

UNIVERSITÉ DE SHERBROOKE
Faculté de génie
Département de génie robotique

Rapport APP 5
Modélisation géométrique et algèbre linéaire

Présenté à :
Alexandre Girard

Présenté par :
Équipe numéro 1
Émile Renaud – Rene0901
Maxence Mougin – Moum0601

Sherbrooke – 17 Mars 2020

Contents

Table des figures	3
Table des tableaux	3
0. Trouver le point P_o	4
1. Angle de la tranche.....	5
2. Hauteur de h_g	7
3. Zone critique.....	8
4. Tracer la pièce.....	8
5. Positions des défauts.....	9
6. Défauts dans la zone.....	11
7. Cinématique différentielle.....	11

Table des figures

Figure 1 : Repère Po et To.....	4
Figure 2 : Code vecteur bras	4
Figure 3 : Code vecteur v_t_{PwT}	5
Figure 4 : Code point P.....	5
Figure 5 : Code angle nominal	5
Figure 6 : Code Position Pente.....	6
Figure 7 : Code v_w_{TiW}	6
Figure 8 : Code distance P - Ti.....	6
Figure 9 : Code Point Ti en base P.....	6
Figure 10 : Calcul moindres-carrés angle.....	7
Figure 11 : Représentation ordonné à l'origine pente	7
Figure 12 : Matrice de point de la pièce	8
Figure 13 : Code afficher ellipse.....	9
Figure 14 : Graphique de la pièce avec les défauts et zone critique	9
Figure 15 : Code points défauts en W.....	10
Figure 16 : Code distance entre défauts et P.....	10
Figure 17 : Code mettre les défauts en base P	10
Figure 18 : Code vérification des points dans la zone	11
Figure 19 : Code pseudo-inverse vitesse	12

Table des tableaux

Tableau 1 : Paramètres ellipse.....	8
Tableau 2 : Position des défauts dans la pièce	10
Tableau 3 : Vitesses des joints, première série d'angles	12
Tableau 4 : Vitesses des joints, deuxième série d'angles	12

0. Trouver le point Po

Avant de pouvoir répondre à la problématique avec les défauts et la pente de la tranche, on doit savoir où se trouve le point Po. Nous avons cru bon de pouvoir mettre le point Po au bout du robot. Pour ce faire nous avons besoin de la distance en Po et To. Pour trouver cette distance, nous avons le point Po et la position des joints quand le robot récupère une pièce :

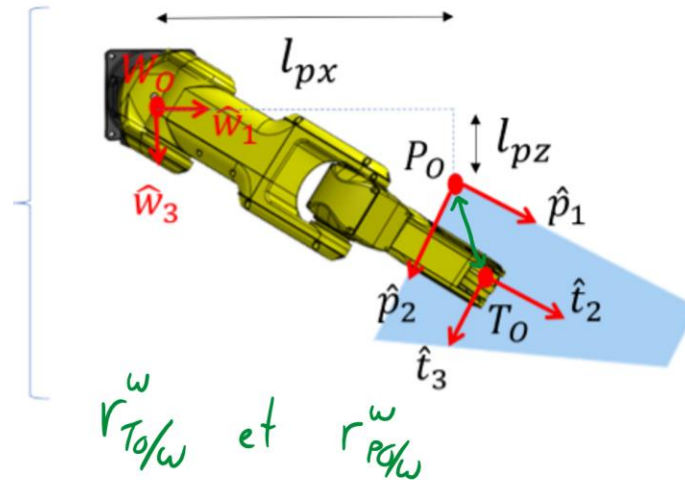


Figure 1 : Repère Po et To

Le vecteur v_{w_PwW} est connu quand le bras récupère une pièce avec l_{px} et l_{pz} .

