REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
الجمهورية الجزائرية الديمقراطية الشعبية

MINISTRY OF HIGHER EDUCATION
AND SCIENTIFIC RESEARCH

HIGHER SCHOOL IN APPLIED SCIENCES
--T L E M C E N--

وزارة التعليم العالي والبحث العلمي

المدرسة العليا في العلوم التطبيقية
-تلمسان-

المدرسة العليا في العلوم التطبيقية
École Supérieure en Sciences Appliquées

Mémoire de fin d'étude

Pour l'obtention du diplôme de Master

Filière : Automatique
Spécialité : Automatique

Présenté par : AISSANI Omar

Thème

Réalisation d'un robot SCARA pour des applications de Pick & Place à base de vision par ordinateur

Soutenu publiquement, le 29/09/2021, devant le jury composé de :

M. CHERKI Brahim	Professeur	ESSA. Tlemcen	Président
M. ARICHI Fayssal	Maître de conférences B		Directeur de mémoire
M. MOKHTARI Rida	Maître de conférences A		Co- Directeur de mémoire
M. ABDELLAOUI Ghouti	Maître de conférences B		Examinateur 1
M. BENSALAH Choukri	Maître de conférences B		Examinateur 2

Année universitaire : 2020/2021

Remerciements

Mes sincères remerciements vont au Professeur CHERKI Brahim pour avoir accepté d'être le président des exa-mineurs. Je suis sincèrement impressionné par ses connaissances et son humilité. Il faut avoir de la chance pour être un de ses étudiants. Je voudrais aussi remercier Dr.ABDELLAOUI Ghouti et Dr.BENSALAH Choukri pour les contributions précieuses à inclure dans cette thèse et ont généreusement passé un temps précieux en donnant les suggestions et les commentaires pour cette thèse.

Je voudrais remercier mes superviseurs Dr.ARICHI Fayssal, ainsi que DR.MOKHTARI Rida, pour leur patience, leurs conseils et leurs révisions.

Plus important encore, je suis reconnaissant pour l'amour et le soutien inconditionnel de mes parents. Je n'ai pas de mots pour reconnaître les sacrifices que vous avez faits, merci à Dieu de m'avoir donné vous.

Mes chères sœurs Ouassila et Hidayet. Vous avez toujours eu le don de me tirer d'un mauvais pas et de me redonner le sourire. Je suis si fière de vous deux. Je vous souhaite le meilleur

Abstract

In this thesis we were documenting the process of building a 4 degrees of freedom SCARA manipulator robot. We discussed the basics of computer vision and image processing, we were able to create a software that detects both the position and orientation of pillbox, which allows the robot to pick and place them in a desired position. We Created a software in unity3D game environment, and we spoke about trajectory generation problem.

Résumé

Le présent travail porte sur la réalisation d'un robot manipulateur de type SCARA à quatre degrés de liberté. L'objectif est de rendre le robot autonome à base de la vision par ordinateur afin qu'il puisse faire la tache de pick & place. On a créé un logiciel de visualisation dans l'environnement de jeux Unity3D. On a abordé un peu de la théorie de vision par ordinateur, et finalement on a travaillé sur le problème du génération des trajectoires.

Mots Clés

Robot Manipulateur SCARA, Arduino, OpenCV, Unity3D, Vision par ordinateur, Traitement des images, Génération des trajectoires, Modèle géométrique, Modèle cinématique.

Table des matières

Table des figures

Introduction Générale

Les robots manipulateurs sont parmi les outils les plus puissants présents au niveau de la fabrication industrielle. Ces robots apportent l'avantage d'être très précis et très robustes dans leurs tâches. Ils diffèrent selon leurs tailles, nombres d'articulations et les tâches à effectuer, ceci les rend une pièce centrale au niveau des usines et des grands sites de fabrication.

Souvent, il est nécessaire de réaliser des tâches qui sont en relation avec la nature des objets à manipuler, ceci peut être difficile à implémenter par les méthodes classiques dans le domaine de la robotique, car elles sont susceptibles à des erreurs qui peuvent engendrer des conséquences terribles. C'est pour ça qu'on s'est intéressé au niveau de ce travail à l'implémentation d'une plateforme à base de vision par ordinateur pour garantir le bon fonctionnement de ces robots. Ça fait partie du domaine de la robotique moderne, donc ça nous pousse à essayer de développer cette branche de la robotique et à tenter de la rendre vastement utilisée.

Afin de pouvoir effectuer des tâches basées sur la vision par ordinateur, un capteur à base de vision doit être introduit dans le processus de fonctionnement des robots. Ces capteurs peuvent être une webcam en 2D, un scanner Laser, une caméra qui construit les images en 3D (Kinect), etc. les données qui arrivent depuis ces capteurs seront analysées et traitées afin de pouvoir produire les commandes nécessaires aux robots. Ces capteurs sont des outils très utiles, et ils permettent d'introduire un nouvel aspect dans le monde de la robotique, leurs implémentations avec les outils et les API de l'intelligence artificielle les rends encore très puissants et leurs permet la réalisation de plusieurs tâches qui étaient une fois considérées impossibles.

Dans le cadre de ce mémoire, nous allons implémenter ces principes pour réaliser la commande visuelle du robot manipulateur SCARA. Ce mémoire est organisé comme suit :

Le premier chapitre est consacré à la citation des bases de la robotique et la modélisation du robot manipulateur de type SCARA et la génération de ses modèles géométriques directe et inverse.

Le deuxième chapitre aborde le sujet de planification des trajectoires, comme il ne suffit pas de dire au robot d'aller vers une position et une orientation donnée, mais il faut lui dicter aussi comment il doit l'approcher, sinon on risque d'avoir un mouvement non répétable et non prédictible. Nous allons montrer qu'il existe deux approches de génération des trajectoires, une dans l'espace articulaire, et l'autre dans l'espace de travail.

Le troisième chapitre est une initiation au domaine de la vision par ordinateur, en utilisant les outils de base du traitement d'images, comme la morphologie mathématiques, convolution 2D, etc. Et cela pour pouvoir écrire un programme qui permet au robot de détecter les positions et les orientations des boîtes de médicaments pour qu'il puisse les manipuler d'une façon autonome.

Dans le quatrième chapitre, Nous montrons les différentes procédures que nous avons suivi pour

désigner le robot, La partie mécanique par un logiciel de CAD. La fabrication des pièces par impression 3D, la programmation du micro-contrôleur, Les Moteurs utilisés pour actionner les articulations, Le design d'une carte de circuit imprimé. La création du software dans l'environnement de jeux Unity3D. La communications entre les processus via le protocole UDP. Le programme de vision par ordinateur.

Chapitre 1

Rappel sur la modélisation du robot manipulateur

L'étude de n'importe quel robot manipulateur nécessite une modélisation. Dans ce chapitre, on rappel le modèle géométrique direct et inverse ainsi que le modèle cinématique du bras robotique SCARA.

1.1 Modèle géométrique direct

Le Robot manipulateur SCARA est l'un des robots les plus simples à étudier, on peut extraire son modèle géométrique directe en utilisant une approches géométriques. Il peut être décomposé en deux parties, la translation suivant l'axe z, et un mouvement planaire contenant trois articulations rotoïdes, comme le montre la figure 1.1

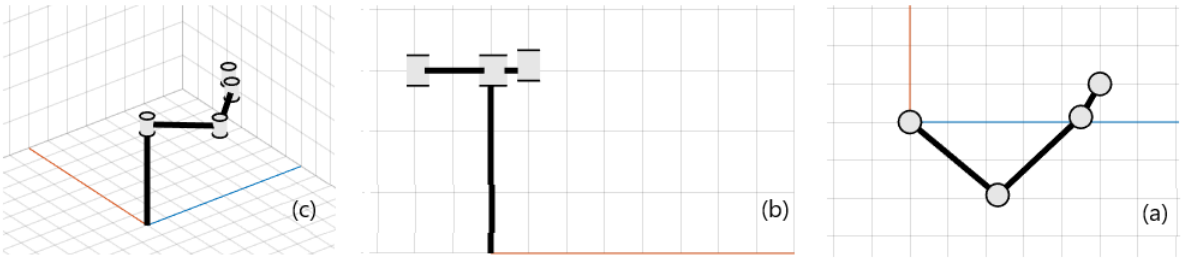

FIGURE 1.1 – Le diagramme du robot planar

Le modèle géométrique directe du robot SCARA est donné par :

$$\begin{cases} x = l_1 \cos(\theta_1) + l_2 \cos(\theta_1 + \theta_2) + l_3 \cos(\theta_1 + \theta_2 + \theta_3) \\ y = l_1 \sin(\theta_1) + l_2 \sin(\theta_1 + \theta_2) + l_3 \sin(\theta_1 + \theta_2 + \theta_3) \\ z = d_4 \\ \varphi = \theta_1 + \theta_2 + \theta_3 \end{cases} \tag{1.1}$$

1.2 Modèle géométrique inverse

C'est le problème le plus compliqué, il s'agit de trouver des valeurs pour les variables articulaires qui prennent le robot vers une position et orientation données. Le modèle géométrique inverse

du robot manipulateur SCARA est donné par :

$$\begin{cases} \theta_2 = \pm \arccos\left(\frac{\bar{x}^2+\bar{y}^2-l_1^2-l_2^2}{2l_1l_2}\right) \\ \theta_1 = \arctan2(\bar{y},\ \bar{x}) \mp \arctan2\left(l_2\sin(\theta_2),\ l_1 + l_2\cos(\theta_2)\right) \\ \theta_3 = \varphi - \theta_1 - \theta_2 \\ d_4 = z \end{cases} \tag{1.2}$$

Il existe deux solutions à ce problème, i.e pour les mêmes (x, y, z, φ) On a deux solutions, une appelée "Elbow Down", et l'autre "Elbow Up", comme le montre les figures 1.3 et 1.2.

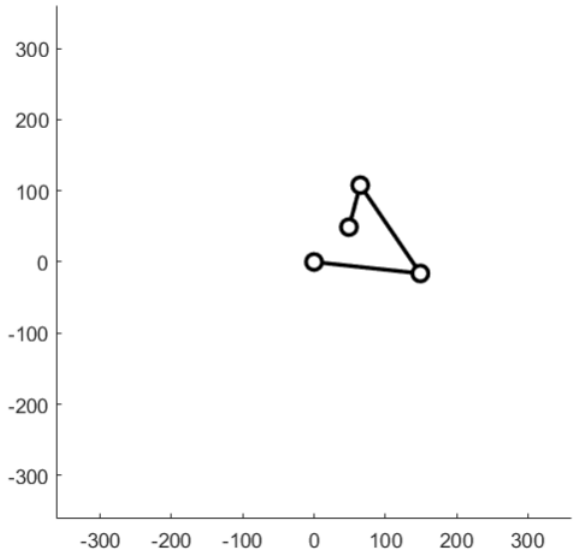

FIGURE 1.2 – Solution du MGI - Elbow Up

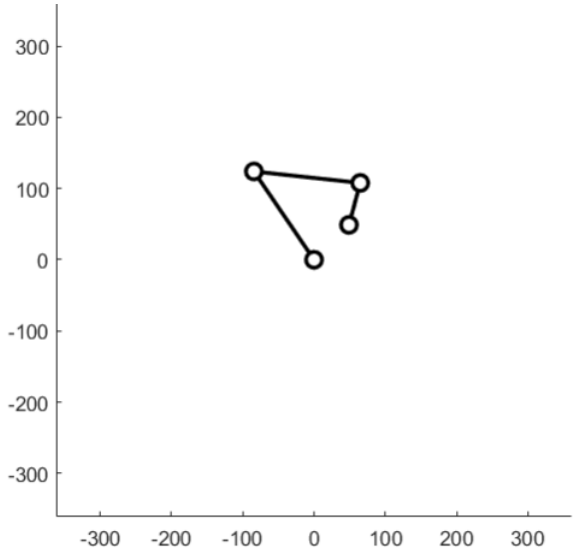

FIGURE 1.3 – Solution du MGI - Elbow Down

1.3 Modèle cinématique direct

Parfois on a besoin des vitesses opérationnelles en fonction des vitesses articulaires, cela est effectué à l'aide du modèle cinématique directe. Dans le cas du robot SCARA, on peut l'obtenir

par la dérivations des équations du modèle géométrique directe.
Le modèle cinématique directe du robot SCARA est le suivant :

$$\begin{cases} \dot{x} = -\dot{\theta_1}\left(l_1 s\theta_1 + l_2 s\theta_{12} + l_3 s\theta_{123}\right) - \dot{\theta_2}\left(l_2 s\theta_{12} + l_3 s\theta_{123}\right) - \dot{\theta_3}\left(l_3 s\theta_{123}\right) \\ \dot{y} = \dot{\theta_1}\left(l_1 c\theta_1 + l_2 c\theta_{12} + l_3 c\theta_{123}\right) + \dot{\theta_2}\left(l_2 c\theta_{12} + l_3 c\theta_{123}\right) + \dot{\theta_3}\left(l_3 c\theta_{123}\right) \\ \dot{z} = \dot{d_4} \\ \dot{\phi} = \dot{\theta_1} + \dot{\theta_2} + \dot{\theta_3} \end{cases} \tag{1.3}$$

On peut extraire la matrice Jacobienne à partir de 1.3. On aura donc :

$$J_{4\times4}(q) = \begin{bmatrix} 0 & -l_1 s\theta_1 - l_2 s\theta_{12} - l_3 s\theta_{123} & l_2 s\theta_{12} - l_3 s\theta_{123} & -l_3 s\theta_{123} \\ 0 & l_1 c\theta_1 + l_2 c\theta_{12} + l_3 c\theta_{123} & l_2 c\theta_{12} + l_3 c\theta_{123} & l_3 c\theta_{123} \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix} \tag{1.4}$$

On peut écrire 1.3 sous forme matricielle par l'équation :

$$\xi = J_{4\times4}(q)\,\dot{q} \tag{1.5}$$

Ainsi,

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\phi} \end{pmatrix} = \begin{bmatrix} 0 & -l_1 s\theta_1 - l_2 s\theta_{12} - l_3 s\theta_{123} & l_2 s\theta_{12} - l_3 s\theta_{123} & -l_3 s\theta_{123} \\ 0 & l_1 c\theta_1 + l_2 c\theta_{12} + l_3 c\theta_{123} & l_2 c\theta_{12} + l_3 c\theta_{123} & l_3 c\theta_{123} \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix} \begin{pmatrix} \dot{d_4} \\ \dot{\theta_1} \\ \dot{\theta_2} \\ \dot{\theta_3} \end{pmatrix} \tag{1.6}$$

NB : l'écriture θ_{ijk} veut dire $\theta_i + \theta_j + \theta_k$ et vice versa.

1.4 Modèle cinématique inverse

C'est le modèle qui permet d'obtenir les vitesses articulaires en fonction des vitesses opérationnelles. On peut l'obtenir en inversant la matrice Jacobienne 1.4, On aura par conséquent

$$\begin{pmatrix} \dot{d_4} \\ \dot{\theta_1} \\ \dot{\theta_2} \\ \dot{\theta_3} \end{pmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ \dfrac{c\theta_{12}}{l_1 s\theta_2} & \dfrac{s\theta_{12}}{l_1 s\theta_2} & 0 & \dfrac{l_3 s\theta_3}{l_1 s\theta_2} \\ \dfrac{-l_2 c\theta_{12}^{-} - l_1 c\theta_1}{l_1 l_2 s\theta_2} & \dfrac{-l_2 s\theta_{12}^{-} - l_1 s\theta_1}{l_1 l_2 s\theta_2} & 0 & -l_3 \dfrac{l_1 s\theta_{23} + l_2 s\theta_3}{l_1 l_2 s\theta_2} \\ \dfrac{c\theta_1}{l_2 s\theta_2} & \dfrac{s\theta_1}{l_2 s\theta_2} & 0 & \dfrac{l_3 s\theta_{23} + l_2 s\theta_2}{l_2 s\theta_2} \end{bmatrix} \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \omega_z \end{pmatrix} \tag{1.7}$$

1.5 L'espace opérationnel

Appelé aussi espace cartésien, il est défini par la position et l'orientation de l'organe terminal. L'obtention de l'espace de travail des robots manipulateurs n'est pas facile.

Une manière de visualiser l'espace de travail du robot SCARA est décrite par le script suivant :

```
A1 = 150; A2 = 150; A3 = 61;
xStep = 10; yStep = 10; thetaStep = 5;
xyLim = L1+L2+L3;
X = –xyLim: xStep: xyLim;
```

```
Y = -xyLim: yStep: xyLim;
THETA = -180: thetaStep: 180;
% Joint Constraints:
q1m = -20; q1M = 180+20;
q2m = -90-20; q2M = 90+20;
q3m = -90-20; q3M = 90+20;

zero_ = zeros(length(X), length(Y));
isWS_ElbowDown = zero_;
isWS_ElbowUp = zero_;

%% Check for Workspace
for i=1:length(X)
    for j=1:length(Y)
        for k=1:length(THETA)
            x = X(i); y = Y(j); theta = THETA(k);

            X_ = x-A3*cosd(theta);
            Y_ = y-A3*sind(theta);
            A = (X_^2+Y_^2-A1^2-A2^2)/(2*A1*A2);

            if(A>1 || A<-1) % if there is no solution
               isWS_ElbowDown(i, j) = isWS_ElbowDown(i, j) - 1;
               isWS_ElbowUp(i, j) = isWS_ElbowUp(i, j) - 1;
            else
                % Elbow Down:
                q2 = -acosd(A);
                q1 = atan2d(Y_, X_) - atan2d(A2*sind(q2), A1+A2*cosd(q2));
                q3 = wrapTo180(theta-q1-q2);
                if(q1>q1M || q1<q1m || q2>q2M || q2<q2m || q3>q3M || q3<q3m)
                    isWS_ElbowDown(i, j) = isWS_ElbowDown(i, j) - 1;
                end

                % Elbow Up:
                q2 = acosd(A);
                q1 = atan2d(Y_, X_) - atan2d(A2*sind(q2), A1+A2*cosd(q2));
                q3 = wrapTo180(theta-q1-q2);
                if(q1>q1M || q1<q1m || q2>q2M || q2<q2m || q3>q3M || q3<q3m)
                    isWS_ElbowUp(i, j) = isWS_ElbowUp(i, j) - 1;
                end
            end
        end
    end
end
```

L'espace de travail du robot SCARA sans contrainte sur les articulation est donné par la figure 1.4.
Un robot réel possède généralement des contraintes physiques sur les articulation peut être des limites des actionneurs ou pour éviter que le robot entre en collision avec lui même.
Si on ajoute des contraintes sur les articulations, On remarque que l'espace de travail obtenu par la solution qui correspond à la configuration "Elbow Down" et celui de la configuration

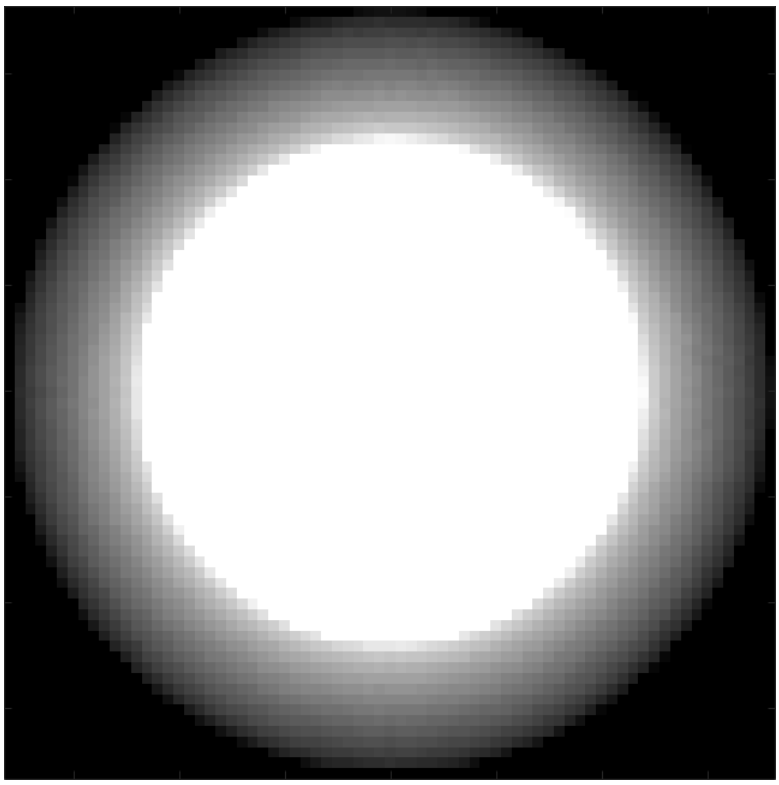

FIGURE 1.4 – L'espace de travail du robot SCARA sans contraintes sur les articulations

"Elbow Up" ne sont plus les mêmes ! On aura donc les espaces de travail suivants illustrés dans 1.5 1.6 :

