SCOR : Sport Collectif pour Robots

Marwane Kalam-Alami, Jules Lallouette, Marc Mauzole, Kevin Tran

March 20, 2011

Abstract

Ce rapport décrit dans les grandes lignes le fonctionnement de l'application créée à l'occasion du projet SCOR. Il explique les méthodes que nous avons utilisées, présente l'architecture du programme ainsi que son utilisation. Nous avons aussi présenté les inconvénients des approches utilisées et les points à améliorer dans le futur.

Contents

1 Introduction

Le projet SCOR (Sport COllectif pour Robots) consiste en l'affrontement de deux équipes de deux robots Khepera dans un jeu de type Pong en équipe. Les concurrents doivent créer un programme qui commande les robots et ne perçoit l'environnement (le terrain) que par l'intermédiaire d'une caméra fixée au terrain. Ce projet fait appel à des compétences très différentes : l'analyse d'image pour la localisation des robots, l'intelligence artificielle pour la mise en place de stratégies de jeu et évidemment la robotique pour les modes de contrôle des robots.

Ce rapport présente le programme que nous avons élaboré, les problèmes que nous avons rencontré et les améliorations auxquelles nous avons pu penser mais que nous n'avons pas eu le temps d'implémenter.

2 Etude des travaux précédents

Les équipes précédentes ont réalisé des programmes "défrichant" une grande partie des fonctionnalités nécessaires au projet SCOR : localisation des robots et de la balle sur le terrain, envois d'ordres aux robots, stratégie de jeu rudimentaires. Nous avons souhaité améliorer les points qui nous paraissaient poser problème et reprendre le code qui fonctionnait correctement.

Ainsi, nous avons décidé de revoir l'architecture du programme de l'équipe gagnante de l'année précédente, de la rendre plus modulaire et multi-tâches, ces améliorations seront présentées dans la partie 3. La localisation des robots à partir des images de la webcam marchait plutôt bien mais était sensible aux conditions d'éclairage et à l'occlusion des robots par la barre mobile les connectant à l'ordinateur, nous avons décidé d'explorer des pistes de solutions possibles à ces problèmes (présentées en partie 4) ; nous avons cependant réutilisé le code de localisation de l'année précédente, adapté à notre architecture, celui-ci s'étant révélé plus fiable. La stratégie des robots a été améliorée, nous avons notamment pris en compte des prédiction de trajectoires avec rebonds (les stratégies sont présentées en partie 5). Enfin, nous avons revu le système de contrôle des robots en utilisant un asservissement avec régulateur PID que nous présentons en partie 6.

3 Architecture de l'application

L'aricteture globale de l'application est schématisée sur la figure 1 ; c'est une architecture en tableau noir, centrée sur les données [1]

[1] Cf. `http://en.wikipedia.org/wiki/Blackboard_system`
et `http://www.vico.org/pages/PatronsDisseny/Pattern%20Blackboard/`

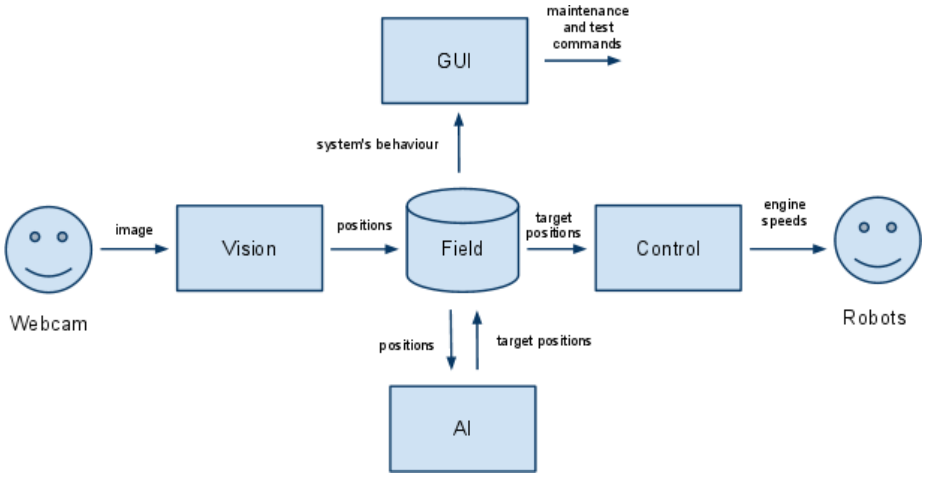

Figure 1: Architecure globale de l'application.

3.1 Librairies et utilisation

Nous avons utilisé, comme l'équipe de l'année précédente, la librairie de traitement d'image openCV. Le projet peut être compilé au moyen du fichier `Makefile` fourni ou en utilisant l'IDE Eclipse moyennant l'ajout des répertoires d'includes et des liens vers les librairies dynamiques contenus dans le fichier `Makefile`. Il est à noter que seulement un des deux ordinateurs de la salle SCOR permet de compiler notre programme pour des raisons que nous ne comprenons pas tout à fait mais qui semblent provenir d'une différence de version de la bibliothèque openCV, la compilation marche correctement avec la dernière version d'openCV sur d'autres postes.

