

# Rapport de stage

*Lucie MATHÉ*

13 Mai - 13 Août 2019



# Table des matières

|  |           |
|--|-----------|
| <b>Introduction</b>                                    | <b>3</b>  |
| <b>1 Contexte du stage</b>                             | <b>5</b>  |
| 1.1 Rhoban . . . . .                                   | 5         |
| 1.2 La RoboCup . . . . .                               | 6         |
| <b>2 Description de l'existant</b>                     | <b>8</b>  |
| 2.1 Présentation du code . . . . .                     | 8         |
| 2.2 La création d'un mouvement . . . . .               | 8         |
| 2.3 Localisation sur le terrain . . . . .              | 9         |
| 2.4 Les outils de développement . . . . .              | 10        |
| <b>3 Tâches effectuées</b>                             | <b>13</b> |
| 3.1 Comportement de haut niveau : le gardien . . . . . | 13        |
| 3.1.1 Comportement par machine à états . . . . .       | 13        |
| 3.1.2 Découpage du terrain . . . . .                   | 15        |
| 3.1.3 La position de défense . . . . .                 | 16        |
| 3.1.4 Tests . . . . .                                  | 17        |
| 3.1.5 Résultat . . . . .                               | 18        |
| 3.2 Motricité : le Throw-in . . . . .                  | 20        |
| 3.2.1 La création du mouvement . . . . .               | 20        |
| 3.2.2 Les mains . . . . .                              | 21        |
| 3.2.3 Intégration au code . . . . .                    | 23        |
| 3.2.4 Résultat . . . . .                               | 24        |
| 3.3 Perception : la Localisation . . . . .             | 25        |
| 3.3.1 Expectation-Maximisation Algorithm . . . . .     | 26        |
| 3.3.2 Intégration au code . . . . .                    | 28        |
| 3.3.3 Visualisation . . . . .                          | 29        |
| 3.3.4 Résultat . . . . .                               | 29        |
| 3.4 Mécanique : les Kickers . . . . .                  | 31        |

|          |                              |           |
|----------|------------------------------|-----------|
| <b>4</b> | <b>La RoboCup</b>            | <b>33</b> |
| <b>5</b> | <b>Bilan</b>                 | <b>35</b> |
|          | <b>Bibliographie</b>         | <b>37</b> |
| <b>A</b> | <b>Présentation du robot</b> | <b>39</b> |

# Introduction

En 1997, le superordinateur Deep Blue a battu Kasparov aux échecs [Hsu, 1999] et le robot Mars Pathfinder fut le premier robot autonome à atterrir sur Mars [Stevenson, 1997]. Cette année-là, un nouveau challenge est apparu : créer des robots autonomes dotés d'intelligences artificielles qui concourent dans différentes épreuves. C'est ainsi qu'a eu lieu la première compétition internationale de robotique appelée RoboCup . Différentes équipes s'affrontent dans différentes ligues dans le but de faire évoluer les nouvelles technologies.

Le football est la ligue phare de la compétition, notamment la Standard Platform League avec les robots NAO [D.Gouaillier et al., 2009] mais on y retrouve aussi la Rescue, @Home, Industrial ou Junior<sup>1</sup>.

La ligue humanoïde de football a été créée en 2002 avec des tirs aux buts, les premiers matchs n'apparaissent qu'en 2005. À l'origine, cette ligue avait été créée suite à l'objectif lancé par H. Kitano en 1996 : “by the year 2050 develop a team of fully autonomous humanoid robot that can win against the human world soccer champions” [Kitano and Asada, 1999].

Au-delà de la seule intelligence, le sport nécessite des capacités motrices dans un environnement dynamique et seulement partiellement observable. Un robot qui joue au football rencontre des challenges pour se déplacer, observer et coopérer avec ses coéquipiers [Burkhard et al., 2002] (section Humanoid League).

Nous ne sommes pas encore prêts à battre des humains mais les robots Kid-Size jouent à 4 contre 4 sur un terrain de 9x6m, les premiers matchs à lumière naturelle ont eu lieu cette année et l'évolution est prévue pour compléter ce challenge [Baltes et al., 2019b].

L'équipe de Rhoban, dans laquelle j'ai effectué ce stage, a été créée en 2011. Elle évolue dans la ligue humanoïde en taille Kid-Size (robot entre 40

---

1. [www.robocup.org](http://www.robocup.org)

et 90cm). Dès sa 6ème participation, elle a su se hisser au sommet et cette année, elle a obtenu son quatrième titre consécutif.

La quantité de code pour faire fonctionner les cinq robots footballeurs de Rhoban est impressionnante et il est impossible en trois mois de stage de travailler sur tous les éléments du robot. Une des premières tâches a été de choisir sur quels projets je voulais travailler.

Mon travail s'est donc principalement partagé entre création du comportement du gardien, travail sur les tirs (notamment la touche), amélioration de la localisation et de l'outil de visualisation.

Après la RoboCup, j'ai eu l'occasion de travailler sur la création de mouvements de danse pour une exposition à CapScience<sup>2</sup>. L'objectif principal de ce stage a été de contribuer à l'évolution des robots afin de les rendre plus performant sur le terrain. J'ai donc préféré me concentrer sur mon travail pour la compétition et ne pas détailler mon investissement dans ce projet même si cela m'a permis de comprendre la difficulté de coopérer avec des gens extérieurs à l'équipe.

Après une contextualisation un peu plus détaillée sur Rhoban et la RoboCup dans le chapitre 1, j'exposerai chapitre 2 différents éléments du code nécessaires à la compréhension de mon travail. Dans le chapitre 3, je présenterai les majeures modifications que j'ai effectué. Enfin, le chapitre 4 résumera mon expérience de la RoboCup suivi d'un bilan du stage chapitre 5.

---

2. lien vidéo de la danse pour l'exposition de Cap Science : <https://photos.app.goo.gl/6fCoPqk7SyRB5sAL7>

# Chapitre 1

## Contexte du stage

### 1.1 Rhoban

J'ai effectué mon stage dans le cadre du projet Rhoban encadrée par des membres du LaBRI<sup>1</sup>. L'équipe avec laquelle j'ai travaillé durant ce stage développe des robots humanoïdes voués à jouer au football dans le cadre de la RoboCup.

Les robots utilisés sont appelés Sigmaban + [Ly et al., 2019], ils sont conçus pour évoluer dans la ligue Kid-Size et mesurent environ 65cm pour 6kg.

J'ai rencontré l'équipe lors du projet de programmation effectué au cours de mon Master 1 qui avait pour but de créer un outil de visualisation de matchs de football robotiques. J'avais pu alors découvrir une partie du code en travaillant sur les messages envoyés par les robots au cours du match.

Le stage s'est déroulé à l'Eirlab, fablab<sup>2</sup> de l'Enseirb qui met à disposition un terrain d'entraînement pour tester le comportement des robots et des machines-outils nous permettant de créer des pièces pour fabriquer les robots.

En effet, l'équipe de Rhoban ne limite pas son travail à l'informatique. Les Sigmaban sont prototypés et conçus par l'équipe et ils évoluent chaque année. Travailler au sein de l'équipe de Rhoban implique une grande partie de mécanique, il est donc nécessaire d'être polyvalent et de ne pas se limiter à travailler sur ordinateur.

L'Eirlab nous permet de travailler de façon communicative en s'entraînant, non de s'enfermer seul dans un bureau. En parallèle de Rhoban, l'équipe

---

1. Laboratoire Bordelais de Recherche en Informatique

2. un Fab Lab est un laboratoire de fabrication où est mis à disposition des machines permettant de concevoir et réaliser des pièces

de NaMeC, travaillant sur des robots de Small-Size League (SSL), préparait leur robot pour la RoboCup. Les Sigmabans étaient déjà prêts lorsque je suis arrivée, nous avons fait quelques changements mécaniques mais je n'ai pas pu assister au montage d'un de nos robots. En revanche, j'ai pu voir les différentes étapes de la création d'un robot de l'équipe SSL et même si je n'ai pas travaillé dessus, cela m'a aidé à comprendre les mécanismes de fonctionnement.

## 1.2 La RoboCup

La finalité du travail sur les robots est dévoilée lors des matchs pendant les différents tournois existants au cours de l'année. La RoboCup reste l'enjeu majeur mais, outre la compétition, c'est avant tout un évènement de recherche dont le but est de développer de nouvelles technologies.

En dehors de la compétition classique, la ligue humanoïde propose aussi des challenges techniques. Cette année, il y avait 4 challenges :

1. **High Jump** : Faire sauter le robot pour qu'il reste sans contact avec le sol le plus longtemps possible.
2. **High Kick** : Marquer un but avec la balle la plus haute possible
3. **Goal Kick From Moving Ball** : Marquer un but alors que la balle est en mouvement après une passe faite par un robot.
4. **Push Recovery** : le robot doit rester stable alors qu'il est heurté par un poids en marchant sur place.

Ces challenges nous permettent de travailler sur des problématiques de recherche qui nous poussent à améliorer les robots. Par exemple, le **Goal Kick From Moving Ball** représente une simple passe décisive mais, le vrai challenge a été d'ajouter un côté dynamique à la perception et à la prise de décision embarquée. Nous avons réussi ce challenge mais le mouvement n'a pas pu être effectué dans un match car il est plus difficile d'intégrer le mouvement dans un comportement de match que simplement le créer pour un challenge.

Nous avons également améliorer les tirs pour le **High Kick** (3.4).

