Projet Scor

Florian Boucher, Florent Lavy, Mathieu Perez, Do Quoc Cuong

Rappel du projet

Règles du jeu

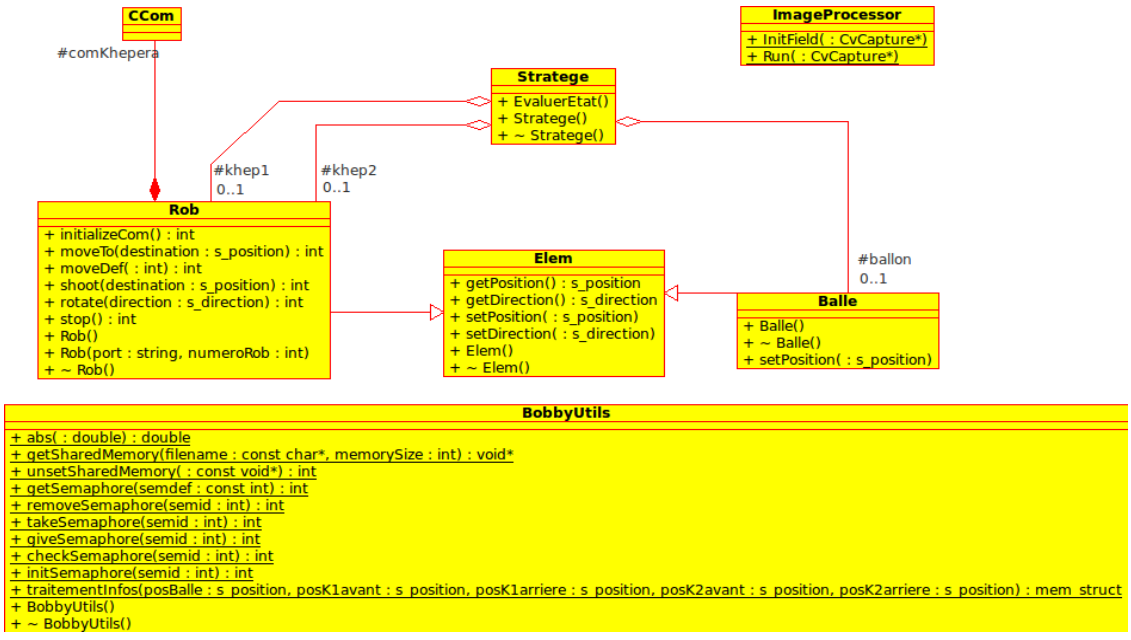
1. Deux équipes s'affrontent, et sont constituées de deux robots chacune.
2. Le vainqueur est celui qui aura marqué le plus de points en 5mn de jeu.
3. Un but est marqué quand la balle touche le fond des cages de l'adversaire. Un but vaut 3 points.
4. Une faute est réalisée quand la balle reste plus de 30s dans son camp. Une faute vaut 1 point, donné à l'adversaire.
5. Quand la balle est à l'arrêt sur un côté, les coachs peuvent remettre la balle en jeu où ils le souhaitent dans leur zone.

Le projet

Le projet SCOR existe depuis 3 ans.
Il fait intervenir des notions d'IA, d'analyse d'image, de stratégie et de communication.
D'années en années, le projet évolue et s'améliore.
Cette année, nous avons réutilisé l'analyse d'image de l'année précédente et nous nous sommes efforcés à obtenir un résultat le plus fonctionnel possible.
Au final, nos robots se déplacent, réceptionnent la balle et la renvoient dans le terrain adverse.
Cette année le problème le plus important a été les robots. En effet, ils se déplaçaient doucement et en zigzag (cf. Insalien sortant de la KFet). Ce problème est peut être du au câblage (trop long donc beaucoup de perte) et donc une perte de puissance fournie au robot.

Le diagramme des classe

Voici le diagramme des classes de notre projet.



Gestion de Projet

L'équipe est composée de 4 personnes. Do Quoc Cuong, Florent Lavy, Floriant Boucher et Mathieu Perez (chef de projet).
Au début nous avions pensé à chacun s'assigner à un aspect du problème:

- Acquisition Vidéo (Do Quoc Cuong)
- IA (Mathieu Perez / Floriant Boucher)
- Déplacement (Florent Lavy)
- Communication

Il s'est avéré qu'en fonction de l'emploi du temps, de la répartition de chacun dans des groupes différents et de la période de séminaires, il était quasiment impossible d'avancer correctement à 4. De plus le fait qu'il n'y ai que 2 postes réduisait l'efficacité du regroupement à 4.
Aussi 2 semaines avant le rendu nous n'avions réussi qu'à faire bouger un robot, et encore...
A partir de ce moment nous avons changé de stratégie. Nous nous sommes fixés des étapes phares:

- Mobilité du robot amélioré
- IA fonctionnelle
- liaison de l'IA a la vidéo
- liaison de le L'IA au mouvement
- amélioration du tout

Plutôt que d'attendre que tout le monde soit disponible, nous avons mis des heures de travail et ceux qui pouvaient/voulaient venir venaient. (En général de 18h à 22h).
L'avancée à été fulgurante, nous étions en général 2 par périodes et l'efficacité était grandement améliorée.