Le programme peut être lancé en ligne de commande avec les options suivantes :

- `-winSettings` : Affiche une fenêtre de reglage d'options pour l'analyse d'images.
- `-cam x` : Demande au programme d'utiliser la webcam d'indice x.
- `-noSim` : Désactive la fenêtre de simulation.
- `-noStrat` : Désactive l'utilisation des stratégies.
- `-noVideo` : Désactive l'utilisation de la webcam.
- `-noCont1` : Désactive l'utilisation du contrôleur du robot 1.
- `-noCont2` : Désactive l'utilisation du contrôleur du robot 2.
- `-noGui` : Désactive l'utilisation de l'interface de contrôle manuel des robots.
- `-daemon` : Active le démon de la simulation (remplace le module d'analyse d'images par une simulation physique).
- `-dNotUs` : Désactive le contrôle des robots de notre équipe par le démon.

- `-dNotThem` : Désactive le contrôle des robots de l'équipe adverse par le démon.
- `-dNotBall` : Désactive le contrôle de la balle par le démon.
- `-dTFact x` : Fixe un facteur de vitesse de simulation du démon à x.
- `-dDefault` : Utilise les réglages du démon par défaut.

3.2 Gestion des threads

3.2.1 Architecture multi-tâche

Afin d'optimiser les performances de notre solution, nous avons cherché à mettre au point une gestion des threads efficace, permettant de paralléliser au maximum les tâches. Le choix technologique s'est assez rapidement porté vers la librairie POSIX Threads ou pthread, largement utilisée et adaptée à nos besoins.

La construction des tâches s'est faite en distinguant les processus parallélisables du système. Nous sommes arrivés à un total de 8 à 9 tâches :

- Le traitement de l'image, consistant simplement à extraire l'image du flux vidéo et la rendre exploitable au reste du programme (1 tâche)
- La recherche des éléments pertinents de l'image, à savoir la balle (1 tâche) et les deux Kheperas de notre équipe (1 tâche). A noter que l'architecture est prévue pour rendre facilement possible l'intégration future du traitement des Kheperas adverses.
- La partie de l'IA dédiée aux choix stratégiques de l'équipe (1 tâche)
- La partie de l'IA prenant en charge les objectifs simples de chaque robot (2 tâches)
- La partie de l'IA prenant en charge le déplacement réel de chaque robot (2 tâches)
- Enfin, un éventuel interface graphique (1 tâche)

Note : D'autres threads s'ajoutent encore dans le cadre de la simulation, que l'on ne traitera pas ici.

3.2.2 Création/destruction des tâches

Enfin, il convient de mentionner que l'opération de création et de destruction de ces threads a été placée dans une classe dédiée, appelée `ThreadManager`, dont l'interface est visible sur la figure 2.

3.3 Gestion des ressources

Étant donné que les tâches précédemment décrites doivent se partager les mêmes ressources, la question de l'accès concurrent se révèle primordiale à la fiabilité de l'architecture. Le problème est par ailleurs particulier, puisque les besoins des différentes tâches ne sont pas exactement les mêmes vis-à-vis des données : alors que la partie déplacement nécessite de connaître la position la plus

Figure 2: Classe de gestion des threads.

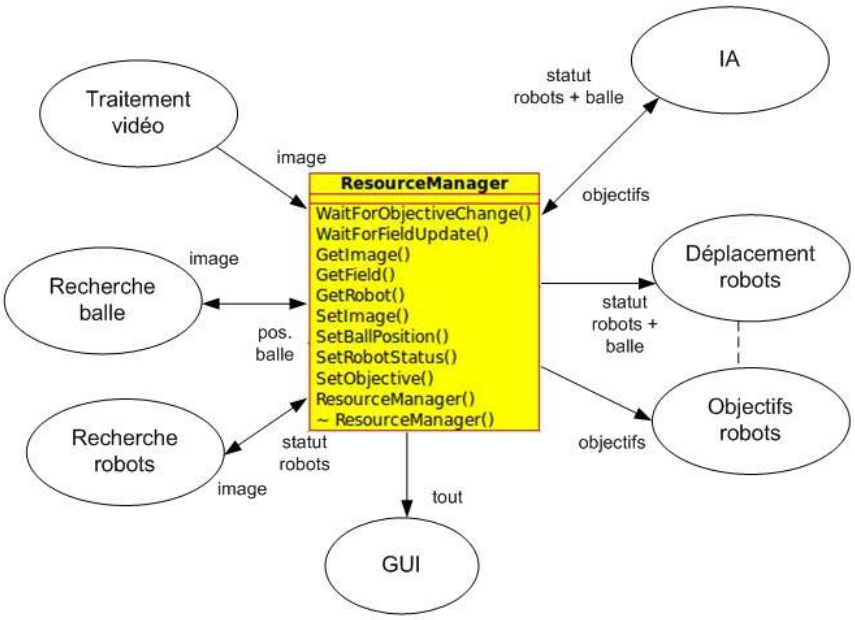

Figure 3: Schéma synthétique des relations entre les tâches et le modèle de données.

récente possible des robots, la partie stratégie à la contrainte supplémentaire de devoir accéder à l'ensemble des données du terrain à un instant t identique. Ces contraintes liées à la datation des données nous ont conduits à centraliser les ressources derrière une classe unique.

