Le génie pour l'industrie

PROJET: VISION STEREO

SYS809 –VISION PAR ORDINATEUR

AUTEUR:

FAHED BOUKHTIOUA (BOUF28098809)

PRÉSENTÉ À

MATTHEW TOEWS & AHMAD CHADDAD

MONTRÉAL, LE 07 AVRIL 2016

Table des matières

Introduction

Description haute niveau des buts du projet, l'approche, etc. Quelle est l'utilité du projet?

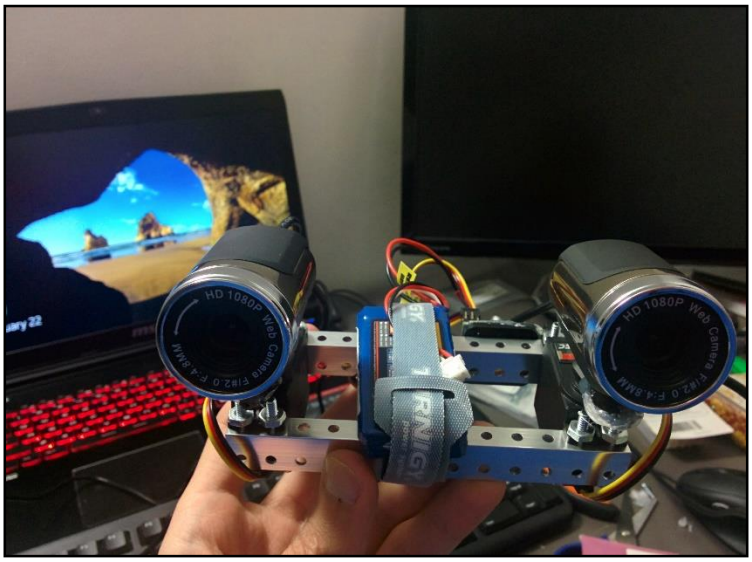
L'objectif final du projet est de traquer un objet sur le plan horizontal et déterminer sa distance en unité réelle (cm) en utilisant la vision stéréo.

L'utilité du projet est :

1. Didactique : pour appliquer la théorie vue en classe de vision stéréo sur un projet réel avec du vrai matériel.
2. Se familiariser à créer une application temps réel avec OPENCV and VC++

Pour cela il a fallu :

- Construire une plateforme de vision servo-controlée
- Appliquer des algorithmes de vision pour :
    a. calibrer les caméras
    b. corriger les images stéréo
    c. détecter l'objet
    d. Le traquer tout en estimant sa distance.



Revue de la littérature

Description des articles scientifiques et d'autres références pertinentes au projet.

Le livre de Bradski, Gary, and Adrian Kaehler. *Learning OpenCV: Computer vision with the OpenCV library*. " O'Reilly Media, Inc.", 2008 et tout particulièrement le **chapitre 11** '*Camera Models and Calibration*' et **le chapitre 12** *'Projection and 3D Vision'* a permis d'avoir la base théorique et pratique

(OPENCV) pour effectuer ce projet. En effet, le livre explique de manière complète et **très pédagogique** la géométrie du problème de vision stéréo et les maths qu'il y'a derrière ainsi que les fonctions d'OPENCV qu'il faut utiliser, sans tomber dans trop d'équations et inonder le lecteur.

Pour l'utilisation de la bibliothèque OPENCV, j'ai suivi la documentation en ligne docs.opencv.org qui explique les paramètres des différentes fonctions. Aussi, j'ai consulté le forum answers.opencv.org qui offre un support pour les utilisateurs. Par ailleurs, les exemples dans le chemin opencv\sources\samples\cpp ont été très utiles pour tester plusieurs algorithmes démos et les adapter ou s'en inspirer pour résoudre ma problématique.

Pour le volet mécanique, j'ai fait le dimensionnement du système sur le site servocity.com qui offre tous les outils nécessaires pour fabriquer ce genre de plateforme.

Méthodes

Description détaillée des techniques utilisées, les algorithmes, etc. Des diagrammes montrant les parties importantes du système et ou des mathématiques sont très utiles.

Conception matérielle :

Cameras :

J'utilise deux Webcam identiques de la marque *Ausdom* (voir annexe) qui ont :

- Un capteur CMOS 1/4.5¨
- F/NO: 2.0
- Zoom logiciel
- Focus de 30cm jusqu'à l'infinité
- Un angle de vue horizontal de 65deg.
- Résolution de la caméra 12Mpx
- Poids 130gr
- La résolution utilisée dans l'application est de 640x480 pixels

1. Pour la vision stéréoscopique, des webcams standards du marché grand public font l'affaire car nous allons les calibrer et donc corriger les distorsions dans l'image.
2. Aussi, vue le budget limité, ces caméras ont un pris adéquat pour ce type de projet didactique.
3. La résolution de l'application a été choisie pour ne pas avoir trop de temps de latence dans le suivie de l'objet. La résolution peut être augmenté mais au détriment des performances temps réel (voir chapitre amélioration).

Servomoteurs :

J'utilise deux servomoteurs identiques de la marque Hitec D645MW

Explication sur le choix du servomoteur digital :

1. Le couple :

Le poids de la caméra est de 130gr sans compter le câble qui est assez long et crée une contrainte un couple résistant important. Pour cela j'ai choisie de sur-dimensionner le servomoteur avec un couple de 12.9Kg/cm pour une alimentation 7.4V. Ceci permet de monter la camera directement sur le bras du servomoteur avec une distance de 4cm entre l'axe du moteur et le centre de la camera. Ceci nous donne donc 12.9* 1/4cm = 3.2 Kg/cm de couple, ce qui est suffisant pour la camera (130gr) et le fils. Aussi la caméra est montrée directement sur le bras du servomoteur sans système de roulement ce qui crée une tension sur les engrenages du moteur mais puisque le poids de la caméra est assez faible devant le couple du servomoteur, le système fonctionne correctement tout en restant très simple mécaniquement et pas cher.

2. La vitesse :
   Le fait d'alimenter le servomoteur avec une tension de 7.4v, permet d'obtenir une vitesse maximale de **0.17 sec/60°** à vide. Par conséquent, les caméras tournent très rapidement.

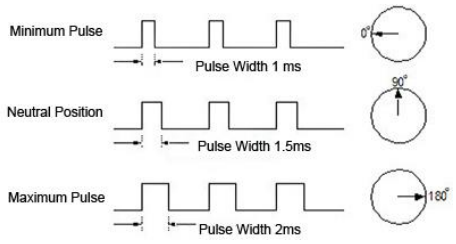
3. Réponse lisse (important pour ne pas avoir de saccade dans l'image lors de l'asservissement)
   Le choix du servomoteur digital permet :
   - ✓ Une meilleure résolution, moins de zone morte (2us seulement pour ce servomoteur) et une précision de positionnement accrue.
   - ✓ Une régulation plus rapide avec une accélération accrue.
   - ✓ Un **couple constant** (**pas de secousse sur l'arbre**) sur toute la plage de positionnement du servomoteur.

Plus d'explication sur le servomoteur digital :

Un servomoteur digital a presque la même construction qu'un servomoteur standard avec un moteur couplé à une boite d'engrenage et un potentiomètre pour le retour de position angulaire.
Cependant, la différence est que le servomoteur digital est équipé d'un microcontrôleur qui acquiert le signal de commande PWM et produit un commande adéquate pour minimiser la zone morte du moteur (zone ou le moteur ne change pas de vitesse même si la commande change) et aussi améliorer la précision et le couple.

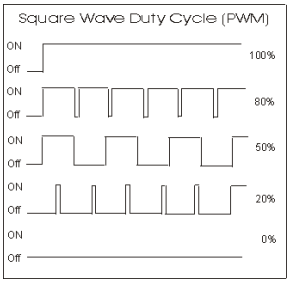
Le fonctionnement général d'un servomoteur est comme suit :

Le servomoteur reçoit un signal PWM de **50Hz** de largeur de pulse 1-2ms qui est l'image d'une position angulaire. Dans notre cas, le moteur opère dans la région [-45 º .. 0 º .. +45º] qui correspond à une commande PWM de [1100 usec .. 1500 usec .. 1900 usec].



Fonctionnement servomoteur analogique :

Lorsque on applique un signal de commande pour aller à une certaine position angulaire $\theta_{désiré}$, le servomoteur réponds en envoyant directement ce signal PWM vers le hacheur de puissance qui va créer un deuxième signal PWM de puissance d'une période de 50Hz image du signal de commande. Ainsi, on change progressivement la vitesse du moteur en fonction de la largeur d'impulsion du signal PWM. Plus la largeur d'impulsion augmente, plus la vitesse du moteur augmente.



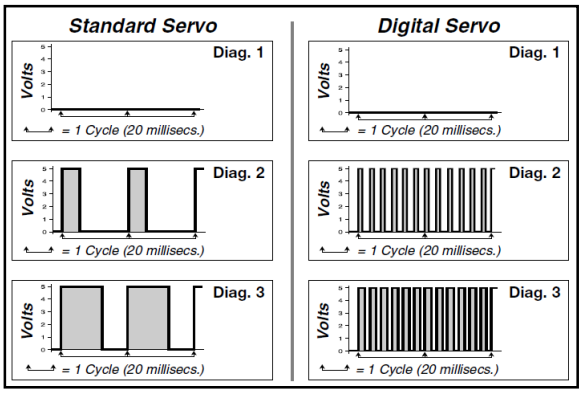
Aussi, plus largeur d'impulsion est importante, plus le moteur aura un couple plus grand qui va lui permettre d'accélérer rapidement. En parallèle, la valeur du potentiomètre qui donne la position angulaire actuelle du moteur $\theta_{mesuré}$ est lue. Dès que $\theta_{mesuré}$ se rapproche de $\theta_{désiré}$, on commence à ralentir le moteur (diminuer la largeur d'impulsion du signal PWM) jusqu'à ce qu'on arrive à la position désiré. Il faut noter que cette régulation se fait de manière transparente à l'utilisateur à une fréquence de 50Hz. L'utilisateur en effet envoie juste une seule commande au début pour demander au servomoteur d'aller à une position angulaire désiré).

