’ajout de clés**, la **possibilité d’en ajouter autant que l’on souhaite** et c’est un **format standard très utilisé**. Ce fichier contient une liste de clés. Chacune d’entre-elles possède deux attributs : *“name”*, et *“data”* qui inclut la liste des codes et des profondeurs fournies par le fabricant.

Tout d’abord, on sélectionne la clé correspondante en spécifiant son nom. Ensuite, pour chaque profondeur mesurée d’un cran, on soustrait cette valeur à la largeur de la clé pour avoir **la distance entre le dos de la clé jusqu’à la partie inférieure du cran**, puis on vérifie si elle appartient à un intervalle centré sur la valeur du fabricant. L’écart de 0.25mm est suffisant pour **prendre en compte les potentielles erreurs de mesure** tout en **évitant des erreurs dans la détermination du code** lié au cas où un intervalle chevaucherait (*“overlap”*) un autre. *(valeur\_fabricant - 0.25 <= valeur\_mesuree < valeur\_fabricant + 0.25)*

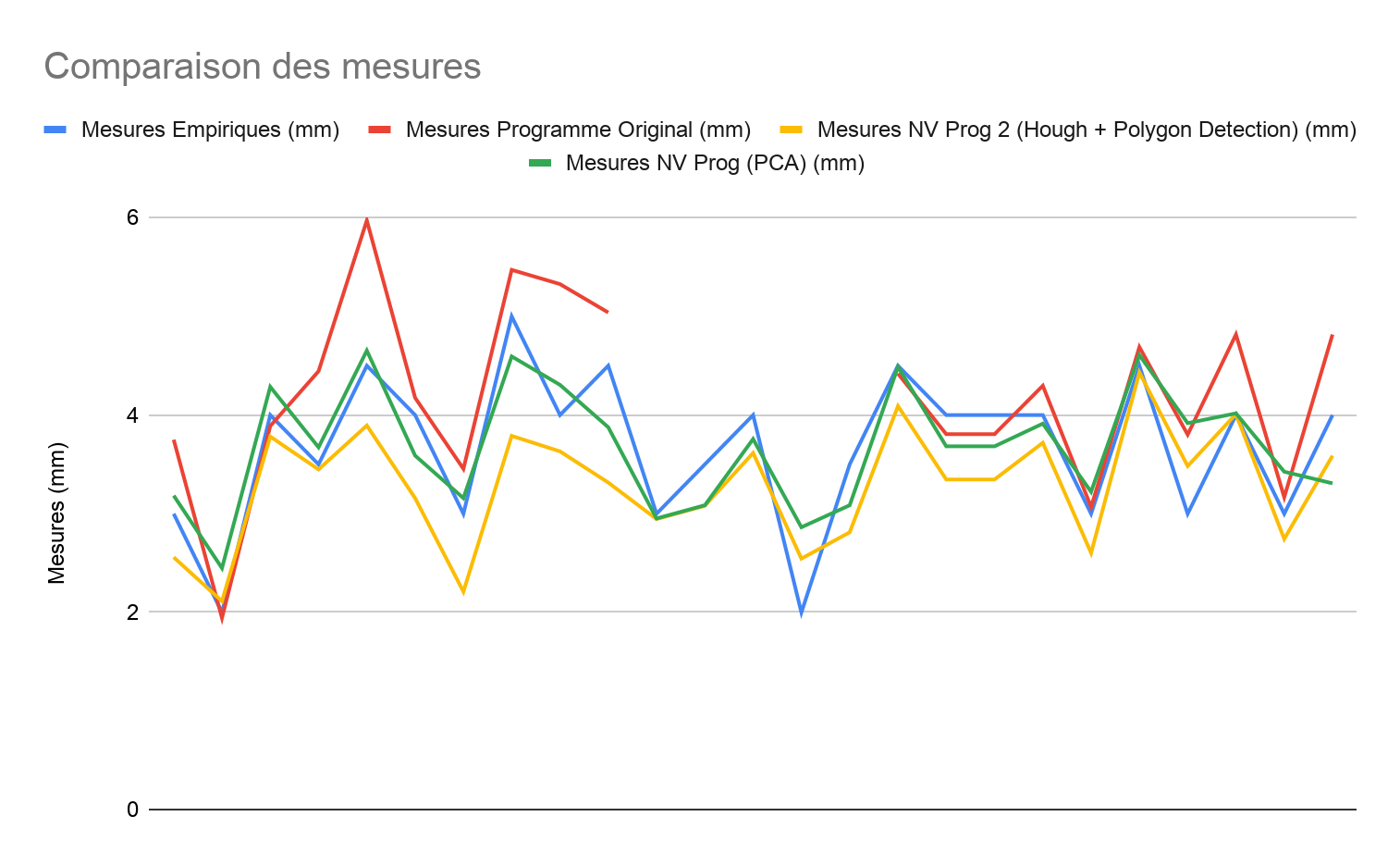


# 5. Résultats

## a. Comparaison des mesures empiriques et logicielles

Pour savoir si les solutions adoptées ont apporté ou non une amélioration dans les mesures, nous avons, dans un premier temps, comparé les **mesures avec toujours la même image** de chaque clé. Ci-dessous ont été comparés :

* les mesures empiriques (profondeurs mesurées avec une règle)
* les mesures obtenues avec le programme réalisé par l’étudiant précédent
* les mesures obtenues avec l’outil “PCA” pour la rotation
* les mesures obtenues avec les solutions adoptées citées ci-dessus

