

# Robot kinematics

## Spider SEALK

Laurent Beaudoin & Loïca Avanthey  
Épita



## 1 Position statique

### 1.1 Les équations de cinématique inverse

Vérifiez les résultats théoriques demandés lors de la dernière séance avec ceux des tableaux suivants.

◦ Résultats intermédiaires :

$$v = \sqrt{x^2 + y^2} - c \quad (1)$$

$$d = \sqrt{v^2 + z^2} \quad (2)$$

$$\alpha_1 = \arctan\left(\frac{z}{v}\right) \quad (3)$$

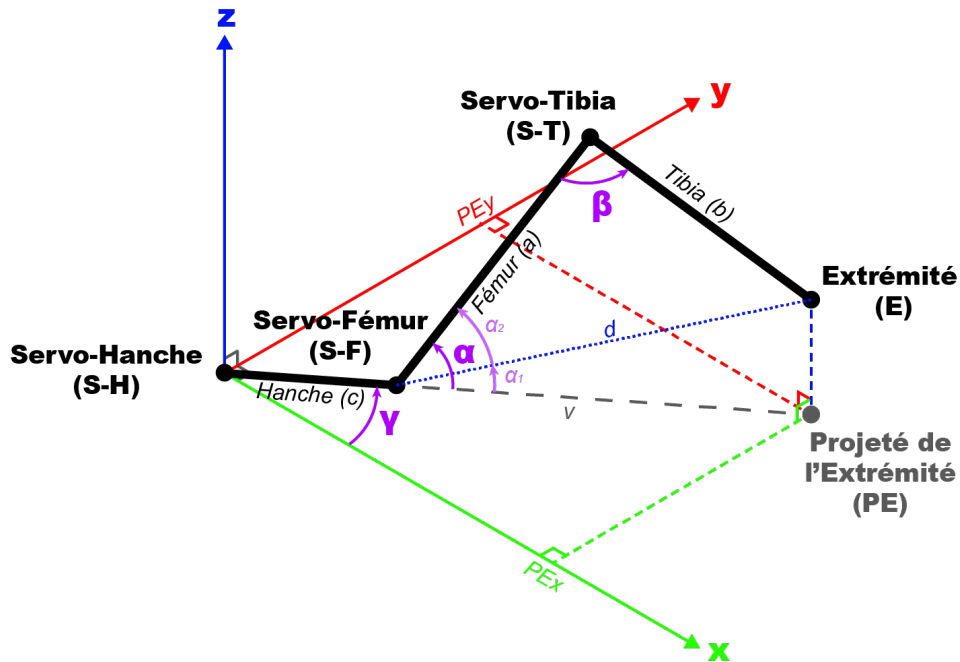
$$\alpha_2 = \arccos\left(\frac{d^2 + a^2 - b^2}{2da}\right) \quad (4)$$

◦ Angles des segments :

$$\gamma = \arctan\left(\frac{y}{x}\right) \quad (5)$$

$$\alpha = \arctan\left(\frac{z}{v}\right) + \arccos\left(\frac{d^2 + a^2 - b^2}{2da}\right) \quad (6)$$

$$\beta = \arccos\left(\frac{a^2 + b^2 - d^2}{2ab}\right) \quad (7)$$



La patte dans son repère 3D, les différents segments, angles et variables utilisés

## 1.2 Les différents repères

Dans ce qui suit, on appellera un **repère patte** le repère qui a pour **origine** une **hanche** et qui est formé par l'**axe gauche/droite** du robot, l'axe  $x$  (qui correspond à la position au neutre du servomoteur de la hanche au moment du montage), par l'**axe avant/arrière** du robot, l'axe  $y$  et l'**axe vertical**  $z$ . Les  $x$  et  $y$  sont dans le sens **positif** quand on s'éloigne du robot et  $z$  est **positif** vers le **haut**. Il y a **4 hanches**, il y a donc **4 repères patte** différents pour le robot et qui sont matérialisés sur la mire de calibration. C'est dans ces différents repères que l'on va **décrire les mouvements des pattes** pour obtenir le mouvement global du robot.



Pour se simplifier la vie, on va d'abord décrire un mouvement dans un **repère direct**, c'est-à-dire celui de la **patte avant droite** (R2), puis on fera les **changements adaptés** pour l'exécuter sur les **autres pattes**.

Nous avons calculé les **angles**  $(\gamma, \alpha, \beta)$  **théoriques**, mais pour **exploiter** ces résultats, il faut maintenant **convertir** ces valeurs en **consignes angulaires** pour les différents **servomoteurs**. Les points durs de cette conversion sont :

- d'identifier à **quelle valeur de consigne** pour chaque servomoteur correspond un **angle théorique nul**,
- d'identifier dans **quel sens** doit tourner un servomoteur pour que la consigne donnée permette d'obtenir physiquement l'**angle théorique**.



Dans ce qui suit, on vous demande **d'abord un résultat théorique** issu de votre **réflexion**. Vous devrez **impérativement nous faire vérifier vos résultats théoriques** avant de passer à l'implémentation (section 1.4) pour ne pas risquer de **casser une patte**.

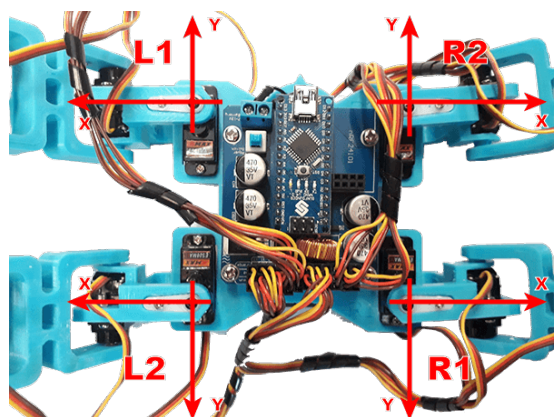
### 1.2.1 Angle $\gamma$ sur les hanches

Commençons par les **consignes** à donner aux **servomoteurs** des **hanches** (S-H) pour les angles  $\gamma$ . Un angle  $\gamma$  **nul** sur chaque revient à **aligner la hanche sur l'axe  $x$** . Si tous les angles  $\gamma$  sont à 0, on obtient la position illustrée à la figure suivante à gauche.