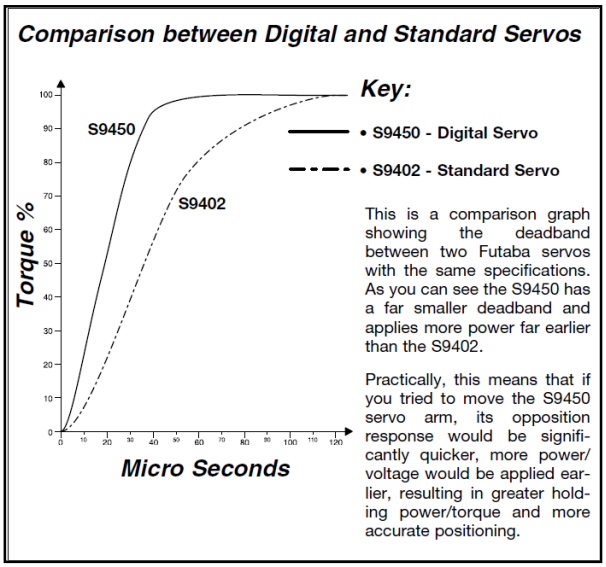
Le problème avec ce type de fonctionnement est que pour de petits changements d'angle, on a une petite largeur d'impulsion qui ne crée pas assez de couple pour changer la position du servomoteur. Ceci crée une zone morte (une zone ou le changement de vitesse ne crée aucun effet sur la position du servomoteur car ce changement est trop petit).

Fonctionnement servomoteur digital :

Le microcontrôleur du servomoteur digital acquiert le signal PWM de 50hz provenant de l'utilisateur et envoie un signal adapté à une fréquence de 300Hz, les largeurs d'impulsion seront plus courte mais avec cette fréquence élevé le moteur reçoit 6x plus de signal. Il plus réactif aux commandes, accélère et décélère nettement plus rapidement avec un couple constant.

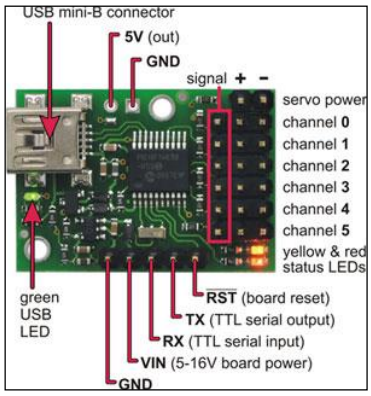
En conclusion, ceci réduit la zone morte du servomoteur avec une réactivité accrue, plus de précision et plus de couple de maintien.

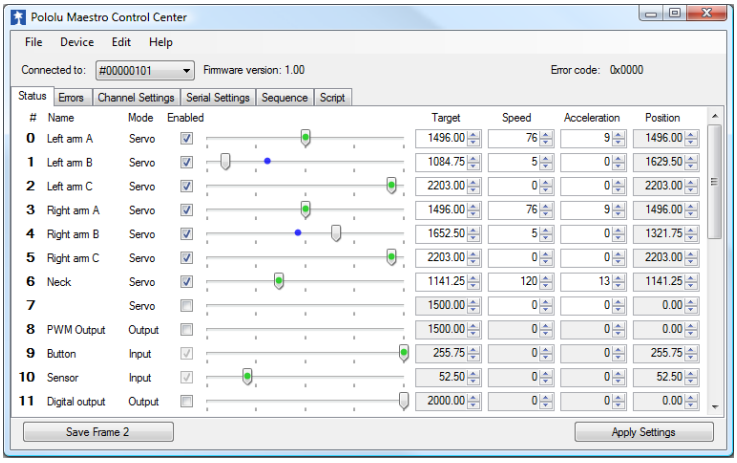




Carte Polulu :

Cette carte permet de contrôler les servomoteurs à partir de l'ordinateur grâce à une SDK en C++, C# .NET, VB et Script bash. La précision du signal PWM est de 0.25us.



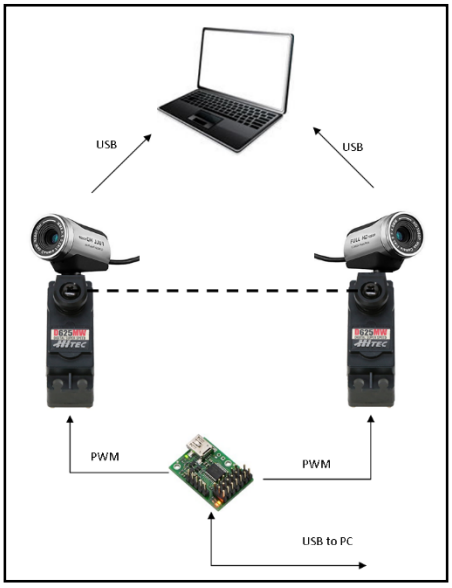


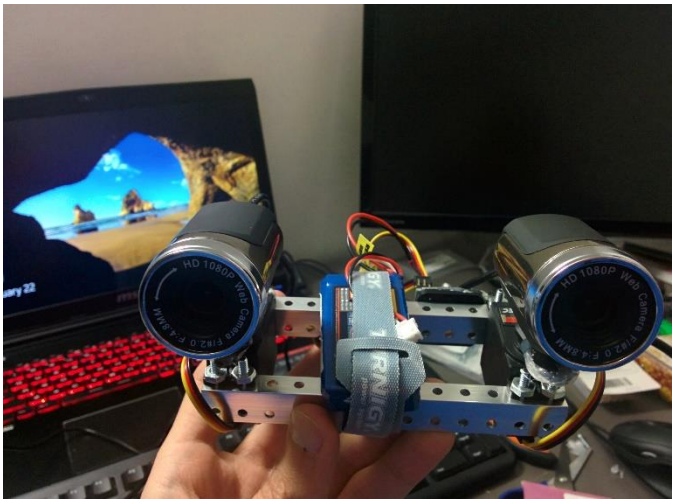
On a aussi accès à une interface graphique qui nous permet de :

1. Changer la position du servomoteur.
2. Lire la position actuelle du servomoteur.
3. Changer la vitesse du servomoteur
4. Changer l'accélération du moteur.

En changeant la position du slider, l'utilisateur ajuste la position désiré qui est noté en bleu, le rond vert indique la position actuelle du servomoteur. Par exemple : pour le canal 1 de la figure au-dessus, on change la vitesse du moteur à 5 ce qui le rend plus lent. Lorsque on change la position du curseur le rond vert va bouger lentement jusqu'à ce qu'il arrive au rond bleu.

Structure du système :





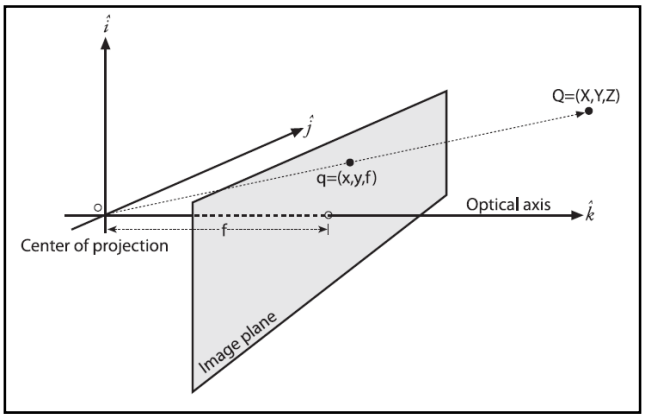
- La distance entre les deux caméras est de **11cm** (Baseline). Elle a été choisie pour avoir une bonne précision en comparant avec d'autre produit équivalent sur le marché comme la *bumbelbee* qui une *baseline* de 12cm, tout en restant une plateforme compacte.
- L'angle de balayage des servomoteurs est de **±45deg**.
- Le champ de vision horizontal des caméras est de **65deg.**
- Les caméras sont **toujours gardées parallèles** et **horizontale** comme lors de la calibration pour que les matrices de projection, de translation et rotation restent valides. (plus dans le chapitre amélioration pour l'autre cas).
- Nous avons 3 câbles USB vers le PC : deux pour les caméras et un pour la carte Polulu.
- L'alimentation de la carte Polulu se fait par le port USB
- L'alimentation des servomoteurs se fait par une batterie LIPO de marque VENOM de 2400mAH 5C 2S (7.4v)

La théorie de la vision stéréo :

Modèle de caméra simplifié (*Pinhole*) :

Dans notre projet, on utilise le modèle Pinhole pour modéliser notre caméra couplé avec une modélisation des aberrations de la caméra pour se rapprocher le plus possible du modèle réel sans pour autant être trop complexe.

Par principe, le modèle de la caméra *pinhole* bloque tous les rayons sauf ceux qui passent par la petite ouverture au centre. Dans la figure suivante, le plan d'image a été translaté d'une distance correspondant à la distance focal f pour avoir un modèle mathématique plus simple.



Ainsi, les rayons qui quittent le point **Q** de l'objet distant se dirigent vers le centre de projection **O.** L'intersection du rayon avec le plan de l'image est le point **q**.

Suivant le schéma on a :

- Q=[X Y Z] : Le point de l'objet distant
- q=[x,y,f] La projection de Q sur le plan d'image.
- F est la distance focale de la camera, qui est aussi la distance entre l'ouverture et le plan d'image
- Z est la distance entre la camera et l'objet.
- X est la hauteur de l'objet.

En utilisant la relation des triangles similaires, on obtient la relation suivante :

$$x = f\frac{X}{Z}$$

Modélisation des imperfections et géométrie spatiale des caméras

Le modèle de caméra *pinhole* est trop simpliste pour rester fidèle à la vraie caméra. En effet, il existe des imperfections dans la fabrication industrielle de la caméra et la lentille qu'on doit prendre en considération pour essayer de rester le plus fidèle possible au modèle de la caméra réel.

1. En premier lieu, le centre du capteur CMOS de la caméra n'est pas toujours sur l'axe optique.

Pour cela, on ajoute deux paramètres **cx** et **cy** (en pixels) pour modéliser ce décalage. **Par conséquent, la paire de coordonnées (cx,cy) définie notre nouveau centre optique qu'on appelle point principal**.

2. Un capteur CMOS de caméra d'entrée de gamme est **rectangulaire** au lieu d'être carrée.

Pour cela, nous avons deux distances focales fx et fy **(en pixel)** qui sont proportionnelle à la distance focale de la caméra F **(mm)**, la largeur et longueur du capteur CMOS sx , sy **[pixels/mm] .**

$$fx = F.sx$$

$$fy = F.sy.$$

Les coordonnées X, Y en pixels:

$$Xscreen = fx.\frac{X}{Z} + cx$$

$$Yscreen = fy.\frac{Y}{Z} + cy$$

NB:

Les paramètres **F**, **sx** et **sy** ne peuvent être mesuré directement par la calibration. En effet, seulement la combinaison $fx = Fsx$ **et** $fy = Fsy$ peut-être estimer avec le processus de calibration.