Il manque à trouver le point To. Nous l'avons trouvé à modélisant tous les vecteurs du bras robotisé :

```
73      %----- Vecteurs / Section du bras -----%
74      v_w_AwW = [0,0.15,0]';
75      v_w_BwW = wRa * [0.05,0.1,0]' + v_w_AwW;
76      v_w_CwW = wRa * aRb * [0,0.5,0]' + v_w_BwW;
77      v_w_DwW = wRa * aRb * bRc * [0.1,0.02,0]' + v_w_CwW;
78      v_w_EwW = wRa * aRb * bRc * cRd * [0.3,0,0]' + v_w_DwW;
79      v_w_TwW = wRa * aRb * bRc * cRd * dRe * [0.02,0,0]' + v_w_EwW;
```

Figure 2 : Code vecteur bras

Pour nous aider, nous avons mis tous les vecteurs en fonction de W et en base W. À l'aide des matrices de rotation¹ et des longueurs fournies dans la problématique, nous avons pu trouver le vecteur v_{w_TwW} . Les autres vecteurs ne sont pas obligatoires, mais nous les avons tous calculés pour être capables de bien les tracer dans un graphique. Une fois

¹ Les matrices de rotation sont toutes présentes dans le code source MatLab

la position de To connu quand les angles du robot sont ajustés à la position où il récupère des pièces, nous pouvons calculer le vecteur entre Po et To en base W :

```
v_w_PwT = [0.5994,0,0.1991]' - v_w_TwW;  
v_t_PwT = inv(eRt) * inv(dRe) * inv(cRd) * inv(bRc) * inv(aRb) * inv(wRa) * v_w_PwT;
```

Figure 3 : Code vecteur v_t_PwT

Ce vecteur qui représente la distance entre P et T en base T est calculer dans une autre fonction pour être sûr que les angles du robot soient bien ajustés.

Nous pouvons alors trouver le notre point P en fonction de W et en base W. comme pour tous les autres segments du bras, nous multiplions la matrice de rotation de T à W au vecteur P à T en base T et ajoutons la translation requise.

```
81 % point P  
82 v_w_PwW = wRa * aRb * bRc * cRd * dRe * eRt * find_v_t_PwT() + v_w_TwW;
```

Figure 4 : Code point P

Maintenant que nous avons notre point P par rapport à W en base W, nous pouvons commencer à situer les défauts et les points de la tranche sur la pièce. De plus, nous avons tous les vecteurs nécessaires pour tracer notre bras robotisé.

1. Angle de la tranche

On doit trouver si la tranche respecte l'angle nominal. Avant de trouvé l'angle de la tranche nous avons commencé par trouver l'angle nominal avec une simple relation trigonométrique avec les longueurs fournies dans la problématique:

```
% trouver angle nominal  
theta_nom = atand((hg-hd)/lb);
```

Figure 5 : Code angle nominal

Nous avons trouvé un angle nominal de 18.4349°.

Pour trouver l'angle de la tranche avec les points de la caméra, nous avons la distance entre les points et la caméra dans la base de la caméra :

```

19      % Position Pente
20      Ti1_v_Ti1wV = [0.158920,0.013914,0.028686]';
21      Ti2_v_Ti2wV = [0.157470,0.021067,0.008891]';
22      Ti3_v_Ti3wV = [0.153781,0.039266,-0.040587]';
23      Ti4_v_Ti4wV = [0.152420,0.04597,-0.060395]';
24      Ti5_v_Ti5wV = [0.150931,0.053326,-0.080185]';

```

Figure 6 : Code Position Pente

Nous avons commencé par convertir ces vecteurs par rapport à W et en base W :

```

117      %----- Points tranche -----%
118      % Mettre les points en base W par rapport à W
119      v_w_Ti1wW = wRv * Ti1_v_Ti1wV + [0.8,0.7,0]';
120      v_w_Ti2wW = wRv * Ti2_v_Ti2wV + [0.8,0.7,0]';
121      v_w_Ti3wW = wRv * Ti3_v_Ti3wV + [0.8,0.7,0]';
122      v_w_Ti4wW = wRv * Ti4_v_Ti4wV + [0.8,0.7,0]';
123      v_w_Ti5wW = wRv * Ti5_v_Ti5wV + [0.8,0.7,0]';

