Modélisation d'un chariot élévateur et optimisation de son parcours dans un hangar

---

Rapport TX-TIPE

Anthony Tordjmann, Roma Auguste

printemps 2019

Sommaire

Introduction

0.1 Problématique

La problématique initiale était la suivante : comment optimiser le trajet d'un chariot élévateur dans un entrepôt sans qu'il ne se retourne ou ne renverse les marchandises qu'il transporte ? Nous avons choisi de dérouler cette problématique jusqu'à isoler plusieurs problèmes précis. Ces problèmes sont interdépendants, mais leur résolution demande des méthodologies différentes.

Nous avons choisi de résoudre un problème d'optimisation, consistant à minimiser le temps de parcours d'une trajectoire dont seuls certains points sont définis. Cela signifie qu'il faut déterminer un chemin à coût minimal : c'est un problème de cheminement. Des algorithmes connus et prouvés permettent de résoudre de trouver des chemins optimaux étant donné des coûts, comme nous le verrons en 2. Cependant, il reste le problème de déterminer la fonction de coût en temps.

L'approche la plus simple est de faire l'hypothèse que la vitesse est constante, donc que le coût en temps est égal au coût en distance. Cependant, cette approche est très insatisfaisante car les chariots élévateurs ne sont pas très stables :

- Ils peuvent tomber dans un virage, si leur vitesse est trop élevée
- Ils peuvent faire tomber leur colis au cours d'un freinage, si leur décélération est trop importante

Le second point fait qu'il faut beaucoup de temps aux chariots pour décélérer. Par conséquent, la longueur d'un segment de droite n'est pas directement proportionnelle à la vitesse max qui pourra être atteinte ; on ne peut "profiter de la ligne droite" car on ne peut accélérer sensiblement qu'à partir d'une certaine distance de voie libre pour freiner. Le premier point fait que la vitesse du chariot élévateur dans un virage est sensiblement diminuée, ce qui fait qu'à distance égale, le temps nécessaire pour parcourir un segment de droite et une succession de virage est sensiblement différent.

Pour ces raisons, il est absolument nécessaire de modéliser le comportement du chariot sur certaines trajectoires. Les objectifs dégagés sont donc :

- Déterminer la vitesse max du chariot garantissant :
  - Un non-renversement du chariot en virage
  - Un non-glissement du colis transporté en ligne droite
- Déterminer le coût en temps d'une trajectoire donnée
- Sélectionner la meilleure trajectoire (en temps) passant par des points imposés

C'est la résolution de ces problèmes qui sera abordée dans ce rapport.

0.2 Remerciements

Nous tenons à remercier M. Jean-Luc DULON (Maître de Conférences Roberval FRE UTC-CNRS 2012 jean-luc.dulong@utc.fr) ;
et Mme. Delphine VIANDIER (auteur d'ouvrages et professeur de physique-chimie MP, lycée Pierre d'Ailly Compiègne delphine.viandier@worldonline.fr),
pour leur suivi pédagogique tout au long de ce projet. Nous remercions également le lycée Pierre d'Ailly qui a financé notre prototype mécanique.

0.3 Auteurs

Anthony TORDJMANN, étudiant en MP au lycée Pierre d'Ailly de Compiègne (anthony.tordjmann@
Romain MALIACH-AUGUSTE, étudiant en TC04 à l'Université de Technologie de Compiègne
(romain.maliach-auguste@etu.utc.fr)

1 Calculs de statique et de dynamique

Le but est de déterminer une vitesse max en virage sans renversement du chariot ; et une vitesse max en ligne droite garantissant une décélération sans glissement du colis.

Pour cela, nous utiliserons des équations de dynamique, ce qui suppose de modéliser le chariot. Nous ferons donc des hypothèses sur sa forme géométrique et la répartition de sa masse. Ensuite, nous avons obtiendrons par des théorèmes les équations attendues : vitesse max en ligne droite, vitesse max en virage. Enfin, nous vérifierons l'adéquation de ces résultats avec des mesures sur un modèle physique de chariot que nous avons construit (nous l'appelerons le prototype).

Mais la validation des hypothèses sur le chariot élévateur par des mesures sur notre prototype mécanique nécessite de supposer que notre prototype mécanique est lui-même un bon modèle qui se comporte comme un chariot élévateur. Elle permet en fait de généraliser l'ensemble des productions de notre TX (modèles mécaniques et numériques, voir 3) à tous les chariots élévateurs, et à tous les véhicules verticaux, fins et avec un centre de masse haut. Elle pourrait être appuyée par des arguments théoriques, mais le seul argument que nous fournirons est le fait que toutes les dimensions sont du même ordre de grandeur. Cette hypothèse est très forte, mais elle n'est pas critique : si elle n'était pas vraie, cela signifierait que nos productions ne seraient valables que pour notre prototype.

1.1 Description de l'étude

Le cadre qu'on s'est fixé pour cette étude est fortement inspiré de [1].