Une scène est formée par la projection de point 3D sur le plan d'image en utilisant une **transformation en perspective.** On obtient la relation qui suit entre le point de l'objet en 3D et les coordonnées homogènes en pixel sur le plan d'image de la projection de ce dernier.

$$s.q = M.[R|T].Q = s\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \left[\begin{array}{ccc:c} r11 & r12 & r13 & t1 \\ r21 & r22 & r23 & t2 \\ r31 & r32 & r33 & t3 \end{array}\right] \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

- ✓ **M=** Matrice des paramètres intrinsèque de la camera
- ✓ **Q** = coordonnées physique (3D)
- ✓ **q** = Coordonnées homogènes du point projeté sur le plan d'image.
- ✓ **[R|T]** = matrice de rotation et vecteur de translation du repère 3D de l'objet vers le repère 3D de la caméra. Cette matrice est aussi appelé **matrice des paramètres extrinsèques.**
- ✓ **s** facteur d'agrandissement (==1 dans notre cas).

Paramètres intrinsèques de la caméra :

La matrice M = $\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$ contient les paramètres intrinsèques de la camera.

Les résultats suivant sont obtenu après calibration sur OPENCV.

CAMERA LEFT

Instrinsic parameter matrix M1: [3x3] = $\begin{matrix} 805.13 & 0 & 329.56 \\ 0 & 805.13 & 208.95 \\ 0 & 0 & 1 \end{matrix}$

CAMERA RIGHT

Instrinsic parameter matrix M2: [3x3] = $\begin{matrix} 805.13 & 0 & 329.72 \\ 0 & 805.13 & 207.43 \\ 0 & 0 & 1 \end{matrix}$

Paramètre extrinsèque de la caméra

Les paramètres extrinsèques nous informent sur la géométrie de la scène. Ils nous permettent de localiser, dans chaque prise de vue, l'objet calibré par rapport à la caméra.

Pour cela nous avons les matrices de rotations suivantes pour localiser un objet dans un espace 3D :

- 3x rotations $R(\psi); R(\theta); R(\phi)$
- 3x translations T = ($T_X; T_Y; T_Z$)

La matrice de rotation qui combine les 3 angles d'Euler suivant la convention ZYX est :

$$R = R_z(\theta)R_y(\phi)R_x(\psi)$$

Le résultat obtenu après calibration est sauvegardé dans le fichier **extrinsic.xml**

NB :

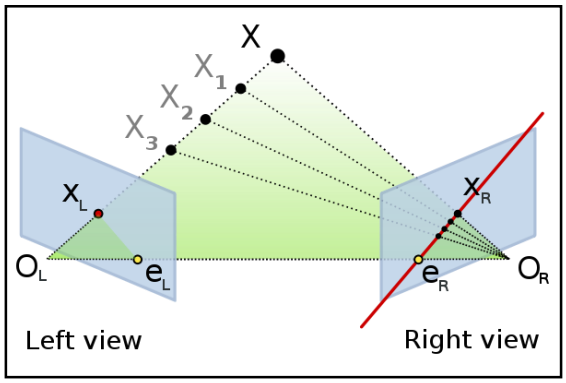
La matrice de rotation qui est stocké dans le fichier extrinsic est sous le format compressé (3 vecteurs de rotations). Si on veux obtenir la matrice de rotation standard, on peut appeler la fonction OPENCV *cvRodrigues2* qui convertit un vecteur de rotation en une matrice de rotation.

R= $\begin{bmatrix} 0.9999 & -0.01409 & 0.0043 \\ 0.0142 & 0.9998 & -0.0146 \\ -0.00411 & 0.0146 & 0.9999 \end{bmatrix}$

T= $[-11.013 \quad 0.01934 \quad 1.7865]$

On remarque dans le vecteur T que T(1)= **-11.013 cm** qui est la distance entre les deux caméras (*baseline*). OPENCV utilise la règle de la main droite pour le système de coordonnée. L'origine du système de coordonnée est centre optique de la caméra de gauche.

Géométrie épipolaire :



Le centre d'image de la caméra de gauche noté $\mathbf{e_L}$ se projette sur un point distinct noté $\mathbf{e_r}$ dans l'autre plan de l'image de la caméra de droite. On appelle $\mathbf{e_L}$ et $\mathbf{e_r}$ des points épipolaires.

Si on observe la figure, on remarque qu'avec une seule caméra, on a juste XL qui est la projection du point X sur le plan d'image. Ce dernier, peut être localisé n'importe où sur la ligne (OL,XL).

Cependant, si on avait deux caméras, toutes les localisations possibles du point XL vu dans le plan d'image gauche sont définie par la ligne (eR,xR) qui passe par la projection de X sur le plan droit (xR) et le point épipolaire (eR).

Ainsi, le problème se transforme d'une recherche 2D pour un point correspondant sur les deux images vers une recherche 1D le long d'une ligne épipolaire. Ceci nous amène vers un algorithme de vision stéréo beaucoup moins complexe qui permet aussi de détecter et enlever les mauvaises correspondances.

Nous introduisons dans ce qui suit deux matrices importantes pour la géométrie épipolaire.

Matrice essentielle

Cette matrice contient les informations de rotation et translation => elle relie les deux caméras dans le **système de coordonnées physiques.**

Pour définir la matrice E, on considère la démarche suivante :

1. Le point P de l'objet vu par la caméra de droite, dans le repère de cette caméra est :
$$Pr = R(Pl - T)$$
2. Le plan épipolaire défini en vert sur la figure s'écrit comme un vecteur du plan et la normale à ce plan, ce qui donne :
$$(Pl - T)^T (T \times Pl) = 0$$

On remplace $(Pl - T)^T$ en utilisant 1 et en l'injectant dans 2 pour obtenir la relation $R^{-1}Pr\ (T \times Pl) = 0$

Puisque R est une matrice de rotation, $R^{-1} = R^T$

On alors $(R^T \mathrm{Pr})^T\ (T \times Pl) = Pr^T\ Rzyx \begin{bmatrix} 0 & -Tz & Ty \\ Tz & 0 & -Tx \\ -Ty & Tx & 0 \end{bmatrix} Pl = \boldsymbol{Pr^T\ E\ Pl} = 0$

avec

$$\boldsymbol{E} = Rzyx \begin{bmatrix} 0 & -Tz & Ty \\ Tz & 0 & -Tx \\ -Ty & Tx & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\psi & -sin\psi \\ 0 & sin\psi & cos\psi \end{bmatrix} \begin{bmatrix} cos\theta & 0 & sin\theta \\ 0 & 1 & 0 \\ -sin\theta & 0 & cos\theta \end{bmatrix} \begin{bmatrix} cos\phi & -sin\phi & 0 \\ sin\phi & cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & -Tz & Ty \\ Tz & 0 & -Tx \\ -Ty & Tx & 0 \end{bmatrix}$$

ψ, θ, ϕ are yaw, pitch and roll angles.

Matrice fondamentale :

La matrice fondamentale contient les informations de translations, rotation et aussi les paramètres intrinsèques de la caméra => Elle relie donc les deux caméras **dans le système de coordonnées de pixels.**

En connaissant la relation entre le point **q**= $\begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \\ \boldsymbol{1} \end{bmatrix}$ en coordonnées homogènes en pixel et le point p=$\begin{bmatrix} \boldsymbol{X} \\ \boldsymbol{Y} \\ \boldsymbol{Z} \end{bmatrix}$

dans le repère de coordonnées physique du système et aussi en connaissant les paramètres intrinsèque de la caméra défini par la matrice de la caméra M, on obtient les relations suivantes entre le point q et le point r pour chaque caméra indépendamment.

$$qr = Mr\ pr \quad \text{(Camera de droite)}$$

$$ql = Ml\ pl \quad \text{(Camera de gauche)}$$

En utilisant la relation obtenu dans la matrice essentielle et en remplaçant p par $M^{-1}q$ on peut trouver l'équation suivante :

$$qr^T (Mr^{-1})^T\ E\ \ Ml^{-1}\ ql$$

$$= qr^T F\ ql = 0$$

On définit alors la matrice fondamentale comme :

$$F = (Mr^{-1})^T\ E\ \ Ml^{-1}$$

Il existe plusieurs algorithmes robustes pour l'estimation de la matrice F, on cite RANSAC et LMEDS.

- ✓ Pratiquement, on peut appeler la fonction OPENCV **cvFindFundamentalMat()** pour retrouver la matrice fondamentale.

Après, l'obtention de la matrice fondamentale, on peut calculer les lignes épipolaire pour la caméra de droite, en ayant une liste de points dans la caméra de gauche et vis versa. L'équation d'une ligne épipolaire est $ax + by + c = 0$.

- ✓ On peut aussi utiliser la fonction OPENCV **CvComputeCorrespondEpilines()** pour calculer cette équation.

Processus de calibration stéréo

Concept général de la procédure

La calibration de la caméra estime les paramètres de la lentille et du capteur de la caméra. Pour résumer nous avons besoins d'estimer les paramètres suivants que nous allons expliquer dans les sections suivantes :

- **5 paramètres de distorsion :**
  - **3 Radial [k1 k2 k3]**
  - **2 Tangentiel [p1 p2]**
- **4 intrinsèque paramètres : [fx,fy,cx,cy]**
- **6 extrinsèques paramètres : [3 rotations and 3 translations]**

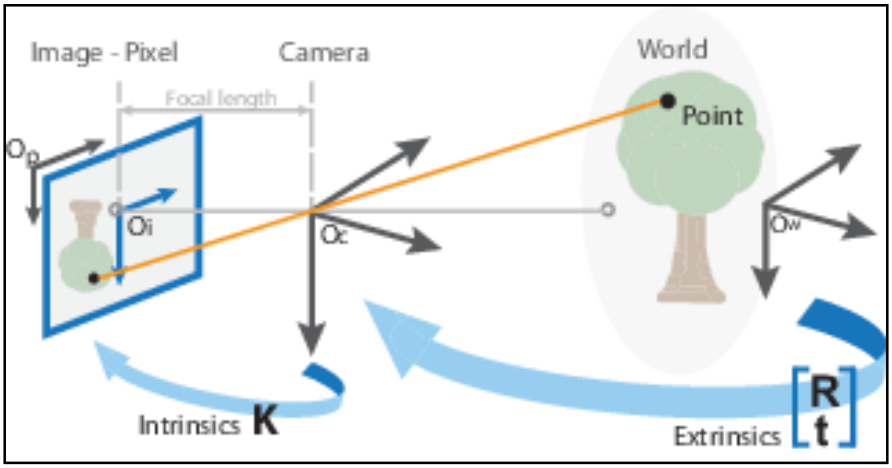
Pour estimer les paramètres de la caméra, nous avons besoin d'avoir un set de points réels en coordonnées 3D et leurs points 2D correspondants. Une méthode pour obtenir la correspondance est d'utiliser plusieurs images d'un modèle de calibration (un échiquier par exemple). On choisit l'échiquier parce qu'il offre une forme très structuré qui est bien adapté pour les algorithmes de détection. En utilisant, les correspondances trouvées, on peut résoudre le problème pour trouver les paramètres de la caméra.