```

Figure 7 : Code v_w_TiwW

Par la suite nous avons trouvé la distance entre les points et le point P (origine de la pièce) :

```

125      % Trouver distance entre P et Ti en base W
126      v_w_T1wP = v_w_Ti1wW - v_w_PwW;
127      v_w_T2wP = v_w_Ti2wW - v_w_PwW;
128      v_w_T3wP = v_w_Ti3wW - v_w_PwW;
129      v_w_T4wP = v_w_Ti4wW - v_w_PwW;
130      v_w_T5wP = v_w_Ti5wW - v_w_PwW;

```

Figure 8 : Code distance P - Ti

Nous utilisons le vecteur v_w_PwW que nous avons trouvé dans une autre étape.

Finalement nous avons ramener la distance entre nos points de la tranche et P en base P pour être capable de les utiliser efficacement :

```

132      % Mettre le vecteur en base P
133      v_p_T1wP = pRw * v_w_T1wP;
134      v_p_T2wP = pRw * v_w_T2wP;
135      v_p_T3wP = pRw * v_w_T3wP;
136      v_p_T4wP = pRw * v_w_T4wP;
137      v_p_T5wP = pRw * v_w_T5wP;

```

Figure 9 : Code Point Ti en base P

Maintenant que tous nos points de la tranche sont utilisables, nous avons fait la technique des moindres carrés pour trouver l'équation de la pente de la tranche. Nous avons utilisé cette technique, car nous avons plus d'équations que de variable à trouver. On cherchait une formule $y = a*x+b$ alors on a ajouté une colonne de 1 dans la matrice pointTranche_X (ligne 142) . Le résultat de sortie « xb » est une matrice 2x1 qui contient la pente et l'ordonnée à l'origine de notre série de points.

```

139 % Stocker les points dans une matrice
140 matricePointTranche = [v_p_T1wP,v_p_T2wP,v_p_T3wP,v_p_T4wP,v_p_T5wP];
141 % Faire une matrice avec une colonne de X et une colonne de 1
142 pointTranche_X = [matricePointTranche(1,1:end)',ones(1,5)'];
143 % Matrice avec une colonne de Y
144 pointTranche_Y = matricePointTranche(2,1:end)';
145
146 % Trouver l'angle de la tranche
147 xb = inv(pointTranche_X' * pointTranche_X) * pointTranche_X' * pointTranche_Y;
148 theta_tranche = abs(atan(xb(1))); % mettre en abs pour avoir le positif

```

Figure 10 : Calcul moindres-carrés angle

Pour trouver l'angle, on fait simplement un arctan de notre pente. On peut alors trouver la différence entre l'angle nominal et l'angle mesuré sur la tranche. Nous obtenons un angle de 18.5857° ce qui nous donne une différence de 0.1507° . Cette pièce entre dans les tolérances demandées.

2. Hauteur de h_g

À l'aide de notre la technique des moindres-carrés à l'étape précédente, nous avons trouvé la pente et l'ordonnée à l'origine de la fonction que représente les points mesurés sur la tranche. On peut constater que notre ordonné à l'origine est directement la hauteur h_g recherché :

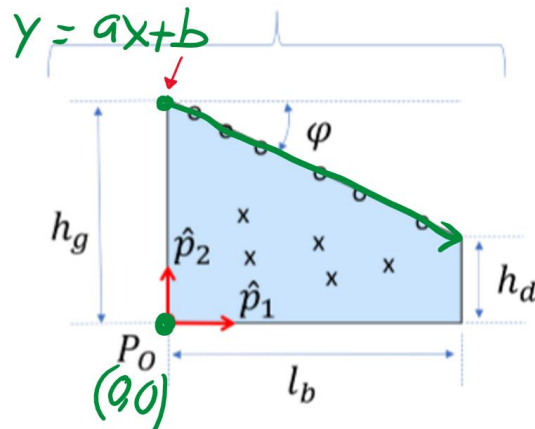


Figure 11 : Représentation ordonné à l'origine pente

On obtient une valeur de 0.1013m. Nous avons donc une différence de 0.0013m, ce qui ne rentre pas dans les tolérances de +/- 1mm.

