Rapport - Travail d'investigation Technologique et Scientifique

Axel Mousset Aurélien Labate
Université de Technologie de Troyes

Printemps 2015

Remerciements

Merci à Alexandre Vial pour avoir bien voulu nous encadrer et croire en notre projet.
Merci à Olivier Didon pour avoir réalisé les cartes électroniques du robot.
Merci à Benoit Panicaud pour ses conseils en asservissement.
Merci au club Robotik, pour son cadre de travail et son matériel.
Merci au Bureau des étudiants, à l'institut des sports mécaniques et à l'administration pour leurs subventions.
Et enfin merci à François [nom de famille] pour son usinage irréprochable de la mécanique.

Sommaire

Chapitre 1

Introduction

L'objectif d'une TITS est de réaliser un projet avec un aspect scientifique de recherche, et une mise en application technique en découlant. La robotique, en tant que discipline hybride réunit parfaitement ces deux aspects tout en découpant sa dimension technique en un large spectre de compétences différentes : électroniques, informatiques et mécaniques, sans parler de management et de gestion de projet.

Nous avons donc choisis de mener à bien le développement d'un robot. Les objectifs à remplir pour ce robot étaient sa parfaite autonomie, sa capaciter à pouvoir se repérer et se déplacer, et la gestion d'une pince mono-axe.
Ce projet s'inscrit dans une dynamique associative : le robot réalisé a participé à la coupe de France de robotique, qui est un concours scientifique à l'échelle nationale réunissant plus de 200 équipes, sur lequel nous avons finit 54èmes. Ce projet a donc impliqué plus de quatres autres personnes régulièrement, étudiants, professeurs, et techniciens de l'UTT. Il est donc important de noter que ce document décrit l'essentiel de notre travail, à savoir électronique et informatique, mais aussi le projet dans sa globalité car il nous parait pertinent de replacer les réalisations dans leurs contexte.

Chapitre 2

Odometrie

L'odometrie désigne l'évaluation de la position du robot dans un repère fixe à l'aide de mesures sur son déplacement. On se place dans le cas d'une odométrie calculée à l'aide de **deux codeurs incrémentaux à quadrature montés sur roue codeuses**. On distingue roues motrices et roues codeuses : en effet, disposer les codeurs directement sur l'arbre moteur risque de provoquer des enregistrement de déplacements liés à un patinage des moteurs en cas de roue motrice bloquée. Les methodes d'odometrie liés à l'utilisation de capteurs optiques ne seront pas traités ici.

2.1 Contexte

On échantillonne les impulsions des encodeurs périodiquement tous les $d\tau$. On choisit l'origine au millieu de l'essieu du robot, et on assimile le robot à ce point. La position du robot est représentée par le vecteur

$$p = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} \tag{2.1}$$

Avec :
— (x, y) La position du millieu de l'essieu
— θ l'orientation du robot

2.2 Interprétation des signaux

2.2.1 Valeur et sens de rotation

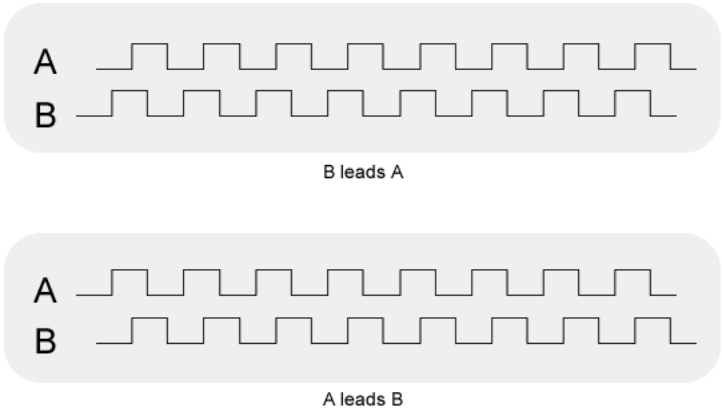

FIGURE 2.1 – Signaux envoyés par un encodeur

Chaque encodeur donne deux signaux : le signal A et le signal B. Chaque signal est déphasé de $\pi/4$, et le sens d'un signal peut être déduit de la valeur du deuxième signal au même moment.
Exemple : l'écoute d'un front montant sur A peut être interpété comme une impulsion positive si l'état de B est à bas au même moment.