NB : L'unité du système de coordonnée de la caméra sont ceux du carré choisi de l'échiquier. Dans notre cas, on a des carrés de **3x3 cm** pour un échiquier de **9x6.**

L'unité du système de coordonnées est donc <u>en cm.</u>

L'origine du système de coordonnée de la caméra est localisée à son centre optique et ces axes x et y définissent le plan d'image.

En se référant à la figure suivante, on voie le passage entres les systèmes de coordonnées:



- ✓ **Les points 3D sont transformés vers les coordonnées de la caméra en utilisant les paramètres extrinsèques estimés.**
- ✓ **Les coordonnées de la caméra sont ensuite mappées dans le plan d'image en utilisant les paramètres intrinsèques.**

NB : Nous avons besoins de prises de vues assez différentes pour chaque paire d'images de l'échiquier pour produire une bonne estimation des paramètres de la caméra. Pour cela, il nous faut suivre les directives suivantes lors de la calibration :

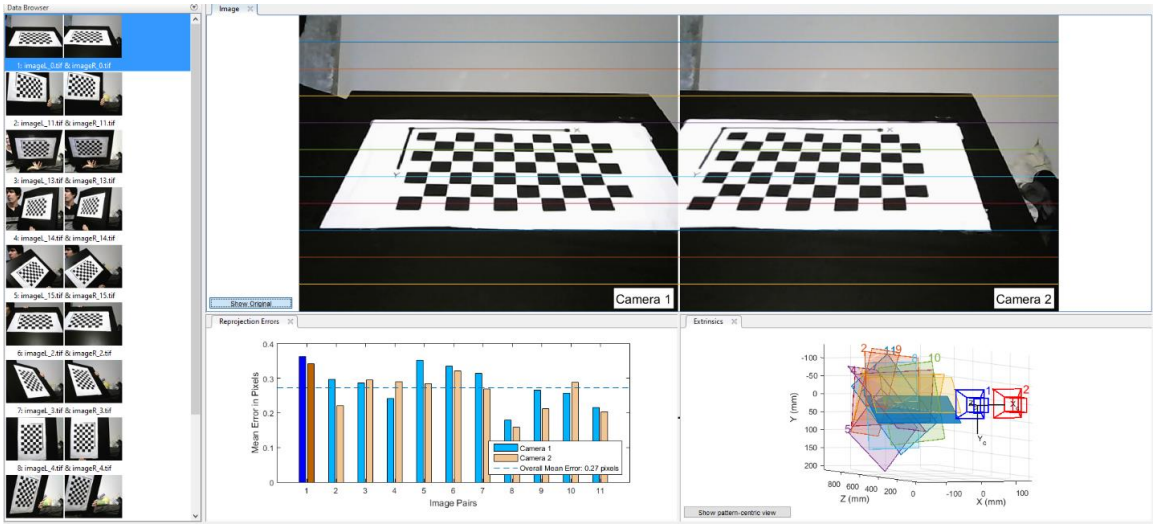
- ✓ **Calibration sur l'axe X :**
  L'échiquier est détecté au niveau des bords droits et gauches du champ de vision.
- ✓ **Calibration sur l'axe Y:**
  L'échiquier est détecté au niveau des bords haut et bas du champ de vision.
- ✓ ***Skew* :**
  L'échiquier est détecté à différents angles de la caméra.
- ✓ **Calibration de la taille :**
- ✓ L'échiquier couvre le champ de vision en entier.
- ✓ **Calibration X, Y et taille:**
- ✓ L'échiquier est orienté vers la gauche, droite, haut et bas.

En sortie de l'algorithme de calibration, on obtient deux fichiers qu'on utilisera dans la phase suivante de rectification des images :

« intrinsinc.xml » qui contient les paramètres intrinsèque de la camera.

« extrinsinc.xml » qui contient les paramètres extrinsèques de la caméra.

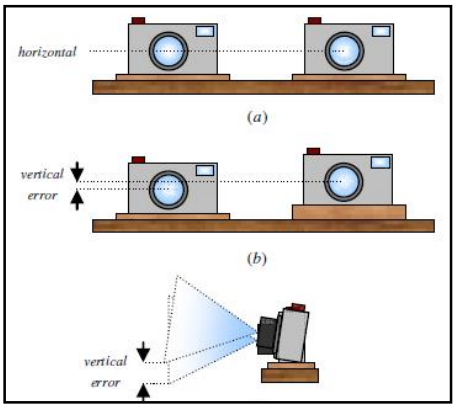
Matlab a aussi une application qui permet de calibrer les caméras stéréoscopique basé sur le même algorithme qu'OPENCV.



Algorithme de vision stéréo du projet

Les règles suivantes sont à considérer lors de la construction d'une plateforme de vision stéréoscopique avec modèle parallèle.

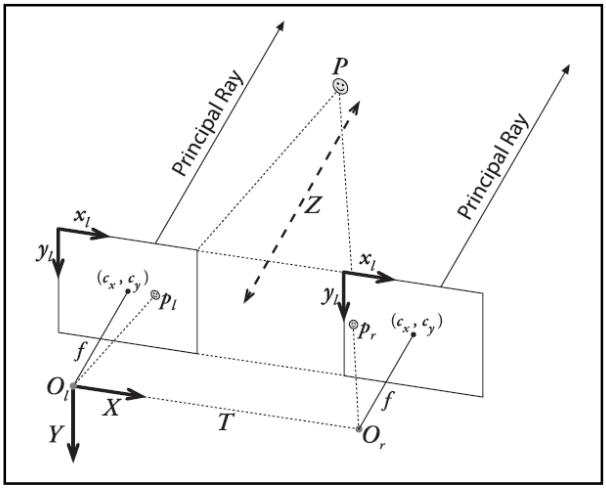
- Il faut arranger les deux caméras de façon qu'elles soient **parallèles** et **alignés horizontalement** sinon on aurait une erreur verticale comme le montre la figure ci-dessous.
- Les caméras doit être synchronisées (les images prises au même moment).



NB :

Si on veut obtenir une plus fine résolution de l'estimation de la profondeur pour les objets proches, on peut tourner les caméras un petit peu vers l'intérieur. Ainsi, leurs rayons principaux vont s'intercepter à une distance finie. Donc, après un alignement mathématique, cette configuration va introduire un décalage sur l'axe x qui faut soustraire de la valeur de disparité.

Calibration stéréo



On considère le point P de l'objet qu'on veut calculer la distance, on veut trouver la relation entre le point en 3D et le point projeté en pixel.
D'abord, on considère les relations suivantes :

- Le Point P dans le système de coordonnée de gauche de la camera est noté Pl:

$$Pl = Rl\ P + Tl$$

avec (Rl et Tl matrice de rotation et translation du système de coordonnée de l'objet vers le système de coordonnée de la caméra droite)

- le Point P dans le système de coordonnée de droite de la camera est noté Pr:

$$Pr = Rr\ P + Tr$$

avec (Rr et Tr matrice de rotation et translation du système de coordonnée de l'objet vers le système de coordonnée de la caméra droite)

- La matrice de Rotation entre les deux cameras:

$$R = Rr(Rl)^T$$

- Le vecteur de translation entre les deux cameras:

$$T = Tr - R\ Tl$$

Ainsi, la relation entre les deux vues est donnée par :

$$Pl = R^T(Pr - T)$$

La matrice de rotation R va transformer la caméra de droite dans le même plan que la caméra de gauche. **Ceci permet d'avoir deux plans d'image qui sont coplanaire mais pas encore aligné sur les lignes.**

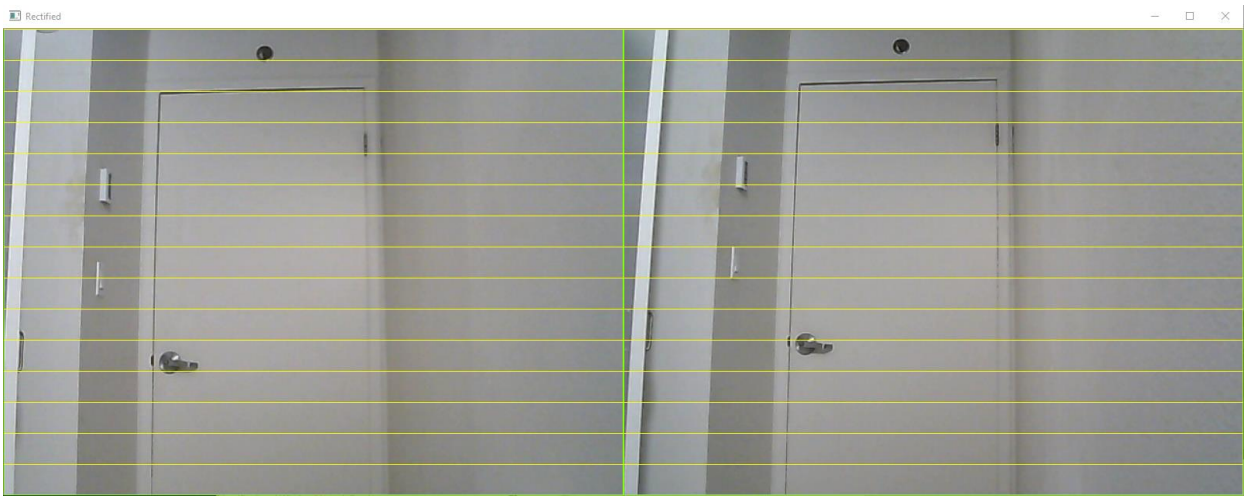
La fonction OPENCV **cvStereoCalibrate**() utilise l'algorithme suivant pour trouver la matrice de rotation R et le vecteur de translation T.

a. Trouver les paramètres de rotation et translation de l'image de l'échiquier relativement à la caméra. Ceci ce fait pour chaque caméra séparément. On obtient alors les matrices **Rl,Rr** et **Tl,Tr.**
b. Pour chaque image de l'échiquier, on utilise les équations cité ci-dessus pour trouver les matrice de rotation R et T qui lient les deux caméras entre elle.

c. On calcule la médiane de R et T comme paramètre d'initialisation pour un algorithme de *Levenberg-Marquardt* pour trouver le minimum local de l'erreur de re-projection des coins de l'échiquier. Nous obtenons alors, une bonne estimation de la matrice **R** et du vecteur **T**.

