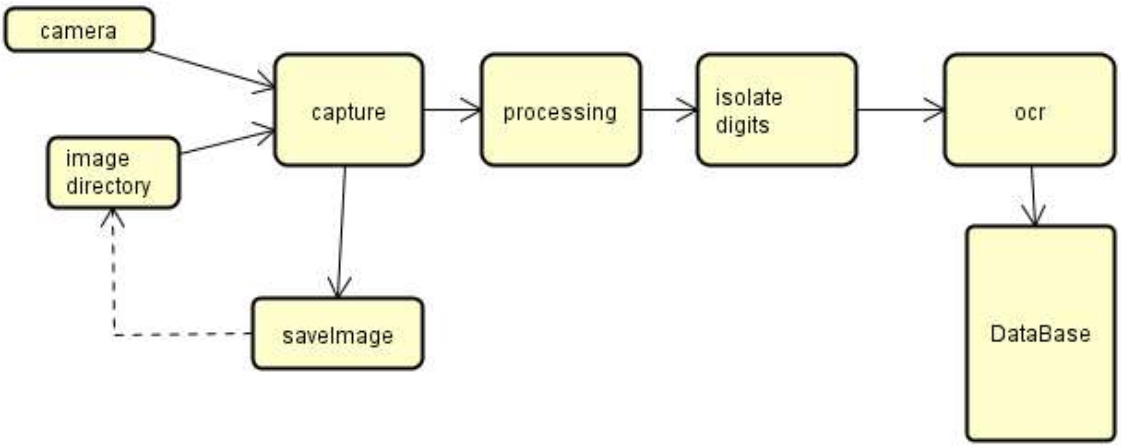
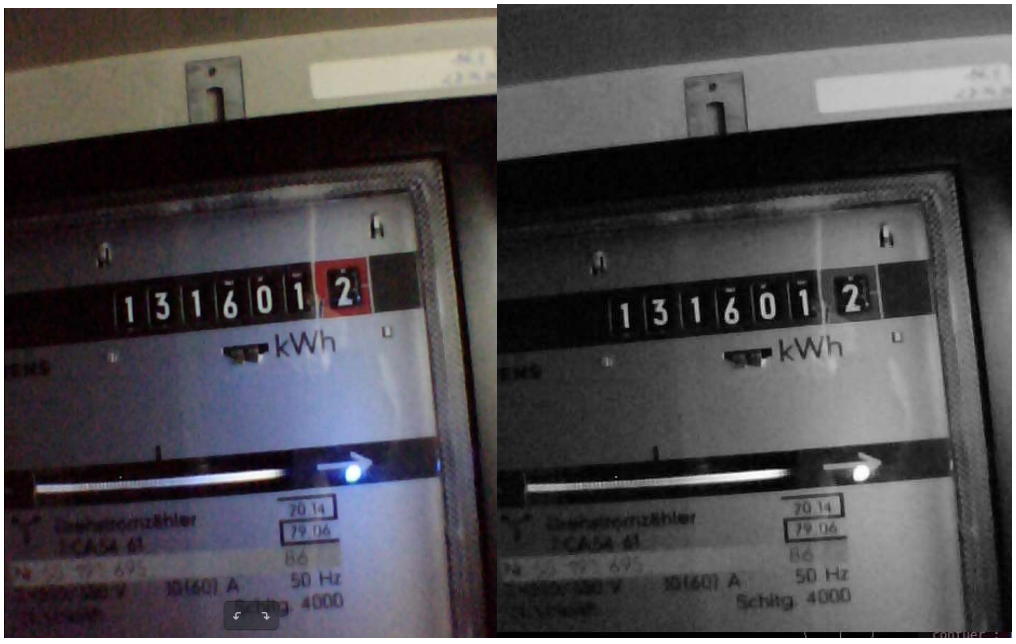
FONCTIONNEMENT DU PROGRAMME OCR