On note :

$$\begin{cases} impulsions_{gauche} \\ impulsions_{droite} \end{cases}$$

la *somme algébrique des impulsions depuis le dernier échantillonnage* sur les roues gauche et droite.

2.2.2 Multiplication logicielle de la résolution

Il est possible de multiplier la résolution de son encodeur pour peu qu'il soit de bonne qualité, i.e que le rapport cyclique des signaux envoyés soit de 0.5. Notons que cette multiplication est purement logicielle, et que par conséquent elle induit une imprécision (bien qu'elle amène aussi une précision suplémentaire du au nombre d'impulsions plus important !).
On peut ainsi doubler la résolution en comptant aussi bien les fronts montant que déscendant sur A, voir même quadrupler si on applique le même principe sur B.
Dans le reste de ce document, on notera C_r le *coefficient de résolution*, qui représente le nombre de fronts comptés par période du signal.

2.3 Formules de passage

On raisonnera la plupart du temps en impulsions d'encodeur et non en mètres. Voyons alors comment passer d'un système à un autre.

On note :

- $impulsions_{parTour}$ le nombre d'impulsion d'encodeur par **tour de roue codeuse**.
- $impulsion_{encodeur}$ le nombre d'impulsion d'un tour d'encodeur **originel**.
- $rapport_{réduction}$ le rapport de réduction entre le nombre de tour de roue codeuse et le nombre de tours d'encodeur.

On a la relation suivante :

$$impulsions_{parTour} = C_r \times rapport_{réduction} \times impulsion_{encodeur} \tag{2.2}$$

Ainsi, pour passer de mètres en impulsions :

$$[m] = \frac{[impulsions] \times 2\pi \times R}{impulsions_{parTout}} \tag{2.3}$$

En pratique, on utilise le fait que la relation ci-dessus ne fait intervenir que des constantes. On peut donc la simplifier :

$$[m] = K_1 * [impulsions]$$

Avec K_1 le facteur permettant de passer d'un système à l'autre. On le détermine alors empiriquement en faisant parcourir une distance L en mètres au robot et on mesure la distance ρ en impulsions parcourue. On trouve alors :