Ainsi, la classe `ResourceManager` a pour objectif de gérer ces contraintes. Elle constitue pour les différents tâches du programme le point de lecture et d'écriture des différentes données partagées, à savoir : l'image brute, les données stratégiques et la situation sur le terrain. Le schéma visible sur la figure 3 synthétise la relation de chaque tâche au modèle de données.

La résolution des problèmes de synchronisation s'observe dans le rôle des différents getters : à part `getField()`, qui garantit que les données transmises correspondent à une même date, les autres visent à donner les versions les plus récentes possibles des informations connues. Les différentes méthodes sont bien entendu thread-safe; on notera néanmoins que cela est facilité par le fait que chaque type de donnée n'est écrite que par une tâche (l'image par le traitement vidéo, etc.).

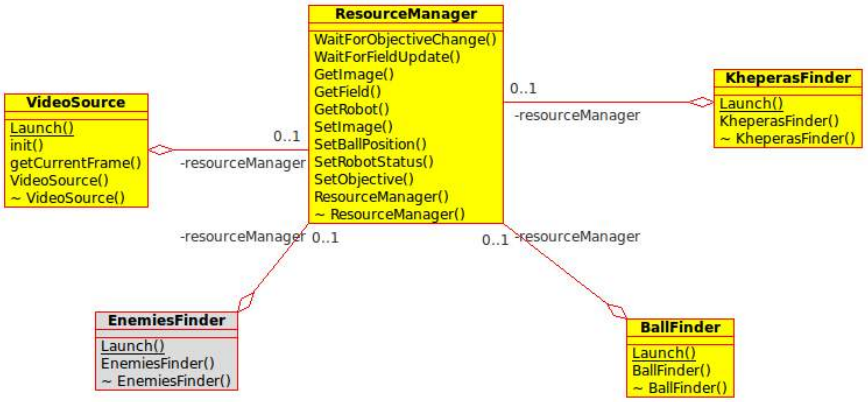

Figure 4: Diagramme de classes du module vision.

4 Module vision

4.1 Détection du contour du terrain

Afin de pouvoir identifier les limites du terrain, une méthode de détection automatique a été implémentée. On filtre d'abord la couleur du terrain puis on exécute l'algorithme de recherche de contour d'OpenCV `cvStartFindContours`. Les 4 coins du contour trouvé correspondent aux extrémités du terrain.

4.2 Détection des Khepera

Plusieurs techniques de détection ont été explorées, la détection de cercles avec la transformés de Hough, le tracking de point d'intérêt avec l'algorithme Optical flow et enfin la détection de couleur.

4.3 Transformée de Hough

L'algorithme de la transformé de Hough nous permet de retrouver des lignes et des cercles dans une image. En appliquant les motifs appropriés sur les Kheperas recouvert de certains motifs (cf. figure 5) on devient capable d'obtenir leur position et orientation. Malheureusement, du fait de la résolution trop basse des images, le résultat de la détection d'orientation n'était pas assez précis, nous avons donc abandonné cette technique.

Dans l'exemple du motif de gauche de la figure 5, le plus grand cercle, de la taille du Khepera, permet d'obtenir la position du robot, le petit cercle intérieur permet d'obtenir l'orientation.

Le motif de droite est variante utilisant la détection de ligne pour trouver un segment dans le cercle principal et ainsi obtenir l'orientation du robot.

4.4 Points d'intérets et Optical Flow

L'algorithme optical flow d'openCV permet de retrouver des points d'une image dans une autre. En suivant les points se trouvant sur un robot sur les images

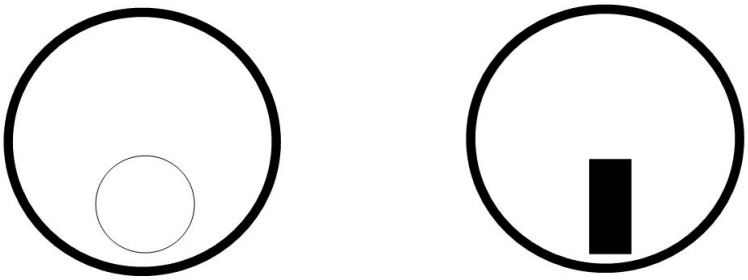

Figure 5: Motifs à appliquer sur les robots de notre équipe pour l'utilisation de la transformée de Hough.

successives il est possible de d'obtenir sa position et son orientation. Le défaut cette méthode est que l'on a peu de contrôle sur les points à tracker, si le robot est occulté par la perche, les points trackés deviennent ceux de la perche.