Flux de Programme

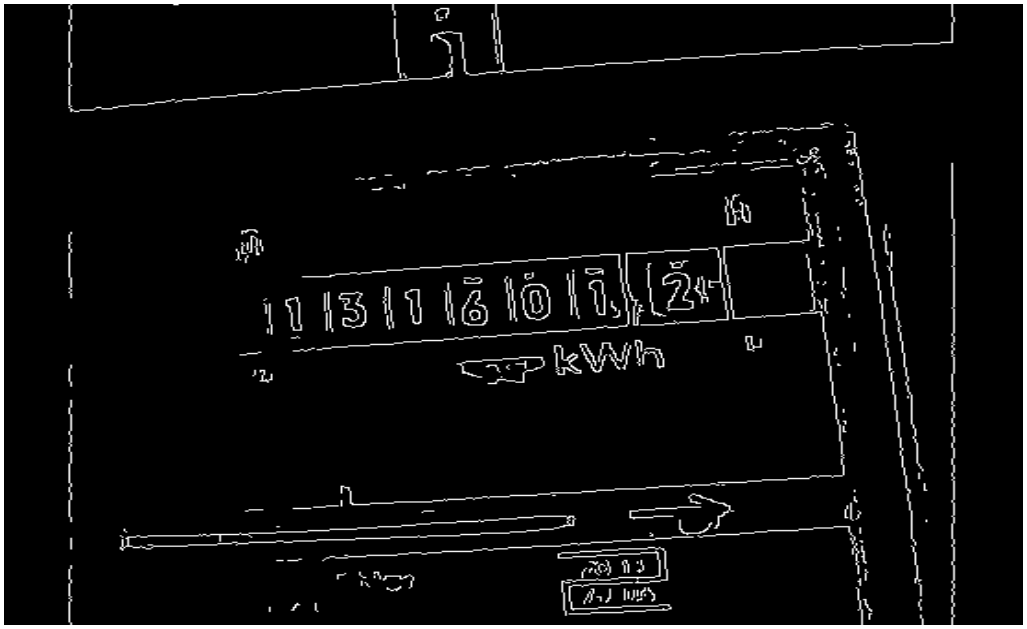
1. La figure montre le flux principal de cette partie. Au début l'image de conteur électrique doit être recordé avec la résolution précise de notre picamera. Et après le program peut le sauvegarder dans un dossier pour qu'on puisse le réutiliser. Ça va nous aider aussi pour le but de développer et de tester.
2. Dans l'étape de processing, on change et optimiser l'image pour préparer l'étape suivante, reconnaissance des chiffres. Ces traitements sont les suivants :
    a. Mis en niveau de gris

    C'est un fait bien connu, les OCR n'ont pas besoin de couleur. Pour fonctionner ceux-ci n'ont besoins que des informations de contraste de l'image.
    De plus, de nombreuses fonctions implémentées dans les langages de programmation n'acceptent de travailler qu'avec des images préalablement décolorées.
    Nous avons utilisé, pour effectuer cette tâche, la fonction de OpenCV, « cvtColor() ». Ce fonction prend comme paramètre l'image original et donner une image grise.