- Le hangar est muni du repère $R_g = (O, \vec{x}_g, \vec{y}_g, \vec{z}_g)$.
- Le chariot (0) est animé d'un mouvement de rotation par rapport au sol dont le centre instantané de rotation est O.
- Le rayon de courbure de la trajectoire du point C dans R_g est R.
- Le repère lié à (0) est $R_0 = (O, \vec{x}_0, \vec{y}_0, \vec{z}_0)$ tel que $\vec{z}_0 = \vec{z}_g$ et on note $\theta = (\vec{x}_g, \vec{x}_0) = (\vec{y}_g, \vec{y}_0)$.
- Donc $\vec{OC} = R \cdot \vec{x}_0$. On remarque bien que R_0 est mobile par rapport à R_g.
- On considère que les frottements du colis (1) l'encastrent avec le chariot (0).
- Le repère lié au colis est $R_1 = (C, \vec{x}_1, \vec{y}_1, \vec{z}_1)$ et tant qu'il n'y a pas glissement du colis, $\vec{e}_1 = \vec{e}_0 \forall e \in \{x, y, z\}$.
- Le centre de gravité du colis est G tel que $\vec{CG} = e \cdot \vec{z}_1$. La masse du colis est m.
  - On donne son opérateur d'inertie en G : $\overline{\overline{I}}_A(1) = \begin{bmatrix} A & -F & -E \\ -F & B & -D \\ -E & -D & C \end{bmatrix}_{B0}$
- Les roues arrière (2) et (3) sont liées à (0) par des liaisons pivots d'axe $(C, \vec{x}_0)$.
- Les contacts entre les roues (2) et (3) et le hangar (R) ont lieu en A et B définis par $\vec{CA} = \frac{l}{2} \cdot \vec{x}_0 - r \cdot \vec{z}_0$ et $\vec{CB} = -\frac{l}{2} \cdot \vec{x}_0 - r \cdot \vec{z}_0$, r désignant le rayon des roues et l la voie arrière du chariot.
- Les contacts sont modélisés par des liaisons sphère-plan de centres A et B et de normale $\vec{z}_0$.

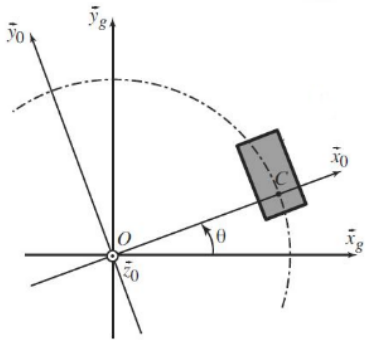

FIGURE 1 – Le virage du chariot paramétré

— Le contact dans ces liaisons se fait avec frottement et le coefficient de frottement est noté f (on supposera pour simplifier que les coefficients de frottement et d'adhérence sont identiques).
— Les actions mécaniques du hangar (R) sur les roues (2) et (3) sont modélisées dans le plan $(\vec{x}_0, \vec{y}_0)$ par des glisseurs en A et B de résultantes :

$$F(R \to 2) = T_A \cdot \vec{x}_0 + N_A \cdot \vec{z}_0 \text{ et } F(R \to 3) = T_B \cdot \vec{x}_0 + N_B \cdot z_0$$

— A part les liaisons entre (R) et (2) et entre (R) et (3), pour lesquelles le frottement est pris en compte, toutes les liaisons sont considérées parfaites.
— On néglige la masse et l'inertie de (0), (2) et (3) devant celle du colis (1). On note $E = 0 \cup 1 \cup 2 \cup 3$. L'accélération de la pesanteur est $\vec{g} = -g \cdot \vec{z}_0$.
— On se place dans un cas où le rayon de courbure R_C de la trajectoire du point C, ainsi que la vitesse V de ce point par rapport au référentiel R_g sont constants.
— Le contact entre le colis et le chariot est modélisé par une liaison plan à plan, avec un coefficient de frottement μ.

1.2 Calculs

Vitesse de G dans R_g

$$\vec{OG} = \vec{OC} + \vec{CG} = R_c \cdot \vec{x}_0 + e \cdot \vec{z}_1 = R_c \cdot \vec{x}_0$$

$$\vec{V}(G/R_g) = \left.\frac{d\vec{CG}}{d_t}\right|_{R_g} = \dot{\theta}\vec{z}_0 \wedge R_c\dot{\theta}\vec{y}_0 = -R_c\dot{\theta}^2\vec{x}_0$$

Avec $V = R_c\dot{\theta}$ on obtient :

$$\vec{V}(G/R_g) = V\vec{y}_0 \tag{1}$$

Accélération de G dans R_g

$$\vec{a}(G/R_g) = \frac{d}{dt}\vec{V}(G/R_g)_{R_g} = \dot{\theta}\vec{z}_0 \wedge R_c\dot{\theta}\vec{y}_0 = -R_c\dot{\theta}^2\vec{x}_0$$

Avec $V = R_c\dot{\theta}$ on obtient :

$$\vec{a}(G/R_g) = -\frac{V^2}{R_c}\vec{x}_0 \tag{2}$$

Moment dynamique en G de E dans son mouvement par rapport à R_g

$$\vec{\delta}(G,E/R_g) = \frac{d}{dt}\vec{\sigma}(G,E/R_g)_{R_g} = \frac{d}{dt}\left(I(G,E)\cdot\vec{\Omega}(E/R_g)\right)_{R_g} = \frac{d}{dt}\left(-E\dot{\theta}\vec{x}_0 - D\dot{\theta}\vec{y}_0 + C\dot{\theta}\vec{z}_0\right)_{R_g}$$

Soit

$$\vec{\delta}(G,E/R_g) = -E\dot{\theta}^2\vec{y}_0 + D\dot{\theta}^2\vec{x}_0 \tag{3}$$

Actions mécaniques de contact entre le hangar et les roues

$$\text{TRD} : \vec{F}(R\to 2) + \vec{F}(R\to 3) + \vec{F}(\text{pes}\to E) = m\cdot\vec{a}(G/R_g)$$