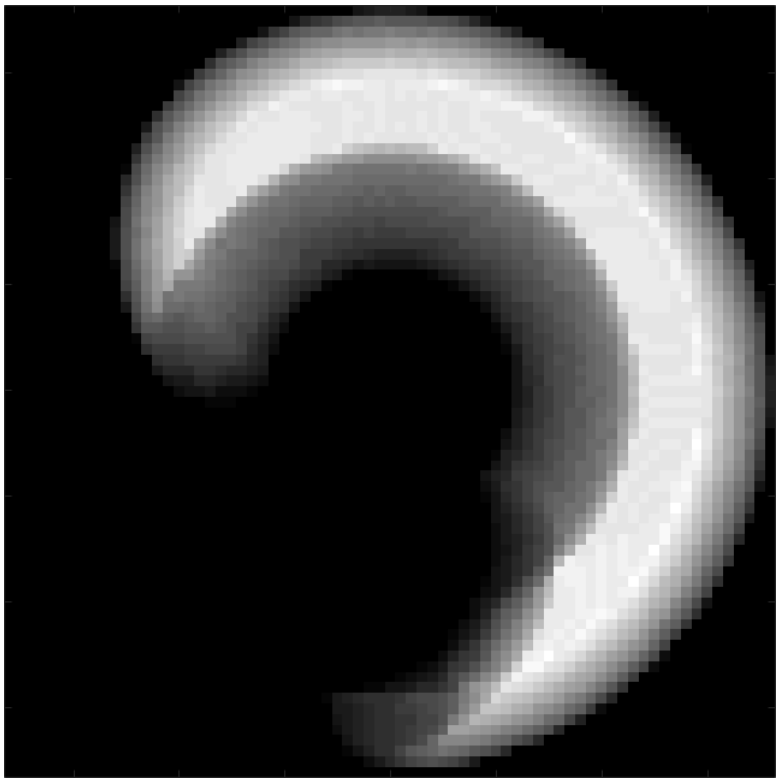

FIGURE 1.5 – L'espace de travail du robot SCARA avec contraintes - Elbow Down

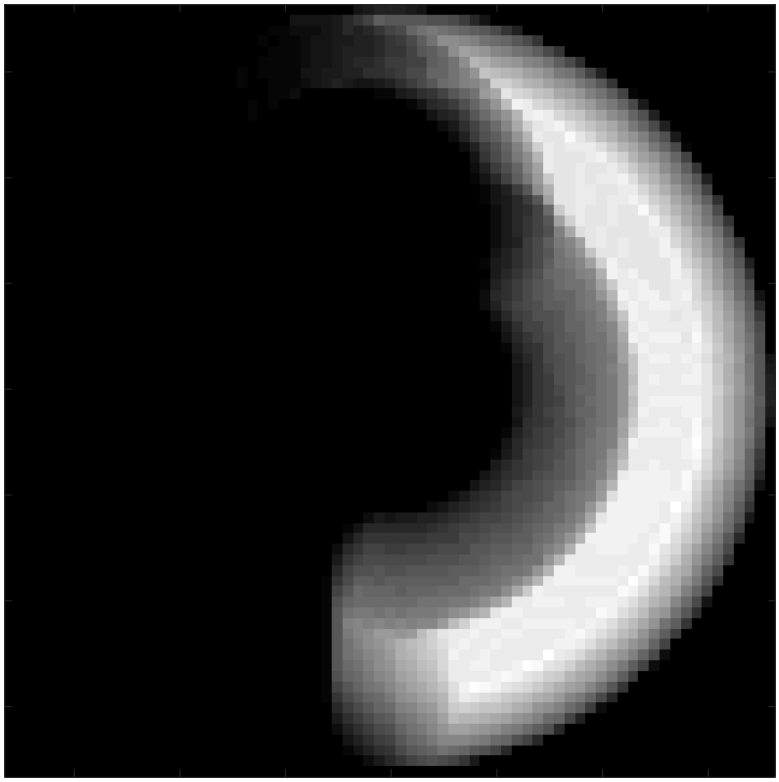

FIGURE 1.6 – L'espace de travail du robot SCARA avec contraintes - Elbow Up

Chapitre 2

Planification des trajectoires

2.1 Introduction

Les robots manipulateurs ont en général des différents tâches à faire comme le Pick & Place, Soudure, Peinture, etc. La différence entre ces tâches réside dans la trajectoire suivi par l'organe terminale. On doit donc s'intéresser au problème de génération des chemins et trajectoires.
Un chemin est un ensemble de point dans l'espace. En revanche, une trajectoire est une représentation temporelle et spatiale, elle englobe des informations comme la position, la vitesse, l'accélération, etc.

On peut générer des trajectoires dans l'espace articulaire, comme on peut les générer dans l'espace Cartésien. Chacune de ces deux approches possède des bienfaits et des inconvénients. Une trajectoire dans l'espace articulaire est toujours régulière et lisse, mais elle n'est pas prédictible dans l'espace opérationnel. Une trajectoire dans l'espace cartésien est prédictible. mais elle est plus lente dans son exécution comme elle résout le problème du modèle géométrique inverse qui est souvent non-linéaire chaque itération, les mouvement des articulations n'est pas lisse.
Dans ce chapitre on s'intéresse au génération de trajectoire dans l'espace articulaire.

2.2 Profils de mouvement de base

2.2.1 Les trajectoires polynômiales

Dans le cas le plus simple, le mouvement est définie par poser un temps initial t_0 et un temps final t_1, une position, vitesse, et accélération initiales q_0, v_0, et a_0 respectivement et finales q_1, v_1, et a_1. Il s'agit donc de trouver une fonction $q\colon t \to [t_0, t_1]$ qui satisfait les conditions.
Une parmi d'autre solution à ce problème et l'utilisation d'un polynôme :

$$q(t) = a_0 + a_1 t + \cdots + a_n t^n \tag{2.1}$$

Les coefficients a_i sont choisis de tel sorte que les conditions initiales et finales sont satisfaites. Le degré de p dépend directement du nombre de conditions à satisfaire, mais aussi de la régularité (smoothness) du mouvement.
Le nombre de conditions est souvent pair, il en résulte donc des polynômes du degré impaire $3, 5, 7, ...$ En plus des conditions finales en terme de trajectoires, on peut aussi spécifier des conditions sur ses dérivées par rapport au temps (vitesse, accélération, l'à-coup, etc). A un instant t_j quelconque $\in [t_0, t_1]$, en d'autres termes, on cherche donc à déterminer une fonction polynômiale dont la dérivée Kème dans l'instant t_j est égale à une valeur spécifique $q^{(k)}(t_j)$:

$$q^{(k)}(t_j) = k!\ a_k + (k+1)!\ a_{k+1} t_j + ... + \frac{n!}{(n-k)!} a_n t_j^{n-k} \tag{2.2}$$

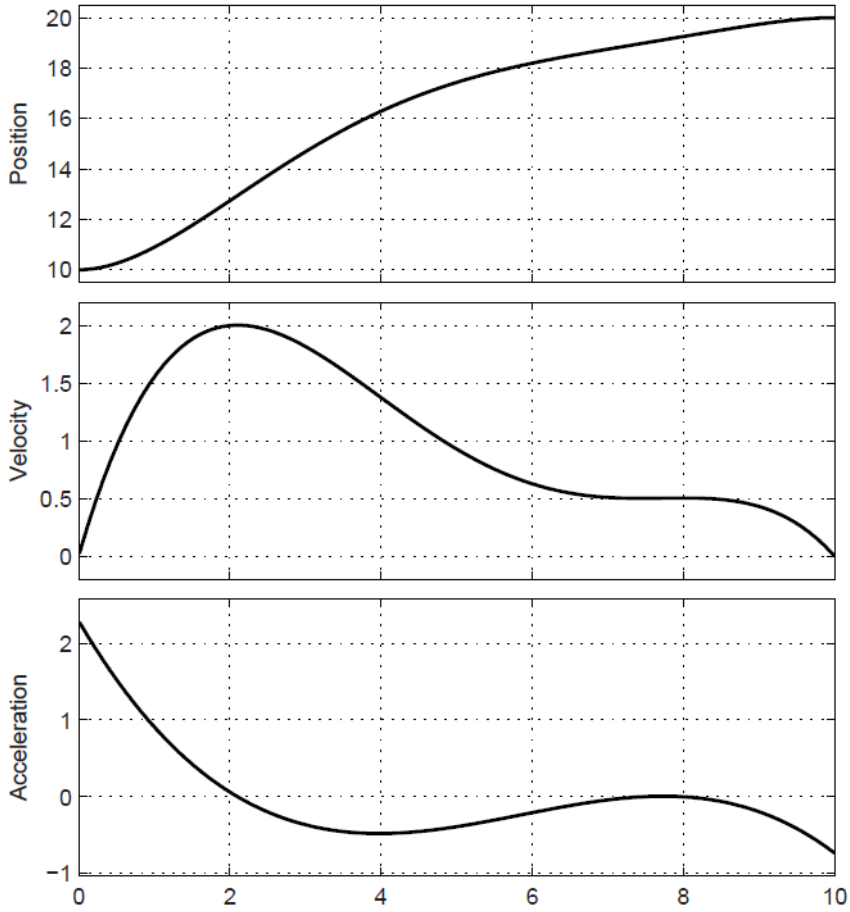

FIGURE 2.1 – Trajectoire polynômiale qui vérifie les conditions 2.4

Ou sous forme matricielle :

$$Ma = b \tag{2.3}$$

avec M est une matrice carré de taille $(n+1)$, b contient les $n+1$ conditions à satisfaire, et a est le vecteur de coefficients $a = \begin{bmatrix} a_0 & a_1 & a_n \end{bmatrix}^T$. On peut avoir une solution à ce problème en inversant la matrice M et calculer le vecteur a tel que : $a = M^{-1}b$, par contre pour de grandes valeur de n ou dans le cas où la matrice M est mal conditionnée, cette procédure peut causer des problèmes numériques.

Exemple : La figure 2.1 illustre la position, vitesse et l'accélération d'une trajectoire polynômiale qui a été calculée suivant ces contraintes :

$$\begin{array}{llll} t_0 = 0, & t_1 = 10, & q_0 = 10, & q_1 = 20, \\ v_0 = 0, & v_1 = 0, & v(t=2) = 2, & a(t=8) = 0. \end{array} \tag{2.4}$$

Pour ce problème, il existe 4 conditions aux limites et 2 conditions intermédiaires, pour ce nombre de conditions on doit choisir un polynôme de degré supérieur ou égal à 5 :

$$q(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5 \tag{2.5}$$

La vitesse est donc obtenu en dérivant le polynôme de position par rapport au temps :

$$v(t) = \dot{q}(t) = a_1 + 2a_2 t + 3a_3 t^2 + 4a_4 t^3 + 5a_5 t^4 \tag{2.6}$$

De la même façon on obtient le polynôme d'accélération :

$$a(t) = \ddot{q}(t) = 2a_2 + 6a_3 t + 12a_4 t^2 + 20a_5 t^3 \tag{2.7}$$

En remplace par les conditions qu'on a fixé.

$$\begin{cases} q(0) = a_0 = 10 \\ q(1) = a_0 + a_1 + a_2 + a_3 + a_4 + a_5 = 20 \\ v(0) = a_1 = 0 \\ v(1) = a_1 + 2a_2 + 3a_3 + 4a_4 + 5a_5 = 0 \\ v(t=2) = a_1 + 4a_2 + 12a_3 + 32a_4 + 80a_5 = 2 \\ a(t=8) = 2a_2 + 48a_3 + 768a_4 + 10240a_5 = 0 \end{cases} \tag{2.8}$$

Finalement, on trouve :

$$\begin{aligned} a_0 = 10, \quad & a1_{=}0, \quad & a_2 = 1,1462, \\ a_3 = -0.2806 \quad & a_4 = 0.0267, \quad & a_5 = -0.0009. \end{aligned}$$

2.2.2 Trajectoire linéaire

La trajectoire la plus simple qui peut décrire le mouvement entre un point initial q_0 et un point final q_1 est définie par :

$$q(t) = a_0 + a_1(t - t_0) \tag{2.9}$$

Une fois on fixe les instants t_1 et t_0 et les positions q_0 et q_1, les coefficients a_0 et a_1 peuvent être déterminées en résolvant le système matriciel suivant :

$$\begin{cases} q(t_0) = q_0 = a_0 \\ q(t_1) = q_1 = a_0 + a_1(t_1 - t_0) \end{cases} \tag{2.10}$$

Ou sous forme matricielle :

$$\begin{bmatrix} 1 & 0 \\ 1 & T \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} = \begin{bmatrix} q_0 \\ q_1 \end{bmatrix} \tag{2.11}$$

avec $T = t_1 - t_0$ signifiant la durée du mouvement. La solution du système précédent nous donne les coefficients suivants :

$$\begin{cases} a_0 = q_0 \\ a_1 = \dfrac{q_1 - q_0}{a_1 - a_0} = \dfrac{h}{T} \end{cases} \tag{2.12}$$

h étant le déplacement qui est donné par $h = q_1 - q_0$. La vitesse est donc constante et est égale à $a_1 = \dfrac{h}{T}$ et l'accélération est nulle dans l'intérieur de l'intervalle de temps et impulsive dans les extrémités, c'est pour cette raison que ce type de trajectoire linéaire n'est pas trop utilisé dans les robots industriels.
Exemple : La figure 2.2 représente la position, la vitesse et l'accélération d'une trajectoire linéaire qui satisfait les conditions : $t_0 = 0$, $t_1 = 8$, $q_0 = 0$, $q_1 = 10$.

2.2.3 Trajectoire parabolique

Appelée aussi "gravitational trajectory" ou aussi trajectoire à accélération constante, comme son nom l'indique, elle est caractérisée par une accélération constante en valeur absolue et par signe opposé dans la phase d'accélération et de décélération. Elle est la composition de deux polynômes de degré deux, un allant de t_0 à t_p et l'autre de t_p à t_1, t_p étant le point d'inflexion (voir figure 2.3).

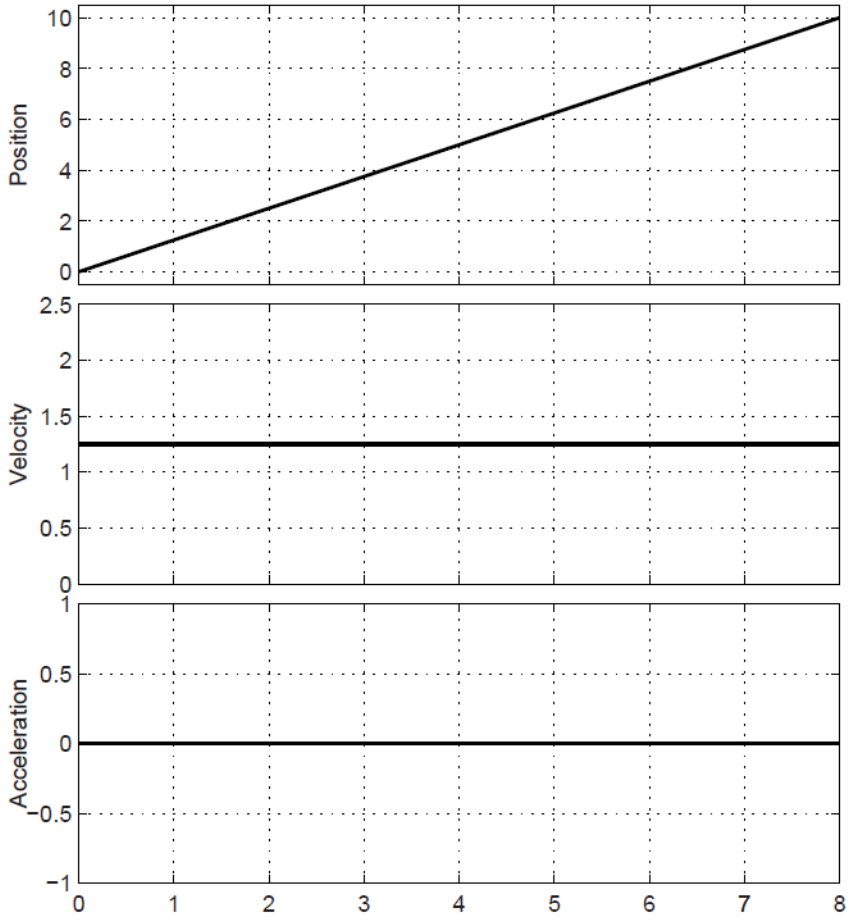

FIGURE 2.2 – Trajectoire linéaire

Considérons maintenant le cas d'une trajectoire symétrique par rapport à son milieu, ceci revient à poser $t_p = \frac{t_1 + t_0}{2}$ et $q(t_p) = q_p = \frac{q_1 + q_0}{2}$, notons que $T_a = t_p - t_0 = \frac{T}{2}$ et $q_p - q_0 = \frac{h}{2}$
Dans la phase d'accélération, la trajectoire est définie par un polynôme de degré 2 appelé $q_a(t)$:

$$q_a(t) = a_0 + a_1(t - t_0) + a_2(t - t_0)^2 \tag{2.13}$$

pour $t \in [t_0, t_p[$.
Les coefficients a_0, a_1 et a_2 peuvent être calculer en posant des conditions sur la trajectoire dans les points q_0 et q_p et une condition sur la vitesse initiale v_0 :

$$\begin{cases} q_a(t_0) = q_0 = a_0 \\ q_a(t_p) = q_p = a_0 + a_1(t_p - t_0) + a_2(t_p - t_0)^2 \\ \dot{q}_a(t_0) = v_0 = a_1 \end{cases} \tag{2.14}$$

Les coefficients du polynôme trouvé après la résolution du système d'équation précédent sont :

$$\begin{cases} a_0 = q_0 \\ a_1 = v_0 \\ a_2 = \dfrac{4(q_p - q_i) - 2v_0T}{T^2} = \dfrac{2}{T^2}(h - v_0T) \end{cases} \tag{2.15}$$

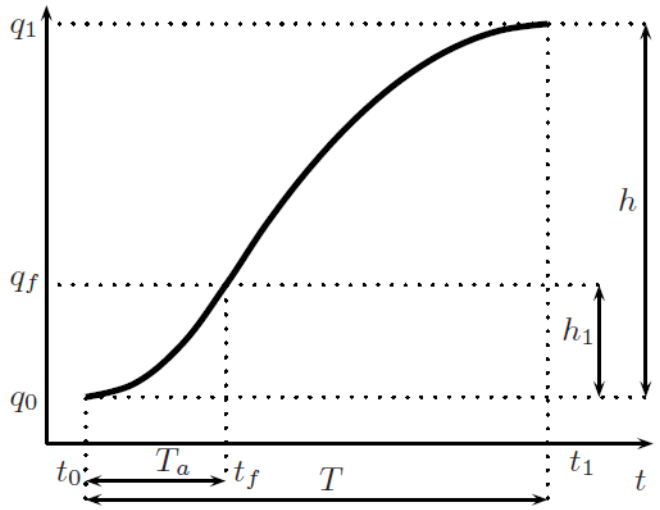

FIGURE 2.3 – Trajectoire à accélération constante par segment

avec $h = q_p - q_i$. Et donc pour $t \in [t_0, t_1]$, la trajectoire est définie par :

$$\begin{cases} q_a(t) = q_0 + v_0(t - t_0) + \dfrac{2}{T^2}(h - v_o T)(t - t_0)^2 \\ \dot{q}_a(t) = v_0 + \dfrac{4}{T^2}(h - v_0 T)(t - t_0) \\ \ddot{q}_a(t) = \dfrac{4}{T^2}(h - v_0 T) \end{cases} \tag{2.16}$$

La vitesse dans le point d'inflexion est donnée par :

$$v_p = v_{max} = \dot{q}_a(t_p) = \frac{2h}{T} - v_0 \tag{2.17}$$

Remarque

Si $v_0 = 0$, la vitesse maximale est deux fois plus grande que la vitesse d'une trajectoire linéaire

$$v_{max} = \frac{2h}{T} \tag{2.18}$$

Dans la deuxième phase (phase de décélération), la trajectoire est donnée par la relation suivante :

$$q_b(t) = a_3 + a_4(t - t_p) + a_5(t - t_p)^2 \tag{2.19}$$

pour $t \in [t_p, t_1]$
Les paramètres a_3, a_4, et a_5 sont obtenues par :

$$\begin{cases} q_b(t_p) = q_p = a_3 \\ q_b(t_1) = q_1 = a_3 + a_4(t_1 - t_p) + a_5(t_1 - t_p)^2 \\ \dot{q}_a(t_1) = v_1 = a_4 + 2a_5(t_1 - t_p) \end{cases} \tag{2.20}$$

Ce qui nous donne :

$$\begin{cases} a_3 = q_p \\ a_4 = \dfrac{2h}{T} - v_1 \\ a_5 = \dfrac{2}{T^2}(v_1 T - h) \end{cases} \tag{2.21}$$

L'expression de la trajectoire pour $t \in [t_0, t_1]$ est donnée par :

$$\begin{cases} q_b(t) = q_p + (\dfrac{2h}{T} - v_1)(t - t_p) + \dfrac{2}{T^2}(v_1 T - h)(t - t_p)^2 \\ \dot{q}_b(t) = \dfrac{2h}{T} - v_1 \dfrac{4}{T^2}(v_1 T - h)(t - t_p) \\ \ddot{q}_b(t) = \dfrac{4}{T^2}(v_1 T - h) \end{cases} \tag{2.22}$$

Notons que si $v_0 \neq v_1$, le profile de vitesse sera discontinu en $t = t_p$
Exemple : La figure 2.4 représente la position, la vitesse et l'accélération pour une trajectoire parabolique qui vérifier les conditions : $t_0 = 0$, $t_1 = 8$, $q_0 = 0$, $q_1 = 10$, $v_0 = v_1 = 0$