Correction des distorsions

L'image suivante montre les aberrations que peux créer la webcam que j'utilise :



On remarque **que :**

1. **les lignes ne sont pas bien droites, surtout sur les bords de l'image**.
2. On note aussi que la paire d'image n'est pas aligné horizontalement.

Pour corriger cela, il faut estimer des paramètres de distorsions pour corriger l'image.

Modélisation de la distorsion de la lentille

Distorsion radiale

Ce type de déformation est induit par la forme de la lentille. Cette dernière, déforme la localisation des pixels près des bords de l'image (effet fish-eye).

La distorsion radiale r est 0 au voisinage du centre optique (x=0, y=0) et augment lorsque on se rapproche vers la périphérie comme le montre la figure ci-dessous.

OPENCV introduit 6 paramètres pour corriger cette distorsion **k1,k2,k3,k4,k5,k6.**

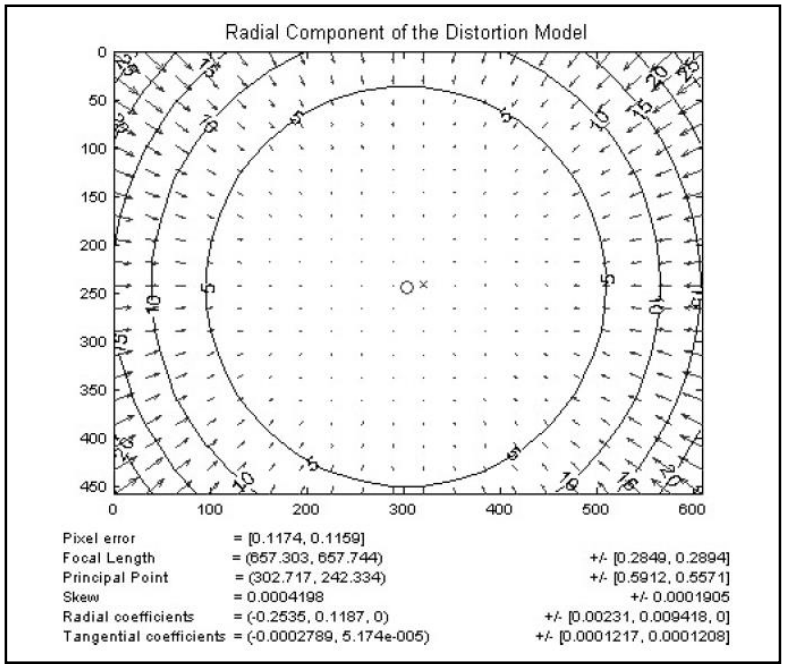
On obtient les coordonnées corrigée en pixel avec la relation suivante :

$$X_{corrected} = x\,(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)/(1 + k_4 r^2 + k_5 r^4 + k_6 r^6)$$
$$Y_{corrected} = y\,(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)/(1 + k_4 r^2 + k_5 r^4 + k_6 r^6)$$

Avec $r^2 = x^2 + y^2$.

La distorsion radiale est la plus importante par rapport aux autres types de déformations (tangentielle et skew).

NB : OPENCV a utilisé une approximation polynomiale de la fonction de distorsion. Cette fonction est symétrique c'est pour cela que nous avons que de puissance paires.



Distorsion tangentielle

Cette déformation est induite par le processus de fabrication de la caméra. Les lentilles ne sont pas exactement parallèles au plan du capteur CMOS. OPENCV introduit deux paramètres p1 et p2 pour corriger cette distorsion. Ainsi, nous obtenons les coordonnées corrigées en pixel comme suit :

$$X_{corrected} = x + 2p_1 y + p_2(r^2 + 2x^2)$$
$$Y_{corrected} = y + 2p_2 x + p_1(r^2 + 2y^2)$$

Résultats :

Au final, le vecteur de distorsion obtenue d'OPENCV après la calibration est le suivant :

$D = [\boldsymbol{k1}; \boldsymbol{k2}; \boldsymbol{p1}; \boldsymbol{p2}; k4; k5; k6]$

Distortion vector for CAM1 :

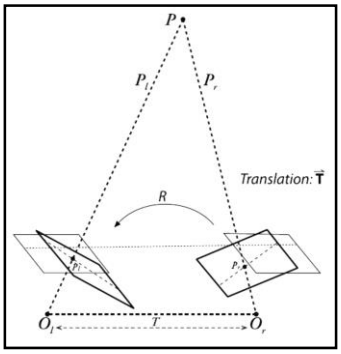
D1: [1x8] $= -0.1915 \quad 0.3651 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad -0.635$

Distortion vector for CAM2 :

D2: [1x8]$= -0.1928 \quad 0.7990 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 2.2225$

NB : La distorsion tangentielle est considéré négligeable.

Rectification



En utilisant, la matrice de rotation **R** et le vecteur de translation **T** obtenu dans la phase de Notre objectif dans cette phase est d'aligner les deux plans de la caméra dans un même plan pour que chaque ligne de pixel de la caméra de gauche s'aligne parfaitement avec sa ligne correspondante pour la caméra de droite.

La méthode d'OPENCV **cvStereoRectify()** utilise la méthode de 'Bouguet' . Cet algorithme utilise R et T comme données en entrée de son algorithme. Cette méthode essaie de minimiser les distorsions après re-projection tout en maximisant la zone de visualisation.

On voie bien sur la figure, qu'après le réalignement, il existe des zones noires qui ne peuvent pas être projeté dans les deux images.

NB : La fonction **cvStereoRectify()** calcule les termes qu'on va utiliser pour la rectification **mais n'applique pas la rectification à l'image**. Pour cela, on appelle la fonction **cvInitUndistortRectifyMap()** qu'on verra dans les sections qui suivent.

Boguet algorithme, calcule la matrice de rotation R_{rect} qui fait tourner la caméra de gauche autour de son centre de projection pour que les lignes épipolaire deviennent horizontales et que les épipoles sont à l'infini.

Pour minimiser les distordions après re-projection, la caméra de gauche est tourner mathématiquement une moitié de rotation par la matrice r_l et la caméra de droite est tourner mathématiquement une moitié de rotation r_r.

$$\boldsymbol{R1} = \boldsymbol{R_{rect}}\, \boldsymbol{r\_l}$$

$$\boldsymbol{R2} = \boldsymbol{R_{rect}}\, \boldsymbol{r\_r}$$

L'alignement des lignes des deux caméras est fait avec les matrices de rotation résultante R1 et R2.

On sait que la matrice de projection P prend un point 3D et le transforme en un point 2D dans les coordonnées homogènes comme suit :

$$P\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

Les coordonnées du point sur l'écran sont : X_{screen}= x/w, Y_{screen}= y/w.

En utilisant le système de coordonnées de gauche, on a la matrice de projection **P1** pour la caméra de gauche comme suit :

$$\boldsymbol{P1} = Mrect\_l\ Pl' = \begin{bmatrix} fx & 0 & cx \\ 0 & fy & cy \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- La matrice de projection de la caméra de droite **P2** :

$$\boldsymbol{P2} = Mrect\_r\ Pr' = \begin{bmatrix} fx & 0 & cx \\ 0 & fy & cy \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 & Tx \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

- Les points en 2D peuvent être projetés en 3D en ayant comme entré de l'algorithme les coordonnées en pixel et les paramètres intrinsèques de la caméra. En utilisant la matrice Q de re-projection

$$Q = \begin{bmatrix} 1 & 0 & 0 & -cx \\ 0 & 1 & 0 & -cy \\ 0 & 0 & 0 & f \\ 0 & 0 & -\frac{1}{Tx} & \frac{cx - c'x}{Tx} \end{bmatrix}$$

avec

- c'x est le point principal projeté sur l'axe x de la caméra de droite.
- cx-c'x= 0 si on est en configuration parallèle (ce qui est notre cas).

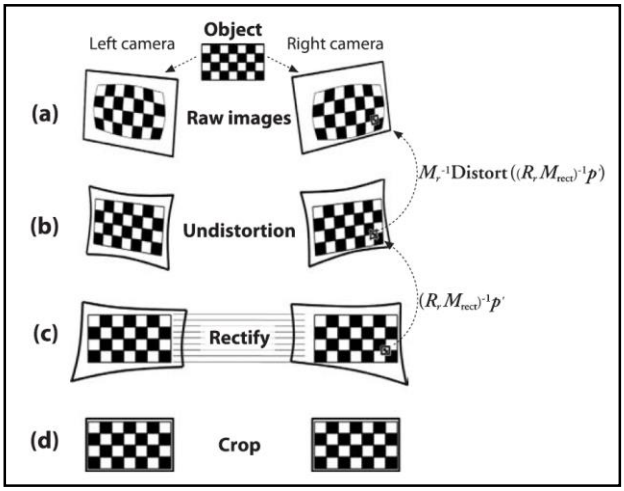
- **Ainsi, on ayant les coordonnées homogènes 2D d'un point et la disparité associé à ce point, on peut projeter ce dernier en 3D en utilisant la formule suivante :**

$$Q\begin{bmatrix} x \\ y \\ d \\ 1 \end{bmatrix} = \begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix}$$

Les coordonnées 3D sont : X/W, Y/W, Z/W.

Finalement, on peut calculer les vecteurs de rectifications pour la caméra de gauche et de droite en appelant la fonction **cvInitUndistortRecitfyMap()** pour chaque caméra.

En se référant à la figure ci-dessous, le processus de rectification procède à l'envers de l'étape **c** vers l'étape **a** dans un processus de *mapping* inverse.



Pour chaque pixel de type entier dans l'image rectifié **c**, on cherche ces coordonnées dans l'image **b** distordue et puis nous utilisons ces derniers pour chercher les vrais coordonnées de type float dans l'image originale **a**. Ces coordonnées float sont ensuite interpolés pour le pixel entier le plus proche dans l'image originale et la valeur trouvé est enfin utilisé pour remplir le pixel rectifié de type entier dans l'image **c** de destination.

Au final, nous obtenons deux vecteurs de *mapping* **mapx** et **mapy** qui sont la sortie de la fonction **cvInitUndistortRecitfyMap**(). Ces vecteurs nous indiquent comment on doit relocaliser les pixels sources pour chaque pixel de l'image de destination. Ensuite, nous utilisons ces vecteurs comme entrée pour la fonction **cvRemap()** qui va appliquer les transformations sur l'image.