Soit :

$$T_A + T_B = -m\frac{V^2}{R_c} \tag{4}$$

et

$$N_A + N_B - m\cdot g = 0 \tag{5}$$

$$\text{TMD} : \vec{M}(B,R\to 2) + \vec{M}(B,R\to 3) + \vec{M}(B,\text{pes}\to E) = \vec{\delta}(B,E/R_g)$$

$$\vec{M}(B,R\to 2) = B\vec{A}\wedge\vec{F}(R\to 2) = l\vec{x}_0\wedge(T_A\vec{x}_0 + N_A\vec{z}_0) = -l\cdot N_A\cdot\vec{y}_0$$

$$\vec{M}(B,pes\to E) = B\vec{G}\wedge -mg\vec{z}_0 = (\frac{l}{2}\vec{x}_0 + r\vec{z}_0 + e\vec{z}_1)\wedge -mg\vec{z}_0 = \frac{l}{2}mg\vec{y}_0$$

$$\vec{\delta}(B,E/R_g) = \vec{\delta}(G,E/R_g) + B\vec{G}\wedge m\vec{a}(G/R_g) = (\frac{l}{2}\vec{x}_0 + r\vec{z}_0 + e\vec{z}_1)\wedge -m\frac{V^2}{R_c}\vec{x}_0$$

$$\vec{\delta}(B,E/R_g) = -E\dot{\theta}^2\cdot\vec{y}_0 + D\dot{\theta}^2\cdot\vec{x}_0 - m(r+e)R_c\vec{y}_0$$

Sur $\vec{y}_0$:

$$-LN_A + \frac{l}{2}mg = -E\dot{\theta}^2 - m(r+e)\frac{V^2}{R_c} \tag{6}$$

Efforts normaux N_A et N_B (en fonction de m, l, r, e, g, R_c et V) En négligeant $E\dot{\theta}^2$:

$$N_A = m(r+e)\frac{V^2}{l\cdot R_C} + \frac{1}{2}mg \tag{7}$$

$$N_B = -m(r+e)\frac{V^2}{l\cdot R_C} + \frac{1}{2}mg \tag{8}$$

Hypothèses : $e < R_c$ et $e < \frac{l}{2}$

Il y a décollement si l'effort normal devient négatif :

- $N_A < 0$ impossible, donc la roue extérieure reste en contact avec le hangar
- $N_B < 0$ possible pour V assez grand

Condition de non-renversement Non décollement si $N_B > 0$ implique $-m(r+e)\frac{V^2}{l\cdot R_c} > -\frac{1}{2}mg$

$$\boxed{\frac{V^2}{R_c} < \left(\frac{l}{2(r+e)}\right)\cdot g} \tag{9}$$

Condition d'adhérence Adhérence si $\left|\frac{T_A}{N_A}\right| < f$ et $\left|\frac{T_B}{N_B}\right| < f$, la tendance au glissement est suivant $\vec{x_0}$ donc T_A et N_A sont de même signe soit :

$$|T_A + T_B| < f(N_A + N_B)$$

$$T_A + T_B = -m\frac{V^2}{R_c}$$

et

$$N_A + N_B - mg = 0$$

Soit

$$m\frac{V^2}{R_c} < f \cdot mg$$

$$\boxed{\frac{V^2}{R_c} < f \cdot g} \tag{10}$$

1.3 Autres résultats et bilan

On a maintenant deux équations qui nous permettent de connaître la vitesse max qu'on pourra avoir sur une trajectoire.

On peut aussi montrer que la durée minimale de parcours d'un virage est $t_1 = \pi\sqrt{\frac{R_c(r+e)}{2lg}}$, et que la durée minimale de parcours d'un couloir est $t_2 = 2\sqrt{\frac{l}{fg}}$.

2 Cheminement

2.1 Décomposition du problème

2.1.1 Démarche

On va dégrossir le problème. On va partir de l'approche la plus grossière (avec une combinatoire explosive) pour affiner le modèle et l'adapter au problème du hangar qu'on s'est donnés.

Partons du modèle de la grille, qui modélise le hangar (en figure 2).

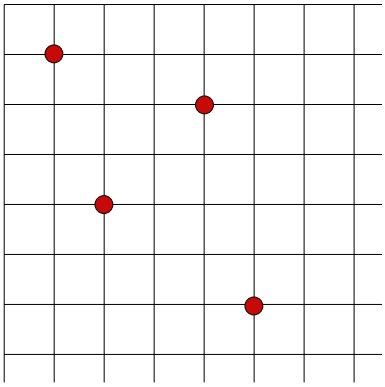

FIGURE 2 – Modèle du hangar avec les points imposés de la trajectoire

Application directe de Dijkstra On tente de modéliser le hangar avec une structure de données acceptable par l'algorithme de Dijkstra. [2] [3] Pour modéliser cette grille, il faut construire un graphe non orienté à arêtes valuées. Pour notre problème, chaque arête doit représenter un unique chemin, et sa valuation le temps parcouru.
Or 5 segments alignés, ce n'est pas la même chose que 5 segments perpendiculaires : le modèle mécanique nous dit qu'on va être obligé de ralentir avant de prendre chaque virage. Ces deux chemins auront donc une valuation différente. Ils seront donc représentés par deux arêtes différentes. Dans certains cas, il y a donc plus d'arêtes que de segments de couloir. On ne peut donc pas appliquer directement Dijkstra à la grille, ce qui provoque une explosion du nombre d'arêtes (on détaillera plus loin comment opérer cette transformation).

L'impossibilité d'appliquer directement Dijkstra peut être vue différemment. Ce qui nous intéresse, c'est de choisir le meilleur chemin passant par les différents points imposés.En supposant qu'il n'y ait pas de grille, on pourrait relier les points par des segments de droite, valués par leur longueur ou leur temps, et directement appliquer Dijkstra. Mais la grille impose un chemin le long de ses couloirs, donc on ne peut pas assimiler le graphe à l'ensemble des points imposés et les relier, puisque la ligne droite n'existe plus, et qu'il existe plusieurs moyens d'aller d'un point imposé à l'autre, et que certains sont meilleurs que d'autres. Si on voulait énumérer l'ensemble des chemins possibles, on obtiendrait un graphe avec beaucoup plus d'arêtes que de segments de couloir. Ce qui fait qu'on doit calculer un très grand nombre de valuations. Il est donc impossible d'appliquer directement Dijkstra à la grille.