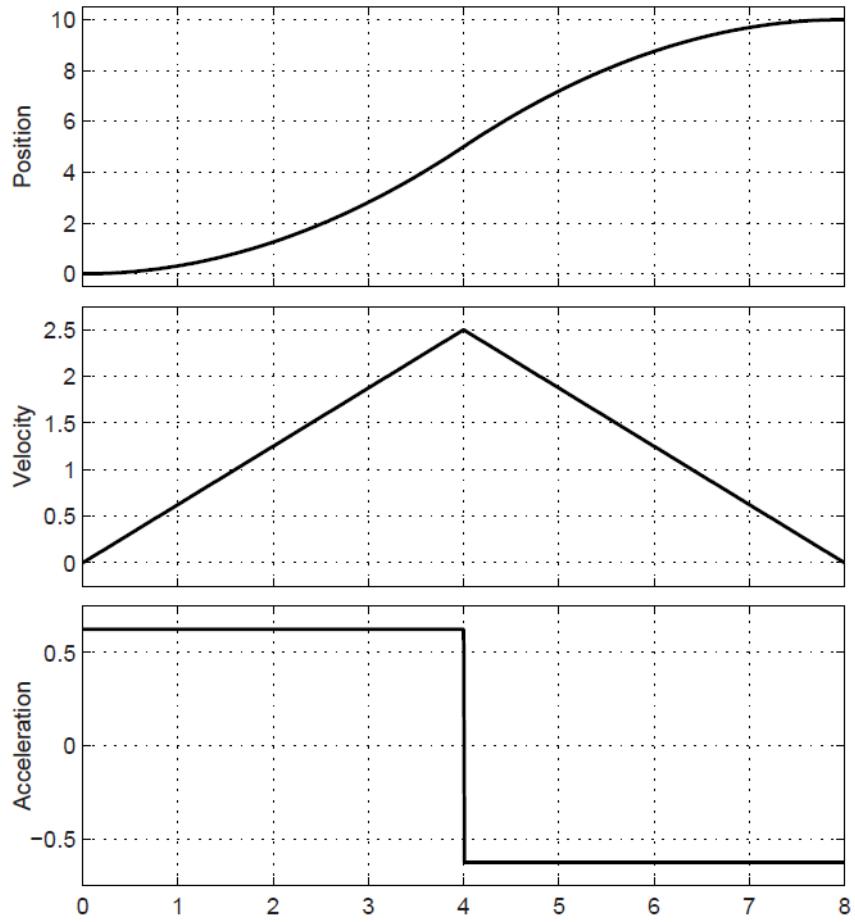

FIGURE 2.4 – Trajectoire parabolique à accélération symétrique et constante

Dans le cas où la contrainte $q(t_p) = q_p = \dfrac{q_0 + q_1}{2}$ n'est pas exigée, on détermine les coefficients a_i de façon que la vitesse soit continue en $t = t_p$ *i.e* $\dot{q}_a(t_p) = \dot{q}_b(t_p)$. Les conditions seront donc :

$$\begin{cases} q_a(t_0) = a_0 = q_0 \\ \dot{q}_a(t_0) = a_1 = v_0 \\ q_b(t_1) = a_3 + a_4 \dfrac{T}{2} + a_5 (\dfrac{T}{2})^2 = q_1 \\ \dot{q}_b(t_1) = a_4 + 2a_5 \dfrac{T}{2} = v_1 \\ q_a(t_p) = a_0 + a_1 \dfrac{T}{2} + a_2 (\dfrac{T}{2})^2 = a_3 = q_b(t_p) \\ \dot{q}_a(t_p) = a_1 + 2a_2 \dfrac{T}{2} = a_4 = \dot{q}_b(t_p) \end{cases} \tag{2.23}$$

avec $\frac{T}{2} = t_p - t_0 = t_1 - t_p$. On obtient donc :

$$\begin{cases} a_0 = q_0 \\ a_1 = v_0 \\ a_2 = \dfrac{4h - T(3v_0 - v_1)}{2T^2} \\ a_3 = \dfrac{4(q_0 + q_1) + T(v_0 - v_1)}{8} \\ a_4 = \dfrac{4h - T(v_0 + v_1)}{2T} \\ a_5 = \dfrac{-4h + T(v_0 + 3v_1)}{2T^2} \end{cases} \tag{2.24}$$

Exemple : La figure 2.5 représente la position, la vitesse et l'accélération pour cette trajectoire, les conditions exigées sont : $t_0 = 0$, $t_1 = 8$, $q_0 = 0$, $q_1 = 10$, $v_0 = 0.1$, $v_1 = -1$.

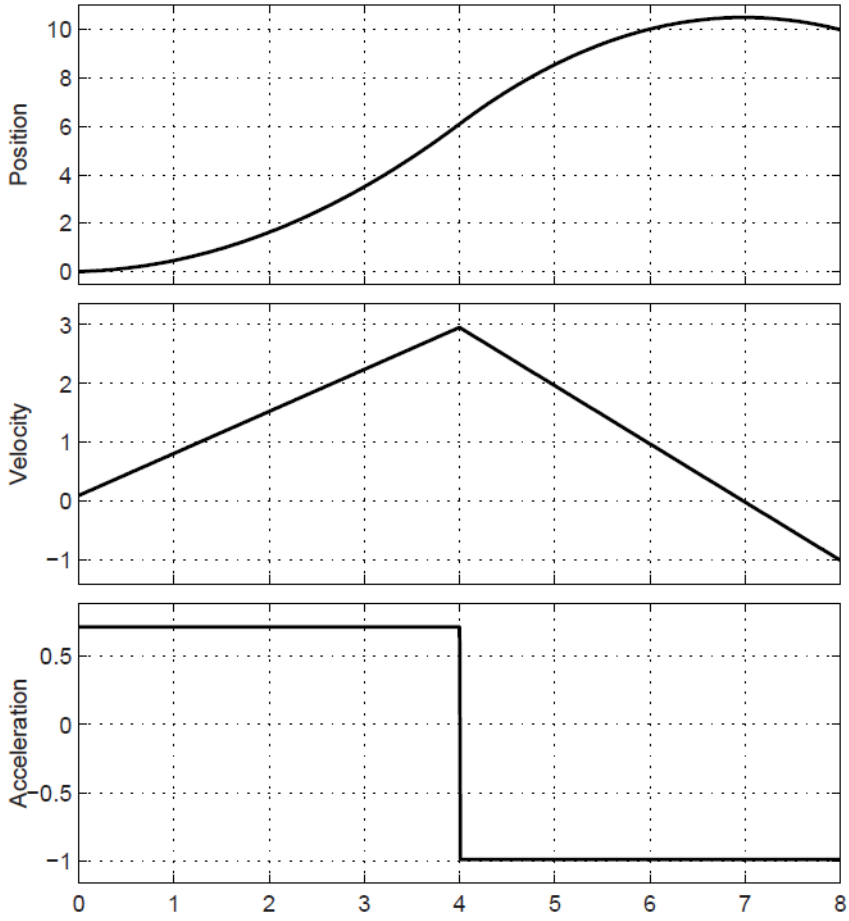

FIGURE 2.5 – Trajectoire parabolique ayant une vitesse finale non nulle

2.2.4 Trajectoire ayant une accélération asymétrique

Cette trajectoire est le cas général de la trajectoire précédente, on peut l'obtenir en posant le point d'inflexion sur un instant quelconque $t_p \in]t_0, t_1[$ et pas forcément au milieu $i.e\ t_p = \dfrac{t_0 + t_1}{2}$. Cette trajectoire est définie par deux polynômes :

$$q_a(t) = a_0 + a_1(t - t_0) + a_2(t - t_0)^2 \tag{2.25}$$

pour $t \in [t_0, t_p]$, et :

$$q_b(t) = a_3 + a_4(t - t_p) + a_5(t - t_p)^2 \tag{2.26}$$

pour $t \in [t_p, t_1]$. Les paramètres a_i seront obtenus en posant les contraintes suivantes :

$$\begin{cases} q_a(t_0) = a_0 = q_0 \\ \dot{q}_a(t_0) = a_1 = v_0 \\ q_b(t_1) = a_3 + a_4(t_1 - t_p) + a_5(t_1 - t_p)^2 = q_1 \\ \dot{q}_b(t_1) = a_4 + 2a_5(t_1 - t_p) = v_1 \\ q_a(t_p) = a_0 + a_1(t_p - t_0) + a_2((t_p - t_0))^2 = a_3 = q_b(t_p) \\ \dot{q}_a(t_p) = a_1 + 2a_2(t_p - t_0) = a_4 = \dot{q}_b(t_p) \end{cases} \tag{2.27}$$

On définit $T_a = (t_p - t_0)$, et $T_d = (t_1 - t_p)$. Il en résulte :

$$\begin{cases} a_0 = q_0 \\ a_1 = v_0 \\ a_2 = \dfrac{2h - v_0(T + T_a) - v_1 T_d}{2TT_a} \\ a_3 = \dfrac{2q_1 T_a + T_d(2q_0 + T_a)(v_0 - v_1)}{2T} \\ a_4 = \dfrac{2h - v_o T_a - v_1 T_d}{T} \\ a_5 = \dfrac{2h - v_0 T_a - v_1(T + T_d)}{2TT_d} \end{cases} \tag{2.28}$$

Le vitesse et l'accélération pour $t \in [t_0, t_p[$:

$$\begin{cases} \dot{q}_a = a_1 + 2a_2(t - t_0) = v_0 + \dfrac{2h - v_0(T + T_a) - v_1 T_d}{TT_a}(t - t_0) \\ \ddot{q}_a = 2a_2 = \dfrac{2h - v_0(T + T_a) - v_1 T_d}{TT_a} \\ \dot{q}_b = a_4 + 2a_5(t - t_p) = \dfrac{2h - v_o T_a - v_1 T_d}{T} + \dfrac{2h - v_0 T_a - v_1(T + T_d)}{TT_d}(t - t_p) \\ \ddot{q}_b = \dfrac{2h - v_0 T_a - v_1(T + T_d)}{TT_d} \end{cases} \tag{2.29}$$

Remarque

Lorsque $v_0 = v_1 = 0$, la vitesse maximale est égales à la même que celle d'une trajectoire symétrique au point d'inflexion :

$$v_{max} = \dot{q}_a(t_p) = \frac{2h}{T} \tag{2.30}$$

Il est clair que si $t = \dfrac{t_0 + t_1}{2}$, la trajectoire précédent sera obtenue.

Exemple : La figure 2.6 représente la trajectoire à accélération constante non symétrique qui vérifie les contraintes : $t_0 = 0$, $t_1 = 8$, $q_0 = 0$, $q_1 = 10$, $v_0 = v_1 = 0$.

2.2.5 Trajectoire Cubique

Dans le cas où on spécifie les positions et les vitesses aux instants t_0 et t_1 (q_0, q_1, v_0, et v_1), on doit choisir un polynôme de degré 3. Comme il existe 4 contraintes :

$$q_b(t) = a_0 + a_1(t - t_0) + a_2(t - t_0)^2 + a_3(t - t0)^3 \tag{2.31}$$

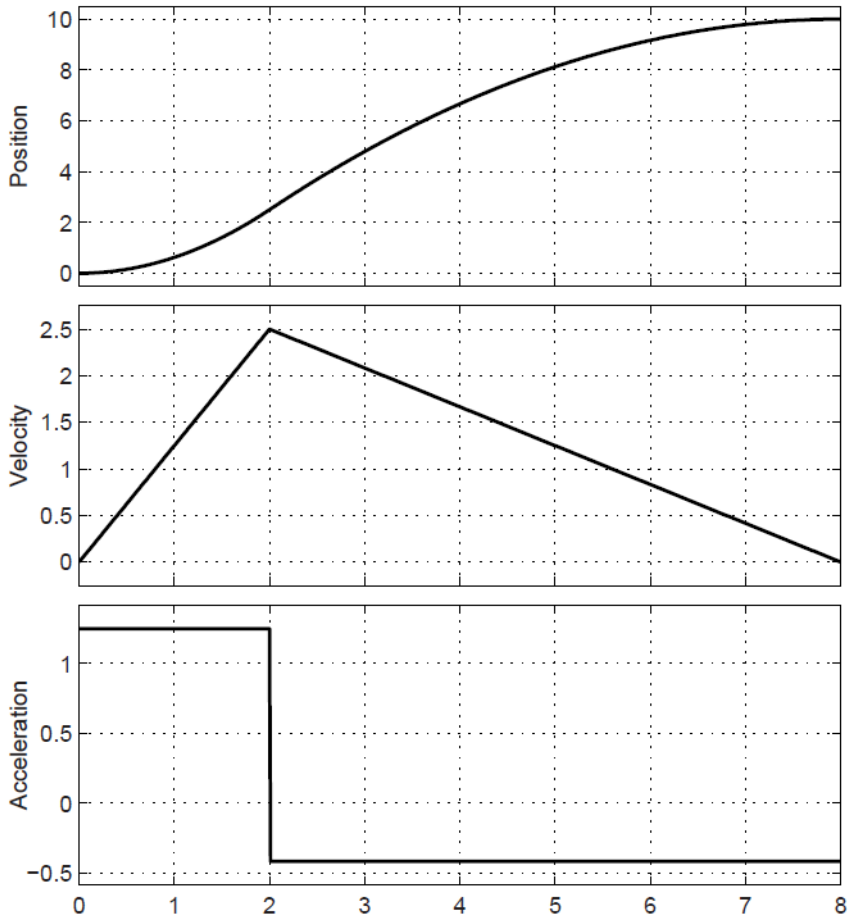

FIGURE 2.6 – Trajectoire parabolique à accélération non symétrique

Les paramètres a_i s'écrient comme suivant :

$$\begin{cases} a_0 = q_0 \\ a_1 = v_0 \\ a_2 = \dfrac{3h - (2v_0 + v_1)T}{T^2} \\ a_3 = \dfrac{-2h + T(v_0 + v_1)}{T^3} \end{cases} \tag{2.32}$$

On peut exploiter le résultat suivant pour calculer une trajectoire ayant une vitesse continue dans une séquence de n points, le mouvement global est subdivisé en $n-1$ segments, chacun de ces segments connecte les points q_k avec q_{k+1} sur l'intervalle t_k,t_{k+1}. Les équations 2.32 sont utilisées pour calculer les 4(n-1) paramètres, qui sont : a_{0k}, a_{1k}, a_{2k}, et a_{3k}

Exemple :
La figure 2.7(a) illustre la position, la vitesse et l'accélération pour cette trajectoire avec comme contraintes : $q_0 = 0$, $q_1 = 1$, $t_0 = 0$, $t_1 = 8$, $v_0 = 0$, $v_1 = 0$. La figure 2.7(b) représente la position, la vitesse et l'accélération pour cette trajectoire avec comme contraintes : $q_0 = 0$, $q_1 = 1$, $t_0 = 0$, $t_1 = 8$, $v_0 = -5$, $v_1 = -10$

Exemple :
La figure 2.8 représente la trajectoire, la vitesse et l'accélération pour une trajectoire multi-point

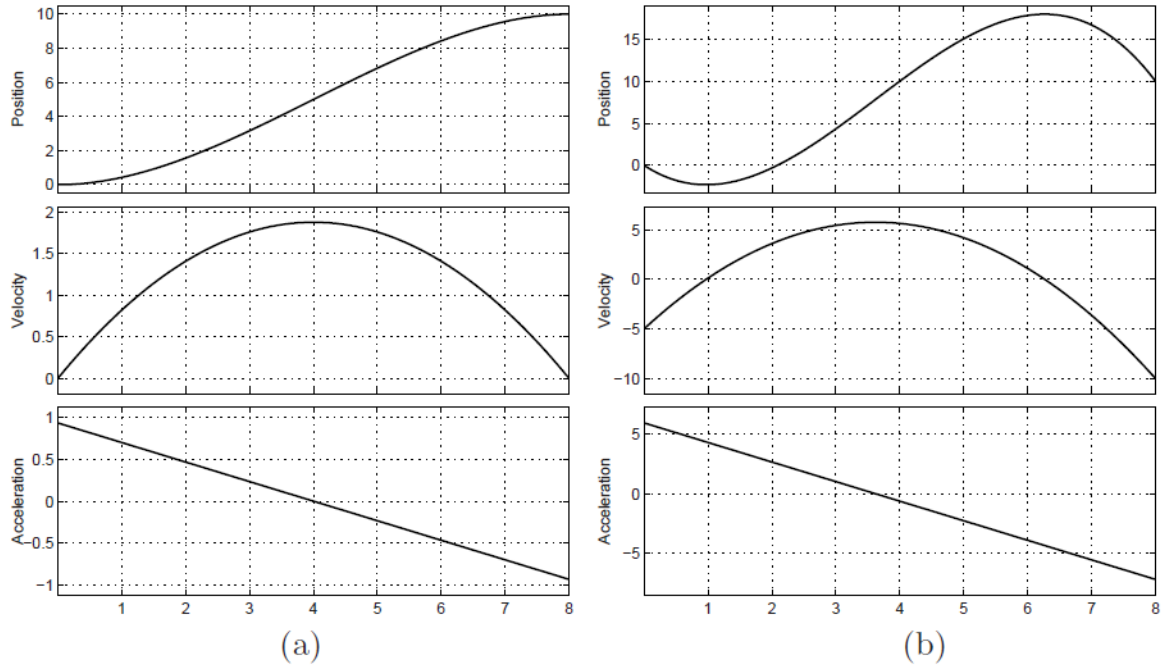

FIGURE 2.7 – Trajectoire cubique

avec :

$$t_0 = 0, \quad t1 = 2, \quad t_2 = 4, t_3 = 8, t_4 = 10, \tag{2.33}$$

$$q_0 = 0, \quad q1 = 2, \quad q_2 = 4, q_3 = 8, q_4 = 10, \tag{2.34}$$

$$t_0 = 0, \quad v1 = 2, \quad v_2 = 4, v_3 = 8, v_4 = 10,$$

Parfois on ne définit pas des vitesses dans les points intermédiaire, dans ce cas on doit calculer des vitesses adéquates à l'aide des règles heuristiques, une parmi d'autres est :

$$\begin{aligned} & v_0 \quad (defini) \\ & v_k = \begin{cases} 0, & \text{si } \operatorname{sign}(d_k) \neq \operatorname{sign}(d_{k+1}) \\ \frac{1}{2}(d_{k+1} + d_k), & \text{si } \operatorname{sign}(d_k) = \operatorname{sign}(d_{k+1}) \end{cases} \\ & v_n \quad (defini) \end{aligned} \tag{2.35}$$

d_k étant la pente du segment entre les instants t_{k-1} et t_k *i.e* $d_k = \dfrac{q_k - q_{k-1}}{t_k - t_{k-1}}$

Exemple : La figure 2.9 représente la position, la vitesse et l'accélération d'une trajectoire multi-points, On garde les contraintes 2.33 mais cette fois-ci les vitesses intermédiaires sont calculées par la règle heuristique 2.35

2.2.6 Trajectoire Polynomiale de degré 5

On a vu précédemment qu'une trajectoire qui passe par $q_0, q_1, ..., q_n$ basée sur un polynôme de degré 3 est caractérisée par une trajectoire continue, et un profil de vitesse continu, par contre, l'accélération subit en général des discontinuités dans t_k, (voir figure 2.9).
Cette trajectoire peut être suffisamment "smooth", mais parfois elle cause des problèmes lorsque le bras manipulateur à une élasticité relativement importante, dans ce cas on aura par conséquent des vibrations, chose qui peut être dangereuse.