4.5 Clustering sur filtrage par couleur

Avec la résolution d'image dont on dispose la technique la plus adaptée que nous avons trouvé est le clustering de points sur des images filtrées par couleur. Cette technique est celle utilisée par une des équipe de 2009. Les robots de notre équipe sont recouverts du motif visible sur la figure 6. Voici l'explication de l'algorithme en partie extraite de leur rapport.

Principe:

1. On filtre une couleur ;
2. On recherche les points de concentration.

Sauf que pour les Kheperas on a deux nouvelles contraintes :

- On recherche deux objets de la même couleur sur le terrain ;
- On doit être capable d'identifier l'avant de l'arrière du Khepera.

Pour cela, chaque Khepera avait sur son dessus deux demi-cercles : un bleu et un blanc pour identifier l'avant. Ainsi la recherche des Kheperas s'effectuait comme suit :

1. Sur l'image issue de la webcam (après transformation), nous ne gardons que les pixels à majorité de bleu. Nous obtenons alors une nouvelle image monochrome ;
2. On effectue une érosion afin d'éliminer le bruit ;
3. On recherche ensuite deux clusters (nous ne prenions pas en compte les kheperas de l'adversaire) selon un calcul de clustering classique ;
4. Sur l'image issue de la webcam (après transformation), nous ne gardons que les pixels à majorité de blanc. Nous obtenons alors une nouvelle image monochrome ;
5. Dans un carrée suffisamment grand et centré sur chaque cluster identifié précédemment, on recherche une concentration de points blancs : cela nous donne alors l'avant du robot.

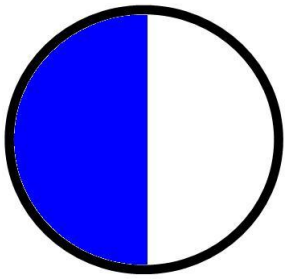

Figure 6: Motif plaqué sur les robots de notre équipe.

4.6 Détection de la balle

La détection de la balle est elle aussi emprunté à une équipe de 2009. On filtre la couleur rouge et on recherche le barycentre des points obtenus.

5 Module simulation et stratégies

Le module simulation et stratégies utilise les informations récupérées par le module vision et utilise une version simulée, idéalisée, de l'environnement permettant de formuler des prédictions quant à l'état futur de l'environnement. Cet aspect prévisionnel est géré par la classe `Simulator` du diagramme de classe visible en figure 7 ; elle permet notamment de prévoir la trajectoire de la balle. Ces informations sont utilisées par les classes héritant de la classe `RobotStrategy` afin de "juger" de la situation de jeu et de donner aux robots les ordres qui conviennent.

5.1 Simulation

La partie simulation du module simulation et stratégies permet, en plus de son rôle prédictif, de simuler complètement un environnement en remplaçant l'acquisition vidéo et la détection des robots et de la balle par une simulation physique de ceux-ci (Cf. image de gauche de la figure 8). La simulation est assez simple, elle permet de rendre compte des rebonds de la balle, des accélérations des robots et de la balle. Cependant, les robots sont simulés sans orientations, on considère qu'ils s'orientent instantanément dans la direction de leur objectif ; il ne faut donc pas considérer la simulation comme une reproduction fidèle de la réalité mais plutôt comme une façon de tester des stratégies en développement dans des conditions plus contrôlées que les conditions réelles. On pourra ainsi spécifier une certaine configuration de jeu et vérifier que la stratégie donne les ordres attendus aux robots. Lorsque le développement d'une stratégie est considéré comme étant bientôt fini, la simulation pourra servir à tester deux stratégies différentes en les faisant jouer l'une contre l'autre. Il convient cependant de garder à l'esprit que le résultat de telles simulations ne reflètera pas nécessairement la qualité d'une stratégie en condition réelle.

Les simulations ne permettent pour l'instant d'obtenir qu'une appréciation qualitative d'une stratégie, en observant son comportement ; un système plus élaboré de comptage de points et d'implémentation des règles du jeu dans le