#### QUESTION 1 (*Angle $\gamma$ nul*)



Déterminez pour chaque patte la **consigne angulaire** qui correspond à un angle  $\gamma$  **nul**. Les **résultats** sont à mettre dans le **tableau** de la figure suivante à droite.

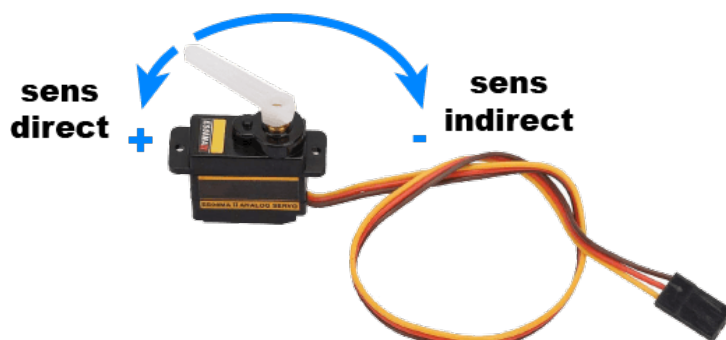


Angle $\gamma$ nul			
L1		90°	R2
L2			R1

Position des hanches pour  $\gamma = 0$  (vue du dessus) à gauche et tableau correspondant des consignes pour les servomoteurs (S-H) à droite.

On souhaite maintenant appliquer un **angle  $\gamma$  non nul**. Par convention, un angle  $\gamma$  **théorique** est **positif** (sens direct) quand la **hanche** tourne vers l'**axe des  $y$  positifs**. Mettre un  $\gamma$  **positif** et **non nul** sur chacun des **servomoteurs** (S-H) revient à mettre les pattes du robot dans la position illustrée à la figure de la question 2 à gauche. Pour trouver les **consignes correspondantes**, il faut déterminer dans **quel sens** doivent se faire les **rotations** des **servomoteurs**.

 Pour information, les servomoteurs fournis sont orientés **positifs** dans le **sens direct** (comme illustré sur la figure suivante).



Les servomoteurs fournis sont orientés positifs dans le sens direct

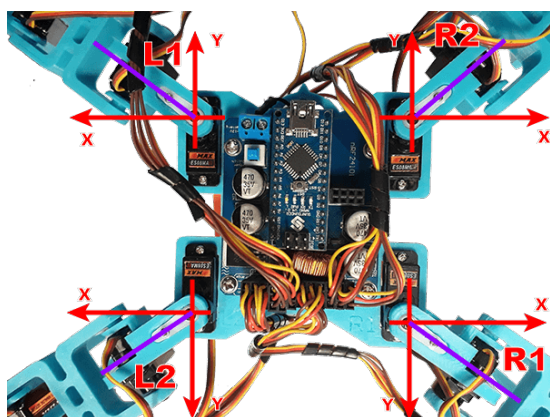
## QUESTION 2 (*Angle $\gamma$ non nul*)



**Déterminez** pour chaque patte la **consigne angulaire** qui correspond à un angle  $\gamma$  théorique **positif** et **non nul** pour obtenir les positions de la figure suivante à gauche.

Pour cela, regardez comment sont **physiquement positionnés** les servomoteurs (S-H) concernés pour déterminer dans **quel sens** ils doivent tourner (direct/positif ou indirect/négatif).

Les **résultats** sont à mettre dans le **tableau** de la figure suivante à droite.



Angle $\gamma$ non nul			
L1		$90^\circ + \gamma$	R2
L2			R1

Position des hanches pour un angle  $\gamma$  non nul (vue du dessus) à gauche et tableau correspondant des consignes pour les servomoteurs (S-H) à droite.

### 1.2.2 Angle $\alpha$ sur les fémurs

Passons maintenant aux **consignes** pour les **servomoteurs** des **fémurs** (S-F) pour les angles  $\alpha$ . Un angle  $\alpha$  théorique **nul** revient à mettre le **fémur** à l'**horizontal**, c'est-à-dire dans le **plan**  $(x, y)$ .

### QUESTION 3 (*Angle $\alpha$ nul*)



Déterminez pour chaque patte la **consigne angulaire** qui correspond à un angle  $\alpha$  **nul** (fémur à l'horizontal). Les **résultats** sont à mettre dans le **tableau** suivant.

Angle $\alpha$ nul			
L1		90°	R2
L2			R1

Tableau des consignes pour les servomoteurs (S-F) pour  $\alpha$  nul.

On souhaite maintenant appliquer un **angle  $\alpha$  positif** et **non nul** sur chacun des **servomoteurs** (S-F). Par convention, un angle **théorique  $\alpha$  positif** (sens direct) signifie un **fémur** qui **monte** (donc vers le sens des  $z$  positifs). Pour cela, il faut **déterminer** dans **quel sens** doivent se faire les **rotations**.

### QUESTION 4 (*Angle $\alpha$ non nul et positif*)



Déterminez pour chaque servomoteur (S-F) la **consigne angulaire** qui correspond à un angle  $\alpha$  théorique **positif** et **non nul** (c'est-à-dire dans le sens des  $z$  positifs).

Pour cela, regardez comment sont **physiquement positionnés** les servomoteurs (S-F) concernés pour déterminer dans **quel sens** ils doivent tourner (direct/positif ou indirect/négatif).

Les **résultats** sont à mettre dans le **tableau** suivant.