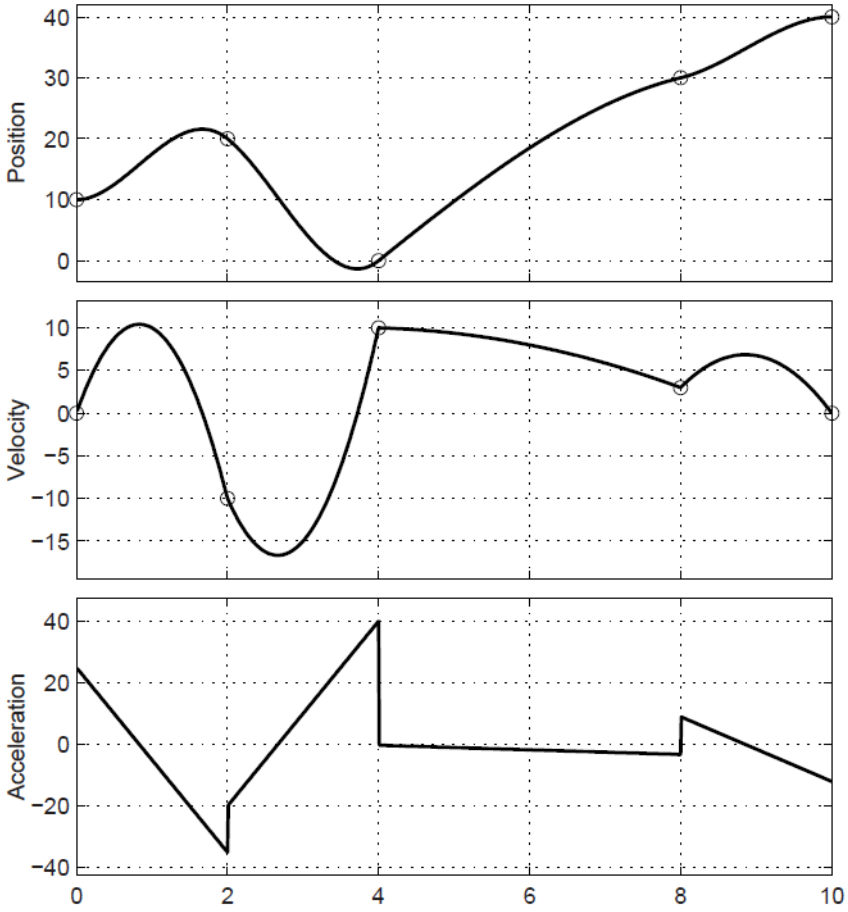

FIGURE 2.8 – Trajectoire multi-points

Pour remédier à ce problème, on souhaite que l'accélération soit continue, cela se traduit mathématiquement par l'ajout des conditions initiales et finales sur l'accélération, on aura donc 6 conditions au total, ce qui justifie le choix d'un polynôme de degré 5 :

$$q(t) = a_0 + a_1\tau + a_2\tau^2 + a_3\tau^3 + a_4\tau^4 + a_5\tau^5 \tag{2.36}$$

avec $\tau = (t - t_0)$. Les conditions seront donc :

$$\begin{cases} q(t_0) = q_0 \\ \dot{q}(t_0) = v_0 \\ \ddot{q}(t_0) = a_0 \\ q(t_1) = q_1 \\ \dot{q}(t_1) = v_1 \\ \ddot{q}(t_1) = a_1 \end{cases} \tag{2.37}$$

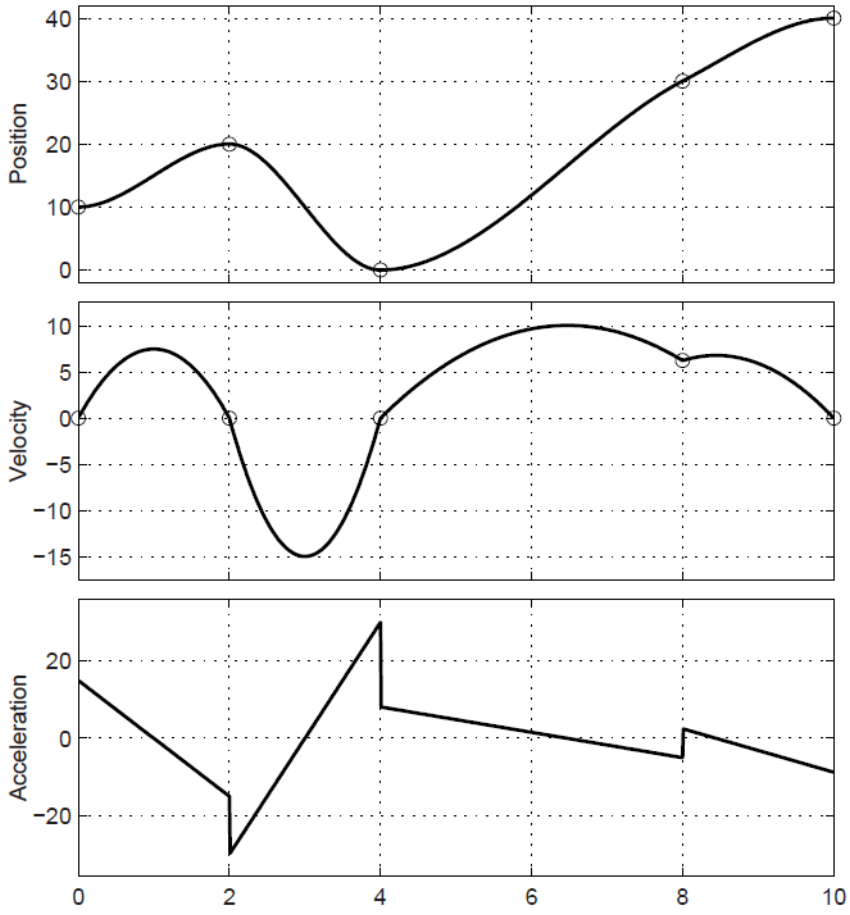

FIGURE 2.9 – Trajectoire multi-points dont les vitesses sont calculées par une règle heuristique

Après avoir résolu le système d'équations précédent, on aura :

$$\begin{cases} a_0 = q_0 \\ a_1 = v_0 \\ a_2 = \dfrac{1}{2}a_0 \\ a_3 = \dfrac{1}{2T^3}[2h - (8v_1 + 12v_0)T - (3a_0 - a_1)T^2] \\ a_4 = \dfrac{1}{2T^4}[-30h + (14v_1 + 16v_0)T + (a_0 - 2a_1)T^2] \\ a_5 = \dfrac{1}{2T^5}[12h - 6(v_1 + v_0)T + (a_1 - a_0)T^2] \end{cases} \tag{2.38}$$

Exemple : Deux trajectoires de degré 5 sont illustrées dans la figure 2.10.
La trajectoire dans (a) a été calculée en posant ces contraintes : $t_0 = 0$, $t_1 = 8$, $q_0 = 0$, $q_1 = 10$, $v_0 = v_1 = 0$, $a_0 = a_1 = 0$.
La trajectoire dans (b) a les mêmes contraintes sauf pour les vitesses initiales et finales, $v_0 = -5$ et $v_1 = -10$.

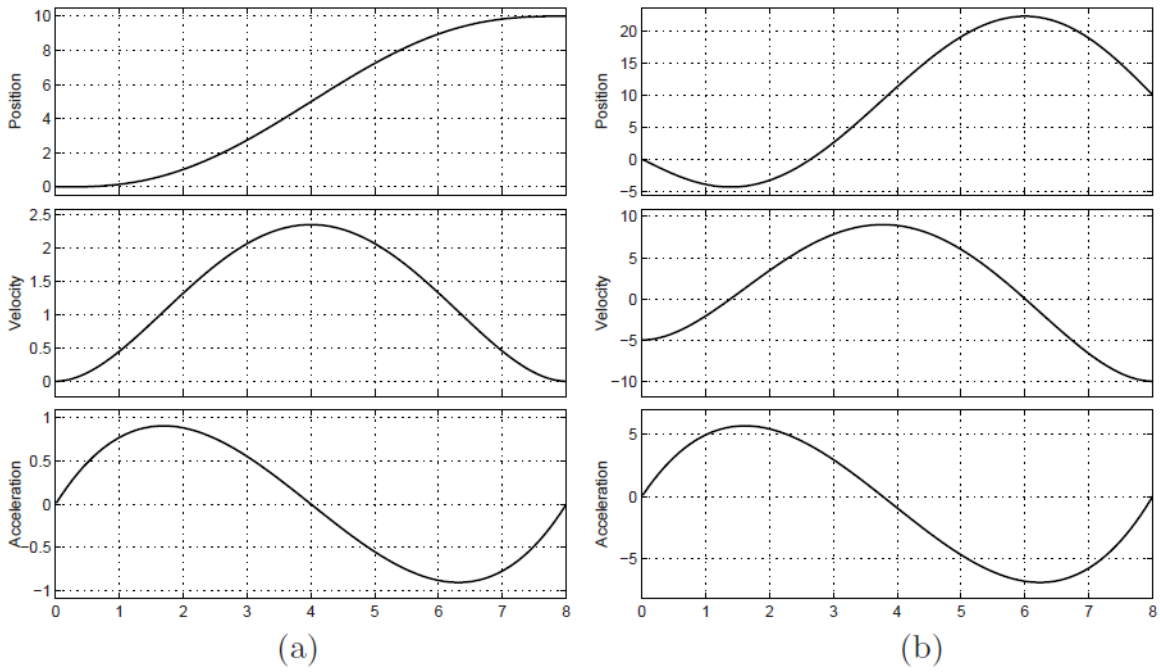

FIGURE 2.10 – Trajectoires polynômiales de degré 5

Chapitre 3

Vision par Ordinateur

3.1 Introduction

3.2 Vision par ordinateur vs traitement d'image

Le traitement d'image est la science qui s'intéresse à améliorer une image en réglant plusieurs arguments (luminance, contraste, etc). Et donc des transformations seront appliquées sur l'image comme le lissage, détection de contour, etc. Les transformations utilisée dépendent du contexte et du problème à régler.

La vision par ordinateur en revanche est la science qui s'intéresse à "comprendre" une image et d'en extraire des informations. La vision par ordinateur utilise des algorithmes du traitement d'image pour arriver à atteindre son objectif principal.

La différence entre ces deux approches donc s'agit de l'objectif à atteindre et non pas les méthodes utilisées. Si le but est d'améliorer une photo pour une utilisation ultérieure, alors on peut dire que c'est un problème du traitement d'image. Et si on souhaite imiter la vision humaine pour faire des taches tels que la reconnaissance des formes, reconnaissance des défauts, etc. Alors on peut dire qu'il s'agit d'une problème de vision par ordinateur.

3.3 Domaine d'application de la vision par ordinateur

La vision par ordinateur est maintenant utilisée dans de nombreux domaines et a des applications dans le monde réel, entre autres :

Reconnaissance optique de caractères ou encore océrisation du mot Anglais Optical Character Recognition (OCR). Ça consiste à reconnaître un texte (imprimé ou écrit par la main) dans une image et le traduire en un fichier texte. Une des exemples de cette application est la détection automatique des immatriculation des véhicules (*automatic number plate recognition - ANPR*).

Inspections des machines qui utilise de la vision par ordinateur pour détecter et analyser les défauts, afin d'assurer la qualité requise. Exemple de mesure des tolérances dans les ailes d'avion en utilisant la stéréovision avec un éclairage particulier.

Automatisation des usines On donne l'exemple du tri automatique des objets sur un convoyeur.

Imagerie médicale, L'utilisation extensives des images médicales pour aider à faire une meilleur diagnistique des tumeurs par exemple.

Sports pour aider à la prise de décision, on donne l'exemple de la *Goal-Line technology* utilisé dans le football, et aussi le *Hawk-Eye technology* dans le tennis.

Aide à la conduite rendre l'expérience de conduire une voiture beaucoup plus agréable et surtout plus sécurisée grâce aux systèmes de vision notamment celui de l'alerte de franchissement involontaire de ligne, détection de panneau de signalisation, etc.

Divertissement tel que dans les jeux interactifs utilisant la kinect. Et aussi dans le domaine du cinéma pour générer les effets spéciaux (CGI)

Photogrammétrie qui est une technique qui consiste à effectuer des mesures dans une scène, en utilisant la parallaxe obtenue entre des images acquises selon des points de vue différents. Recopiant la vision stéréoscopique humaine afin de reconstituer une copie 3D exacte de la réalité.

Capture de mouvement parfois abrégé mocup, il s'agit de capturer le mouvement d'un objet ou d'une personne à l'aide des caméras infrarouges et des marqueurs passifs réfléchissants, elle est utilisée dans le domaine de l'animation 3D et pour faire l'étape de validation dans la robotique.

Authentification biométrie ce qui signifie l'utilisation de mesures biométrique pour identifier les personnes, comme dans la reconnaissance des empreintes digitales et aussi la reconnaissance de l'iris.

Exploration spatiale, La vision par ordinateur est utilisée d'une façon monstrueuse dans les rovers comme curiosity et perseverance qui cherchent une forme de vie dans d'autres planètes.

3.4 Qu'est ce qu'une image numérique ?

L'appellation d'image numérique désigne toute image (dessin, icône, photographie...) acquise, créée, traitée et stockée sous forme binaire

- Acquise par des convertisseurs analogique-numérique situés dans des dispositifs comme les scanners, les appareils photo.
- Créée directement par des programmes informatiques, grâce à une souris, des tablettes graphiques ou par de la modélisation 3D.
- Traitée grâce à des outils graphiques, de façon à la transformer, à en modifier la taille, les couleurs, d'y ajouter ou d'en supprimer des éléments, d'y appliquer des filtres variés, etc.
- Stockée sur un support informatique.

3.5 Types d'images

3.5.1 Images matricielles

Elle est composée d'une matrice (tableau) de points à plusieurs dimensions, chaque dimension représentant une dimension spatiale (hauteur, largeur, profondeur), temporelle (durée).

Images 2D
Dans le cas des images 2D, les points sont appelés pixels. D'un point de vu mathématique, une image est une fonction de $\mathbb{R} \times \mathbb{R}$ dans $\mathbb{R}$ où le couplet d'entrée est considéré comme une position spatiale, le singleton de sortie comme un codage.
Images 2D + t, images 3D, images multi-résolution
Ces cas sont une généralisation du cas 2D, la dimension supplémentaire représentant respectivement le temps, une dimension spatiale ou une échelle de résolution. D'un point de vue mathématique, il s'agit d'une fonction de $\mathbb{R} \times \mathbb{R} \times \mathbb{R}$ dans $\mathbb{R}$

— Lorsqu'une image possède une composante temporelle, on parle d'animation.

— Dans le cas des images à trois dimensions, les points sont appelés des *voxels*. Ils représentent un volume.

3.5.2 Images vectorielles

Le principe est de représenter les données de l'image par des formules géométriques qui vont pouvoir être décrites d'un point de vue mathématique. Cela signifie qu'au lieu de mémoriser une mosaïque de points élémentaires, on stocke la succession d'opérations conduisant au tracé. Par exemple, un dessin peut être mémorisé par l'ordinateur comme une droite tracée entre les points (x_1, y_1) et (x_2, y_2), puis un cercle tracé de centre (x_3, y_3) et de rayon 30 de couleur rouge.

L'avantage de ce type d'image est la possibilité de l'agrandir indéfiniment sans perdre la qualité initiale, ainsi qu'un faible encombrement. L'usage de prédilection de ce type d'images concerne les schémas qu'il est possible de générer avec certains logiciels de DAO (Dessin Assisté par Ordinateur) comme AutoCAD ou CATIA. Ce type d'images est aussi utilisé pour les animations Flash, utilisées sur Internet pour la création de bannières publicitaires, l'introduction de sites web, voire des sites web complets.

Étant donné que les moyens de visualisation d'images actuels comme les écrans d'ordinateur reposent essentiellement sur des images matricielles, les descriptions vectorielles (Fichiers) doivent préalablement être converties en descriptions matricielles avant d'être affichées comme images.

3.6 Travailler avec OpenCV

OpenCV est l'abréviation de Open Source Computer Vision. Elle est une bibliothèque open source spécialement conçu pour le domaine du traitement d'image et de la vision par ordinateur. Elle est écrite en langage C et C++ d'une façon très optimisée ce qui la rend adéquate pour les applications en temps réel.

OpenCV est la bibliothèque de référence pour la vision par ordinateur, aussi bien dans le monde de la recherche que celui de l'industrie, car elle propose un ensemble de plus de 2500 algorithmes de vision par ordinateur.

Bien qu'elle est développée en langage C, il existe des version pour les langages haut niveau tels que Python, Javascript et Matlab, rendant la tache de développement beaucoup plus simple.

3.6.1 Installation des dépendances

On va utiliser la bibliothèque OpenCV dans Python même si la documentation dans ce langage est moins développée que celle du C++. Une des raisons qui nous a poussé à choisir python

est l'existence d'une variété de bibliothèque de visualisation (en dehors de OpenCV) tel que Matplotlib, des bibliothèque des opérations sur les matrices (NumPy) et d'autres bibliothèques beaucoup plus adaptées pour l'intelligence artificielle (si l'on souhaite intégré l'IA dans notre application).

On aura donc besoin du Python, la version qu'on a utilisé est 3.8.5 et aussi du PIP (Package Installer for Python) pour télécharger les bibliothèques dont aura besoin et qui ne sont pas disponible par défaut.

Il est préférable d'installer Python de manière à utiliser les environnements virtuels, cela nous permet d'éviter les conflits des versions des bibliothèques, cela est possible grâce aux commandes montrées dans la Figure 3.1

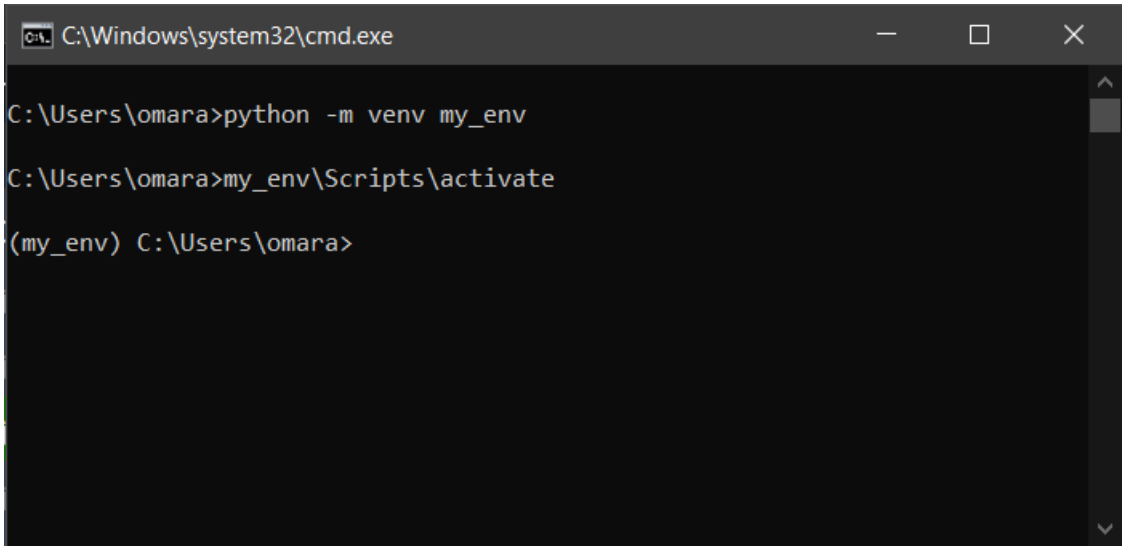

FIGURE 3.1 – Activation de l'environnement virtuel my_env

On aura besoin également de la bibliothèque OpenCV qui elle même dépend de NumPy. On installe aussi la bibliothèque Matplotlib pour afficher les images, les plots, etc. Les commandes des installations sont affichées dans les figure 3.2 et 3.3

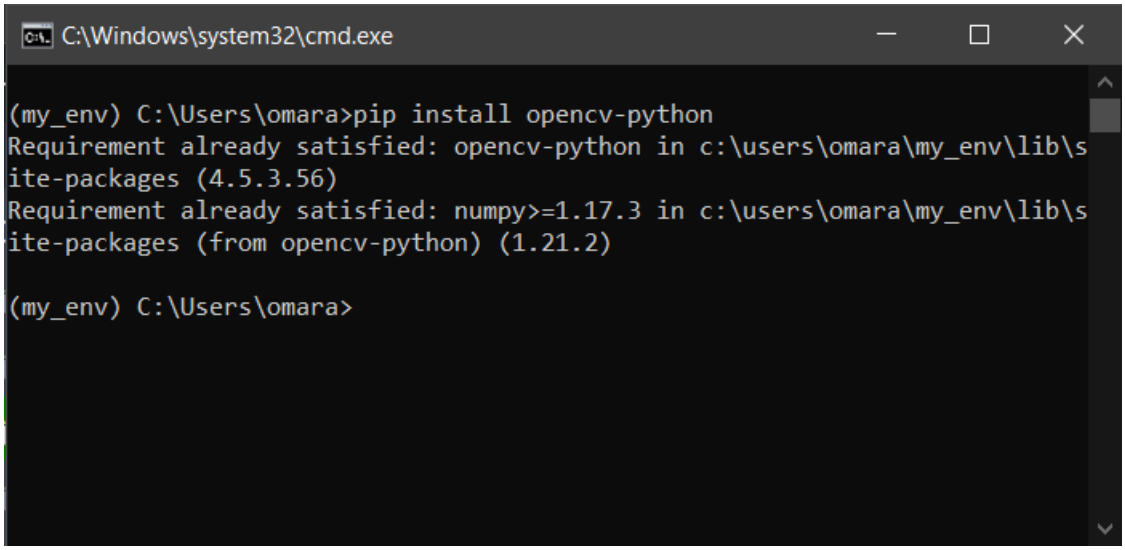

FIGURE 3.2 – Installation de OpenCV

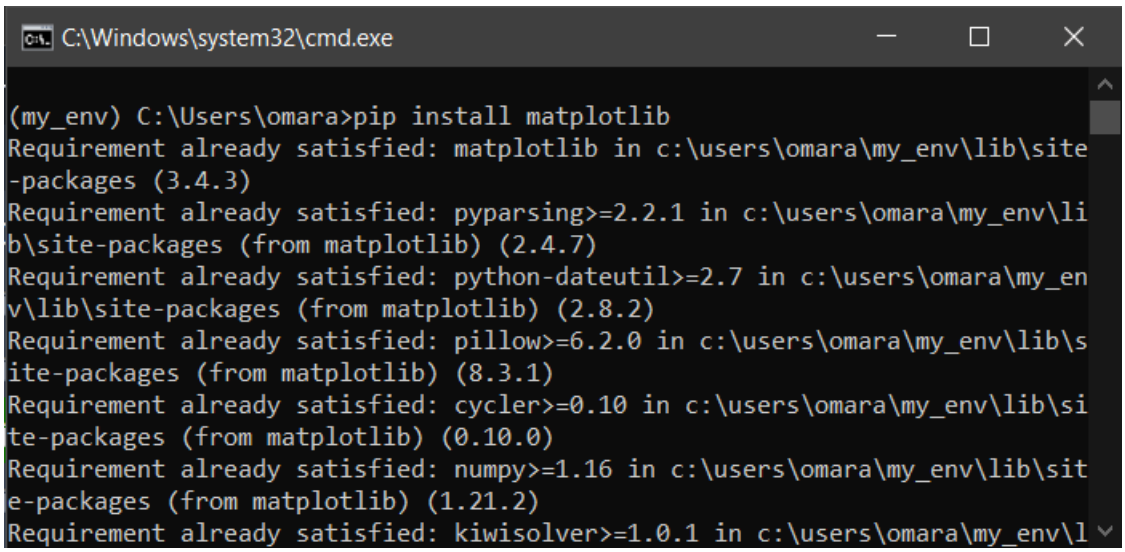

FIGURE 3.3 – Installation de matplotlib

3.6.2 Lire une image

Après l'installation des dépendances, on est prêt pour travailler avec OpenCV. Commençant par lire une image, on le fait à l'aide de la méthode **cv.imread(file, flag)**.
L'argument *file* correspond au chemin d'accès de l'image qu'on souhaite lire.
L'argument *flag* représente le mode de lecture de l'image. Il prend ces trois valeurs possibles :

- cv2.IMREAD_COLOR : Lire une image en couleur et négliger la transparence, c'est le flag par défaut.
- cv2.IMREAD_GRAYSCALE : Lire une image en niveaux de gris.
- cv2.IMREAD_UNCHANGED : Lire une image telle quelle, i.e tenir en compte la transparence (le canal alpha).