La RoboCup a véritablement rythmé mon stage. Elle a eu lieu du 1<sup>er</sup> au 8 juillet à Sydney et a partagé mon stage en 3 phases : avant, pendant et après la RoboCup.

La phase de préparation représente presque les 2/3 de mon stage et représente presque tout le développement du code. Le départ représentait une date limite importante, il fallait que tout soit fini avant le premier match.

Durant les deux jours de préparation à Sydney, la partie la plus importante a été d'adapter les robots à leur nouvel environnement. Alors que la vision et la localisation étaient très efficaces à l'Eirlab, le changement de luminosité (plus faible sur les terrains intérieurs et plus forte sur ceux à lumière naturelle) et de terrains (poteaux moins large que les nôtres) a nécessité des ajustements.

La préparation nous a donc majoritairement servi à réadapter la vision des robots. Nous en avons également profité pour modifier les tirs afin qu'ils s'adaptent aux terrains sur place (changement d'herbe).

Pendant la compétition, il est nécessaire que pour chaque match les robots soient en condition de jouer et que les problèmes du match précédent aient été résolus. Nous avons eu 9 matchs sur les deux premiers jours de compétition ce qui nous a imposé un rythme assez épuisant.

Je n'avais jamais participé à des concours de type hackathon, assister à la RoboCup m'a permis de découvrir cette ambiance de travail avec des limites de temps très restreintes.

# Chapitre 2

## Description de l'existant

### 2.1 Présentation du code

Afin de mieux comprendre ce que j'ai fait, il est nécessaire que je fasse une brève introduction au code du projet.

Le package **Kid-Size** est celui qui contient tout le code permettant de faire fonctionner le robot, il se partage entre **Motion** et **Vision**.

Nous pouvons retrouver dans la partie **Motion** tous les mouvements des robots et la partie décisionnelle où il y a les stratégies de haut niveau. Nous avons donc dans la partie **Vision** le traitement de l'image et la localisation.

Au départ, je trouvais intéressant de comprendre le fonctionnement des mouvements du robot, d'où le choix de travailler sur le gardien et la touche. Ensuite, la localisation m'a permis d'avoir une vue d'ensemble car elle se base sur la vision pour impacter sur les stratégies de jeu.

Une des grandes difficultés a été de comprendre l'implémentation du code. Dès que je commençais un nouveau projet, il était nécessaire de prendre du temps pour assimiler les différents éléments dont j'allais avoir besoin. Je pense qu'il n'est pas essentiel de présenter tout le code que j'ai utilisé lors de mon stage. J'ai choisi donc d'exposer seulement les éléments nécessaires à la compréhension de mon travail.

### 2.2 La création d'un mouvement

L'implémentation des **Moves** est déjà prédéfinie dans le code de Rhoban et se présente sous la forme de :

- **OnStart** : action effectuée au démarage ;

- **OnStop** : action effectuée à la fin ;
  - **step** : action effectuée à chaque tic d'horloge du mouvement.
- OnStart nous sert souvent à initialiser l'état du robot pour le mouvement et OnStop pour rétablir l'état avant OnStart.

Un Move va englober toutes les actions que va faire le robot. On peut discerner différents types mouvements : certains sont mécaniques permettant d'activer certaines parties du corps (**Walk**, **Head**, **Arms**).

D'autres Moves comme **Kick**, **StandUp** permettent de lire des *splines* définissant des trajectoires pour chaque moteur, associant à chaque temps une position angulaire à atteindre. Entre les différents points de contrôle, la position est obtenue par interpolation linéaire entre le point précédent et le suivant. Nous pouvons changer le temps que prend chaque étape en influançant l'écoulement du temps dans le mouvement.

Enfin, il existe des Moves qui s'apparentent davantage à des comportements comme (**GoalKeeper**, **Playing**, **Robocup**) qui vont utiliser d'autres mouvements pour créer une stratégie de jeu.

La machine à états est utilisée régulièrement pour les comportements. Rhoban a créé une implémentation et comme pour les mouvements, nous avons deux fonctions **enterState** et **exitState** qui nous permettent d'agir sur les autres comportements à l'entrée et à la sortie de chaque état.

## 2.3 Localisation sur le terrain

Les coorodonnées des robots sur le terrain sont définies par un repère ayant pour origine le centre du terrain. L'axe des abscisses du terrain est défini en partant du centre du terrain jusqu'au centre des cages situées à droite du terrain. L'axe des ordonnées part du centre jusqu'à la ligne de touche opposée à la table d'arbitrage.

Dans le cas du gardien, le déplacement en **x** du robot se fera donc d'avant en arrière et en **y** sur les cotés, en parallèle à la ligne de but.

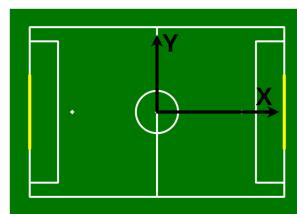


FIGURE 2.1 – Axes du terrain

## 2.4 Les outils de développement

De nombreux outils de travail sont fournis dans le code de Rhoban. Le plus utilisé est l'interface **RhIO** (Rhoban Input/Output) permettant de se connecter sur un robot lorsque celui-ci fonctionne. Cet outil est donc utilisé dès que l'on travaille sur un robot<sup>1</sup>.

L'interface **RhIO** utilise une architecture client/serveur [Rouxel et al., 2015]. Le client est le côté utilisateur, sous un interface de type terminal, il permet d'inspecter l'états de variables, de les modifier à la volée et de déclencher des **Moves**. Il est également possible de voir la vision du robot avec les différents filtres. Le serveur contenu à bord du robot, permet de stocker des paramètres et de répondre aux services demandés par l'utilisateur.

La modification des variables se fait par le *bind*. Pour pouvoir faire des réglages lors des phases de tests on utilise **Pull**. **Push** sert à afficher l'état des variables au cours du mouvement. Cela sert à rendre le code modulaire et changer rapidement les variables *bindées* pendant les phases de tests.

Par exemple, si nous voulons tester le **Throw-In** il suffit de changer le tir avec la commande `moves/kick/kickName = throwin` puis d'exécuter le tir avec **kick**.

Également, nous avons le **BehaviorViewer** (Figure 2.2) qui permet d'observer le comportement d'un robot sur le terrain. Il représente un terrain vu de dessus avec un robot, une balle et si besoin des obstacles. Nous pouvons les déplacer manuellement ou lancer des mouvements et des comportements. Il sert beaucoup lorsque l'on développe des stratégies de jeu (le **GoalKeeper** par exemple).

---

1. vidéo explicative : <https://www.youtube.com/watch?v=M0izgXYENLc&feature=youtu.be>



FIGURE 2.2 – BehaviorViewer

PyBullet est un outil de simulation 3D, à partir du modèle 3D d'un Sigmaban, nous pouvons observer les mouvements que nous développons (Figure 2.3).

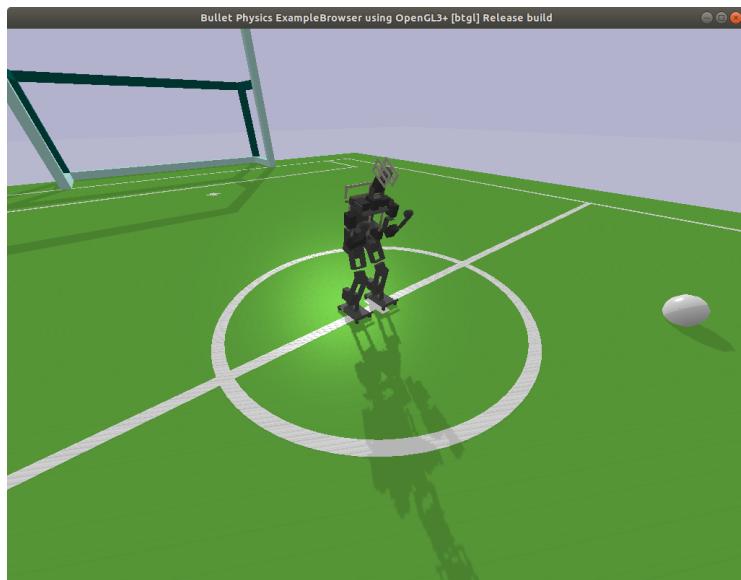


FIGURE 2.3 – PyBullet

L'outil `rhoban_monitoring` (Figure 2.4) que j'ai commencé pendant le projet de programmation permet de visionner les robots pendant le match. Nous pouvons y retrouver les informations utiles sur l'état des robots sur leur perception, localisation et leur comportement en cours. Nous pouvons également regarder des matchs en replay ce qui permet d'analyser les comportements à posteriori.