$$K_1 = \frac{L}{\rho} \tag{2.4}$$

2.4 Traitement des données

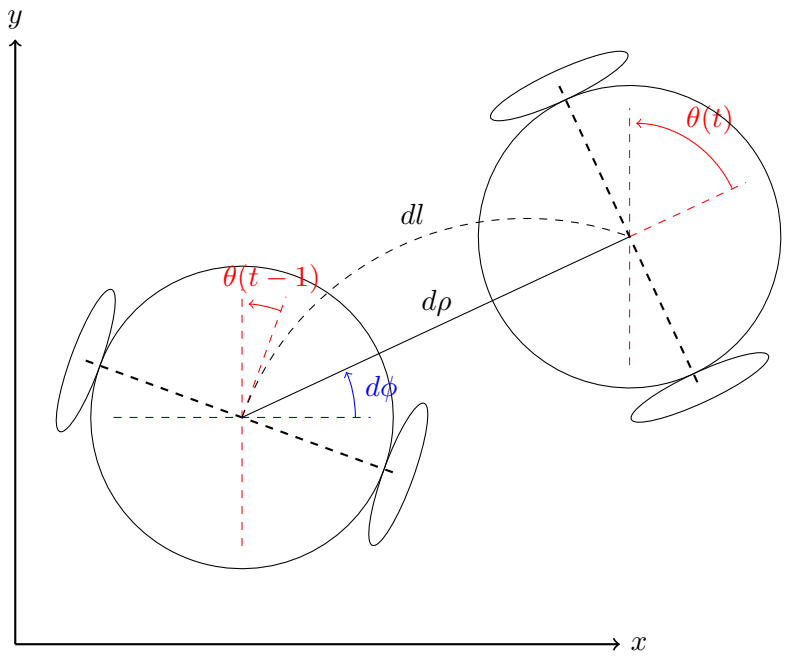

FIGURE 2.2 – Evolution du système pendant $d\tau$

On fait une **approximation linéaire** sur le déplacement du robot, c'est à dire que l'on considère que sur un intervalle de temps $d\tau$ *assez court* (en pratique 5ms semble suffisant), le déplacement dl du robot est une droite, i.e $d\rho = dl$.
De même, on consièdre que l'orientation de cette droite est la moyenne de l'orientation en début de déplacement et de l'orientation en fin de déplacement. On note alors ϕ l'angle de ρ :

$$\phi = \frac{\theta(t) + \theta(t-1)}{2} = \theta(t-1) + \frac{d\theta}{2}$$

Au cours de $d\tau$ à l'instant t :

$$d\rho = \frac{impulsions_{gauche} + impulsions_{droite}}{2} \tag{2.5}$$

$$d\theta = \frac{impulsions_{gauche} - impulsions_{droite}}{entraxe} \tag{2.6}$$

Avec $entraxe$ la distance entre le centre des deux roues codeuses.

En pratique : Comme pour le facteur de passage entre ticks et mètres, la valeur de l'entraxe utilisée sur le système final doit être déterminée **empiriquement**. On fait alors tourner le robot sur $n\pi$ en sommant la différence de ticks. On trouve alors :

$$entraxe = \frac{\sum(impulsions_{gauche} - impulsions_{droite})}{n\pi} \tag{2.7}$$

On procède à une intégration numérique [1] pour obtenir l'orientation du robot dans le repère fixe :

1. i.e une somme discrète, considérée continue car $d\tau$ est petit

$$\theta(n) = \int_0^t d\theta$$

Pour obtenir les coordonnées du robot dans le repère fixe, on projette la droite portée par ρ.

$$dx = d\rho \times cos(\theta(t-1) + \frac{d\theta}{2}) \tag{2.8}$$

$$dy = d\rho \times sin(\theta(t-1) + \frac{d\theta}{2}) \tag{2.9}$$

A nouveau par intégration numérique, on obtient les coordonnées du robot dans le repère fixe :

$$x(t) = \int_0^t dx$$

$$y(t) = \int_0^t dy$$

Finalement, en notant p la position du robot, on peut éxprimer p' sa position après un échantillonnage.

$$p' = f(p, ticks_{left}, ticks_{right}) = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} + \begin{pmatrix} \frac{ticks_{left}+ticks_{right}}{2} \times cos(\theta + \frac{ticks_{left}-ticks_{right}}{2\times entraxe}) \\ \frac{ticks_{left}+ticks_{right}}{2} \times sin(\theta + \frac{ticks_{left}-ticks_{right}}{2\times entraxe}) \\ \frac{ticks_{left}-ticks_{right}}{entraxe} \end{pmatrix}$$

Code d'exemple

```
// Constantes theoriques
const REFRESH_TIME    = 5; //ms
const RESOLUTION_COEF = 2;
const ENCODER_TICKS   = 500;
const REDUCTOR_RATIO  = 1.2;
const WHEEL_RADIUS    = 0.06; //metres
const ENTRAXE         = metersToTicks(0.5); //impulsions


let leftEncoder       = new Encoder();
let rightEncoder      = new Encoder();
let distance          = 0;
let orientation       = 0;
let x                 = 0;
let y                 = 0;
let lastCall          = null;


function ticksToMeters(ticks) {
    return (ticks * 2 * Math.PI * WHEEL_RADIUS) /
        (RESOLUTION_COEF * REDUCTOR_RATIO * ENCODER_TICKS);
}


function metersToTicks(meters) {
    return (meters * RESOLUTION_COEF * REDUCTOR_RATIO * ENCODER_TICKS) /
        (2 * Math.PI * WHEEL_RADIUS);
}


/**
 * Fonction de rafraichissement appelee le plus frequemment possible
 */
function compute() {
    let now = Date.now();

    // On s'assure que les calculs sont fait a intervalles reguliers
    if (now - previousOrientation >= REFRESH_TIME) {
        lastCall = now;

        let leftTicks = leftEncoder.getTicks();
        let rightTicks = leftEncoder.getTicks();

        leftEncoder.resetTicks();
        rightEncoder.resetTicks();


        let rho = (leftTicks + rightTicks) / 2;
        let dTheta = (leftTicks - rightTicks) / ENTRAXE;


        let dx = rho * Math.cos(orientation + dTheta / 2);
        let dy = rho * Math.sin(orientation + dTheta / 2);

        x += dx;
        y += dy;
        orientation += dTheta;
    }
}


function getRobotStatus() {
    return {
        x : ticksToMeters(x),
        y : ticksToMeters(y),
        orientation: orientation
    }
}
```