On donne l'exemple de ce script qui lit une image "cat.jpg" en mode en couleur, comme un fichier jpg ne contient pas le canal alpha.

```
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt
img = cv.imread("images/cat.jpg", cv.IMREAD_COLOR)
```

3.6.3 Afficher une image

Pour afficher une image on utilise la méthode **cv.imshow(winname, mat)**. L'argument *winname* désigne le nom de la fenêtre sur laquelle on veut afficher l'image. L'argument *mat* représente l'image qu'on vient de lire.

```
cv.imshow("Cat", img)
cv.waitKey(0)
```

L'exécution de code nous ouvre une fenêtre nommée "Cat" et affiche l'image dont on a lu, comme le montre la figure 3.4.
NB : La méthode **cv.waitKey(delay)** a pour rôle d'attendre une période de *delay* millisecondes. Sans cette ligne l'image s'affiche et se ferme rapidement. Si $delay = 0$, l'image sera affichée indéfiniment en attendant un clic sur l'une des touches du clavier ou que l'on ferme la fenêtre.

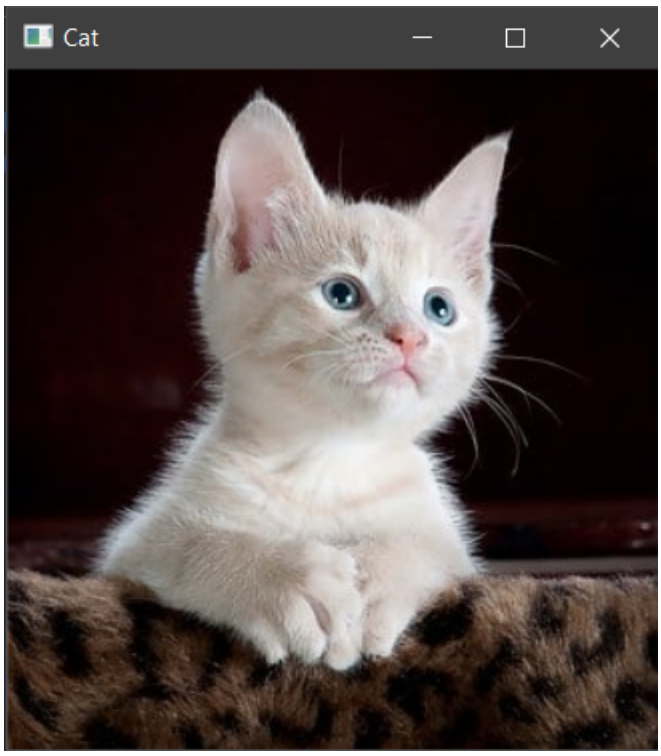

FIGURE 3.4 – L'affichage de l'image Cat.jpg

On remarque que la fenêtre qui s'ouvre, possède la même taille que l'image, cela peut être utile parfois (pour avoir un ordre d'idée sur la taille réelle du photo), mais dans la plus part des temps ce comportement est indésirable. On souhaite avoir un affichage qui plus dynamique (dans le sens où on peut redimensionner la fenêtre), cela est possible grâce à la bibliothèque *matplotlib* qu'on a installé. Le problème qui se pose c'est que la bibliothèque Matplotlib utilise le fameux espace de couleur RGB, or que OpenCV utilise l'espace BGR (pour des raisons historiques), et donc si on essaye d'afficher une image directement i.e sans la convertir vers l'espace de couleur RGB on aura une image complètement différente. Voir figure 3.5. Pour régler ce problème on utilise la méthode qui nous permet de transformer une image d'un espace de couleur vers un autre (dans ce cas on cherche de passer de BGR vers RGB). Cela peut être fait en utilisant la méthode **cv.cvtColor(img, cv.COLOR_BGR2RGB)**. Le script devient donc :

```
img = cv.cvtColor(img, cv.COLOR_BGR2RGB)
plt.imshow(img)
plt.imshow()
```

Après avoir exécuté ce script on aura la fenêtre illustré dans la figure 3.6.
Cette bibliothèque nous offre la possibilité d'obtenir la position (x, y) du curseur et d'inspecter les couleurs à cette position, cela peut être utilisé pour connaître les coordonnées des points de la région d'intérêt *Region of interest* (ROI). On a aussi la possibilité de zoomer sur une région.

3.6.4 Écrire sur une image

On a déjà mentionné qu'une image n'est qu'une grosse matrice contenant des éléments dits "pixels". Chaque pixel est caractérisé par une couleur représentée dans un espace de couleur, et donc il est possible de modifier une image (voire créer une nouvelle) en altérant ses valeurs. L'exemple le plus simple est de créer une image totalement noire, il s'agit d'une matrice de dimensions $h \times w$ dont tous ses éléments sont égales au vecteur nul de dimensions 3×3 (la couleur noire). Après avoir créé cette matrice on utilise la méthode **cv.imwrite(filename, mat)**,

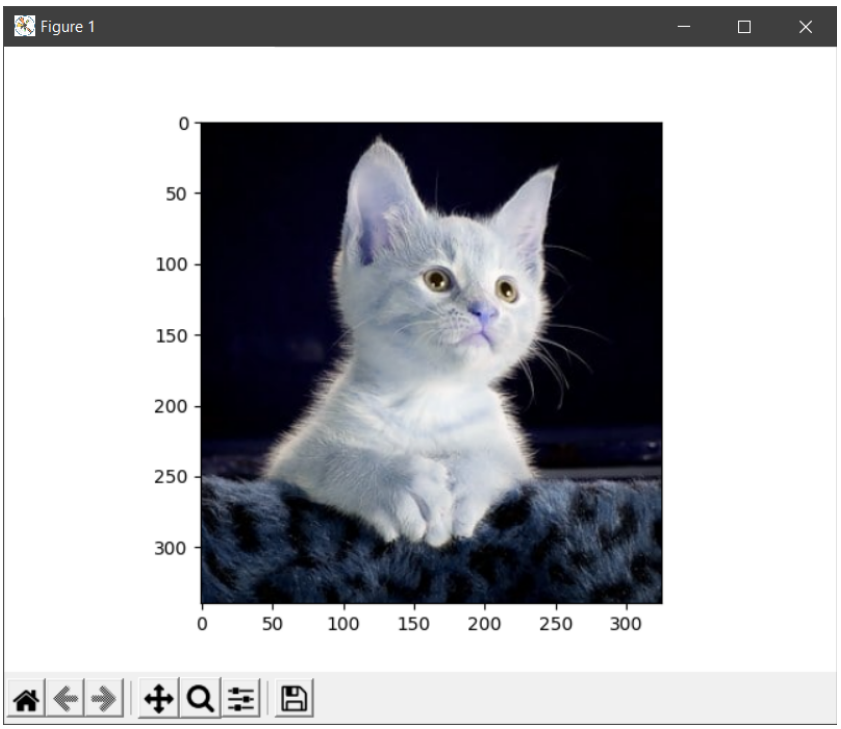

FIGURE 3.5 – Conflit des espaces de couleurs dans matplotlib

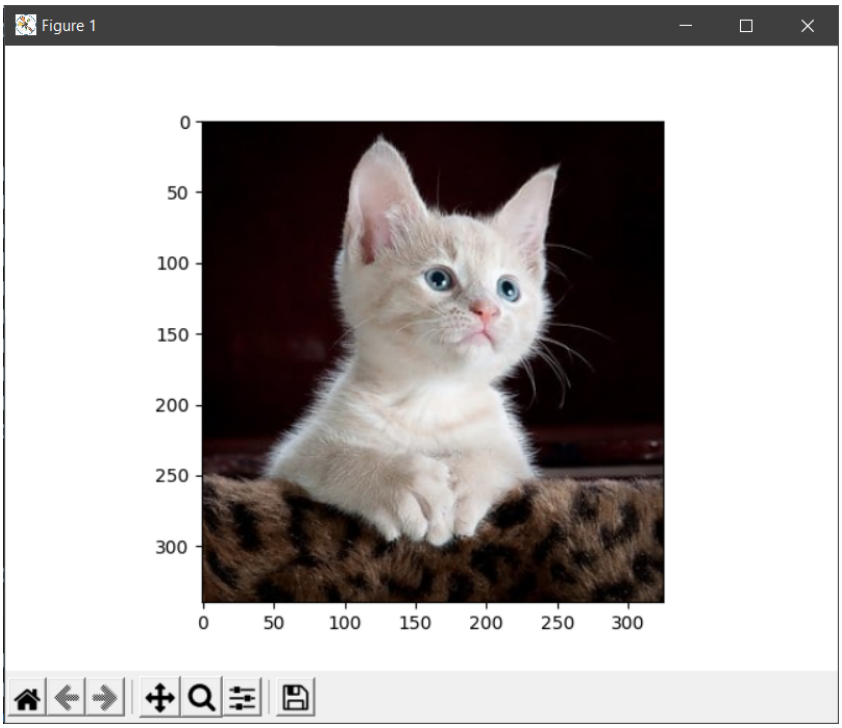

FIGURE 3.6 – Affichage de l'image Cat.jpg avec matplotlib

l'argument *filename* représente le chemin d'accès de l'image que l'on veut créer tandis que mat représente la matrice d'image. Le script suivant créé une image noire de dimension 300, 500.

```
import cv2 as cv
import numpy as np
img = np.zeros([300, 500, 3], np.uint8)
cv.imshow('black image', img)
cv.waitKey(0)
```

On souhaite maintenant modifier l'image d'une façon à ce qu'une région de l'image soit complètement verte, une autre rouge et une troisième bleue. Cette tache est relativement facile et se traduit par remplacer la sous-matrice qui désigne la région d'intérêt par une autre que l'on souhaite. On complète le script précédent pour réaliser cette tache, l'exécution du script résulte en une image illustrée dans la figure 3.7.

```
rH = 50; rW = 100; rx0 = 50; ry0 = 20
redColor = np.array([0, 0, 255], np.uint8)
redRegion = np.zeros([rH, rW, 3], np.uint8) + redColor
img[ry0:ry0+rH, rx0:rx0+rW] = redRegion

gH = 50; gW = 50; gx0 = 350; gy0 = 220
greenColor = np.array([0, 255, 0], np.uint8)
greenRegion = np.zeros([gH, gW, 3], np.uint8) + greenColor
img[gy0:gy0+gH, gx0:gx0+gW] = greenRegion

bH = 100; bW = 50; bx0 = 250; by0 = 85
blueColor = np.array([255, 0, 0], np.uint8)
blueRegion = np.zeros([bH, bW, 3], np.uint8) + blueColor
img[by0:by0+bH, bx0:bx0+bW] = blueRegion

cv.imshow('Colored image', img)
cv.imwrite("Images/Colored image.jpg", img)
```

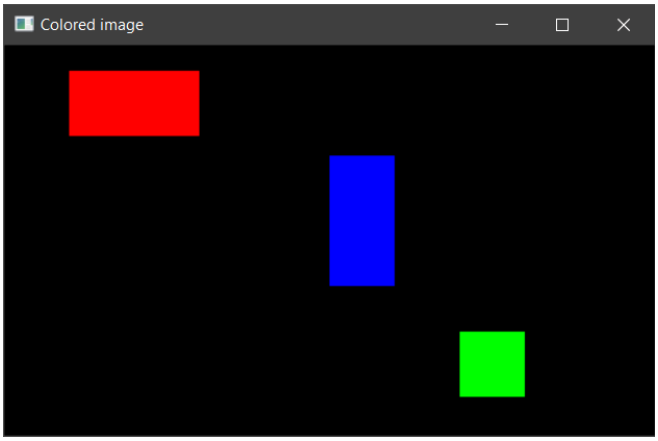

FIGURE 3.7 – Modification d'image par région

3.6.5 Dessiner les formes géométriques de base

On a vu dans la section précédente comment modifier une image en manipulant les pixels par des opérations matricielles, cela est un outil puissant mais on préfère utiliser les méthodes disponible dans OpenCV pour dessiner des formes géométriques un peu plus compliqués.

Commençant par la méthode **cv.line(img, pt1, pt2, color, thickness)** qui comme son nom l'indique permet de dessiner un segment droit entre deux points *pt1* et *pt2* qui sont définies par des tuples (xi, yi). L'argument *color* désigne la couleur de la ligne. Et finalement l'argument *thickness* représente son épaisseur.

Pour dessiner un cercle défini par son centre et son rayon, on utilise la méthode **cv.circle(img, center, raduis, color, thickness)**.

Pour dessiner un rectangle défini par ces deux point opposés *pt1* et *pt2*, on utilise la méthode **cv.rectangle(img, pt1, pt2, color, thickness)**.

D'une manière générale, on peut dessiner un polygone ayant n côtés par la méthode **cv.ploygon(img, [pts], isClosed, color, thickness)**. Le parametre *pts* représent un tableau de dimension $n \times 1 \times 2$ il contient les coordonnées des sommets du polygone. L'argument *isClosed* est une variable booléenne indiquant si le premier et le dernier sommets doivent être connectés. Voici un exemple d'utilisation :

```
pts = np.array([[10, 5], [20, 30], [70, 20], [50, 10]], np.int32)
img = cv.polylines(img, [pts], True, (255, 255, 255), −1)
```

NB : L'argument *thcikness* peut être égale à -1 et cela rend la forme géométrique pleine.

On peut aussi écrire du texte sur une image, cela ce fait par la méthode **cv.putText(img, txt, org, fontFace, fontScale, color)**, l'argument *txt* est une chaîne de caractère elle représente le texte qu'on désire écrire, l'argument *org* représente les coordonnées du point haut en gauche du texte. L'argument *fontFace* représente le police de caractère, il prend comme valeur un flag, par exemple **cv.cv.FONT_HERSHEY_SIMPLEX**. Et finalement, l'argument *fontScale* représente la taille du police.

3.6.6 Lire une vidéo

Une vidéo est un ensemble d'images qui s'affichent rapidement les une après les autres donnant l'impression qu'il s'agit d'une animation continue. Ces images sont connues par le nom anglais *frame*. On définit le *FPS* (frame per second) comme le nombre d'images affichées par seconde.

Pour lire une vidéo avec la bibliothèque OpenCV on utilise la méthode **cap = cv.videoCapture(param)**. L'argument *param* peut être le chemin d'accès d'une vidéo, comme il peut être un entier signifiant l'index de la caméra qu'on souhaite utiliser (il est égal à 0 par défaut).
Après avoir lu la vidéo, on doit créer une boucle **while** qui va tourner tant que la vidéo est ouverte. On sait si la vidéo est ouverte grâce à la méthode **cap.isOpenned()**, dans cette boucle on souhaite lire le *frame* actuel, cela est possible par la méthode **ret, frame = cap.read()** qui retourne deux valeurs, le premier est un booléen qui signifie si le frame a été bien lu, et le deuxième représente le frame en question.
Ensuite, on ouvre une condition qui vérifie si la valeur de *ret* est fausse, si c'est le cas, on affiche un message d'erreur peut-être et on quitte la boucle principale. Si ce n'est pas le cas, cela signifie que tout marche bien et qu'on prêt à faire du traitement sur le frame reçu.
On aura besoin d'une condition pour arrêter cette boucle (parce que sinon ça risque de tourner indéfiniment), une possibilité est d'utiliser la touche "q" pour fermer la fenêtre.
Finalement, en dehors de la boucle, on libère la caméra en utilisant la méthode **cap.realease()**, cela est nécessaire pour que les autres processus puissent avoir accès de nouveau à la caméra.
Voici un exemple qui affiche la vidéo capturer par la caméra ayant l'index 0 :

```
import cv2 as cv

cap = cv.videoCapture(0)
while cap.isOpened():
    ret, frame = cap.read()
```

```
    if not ret:
        print("Une erreur s'est produite")
        break
    cv.imshow("Video", frame)
    k = cv.waitKey(1)
    if k == ord("q")
        break

cap.release()
```

3.7 Les bases du traitement d'image

3.7.1 Histogramme

Outil de base pour l'étude et l'analyse des images, il s'agit d'une représentation graphique utilisée pour visualiser la fréquence d'apparence des couleurs dans une image. Il est vachement utilisé dans la tache de détection des objets peut être suivant leurs couleurs (la teinte i.e le canal h dans l'espace de couleur HSV) ou par leurs intensité, etc. Il est aussi utilisé pour améliorer une image, on parle d'égalisation d'histogramme. Concrètement, l'histogramme d'une image en niveaux de gris est construit de la manière suivante : Pour chaque niveau de gris x, on compte le nombre de pixels ayant la valeur x.
Si on divise chaque valeur de l'histogramme par le nombre de pixel, on aura un histogramme normalisée dont les valeurs sont comprise entre 0 et 1.
La méthode qui nous permet de visualiser l'histogramme d'une image dans *matplotlib* est **plt.hist(data, nbr, (min, max))**, l'argument *data* représente un canal aplati (exemple le canal rouge de l'espace de couleur RGB), *nbr* représente le nombre de points dans l'histogramme, le dernier argument représente l'intervalle des points.
Exemple : Le script suivant a pour but d'afficher les trois histogrammes qui correspondent aux canaux rouge, vert et bleu comme le montre la figure 3.8

```
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt

img = cv.imread('Images/cat.jpg')
img = cv.cvtColor(img, cv.COLOR_BGR2RGB)
r, g, b = cv.split(img)

plt.subplot(2, 2, 1)
plt.imshow(img)
plt.axis('off')
plt.title('Cat.jpg')
plt.subplot(2, 2, 2)
plt.hist(r.ravel(), 256, [0, 256], color='red')
plt.title('Histogramme du canal rouge')
plt.subplot(2, 2, 3)
plt.hist(g.ravel(), 256, [0, 256], color='green')
plt.title('Histogramme du canal vert')
plt.subplot(2, 2, 4)
plt.hist(b.ravel(), 256, [0, 256], color='blue')
```

```
plt.title('Histogramme du canal bleu')
plt.show()
```

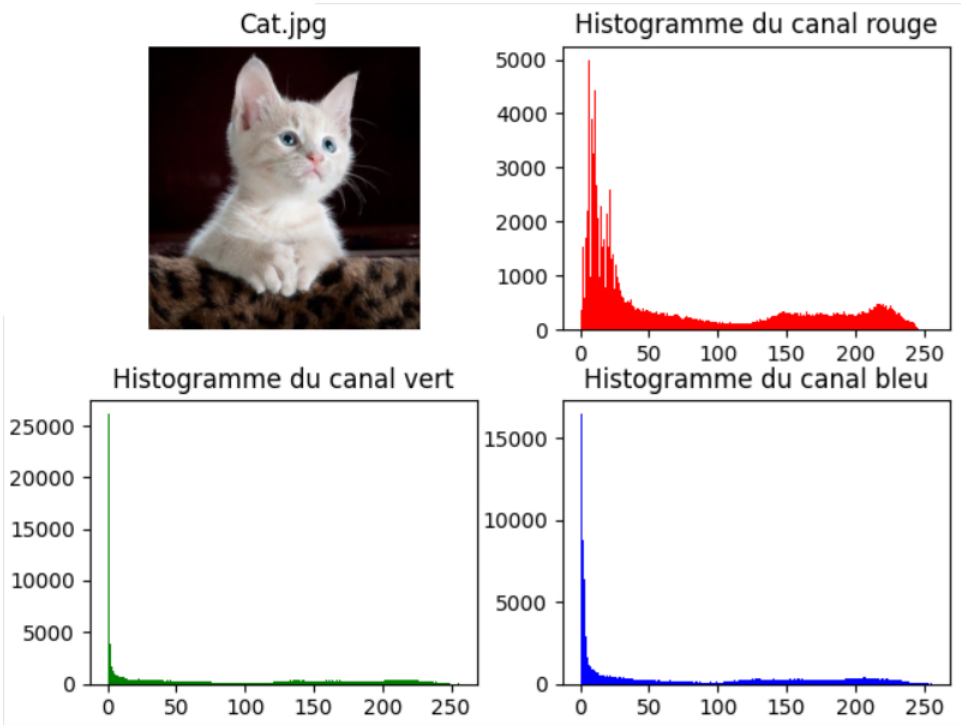

FIGURE 3.8 – Histogramme des canaux RGB de l'image cat.jpg

3.7.2 Convolution

La convolution est un outil puissant à usage général, on peut l'utiliser pour traiter une image ou pour extraire une caractéristique.
La convolution est faite en multipliant terme à terme (produit d'Hadamard) la valeurs de chaque pixels et ses voisins par une matrice appelée noyau de convolution (*Kernel*) ou aussi masque de convolution (*mask*). La convolution est décrite rigoureusement par l'équation 3.1

$$g[i,j] = \sum_{m=1}^{M} \sum_{n=1}^{N} f[m,\, n]\, h[i-m,\, j-n] \tag{3.1}$$

Exemple :
soient une matrice f et un noyau de convolution h donnés par :

$$h = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \qquad f = \begin{bmatrix} 1 & 2 & 0 & 1 & 3 \\ 0 & 3 & 1 & 2 & 4 \\ 2 & 0 & 4 & 0 & 2 \\ 4 & 1 & 3 & 1 & 0 \\ 0 & 2 & 4 & 3 & 2 \end{bmatrix} \tag{3.2}$$