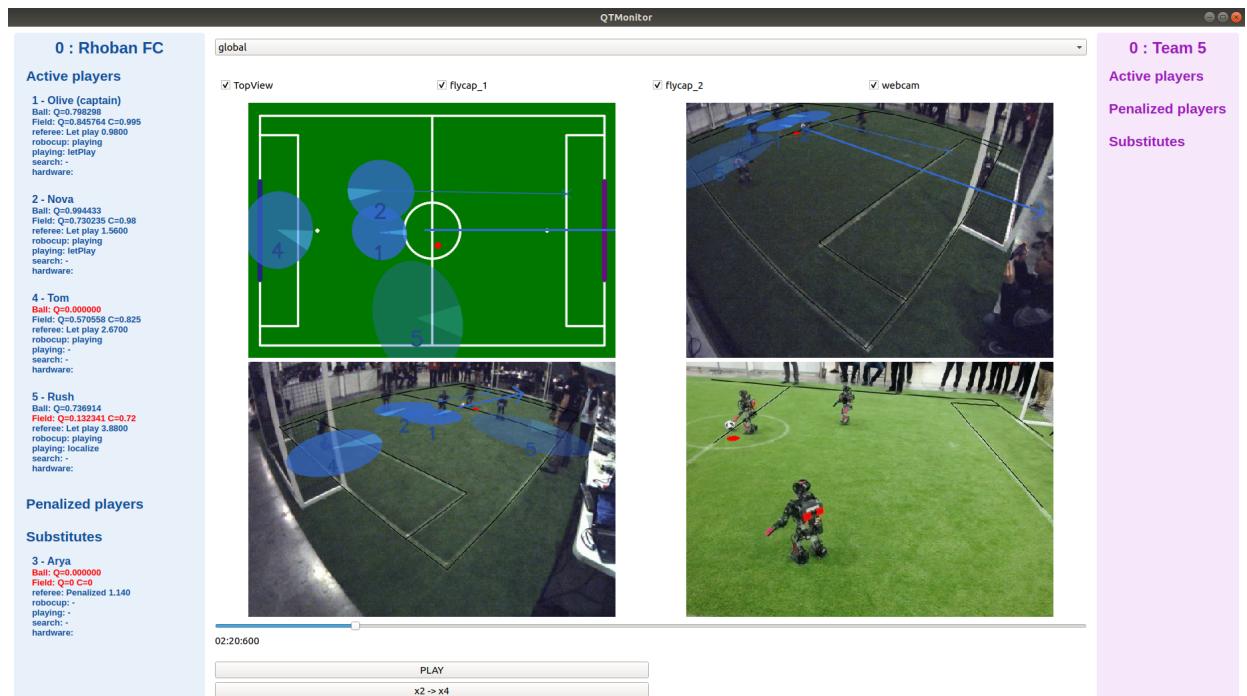


FIGURE 2.4 – `rhoban_monitoring`

Enfin, pour concevoir les pièces, le logiciel utilisé est OnShape, il permet de faire de la conception assisté par ordinateur.

# Chapitre 3

## Tâches effectuées

### 3.1 Comportement de haut niveau : le gardien

Faire le comportement du gardien fut ma première mission au sein de Rhoban. Cela m'a permis de comprendre le fonctionnement des différents mouvements (**Moves**) du robot. J'ai commencé ce projet lors du GermanOpen en amont de mon stage.

Le but du **GoalKeeper** est simplement de créer un comportement où le robot va se positionner avec le **Placer** à un endroit où il a une chance importante d'arrêter les tirs et où il pourra intervenir et dégager la balle avec **ApproachMove** lorsque c'est nécessaire. Il s'agit ici d'utiliser des mouvements déjà existants pour pouvoir avoir un joueur qui défend les cages. Le gardien ne plonge pas car les risques d'endommager le robot sont trop important, il joue davantage le rôle de dernier défenseur.

Il existait déjà un comportement défini pour le gardien mais celui-ci était complexe et avait été créé par un ancien membre. Il n'avait pas été actualisé avec les dernières modifications du code. L'équipe a donc décidé de le renouveler afin d'avoir un code clair et facile à modifier.

Le gardien de but divise son comportement en plusieurs états dont notamment : l'attente, le placement et le dégagement.

#### 3.1.1 Comportement par machine à états

Le gardien a pour but de défendre les cages, il doit pouvoir se placer en défense et dégager la balle lorsque celle-ci se rapproche. Le plus dur a été de déterminer les multiples situations dans laquelle le gardien allait être et donc de définir les différents états.

Lorsque la balle est loin des cages, le **GoalKeeper** se place à une position **home**, positionnée au centre des cages à 80cm devant les cages. Nous avons décidé de prendre cette position afin que le robot puisse facilement faire des dégagements.

Lorsque la balle est hors de danger, le robot retourne à cette position, c'est l'état **GO\_HOME**. Lorsque la balle se rapproche et qu'elle représente un danger, nous avons un état pour le placement en défense (**ALIGNBALL**). Pour ces deux états, nous avons un état d'attente **STOP** qui est un état de repos pour le robot.

Le dernier état représente celui d'attaque (**ATTACK**). Le gardien entre dans cet état dès que la balle s'approche à moins de deux mètres la position **home** et ne le quitte que lorsque la balle en sort.

La difficulté dans la représentation de la machine à état est que les états dépendent de la position de la balle. Toutes les transitions arrivant dans un état ont la même étiquette. J'ai donc fait une version simplifiée Figure 3.1 où à chaque état correspond une transition indiquée en dessous.

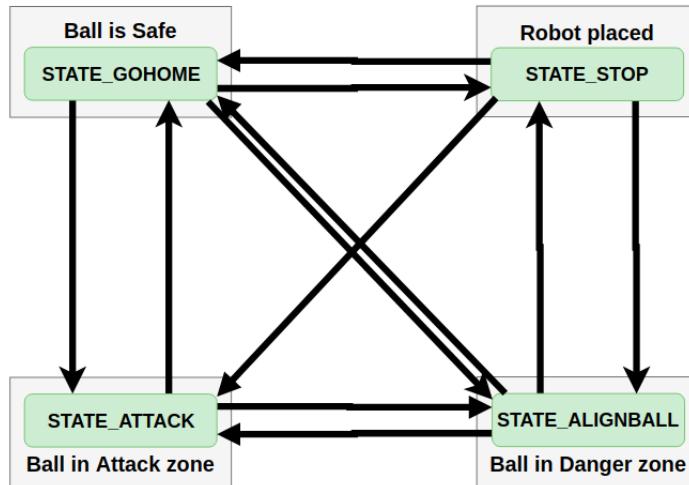


FIGURE 3.1 – Graphe explicatif des états du gardien

La seule transition que l'on ne peut pas effectuer est de l'état **ATTACK** à l'état **STOP** car après une attaque, le robot va devoir automatiquement se replacer soit en position **home** soit en défense.

La vision des robots ne permet pas de détecter d'autres robots. On ne peut pas vérifier par la vision la position des autres robots et on préfère donc que le gardien joue sa stratégie indépendamment en ne prenant en compte que la balle afin d'assurer que la défense soit efficace.

### 3.1.2 Découpage du terrain

Nous voyons donc que le comportement du gardien dépend majoritairement de la position de la balle par rapport à ses cages. On partage donc le terrain en fonction des trois zones pour les trois comportements du robot :

1. **Safe Zone** (en blanc) : la balle est trop loin des cages, le robot est placé en **home**, position d'attente, prêt à intervenir ;
2. **Danger Zone** (en jaune) : la balle se rapproche, le robot se place ;
3. **Attack Zone** (en rouge) : le gardien fait un dégagement.

Nous voyons aussi sur la figure 3.2 la croix noire marquant la position **home** du robot dont on a parlé précédemment.

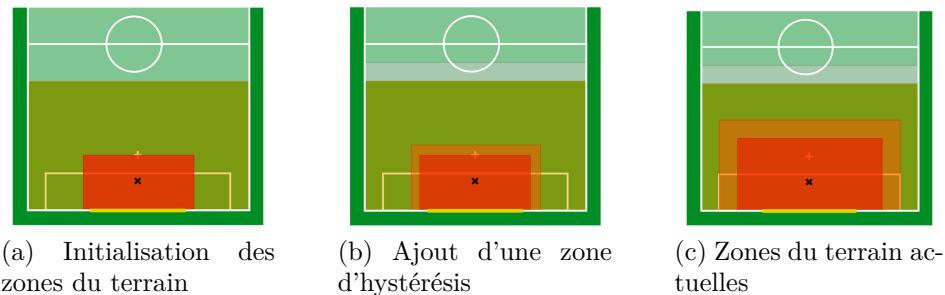


FIGURE 3.2 – Définition des zones du terrain

La localisation nous donne une mesure de la réalité mais la position de la balle et du robot n'est qu'une estimation, elles peuvent varier même si le robot et la balle restent statiques. Pour éviter des oscillations entre les différents états lorsque ces positions sont aux limites des zones, nous introduisons des zones d'hystérésis (en orange et gris image 3.2b) empêchant ces changements de zones. Nous avons créés un espace de 25cm (puis 50cm) entre la zone de danger et la zone d'attaque et 50cm entre la zone avec et sans danger.

Il faut noter donc que la zone d'hystérésis est là pour faire "tampon" et contrer l'imprécision de la position de la balle. Il n'y aura pas de changement d'état dans cette zone, le robot garde celui qu'il avait avant que la balle n'entre cette zone.

Lors de la compétition, nous nous sommes rendus compte que la zone du gardien était trop petite. En effet, il est difficile d'anticiper les niveaux des équipes adverse et donc la taille des zones souhaitées. Comme toutes les variables définissant ces tailles sont en *binding* sur RhIO , il a été facile d'ajuster les dimensions sur place. Les équipes tirant de plus en plus loin, nous avons élargi la zone d'attaque afin que le gardien fasse plus de dégagements. (Figure 3.2c).

### 3.1.3 La position de défense

La dernière difficulté rencontrée pour la création du gardien a été de déterminer la position dans laquelle il devait se positionner pour contrer la balle lorsque celle-ci s'approche des cages, dans l'état **ALIGNBALL**. Le but ici était d'obtenir des positions permettant d'intercepter un maximum la balle pour des tirs tout en ayant une facilité à attaquer pour dégager la balle.

Tout d'abord, on a commencé définir la zone dans laquelle le gardien devrait se déplacer.