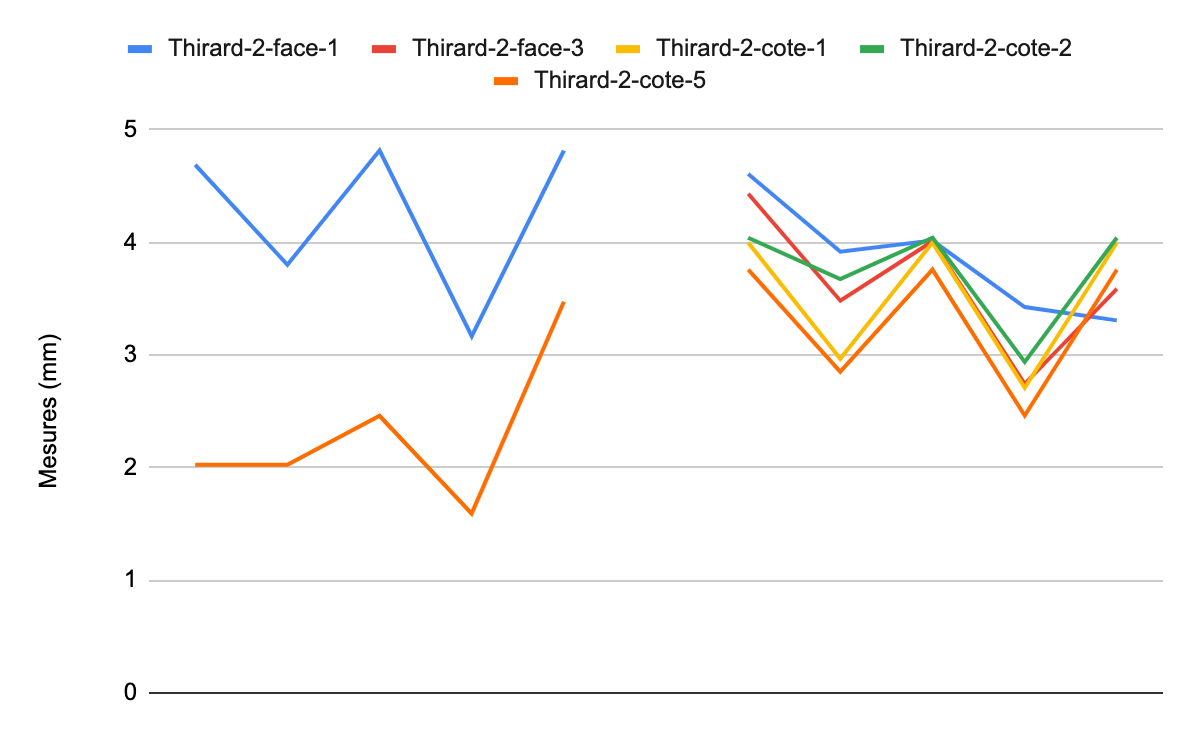


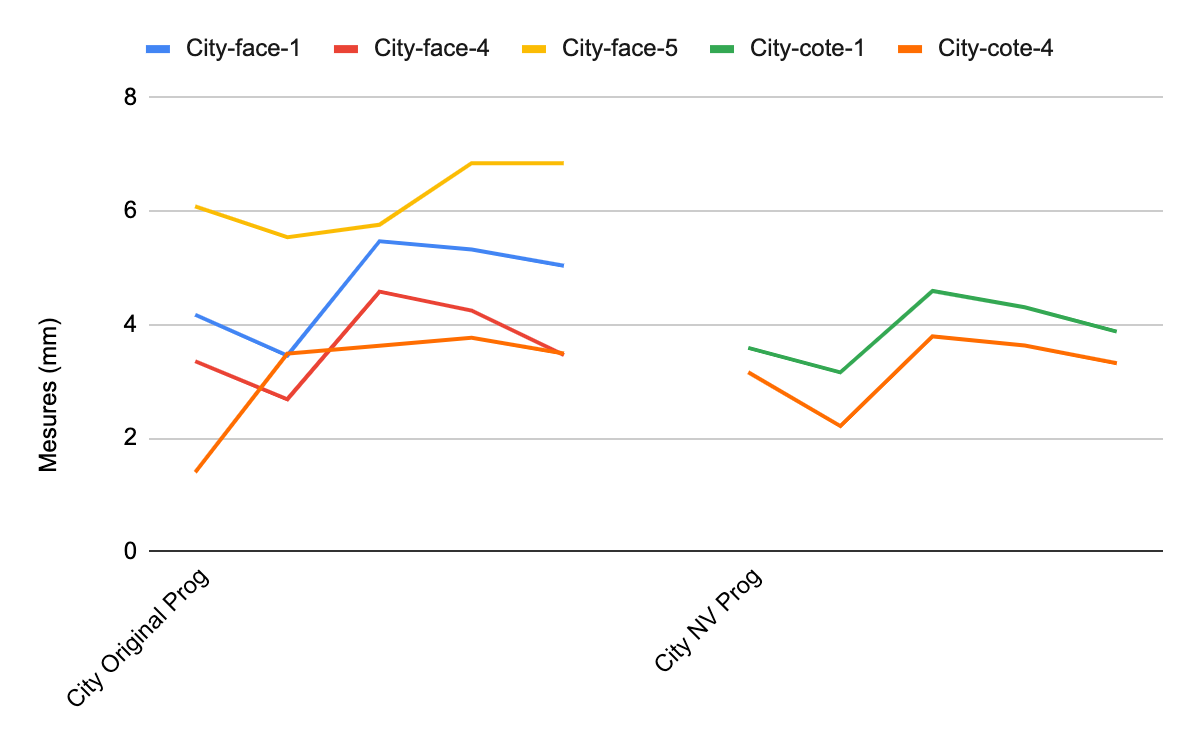
D’abord, on remarque que les mesures obtenues avec le programme réalisé par l’étudiant précédent sont parfois très éloignées de la réalité. Le programme donne parfois même aucun résultat.