Maintenant, les deux images sont bien alignées horizontalement comme le montre la figure ci-dessus.

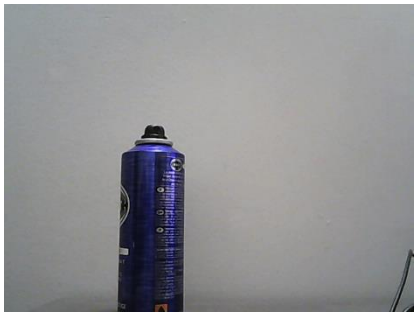
Détection de l'objet d'intérêt

L'objectif de cette phase est calculer la position du centre de l'objet d'intérêt pour les deux images.

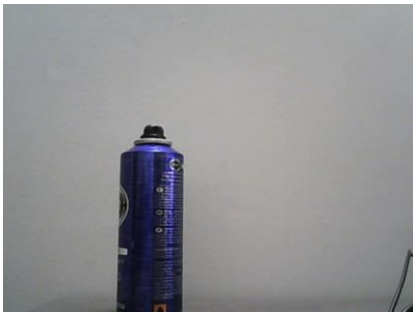
NB : Si le processus de rectification sont bien fait, on aurait les mêmes coordonnées y pour l'objet puisque les images des deux caméras sont alignée horizontalement.

Les étapes de filtrages sont les suivantes :

L'image originale gauche :

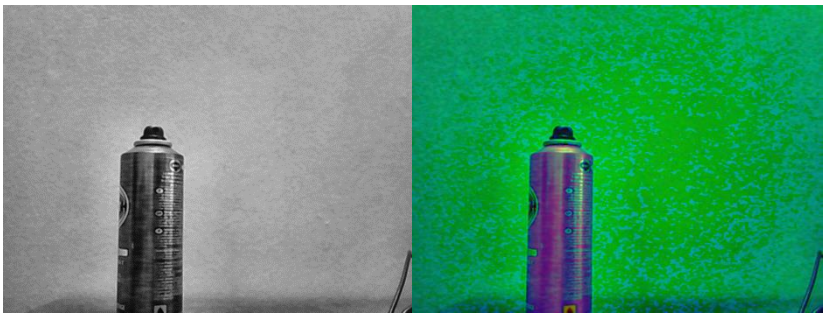


1. Filtrer l'image avec un filtre Gaussian 3x3 pour réduire le niveau de bruit.



2. Convert the image to HLS and extraire le canal de luminescence.

3. Utiliser CLAHE (Contrast Limited Adaptive Histogram Equalization algorithm) pour égaliser l'histogramme et augmenter le contraste.



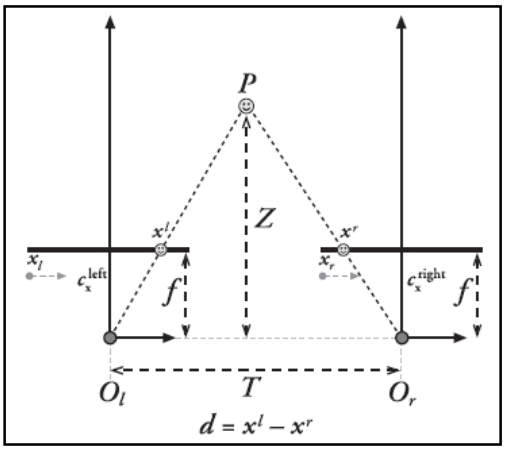
4. Faire un seuillage pour laisser la couleur désiré (HLS Filter)

5. Appliquer des opérations morphologiques (ouverture et fermeture) pour enlever les pixels isolés du premier plan et remplir les petits trous de notre blob.

6. On extrait le blob le plus grand et on calcule son centre

7. Ce filtre est appliqué séquentiellement sur les deux images de caméras. On obtient le couple de coordonnée (XL, YL) et (XR, YR) en coordonnées de pixels.
8. On calcule la disparité D= XL-XR

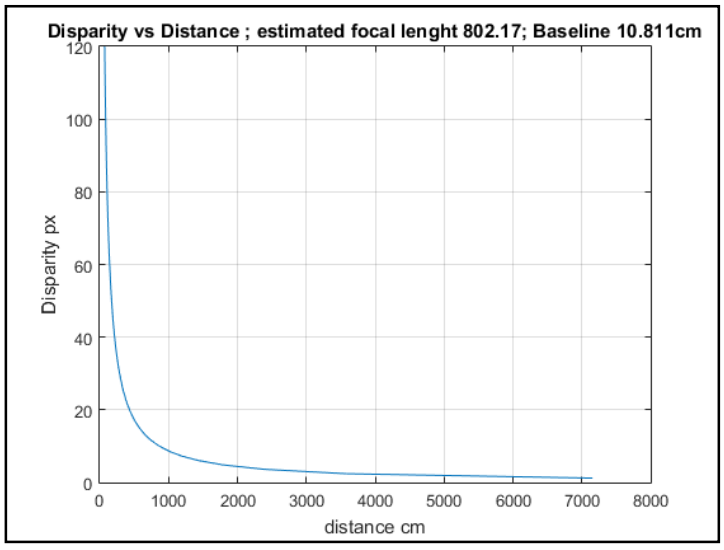
Triangulation



En utilisant les triangles similaires, on obtient l'équation suivante pour calculer la profondeur pour une configuration avec caméra parallèle comme le montre la figure ci-dessus.

$$Z = \frac{fT}{x_l - x_r}$$

avec x_l, x_r le centre de l'objet détecté qu'on a retrouvé avec l'algorithme de détection ci-haut.



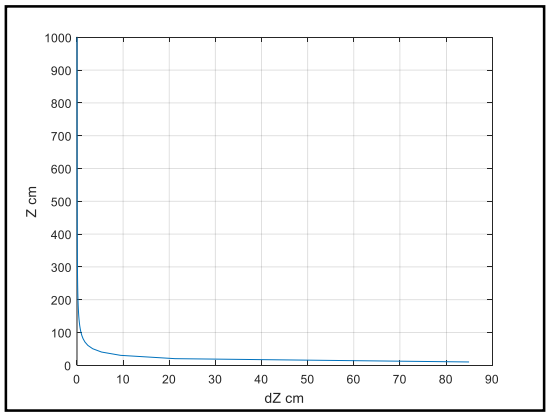
Les points principaux c_x^l, c_x^r sont les points ou les raies principales pour chaque image interceptent le plan d'image correspondant. Si les caméras sont exactement parallèle, alors $c_x^l = c_x^r$.
Cependant, si les caméras ne sont pas parallèles, les raies principales se coupent à une distance finie et par conséquent l'équation de calcul de profondeur devient :

$$Z = \frac{fT}{((x_l - x_r) - (c_x^l - c_x^r))}$$

NB : En réalité, les caméras ne vont jamais être exactement parallèle, c'est pour cela que nous rectifions mathématiquement les images gauches et droite en configuration parallèle (phase rectification de l'algorithme).

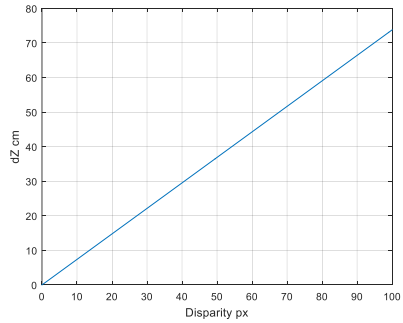
Par ailleurs, on connaît la plus petite incrémentation possible du terme de disparité, on peut déterminer la plus petite incrémentation de distance achevable avec l'équation suivante :

$$\triangle Z = \Delta d \, \frac{Z^2}{\mathrm{fT}}$$



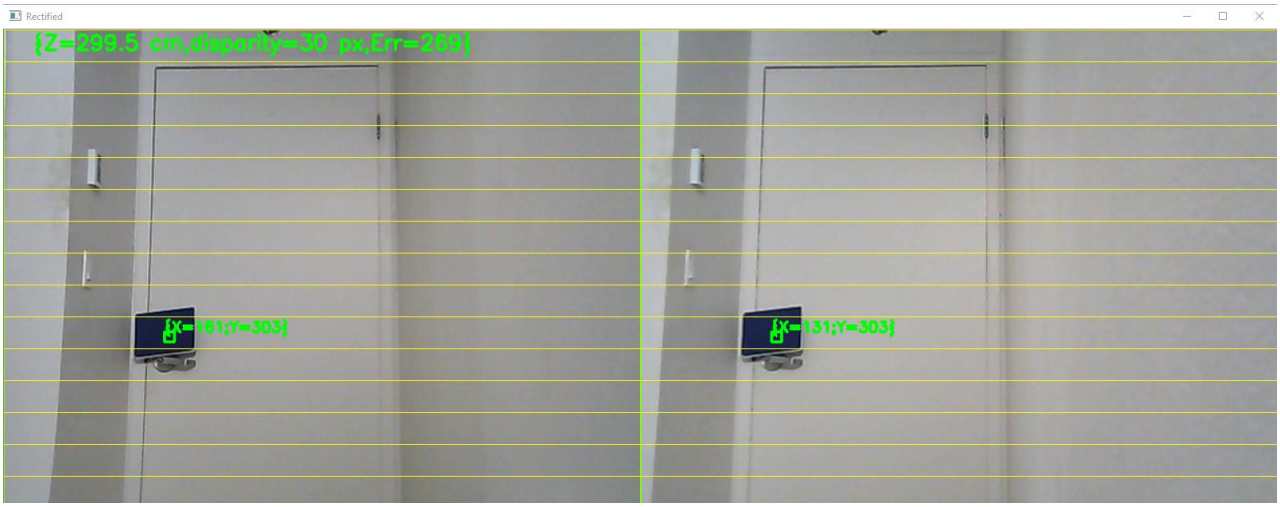
NB: Une bonne resolution de profondeur requiert **une grande distance entre les cameras T, un grande distance focale f et une petite profondeur Z**. Elle depends aussi de la precision d'estimation de la disparité qui est proportionnelle à la taille du pixel. Ainsi, un petit pixel permet une meilleur resolution.

Un exemple pour une profondeur d'objet fixé à Z= 80cm et disparité = (0..100), la pente de la courbe est de 0.738.