Chapitre 3

Asservissement

Que ce soit en robotique ou dans différents processus industriels, il est souvent indispensable de contrôler certaines grandeurs physiques. On appelle **asservissement** d'une grandeur l'ensemble des moyens permettant de contrôler **automatiquement** cette grandeur.
Celui-ci peut être mis en place soit au moyen d'un *système en boucle ouverte*, c'est a dire en calculant une commande sans prendre en compte la réponse du système : le système nécessite alors une parfaite modélisation, nous ne nous attarderons donc pas sur ce type d'asservissement.
L'asservissement dont nous allons parler se place dans le cas d'un **système en boucle fermée** ou **système à contre réaction négative**, qui, contrairement à son homologue ouvert, mesure la réaction du système pour adapter la commande. Toute la difficulté est alors de réussir à déterminer la fonction permettant d'adapter la commande de sortie en fonction de l'objectif voulu et du résultat mesurée.
Le correcteur *PID* est un exemple de système à boucle fermée permettant d'asservir efficacement une grandeur. C'est le correcteur le plus utlisié, qui a largement fait ses preuves dans l'industrie.

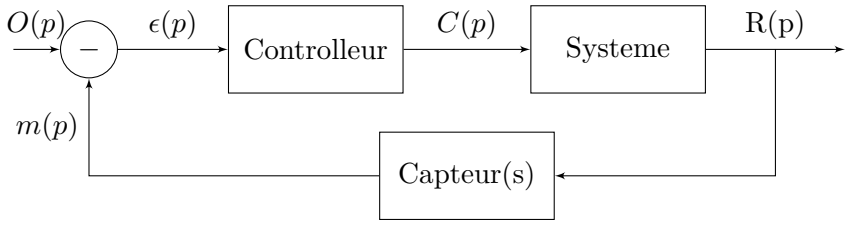

FIGURE 3.1 – Représentation d'un système asservi en boucle fermée

3.1 Correcteur PID

Le correcteur PID est un système à contre réaction **négative**, il calcule donc une commande en fonction de l'erreur de la grandeur, c'est à dire la différence entre la valeur voulue (i.e l'objectif) O, et valeur mesurée m.

$$\epsilon(t) = O(t) - m(t) \tag{3.1}$$

La commande de sortie, passée au travers d'un correcteur PID, qu'on note $C(t)$, consiste en une somme de trois termes : une réponse proportionnelle à l'erreur, une réponse proportionnelle à la somme des erreurs, et une réponse proportionnelle à l'évolution de l'erreur. Une traduction quasi-litéralle de cette définition nous donne la relation suivante :

$$C(t) = f(\epsilon(t)) = K_p.\epsilon(t) + K_i.\int_0^t \epsilon(\tau).d\tau + K_d.\frac{d\epsilon}{dt}(t) \tag{3.2}$$

Ou plus simplement [1] dans le domaine de Laplace :

$$C(p) = f(\epsilon(p)) = K_p.\epsilon(p) + K_i.\frac{\epsilon(p)}{p} + K_d.p.\epsilon(p) \tag{3.3}$$

Ce que l'on peut résumer à l'aide du schéma-bloc suivant :