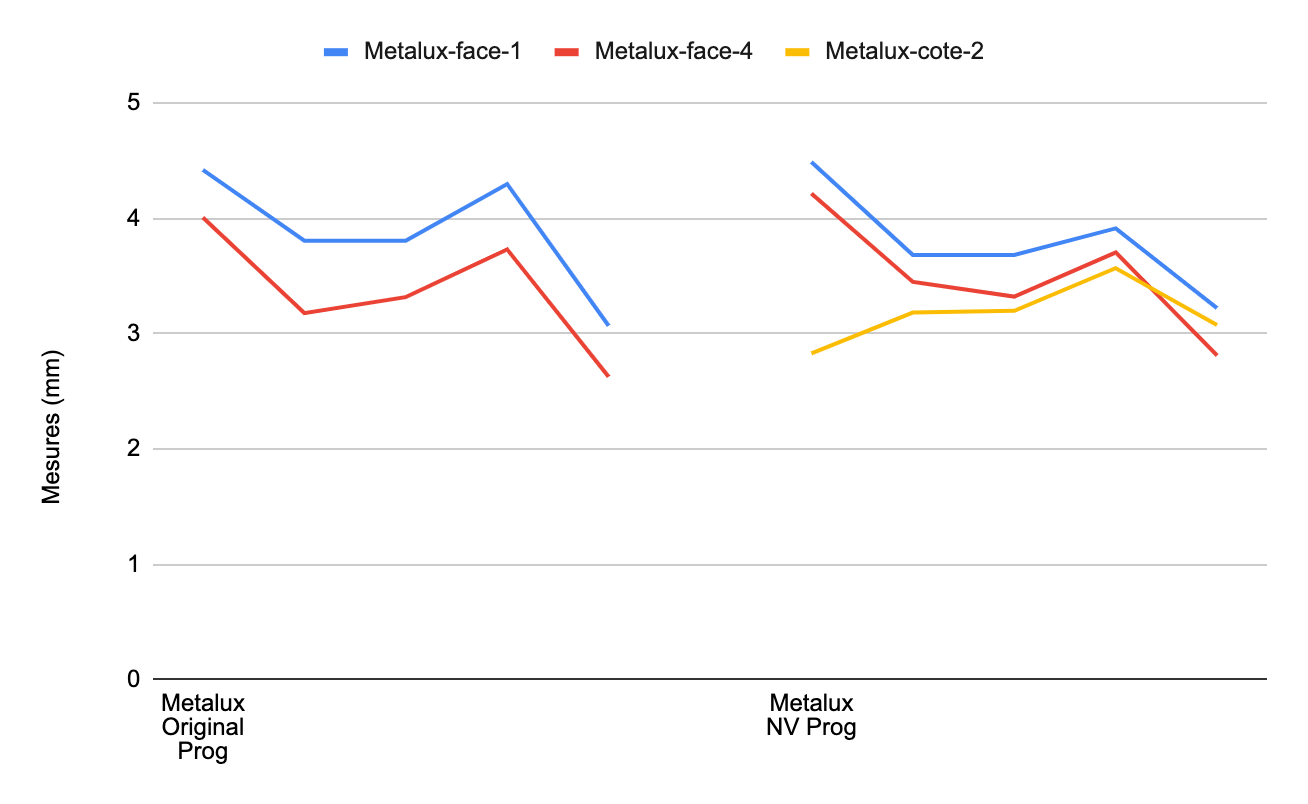
Les **mesures obtenues avec nos deux programmes se rapprochent davantage des mesures terrain**. On remarque parfois que le programme utilisant l’outil PCA semble donner de meilleurs résultats. Cependant, il est difficile d’en tirer des conclusions puisque l’écart entre nos deux programmes n’est pas très grand et que les mesures empiriques sont assez peu précises.