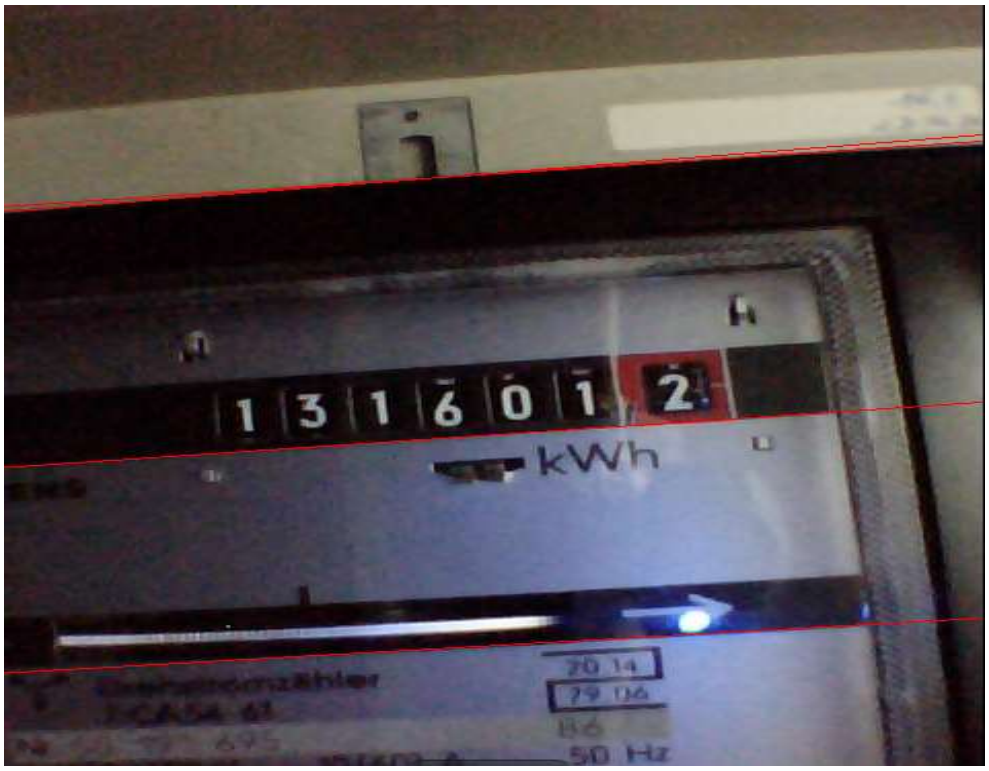


b. Reconnaissance des bords et des lignes

Comme notre cerveau est capable d'identifier les bords et les lignes très rapidement et enfin de déterminer la forme extérieure du compteur et les numéros individuels dans une large gamme de luminosité. Une grande partie d'OpenCV est dédiée à l'identification des arêtes et des lignes.
Une routine utile pour de nombreuses situations est l'algorithme de Canny. Canny () reçoit l'image en niveaux de gris en entrée et fournit une image avec les bords détectés en sortie.
Les deux paramètres de seuil de Canny dépendent de l'éclairage et du contraste de l'image et les images à contraste élevé nécessitent des valeurs de seuil élevées. Nous avons utilisé pour notre image 200 et 250 sont utiliser.

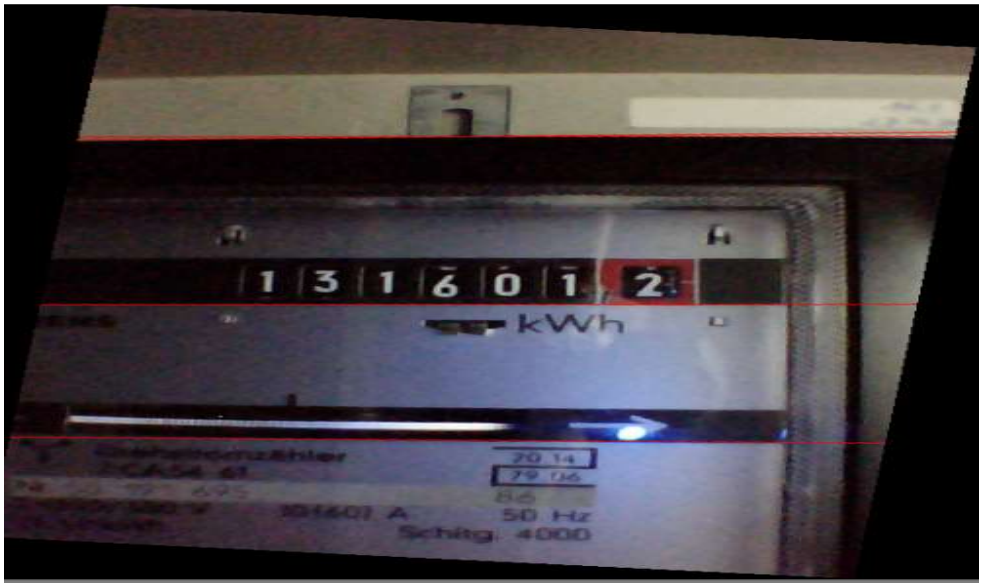


Plusieurs lignes parallèles sont visibles dans notre image bords détecté, par exemple, les limites du boîtier du compteur, du compteur et de l'anneau de comptage. La déviation des lignes horizontales a l'angle dont nous avons besoin pour aligner l'image.
Ce qui reconnaît immédiatement notre œil comme une ligne, est pour OpenCV seulement un tableau de pixels lumineux sur un fond noir à ce moment. Pour identifier les lignes, nous effectuons ensuite une transformation de Hough au moyen de cv2.HoughLines (edges, 1, np.pi/180,140).
Le seuil ici codé en dur de 140 est le nombre de votes qu'un bord doit être identifié comme une ligne.
HoughLines() renvoie les lignes vectorielles contenant toutes les lignes détectées. Chaque élément de lignes est à son tour un vecteur avec deux éléments. Le premier élément est la distance de la ligne du coin supérieur gauche de l'image, le second est l'angle par rapport à la verticale.



c. Détecter le fausser (+- 30 dég)

L'angle que nous avons eu précédemment est exactement ce dont nous avons besoin. Mais de quelle ligne ? Il y a probablement aussi les lignes verticales incluses dans le résultat. Si l'erreur de rotation compensable maximale est limitée à ± 30 °, vous pouvez filtrer toutes les lignes intéressantes avec ce critère et calculer la moyenne sur l'angle. Il convient également de noter que HoughLines () renvoie l'angle en radians, tandis que rotate () nécessite l'angle de rotation en degrés. La figure montre l'image de la caméra alignée avec les lignes utilisées pour la détermination de l'angle.
ndimage.rotate() est utilize pour la rotation d'image.