On cherche à calculer $g[2,\, 3] = (f * h)[2,\, 3]$, on commence par inverser l'ordre des lignes et des colonnes de h comme suit :

$$h = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \qquad \tilde{h} = {}^{v}h = \begin{bmatrix} g & h & i \\ d & e & f \\ a & b & c \end{bmatrix} \qquad \hat{h} = {}^{h}\tilde{h} \begin{bmatrix} i & h & g \\ f & e & d \\ c & b & a \end{bmatrix} \tag{3.3}$$

On forme une matrice $\hat{f}$ centrée dans l'élément (2, 3) et a la même dimension que $\hat{h}$, on aura :

$$\hat{f} = \begin{bmatrix} 2 & 0 & 1 \\ 3 & 1 & 2 \\ 0 & 4 & 0 \end{bmatrix} \tag{3.4}$$

On utilisant le produit d'Hadamard on aura :

$$A = \hat{f} \circ \hat{h} = \begin{bmatrix} 2 \cdot i & 0 \cdot h & 1 \cdot g \\ 3 \cdot f & 1 \cdot e & 2 \cdot d \\ 0 \cdot c & 4 \cdot b & 0 \cdot a \end{bmatrix} \tag{3.5}$$

Finalement,

$$g[2,3] = \sum_{i=1}^{3} \sum_{j=1}^{3} A[i,\, j] = 2i + g + 3f + e + 2d + 4b \tag{3.6}$$

$$\left(\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} * \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \right) [2,2] = (i{\cdot}1){+}(h{\cdot}2){+}(g{\cdot}3){+}(f{\cdot}4){+}(e{\cdot}5){+}(d{\cdot}6){+}(c{\cdot}7){+}(b{\cdot}8){+}(a{\cdot}9) \tag{3.7}$$

Avec : f_{ij} est le coefficient du kernel à la position

3.8 Images binaires

C'est la plus simple image comme ses éléments ne peuvent être que 0 ou 1 (0 ou 255 pour un codage sur 8-bits) représentant la couleur noire et blanche respectivement. Il est donc très facile de les traiter, stocker et les analyser. Mais malgré leurs simplicité, elles sont très utiles, notamment dans les détecteurs d'empreintes digitales, jeu de la vie, les lecteurs de code bar et code QR, l'imagerie médicale, etc.
Les images binaires sont souvent obtenues en faisant des opérations de seuillage sur des images en niveaux de gris. Le seuil peut être automatiquement calculé ou prédéfini par l'utilisateur. Cette opération est traduite par la fonction suivante :

$$b(x,y) = \begin{cases} 0 & si\ g(x,y) < T \\ 1 & si\ g(x,y) \geq T \end{cases} \tag{3.8}$$

L'équation 3.8 s'appelle une fonction indicatrice ou fonction caractéristique. Le seuil T est généralement choisi à partir de l'histogramme de l'image g(x, y). Si on est chanceux, l'histogramme aura deux modes et choisir le seuil manuellement sera relativement facile, il est même possible de construire une procédure automatique pour le faire. Idéalement, si on a un objet clair sur un fond sombre, l'histogramme sera similaire à celui de la figure 3.9, mais dans le cas réel l'existence du bruit rend l'histogramme beaucoup moins clair comme le résultat final est la convolution de l'histogramme "idéal" et la distribution de probabilité dans la figure 3.10
Dans le cas où les niveau du gris de l'arrière-plan et l'objet sont très proche (voir figure 3.11, le bruit peut influencer l'histogramme d'une façon à ce que l'histogramme aura un seul mode, et donc séparer les deux modes sera difficile voire impossible (sans perte d'information). La figure 3.13 montre l'effet de seuillage sur l'image originale et l'image bruitée.
Si on prend l'exemple 3.15, on remarque qu'il existe deux modes, un représente l'arrière-plan et l'autre représente la pièce de monnaie. La partie qui sépare les deux modes désigne une bonne plage pour choisir le seuil.

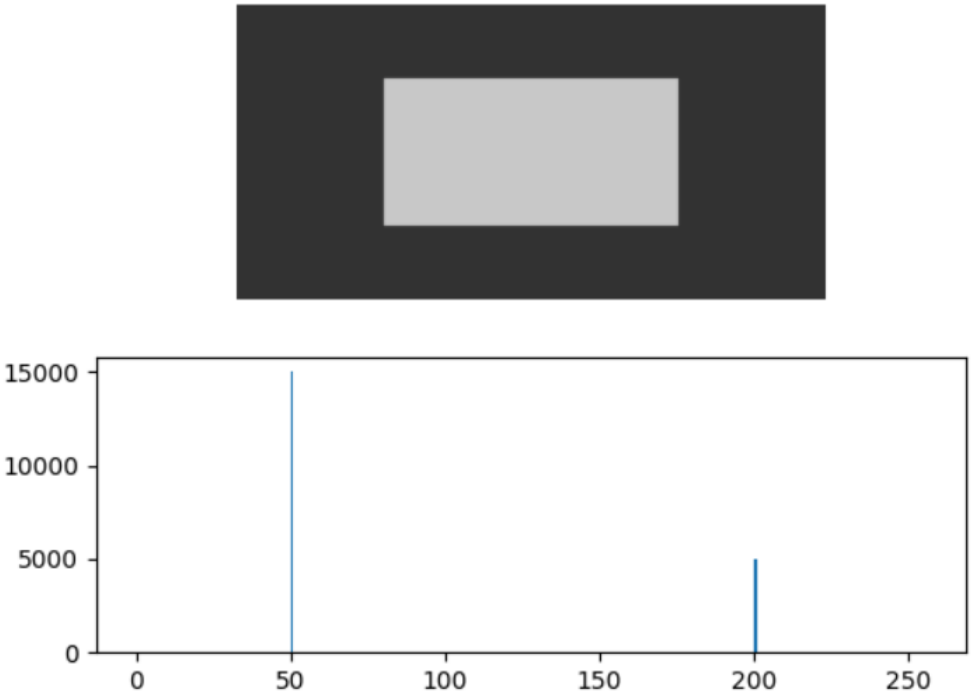

FIGURE 3.9 – Histogramme d'une image contenant deux couleurs

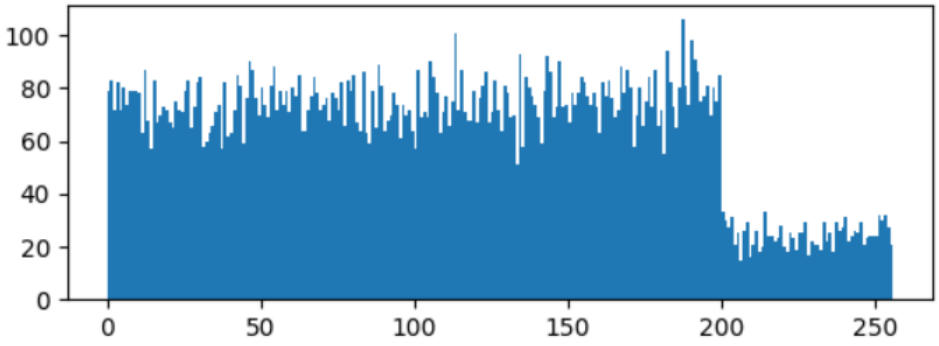

FIGURE 3.10 – Histogramme de la même image + Bruit

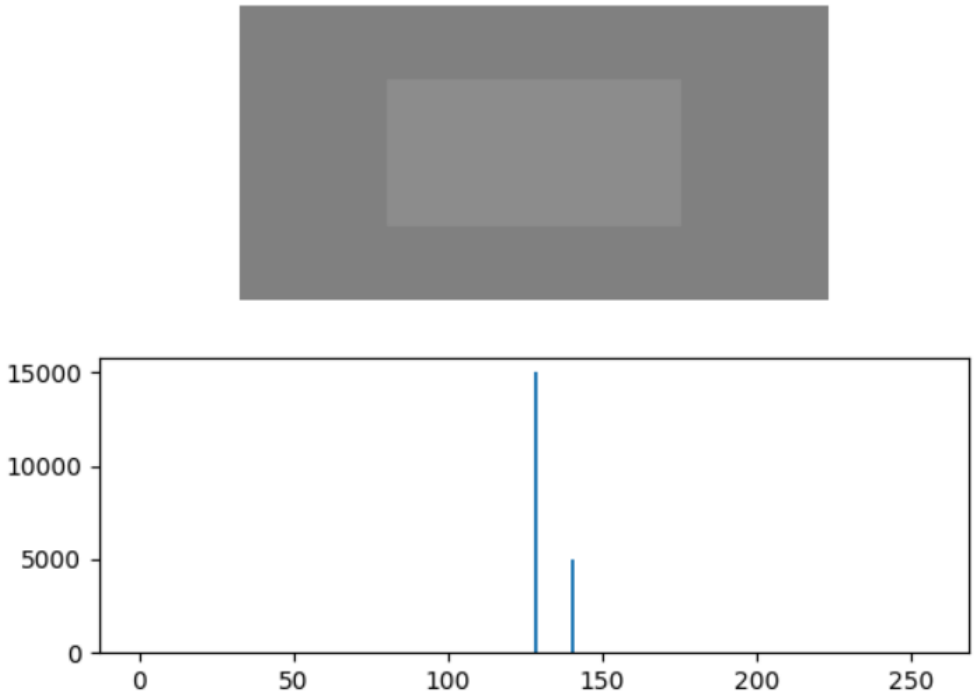

FIGURE 3.11 – Histogramme d'une image contenant deux couleurs proches

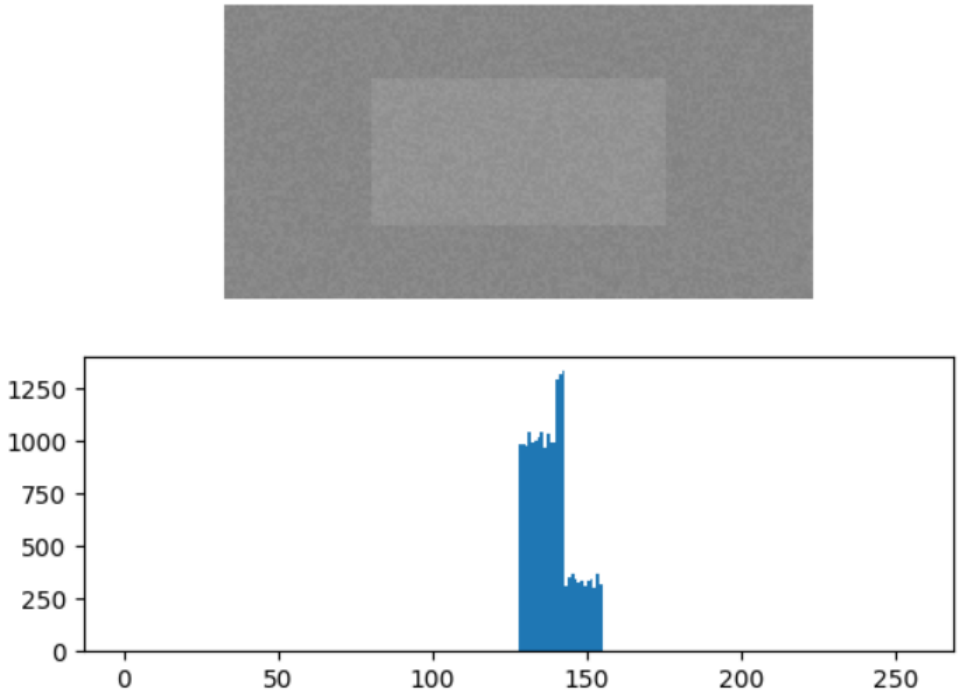

FIGURE 3.12 – Histogramme d'une image contenant deux couleurs proches + Bruit

Si on choisit le seuil T = 38, et qu'on fait l'opération de seuillage, on aura l'image binaire illustrée dans la figure 3.15

Supposant qu'on a une image binaire continue b(x, y) et qu'il en existe un seul objet, comme le montre la figure **??**. On peut définir quelque propriétés géométriques qui nous seront utiles ultérieurement.

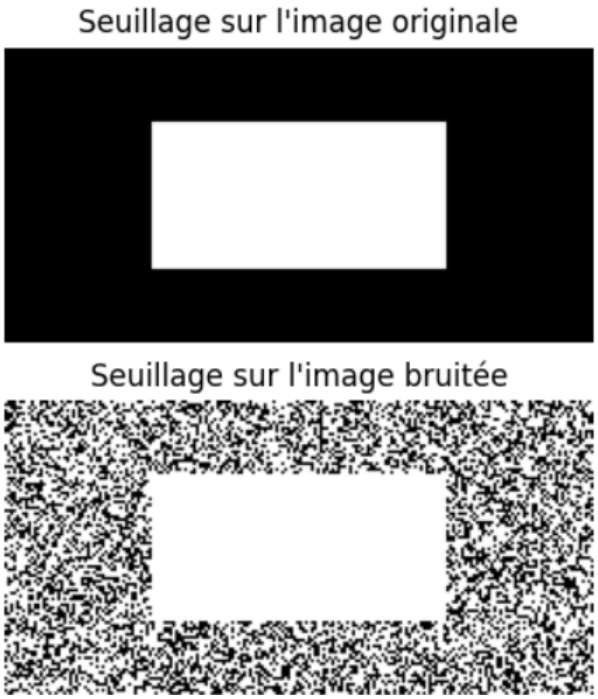

FIGURE 3.13 – seuillage sur l'image originale et l'image bruitée

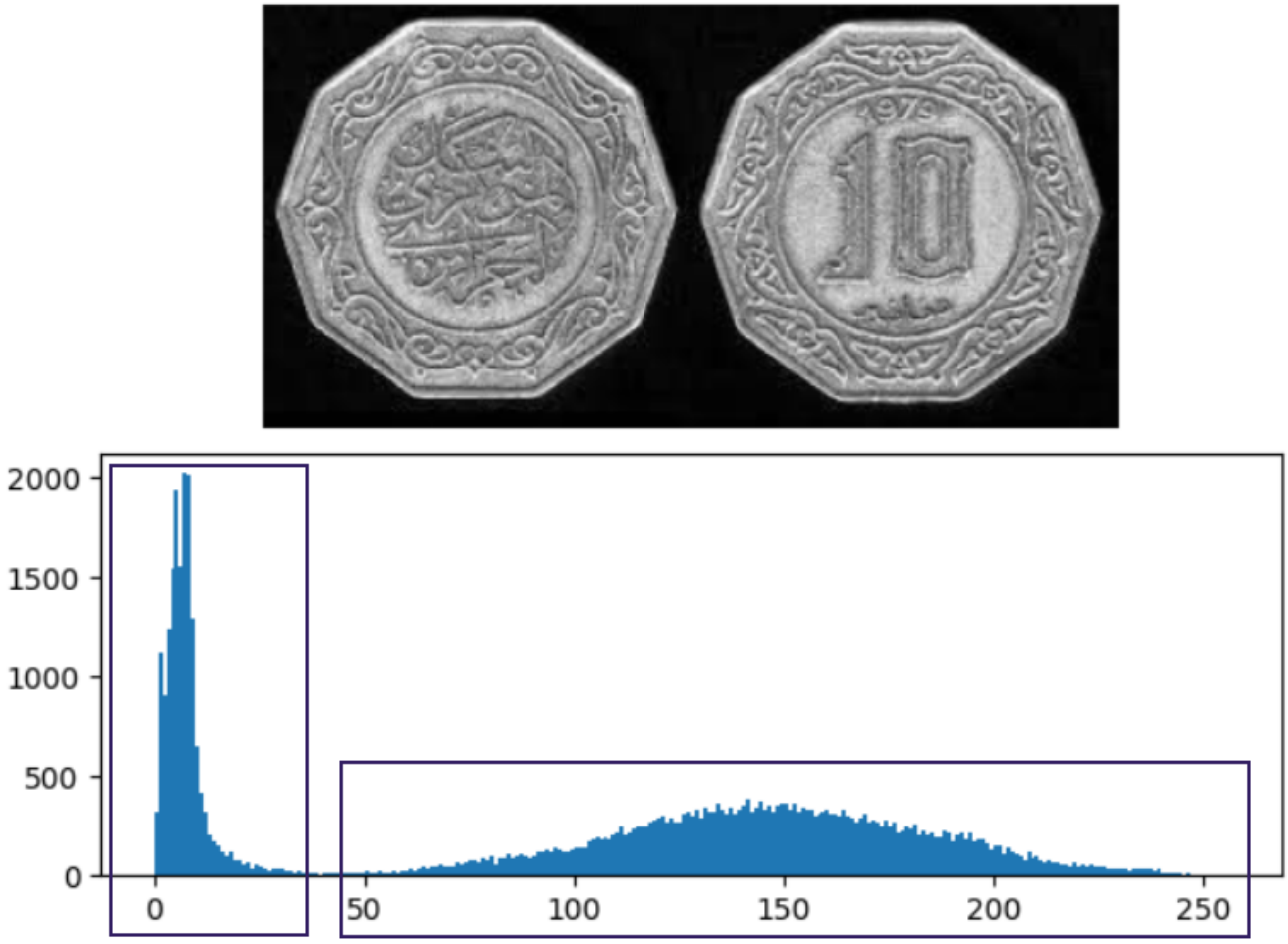

FIGURE 3.14 – Illustration des deux modes de l'histogramme

FIGURE 3.15 – L'image binaire obtenue après le seuillage

3.8.1 La surface

La surface qui est aussi connue sous le nom moment d'ordre 0, est calculée comme on a l'habitude de calculer les surfaces des objets géométriques. On a donc :

$$A = \iint b(x, y)\, dxdy \tag{3.9}$$

3.8.2 Le centroïde

Le centroïde ou le moment d'ordre 1, peut être utilisé pour désigner la position d'un objet, il est calculé comme suit :

$$\bar{x} = \frac{1}{A} \iint x b(x, y)\, dxdy \qquad \bar{y} = \frac{1}{A} \iint y b(x, y)\, dxdy \tag{3.10}$$

3.8.3 L'orientation

La détermination de l'orientation d'un objet n'est pas aussi simple, mais il existe une manière robuste pour le faire, il s'agit de trouver l'axe qui minimise le moment d'ordre 2 qui est défini comme suit.

$$I = \iint r^2 b(x, y)\, dxdy \tag{3.11}$$

On cherche donc à trouver la droite qui minimise 3.11, r étant la distance entre un point (x, y) et l'axes qu'on cherche.
On va utiliser l'équation paramétrique de la ligne définie par 3.12, θ représente l'angle que fait la droite avec l'axe des abscisses et ρ étant la distance entre la droite et l'origine, comme le montre la figure 3.16.

$$x \sin\theta - y \cos\theta + \rho = 0 \tag{3.12}$$

On peut représenter n'importe quel point (x_0, y_0) appartenant à la droite par 3.13, t représente la distance entre le point (x_0, y_0) et le point le plus proche de l'origine :

$$\begin{cases} x_0 = -\rho \sin\theta + t \cos\theta \\ y_0 = \rho \cos\theta + t \sin\theta \end{cases} \tag{3.13}$$

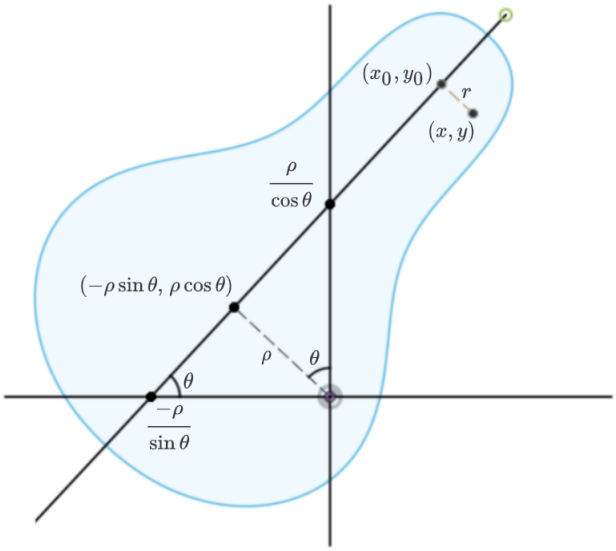

FIGURE 3.16 – Paramétrisation d'une droite par un angle et une distance

Étant donné un point (x, y), on cherche à trouver le point (x_0, y_0) tel que la distance r entre (x, y) et (x_0, y_0) soit minimale, r est donné par :

$$r^2 = (x - x_0)^2 + (y - y_0)^2 \tag{3.14}$$

On remplace par x_0 et y_0 pour avoir :

$$r^2 = x^2 + y^2 + \rho^2 + 2\rho(x\sin\theta - y\cos\theta) - 2t(x\cos\theta + y\sin\theta) + t^2 \tag{3.15}$$