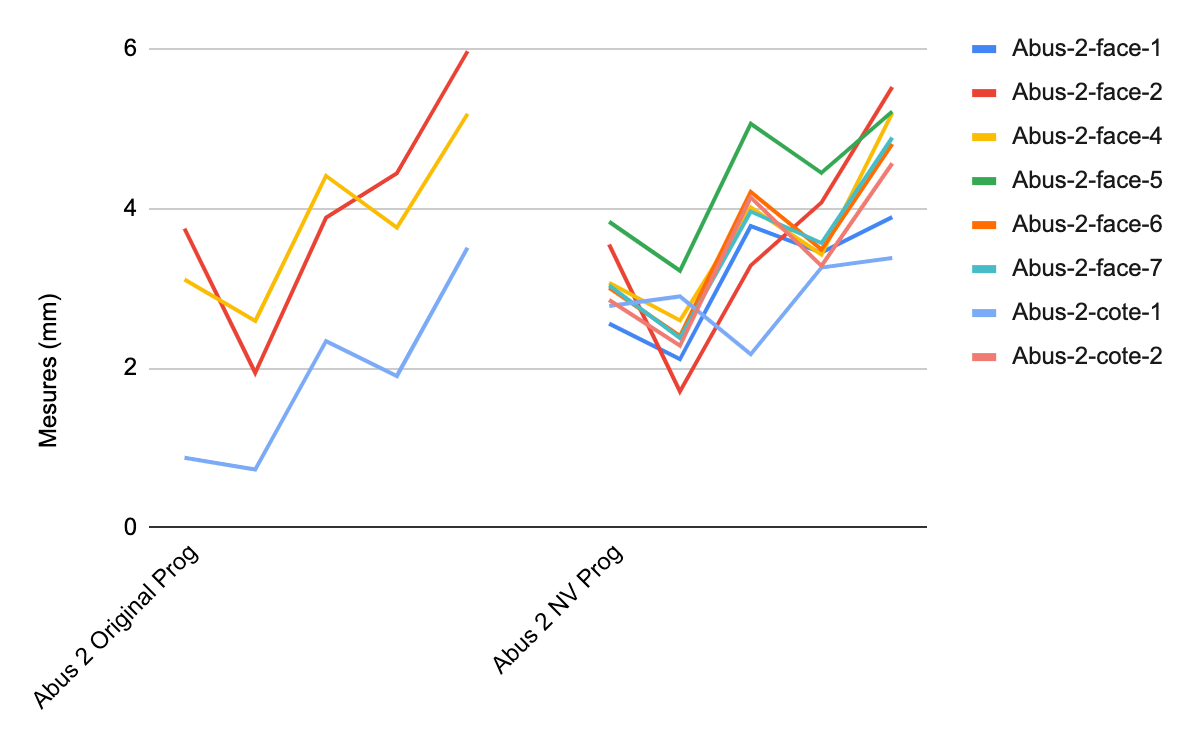
Les différents tests menés ont bien montré que l’algorithme de la transformée de Hough donne de meilleurs résultats que le PCA. Les mesures suivantes ont donc été réalisées avec