Angle $\alpha$ non nul			
L1		90° - $\alpha$	R2
L2			R1

Tableau des consignes pour les servomoteurs (S-F) pour  $\alpha$  non nul et positif.

#### 1.2.3 Angle $\beta$ sur les tibias

Passons maintenant aux **consignes** pour les **servomoteurs** des **tibias** (S-T) pour les angles  $\beta$ . Un angle  $\beta$  théorique **nul** revient à **replier** complètement (théoriquement) le **tibia** sur le **fémur**.



On **ne testera pas** sur le robot un angle  $\beta$  théorique **nul** car il est **physiquement impossible** à réaliser : on risque de **casser** la patte... ou le servomoteur!!!

### QUESTION 5 (*Angle $\beta$ nul*)



Déterminez pour chaque patte la **consigne angulaire** qui correspond à un angle  $\beta$  théorique **nul** (tibia replié sur le fémur). Les **résultats** sont à mettre dans le **tableau** suivant.

Angle $\beta$ nul			
L1		0°	R2
L2			R1

Tableau des consignes pour les servomoteurs (S-F) pour  $\beta$  nul.

On souhaite maintenant appliquer un **angle  $\beta$  positif et non nul** sur chacun des servomoteurs (S-T). Par convention, un angle **théorique  $\beta$  positif** (sens direct) signifie un **tibia** qui s'éloigne du **fémur**. Pour cela, il faut déterminer dans **quel sens** doivent se faire les **rotations**.

#### QUESTION 6 (*Angle $\beta$ non nul et positif*)



**Déterminez** pour chaque servomoteur (S-T) la **consigne angulaire** qui correspond à un angle  $\beta$  théorique positif et non nul (c'est-à-dire qui s'éloigne du fémur).

Pour cela, regardez comment sont **physiquement positionnés** les servomoteurs (S-T) concernés pour déterminer dans **quel sens** ils doivent tourner (direct/positif ou indirect/négatif).

Les **résultats** sont à mettre dans le **tableau** suivant.

Angle $\beta$ non nul			
L1		$0^\circ + \beta$	R2
L2			R1

Tableau des consignes pour les servomoteurs (S-F) pour  $\beta$  non nul et positif.

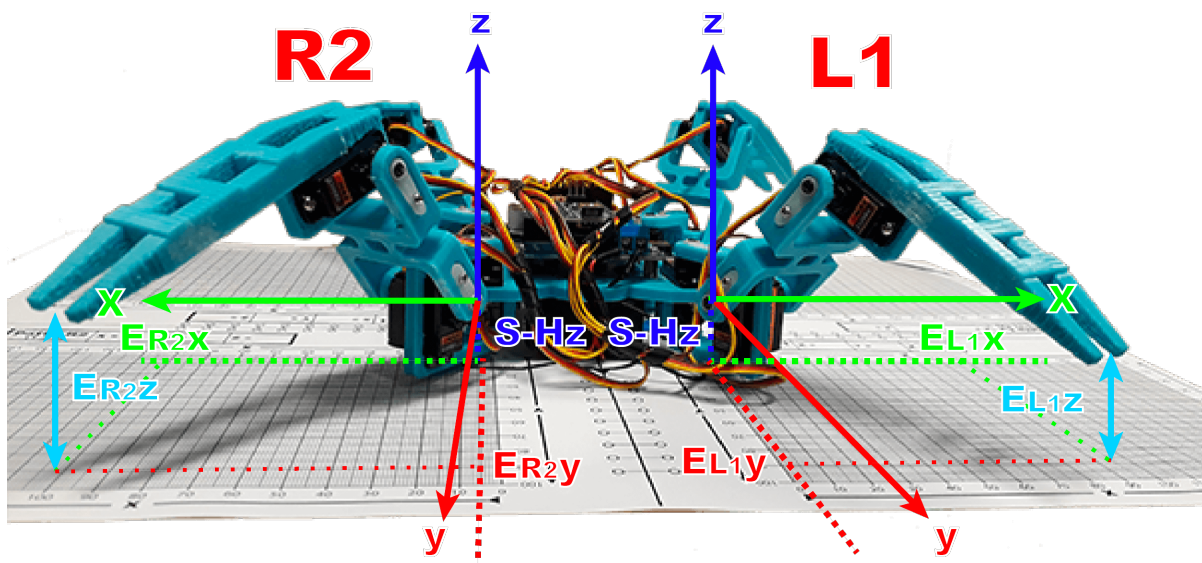
#### QUESTION 7 (*Offset*)



En vous aidant de la figure ci-dessous et sachant que la hauteur S-Hz vaut  $27\text{ mm}$ , indiquez dans le tableau suivant les mesures réelles idéales réalisées sur la mire de calibration lorsque l'on demande à l'extrémité de la patte de se positionner en  $(100, 70, 15)$  ( $E_{R_2}z$  ou  $E_{L_1}z$  par exemple).

Coord.	Position Consigne	Position idéale mesurée
x	100	
y	70	
z	15	

Tableau des consignes en position et des mesures attendues correspondantes.



Repères pattes (vue de face) par rapport au sol

## 1.3 Implémentations & Garde-fou



Pensez bien à faire **valider** vos **résultats théoriques** par un encadrant **avant** de procéder à l'implémentation

### EXERCICE 1 (*Cinématique inverse*)



Implémentez les formules de **cinématique inverse** (section 1.1) pour que vous puissiez calculer les **angles théoriques** ( $\gamma, \alpha, \beta$ ) pour une **position** donnée (créez autant de fonctions que nécessaire). On vous rappelle les **longueurs** des différents **segments** du robot  $a = 53 \text{ mm}$ ,  $b = 79,5 \text{ mm}$  et  $c = 30,5 \text{ mm}$ .



N'oubliez pas que les angles en **radians** doivent être **convertis en degrés** à la fin.



Vous n'êtes **pas encore** autorisés à tester sur le robot à ce stade!