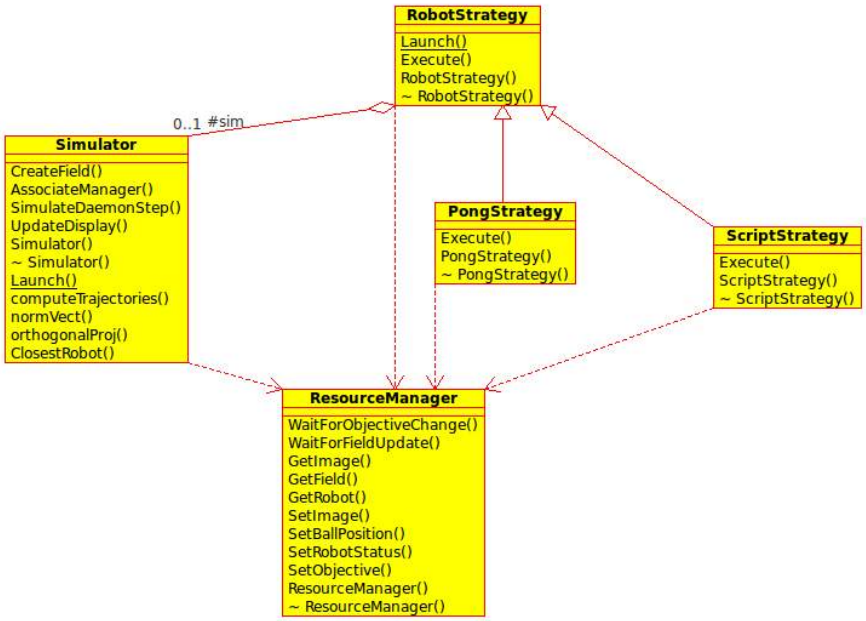

Figure 7: Diagramme de classes du module simulation et stratégies

simulateur pourraient permettre d'obtenir une évaluation quantitative d'une stratégie par rapport à une autre en simulant un nombre de matchs assez important. De plus, dans le cas de stratégies déployant des mécanismes d'apprentissage, ces simulations pourraient permettre de gagner beaucoup de temps comparé à des matchs réels.

La simulation physique est rudimentaire, elle possède encore quelques bugs et si le simulateur devait servir à tester des stratégies de façon automatique, il serait peut être préférable, afin de ne pas fausser les résultats, d'utiliser une bibliothèque de simulation physique tierce.

5.2 Prévision de trajectoires

Comme on peut le voir sur la capture d'écran de droite de la figure8, la trajectoire de la balle en mouvement est prédite par le module de simulation. On peut prévoir la trajectoire de n'importe quel objet dont on possède la position et la vitesse grâce à la méthode `computeTrajectories(...)` de la classe `Simulator` ; cette méthode renvoie une suite de points qui sont les points de rebonds prévus.

En l'état actuel du code, une seule trajectoire est prévue, le résultat de cette prédiction est donc assez sensible aux incertitudes sur la position et la vitesse de l'objet ; dans le cas de la simulation cela n'a aucune influence, mais en conditions réelles, ces incertitudes peuvent être assez élevées. Afin d'améliorer la prédiction de trajectoire, il serait bon de pouvoir quantifier cette incertitude (en spécifiant par exemple une marge d'erreur en pixels sur la position de l'objet et en déduisant les marges d'erreurs sur la position et la vitesse en mètres) puis de simuler plusieurs trajectoires possibles. On peut ainsi prévoir, pour un

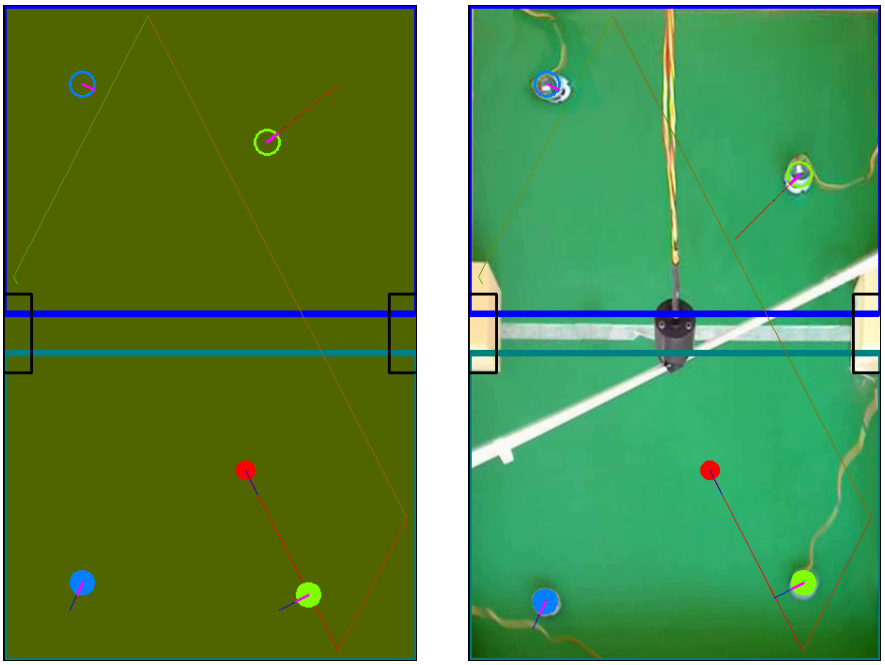

Figure 8: Capture d'écran de la simulation du terrain. Lorsque la webcam est connectée, le fond uni de l'image de gauche est remplacé par l'image du terrain acquise par la webcam (à droite).

temps $t_p = t + \Delta t$ la position de l'objet en calculant le barycentre des positions prévues par les différentes simulations de trajectoire. Afin d'améliorer encore plus la prédiction de trajectoire, on pourrait prendre en compte l'accélération de l'objet, ce qui permettrait prédire des trajectoires non rectilignes.