Finalement, pour obtenir la profondeur dans ce projet, on utilise la fonction OPENCV **cvPerspectiveTransform()** qui implémente l'équation suivante :

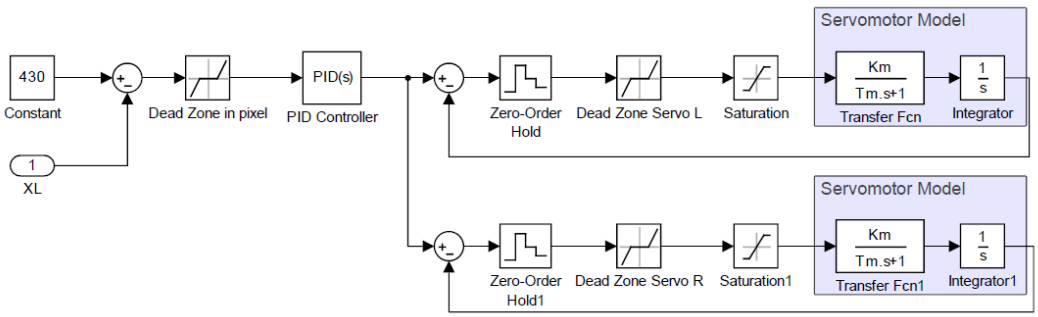
$$Q\begin{bmatrix} xL \\ yL \\ disparity \\ 1 \end{bmatrix} = \begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix}$$



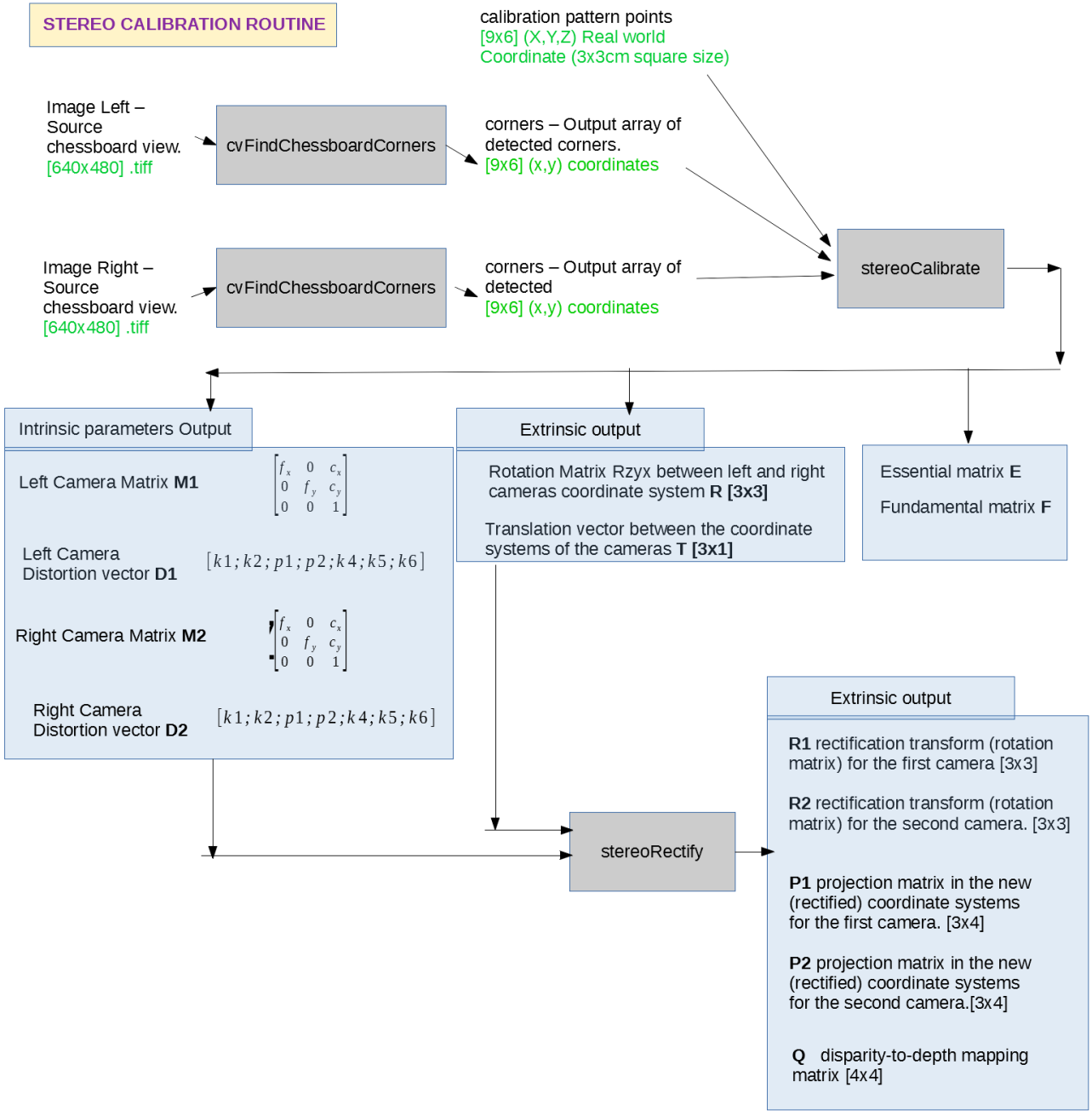
Le cahier bleu est bien à distance de 300cm vérifié avec un ruban métrique. L'objectif est atteint!

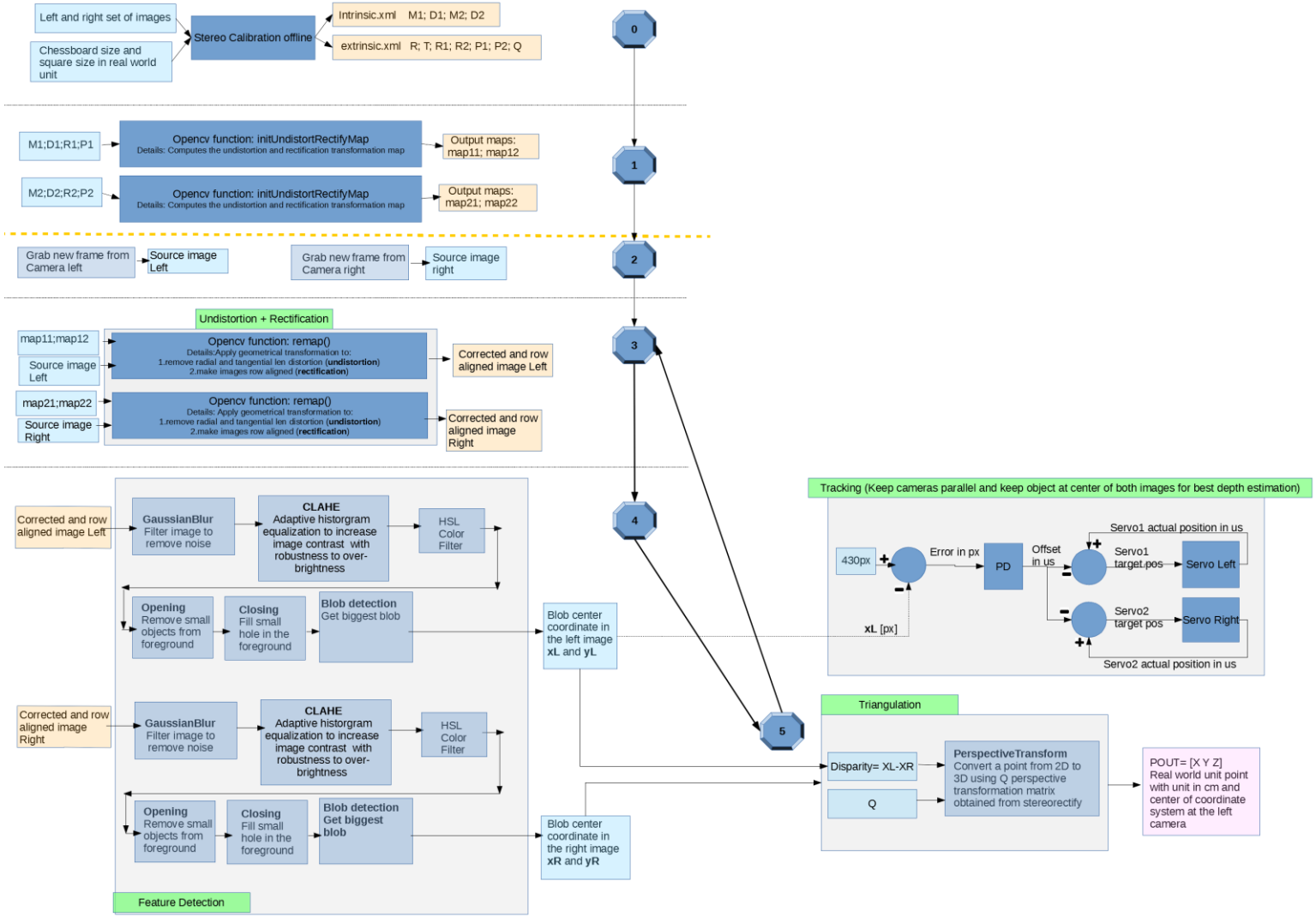
Algorithme de tracking :

Nous utilisons un algorithme de suivi avec un correcteur de type Proportionnel dérivé (PD). On essaye toujours d'avoir les caméras parallèle c'est pour cela qu'on envoie la même commande à la sortie du controleur. On note aussi qu'on a une zone morte de ±2pixel. Aussi on n'utilise pas d'intégrateur parce que le servomoteur a déjà un intégrateur 1/s dans la chaine directe du système. Le terme proportionnel est ce qui va nous permettre de corriger l'erreur de suivi le plus rapidement possible, le terme dérivé permet lui de réduire le dépassement. On a aussi une zone morte pour le servo qui est petite de l'ordre de 2us et une saturation de la valeur de PWM qui est entre 1000 et 2000us. On a choisi la valeur 430px parce qu'elle permet d'avoir la meilleur estimation de distance **en ayant l'objet le plus au centre dans les deux images** sachant qu'au bords de l'image on a plus d'aberration et donc moins de précision.