L'ancien code du gardien utilisait une machine à états pour le fonctionnement du gardien. J'ai donc décidé d'en recréer une pour le nouveau gardien.

D'après les règles de la Robocup [Baltes et al., 2019a], la zone du gardien a une profondeur de 1m et s'étend en largeur jusqu'à 1.2m de chaque côté des poteaux (taille totale de 5m). Étant donné la vitesse de déplacement du robot, j'ai jugé qu'il était amplement suffisant que le robot ne s'avance pas davantage que la position *home*. En revanche, il faut que le robot évite toute collision avec les poteaux, je laisse donc une marge de 40cm entre la ligne de but et le gardien.

La zone du gardien fait 5m de large alors que les cages n'en font que 2.6m. J'ai considéré qu'il n'était pas nécessaire de positionner le gardien en défense à l'extérieur de la zone définie par les deux poteaux.



FIGURE 3.3 – Zone de placement du gardien

J'ai reporté sur la Figure 3.3 les mesures données par le règlement ainsi que la zone de défense choisie.

Pour déterminer la position du robot, nous créons une ligne entre la balle et le centre de la ligne de but. Nous regardons l'intersection entre cette ligne et la zone de défense du gardien.

Nous avons alors trois cas. Si cette ligne coupe la zone sur l'axe de `home.x`, nous gardons alors cette position (Figure 3.4a).

Si ce n'est pas le cas, nous prenons l'intersection de la ligne avec les largeurs de la zone de défense (lorsque  $|y| = \text{field.goal\_width} / 2.0$ ) (Figure 3.4b) .

Enfin, il est possible que la droite n'intersecte pas avec la zone du gardien. Dans ce cas, la balle se situera forcément sur les bords du terrains (sinon on serait en **ATTACK**, non en **ALIGNBALL**), la position du robot sera donc sur les corners inférieurs de la zone de gardien (Figure 3.4c).

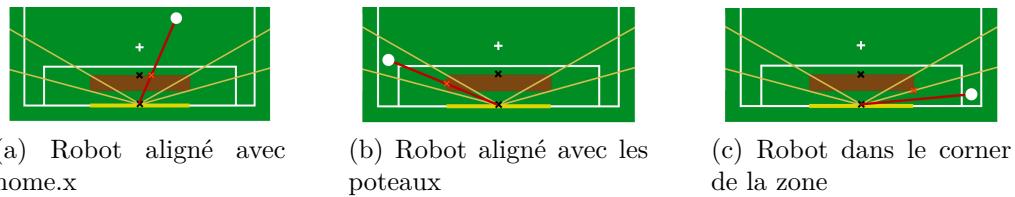


FIGURE 3.4 – Définition des positions du gardien

Enfin, nous avons ajouté une orientation à la position du robot, très utile si la balle se trouve sur les couloirs extérieurs. Nous calculons très simplement cet angle grâce à la fonction `atan2` avec la distance en **x** et en **y** entre le robot et le ballon.

### 3.1.4 Tests

Il est difficile d'établir des tests unitaires pour les mouvements dans l'architecture logicielle actuelle. En revanche, nous pouvons aisément effectuer des tests à la main.

Grâce au **BehaviorViewer**, nous avons pu faire des tests en déplaçant la balle et en observant le comportement du gardien.

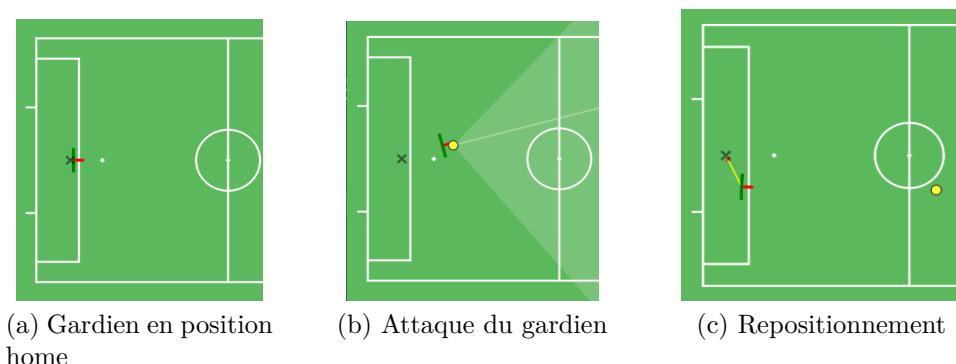


FIGURE 3.5 – Test d'attaque du gardien

Le premier test montré sur la Figure 3.5 portait sur le bon fonctionnement du gardien, le changement d'état en fonction de la position de la balle. Ici nous pouvons voir une phase d'attaque puis de remplacement suite au dégagement.

Sur la Figure 3.5b, la zone éclaircie correspond à l'angle de tir autorisé. En effet, le but du gardien est de faire un dégagement rapide, il y a donc plus de tolérance sur l'angle de tir.

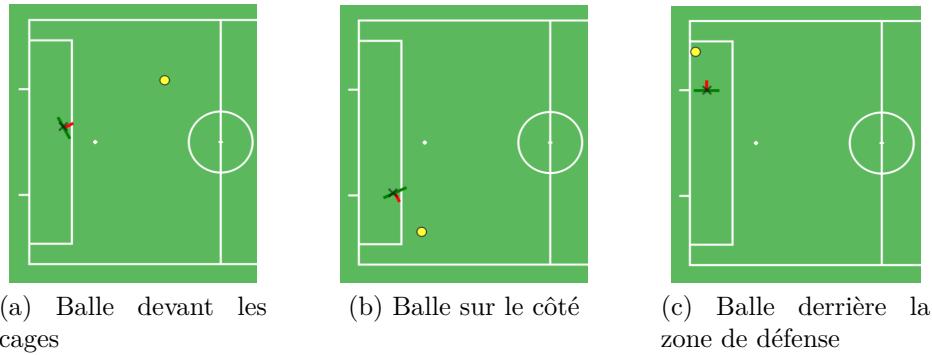


FIGURE 3.6 – Test de positionnement du gardien

Ensuite nous avons testé si le placement du gardien en zone d'attaque était satisfaisant, comme nous le montrent les images de la Figure 3.6.

Nous voyons que les résultats trouvés sur le **BehaviorViewer** correspondent à ceux prévus en Figure 3.4.

### 3.1.5 Résultat

Nous avons pu tester plusieurs fois le gardien au cours des différents matchs d'entraînement et lors de la compétition.

Dans les premiers tests, le robot était sans cesse entraîné de se repositionner. Nous laissons une marge entre la distance entre la position souhaitée et actuelle car la précision de localisation n'est pas parfaite. Le problème était que les marges de précision étaient supérieures dans mon code du gardien par rapport à celle du **Placer**. Une rectification de ces marges à l'initialisation du mouvement du gardien a rapidement réglé ce problème.

Pendant la compétition, nous avons donc agrandi la zone d'attaque comme expliqué précédemment mais le problème majeur du gardien est resté la localisation.



FIGURE 3.7 – Gardien mal placé lors de la finale : on voit le robot à gauche des cages pourtant celui-ci se localise au milieu (ellipse rose)

En effet, la localisation des robots se fait par les poteaux des cages, il est difficile pour le gardien de les voir car il se situe juste devant. Pendant la compétition, la localisation était compliquée (changement de luminosité et de poteaux) et le gardien en a subit les conséquences sur les premiers matchs. Le gardien a même abandonné en cours de match par manque d'informations (lorsque le robot se met en position d'abandon, l'arbitre nous permet de le récupérer, on peut alors le replacer au bord du terrain ce qui lui permet de se relocaliser)

Le dernier jour, nous avons réussi à régler certains problèmes de vision et le robot a réussi à maintenir une certitude importante pendant le match. En revanche, la position du gardien a eu tendance à dévier car le robot voyait toujours un poteau sur le côté et pensait qu'il était toujours bien placé.

Nous avons encaissé 11 buts durant la compétition mais nous n'avions pas toujours de gardien sur le terrain soit car il était sorti suite à une perte de localisation soit car il nous manquait un robot et que nous préférions avoir 3 robots en attaque. Lorsqu'il était présent, il a réussi à dégager plusieurs fois la balle, notamment la dernière avancée des MRL à la fin des prolongations du match de quart finale, but qui aurait pu nous donner la défaite.

C'est également lors de ce match que notre gardien a pris deux buts sur des tirs de longue distance. La réaction du gardien était trop lente, il n'arrivait pas à se déplacer à temps. Une solution envisagée pour l'année prochaine est soit de faire plonger le gardien (mais le risque d'endommager des moteurs est important) soit d'effectuer un grand pas rapide sur le côté.

## 3.2 Motricité : le Throw-in

Cette année, le règlement de la Robocup ayant changé [Baltes et al., 2019a], nous avons maintenant le droit de faire les touches à la main. Cela n'avait jamais été fait auparavant chez Rhoban, il y avait donc tout à faire.

Le travail se décomposait en deux étapes : la création du mouvement et l'intégration au comportement des robots en match.

### 3.2.1 La création du mouvement

Comme expliqué en 2.2, les mouvements de tir sont créés à partir de *splines*, j'ai donc commencé par faire un code simple permettant de lire des *splines* puis j'ai créé le mouvement au fur et à mesure : attraper la balle surélevée, attraper la balle au sol, porter la balle jusqu'au dessus de la tête et enfin la jeter.