5.3 Stratégies

Les stratégies sont globalement constitués d'une méthode appelée régulièrement par le programme ; c'est à l'intérieur de cette méthode que la situation de jeu est analysée et que la stratégie décide des ordres à donner à chaque robot de l'équipe. Un ordre est uniquement constitué d'un point sur le terrain vers lequel la stratégie souhaite que le robot se déplace, le déplacement à proprement parler est géré par le module contrôle (cf. partie 6). Pour pouvoir prendre ces décisions, les stratégies ont accès ont informations détenues par la classe `RessourceManager` ainsi qu'à la partie simulation présentée précédemment.

5.3.1 Stratégies implémentées

Nous avons implémenté deux stratégies ; la première, `PongStrategy` est extrêmement simple et consiste à garder les deux robots en retrait et à intercepter la balle lorsqu'elle arrive dans son camp ou lorsqu'elle est immobile en donnant l'ordre au robot le plus proche de se déplacer vers la balle.

La deuxième stratégie, `ScriptStrategy`, est un peu plus complexe et prend en compte les prédictions de trajectoires de la balle. Lorsque la balle se dirige

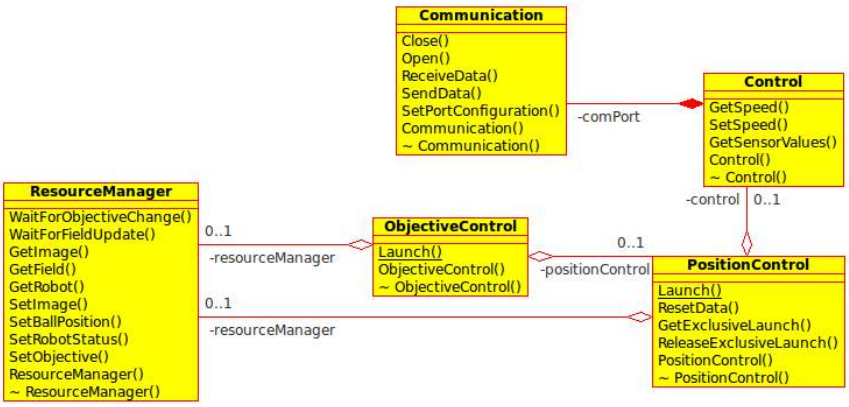

Figure 9: Diagramme de classes du module de contrôle des robots.

vers son camp, la stratégie fait appel à la prédiction de sa trajectoire par le simulateur et déplace le robot le plus proche vers un point se situant sur la trajectoire prédite et pouvant être atteint avant que la balle ne l'atteigne. Ainsi, les robots peuvent intercepter la balle ; des que la balle est assez proche d'un robot ou si jamais la balle est immobile, le robot le plus proche se dirige vers elle pour la renvoyer dans le camp des adversaires. Il serait intéressant d'ajouter une partie de choix d'angle de tir en fonction de la position des adversaires.

5.3.2 Ajouter une nouvelle stratégie

Pour ajouter une nouvelle stratégie, il suffit de faire hériter la nouvelle classe de la classe `RobotStrategy`, de surcharger la méthode `Execute()` et de prendre exemple sur les deux stratégies implémentées pour savoir comment obtenir la prédiction des trajectoires ou comment donner un ordre aux robots.

5.4 Améliorations

Étant donné que notre programme met en place un environnement de gestion des positions des objets et de contrôle des robots assez robuste, nous pensons qu'il serait intéressant que les futurs participants se concentrent sur la création d'une stratégie plus évoluée que les deux que nous avons implémenté en utilisant par exemple des techniques de machine learning faisant correspondre des situations de jeu plus ou moins discrétisées (état de la balle : dans notre camp / dans l'autre ; se déplaçant vers telle partie du terrain, etc.) et des objectifs types (tirer dans la balle, se replacer, faire une passe, etc.) de façon à développer des stratégies originales et non scriptées. Ainsi, contrairement à des tentatives précédentes d'utilisation de réseaux de neurones pour contrôler directement les robots (en contrôlant la vitesse de ses roues), nous proposons d'utiliser des méthodes semblables mais pour donner des objectifs aux robots dont la réalisation sera laissée au module contrôle qui est largement plus fiable. Ce changement de mode d'utilisation d'un réseau de neurones (ou d'autres techniques de machine learning nécessitant un apprentissage assez long) permet de rapprocher les