Conception logicielle :




Left and right set of images
Chessboard size and square size in real world unit
Stereo Calibration offline
Intrinsic.xml M1; D1; M2; D2
extrinsic.xml R; T; R1; R2; P1; P2; Q
0
M1;D1;R1;P1
Opencv function: initUndistortRectifyMap
Details: Computes the undistortion and rectification transformation map
Output maps: map11; map12
M2;D2;R2;P2
Opencv function: initUndistortRectifyMap
Details: Computes the undistortion and rectification transformation map
Output maps: map21; map22
1
Grab new frame from Camera left
Source image Left
Grab new frame from Camera right
Source image right
2
Undistortion + Rectification
map11;map12
Source image Left
Opencv function: remap()
Details:Apply geometrical transformation to:
1.remove radial and tangential len distortion (undistortion)
2.make images row aligned (rectification)
Corrected and row aligned image Left
map21;map22
Source image Right
Opencv function: remap()
Details: Apply geometrical transformation to:
1.remove radial and tangential len distortion (undistortion)
2.make images row aligned (rectification)
Corrected and row aligned image Right
3
4
Corrected and row aligned image Left
GaussianBlur
Filter image to remove noise
CLAHE
Adaptive historgram equalization to increase image contrast with robustness to over-brightness
HSL Color Filter
Opening
Remove small objects from foreground
Closing
Fill small hole in the foreground
Blob detection
Get biggest blob
Blob center coordinate in the left image xL and yL
Corrected and row aligned image Right
GaussianBlur
Filter image to remove noise
CLAHE
Adaptive historgram equalization to increase image contrast with robustness to over-brightness
HSL Color Filter
Opening
Remove small objects from foreground
Closing
Fill small hole in the foreground
Blob detection
Get biggest blob
Blob center coordinate in the right image xR and yR
Feature Detection
5
Tracking (Keep cameras parallel and keep object at center of both images for best depth estimation)
430px
Error in px
PD
Offset in us
Servo1 actual position in us
Servo1 target pos
Servo Left
Servo2 target pos
Servo Right
Servo2 actual position in us
xL [px]
Triangulation
Disparity= XL-XR
Q
PerspectiveTransform
Convert a point from 2D to 3D using Q perspective transformation matrix obtained from stereorectify
POUT= [X Y Z]
Real world unit point with unit in cm and center of coordinate system at the left camera

Futur améliorations et problématiques:

- ✓ **Si les paramètres intrinsèques sont connus mais les paramètres intrinsèques sont inconnus:**

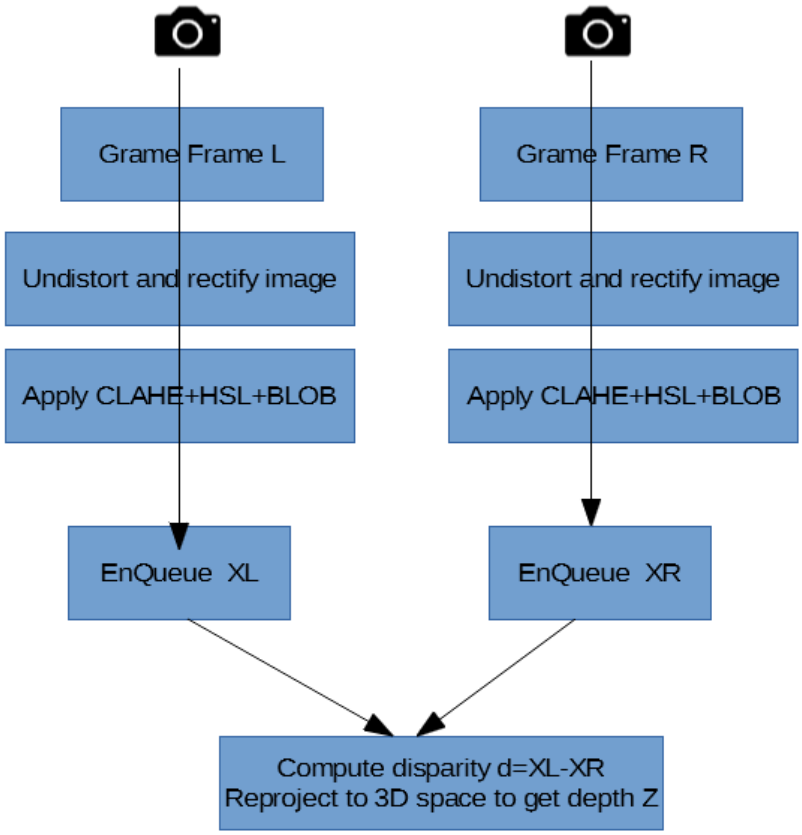
On peut toujours résoudre le problème de re-projection et les paramètres extrinsèques peuvent être estimé mais seulement jusqu'à un paramètre d'agrandissement qui reste indéterminé.

Puisque la distance entre les deux caméras est inconnues, la vrai échelle de la scène visionné ne peut pas être déterminé, la reconstruction se fait jusqu'à un facteur d'échelle. Mais ce facteur peut être déterminé si la distance réelle entre deux points observés dans la scène est connue à l'avance.

- ✓ **Limitation de l'algorithme de rectification d'OPENCV :**

La rectification stéréo d'une image dans OPENCV est possible seulement quand les épipoles sont en dehors du rectangle d'image. Ainsi, cet algorithme de rectification peut ne pas marcher avec des configurations stéréo qui sont caractérisé avec une très grande base ou lorsque les caméras pointent **trop** l'une vers l'autre.

- ✓ **Implémentation d'un algorithme multitâches avec la structure suivante pour améliorer les performances de l'algorithme mono tâche qui exécute tout en séquentiel.**



- ✓ **Recompiler OPENCV avec l'option CUDA de NVIDIA et Thread Building Block (TBB) d'INTEL pour améliorer les performances temps réels en délivrant chaque thread à des cœurs CUDA qui vont s'exécuter en parallèle.**
- ✓ **Tester la version avec caméras qui se croisent un peu.**
- ✓ **Augmenter la résolution de capture des flux vidéo pour améliorer la précision du calcul de disparité et donc la résolution du calcul de distance.**

Conclusion

Dans ce projet, on s'est initié à la libraire OPENCV et à la vision stéréoscopique. L'implémentation de l'algorithme de calcul de distance fut une occasion pour comprendre la géométrie épipolaire appliqué à la vision par ordinateur. Par ailleurs, l'incorporation d'une structure mécanique avec des vraies caméras a permis d'apprendre à utiliser OPENCV et VC++ avec des algorithmes temps réel.
Le calcul de distance peut être utilisé dans les environnements industriels pour équiper un bras robotisé qui va savoir la distance qui le sépare de l'objet d'intérêt et ainsi pouvoir agir en conséquence. On peut aussi l'utiliser pour mesurer des objets et les voir s'il respecte les côtes.

Il faut noter que j'ai testé l'algorithme de pattern matching qui n'est pas très robuste car si la caméra de droite détecte un pattern et l'envoi vers la caméra de gauche, cette dernière voie une version un peu de côté de l'image et donc le matching ne se fait plus…

J'ai aussi testé l'algorithme CAMSHIFT de OPENCV qui permet de tracker un objet d'une certaine couleur. Mais si l'objet sort du champ de vision, il faut absolument qu'il retourne de la même zone de l'image où il est sorti car l'algorithme cherche dans une certaine région localisé lorsqu'il perd l'objet d'intérêt.

NB : *Le code source pour la calibration stéréo et pour l'algorithme complet est joint en fichier .zip. Après calibration il faut mettre le fichier intrinsic.xml et extrinsic.xml dans le répertoire de l'algorithme complet.*

Références Bibliographique:

Bradski, Gary, and Adrian Kaehler. *Learning OpenCV: Computer vision with the OpenCV library*. " O'Reilly Media, Inc.", 2008.

http://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html

http://www.mathworks.com/help/vision/ug/camera-calibration.html

http://www.mathworks.com/discovery/stereo-vision.html

https://en.wikipedia.org/wiki/Epipolar_geometry

https://www.pololu.com/docs/0J40 | *https://www.pololu.com/product/1351*

http://www.futabarc.com/servos/digitalservos.pdf

ANNEXE :

Logiciels utilisés :

Plateforme de développement Visual Studio C++ 2012.

Libraries de vision : OPENCV 2.4.11 et cvblobslib_OpenCV_v83

Librairies de la carte servomoteur pour unmanaged C++: Protocol.h et Servo.h

IDE pour les servomoteurs : Maestro Control Center (Polulu)

PC portable: Intel Core i7 5700 @2.7 Ghz / 16GB RAM DDR3/ Windows 10 Home v1511 / GPU: NVidia GTX 970m .

Camera specifications:

AUSDOM AW615 USB Full HD 1080P Webcam Camera

Specifications:
1. Lens.Specification:
F/NO: 2.0
Focus: 30cm to infinity
Lens Material: PC-6100D1,4-layer film-coated glass lens group
Viewing Angle: Horizontal 65° Zoom: Yes

3. Hardware Specification:
Chipset:P269+MA1080
Control IC: P269(REAL TEK RTS5822)QFN46
Sensor: MA1080(5B3) CSP-481/4.5" CMOS "SAMSUNG"
Sensor size: 1/4.5" Sensor Resolution : 2 Mpx

4. Video/Image Specification:
Camera Resolution : 12Mpx
Window capture size: 4608*3456
Video resolution(video mode:YUY2): Max Resolution:1920*1080 5fps;
Min Resolution: 160*120 30fps
Default: 1280*720 8fps
Video resolution(video mode:MJPG): Max Resolution:1920*1080 30 fps
Min Resoluion:160*120 30fps
Default:1280*720 30fps
Photo resolution: 1920*1080
Photo Format: JPG

Specs Hitec D645mw:

Control System: +Pulse Width Control 1500usec Neutral
Required Pulse: 3-5 Volt Peak to Peak Square Wave
Operating Voltage Range: 3.5 - 8.4 Volts
Operating Temperature Range: -20° to +60° C (-4°F to +140°F)
Operating Speed (4.8V): **0.28 sec/60° at no load**
Operating Speed (6.0V): **0.20 sec/60° at no load**
Operating Speed (7.4V): **0.17 sec/60° at no load**
Stall Torque (4.8V): **115.3 oz/in. (8.3 kg.cm)**

Stall Torque (6.0V): 157.8 oz/in. (11.3 kg.cm)
Stall Torque (7.4V): 180.1 oz/in. (12.9 kg.cm)
Operating Angle: 45 Deg. one side pulse traveling 400usec
Continuous Rotation Modifiable: Yes
Direction: Clockwise/Pulse Traveling 1500 to 1900usec
Dead Band Width: 2usec
Motor Type: 3-Pole Ferrite
Bearing Type: Dual Ball Bearings
Gear Type: Metal Gears
Connector Wire Length: 11.8" (30mm)
Dimensions: 1.59" x 0.77" x 1.48" (40.6 x 19.8 x 37.8mm)
Weight: 2.11oz (60g)

Special Features:
- Wide Voltage Range: DC 3.5-8.4V
- 2 cell Li-Pi (6 cell Ni-MH) compatible
- Programmable digital amplifier with MOSFET drive
- Gold plated connector
- Narrow dead band width & high standing torque
- Applied H25T horn gear
- 3mm horn screw ('+' & HEX socket screw included)
- 32bit MCU / 12bit ADC