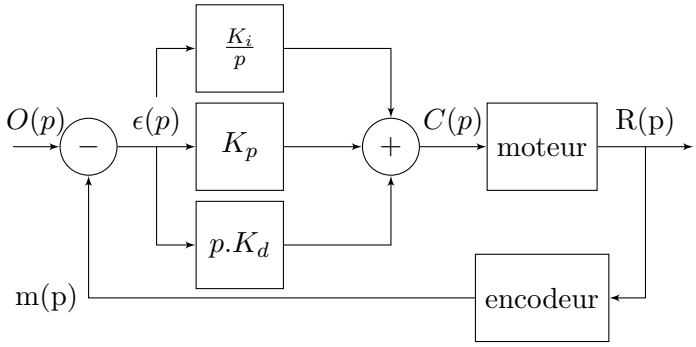

FIGURE 3.2 – Représentation d'un correcteur PID

1. Les relations mettant en jeux des intégrales et des dérivées sont plus simples à exprimer dans le domaine de Laplace

3.1.1 Influence des coefficients

Dans cette section, allons présenter l'influence des différents coefficients sur le système. En guise d'introduction à chacun d'eux, nous allons faire le parallèle avec un système (une voiture) et son correcteur pouvant être assimilé à un PID (un conducteur). On se place dans le cas ou sur une route, le conducteur décide d'être éxactement à la limite de la distance de sécurité de la voiture d'en face.

Réponse proportionnelle

«Plus je suis loin de la limite, plus je dois appuyer sur la pédale d'accélération.»
Le terme proportionnel permet de répondre aux grandes inerties du système, et diminue le temps de montée en donnant l'essentiel de la puissance au moteur : plus l'erreur est grande, plus la réponse est importante. La valeur de K_p est proportionnelle à la vitesse de réaction du système. L'erreur statique est réduite avec l'augmentation de K_p, cependant le système perd en stabilité. En cas de K_p démesuré, le système oscille et ne trouve jamais stabilité.
Dans le cas d'un moteur, une variation trop faible de tension ne fait plus varier sa vitesse. Ainsi lorsque l'erreur devient trop faible, le terme proportionel ne permet pas de combler une erreur statique qui est d'autant plus faible que K_p est grand.

Réponse intégrale

«Moins j'accélère pour une pression de la pédale d'accélération donnée, plus je dois appuyer sur la pédale.»
Le terme intégral fait la somme algébrique des erreurs. Il permet ainsi de donner une réponse de plus en plus forte à mesure que le système persiste dans son erreur. Il complète ainsi le terme proportionel, car permet de combler l'erreur statique laissée par ce dernier en régime permanent.
Augementer K_i permet de diminuer l'impact de faibles perturbations, augemente la précision du système en régime permanent. Un K_i trop important peut cependant augementer le dépassement de la consigne «overshoot», et causer d'oscillations semblables à celles du K_p en cas de valeur démesurée.

Réponse dérivée

«Plus j'accélère vite et que je me rapporoche de la limite (et risque donc de la dépasser), moins je dois appuyer sur la pédale.»
L'action du terme dérivé dépend du signe et de la vitesse de variation de l'erreur. Sa valeur va donc s'opposer à la réponse proportionnelle et intégrale. Elle devient importante lorsque l'erreur faiblit grandement et que le terme proportionnel continue à donner de grandes valeurs : elle freine le système empéchant le dépassement de la consigne et diminuant les petites oscillations qui ralentissent la stabilisation du système. K_d compense donc K_p lorsque l'erreur faiblit, on peut ainsi pousser K_p pour limiter l'erreur statique quitte à dépasser la consigne, puis compenser ce dépassement en augementant K_d.

3.1.2 Réglage des coefficients

Régler les coefficients représente la majeure difficulté de la mise en place d'un asservissement PID. On s'intéresse ici uniquement à l'asservissement d'un moteur (à courant continu, ou pas).