On cherche à minimiser l'expression 3.15 par rapport à t.

$$\frac{\partial r^2}{\partial t} = 0 \implies t = x\cos\theta + y\sin\theta \tag{3.16}$$

On remplace 3.16 dans 3.13 on aura :

$$\begin{aligned} x - x_0 &= x - (-\rho\sin\theta + (x\cos\theta + y\sin\theta)\cos\theta && (3.17)\\ &= x + \rho\sin\theta - x\cos^2\theta - y\sin\theta\cos\theta && (3.18)\\ &= x(1 - \cos^2\theta) + \rho\sin\theta - y\sin\theta\cos\theta && (3.19)\\ &= x\sin^2\theta + \rho\sin\theta - y\sin\theta\cos\theta && (3.20)\\ &= \sin\theta(x\sin\theta - y\cos\theta + \rho) && (3.21) \end{aligned}$$

De la même manière pour $y - y_0$ on aura :

$$y - y_0 = \cos\theta(x\sin\theta - y\cos\theta + \rho) \tag{3.22}$$

Donc r^2 peut être écrit comme :

$$r^2 = (x\sin\theta - y\cos\theta + \rho)^2 \tag{3.23}$$

On remplace r^2 dans 3.11 par 3.23 et on aura :

$$I = \iint (x\sin\theta - y\cos\theta + \rho)^2\, b(x, y)\, dxdy \tag{3.24}$$

On souhaite maintenant trouver les paramètres ρ et θ qui minimisent la distance entre un point dans l'objet et la droite qu'on cherche, comme suit :

$$\frac{\partial I}{\partial \rho} = 0 \implies \iint 2(x\sin\theta - y\cos\theta + \rho)\, b(x,y)\, dxdy = 0 \tag{3.25}$$

qui est juste

$$\iint x\sin\theta\, b(x,y)\, dxdy - \iint y\cos\theta\, b(x,y)\, dxdy + \iint \rho\, b(x,y)\, dxdy = 0 \tag{3.26}$$

multipliant et divisant 3.26 par la surface :

$$A\left(\frac{1}{A}\iint x\sin\theta\, b(x,y)\, dxdy - \frac{1}{A}\iint y\cos\theta\, b(x,y)\, dxdy + \frac{1}{A}\iint \rho\, b(x,y)\, dxdy\right) = 0 \tag{3.27}$$

On aura donc :

$$A(\bar{x}\sin\theta - \bar{y}\cos\theta + \rho) = 0 \tag{3.28}$$

Cela veut dire que l'axe qui minimise le moment du deuxième ordre passe par le centroïde. On doit maintenant trouver la valeur de θ qui minimise I, pour faire cela, on change les coordonnées pour travailler par rapport au centroïde, posant :

$$\left\langle \begin{matrix} x' = x - \bar{x} \\ y' = y - \bar{y} \end{matrix} \right\rangle \tag{3.29}$$

Cela simplifie l'équation de notre droite

$$x\sin\theta - y\cos\theta + \rho = x'\sin\theta - y'\cos\theta \tag{3.30}$$

Et donc

$$I = \iint (x'\sin\theta - y'\cos\theta)^2\, b(x',y')dx'dy' \implies I = a\sin^2\theta - b\sin\theta\cos\theta + c\cos^2\theta \tag{3.31}$$

Avec

$$\begin{aligned} a &= \iint x'^2\, b(x',y')\, dx'dy' \\ b &= 2\iint x'y'\, b(x',y')\, dx'dy' \\ c &= \iint y'^2\, b(x',y')\, dx'dy' \end{aligned}$$

Les constantes a, b et c sont appelées les moments du deuxième ordre ou en anglais *second moments*.
Si on utilise le fait que $b\sin\theta\cos\theta = \frac{b}{2}\sin 2\theta$ et $\cos 2\theta = 2\cos^2\theta - 1$, on aura :

$$\begin{aligned} I &= a(1-\cos^2\theta) + c\cos^2 - \frac{b}{2}\sin 2\theta \\ &= a + (c-a)\cos^2\theta - \frac{b}{2}\sin 2\theta \\ &= a + \frac{c-a}{2}\cos 2\theta + \frac{c-a}{2} - \frac{b}{2}\sin 2\theta \end{aligned}$$

Ainsi

$$I = \frac{1}{2}(c+a) - \frac{1}{2}(a-c)\cos 2\theta - \frac{1}{2}b\sin 2\theta \tag{3.32}$$

$$\frac{\partial I}{\partial \theta} = 0 \implies (a-c)\sin 2\theta - b\cos 2\theta = 0 \tag{3.33}$$

Si $b \neq 0$ et $c \neq a$, on aura,

$$\tan 2\theta = \frac{b}{a-c} \tag{3.34}$$

Donc,

$$\frac{\sin^2 2\theta}{\cos^2 2\theta} = \frac{\sin^2 2\theta}{1-\sin^2 2\theta} = \frac{b^2}{(a-c)^2} \tag{3.35}$$

Il s'agit d'une équation quadratique en $\sin 2\theta$, résoudre cette dernière nous donne :

$$\sin 2\theta = \frac{\pm b}{\sqrt{b^2+(a-c)^2}} \tag{3.36}$$

$$\cos 2\theta = \frac{\pm (a-c)}{\sqrt{b^2+(a-c)^2}} \tag{3.37}$$

Lorsqu'on choisit explicitement la solution positive cela revient à minimiser I, et si on choisit la solution négative, I sera maximisée.
Dans le cas où $b = 0$ et $a = c$, on remarque que I ne sera pas affectée par la direction de l'axe de l'orientation, ceci signifie que l'objet ait une symétrie de rotation.
Le ratio $\frac{I_{min}}{I_{max}}$ nous informe sur le "roundedness" de l'objet, ce ratio est nul pour une ligne et il est égal à un 1 pour un cercle.
Jusqu'à maintenant nous avons supposé que la fonction est continue, or qu'on réalité, il s'agit d'une fonction discrète, on doit donc redéfinir les propriétés géométriques dans le cas discret.
La surface est donnée par :

$$A = \sum_{i=1}^{n}\sum_{j=1}^{m} b_{ij} \tag{3.38}$$

Les coordonnées du centroïde sont données par :

$$\bar{x} = \frac{1}{A}\sum_{i=1}^{n}\sum_{j=1}^{m} i b_{ij} \qquad\qquad \bar{y} = \frac{1}{A}\sum_{i=1}^{n}\sum_{j=1}^{m} j b_{ij} \tag{3.39}$$

Les moments d'ordre deux sont donnés par :

$$a' = \sum_{i=1}^{n}\sum_{j=1}^{m} i^2 b_{ij} \qquad b' = 2\sum_{i=1}^{n}\sum_{j=1}^{m} ij b_{ij} \qquad c' = \sum_{i=1}^{n}\sum_{j=1}^{m} j^2 b_{ij} \tag{3.40}$$

NB : a', b' et c' sont les moments d'ordre 2 par rapport à l'origine. Les moments par rapport au centroïde a, b, c peuvent être écrits en fonction de $a', b', c', \bar{x}, \bar{y}, A$,

Exemple : Le moment a est donné par :

$$\begin{aligned}
a &= \sum_{i=1}^{n}\sum_{j=1}^{m}(i-\bar{x})^2 b_{ij} = \sum_{i=1}^{n}\sum_{j=1}^{m}(i^2+\bar{x}^2-2i\bar{x})b_{ij} \\
&= \sum_{i=1}^{n}\sum_{j=1}^{m} i^2 b_{ij} + \bar{x}^2 \sum_{i=1}^{n}\sum_{j=1}^{m} b_{ij} - 2\bar{x}\sum_{i=1}^{n}\sum_{j=1}^{m} i b_{ij} \\
&= a' + A\bar{x}^2 - 2A\bar{x}\frac{1}{A}\sum_{i=1}^{n}\sum_{j=1}^{m} i b_{ij} \\
&= a' + A\bar{x}^2 - 2A\bar{x}^2 \\
a &= a' - A\bar{x}^2
\end{aligned}$$

De la même manière on peut obtenir b et c.

Chapitre 4

Réalisation du robot SCARA

4.1 Les moteurs électriques

Dans ce projet on a utilisé trois servomoteur DC pour les articulations rotoïdes, et un moteur pas à pas pour l'articulation prismatique.
Le servo moteur utilisé est **RDS3218** (voir figure 4.1). Il est doté par les caractéristiques suivantes :

- Référence : RDS3218
- Tension : 4.8V - 6.8V
- Vitesse (caractérisée par le temps pour atteindre 60°) : 0.19s à 5V, 0.17s à 6.8V
- Couple : 18.5kg.cm à 5V, 21kg.cm à 6.8v
- Masse : 60g
- Intervalle d'angle : 270°

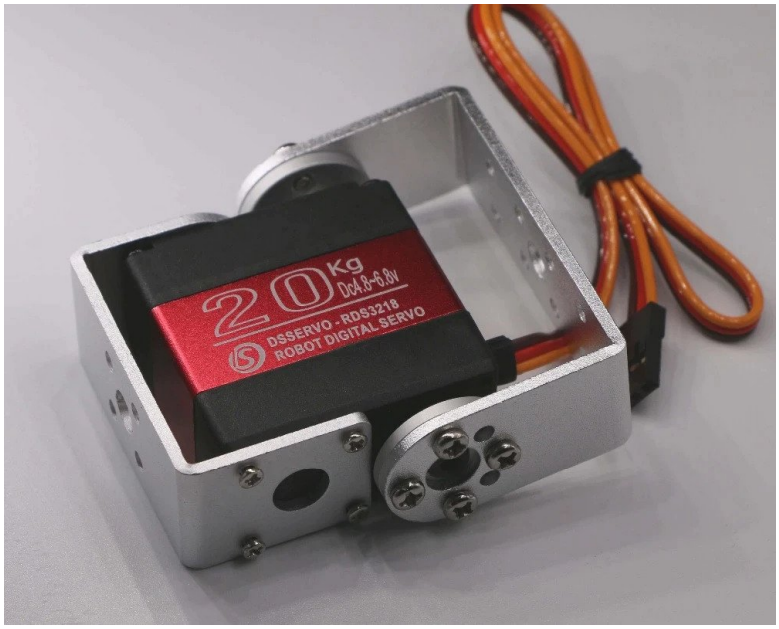

FIGURE 4.1 – Le servomoteur RDS3218

le moteur pas à pas utilisé est NEMA 17 17HS4401 (voir figure **??**). Il est caractérisé par :

- Référence : 17HS4401
- Courant/phase : 1.7A

— angle/pas : 1.8°
— Couple de maintien : 4.2kg.cm / 0.42N.m
— Diamètre de l'arbre du moteur : 5mm

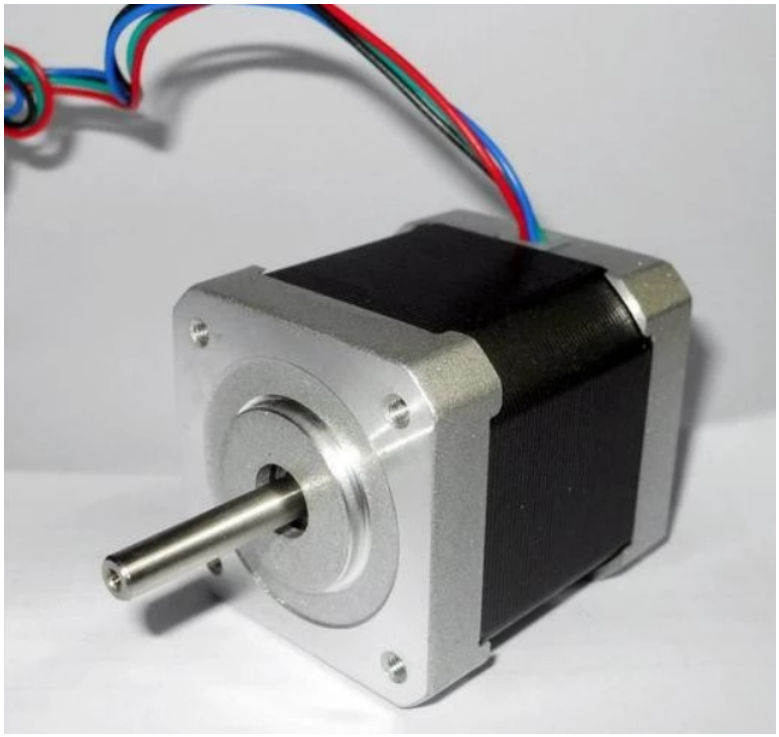

FIGURE 4.2 – Le moteur pas à pas NEMA 17 17HS4401

4.2 Driver a4988

Contrôler le moteur pas à pas n'est pas facile, il est donc nécessaire d'utiliser un driver qui fait l'affaire. On a utilisé le driver **Pololu a4988** comme le montre la figure 4.3.

Caractéristique du driver **a4988** :
— Tension minimale VDD : 3V
— Tension maximale VDD : 5.5v
— Courant maximal par phase : 2A
— Tension minimale d'alimentation du moteur VMOT : 8V
— Tension maximale d'alimentation du moteur VMOT : 24V

Les broches du driver sont illustré dans la figure 4.4,
— GND et VDD sont les broches d'alimentation logique.
— $1A$, $1B$ se connectent avec une bobine du moteur pas à pas.
— $2A$, $2B$ se connectent avec l'autre bobine du moteur pas à pas.
— GND et $VMOT$ sont les broches d'alimentation du moteur (souvent égale 12V)
— $\overline{ENA}$ est la broche qui contrôle si les sorties $1A$, $1B$, $2A$, $2B$ sont activées ou non.
— $MS1$, $MS2$, $MS3$ contrôlent le microstepping. selon le tableau (4.5).

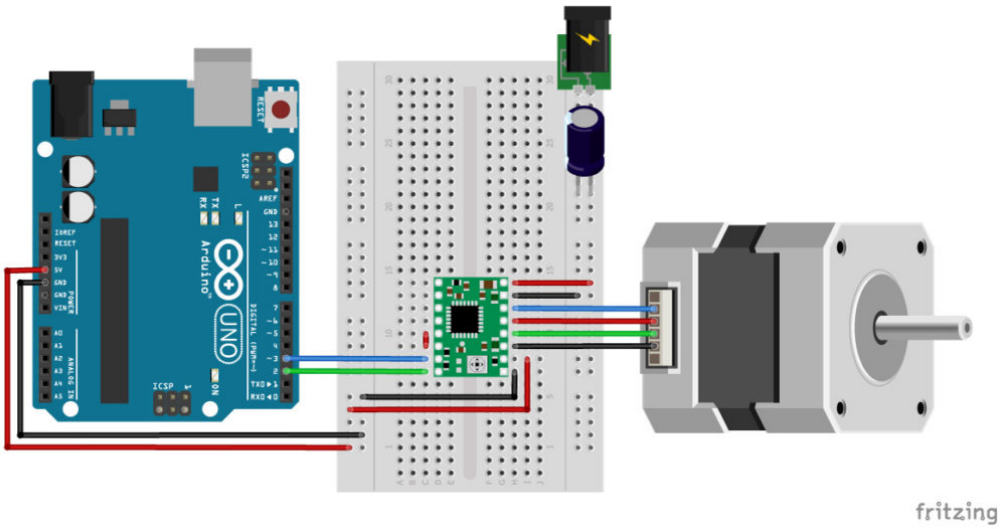

FIGURE 4.3 – Schéma d'utilisation du driver a4988 avec NEMA 17

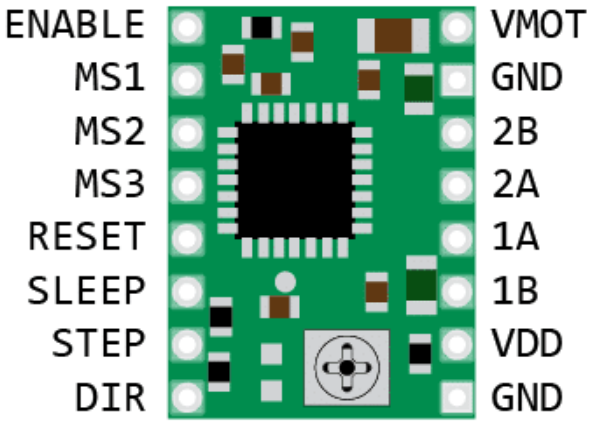

FIGURE 4.4 – Broches du driver a4988

— $\overline{RESET}$ et $\overline{SLEEP}$ doivent être connectées pour que le driver fonctionne.
— $STEP$ chaque impulsion à cette broche fait tourner le moteur par un pas.
— DIR contrôle la direction de rotation du moteur.

Le tableau 4.5 représente la résolution du moteur en fonction du niveau logique de $MS1$, $MS2$ et $MS3$, Le microstepping est une technique qui permet d'avoir des mouvement plus lisses en divisant la résolution. donc si la résolution en mode full step est 1.8° la résolution en half step sera 0.9°, etc.

4.3 L'organe terminal

Un robot manipulateur peut faire plusieurs tâches, il suffit juste d'utiliser l'organe terminal adéquat, on peut citer des exemples d'organes terminaux : pince, ventouse, électro-aimant, etc. On a opté pour la ventouse car elle est simple à employer, elle marche avec de différents matériaux, pas comme l'électro-aimant qui marche exclusivement avec les métaux. en revanche la ventouse ne peut pas attraper un objet ayant une surface plein de pores, comme une éponge, un objet imprimé en 3D, etc.
On aura besoin d'une pompe à vide, une ventouse et une électrovanne. La pompe utilisée (figure 4.6) est caractérisée par :

MS1	MS2	MS3	Microstep Resolution
Low	Low	Low	Full step
High	Low	Low	Half step
Low	High	Low	Quarter step
High	High	Low	Eighth step
High	High	High	Sixteenth step

TABLEAU 4.1 - Tableau de microstepping

FIGURE 4.6 – Pompe à vide

— Tension nominale : DC 5V

— Courant à vide : 0.35A

— Intervalle de pression : 400-650mmhg

— Masse : 60g

L'electrovanne (figure 4.7) est le composant qui permet d'attraper ou libérer l'objet, elle est caractérisée par :

— Tension nominale : DC 12V

— Courant nominal : 200mA

Finalement, la ventouse qui se compose d'une partie fixe et une partie en mouvement grâce au ressort qui minimise la possibilité d'accidents. On trouve aussi un objet en silicone qui permet de "copier" la forme de l'objet qu'on souhaite attraper et donc empêcher l'air de s'échapper (voir figure 4.8)

FIGURE 4.7 – Electrovanne

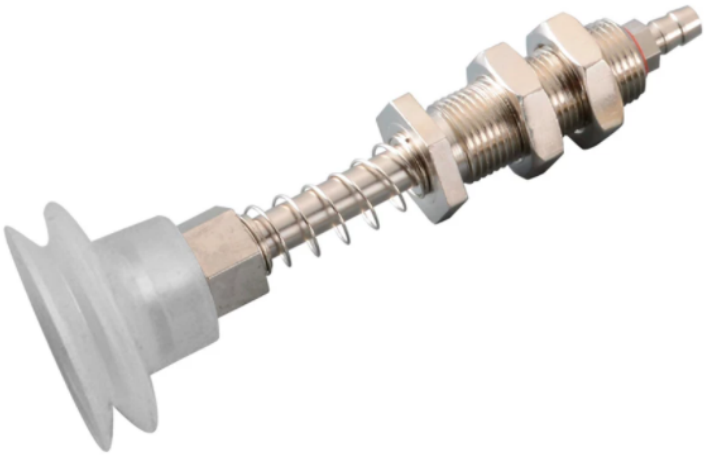

FIGURE 4.8 – Ventouse

4.4 Mouvement de translation

Pour générer le mouvement de translation, on utilise le système de **vis-écrou**, ce système est largement utilisé dans les imprimantes 3D, les CNC, les moteurs DC linéaire, etc. Le principe est simple, on fait tourner une tige filetée et on met des contraintes sur l'écrou de tel sorte que le seul degré de liberté restant est la translation. La tige filetée est caractérisée par un paramètre appelé **pitch**, il correspond à la distance parcourue par l'écrou si on fait tourner la tige par une révolution. et donc la distance parcourue est donnée par la formule suivante :

$$z = \frac{\theta}{360}\, p \tag{4.1}$$

θ étant l'angle de rotation de la tige filetée et p etant son pitch.

4.5 Modélisation 3D

La modélisation 3D du robot a été faite en utilisant le logiciel **Autodesk Fusion 360**. Les raisons qui nous ont poussées à utiliser ce logiciel sont :

- Il est très puissant, comme il offre plusieurs espace de travail dans le même logiciel, notamment la conception, animation, simulation (résistance des matériaux), fabrication (génération de code G pour les machines CNC et imprimantes 3D), etc.
- Il est toujours en développement, des mises à jours mensuelles sont exigées aux utilisateurs.
- Il est Cross-platform, i.e il marche dans les systèmes d'exploitation les plus connus (Windows, Linux, macOS).
- Il prend en charge la modélisation paramétrique, c'est à dire on peut modéliser une pièce en se basant sur des équations et des variables.

On peut citer quelques défauts de ce logiciel, entre autres :

- Il est payant, bien qu'il existe une licence pour l'utilisation personnelle mais qui n'offre pas tous ses outils.
- Il est cloud-based, donc on doit avoir accès à internet.
- Les mises à jour mensuelles sont faites automatiquement et donc on peut pas choisir une version qui nous convient.