2.1.2 Algorithmes locaux

Nous allons partir d'un algorithme naïf, qu'on va tenter d'améliorer. On va ensuite partir d'une heuristique, et montrer qu'elle est quasiment toujours optimale.

Par énumération naïve Pour éviter une énumérations de tous les chemins préalable à leur valuation : on va faire cette énumération de façon plus locale, seulement là où on en a besoin. En effet, si on sait à l'avance qu'un segment de couloirs ne fera jamais partie d'un chemin optimal, il est inutile de calculer l'ensemble des chemins passant par ce segment. Une condition suffisante (mais pas nécessaire) permettant d'affirmer qu'un segment de couloir ne fera jamais partie d'un meilleur chemin entre deux points imposés, est de vérifier s'il appartient à la grille de taille minimum contenant les deux points imposés. Dans la figure 3 on montre un exemple de segment de couloir qui ne peut constituer un chemin optimal.
Une fois qu'on a réduit l'étude des chemins possible à ceux de la sous-grille, il devient

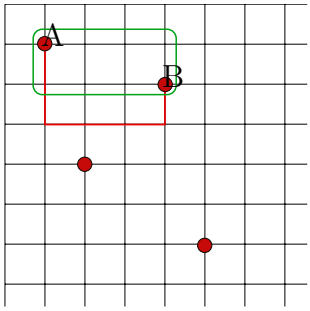

FIGURE 3 – Le chemin rouge ne peut être un chemin optimal entre A et B, car il sort de la grille verte délimitée par A et B

plus raisonnable d'énumérer tous les chemins. On a commencé à dessiner des chemins entre A et B en figure 4.

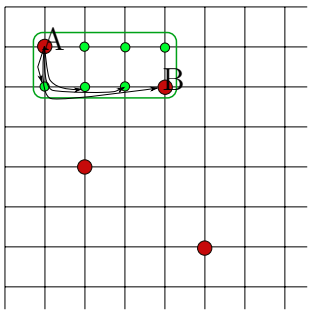

FIGURE 4 – On commence à énumérer les chemins dans la sous-grille délimitée par A et B

Pour encore améliorer ce traitement, on peut aussi enlever les arêtes qui ne vont pas jusqu'à B. On obtient alors un graphe non traitable par un Dijkstra, avec un algorithme mauvais, mais qui a été appliqué localement donc sur des données de taille raisonnable. On peut voir un exemple de résultat en figure 5.

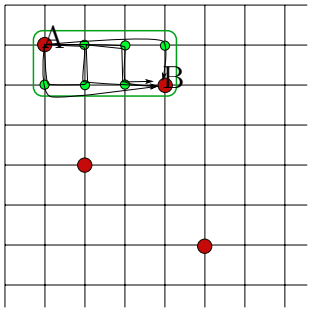

FIGURE 5 – On énumèré tous les chemins possibles ; c'est un mauvais algorithme, mais il est appliqué sur de petites données.

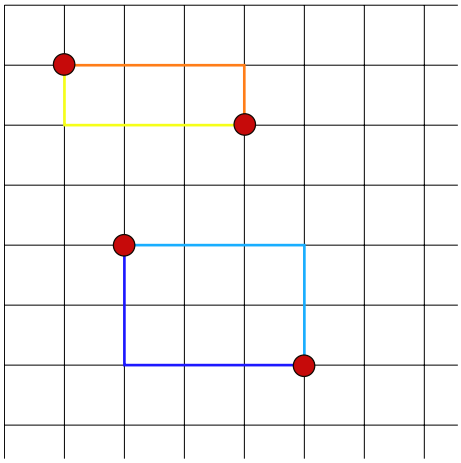

FIGURE 6 – Entre deux points de la grille, il n'existe que deux chemins minimisant le nombre de virages.

Par minimisation des virages L'idée de la décomposition en sous-problèmes locaux est bonne, mais l'algorithme d'énumération utilisé dans chaque sous-problème local est très mauvais.
On peut le remplacer par un algorithme beaucoup plus puissant, mais pas forcément exact, en faisant une hypothèse : en supposant qu'une ligne droite est forcément plus rapide qu'un virage. Nous discuterons de la validité de cette hypothèse plus bas. En la supposant vraie, on n'a plus qu'à minimiser le nombre de virages, il suffit de construire la plus grande ligne droite, donc de parcourir la sous-grille par l'un de ses deux côtés (voir figure 6). On a ainsi une heuristique très simple et très puissante permettant de calculer un bon chemin pour tout couple de points, qui est le meilleur chemin lorsque l'hypothèse est vraie.
L'hypothèse peut être formulée ainsi : pour tout couple de chemins de même longueur (en distance), le chemin le plus court (en temps) est celui qui a le moins de virages.
Cette hypothèse est bien suffisante à l'heuristique qu'on vient de présenter, puisque pour chaque couple de points imposés, tous les chemins de la sous-grille optimale reliant ces deux points sont de même longueur.
Mais est-elle toujours vraie ?
Intuitivement, on pourrait penser que cela dépend de la largeur de la grille et du modèle mécanique. Si la grille est suffisamment fine (càd. la longueur d'un couloir est suffisamment

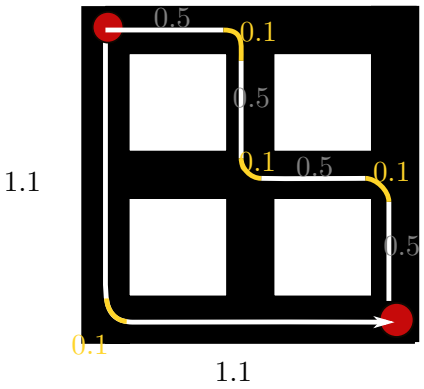

FIGURE 7 – Le cas extrême : la vitesse en ligne droite est constante et ne nécessite pas de ralentissement avant le virage

.