le dernier programme (utilise la Transformée de Hough et la fonction de détection de contours gauche/droite).

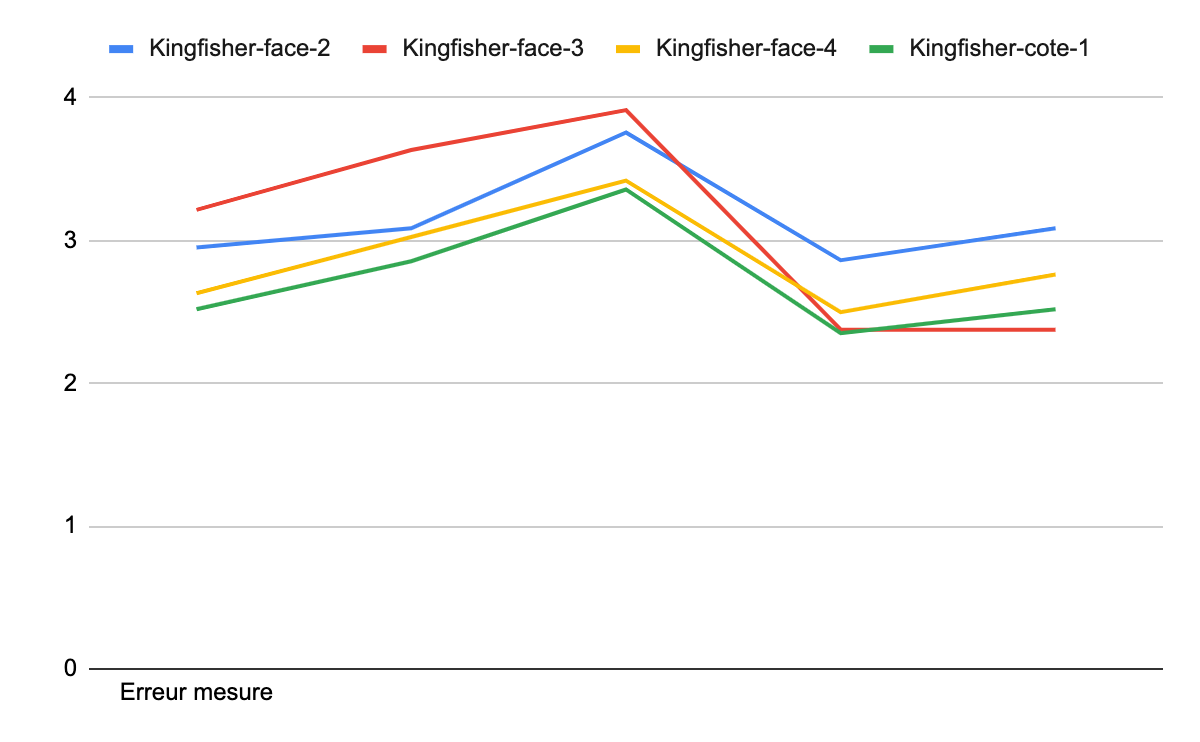
## b. Comparaison des mesures sur plusieurs prises de vue