J'ai défini 10 étapes qui rythment les *splines* : initialisation(3.8a), plier les genoux, déplier les bras, basculer en avant (3.8b), prendre le ballon, redresser le torse(3.8c), lever les bras et déplier les genoux (3.8d), reculer les bras derrière la tête(3.8e), jeter(3.8f), retour au départ<sup>1</sup>.

Chaque étape du Throw-In a été testé dans un premier temps avec l'outil de visualisation PyBullet puis sur le robot. Pour tester les *splines*, nous utilisons un mode appelé `manualT` qui nous permet de contrôler le temps dans le mouvement. Grâce à ça, il est possible de ralentir le temps qui s'écoule ou revenir en arrière pour réitérer une partie du mouvement en modifiant les *splines*.

---

1. Lien vidéo du mouvement de Throw-In : <https://photos.app.goo.gl/QvVRzVezJ5QaGYY98>

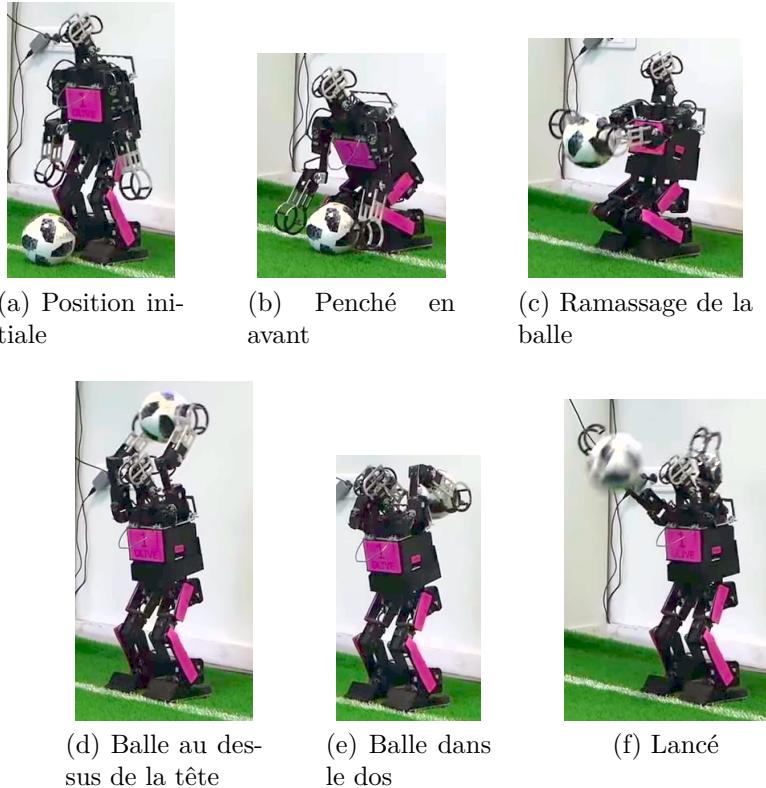


FIGURE 3.8 – Différentes étapes du Throw-in

Le problème majeur a été pour prendre le ballon, il manque un degré de liberté sur les bras afin que le robot puisse aisément porter la balle. Le robot ne peut plier les coudes que vers ses épaules (`pitch`) et non vers l'intérieur (`roll`), comme on le voit sur le schémas du robot Annexe A. Pour resserrer les mains vers le ballon, le seul moteur à solliciter est `shoulder_roll`.

L'autre difficulté a été de garder une bonne stabilité du robot alors que le robot était penché en avant puis déséquilibré avec le poids du ballon et avec l'impulsion du lancé.

### 3.2.2 Les mains

Afin que le robot porte plus aisément la balle, j'ai dessiné de nouvelles mains. En effet, les anciennes mains avaient été conçues dans l'unique but d'aider au relevage du robot, la forme de la main ne permettait pas de prendre correctement la balle.

Comme nous voyons figure 3.9, la surface pour prendre la balle était faible et nous obligeais à la porter en forçant sur les moteurs des épaules.



FIGURE 3.9 – Modèle 3D de l'ancienne main sur OnShape

Nous avons donc décidé de créer de nouvelles mains plus adaptées pour pouvoir mieux englober la balle lorsqu'on la tient. La difficulté ici était d'arriver à faire des mains solides, permettant de porter la balle et de se relever sans problème.

L'idée développée a donc été de faire deux plaques (une vissée de chaque côté du moteur) reliées par des entretoises. Nous avons donc une plaque avec d'un côté une encoche pour le moteur et de l'autre un large trou permettant d'encercler la balle.



(a) Premier prototype en bois



(b) Plaque en aluminium actuelle

FIGURE 3.10 – Évolution des mains

Après un premier prototype en bois fait avec la découpeuse laser, nous avons allégé les plaques et agrandi le diamètre de trou pour la balle. La version finale a été découpée dans des plaques d'aluminium de 3mm.

Les tests ont été satisfaisant sur la prise du ballon et nous n'avons pas eu à changer le **Stand-Up**, les mains étaient assez solides pour supporter le poids du robot.

La dernière modification faite a été de mettre du scotch noir sur l'aluminium. L'aspect brillant des plaques et des entretoises créait de la confusion dans la vision du robot car il y voyait des balles.



FIGURE 3.11 – Nouvelles mains

### 3.2.3 Intégration au code

Pour intégrer le Throw-In au comportement des joueurs, nous avons commencé par l'intégrer au mouvement Kick car le Throw-In est avant tout un tir, même s'il se fait avec les mains.

Il n'y avait qu'une condition de jeu dans laquelle nous souhaitions que le tir effectué soit un tir avec les mains : lorsque l'arbitre donne une touche en notre faveur. Lorsque cette condition est respectée, le tir désigné est alors "throwin".

#### Les bras

Une fois que le Throw-In ait été ajouté au comportement, nous avons alors remarqué quelques problème que l'on avait pas avec les autres tirs. Le premier assez évident était la position initiale des bras. Lorsque nous avons testé le mouvement, nous commençons de la position initiale (robot droit et bras le long du corps, 3.8a). La position de la marche est assez différentes étant donné que les bras sont totalement repliés, les mains au niveau des épaules (3.12a).

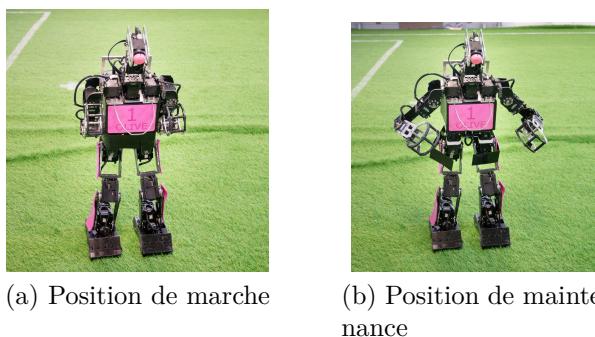


FIGURE 3.12 – Différentes positions des bras

Pour contrer cela, nous avons créé un mouvement de bras à part entière. Auparavant, les mouvements de bras étaient intégrés au mouvement **Walk**. Nous avons donc trois positions pour les bras : dépliés comme à l'initialisation, pliés comme pendant la marche et une position de maintenance (3.12b) où le robot écarte les bras pour accéder facilement à la ceinture du robot où se trouve la batterie par exemple.

Séparer les mouvements de bras de la marche nous a permis de les activer/désactiver plus facilement dans les mouvements tels que le **Stand-Up** ou le **Kick**.

Nous avons également désactivé la tête pendant le tir car nous n'en avions pas besoin et que celle-ci tapait dans les bras lorsqu'ils étaient en l'air.

### Le Stand-Up

Nous avions un second problème car la détection de la chute est basée sur la centrale inertuelle. Lorsque le robot se penchait en avant, le robot pensait qu'il était entrain de tomber et commençait le mouvement de relevage.

Nous avons simplement empêcher l'activation du mode **fallStatus** lors des tirs. Cela nous permet de finir entièrement le tir sans que le robot puisse estimer avoir fait une chute. Comme le robot n'est plus penché à la fin du **Throw-In**, le **Stand-Up** ne s'active pas. Pour les autres tirs, il nous semble préférable que le **Kick** finisse en cas de chute car il peut quand même toucher le ballon.

### 3.2.4 Résultat

Une fois tous les problèmes d'intégration réglés, nous avons pu faire des tests lors des matchs d'entraînement qui nous ont permis de régler les derniers problèmes notamment avec l'activation/désactivation des bras.

Pour la réalisation en match, nous avons manqué d'occasion car il fallait que l'équipe adverse sorte la balle du terrain et ça a été plutôt rare. La première fois, le tir a été annulé à cause d'une fonctionnalité sur le tir ajouté juste avant la compétition et non testée sur le **Throw-In** qui permettait d'annuler le tir lorsque la balle se déplaçait. Nous avons donc empêcher cette fonctionnalité sur la touche.

La seconde fois, le robot totalement perdu a jeté la balle dans le public.

Nous avons enfin réussi le tir lors de la finale, ce qui a donné une magnifique passe au robot placé en attaque<sup>2</sup>. C'était la première fois qu'une touche à la main a été vue en match dans l'histoire de la RoboCup<sup>3</sup>. La difficulté à réaliser le Throw-In en match nous montre la grande différence entre le code, la simulation et la réalité.

La touche semble donc maintenant fonctionnelle, l'amélioration intéressante à ajouter serait un moteur en `roll` pour les coudes afin de lancer la balle plus loin.