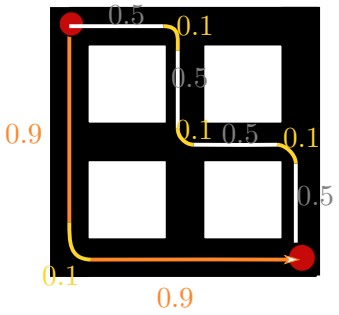

FIGURE 8 – Un cas réaliste

petite), ou si le chariot est suffisamment agile dans un virage (càd. capable d'accélerer suffisamment dans un virage sans se renverser) alors on pourrait être dans un cas où la longueur de deux couloirs alignés n'est pas suffisante pour que le chariot accélère suffisament pour dépasser la vitesse qu'il aurait dans un virage. Dans ce cas, prendre plusieurs virages pourrait être plus efficace.
Mais nous allons montrer que ce modèle est faux, en prenant le cas extrême. Supposons que la vitesse en ligne droite est constante, et suffisamment basse pour que le chariot n'aie pas besoin de ralentir avant de prendre un virage. On peut voir un exemple en figure 7. Alors dans ce cas, la seule différence entre deux chemins de la sous-grille est le nombre de virages. Or, les virages ont eux-même un coût en temps non-nul. Dans ce cas, la différence en temps entre deux chemins est proportionnelle à la différence en nombre de virages.
En pratique, il y a également une accélération sur une ligne droite qui permet d'aller encore plus vite ; c'est illustré en figure 8. Finalement, on peut donc utiliser cette heuristique dans tous les cas.

2.2 Recombinaison des solutions locales et application du modèle mécanique

On a donc montré que notre heuristique locale était valable quel que soit le modèle mécanique (à condition d'utiliser une grille). Quel que soit le couple de points imposés, on peut donc déterminer un chemin optimal entre ces deux points. Il s'agit à présent de

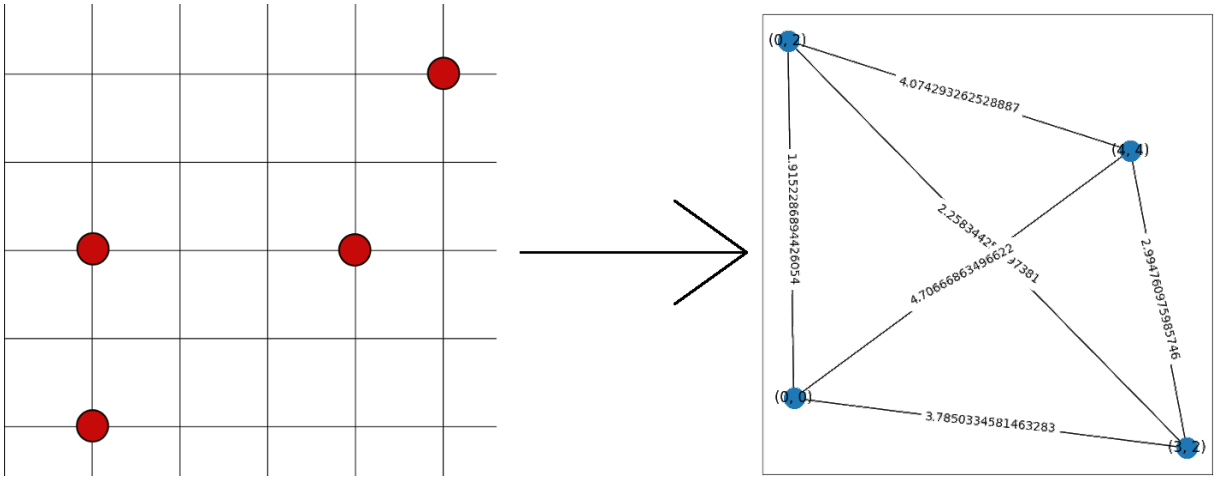

FIGURE 9 – Comme on connaît le chemin optimal entre chaque couple de points, et qu'on sait à l'avance par quels points on veut passer, on peut passer de la grilleà un graphe simplifié, dont les arêtes sont valuées en temps.

tracer tous les chemins localement optimaux, puis d'appliquer Dijkstra au graphe ainsi obtenu.
Le nombre d'arête est donc $\frac{n(n-1)}{2}$ où n est le nombre de points imposés. On obtient ainsi un graphe simplifié, comme illustré en figure 9. Ce nombre est totalement indépendant de la taille de la grille.

2.3 Application d'un Dijkstra simplifié

L'algorithme de Dijkstra L'algorithme de Dijkstra s'applique à un graphe non orienté dont toutes les arêtes ont un poids. Il a pour but de trouver le chemin le plus court entre un sommet de départ et un sommet d'arrivée. En langage naturel, l'algorithme de Dijkstra peut être décrit ainsi :

- A chaque sommet est associée une distance au sommet de départ, qu'on appelera la "distance à 0". Initialement, on met tous les sommets à l'infini (sauf le sommet de départ, qui est à 0 de lui-même).
- A chaque sommet, on associe l'attribut "non visité".
- Il faut choisir un sommet, qu'on appelera le sommet "en cours". On choisit le sommet initial.
- Tant qu'une des conditions d'arrêt ci-dessous n'est pas atteinte :
  - Pour chaque voisin non visité du sommet en cours :
    - Calculer une nouvelle distance à 0 du voisin, en additionnant le poids de l'arête à la distance à 0 du sommet courant.
    - Si la nouvelle distance à 0 du voisin est inférieure à la distance à 0 qui lui est actuellement associée, alors remplacer l'ancienne par la nouvelle. Sinon, garder l'ancienne.
  - Marquer le sommet en cours comme étant visité.
  - Si le sommet de destination a été atteint, sortir de la boucle.