Méthode théorique : Modélisation du système

Dans un premier temps, il est toujours intéressant de s'intéresser à la réponse théorique du système. Cette méthode n'est donc pas une méthode complète en soit, mais juste un outil permettant de "dégrossir" les coefficients avant de les affiner sur le système final.
Les démonstrations des fonctions de transfert ci-dessous ne seront pas développées ici, car elle ne font pas l'objet de ce document. En effet, seul le modèle final et son utilisation nous intéresse.
La fonction de transfert entre la vitesse angulaire de sortie et la tension d'entrée d'un moteur peut être modélisée comme un filtre du second ordre :

$$H(p) = \frac{\omega(p)}{U(p)} = \frac{A}{1 + \frac{2.\xi}{\omega_0} + \frac{1}{\omega_0^2}.p^2} \tag{3.4}$$

Avec :
- $\omega(p)$ Vitesse de rotation du rotor
- $U(p)$ Tension appliquée au moteur
- $A = \frac{1}{K_e}$ Gain statique
- $\omega_0 = \sqrt{\frac{K_e.K_c}{L.J_T}}$ Pulsation propre
- K_e/K_c Constante de vitesse/couple
- J_T Moment d'inertie apporté au rotor

Moteur à courant continu
- $\xi = \frac{R}{2}\sqrt{\frac{J_T}{K_e.K_c.L}}$ Facteur d'amortissement
- R Résistance au bornes du moteur
- L Inductance du moteur

Moteur sans balais
- $\xi = \frac{3R}{2}\sqrt{\frac{J_T}{K_e.K_c.L}}$ Facteur d'amortissement
- $Ke = 0,0605.K_t$
- R Résistance phase-à-phase
- L Inductance phase-à-phase

Il est possible de modéliser le moment d'inertie J_T perçu par le moteur sur son arbre dans le cas d'une utilisation sur un robot de masse M en connaissant le rayon des roues R.

$$J_T = M.R^2 \tag{3.5}$$

Méthode empirique 1 : dite de «Ziegler-Nichols»

La méthode de Ziegler-Nichols ne nécessite aucune modélisation au préalable, seule une batterie d'éssais expérimentaux souvent automatisables est nécessaire.
Le principe est de fixer K_i et K_d à 0, puis d'augementer K_p jusqu'à obtenir une réponse oscillante en régime permanent. On mesure alors la période T_osc de cette réponse à $(K_p)_{lim}$.
Les coefficients sont déduits de ces deux mesures par les relations suivantes :

$$\begin{cases} K_p = 0.6.(K_p)_{lim} \\ K_i = \frac{1}{0.5.T} \\ K_d = 0.125.T \end{cases} \tag{3.6}$$

Les coefficients déduits sont assez polyvalents car ils proposent un bon compromis entre rapidité, dépassement de la consigne, stabilité, et précision. Néammoins, selon la spécification de la réponse désirée, il est possible d'ajuster les différents coefficients à la main. Par exemple, dans le cas de l'asservissement d'un robot, on va vouloir limiter au maximum le dépassement de consigne et donc baisser K_p.

Méthode empirique 2 : dite «À la main»

Cette dernière méthode est utile lorsque l'on a des besoins spécifiques quant à la réponse du système. Par exemple, sur un asservissement en position, on peut vouloir complètement éliminer le dépassement de la consigne (il peut être critique d'aller trop loin et de déplacer un objet à la coupe de France !), quitte à avoir un temps de stabilisation plus long. Dans ce cas là, un asservissement de type PD est recommendable. Le principe est de faire varier K_p jusqu'à qu'il y ai dépassement de consigne, puis de compenser en augmentant K_d jusqu'à élimination de l'erreur statique.

3.2 Asservissement en vitesse

Première manière de contrôler le déplacement du robot : on asservit indépendemment chaque moteur en vitesse.
Incovénient : pour aller à une position donnée, il faut donc intégrer numériquement les consignes de vitesse.