### EXERCICE 2 (*Consignes*)



Implémentez les **fonctions** permettant de **convertir** les angles théoriques en degrés en **consignes** pour les **servomoteurs** des hanches, des fémurs et des tibias de chaque patte (créez autant de fonctions que nécessaire).



Vous n'êtes **pas encore** autorisés à tester sur le robot à ce stade!

Comme vous pouvez le voir sur le robot, tous les servomoteurs **ne peuvent pas** utiliser leur  $180^\circ$  de débattement car ils entrent en **butée** sur des **éléments mécaniques** bien avant. Aussi, il est prudent de **limiter les débattements autorisés** par servomoteur pour limiter les **risques de casse**.

### EXERCICE 3 (*Garde-fou*)



Écrivez une **fonction** qui **vérifie** que la **consigne** que l'on souhaite donner à un servo-moteur est bien dans la **plage autorisée** de ce dernier. Cette fonction sera **systématiquement** appelée **avant** l'application d'une consigne à un servomoteur.

En cas d'**erreur** on appliquera les consignes positionnant l'araignée en **PLS** pour reconnaître facilement visuellement qu'il y a un **problème** (on actionnera d'abord les tibias puis après un délai de  $200 \text{ ms}$  les hanches et les fémurs) et on quittera le programme après un délai de  $200 \text{ ms}$  (le temps que les servomoteurs se soient mis en position).



Maintenant c'est bon, vous pouvez tester. Vous vous assurerez que la fonction garde-fou fonctionne bien grâce à une consigne d' $1^\circ$  en dehors de la plage (pas plus!!!).

R2    L2		R1    L1	
Consigne Hanche	$[60^\circ, 175^\circ]$	Consigne Hanche	$[5^\circ, 120^\circ]$
Consigne Fémur	$[5^\circ, 120^\circ]$	Consigne Fémur	$[60^\circ, 175^\circ]$
Consigne Tibia	$[40^\circ, 160^\circ]$	Consigne Tibia	$[20^\circ, 140^\circ]$

Position PLS	
Consigne Hanche	$90^\circ$ (R2, R1, L1, L2)
Consigne Fémur	$90^\circ$ (R2, R1, L1, L2)
Consigne Tibia	$160^\circ$ (R2, L2)    $20^\circ$ (R1, L1)

## 1.4 Calibration : le retour

Maintenant, on va revenir sur la **différence** qu'il y a entre le **modèle théorique** du robot et la réalité du **prototype**. On va donc faire prendre une pose particulière au robot, la **pose de calibration**, mais que l'on ne définit plus maintenant sous forme de **consignes servomoteurs** (comme à la séance précédente) mais en précisant la **position extrême** de chaque patte.

### EXERCICE 4 (*Reprendre la pose de calibration*)



Utilisez les fonctions que vous avez implémentées à la section 1.3 pour **positionner** l'extrémité des pattes sur le point (100, 70, 15) dans chaque repère patte. On vous rappelle qu'il y a un offset en z (l'origine du repère patte n'est pas en contact avec le sol) de 27 mm.

Pour **corriger les erreurs** et faire prendre à votre robot une position réelle qui s'approche au mieux de la position idéale, il faut (après avoir mis l'araignée en position de calibration) :

- **mesurer** les **positions réelles** extrêmes des pattes
- **calculer** la **correction** à appliquer pour chaque segment de chaque patte.
- **appliquer** la **correction** correspondante à chaque segment de chaque patte.

Pour **calculer la correction** (étape réalisée une seule fois au début du programme), il faut :

- calculer les **angles théoriques** ( $\gamma, \alpha, \beta$ ) que l'on obtient avec la consigne (100, 70, 15)
- puis calculer pour **chaque patte** ces mêmes **angles** ( $\gamma_{leg}, \alpha_{leg}, \beta_{leg}$ ) obtenus avec la **position réelle mesurée** ( $x_{leg}, y_{leg}, z_{leg}$ ) (on n'oublie pas de retrancher l'offset du z mesuré!).
- et enfin pour **chaque patte**, on fait la **différence** entre les valeurs d'angles **théoriques** et les valeurs d'angles **réels** et on **sauvegarde** ces valeurs dans un **tableau**.

Pour l'utilisation, il faudra **ajouter** ces corrections aux valeurs des angles théoriques **à chaque fois** qu'ils sont calculés et ce, **en fonction de la patte concernée**, avant de procéder au calcul des consignes des servo-moteurs.

### EXERCICE 5 (*Calcul des correctifs*)



Implémentez la fonction qui calcule les **différences** d'angles **théoriques** et **réels** et qui les **sauvegarde** dans un tableau. Cette fonction sera appelée dans le **setup**.

### EXERCICE 6 (*Correction des angles on the fly – Exercice noté*)



Implémentez ensuite la **correction d'erreur** grâce au tableau qui contient les correctifs. Pour **valider** votre code, faites prendre à votre robot la position de calibration (100, 70, 15) et **vérifiez** que la position réelle est maintenant **beaucoup plus proche** de la position théorique.

\*\*\* Cet exercice est noté : appelez un encadrant quand vous êtes prêts! \*\*\*



Pour la suite, toutes les consignes que vous passerez aux servomoteurs devront être corrigées !



## 2 Action! Move in the right way!

### 2.1 Mouvement sur une patte : Gymnastique...

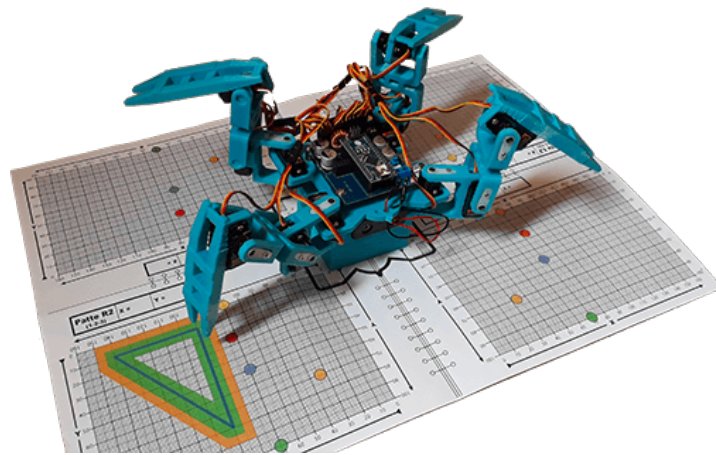
Vous allez commencer par tester votre nouvelle capacité à **positionner** précisément l'**extrémité** d'une patte en **3D** en effectuant un **parcours imposé**.