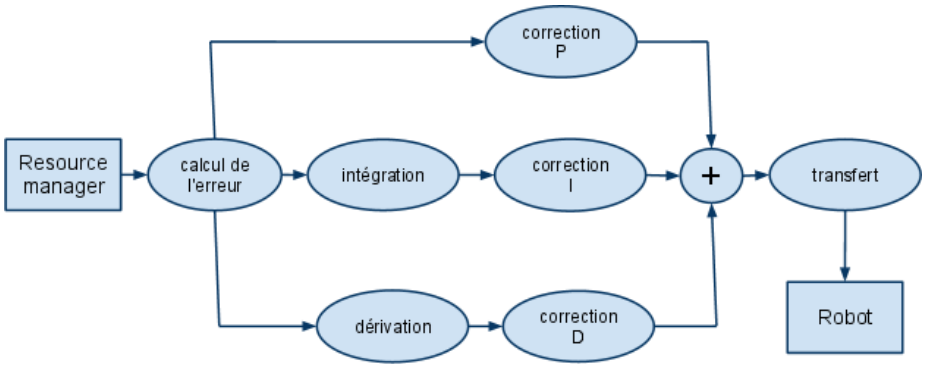

Figure 10: Représentation schématique de l'asservissement PID.

conditions de simulation et les conditions réelles lors de l'apprentissage, point critique responsable de l'échec de l'approche des années précédentes. Nous ne sommes pas certains de la validité de l'approche ni de son potentiel de succès mais il nous parait intéressant d'explorer cette voie.

6 Module contrôle

Le principal problème auquel doit répondre ce module est la gestion de la trajectoire du robot. En effet, à cause de l'environnement (surface rugueuse et peut-être penchée, traction du câble, perche, ...), la trajectoire du robot peut être fortement déviée à forte vitesse. En employant une vitesse modérée, la trajectoire obtenue est meilleure, mais la lenteur résultante du robot fait qu'elle ne peut frapper la balle à une vitesse suffisante pour atteindre le but adverse. Par rapport au code des gagnants de l'année passée, il faut donc rajouter un contrôleur de trajectoire pour corriger les écarts possibles.

6.1 Principe du contrôle de la trajectoire

En automatisme on parle d'asservissement en position du robot. Le schéma visible sur la figure 10 en présente le fonctionnement ; des explications plus détaillées sont accessibles sur le site du club'elek[2].

L'idée est de, périodiquement, calculer l'erreur que l'on observe par rapport à l'objectif que l'on s'est fixé, puis de corriger cette erreur en compensant sur la vitesse sur les roues. On cherche à obtenir la vitesse maximale pour le robot, une solution simple est de l'erreur calculée par une constante afin d'obtenir une valeur qui correspondra à la différence de vitesse que l'on appliquera sur les roues. Cette première approximation est appelée "asservissement P" pour indiquer que la correction est proportionnelle à l'erreur.

Cependant cette première approximation ne prend pas en compte le temps. En effet, si on a une force appliquée au robot qui compense la différence de vitesse appliquée au robot, l'erreur devient constante et donc non compensée.

[2] http://clubelek.insa-lyon.fr/joomla/fr/base_de_connaissances/informatique/asservissement_et_pilotage_de_robot_autonome_introduc_5.php

De plus, Si l'erreur augmente avec le temps, on souhaiterait que la correction appliquée s'adapte en compensant plus fortement.

Le calcul de l'intégrale et de la dérivée de l'évolution de l'erreur avec le temps permet de prendre en compte ces facteurs. Calculer l'intégrale revient à sommer les erreurs depuis le début de l'asservissement, tandis que la dérivée et la différence d'erreur entre deux itérations. On multiplie alors les deux termes obtenus par les constantes de correction respectives, puis on somme les trois valeurs afin d'obtenir notre différence de vitesse.

6.2 Implémentation

Le module comporte quatre threads, deux pour chaque robot. Le diagramme de classe correspondant au module est visible sur la figure 11. Un des threads, ObjectiveControl, n'est là que par commodité. Il s'occupe de réinitialiser les variables de calculs de la dérivée de l'intégrale notamment, à chaque changement d'objectif. "PositionControl" est le thread qui récupère les valeurs de position et d'orientation donnés par le module de vision, applique l'asservissement PID et donne les consignes de vitesse aux robots.

- on considère que le robot a atteint son objectif si la distance entre entre sa position et l'objectif est en-dessous d'un certain seuil (donné en mètres).
- une première étape consiste à voir comment est orienté le robot par rapport à son objectif, pour voir s'il n'est pas plus judicieux de faire reculer le robot plutôt que de le faire avancer.
- la vitesse n'est appliquée que si elle est différente de la valeur précédente.
- réglage des valeurs : on est partis des rapports de grandeurs des coefficients utilisés dans l'exemple du club'elek, rapportés à des ordres de grandeurs souhaités, puis on a ajusté les valeurs en fonction du comportement du robot[3] :
  - la constante de la proportionnelle joue sur la vitesse à laquelle le robot retrouve sa trajectoire, mais provoque des oscillations.
  - la constante de l'intégrale agit sur le temps de montée et l'erreur constante, mais provoque des oscillations
  - la constante de la dérivée agit sur les oscillations, mais peut rendre l'asservissement instable
- il existe une interface qui permet de donner manuellement des objectifs aux robots, et rendre plus commode le réglage des paramètres. Il suffit de faire un clic-gauche ou un clic-droit sur le terrain pour donner une consigne au robot associé. Pour le lancer, il faut lancer l'interface associée, et désactiver le module IA (voir la partie "gestion des threads" pour plus de détails).
- beaucoup d'implémentations d'asservissement en position ou en vitesse n'utilisent qu'un asservissement PI, voire un asservissement P. Pour obtenir un asservissement PI, il suffit de mettre la constante de correction de la dérivée à 0 !