Également, il serait intéressant d'ajouter le Throw-In au gardien afin qu'il puisse faire des dégagements à longue distance. Le gardien peut porter la balle dans ses mains seulement s'il se trouve dans la zone du gardien. Nous estimons que la localisation du gardien n'est pas assez bonne et qu'il risque de prendre la balle en se croyant dans cette zone alors qu'il ne l'est pas. La taille du terrain devrait augmenter l'année prochaine [Baltes et al., 2019b], la taille de la zone du gardien devrait donc changer, ce qui rendrait possible le dégagement à la main pour le gardien.

### 3.3 Perception : la Localisation

Après avoir travaillé sur la Motion, j'ai décidé de changer de domaine, le but de ce stage étant de découvrir un maximum de choses dans le fonctionnement d'un robot.

La localisation se fait par un filtre particulaire [Doucet and Johansen, 2009]. Nous avons donc 3000 particules qui vont être créées à partir des informations de la vision et de l'odométrie. À partir de ces particules, nous récupérons la position du robot.

Avant mon travail sur la localisation, nous prenions la valeur moyenne, or, cette solution était très limitée dans certains cas, notamment lorsque l'on remet un robot en jeu après une pénalité. En effet, nous devons replacer le robot sur la ligne de touche en face du point de penalty mais, il peut être de n'importe lequel des deux côtés du terrain.

Nous pouvons observer l'état des particules dans RhIO à partir de la TopView.

---

2. lien vidéo du Throw-In en finale :<https://twitter.com/LauRn964/status/1147733578211479553>

3. le Throw-In a été réussi en Technical Challenge en 2011 mais il y a une différence de difficulté entre le challenge et la réalisation en match, comme on le voit d'ailleurs avec le Goal Kick From Moving Ball que nous n'avons pas pu faire en match cette année

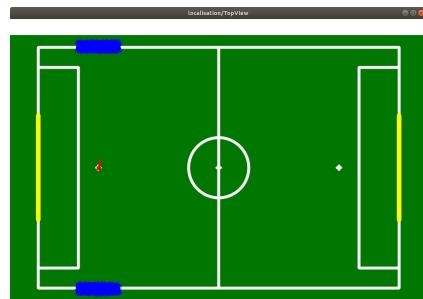


FIGURE 3.13 – Résultat borderReset avant l’algorithme EM

En bleu nous observons les différentes propositions et en rouge la position renvoyée. La figure 3.13 nous montre les particules après un `borderReset`<sup>4</sup>.

Nous voyons qu’en aucun cas, il n’est possible que le robot soit sur le point de penalty étant donné qu’il n’y a aucune particule à cet endroit.

Faire la moyenne n’est pas la bonne solution pour récupérer la position du robot surtout quand comme ici, deux positions symétriques se démarquent. Pour empêcher ce problème, nous avons décidé de regrouper les particules, nous créons alors plusieurs *clusters* représentant les différents amas de particules et donc les différentes positions pensées par le robot.

### 3.3.1 Expectation-Maximisation Algorithm

L’algorithme expectation-maximisation nous permet de récupérer ces *clusters* en calculant de manière itérative les maximums de vraisemblance de chaque groupe. Lors de notre calcul de l’algorithme EM, nous prenons seulement en compte la position des particules, non leur direction.

Nous commençons donc par séparer les particules en deux vecteurs : un avec les positions en  $x$  et  $y$  et l’autre avec les angles.

Pour effectuer cet algorithme, nous utilisons la fonction OpenCV EM (`cv::ml::EM`) qui nous permet à partir d’un vecteur et d’un nombre de *clusters* choisi de donner un label à chaque particule, nous permettant donc de les regrouper.

Le but ici est donc de déterminer combien de *clusters* représente notre état actuel du filtre particulaire. À chaque itération de la fonction EM, nous calculons l’amélioration des variances des groupes obtenus et nous augmentons le nombre de *clusters* tant que cette amélioration est significative (ligne 3).

---

4. le `borderReset` correspond à l’action sur le filtre particulaire qui correspond au retour sur le terrain après une pénalité

Dans notre algorithme 1, nous avons ligne 6 l'appel à notre fonction EM-TRAINEDLABEL qui initialise le modèle de la fonction OpenCV EM et qui lance l'entraînement.

La fonction GETCLUSTERS de la ligne 7 transforme les vecteurs de positions et d'angles en *clusters* : une `std::map` de `Point` et une `dAngle` (ce sont tous les deux des objets de `rhoban_geometry`).

Enfin, après avoir calculé les variances actuelles et précédentes en faisant la moyenne des variances de chaque *cluster*, nous obtenons lignes 8 et 9 l'amélioration des variances globales pour la position et l'angle.

À la fin de la boucle **while**, nous avons utilisés donc les derniers *clusters* pour lesquels l'amélioration a été significative, c'est-à-dire ceux de l'itération précédente (stockés en début de boucle ligne 4).

L'algorithme 1 retourne à la fin nos différents *clusters* triés en fonction du nombre de particules qu'ils représentent. Pour avoir la localisation du robot, nous prenons donc la position et la direction moyenne du premier *cluster*.

---

#### Algorithm 1 EM algorithm

---

```

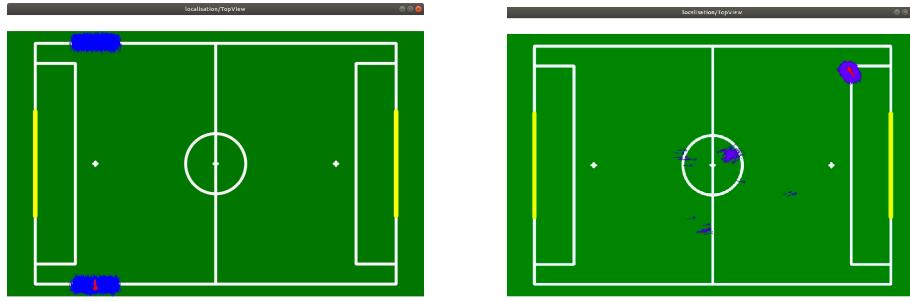
1: function UPDATEEM(position, angle, maxCluster)
2:   Initialization           ▷ nbCluster = 0, Vars/VarReductions = 1
3:   while posVarReduction > 0.5 || angleVarReduction > 0.5 && nb-
   Cluster <= maxClusters do
4:     Stock old clusters (posCluster, angleCluster)
5:     nbCluster ← +1
           ▷ label all positions with nbCluster clusters
6:     Call EMTRAINEDLABELS(positions, nbCluster, &label)
           ▷ create new clusters with new labels
7:     Call GETCLUSTERS(positions, angles, labels, &posClusters, &an-
       gleCluster)
           ▷ get new variance improvement
8:     posVarReduction ← 1 - (posVar / stockedPosVar)
9:     angleVarReduction ← 1 - (angleVar / stockedAngleVar)

10:    Call EXPORTTOCLUSTER(posCluster, angleCluster)
11:    return cluster
12:    if nbCluster > 1 then
13:      sort(cluster)
14:    return cluster

```

---

## Résultat



(a) Résultat borderReset avec l'algorithme EM

(b) Exemple en match

FIGURE 3.14 – Localisation avec l'algorithme EM

Maintenant, après un `borderReset`, nous voyons que le robot choisit un côté du terrain (3.14a). Cela nous permet donc d'avoir une position plausible pour le robot.

En match (3.14b), nous voyons que les particules plus éloignées qui peuvent correspondre à des positions pensées suite à un faux positif ou une ancienne divergence de position sont totalement ignorées.

La modification du filtre particulaire nous permet donc d'avoir un meilleur positionnement du robot.

### 3.3.2 Intégration au code

La position des robots est partagée grâce aux messages qu'ils envoient. Ces messages sont au format `protobuf`. Nous avons décidé que les *clusters* renvoyés par la fonction `UPDATEEM` seraient directement en `protobuf` pour faciliter l'acheminement des données. La fonction `EXPORTTOCLUSTER` à la ligne 10 de l'algorithme 1 est en fait une fonction `EXPORTTOPROTO`.

Les informations sur la position fournie dans les `protobuf`s sont :

- la probabilité de la pose.
- une position  $x$  et  $y$
- une matrice de covariance associée à la position
- une direction en radian
- l'écart-type sur cette direction

Ces informations sont facilement calculables à partir des derniers *clusters* stockés. La probabilité de la pose est calculée en fonction du nombre de particules qu'elle représente.

L'avantage de ces `WeightedPose` c'est qu'elles sont définies comme `repeated`, c'est à dire que nous pouvons en avoir plusieurs par message et donc nous stockons tous nos *clusters*.

Avant nous utilisions qu'une seule position dans les messages, nous avons donc dû rajouter des fonctions pour récupérer ces `WeightedPose` depuis la localisation jusqu'à la création des messages des robots.

### 3.3.3 Visualisation

Alors que nous utilisons seulement la position définie par le regroupement de particules majoritaire, nous stockons tous les *clusters*. En effet, nous utilisons les autres *clusters* afin de mieux comprendre la position du robot et l'hésitation qu'il peut avoir entre deux positions.

Avant, l'outil de visualisation de match `rhoban_monitoring` affichait des cercles avec un trait pour la direction pour représentait le robot (3.15a). Maintenant que nous avons des positions pondérées, nous pouvons afficher la position avec des ellipses, ce qui nous permet d'avoir une meilleur perception de l'incertitude du robot.

Pour créer l'ellipse de la position, nous récupérons à partir de la matrice de covariance deux axes représentant le plus grand et le plus petit diamètre de l'ellipse. Ensuite, nous utilisons la fonction OpenCV `GETELLIPSEPOINTS` nous permettant d'avoir un vecteur de points à partir desquels on la dessinera.