3. Zone critique

Il est possible de trouver que l'inéquation que nous avons nous donne une ellipse, car les valeurs propres que l'on en retire sont non nul et de mêmes signes. Par manque de temps, nous n'avons pas fait tous les calculs à la main pour trouver les paramètres de l'ellipse. Nous avons trouvé sur l'interweb un site qui nous donne tous les paramètres : [lien vers le site](#).

Tableau 1 : Paramètres ellipse

Centre	(0.11 ; 0.03)
Longueur grande axe	0.06
Longueur petite axe	0.04
F ₁	(0.08879 ; 0.03707)
F ₂	(0.131213 ; 0.2293)

4. Tracer la pièce

Pour tracer la pièce en 2D on doit s'assurer d'avoir nos mesures par rapport au point P et aussi en base P. Les dimensions nominales données dans la problématique sont déjà correctes. On les a mis dans une matrice pour faciliter l'affichage avec la fonction `plot` :

```
184 — point_piece = [0,0,lb,lb,0;  
185                  0,hg,hd,0,0];
```

Figure 12 : Matrice de point de la pièce

Pour la zone critique, nous avons assumé qu'elle était exprimée par rapport à P et en base P. Nous commençons par créer une fonction anonyme avec notre inéquation et ensuite nous utilisons la fonction `fcontour` pour afficher le contour de notre fonction :


```

158 % Afficher inégalité
159 figure(2);
160 zone_critique = @(x,y) 45*x.^2 + 30*x.*y + 85*y.^2 - 10.8*x - 8.4*y + 0.684
161 fcontour(zone_critique,'LineWidth',1,'LineColor','r','LevelList',0);
162 axis([-0.02 0.16 -0.05 0.16]);
163 grid on;
164 hold on;

```

Figure 13 : Code afficher ellipse

Avec la fonction plot on peut facilement afficher le contour de la pièce, la zone critique et les défauts trouvés. Les points des défauts sont trouvés à l'étape 5.