[3]Source : http://en.wikipedia.org/wiki/PID_controller

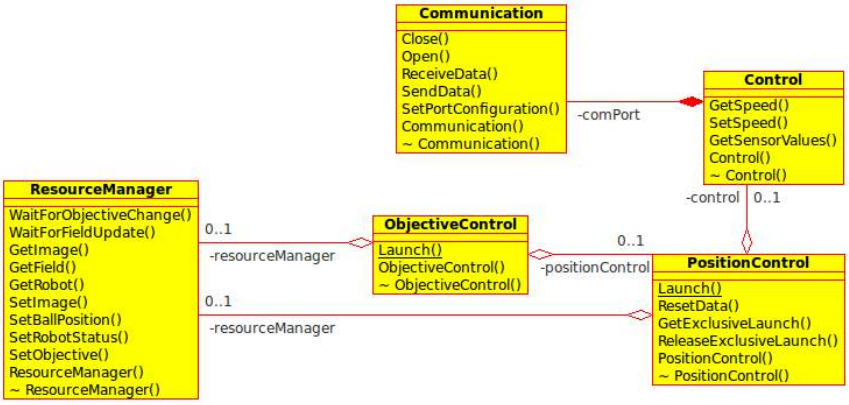

Figure 11: Diagramme de classes du module de contrôle des robots.

- de même, dans la littérature, les constantes de correction pour la dérivée et l'intégrale sont souvent exprimés en 1/K plutôt qu'en K. C'est à prendre en compte lors du réglage par petits incréments, ou de l'application de méthodes de recherche empirique des valeurs.

6.3 Conclusion

A cause de problèmes matériels, nous n'avons pu configurer correctement les paramètres de correction. Le robot atteint bien et relativement sa trajectoire, mais plus son orientation de départ diffère de son objectif, plus le robot ralentit et oscille au moment d'atteindre son but. Une meilleure configuration de ces constantes, et une prise en compte de la distance à l'objectif dans le calcul de l'erreur permettrait d'enlever ces oscillations. Cependant, le résultat nous paraît très satisfaisant puisque le robot tiens sa trajectoire à vitesse maximale, même dans des positions extrêmes (les deux robots côte à côte).

6.4 Possibilités d'amélioration

- La vitesse maximale mesurée du robot est de 60 en roue libre, et de 40 sur une surface. Or la documentation spécifie une vitesse maximale de 127. Il existe peut-être une configuration à faire pour atteindre la vitesse maximale.
- Vérifier le comportement du robot en bordure, qu'il ne se bloque pas pendant son calcul.
- Ajouter une commande pour orienter le robot, afin d'optimiser son futur déplacement.
- Faire une commande spéciale "tirer" semblable à la commande actuelle, mais dans laquelle le robot prend la vitesse maximale lorsqu'il est suffisamment proche de la balle (selon la caméra ou les capteurs) pour pouvoir frapper avec la vitesse maximale.

- Voir les paramètres utilisables pour le calcul de l'erreur. Pour l'instant, seul l'orientation est pris en compte. Les possibilités sont :
  - la distance à la cible (pourrait corriger les oscillations à l'approche de l'objectif)
  - la vitesse maximale (rend le paramétrage indépendant de la vitesse maximale. Ces deux facteurs semblent proportionnels)
  - la vitesse actuelle
  - la différence entre la vitesse ordonnée et celle mesurée par le robot (disponible par une méthode de la classe Contrôleur)
- Trouver les paramètres optimaux pour les constantes de correction. Il peut être intéressant de chercher sur Internet des méthodes empiriques pour approximer ces valeurs. Il est à noter qu'un asservissement PI, voire P, suffise pour ce projet ; et qu'un meilleur calcul de la dérivée et de l'intégrale en prenant en compte le temps pourrait donner des résultats plus précis.
- On peut imaginer un asservissement en vitesse : sachant que l'asservissement en position ne peut se faire qu'à la vitesse de rafraichissement de la webcam, y ajouter un asservissement en vitesse pourrait faire gagner en précision et efficacité. L'asservissement en position nous donnerait une vitesse cible à atteindre. A partir de la vitesse mesurée sur le robot, on peut réaliser un asservissement en vitesse à boucle fermé (il existe beaucoup de littérature sur ce sujet sur Internet), à une fréquence bien plus grande. Cependant, cela serait, selon la fréquence, très consommatrice en ressource processeur pour un résultat marginalement meilleur, voire inefficace.