Code d'exemple

```
function velocityControl() {
    let velocity = encoder.getTicks(); // Vitesse en implusion par DeltaT
    encoder.resetTicks();

    previousVelocityError = velocityError;
    velocityError = velocityObjective - velocity;

    let derivative = velocityError - previousVelocityError; // On inclut le 1/DeltaT dans K_d
    integral += velocityError;

    velocityCommand = kp * velocityError + ki * integral + kd * derivative;

    return velocityCommand;
}
```

3.3 Asservissement en position

Asservir indépendemment chaque moteur en position permet de résoudre le problème d'intégration numérique. Par contre, la vitesse n'étant plus asservie, l'accélération peut être beaucoup trop grande, on s'expose alors à des risques de dérapages.

Code d'exemple

```
function positionControl() {
    position += encoder.getTicks(); // Position en ticks
    encoder.resetTicks();

    previousPositionError = positionError;
    positionError = velocityObjective - velocity;

    let derivative = positionError - previousPositionError; // On inclut le 1/DeltaT dans pKd
    integral += positionError;

    positionCommand = pKp * positionError + pKi * integral + pKd * derivative;

    return positionCommand;
}
```

3.4 Asservissement en vitesse et en position

Réunir à la fois un asservissement en vitesse et en position permet de réunir le meilleur des deux mondes. On peut controller la vitesse, et donc l'accélération tout en se passant d'intégration numérique grace à l'asservissement en position.
Un premier asservissement en position va donner une consigne de vitesse, qui après traitement algorithmique pour écrêter vitesse et accélération, va servir de consigne pour un deuxième asservissement en vitesse. On est donc capable de faire atteindre aux moteurs n'importe quelle position tout en controlant la vitesse ainsi que l'accélération à laquelle ils y vont.

Choix du profil de vitesse

Le profil de vitesse adopté est trapézoïdal. On commence par une phase d'accélération jusqu'à une vitesse de croisière, puis une déccélération jusqu'à atteindre la position voulue. L'avantage est de ne pas brusquer les moteurs, de ne pas déraper au démarrage et de ne pas dépasser l'objectif de position par inertie.

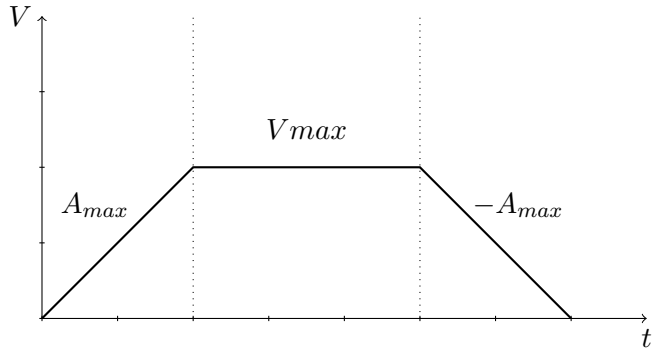

FIGURE 3.3 – Profil de vitesse trapézoïdal

Au niveau de l'algorithme, on tronque juste la valeur de la vitesse si elle dépasse une valeur V_{max}. On fait de même avec l'accélération en tronquant sa valeur absolue à A_{max}.

Code d'exemple

```
function motionControl() {
    previousVelocityObjective = positionCommand;
    velocityObjective = positionControl();

    // Ecretage de la vitesse
    if (Math.abs(velocityObjective) > maxVelocity) {
        velocityObjective = sign(velocityObjective) * maxVelocity;
    }

    let acceleration = velocityObjective - previousVelocityObjective;
    // Ecretage de l'acceleration
    if (Math.abs(acceleration) > maxAcceleration) {
        velocityObjective = previousVelocityObjective + sign(acceleration) * maxAcceleration;
    }

    motor.run(velocityControl());
}
```

3.5 Asservissement polaire

Jusqu'à présent, tous les types d'asservissement proposés se faisaient sur une ou deux grandeurs, indépendemment sur chaque moteur. Il est possible de travailler sur des grandeurs «couplée», réunissant les informations fournies par les deux moteurs.