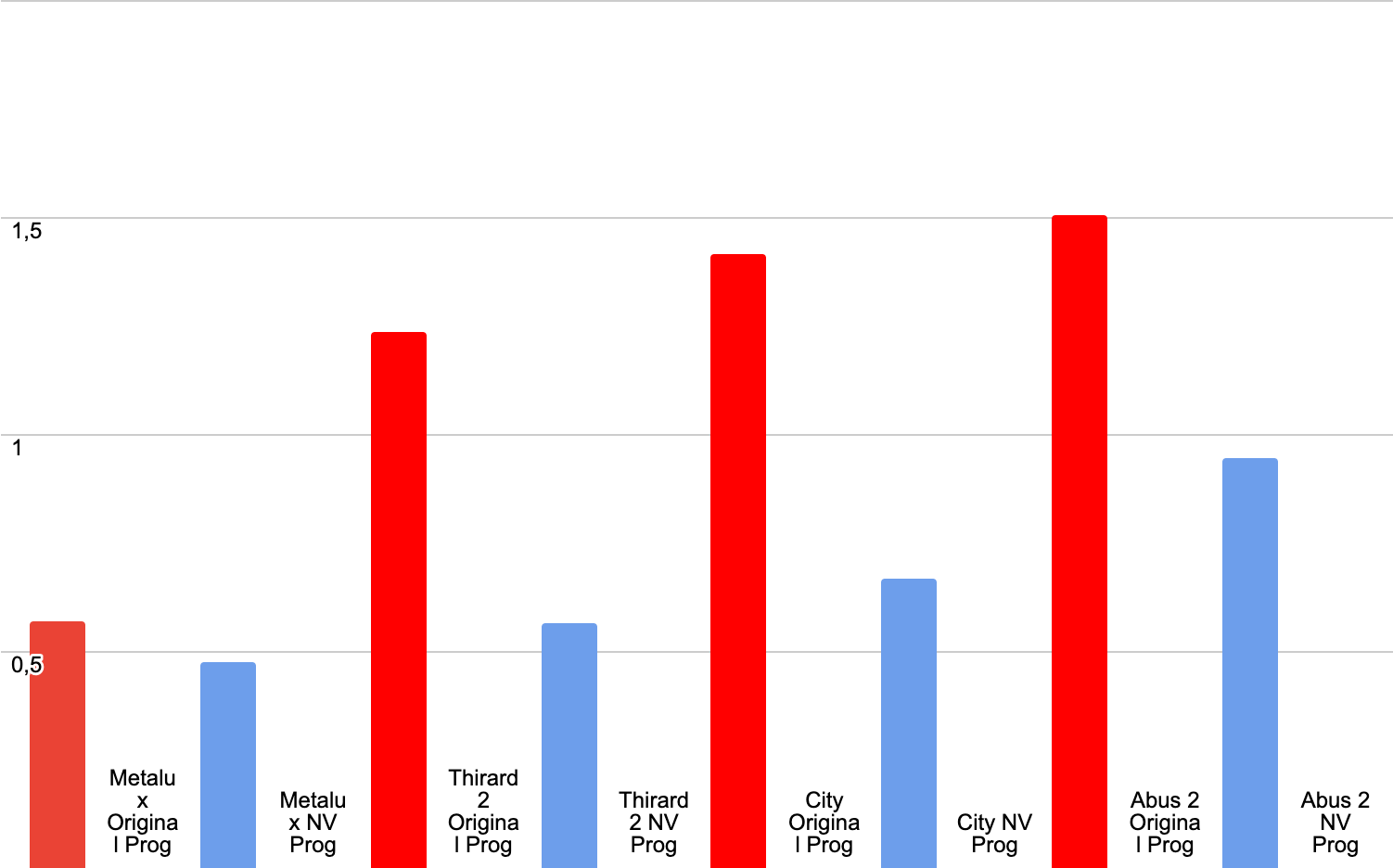




Pour **comparer la fiabilité de notre programme avec le précédent**, nous avons effectué pour chaque clé fournie des mesures sur différentes photos, sur lesquelles la **clé est orientée de différentes façons.**