#### EXERCICE 7 (*Parcours imposé d'une patte sur la mire de calibration – Exercice noté*)



Sur la **mire de calibration** dans le quadrant de la **patte R2**, il y a un **triangle bleu** dessiné. Codez le robot pour que l'**extrémité** de la patte R2 **suive** le centre de ce tracé bleu. On prendra comme **hauteur**  $z = -27$ , la valeur de l'offset, pour que la patte "touche le sol". Servez vous d'un **cube** pour **surélever** l'araignée lors de cet exercice.

\*\*\* Cet exercice est noté : appelez un encadrant quand vous êtes prêts! \*\*\*



Parcours sur une patte.

### 2.2 Mouvements multi-pattes : Get up, stand up!

#### EXERCICE 8 (*Sit down*)



**Implémentez** une fonction qui permet à l'araignée d'être en **position assise**, prête à se lever. Pour cela, le bout des pattes doivent **toucher le sol** ( $z = -27$ , la valeur de l'offset!) et les pattes doivent être **orientées** dans leur **sens de marche** ( $x = 70, y = 50$ ).



La fonction **sit** sera **appelée systématiquement avant** la fonction **stand** ainsi qu'à la fin d'une action de marche ou de la réalisation d'un parcours pour remettre l'araignée en **position repos**.

#### EXERCICE 9 (*Stand up*)



**Implémentez** une fonction qui permet à l'araignée de **se lever**, prête à marcher. Pour cela, le bout des pattes doivent **appuyer sur le sol** pour lever le corps. On donnera donc une valeur de  $z$  **plus basse** que le sol ( $z = -40$ ). Les pattes doivent être **orientées** dans leur **sens de marche** ( $x = 70, y = 50$ ).




Quant à la fonction **stand** elle sera **obligatoirement appelée avant** d'appeler pour la première fois une **fonction de marche**, quelle qu'elle soit, jusqu'au **prochain sit**.



## 2.3 Mouvements multi-pattes : En avant marche !


Pour faire des actions plus **complexes** comme un déplacement en **avant** ou **arrière** ou encore une **rotation** à **gauche** ou à **droite**, il faut **combinaison** des **actions différentes** sur **plusieurs pattes** à la fois. Dans un premier temps, vous allez **étudier** un mouvement à partir d'une **vidéo** pour le **modéliser** pour comprendre quelles actions vous devez faire, sur quelle patte et quand.

 À la **fin** de chaque **séquence** (mouvement en avant, en arrière, rotation à gauche ou à droite), le robot reprend la **pose** qu'il avait au **début** de la séquence. En faisant cela, on s'assure de toujours connaître la **position de départ** du robot pour chaque nouvelle séquence.

### QUESTION 8 (Modélisation de la marche d'un pas en avant – **Exercice noté**)




À l'aide de la **vidéo** fournie, **modélisez** à chaque instant (c'est-à-dire à chaque fois qu'une ou plusieurs pattes bougent) la **position extrême** de chaque patte pour effectuer un pas en avant. Pour cela, complétez le tableau qui suit (fichier pdf formulaire à remplir fourni sur le Moodle).

 Il y a deux hauteurs : patte au sol  $z = -40$  (position du stand up) et patte levée  $z = -20$ .

 Pour plus de **lisibilité**, écrivez des valeurs dans le tableau pour une patte **uniquement** si cette dernière s'est **déplacée** par rapport à sa précédente position (ligne précédente).

\*\*\* Cet exercice est noté : appelez un encadrant quand vous êtes prêts! \*\*\*

	Front						Back					
	Left (L1)			Right (R2)			Left (L2)			Right (R1)		
	x	y	z	x	y	z	x	y	z	x	y	z
start	70	50	-40	70	50	-40	70	50	-40	70	50	-40
move 1				70	100	-20						
move 2												
move 3												
move 4												
move 5												
move 6												
move 7												
move 8												
move 9												
end	70	50	-40	70	50	-40	70	50	-40	70	50	-40

 La **position des points** verts (et jaunes) sur la mire a été **déterminée** par rapport aux **dimensions** du robot : c'est le **pas** le plus **grand** que l'on peut faire pour avancer bien droit sans être déséquilibré.



Pas "un" mais "des" algorithmes de marche

Il y a de nombreux algorithmes de marche à 4 pattes / 3 segments, celui que nous utilisons n'est donc qu'un exemple parmi d'autres !

## EXERCICE 10 (*Ça marche ? – Exercice noté*)



Après validation, **codez** une fonction de **marche**, qui permet d'implémenter le **pas** vu précédemment et de le **multiplier** autant de fois que spécifié en **argument**. Vous appliquerez un délai de 500 *ms* entre chaque mouvement dans le plan horizontal et un délai de 300 *ms* entre chaque mouvement dans le plan vertical.



Vous n'oublierez pas d'appeler **avant** cette fonction la fonction **sit** et la fonction **stand**. Et **après** l'appel de la fonction de marche, vous rappellerez **sit**.

Posez l'araignée sur un cube sur la mire et vérifiez que vous obtenez bien ce que vous souhaitez.