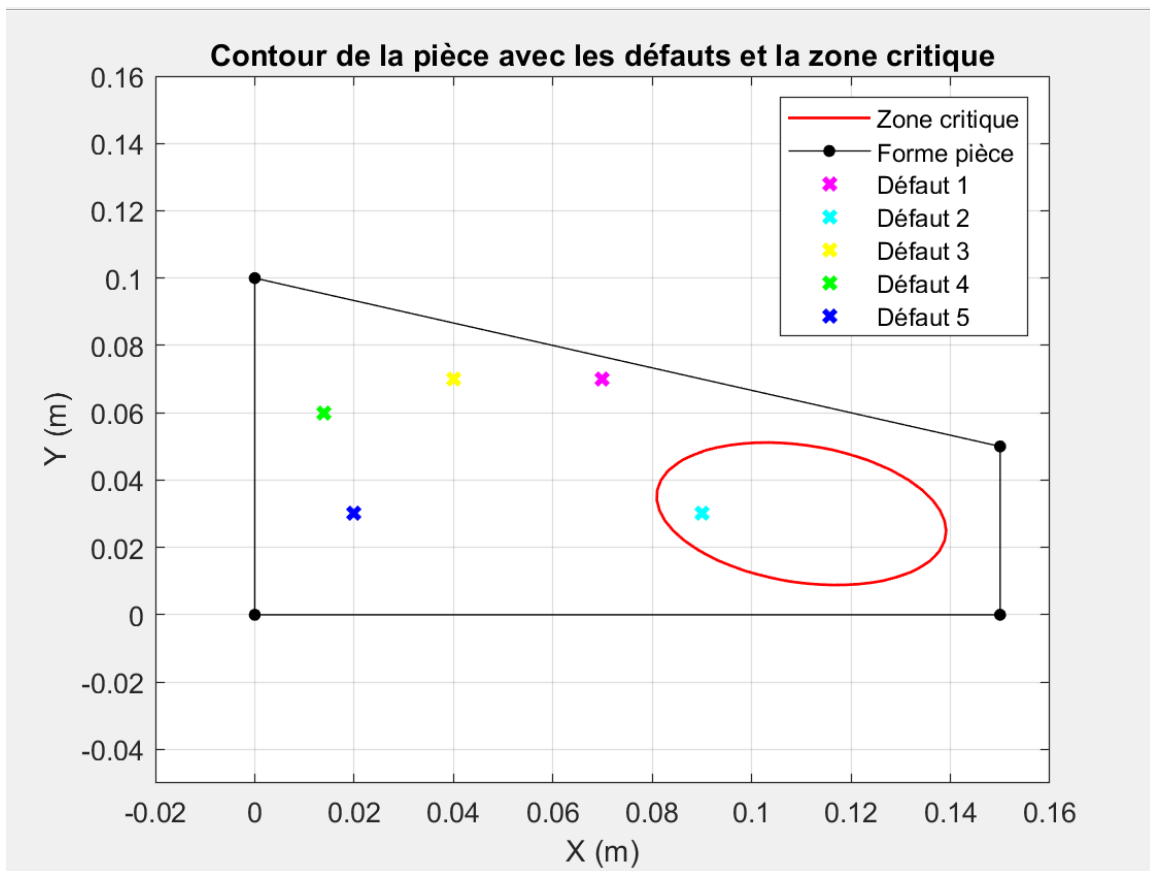


Figure 14 : Graphique de la pièce avec les défauts et zone critique

5. Positions des défauts

Avec le même principe que les points sur la tranche, nous sommes capables facilement de situer les défauts sur la pièce.

Nous commençons par mettre nos défauts par rapport W, en base W :

```

96      % Mettre les défauts par rapport W en base W
97      v_w_D1wW = wRv * Df1_v_Df1Wv + [0.8,0.7,0]';
98      v_w_D2wW = wRv * Df2_v_Df2Wv + [0.8,0.7,0]';
99      v_w_D3wW = wRv * Df3_v_Df3Wv + [0.8,0.7,0]';
00      v_w_D4wW = wRv * Df4_v_Df4Wv + [0.8,0.7,0]';
01      v_w_D5wW = wRv * Df5_v_Df5Wv + [0.8,0.7,0]';

```

Figure 15 : Code points défauts en W

Ensuite on peut trouver la distance entre les défauts et le point P :

```

103     % Trouver distance entre les défauts et P en base W
104     v_w_D1wP = v_w_D1wW - v_w_PwW;
105     v_w_D2wP = v_w_D2wW - v_w_PwW;
106     v_w_D3wP = v_w_D3wW - v_w_PwW;
107     v_w_D4wP = v_w_D4wW - v_w_PwW;
108     v_w_D5wP = v_w_D5wW - v_w_PwW;

```

Figure 16 : Code distance entre défauts et P

Finalement nous pouvons les mettre en base P pour avoir des mesures qui ont du sens sur le graphique :

```

110     % Mettre les défauts par rapport a P et base P
111     v_p_D1wP = pRw * v_w_D1wP;
112     v_p_D2wP = pRw * v_w_D2wP;
113     v_p_D3wP = pRw * v_w_D3wP;
114     v_p_D4wP = pRw * v_w_D4wP;
115     v_p_D5wP = pRw * v_w_D5wP;

```

Figure 17 : Code mettre les défauts en base P

On stock les points dans une matrice pour faciliter l'affichage :

Tableau 2 : Position des défauts dans la pièce

```

MatriceDefauts =

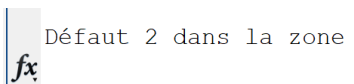
    0.0700    0.0900    0.0400    0.0140    0.0200
    0.0700    0.0300    0.0700    0.0600    0.0300
   -0.0000   -0.0000   -0.0000   -0.0000   -0.0000

```

On remarque que la rangée Z est à 0, ça nous aide à valider que nos défauts sont bien convertis et qu'ils sont à la surface de la pièce.