Sur les graphiques sont tracées ces mesures en millimètres. **Chaque couleur correspond à une photo différente.** Sur la gauche, on retrouve les mesures de l’ancien programme et sur la droite celles de notre programme. Pour la clé de la marque Kingfisher, le programme réalisé en P3 n’a donné aucun résultat. **Plus les tracés sont proches les uns des autres, plus le programme est fiable** puisque cela signifie qu’il renvoie des résultats similaires quel que soit l’image.

Sur l’ensemble des graphiques, on remarque donc qu’avec les solutions que nous avons adoptées, les mesures sont beaucoup plus proches, ce qui montre que le programme est plus fiable dans plus de conditions de prise de vue.

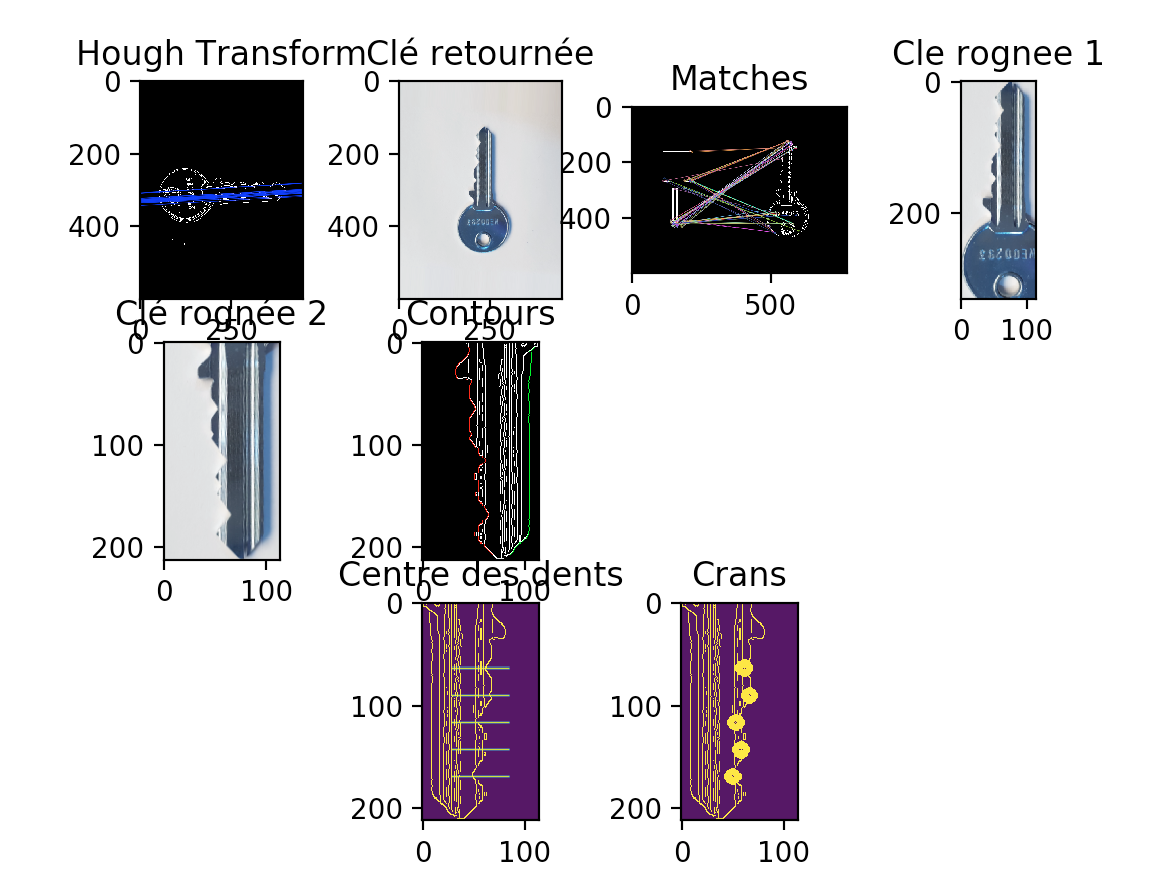


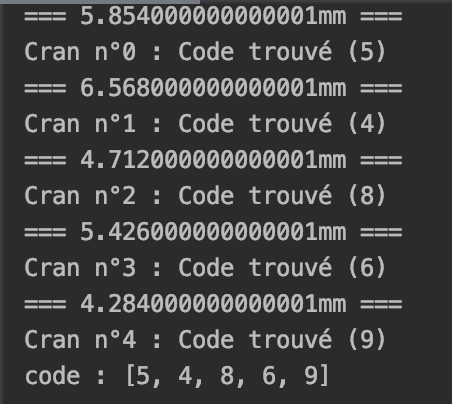
En conséquence, **l’écart type des mesures** sur les différentes photos pour chaque clé est beaucoup plus faible avec notre programme.

# 6. Bilan



*Résultats du programme*





En conclusion, notre programme constitue une **avancée significative mais imparfaite** vers un programme universel de reconnaissance du taillage de clés plates. La fiabilité et la manque de précision des mesures peuvent être liés à divers éléments comme :

* Une image de moyenne qualité (reflets, ombres, image trouble…)
* Une clé n’ayant pas les particularités utilisées dans les différentes fonctions (présence d’un cercle en haut, le côté n’ayant pas les crans doit être plat, …)

Avec une **image de meilleure qualité**, nous pourrions en effet **améliorer la fonction de recherche automatique des dents**, le rognage de la clé mais aussi et surtout la détection de contours.

D’autres solutions pourraient être testées dans les prochaines recherches comme un algorithme basé sur du deep learning pour reconnaître le type de clé pris en photo. En connaissant son type et donc ses particularités physiques, les mesures pourraient en effet être facilitées.