\*\*\* Cet exercice est noté : appelez un encadrant quand vous êtes prêts! \*\*\*

## EXERCICE 11 (*Ça marche vraiment ?*)



Après validation de l'exercice précédent, posez l'araignée au **sol** et allumez-la... alors ?

## 2.4 Mouvements multi-pattes : On tourne !

Pour faire **un pas en avant**, on a vu qu'il fallait avancer une patte puis tout tirer en arrière. Pour faire une **rotation d'un pas** vers la gauche, on va appliquer la **même stratégie**. L'angle le plus grand que l'on peut faire par rapport aux dimensions du robot est de 45°.

Cependant, il n'est **pas prudent** d'effectuer directement la rotation depuis la position **start** car on risque d'arriver sur une position où le servomoteur arrive en **butée**. Aussi pour regagner de l'**amplitude angulaire** et bouger en toute **sécurité**, on va **décaler** notre point de départ vers un **point intermédiaire**, le point rouge ou orange selon les pattes, l'autre point (respectivement orange ou rouge) correspondant à la position à 45°.

## QUESTION 9 (*Modélisation de la rotation d'un pas vers la gauche – Exercice noté*)



À l'aide de la **vidéo** fournie, **modélisez** à chaque instant (c'est-à-dire à chaque fois qu'une ou plusieurs pattes bougent) la **position extrême** de chaque patte lors d'une rotation d'un pas vers la gauche (dans la vidéo vous avez 3 rotations successives d'un pas vers la gauche et 3 rotations d'un pas vers la droite). Pour cela, complétez le tableau qui suit (fichier pdf formulaire à remplir fourni sur le Moodle).



Si vous devez faire une rotation de plusieurs pas successifs, il est inutile de repasser par la position **start**. Vous pouvez enchaîner directement les pas comme sur la vidéo.

Les cases bleues (**move 1** à **move 9**) décrivent la séquence d'un pas vers la gauche et les cases vertes (**move 10** à **move 17**) décrivent la séquence permettant de revenir à la position initiale.

\*\*\* Cet exercice est noté : appelez un encadrant quand vous êtes prêts! \*\*\*

	Front						Back					
	Left (L1)			Right (R2)			Left (L2)			Right (R1)		
	x	y	z	x	y	z	x	y	z	x	y	z
start	70	50	-40	70	50	-40	70	50	-40	70	50	-40
move 1	73	25	-20									
move 2												
move 3												
move 4												
move 5												
move 6												
move 7												
move 8												
move 9												
move 10												
move 11												
move 12												
move 13												
move 14												
move 15												
move 16												
move 17												
end	70	50	-40	70	50	-40	70	50	-40	70	50	-40

## EXERCICE 12 (Ça tourne ? – *Exercice noté*)



Après validation, **codez** une fonction de **rotation**, qui permet d'implémenter le **pas** vu précédemment et de le **multiplier** autant de fois que spécifié en **argument**. Vous appliquerez un délai de 500 *ms* entre chaque mouvement dans le plan horizontal et un délai de 300 *ms* entre chaque mouvement dans le plan vertical.



**Vous n'oublierez pas d'appeler *avant* cette fonction la fonction **sit** et la fonction **stand**. Et *après* l'appel de la fonction de rotation, vous rappellerez **sit**.**

Posez l'araignée sur un cube sur la mire et vérifiez que vous obtenez bien ce que vous souhaitez.

**\*\*\* Cet exercice est noté : appelez un encadrant quand vous êtes prêts! \*\*\***

## EXERCICE 13 (Ça tourne vraiment ?)



Après validation de l'exercice précédent, posez l'araignée au **sol** et allumez-la... alors ?

### 3 Un nouveau code pour de nouvelles perspectives

Les mouvements que vous obtenez donnent une impression de **mouvements saccadés** et un peu **précipités**. Pour arriver à les **fluidifier**, il faut entrer beaucoup plus en détail dans la gestion fine du **temps** des consignes.

❗ En robotique, cette **gestion du temps** est **capitale**, car elle **conditionne** ce qu'un robot peut faire ou non et comment il doit le faire.

Prenons un **exemple** : si un mouvement impose une succession **trop rapide** de consignes **trop éloignées** les unes des autres (la téléportation n'existe pas (encore ?) à notre époque ;-)), le servomoteur n'aura **pas le temps** d'exécuter **physiquement** les ordres dans le temps imparti et le mouvement **échouera**. C'est pour cela que la robotique intègre la notion de **vitesse** pour imposer des **contraintes** sur les vitesses admissibles et d'**accélération** pour tenir compte entre autre des phénomènes d'**inertie**.

i Dans ce **module d'introduction**, nous n'aurons **pas le temps** d'aborder ces notions.

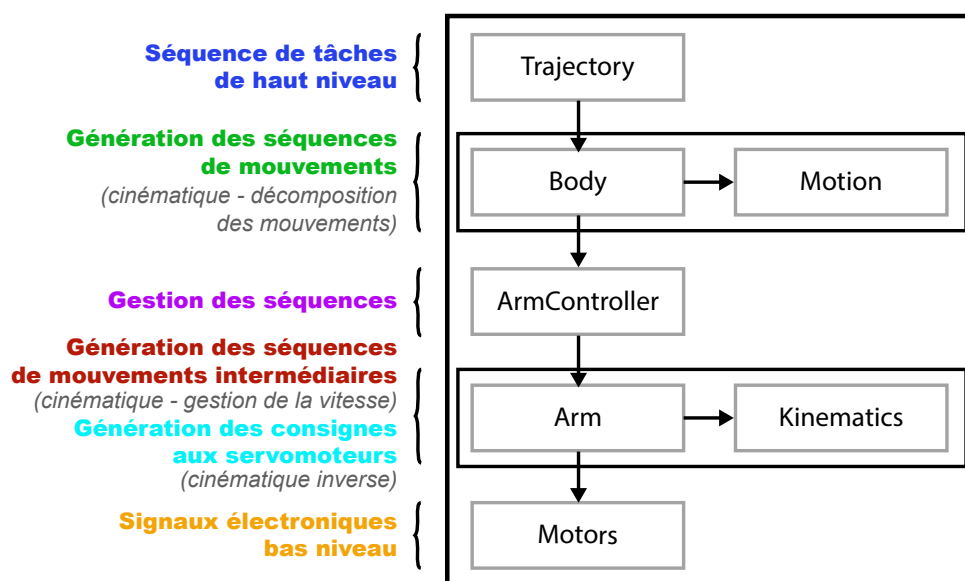
D'autre part, il y a d'autres notions propres aux robots marcheurs que l'on n'a pas abordées comme la gestion de la **position** du **centre de gravité** en fonction de l'empreinte au sol des pattes porteuses. Aussi, pour aller plus loin, on vous fournit un **nouveau code** qui tient compte de toutes ces contraintes et que vous allez **compléter** (code à trous) avec ce que vous avez travaillé jusqu'ici.