De plus, nous avons dû ajouter la direction du robot. Pour cela, on calcule un intervalle de confiance en prenant en compte l'écart-type puis on redéfini le vecteur de points mais en donnant en argument un angle maximum et minimum. Comme la direction représente un cône et non pas une ellipse, nous utilisons la fonction OpenCV `FILLCONVEXPOLY` pour dessiner les points de l'ellipse de la direction auquel on aura ajouté le centre pour que ça représente un cône.

Nous avons également ajouté de l'opacité représentant l'incertitude du robot. Plus une position sera sûre, plus elle sera petite et opaque. À l'inverse, une position très incertaine sera très large (les particules seront davantage dispersées) et peu visible.

### 3.3.4 Résultat

Sur la version précédente de l'affichage 3.15a, nous ne pouvions pas voir que le robot n°2 était totalement perdu alors qu'avec le nouvel affichage 3.15b, l'opacité est tellement faible qu'on ne voit pas la position.

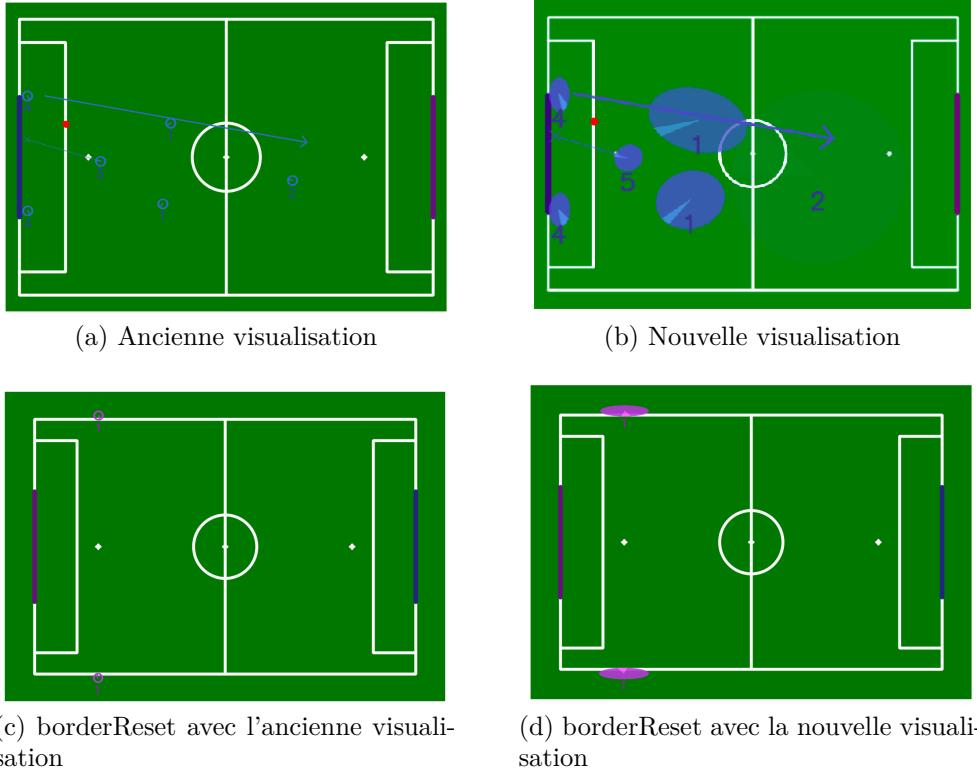


FIGURE 3.15 – Évolution de rhoban\_monitoring

Pour le robot n° 1, nous voyons qu'il hésite entre deux positions mais que celle située en haut est moins probable. Nous pouvons aussi voir que le robot n° 5 est confiant sur sa localisation.

Ici, nous ne voyons la localisation que sur une vue de dessus mais, lorsque nous avons des plusieurs caméras (Figure 2.4), nous pouvons reprojeter ces ellipses sur les images et donc estimer la qualité de la position fournie par le robot.

Également, pour le **borderReset**, nous voyons que cela représente mieux l'étalement des particules que l'on a pu observer en 3.14a.

Pendant les matchs, cette amélioration de la visualisation permet de mieux comprendre les robots et de mieux cerner les problèmes que l'on a. Par exemple, il est fréquent qu'il y ait deux positions probables après la chute d'un robot. On peut maintenant distinguer le cas où les particules sont dispersées par manque d'observation et celui où les observations ne permettent pas de trancher entre les deux possibilités.

### 3.4 Mécanique : les Kickers

Pour améliorer notre distance de tir mais surtout pour le challenge technique du High Kick, nous avons décidé de refaire les kickers, l'embout du pied en plastique qui va entrer en contact avec le ballon.

Le précédent kicker avait pour but de taper dans la balle en lui donnant une rotation pour qu'elle roule sur l'herbe. Maintenant l'objectif est de lever la balle au maximum, il fallait donc une tout autre approche.

Après quelques essais s'inspirant de la version d'origine, nous avons décidé de prendre les mesures de la réalité afin de moduler chaque arête en fonction des contraintes réelles.

Sur OnShape, nous avons créés la version finale en dessinant la balle et les crampons du pied qui nous donnait une limite de hauteur à ne pas dépasser pour éviter que le pied touche le sol lors du tir.

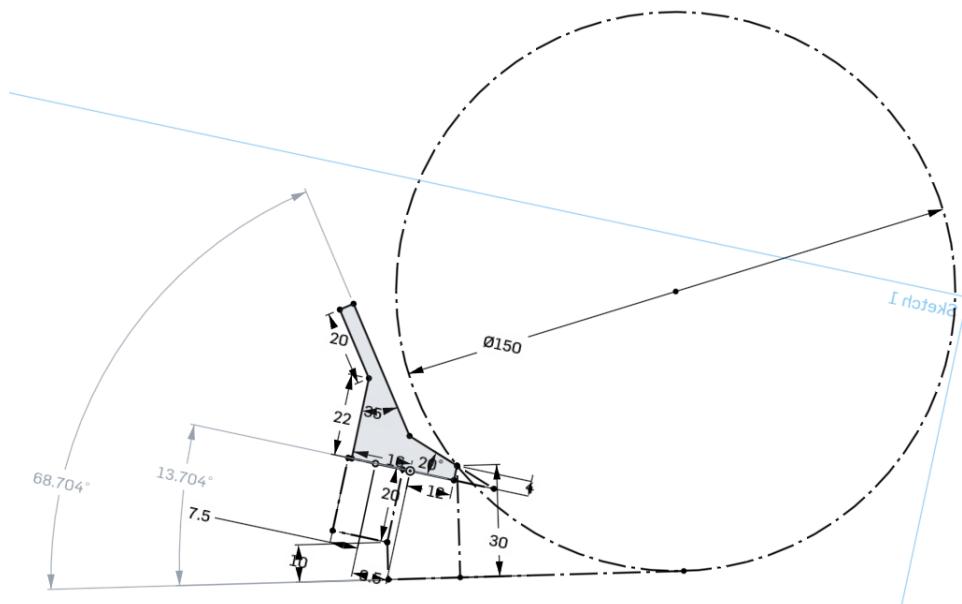
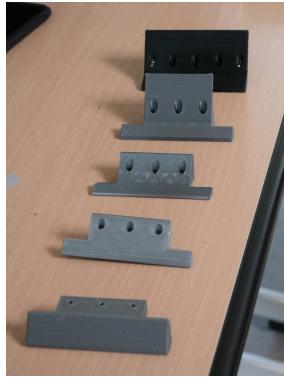


FIGURE 3.16 – Dessin du kicker sur OnShape

Nous pouvons voir la partie grisée qui représente le kicker. Grâce à cette modélisation du ballon, du sol et des crampons du robots, nous avons pu ajuster chaque partie du kicker afin d'obtenir le meilleur angle de tir avec malgré tout une bonne solidité. Le choix ici a été de faire une première partie juste pour soulever la balle puis une large surface permettant de la taper vers le haut.



(a) L'évolution



(b) Kicker sur robot

FIGURE 3.17 – Différents prototypes de kicker (version finale en noir)

Le résultat fut très satisfaisant car nous avons fini 1<sup>er</sup> au challenge avec un tir à 20cm de hauteur (les seconds n'ont fait que 12cm)<sup>5</sup>. Cela nous a aussi aidé à performer sur le tir, la distance maximale a atteint les 7m. Également, nous avons réussi à tirer au dessus d'un autre robot qui était tombé.

---

5. Lien vidéo à l'entraînement : <https://photos.app.goo.gl/4doJ2UJKMBV3w6Bj7>

# Chapitre 4

## La RoboCup

La RoboCup a permis de mettre à l'épreuve le code sur lequel j'ai travaillé pendant le début de mon stage. Toutes les améliorations exposées précédemment ont été utilisées et ont fonctionnées correctement.

Le gardien a été très utile cette année, le niveau de nos adversaires ayant considérablement augmenté, il a joué un rôle bien plus important que l'année précédente.

L'amélioration de la localisation nous a vraiment aidé car cela nous a permis de vraiment comprendre le positionnement du robot surtout lorsque la vision était difficile.

Le Throw-In représente davantage un geste technique, non essentiel pour le match mais il reste un évènement marquant de la RoboCup 2019.