6. Défauts dans la zone

Avec les positions des défauts par rapport à P, en base P et l'équation de la zone, nous sommes en mesure de calculer si les défauts se trouvent dans la zone. On commence par mettre toutes les X et tous les Y des points dans des matrices qui ont des noms moins longs pour faciliter la compréhension. Ensuite on fait le calcul pour chaque X,Y et on stocke le résultat dans une matrice *result*. Finalement il suffit de regarder si le résultat est inférieur à 0. On peut constater que le programme affiche dans le terminal que le défaut 2 est dans la zone :



On peut aussi vérifier sur le graphique que le défaut 2 est dans la zone.

```
169 % Verifier si les points sont dans la zone
170 xd = MatriceDefauts(1,1:end);
171 yd = MatriceDefauts(2,1:end);
172 result = 45*xd.^2 + 30*xd.*yd + 85*yd.^2 - 10.8*xd - 8.4*yd + 0.684;
173 if result(1) < 0
174     disp('Défaut 1 dans la zone');
175 elseif result(2) < 0
176     disp('Défaut 2 dans la zone');
177 elseif result(3) < 0
178     disp('Défaut 3 dans la zone');
179 elseif result(4) < 0
180     disp('Défaut 4 dans la zone');
181 elseif result(5) < 0
182     disp('point 5 dans la zone');
183 end
```

Figure 18 : Code vérification des points dans la zone

7. Cinématique différentielle

Pour la situation A, nous avons pris les informations fournies dans la problématique pour compléter l'équation suivante pour faire de la cinématique différentielle inverse :

$$\dot{q} = J^{-1}(\dot{q}) \cdot \dot{r}$$

Où \dot{q} est une matrice 6x1 avec les vitesses des joints du robot, J est la matrice Jacobienne du robot et \dot{r} est la vitesse cartésienne. On nous demande aussi de minimiser les vitesses des joints. Pour se faire, on utilise la méthode pseudo-inverse, car elle permet de minimiser le carré de la norme.

$$J^{-1} = J^T \cdot (J \cdot J^T)^{-1}$$

Pour inverser notre jacobienne, on trouve la première section de la méthode pseudo-inverse.

$$\dot{q} = J^T \cdot (J \cdot J^T)^{-1} \cdot \dot{r}$$

Pour la situation B et C, la matrice Jacobienne est plus complexe. On arrive avec une matrice 3x6. On dérive les équations en fonction des thêtas pour remplir notre matrice. Elle se trouve dans le fichier *cinematique_diff.m* dans les fichiers sources MatLab. Dans la situation B nous voulions seulement un mouvement vertical de 1m/s c'est pourquoi on met des 0 en X et Z.

```
34 % calcul direct
35 vit_angulaire_B_direct = pinv(j_q_b) * [0,1,0]';
```

Figure 19 : Code pseudo-inverse vitesse

Pour la situation C, comme le professeur a dit pendant son procédural, on ne peut pas bouger seulement sur 1 axe si on a un bras qui a un degré de liberté. Par contre, quand on l'essaie dans MatLab, ça semble fonctionner... On ne comprend pas pourquoi et on va avoir besoin d'une explication d'un pro!

Voici les résultats obtenus pour les situations A,B et C pour la première série d'angle :

Tableau 3 : Vitesses des joints, première série d'angles

Joints	A	B	C
#1	0	-0.0000	0
#2	1.4879	1.8278	1.4628
#3	0.3866	-0.9219	0
#4	0	0	0
#5	0	0	0
#6	0	0	0

Voici les résultats obtenus pour les situations A,B et C pour la deuxième série d'angle :

Tableau 4 : Vitesses des joints, deuxième série d'angles

Joints	A	B	C
#1	1.0e+04 * 0	0	0
#2	-0.7703	1.2340	-0.0000
#3	-0.7703	-2.7747	0
#4	0	0	0
#5	0	0	0
#6	0	0	0