  - Si la plus petite distance à 0 de l'ensemble des sommets non visités est l'infini, cela signifie qu'il n'existe aucun chemin passant par le sommet de départ et le sommet d'arrivée. Sortir de la boucle.
  - Dans l'ensemble des sommets non visités, choisir celui dont la distance à 0 est minimale en tant que sommet en cours.
- Maintenant que les distances à 0 ont été valuées, pour construire le plus court chemin, il suffit de partir du sommet de départ, et de choisir à chaque étape le voisin dont la distance à 0 est minimale.

Simplification de Dijkstra pour notre problème Il existe des différences entre notre problème et celui que Dijkstra résout :

- Il faut passer par tous les sommets ;
- Deux sommets peuvent être alignés ou non alignés dans la grille, et cela modifie le coût. Il y a donc un coût intrinsèque à l'arête, mais aussi un coût lié à un triplet (arête, sommet, arête).
- Le coût entre deux sommets du graphe simplifié est déjà considéré comme "raisonnable" donc on n'est pas dans la situation d'un voyageur de commerce (on peut se contenter d'une solution bonne mais non optimale). [4]
- On a un graphe fortement connexe (ce qui n'est pas forcément le cas d'un problème de Dijkstra). En ignorant le point précédent, cela signifie qu'il suffit de construire le chemin en prenant toujours l'arête la plus légère pour trouver le chemin optimal.

Avec les adaptations mentionnées ci-dessus, on obtient l'algorithme qui a été implémenté dans notre prototype.

2.4 Comparaison avec d'autres algorithmes et performances

Autres algorithmes Pour tester notre méthode, nous avons également implémenté d'autres algorithmes connus :

- Un algorithme génétique de voyageur de commerce ($O(n^2)$ en temps et en mémoire.), pas exact.
- L'algorithme de Ford-Bellman ($O(n \cdot n^2)$), exact.

Faute de temps, nous n'avons pu les tester expérimentalement, mais ils sont déjà prêts à l'usage dans le code que nous avons livré.

Complexité

- En ce qui concerne la complexité en mémoire, comme nous l'avons mentionné, notre graphe simplifié possède $\frac{n(n-1)}{2}$, donc l'ordre est $O(n^2)$ où n est le nombre de points d'arrêt imposé.

- Pour la complexité en temps, l'algorithme est séparé en deux parties :
  - La valuation des chemins optimaux dans les sous-grille, en $O(n^2)$ (une opération par arête du graphe simplifié).
  - Le calcul du chemin optimal, en $O(n)$ dans le meilleur des cas et en $O(n^2)$ dans le pire des cas.

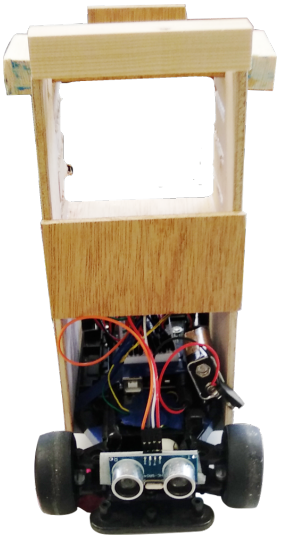

FIGURE 10 – Notre prototype mécanique

3 Utilisation pratique de nos prototypes

Nous avons deux prototypes :

— un prototype mécanique, utilisé pour tester l'adéquation entre notre modèle mécanique et le comportement d'un chariot élévateur ;

— et un prototype numérique, qui simule le cheminement du chariot dans le hangar et assure le calcul d'un plus court itinéraire.

Pour passer à une application réelle, il faudrait :

— effectuer sur un vrai chariot élévateur les mêmes expériences que celles que nous avons réalisées, et, s'il n'y a pas adéquation avec le modèle mécanique, l'affiner ;

— Tester le prototype numérique avec les données d'un vrai chariot et d'un vrai hangar, affiner l'algorithme s'il ne fonctionne pas, puis programmer un contrôle automatique du chariot en l'interfaçant avec le code du prototype numérique.

Mais on peut déjà passer en revue ce que nos prototypes peuvent faire.

3.1 Prototype mécanique

En figure 10 on peut voir une photo du prototype mécanique. Nous avons essayé de reproduire la géométrie et la répartition de masse d'un chariot élévateur ; on peut d'ailleurs modifier la hauteur de l'étagère censée stocker le colis, qui est considéré comme le centre de masse. Nous avons programmé le contrôle de notre prototype. L'interface avec d'autres programmes (comme celui chargé de calculer l'itinéraire) n'est pas encore prêt, en revanche il est suffisamment programmé pour qu'on puisse réaliser des mesures de :

— coefficient de frottement du frein

— vitesse max en virage sans renversement

— vitesse max en ligne droite sans renversement avec de petites déformations de la surface du sol

— vitesse max en ligne droite pour freinage avec un objectif de point d'arrêt (freinage d'urgence si obstacle non prévu détecté)

— etc.

Le prototype embarque :

— Un servomoteur sous consigne d'asservissement permettant de régler l'angle des roues avant ;

— Un moteur dont on peut faire varier la tension d'entrée pour asservir sa vitesse (l'asservissement est géré par notre programme).