# Bibliographie

* Rapport de projet (et code) réalisé par Robin BOCQUILLON
* OpenCV : <https://docs.opencv.org/4.0.1/d2/d96/tutorial_py_table_of_contents_imgproc.html>
* Feature Matching : <https://docs.opencv.org/trunk/dc/dc3/tutorial_py_matcher.html>
* Traitement d’image de bases : <http://miv.u-strasbg.fr/mazet/ofti/bases.pdf> / [https://xmcvs.free.fr/astroart/Chapitre4.pdf](http://xmcvs.free.fr/astroart/Chapitre4.pdf)
* Mobile Physical Key Recognition, Chongxuan Tang & Eric Lam, Stanford University, Electrical Engineering : [https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.377.7389&rep=rep1&type=pdf](http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.377.7389&rep=rep1&type=pdf)
* Key Code Recognition : Case Study of Automatically Deriving the Code of a Physical Key from Mobile Device Camera Images for the EVVA A key profile : <https://www.researchgate.net/publication/325783115_Key_Code_Recognition_Case_Study_of_Automatically_Deriving_the_Code_of_a_Physical_Key_from_Mobile_Device_Camera_Images_for_the_EVVA_A_key_profile>
* Reconsidering Physical Key Secrecy : Teleduplication via Optical Decoding, Benjamin Laxton & Kai Wang & Stefan Savage : <https://cseweb.ucsd.edu/~savage/papers/CCS08OptDecode.pdf>
* Calcul des seuils dans une image :

<https://stackoverflow.com/questions/4292249/automatic-calculation-of-low-and-high-thresholds-for-the-canny-operation-in-open>

Documents sur les types de clés et cylindres :

* <https://www.cylindre-en-ligne.fr/dictionnaire-serrurerie>
* <http://protectvol.online.fr/types.html>
* <https://www.france-cadenas.fr/19-cles-supplementaires>
* <https://conseil.manomano.fr/comment-choisir-son-cylindre-de-serrure-296>
* <https://www.vachette.fr/fr/site/vachette/produits/cylindre-et-cadenas/cle-et-ebauche-pour-cylindre/cle-et-ebauche-pour-cylindre/>
* <http://www.iseo-france.eu/iseo_france/cataloghi/cylindres-2018.pdf>
* <https://www.thirard.fr/catalogue_THIRARD_2019_SS_PRIX.pdf>