Pour un robot, il peut être intéressant d'asservir sur la position et l'orientation. L'intérêt est de pouvoir facilement envoyer le robot à une une position (x, y, θ), sans avoir à faire du traitement algorithmique pour déterminer les consignes de position à envoyer à chaque moteur comme dans un asservissement position ou position/vitesse.
On a donc en tout quatre controlleurs PID : deux pour un asservissement position/vitesse sur la vitesse de déplacement du robot, et deux pour un asservissement position/vitesse sur la vitesse angulaire du robot.
Les grandeurs mesurées ne sont plus directement les impulsions d'encodeur, mais les variations de vitesse et de vitesse angulaire calculées par l'odométrie et décrites par les relations (2.5) et (2.6) pour les PID de vitesse, et l'orientation/distance globale pour les PID de distance.

Code d'exemple

```
function robotControl() {
    /*
        Quadruple asservissement position/vitesse sur la distance et l'angle
        calcules par l'odometrie
    */
    let distanceVelocityCommand = distanceMotionControl();
    let orientationVelocityCommand = orientationMotionControl();

    /*
        Signe arbitraire. On part du principe que l'angle est calcule avec la
        difference entre la position gauche et droite
     */
    leftMotor.run(distanceVelocityCommand - orientationVelocityCommand);
    rightMotor.run(distanceVelocityCommand + orientationVelocityCommand);
}
```

Chapitre 4

Conclusion

La réalisation du robot a été menée à son terme dans les délais que nous nous étions imposé pour concourrir à la coupe de France. Néanmoins, une casse de nature mécanique a empêché le robot d'être oppérationnel dès le premier jour. Ainsi, le robot secondaire (développé par d'autres étudiants) a quand même pu marquer des points.

Malgré cet accident, ce projet nous a donné satisfaction. C'est en effet un réel projet d'ingénierie, alliant autant compétences scientifiques, techniques et humaines pendant plus de six mois d'efforts motivés par l'envie de performer dans une compétition.

Au niveau personnel, l'ensemble des connaissances acquises est très enrichissant. Il a fallu effectuer un large travail documentaire en amont sur des sujets comme l'automatisation, l'asservissement ou encore l'odométrie. C'est donc avec passion que nous nous sommes attachés à mettre en pratique ces nouvelles connaissances aux côtés de celles apprises en cours pour réaliser ce robot.

A un niveau plus large, nous éspérons que ce travail pourra répondre à l'une des problématique majeure dans le monde associatif à l'UTT, à savoir la transmission des savoirs. Le projet a donc largement été penser pour devenir potentiellement de base de référence pour futures années du club de robotique de l'UTT.

Annexe A

Tableau récapitulatif de l'influence des coefficients PID

Coefficient	Temps de montée	Temps de stabilisation	Dépassement	Erreur statique
K_p	Diminue	Augmente	Augmente	Diminue
K_i	Diminue	Augmente	Augmente	Annule
K_d	-	Diminue	Diminue	-

Bibliographie

[1] Christopher Clark : *cos 495 - Autonomous Robot Navigation*. Princeton University, 2011. https ://www.cs.princeton.edu/courses/archive/fall11/cos495/lectures.html.

[2] Robot concept Ville d'Avray : *ODOMETRIE par l'équipe RCVA*. IUT Ville d'Avray, 2010. http ://www.rcva.fr/images/stories/site/cours/Odometrie2010/ODOMETRIE_2010.pdf.

[3] Pinkasfeld Joseph, Morvan Sébastien et Paul Morelle : *Rapport de TX - Projet de Robotique*. Université de Technologie de Compiègne, 2007. http ://assos.utc.fr/utcoupe/old/2007/fichiers/RapportTX.pdf.

[4] Christophe Le Lann : Le pid utilisé en régulation de position et/ou de vitesse de moteurs électriques. 2006-2007. http ://www.totofweb.net/projets/pid/rapport.pdf.

[5] Oludayo John Oguntoyinbo : *PID control of brushless DC motor and robot trajectory planning and simulation*. Thèse de doctorat, Vaasan ammattikorkeakoulu, University of applied sciences, 2009. http ://publications.theseus.fi/bitstream/handle/10024/7467/Oludayo%20Oguntoyinbo.pdf.

[6] Benoît Panicaud et Jérôme Noailles : *EA01 - Automatique et asservissement*. Université de Technologie de Troyess, 2009.