— Un capteur à ultrasons (micro et haut-parleur) permettant de détecter des surfaces réfléchissantes et de mesurer leur distance.

Cela permet de nous assister dans nos mesures et de programmer du recueil de données. Nous avons aussi utilisé des caméras pour faire des mesures de vitesse. Une piste d'amélioration serait l'ajout d'un tachymètre permettant de mesurer en direct la vitesse, ce qui permettrait l'enregistrement de données plus précises, et un asservissement de meilleur qualité.
La programmation d'une expérience se fait directement dans le programme. Il suffit de définir des paramètres qui seront constants pendant l'expérience : vitesse max souhaitée et rayon de courbure par exemple. Il est également nécessaire d'introduire des éléments intrinsèques au prototype, par exemple sa masse qui sera utilisée pour des calculs d'accélération (en effet la tension est proportionnelle à l'accélération mais on veut un asservissement en vitesse).
Une fois qu'on a paramétré l'expérimentation, il faut compiler le code et l'enregistrer sur la mémoire de l'arduino. Cela rend le processus assez laborieux si l'on souhaite récupérer des données ou modifier des paramètres en temps réel, mais facilite la répétition d'une même expérience.
Nous avons ainsi réalisé les expériences qui sont détaillées en section 1 , puis nous avons utilisé les paramètres du prototype (géométrie, répartition de masse) ainsi que les résultats des expériences pour paramétrer le prototype numérique.

3.2 Prototype numérique

Le prototype numérique est pour le moment totalement virtuel. Il prend en entrée les paramètres suivants :

— Dimensions de la grille

— Largeur des couloirs de la grille

— Coordonnées des points par lesquels le chariot doit passer

— Paramètres du modèle physique :

  — g

  — μ coefficient de frottement statique carton/fer

  — r rayon des roues

  — e hauteur du centre de gravité

  — l écartement des roues

Tous les calculs ont été séparés dans plusieurs fonctions python clairement identifiées, donc facilement réutilisables dans un autre projet (par exemple la programmation d'un robot). Mais pour en faire la démonstration, on a codé un fichier main qui permet d'entrer certains paramètres, puis qui lance les calculs. Voici un exemple d'exécution du main :

```
dimensions de la grille:
lignes: 5
colonnes : 4
Soit 20 intersections entre couloirs
Largeur de couloir en m^^c3^^a8tres:0.2
Entrez les coordonn^^c3^^a9es des points par lesquels le chariot doit passer.
(0,0) repr^^c3^^a9sente l'intersection en bas ^^c3^^a0 gauche, (0,3) repr^^c3^^a9sente
en bas ^^c3^^a0 droite.
Le premier point saisi sera le point de d^^c3^^a9part.
Abscisse: 0
Ordonn^^c3^^a9e: 0
Entrer un nouveau point d'arr^^c3^^aat impos^^c3^^a9? (0/1)1
Abscisse: 3
Ordonn^^c3^^a9e: 4
Entrer un nouveau point d'arr^^c3^^aat impos^^c3^^a9? (0/1)1
Abscisse: 8
Ordonn^^c3^^a9e: 6
Ce point d^^c3^^a9passe la capacit^^c3^^a9 de la grille.
Entrer un nouveau point d'arr^^c3^^aat impos^^c3^^a9? (0/1)1
Abscisse: 3
Ordonn^^c3^^a9e: 2
Entrer un nouveau point d'arr^^c3^^aat impos^^c3^^a9? (0/1)0
[[1. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 1.]
 [0. 0. 0. 0.]
 [0. 0. 0. 1.]]
```

Dans un premier temps, le programme calcule le graphe simplifié, en utilisant le modèle mécanique. Pour chaque couple de points imposés, il minimise les virage et calcule le temps de parcours. Les sommets du graphe simplifié représentent les points d'arrêt, les arêtes sont valuées par les temps de parcours du meilleur chemin. On peut voir une capture d'écran en figure 11. Le graphe calcule alors l'ordre optimal de parcours des points imposés. Il prend en compte les temps d'arrêt, et le fait que des points alignés soient préférables à des virages.
Sortie finale :

```
[(3, 2), (3, 4), (0, 0)]
```

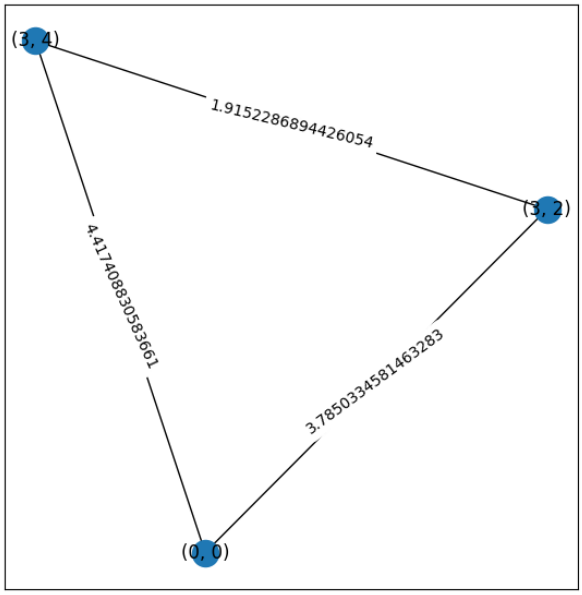

FIGURE 11 – Le graphe simplifié

Les points sont identifiés par leur abscisse et leur ordonnée, et sont ordonnés.