✓ Ce code est **architecturé** de manière **modulaire**.

Il permet de gérer **finement** et de manière automatiquement **adaptative** l'enchaînement temporel des différents ordres aux pattes en maîtrisant la notion de **vitesse**, à l'échelle du robot (vitesse de déplacement du robot dans un référentiel galiléen classique, c'est-à-dire là où le principe d'inertie est vérifié) comme à l'échelle des **vitesses angulaires** au niveau des servomoteurs. Ce code aide aussi à ce que le robot marcheur reste **stable** durant ses déplacements.

#### 3.1 Architecture du nouveau code - vademecum

L'**architecture** du code est **organisée** comme illustré à la figure suivante. Les **abstractions** de plus **haut niveau** se trouvent en haut de ce schéma et concernent les notions qui portent sur le robot dans sa **globalité**. À l'inverse, les notions **bas niveau** proches de la **couche électronique** comme les consignes aux servomoteurs se trouvent en bas du schéma.



Synoptique de l'architecture générale du nouveau code.

**Trajectoire** Le plus haut niveau de l'architecture est la trajectoire. Elle correspond à la **liste des déplacements haut niveau** que doit effectuer le robot dans un référentiel galiléen (avance de 10 pas, tourne à gauche de 3 pas, etc.). C'est dans la méthode `setup()` de l'objet `trajectory` que vous listez la trajectoire à suivre. Voici un **exemple** de trajectoire :

```
void Trajectory::setup() {
    stand();
    step_forward(10);
    turn_left(3);
    sit();
}
```

**Corps (Body)** Le corps est un **objet intermédiaire** permettant d'**initialiser** les **pattes** et de **générer** la **trajectoire** définie précédemment, c'est-à-dire décrire l'**enchaînement des mouvements** des différentes pattes. Cet enchaînement est défini dans l'objet `Motion4` (pour 4 pattes) dans `Motion.cpp`.

```
void Motion4::turn_left(bool lastmovement);
void Motion4::turn_right(bool lastmovement);
void Motion4::step_forward();
```

```
void Motion4::step_back();
void Motion4::sit();
void Motion4::stand();
```

Ces méthodes contiennent la **liste** des positions des extrémités de chaque patte du robot. Ces informations sont ensuite envoyées à l'objet `ArmController`. La génération d'un mouvement élémentaire ou d'une attente fait appel à **deux méthodes** :

```
void Motion4::armMove(int index, Vectorf pos, bool wait);
void Motion4::armWait();
```

- ▶ La méthode `armMove()` prend en entrée les arguments suivants :
  - L'**index** de la patte qui va recevoir un ordre.
  - Le **vecteur position** vers lequel la patte doit aller
  - Un **booléen optionnel** (par défaut à faux) permettant de dire si le contrôleur doit **attendre** que les pattes soient arrivées à leur position cible avant d'exécuter la suite des ordres de sa file.
- ▶ La méthode `armWait()` permet d'ajouter un ordre d'attente indépendamment de la méthode `armMove()`.

**Contrôleur des pattes (ArmController)** Cette partie s'occupe de **coordonner** et de **contrôler** simultanément les **différentes pattes**. Concrètement, les ordres de déplacement ou d'attente des différentes pattes provenant de `Body` sont **ordonnés** sous forme d'une **file** (*queue* ou *FIFO*). Le contrôleur **dépile** ces ordres, les définit comme **positions cibles** et fixe la vitesse de déplacement pour chacune des pattes. Lorsqu'il dépile un ordre `WAIT`, il exécute le mouvement des pattes jusqu'à ce qu'elles arrivent physiquement à leur **cible finale** (estimation en boucle ouverte). Le processus **recommence** jusqu'à ce qu'il n'y ait **plus d'ordres** stockés en mémoire. C'est donc au niveau de ce contrôleur que l'on tient compte de la **vitesse d'exécution réelle** de chaque tâche par l'Arduino pour s'assurer de la bonne **synchronisation** des différentes pattes.

**Patte (Arm)** Cette partie gère le **mouvement** de l'**extrémité E** du bout de la patte entre sa **position actuelle** en  $(x, y, z)$  et la **position cible** voulue. La **vitesse** de déplacement de *E* est constante et fournie par `ArmController`. Pour respecter cette contrainte, un ensemble de points intermédiaires de passage est automatiquement généré. La **cinématique inverse** ainsi que les corrections issues de la **calibration** d'une patte sont calculées dans `Kinematics`.

**Moteurs (Motors)** La couche la plus basse porte sur la **simplification** du **contrôle** des **servomoteurs** en permettant de les manipuler sous forme d'une **liste** de 12 servomoteurs plutôt que d'y accéder par le numéro des pins respectifs où ils sont branchés.



**Conseil :** pour gagner en compréhension, **identifiez** où dans ce nouveau code pourrait se trouver chacune des étapes de **votre propre code**.