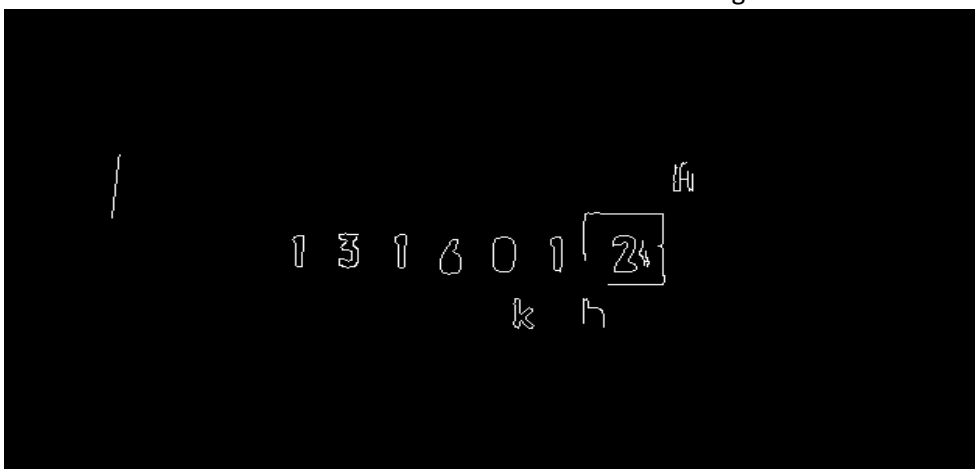
3. Isolation des digits

   a. Trouver et isoler des digits de conteur

      Pour reconnaître les chiffres, nous utilisons la détection de contours d'OpenCV, implémentée dans la fonction findContours (). Les contours trouvés sont renvoyés dans le vecteur npaContours. Chaque élément représente un contour unique, défini comme un vecteur de points. Le paramètre CV_RETR_EXTERNAL indique à findContours () de ne fournir que les limites externes.

   b. Filtrer les contours en délimitant la taille rectale

      A partir du résultat, qui contient maintenant tous les contours possibles, nous devons filtrer les chiffres intéressants. Cela se fait en deux étapes : d'abord, nous filtrons les contours en fonction de la taille de leurs boîtes englobantes:



boundingBoxes

   c. Trouver les boîtes englobantes qui sont alignées à la position « Y »

      L'image contient maintenant encore quelques perturbations en plus des chiffres utilisables. Pour identifier ce dernier, nous évaluons dans la deuxième étape les positions y et les hauteurs des boîtes de délimitation calculées. (findAlignedBoxes()) L'algorithme tente de trouver le plus grand nombre de contours de taille égale sur une ligne horizontale à partir de toutes les combinaisons possibles de boîtes englobantes car aucun autre groupe de contours n'est aussi significativement aligné.

   d. Trier les boîtes englobantes de gauche à droite

      Le résultat devrait contenir les chiffres dans leur arrangement de gauche à droite, donc les boîtes de délimitation sont triées en fonction de leur position x.

      sorted_by_X = sorted(alignedBoundBoxes, key=lambda tup: tup[0]);

e. Découper les rectangles trouvés de l'image bordées

4. OCR

a. Reconnaissance des digits et les envoyer à la base de données

Pour reconnaissance des digits, nous avons choisir le Tesseract, un moteur de reconnaissance optique de caractères pour différents systèmes d'exploitation.
Après avois coupé les parties intéressantes, des digits, et les trier de gauche à droite, nous avons la possibilité de faire la partie plus intéressante et plus « dure » de notre projet.
Comme les digits coupés sont très remplis dans les « box », on doit les éroder un peu pour qu'il soit plus « lisible » à Tesseract, pour ça nous avons utilisé le fonction erode de OpenCV.
L'idée de base de l'érosion est juste comme l'érosion du sol, elle érode les limites de l'objet au premier plan (toujours essayer de garder le premier plan en blanc). Le kernel glisse dans l'image (comme dans la convolution 2D). Un pixel dans l'image originale (1 ou 0) sera considéré 1 seulement si tous les pixels sous le noyau sont 1, sinon il est érodé (mis à zéro).

```
kernel = np.ones((4,3),np.uint8)
erosion = cv2.erode(imgErosion,kernel,iterations = 1)
```

Ici nous avons utilisé un 4 3 kernel avec plein de un.

Et seulement après nous avons utilisé notre fameuse Tesseract en implémentant sa méthode « pytesseract.image_to_string », avec les parametres

« tessedit_char_whitelist=0123456789' » pour qu'il peut reconnaître seulement les caractère number pas les autre.

5. Envoyer les résultats à la BDD

Nous avons obtenu un bon résultat et nous avons l'envoyer à notre base de données via notre serveur en utilisant php $_POST, pour sécuriser notre base de donnée.