Enfin, le changement de kicker a fait ses preuves lors du challenge technique mais également nous a fourni un atout considérable lors des matchs. En effet, grâce au tir allant jusqu'à 7m, les robots pouvaient marquer depuis le milieu du terrain ou faire des dégagements puissants lorsque la balle était proche de nos cages. Cela nous a fourni un avantage considérable par rapport aux autres équipes.

Les éléments sur lesquels j'ai travaillé dépendaient peu de l'environnement du robot. Les tests à Bordeaux étaient satisfaisants et il y a eu besoin de peu d'ajustements lorsque nous sommes arrivés à Sydney.

Pendant la RoboCup, la majeure partie des contributions a servie à réadapter la vision. Je n'avais pas du tout travaillé sur la vision avant d'arriver à Sydney mais il était important que les robots puissent se repérer dans leur nouvel environnement.

Nous avons un réseau de neurones chargé de reconnaître les ballons et les poteaux des cages. Arrivés sur place, il a fallu réentraîné ce réseau. En premier lieu, nous avons enregistré des vidéos pour avoir des données à fournir

au réseau. J'ai donc été chargée de lancer les prises de *logs* où les 5 robots se déplaçaient sur le terrain en observant autour d'eux.

Ensuite, pour récupérer les données, nous avons créé un script de récupération des *logs* permettant de télécharger les données assez rapidement. J'ai amélioré ce script afin qu'il fonctionne sur plusieurs robots simultanément et j'ai ajouté une option rapide permettant de télécharger le dernier match ou la dernière prise de *log*. Également, comme nous avons fait beaucoup de prises, j'ai ajouté une fonctionnalité permettant de supprimer un ou plusieurs *logs* sur le robot.

Il a ensuite fallu tagger les images pour entraîner le réseau de neurones, c'était une tâche très répétitive mais nécessaire pour que le réseau de neurones fonctionne.

Étant la personne la moins expérimentée au sein de l'équipe, il était difficile pendant la compétition de contribuer directement dans le code. Au cours de la compétition, nous avons eu 13 matchs (et les challenges techniques), ce qui nous impose un rythme soutenu et un climat d'urgence permanent.

Les matchs nécessitent beaucoup de tâches manuelles pour la préparation : chargement des batteries, installation des caméras, s'assurer du bon fonctionnement des robots.

J'ai beaucoup aidé à la préparation du matériel afin que mes coéquipiers puissent se focaliser sur les problèmes qu'ils pouvaient plus facilement résoudre.

# Chapitre 5

## Bilan

J'ai choisi d'effectuer ce stage en vue d'une préparation au Master ASPIC (Autonomous System, Perception, Interaction & Control) et au stage de l'année prochaine. Je n'avais jamais fait de robotique auparavant, ce stage m'a énormément apporté et m'a fortement conforté dans mes choix d'orientation.

J'ai pu parfaire les connaissances en C++ que j'ai apprises lors du projet de programmation et améliorer mon utilisation des langages objets avec lesquels je n'avais pas particulièrement d'expériences. Je pense avoir acquis des bases solides et je pars confiante pour la rentrée prochaine.

C'était mon premier stage en informatique et j'ai pu enfin mettre à profit mes connaissances et me confronter à la réalité du développement.

J'ai obtenu ce stage suite à une candidature spontanée. N'ayant pas d'objectif central prédéfini, cela m'a permis de contribuer sur des sujets variés. J'ai été confronté à l'étendue du domaine de la robotique et aux difficultés inhérentes à ce domaine.

Une des grandes difficulté du code de Rhoban est qu'il y a beaucoup de choses à comprendre pour savoir comment fonctionne un robot. Aujourd'hui encore, il me manque certaines pièces du puzzle. Je n'ai pas pu travailler sur tout mais je pense avoir eu une vue d'ensemble satisfaisante.

Mon travail effectué pendant ce stage a fait ses preuves lors de la RoboCup, le résultat étant détaillé dans la chapitre 4 concernant la compétition. Le comportement du gardien, la création du Throw-In, l'amélioration de la localisation et le changement de kickers ont remplis l'objectif du stage : contribuer à l'amélioration des robots pour les rendre plus performant en match.

Après la RoboCup, j'ai anticipé mon travail pour l'année prochaine en commençant des travaux de recherche sur des sujets plus complexes de la robotique. Afin d'améliorer notre localisation, nous aimerais changer le filtre particulaire au profit d'un algorithme de SLAM (*simultaneous localization and mapping*) [Strasdat et al., 2012]. Le filtre particulaire ne récupère que les informations de l'état actuel du robot, le SLAM nous permettrait de prendre en compte l'historique des observations et donc de pouvoir mieux repérer des faux positifs. Nous n'avons pas à nous occuper de la partie *mapping* du SLAM car nous connaissons déjà le terrain ce qui devrait nous permettre d'avoir de meilleurs résultats. La difficulté ici est que l'algorithme et les bibliothèques existantes pour le SLAM sont complexes et difficiles à prendre en main [Dellaert, 2012].

Également, l'approche du robot vers le ballon [Hofer and Rouxel, 2017] nécessite une modification. La marche a été refaite cette année et l'approche nécessite donc une actualisation, elle a été un peu trop hésitante et certains pas sont inutiles. Il serait aussi préférable que le robot puisse tirer en mouvement directement depuis la marche plutôt que de marquer un temps d'arrêt avant chaque tir, ce qui nous ferait gagner un temps considérable avant le tir. Pour les Throw-In, on pourrait ajouter un mouvement du bassin pour changer la direction une fois que la balle est dans les mains du robot. Si nous voulons que le gardien fasse des dégagements à la main, il est intéressant d'ajouter cette fonctionnalité car on pourrait gagner du temps avant la prise de balle.

Travailler au sein de Rhoban permet de s'investir sur des sujets différents de l'informatique et c'est pour moi un des plus grand atout de ce stage. Comme nous avons toujours des idées dans des domaines différents, cela permet de varier le travail mais aussi d'avoir toujours quelque chose à améliorer.

# Bibliographie

- [Baltes et al., 2019b] Baltes, J., Gerndt, R., Paetzel, M., Sadeghnejad, S., Farazi, H., Hofer, L., and Sattler, M. (13 June 2019b). RoboCup Soccer Humanoid League : RoadMap. <http://www.robocuphumanoid.org/wp-content/uploads/RCHL-2019-Roadmap-Draft1.pdf>.
- [Baltes et al., 2019a] Baltes, J., Gerndt, R., Paetzel, M., Sadeghnejad, S., Farazi, H., Hofer, L., and Sattler, M. (31 May 2019a). RoboCup Soccer Humanoid League : Laws of The Game. <http://www.robocuphumanoid.org/wp-content/uploads/RCHL-2019-Rules-final.pdf>.
- [Burkhard et al., 2002] Burkhard, H., Duhaut, D., Fujita, M., Lima, P., Murphy, R., and Rojas, R. (07 August 2002). The road to RoboCup 2050. *IEEE Robotics & Automation Magazine*, 9(2) :31–38.
- [Dellaert, 2012] Dellaert, F. (2012). Factor Graphs and GTSAM : A Hands-on Introduction. Technical Report GT-RIM-CP&R-2012-002, Georgia Institute of Technology.
- [D.Gouaillier et al., 2009] D.Gouaillier, V.Hugel, P.Blazevic, C.Kilner, J.Monceaux, P.Lafourcade, B.Marnier, J.Serre, and B.Maisonnier (12-17 May 2009). Mechatronic design of NAO humanoid. In *2009 IEEE International Conference on Robotics and Automation*, pages 769–774.
- [Doucet and Johansen, 2009] Doucet, A. and Johansen, A. (2009). A tutorial on particle filtering and smoothing : Fifteen years later. *Handbook of Nonlinear Filtering*, 12.
- [Hofer and Rouxel, 2017] Hofer, L. and Rouxel, Q. (2017). An Operational Method Toward Efficient Walk Control Policies for Humanoid Robots. In *27th International Conference on Automated Planning and Scheduling*.
- [Hsu, 1999] Hsu, F.-H. (1999). IBM’s Deep Blue Chess grandmaster chips. *IEEE Micro*, 19(2) :70–81.
- [Kitano and Asada, 1999] Kitano, H. and Asada, M. (1999). The robocup humanoid challenge as the millennium challenge for advanced robotics. *Advanced Robotics*, 13 :723–736.

- [Ly et al., 2019] Ly, O., Allali, J., Gondry, L., Hofer, L., Laborde-Zubieta, P., N'Guyen, S., Pirrone, A., and Rouxel, Q. (2019). Rhoban Football Club - Robot Specification. <https://submission.robocuphumanoid.org/uploads//Rhoban-specs-5c05011864329.pdf>.
- [Rouxel et al., 2015] Rouxel, Q., Passault, G., Hofer, L., N'Guyen, S., and Ly, O. (2015). Rhoban Hardware and Software Open Source Contributions for RoboCup Humanoids. In *Proceedings of 10th Workshop on Humanoid Soccer Robots, 15th IEEE-RAS International Conference on Humanoid Robots*.
- [Stevenson, 1997] Stevenson, S. (27 July-1 Aug. 1997). Mars Pathfinder Rover-Lewis Research Center technology experiments program. In *IECEC-97 Proceedings of the Thirty-Second Intersociety Energy Conversion Engineering Conference*, pages 722–727, Honolulu, HI, USA, USA.
- [Strasdat et al., 2012] Strasdat, H., Montiel, J. M., and Davison, A. J. (2012). Visual SLAM : Why filter? *Image and Vision Computing*, 30(2) :65–77.

## Annexe A

### Présentation du robot