## 3.2 Réimplémentations dans la nouvelle architecture



Attention **ne pas modifier** les parties qui ne vous seront **pas demandées**.



Des **exemples de code** sont disponibles dans le dossier Doc du code fourni.

Parcourons le **fichier principal** du nouveau code, `spider.ino` :

```
#include <Arduino.h>
#include "motors.h"
#include "armcontroller.h"
#include "body.h"

// List of servo pins connected to the arduino
// Hip / Femur / Tibia
int servo_pin[12] = { 4,  2,  3,
                     7,  5,  6,
                     16, 14, 15,
                     19, 17, 18};

// Motors controller
Motors<MOTORS_NUMBER> motors;
// We choose 4 legs which each have 3 segments
ArmController<4, 3> armController{motors};
// Corrections which will be applied to the segments of the legs
Vectorf corrections[] = {{100, 70, 42}, // R2
                          {100, 70, 42}, // R1
                          {100, 70, 42}, // L1
                          {100, 70, 42}}; // L2

// We choose the spider model (all dimensions)
SpiderJDMI model;
// We use a 4-legged body
Body4 body{armController, model};
// The object that manages and stores the trajectory
Trajectory trajectory;

void setup()
{
    Serial.begin(115200); // Initialize the TLL-USB communication link
    motors.attachServo(servo_pin); // First thing to do
    body.init(corrections); // Compute the corrections given the measured calibration values
    trajectory.setup(); // Load the path to follow
}
```



Vous êtes prêts ?



## EXERCICE 14 (*Calibration Never Dies*)



Entrez les **valeurs de calibration** que vous aviez **mesurées** pour votre robot dans le **tableau** **correction** qui se trouve dans le fichier principal **spider.ino** :

```
Vectorf corrections[] = {{100, 70, 42}, {100, 70, 42}, {100, 70, 42}, {100, 70, 42}};  
// Replace values with your measurements
```



La méthode **addPosition** du contrôleur de patte permet d'ajouter des ordres qui seront exécutés pendant la méthode **process\_orders**. Il y a deux types d'ordre : **POS** et **WAIT**. Remplissez la fonction **loop** suivante pour que l'araignée prenne la pose de calibration corrigée :

```
void loop()  
{  
    for (int i = 0; i < 4; i++) {  
        armController.addPosition({Order::POS, i, 10000, {100, 70, 15}});  
        armController.addPosition({Order::WAIT});  
    }  
    armController.process_orders();  
    delay(1000);  
}
```

Programmez l'araignée, placez-la sur la mire et vérifiez que les mesures sont bonnes !

Une fois cet exercice validé, on va pouvoir réimplémenter les déplacements dans l'objet **Motion4**. Avant cela, vous pouvez remettre la fonction **loop** d'origine :

```
void loop()  
{  
    trajectory.reset(); // Reset trajectory position to 0  
    body.process(trajectory); // Convert the trajectory supplied into orders  
    armController.process_orders(); // Process last orders  
}
```

## EXERCICE 15 (*En avant !*)



Dans le fichier **motion.cpp** en vous inspirant de votre code précédent, implémentez les instructions pour faire faire un pas en avant à l'araignée avec les nouvelles méthodes.

```
void Motion4::step_forward();
```



**Un seul pas** Cette fonction sera appelée plusieurs fois par **Body** (déjà fourni) pour réaliser le nombre de pas voulu.



**Attention !** On a besoin d'une vitesse qui sera appliquée à chaque appel de **armMove**.

En fonction du **type de mouvement**, on préférera différents **types de vitesse**. Par exemple, il sera préférable d'aller **moins vite** si on bouge **plusieurs pattes** en même temps. Pour **changer la vitesse** vous devez modifier l'attribut **current\_speed** :

```
current_speed = model->leg_move_speed;
```

Plusieurs **vitesse de références** sont à votre disposition, on appellera ainsi :

- ▶ **model->leg\_move\_speed** // before moving a unique leg while moving backward or forward
- ▶ **model->body\_move\_speed** // before moving several legs at the same time while moving backward or forward
- ▶ **model->stand\_seat\_speed** // before standing or sitting
- ▶ **model->spot\_turn\_speed** // before moving one or several leg while turning left or right

### EXERCICE 16 (*En avant... marche*)



Pour tester votre code, il vous faut définir une trajectoire dans le fichier `trajectory.cpp`. Par exemple :

```
void Trajectory::setup() {  
    stand();  
    step_forward(1);  
    sit();  
}
```

Programmez le robot et testez pour faire un pas en avant puis plusieurs pas en avant.

### EXERCICE 17 (*En arrière aussi ! (Même pas peur...)*)



Implémentez la séquence pour reculer : celle-ci est le déroulement inverse de la marche en avant !

```
void Motion4::step_back();
```

Modifiez votre trajectoire pour tester ce nouveau mouvement avec un puis plusieurs pas.

### EXERCICE 18 (*Puis à gauche*)



Implémentez la séquence pour tourner à gauche en vous inspirant de votre code.

```
void Motion4::turn_left(bool lastmovement);
```



À noter que le paramètre `lastmovement` permet de spécifier dans le cas de plusieurs rotations s'il est nécessaire de remettre les pattes en position initiale (70, 50, -40). Celui-ci vaudra `true` dans ce cas.

Modifiez votre trajectoire pour tester ce nouveau mouvement avec un puis plusieurs pas.

### EXERCICE 19 (*Et enfin à droite*)



Implémentez la séquence pour tourner à droite, comme la marche avant / arrière, on effectue le mouvement inverse que quand on souhaite tourner à gauche !

```
void Motion4::turn_right(bool lastmovement);
```

Modifiez votre trajectoire pour tester ce nouveau mouvement avec un puis plusieurs pas.

### EXERCICE 20 (*Trajectoires tests*)



Essayez-vous sur des petites trajectoires de votre cru pour montrer que vous maîtrisez bien l'araignée !