Algorithmes non implémentés dans le main

Il y a deux fichiers non utilisés par le main :

— Un algorithme génétique utilisable sur le problème du voyageur de commerce (graphe simplifié), il ne prend pas en compte les points alignés

— L'algorithme original de Dijkstra utilisable sur le graphe fortement connexe de la grille

Ces implémentations sont celles qui ont été utilisées pour la comparaison de la complexité des algorithmes en 2.4.

3.2.1 Exécution du code

Le code est disponible sur `https://github.com/tx-chariot-utc-p19`.

Son exécution nécessite python3 ainsi que quelques bibliothèques installables automatiquement en exécutant INSTALL.SH.

4 Conclusion

4.1 Travail de recherche

Cette TX a permis de trouver des modèles qui n'étaient pas forcément déjà connus. Le calcul d'un chemin pour l'optimisation du contrôle d'un robot est un problème ancien sur lequel il y a déjà beaucoup de travaux ; nous avons donc choisi un sujet extrêmement spécifique et simple pour nous permettre de partir de (quasiment) zéro et mener nos recherches jusqu'à un début d'application.
La production de nouvelles connaissances n'est certainement pas grande, puisque nous avons choisi un problème un peu connu, et que nous ne sommes pas chercheurs. Cependant, nous apportons des éléments de réponse à un problème très précis ; si quelqu'un d'autre cherche à le résoudre, il pourra très rapidement utiliser ce rapport, le modèle mécanique, ou le modèle numérique. Voici un exemple simple : le problème de recherche du meilleur chemin dans une grille pour un véhicule pouvant tomber facilement dans un virage est très certainement un problème qui a été résolu par d'autres personnes avant nous ; cependant il est très difficile de trouver directement la réponse à ce problème très précis par des travaux de documentation. Nos productions ont donc pour objectif de répondre directement aux besoins de personnes souhaitant automatiser des chariots élévateurs dans un hangar en grille.
Cependant, nos travaux sont très loin d'une application industrielle réelle pour de nombreuses raisons.

4.2 Ouverture sur des applications

Nous avons produits des modèles physiques et numériques qui mettent un peu en application les résultats que nous avons trouvé au cours de nos recherches. Il s'agit d'une trace concrète de nos travaux expérimentaux au même titre que ce rapport, mais ces modèles ne sont pas utilisables dans de vraies applications industrielles pour plusieurs raisons.

Hypothèses sur les modèles Une hypothèse très forte que nous avons fait est que notre modèle mécanique se comporte comme un vrai chariot (voir 1). La validité de cette hypothèse peut être prouvée par des mesures sur un vrai chariot qui pourront ensuite être utilisées dans des calculs ou comparées aux mêmes mesures sur le prototype. Une fois cette hypothèse validée, on peut envisager de réutiliser les résultats du modèle mécanique. Sinon, il faut le refaire.
L'invalidité du modèle mécanique n'affecte pas le modèle numérique. Bien que le modèle numérique en soit dépendant (puisqu'il utilise des fonctions de coût en temps), les calculs et la théorie du modèle numérique ne font aucune hypothèse, excepté le fait qu'un virage a un coût non nul (donc l'enchaînement de segments droits est priviligié). En supposant qu'on adopte un modèle mécanique où le virage a un coût nul, le modèle numérique n'est plus adapté, mais il suffit de changer de modèle de calcul en utilisant les autres algorithmes que nous avons mentionnés (Dijkstra, génétique, Ford-Bellman...).
Nos différentes productions ont donc des sensibilités différentes aux hypothèses mécaniques que nous avons faites.

Limitations du champ des applications Nous avons donc une hypothèse qui permet (si elle est vraie) de généraliser le modèle mécanique à tous les véhicules verticaux, fins et à centre de masse élevé. Le modèle numérique, lui, fonctionne uniquement dans un hangar en grille, où les virages ont un coût non nul. Ces hypothèses rendent nos productions spécifiques à des applications très particulières.

Difficultés d'implémentation Comme nous l'avons mentionné plus haut et dans l'introduction, notre travail n'est pas suffisamment abouti pour une application industrielle. Il manque un travail de relecture sur des éléments théoriques que nous n'avons pas mentionnés dans ce rapport ; un travail d'adaptation des modèles mécaniques aux véhicules qui seront réellement pilotés ; un travail d'adaptation du code aux plateformes sur lesquelles il sera exécuté (optimisation, réécriture dans des langages bas-niveau) ; la conception d'un programme de contrôle (l'asservissement de la vitesse de notre prototype n'étant pas forcément adapté à des applications demandant des objectifs de position précis) ; la conception de systèmes odométriques permettant d'exécuter correctement les instructions venant du modèle numérique.
De plus, il existe des éléments à rajouter que nous ne pouvons pas prévoir parce qu'ils sont intrinsèques aux applications (par exemple, si des humains travaillent à proximité de robots, implémenter des règles de sécurité sur la vitesse, la détection d'obstacle...).

Références

[1] Stéphane Arnaud. *Etude du véhicule à 3 roues Clever*. Sous la dir. de Lycée Corneille. 2016.

[2] E. W. Dijkstra. « A note on two problems in connexion with graphs ». In : *Numerische Mathematik* 1 (1959), p. 269-271.

[3] Wikipédia. *Algorithme de Dijkstra — Wikipédia, l'encyclopédie libre*. 2019. url : `http://fr.wikipedia.org/w/index.php?title=Algorithme_de_Dijkstra&oldid=159657568` (visité le 21/06/2019).

[4] Wikipédia. *Problème du voyageur de commerce — Wikipédia, l'encyclopédie libre*. url : `http://fr.wikipedia.org/w/index.php?title=Probl%C3%A8me_du_voyageur_de_commerce&oldid=157765984` (visité le 21/06/2019).