4.6 Impression 3D

Après avoir modéliser le robot, on doit concevoir ses pièces par impression 3D. Pour cela on doit utiliser un logiciel de découpage en tranches ou slicer, ce dernier convertir un objet 3D en des instructions instructions spécifiques pour l'imprimante, i.e du fichier STL au fichier g-code. Le slicer qu'on a utilisé est **Ultimaker Cura**, voici quelques paramètres utilisés :

- Profile : 0.2mm
- Infill Density : 30
- Infill Pattern : Cubic Subdivision
- Nozzle Temperature : 200°C
- Build Plate Temperature : 200°C
- Support : Everywhere
- Build Plate Adhesion Type : Brim

Les pièces sont imprimées en **PLA** par une imprimante **Creality Ender 3**.

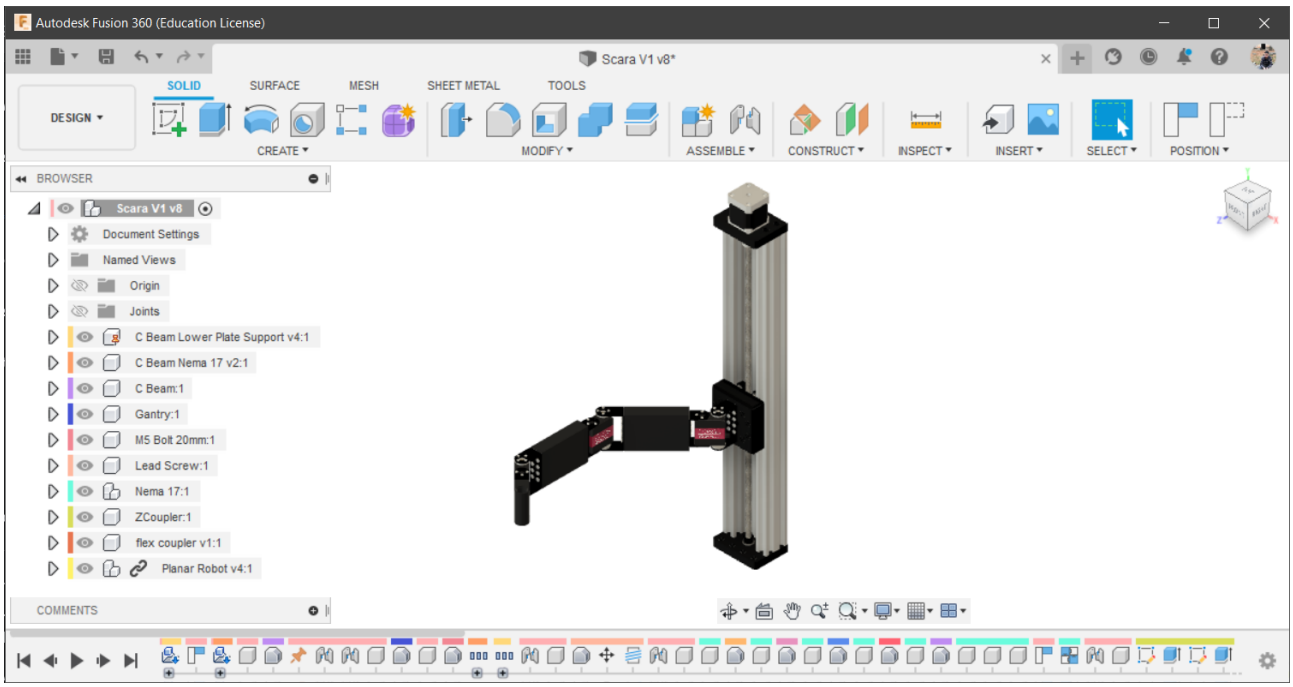

FIGURE 4.9 – Modélisation 3D dans Fusion 360

4.7 Assemblage

Après impression des pièces en PLA, on doit passer à l'assemblage. On a utilisé des **Heat inserts** pour assurer la rigidité au robot et pour empêcher l'usure de filetage , On utilise un fer à souder et on presse verticalement sur ce vis, le pourtour du trou va s'échauffer et donc l'insert et la pièce vont se coller, voir figure 4.13.
Après avoir terminé l'assemblage, on aura la forme finale du robot, illustrée dans la figure 4.14.

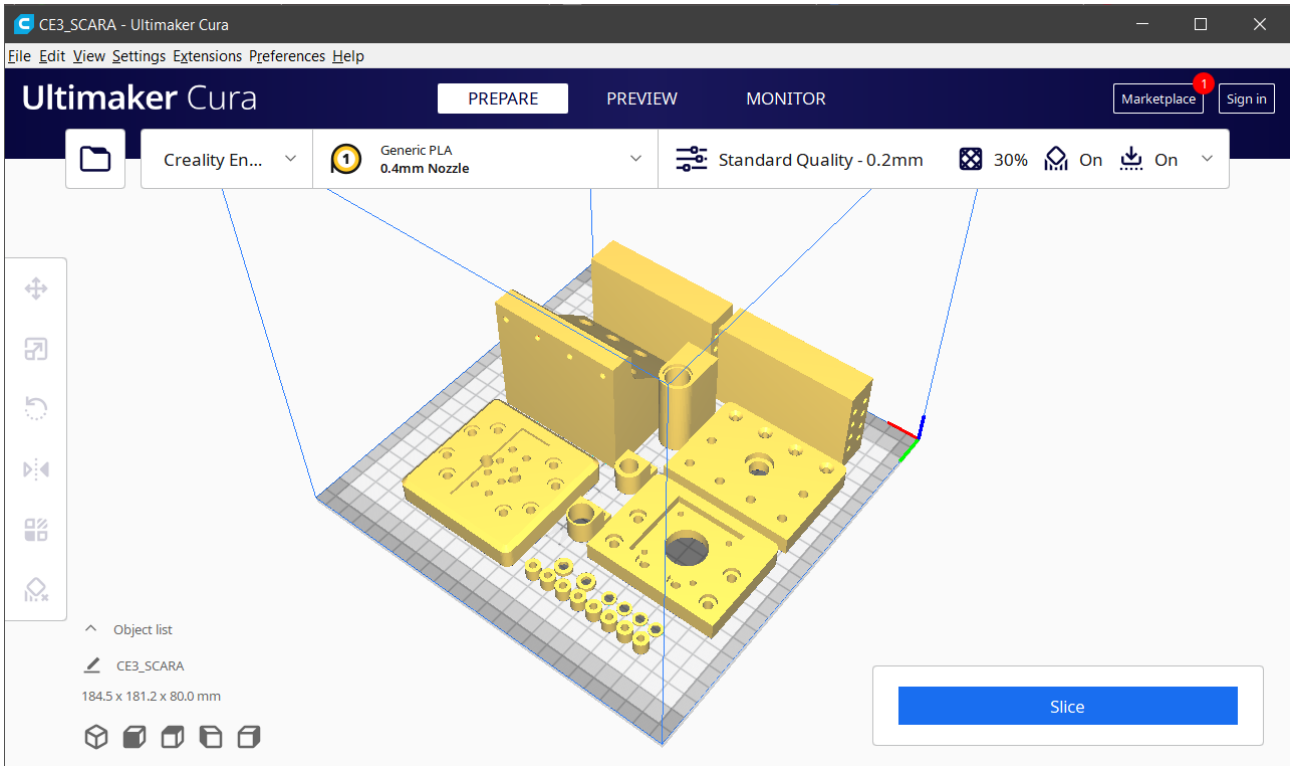

FIGURE 4.10 – l'ensemble de pièces à imprimer

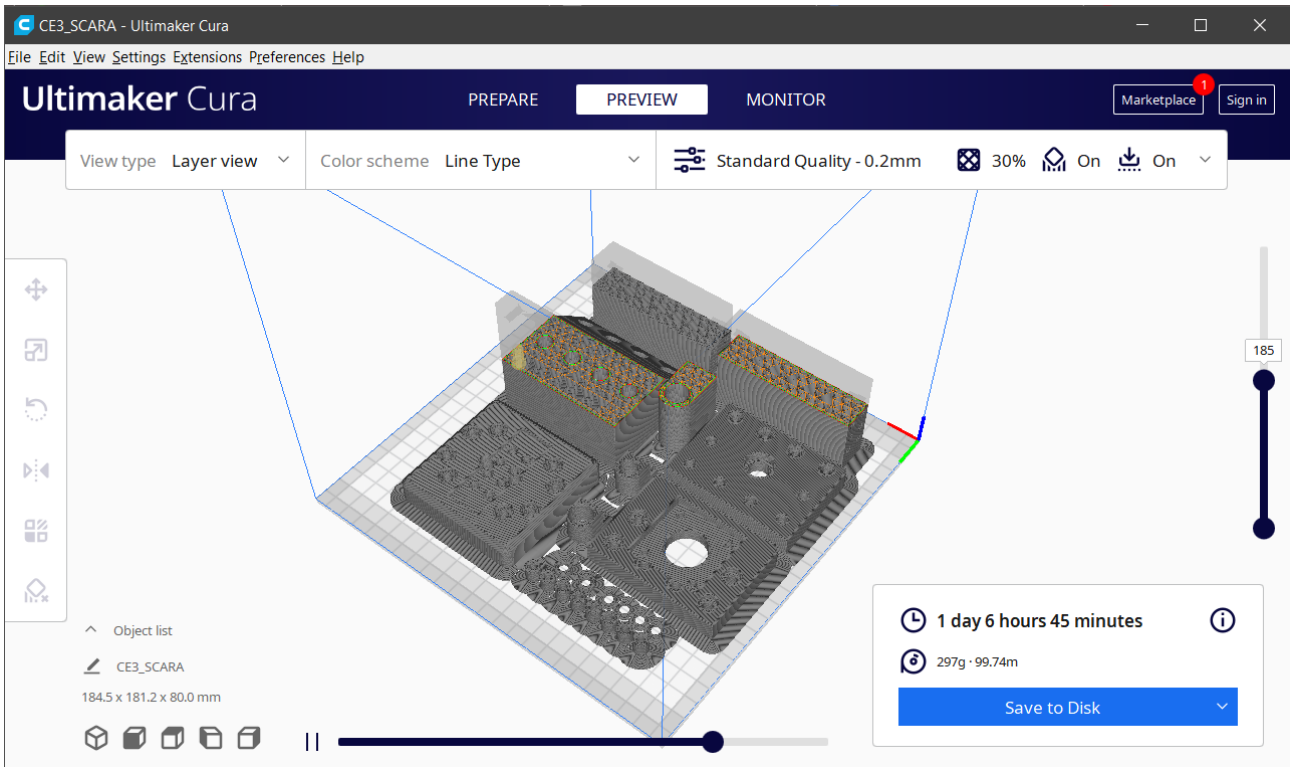

FIGURE 4.11 – Logiciel de découpage en tranches - Ultimaker Cura

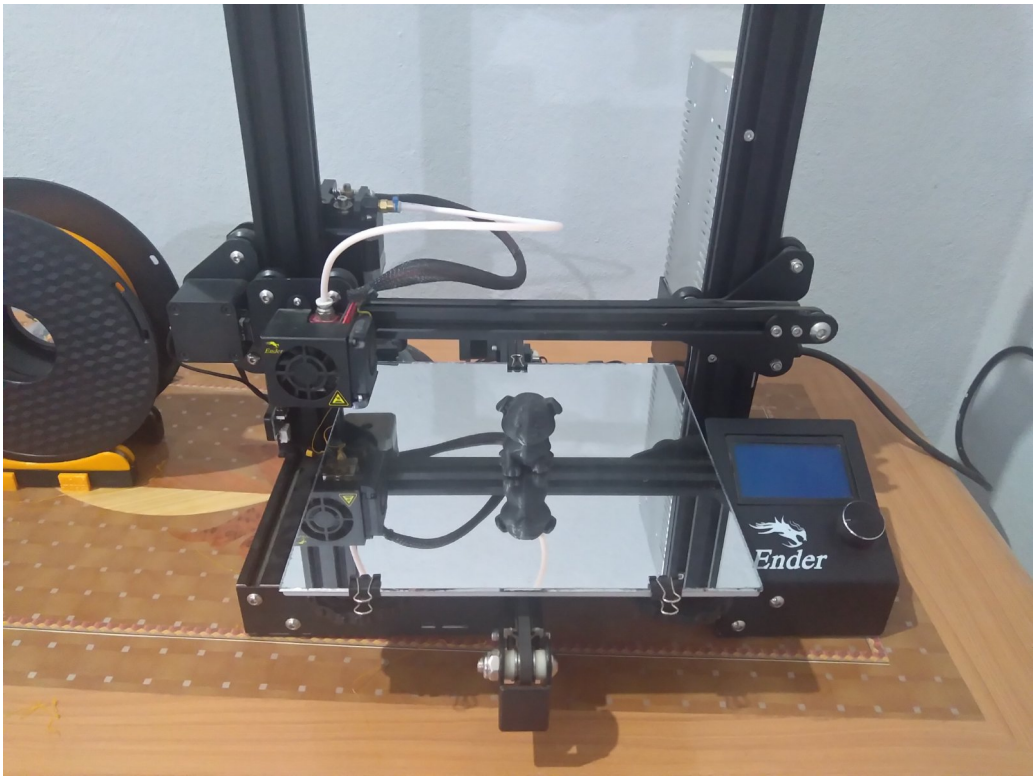

FIGURE 4.12 – Imprimante 3D Creality Ender 3

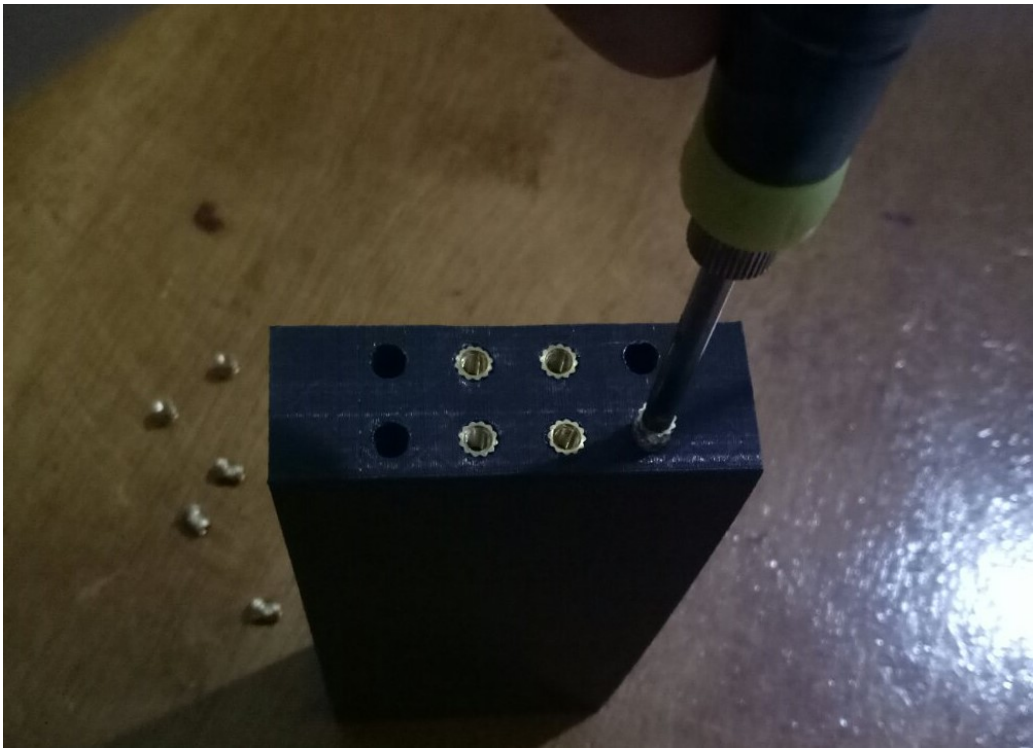

FIGURE 4.13 – Utilisation des Heat Inserts

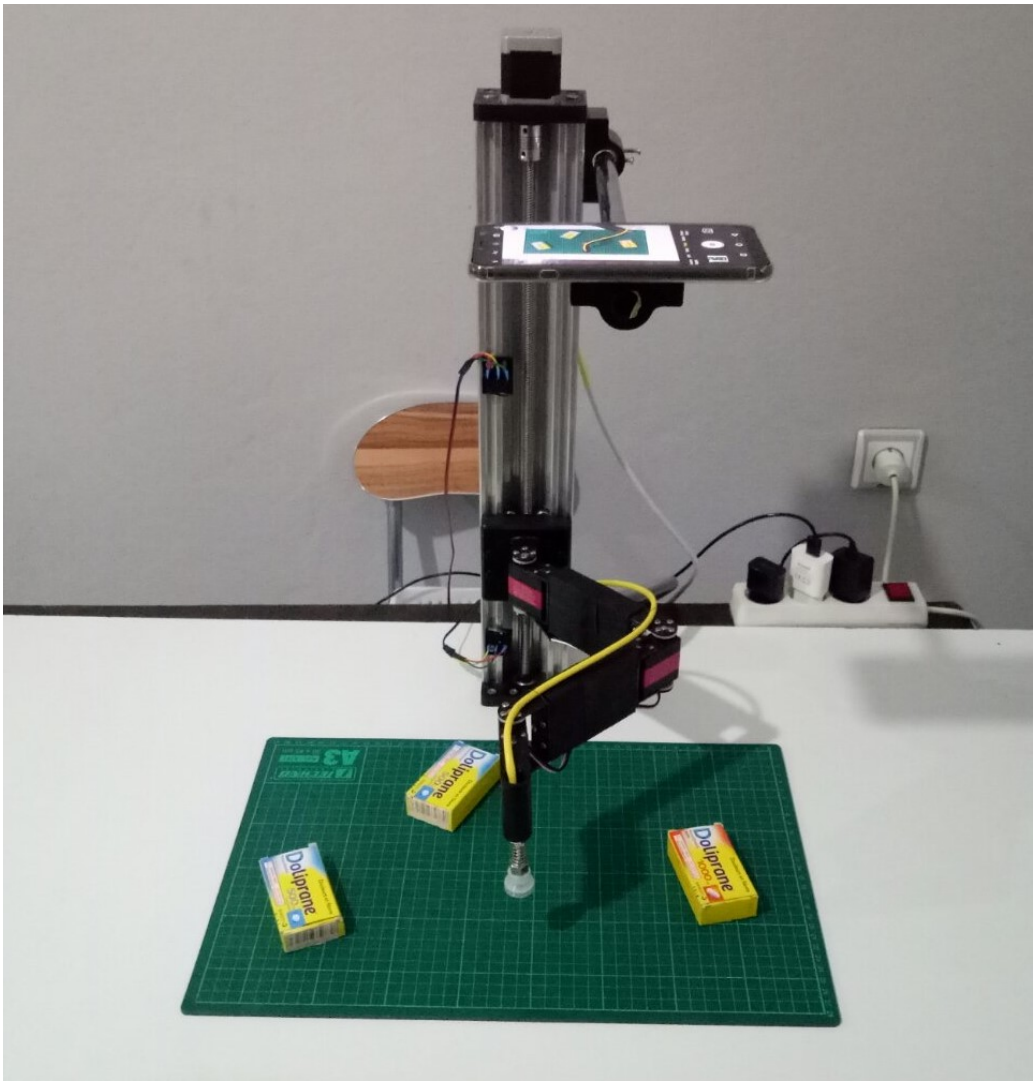

Figure 4.14

Conclusion générale et perspectives

Dans ce travail nous avons rappelé la modélisation géométrique et cinématique du robot SCARA. Nous avons aussi traité le problème de génération des trajectoires pour les robots manipulateurs dans l'espace articulaire. Et finalement, nous avons abordé les bases du traitement d'image utilisé pour la vision par ordinateur. Nous avons réaliser le robot SCARA à 4 degrés de libertés
On peut améliorer ce travail par l'utilisation de l'intelligence artificielle notamment les réseaux de neurones convolutifs (CNN).
Traiter le problème du génération de trajectoires dans l'espace de travail qui est vachement plus difficile.
Utilisation des algorithmes d'évitement d'obstacle comme la théorie des champs potentiels artificiels.

Bibliographie

[1] Kevin M. Lynch and Frank C. Park. *Modern Robotics : Mechanics, Planning, and Control.* Cambridge University Press, 2017.

[2] Sciavicco, Lorenzo, Siciliano, Bruno. *Modelling and Control of Robot Manipulators*

[3] Etienne Dombre and Wisama Khalil. *Modeling, Performance Analysis and Control of Robot Manipulators*

[4] John J. Craig *Introduction to Robotics Mechanics and Control*

[5] Richard M. Murray, Zexiang Li, S. Shankar Sastry. *A Mathematical Introduction to Robotic Manipulation*

[6] R. Kelly, V. Santibáñez and A. Loría *Control of Robot Manipulators in Joint Space*

[7] Peter Corke *Robotics, Vision and Control Fundamental Algorithms in MATLAB*

[8] Claudio Melchiorri, Luigi Biagiotti *Trajectory Planning for Automatic Machines and Robots*

[9] Berthold K.P. Horn *Robot Vision*

[10] Richard Szeliski *Computer Vision : Algorithms and Applications*

[11] David Forsyth *Computer Vision : A Modern Approach*

[12] Gary Bradski and Adrian Kaehler *Learning OpenCV*

[13] Rafael C. GONZALES and Richard E. Woods *Digital Image Processing*