Voici donc quelques conseils pour les années futures:

- Demander le plus tôt possible à M Cortez le badge pour pouvoir venir le soir
- Ne pas chercher à réunir 4 personnes pour faire une séance
- Utiliser la méthode Scrum (réunion rapides en début de séance avec ceux qui sont présent et poser des objectifs à cours termes)
- Limiter le travail aux heures fixées
- Faire des comptes rendus à chaque séances afin d'avoir une bonne communication de l'avancée
- Une équipe soudée et une bonne ambiance pour passer de bonnes soirée au département

Déplacement

Le déplacement du robot vers une destination donnée est réalisé en deux étapes. L'une consiste à orienter le robot dans la direction de la cible, du moins grossièrement, afin de faciliter la réalisation de la deuxième étape. Cette dernière constitue le déplacement à proprement parler, le robot est guidé et la trajectoire est ajustée pour atteindre la cible.

Orientation

Lors de l'orientation du robot, celui-ci est immobile et il est donc impossible d'exploiter sont déplacement pour connaître sont orientation. Des repères bicolores ont été placés sur les robots afin de calculer grossièrement leurs directions. Ces repères sont de simples cercles dont l'un des hémi cercles est bleu et l'autre blanc mais ils sont suffisants pour avoir une première idée de l'orientation grâce à la webcam.

On réalisé alors une rotation en affectant aux roues des vitesses de valeurs absolues communes mais de signes opposés. Les signes sont déterminés de manière à ne pas faire plus d'un demi tour pour l'orientation et ainsi ne pas perde trop de temps sur cette étape. On calcule pour cela la norme du produit vectoriel entre la direction actuelle du robot et la direction qu'il doit adopter pour être face à sa cible. Le signe de ce produit permet de trouver le sens de rotation et donc les signes des vitesses. Pendant la rotation et aussi souvent que possible (idéalement à chaque nouvelle image fournie par la webcam), le produit scalaire normalisé entre la direction du robot et la direction à adopter est calculé. Ce produit correspond donc au cosinus de l'angle entre les deux directions, dès qu'il franchit un certain seuil modifiable, la rotation est arrêté et le robot est alors dans une direction satisfaisante pour lancer la deuxième étape.

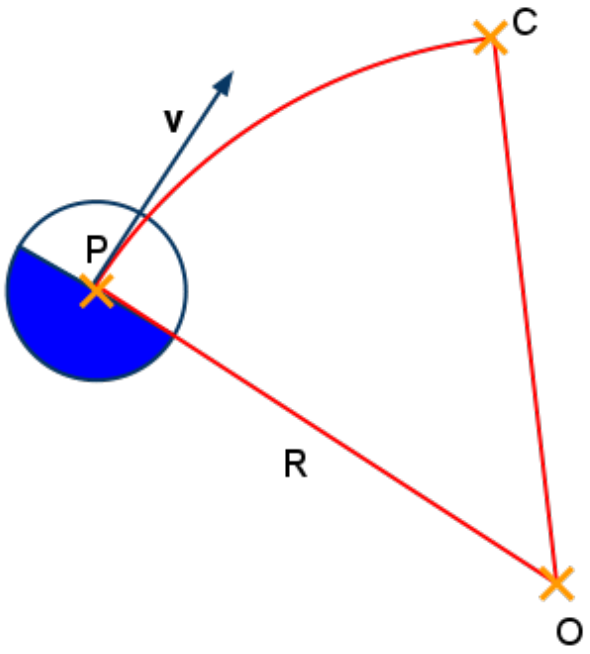
Les calculs sont effectués de manière à éviter l'emploi des fonctions trigonométriques gourmandes en capacités de calcul. Il reste bien sûr l'inévitable racine carrée calculée pour normaliser le produit scalaire mais il semble difficile de s'en passer.

Déplacement

Le robot est désormais correctement orienté. On n'utilise alors plus le témoin blanc et bleu pour déterminer l'orientation du robot, cette méthode est en effet assez efficace lorsque celui-ci est immobile comme lors de l'étape précédente mais devient rapidement insuffisante lorsque la vitesse est trop importante. Il est préférable de profiter du déplacement pour déterminer précisément la direction réelle du Khépéra. On commence donc par le faire avancer à sa vitesse de déplacement nominale. On calcule alors sa direction simplement en soustrayant la position occupée précédemment (sur l'image précédente) à celle occupée actuellement.

Nous disposons alors de trois informations, la position actuelle du robot, sa direction de déplacement qui n'est autre que la direction de son vecteur vitesse ainsi que la position de la destination. Nous pouvons donc facilement conclure si la trajectoire courante est correcte pour atteindre la cible. On calcule alors le produit scalaire normalisé entre la direction actuelle du Khépéra et la direction qu'il faudrait suivre idéalement pour atteindre l'objectif. Si ce produit scalaire est suffisamment grand (supérieur à un seuil fixé), alors l'angle entre les deux direction est assez faible et la trajectoire est conservée. Dans le cas contraire un calcul est mené pour trouver le rayon de l'arc de cercle qui passe par la position actuelle du Khépéra ainsi que par la destination et qui est tangent à la direction actuelle du robot. Ce rayon est le rayon de courbure à adopter. On peut alors calculer les vitesses à affecter aux deux roues pour que cet arc de cercle soit la trajectoire effective de l'appareil. Il est en effet

simple (en théorie) de réaliser un arc de cercle avec un Khépéra car c'est la trajectoire qu'il adopte naturellement lorsque les vitesses des deux roues ne sont pas égales.



Si C est la destination, le calcul de R se fait comme suit :
Soit M le milieu de [PC]. PMO est rectangle en M. ||PM|| = ||PC||/2 est connu. L'angle O dans le triangle PMO est connu, on peut donc calculer R directement. On détermine le vecteur U de norme ||PM|| et perpendiculaire à PC. On détermine ensuite le vecteur W, unitaire et perpendiculaire à v.

$$R = \frac{\|\vec{U} \wedge \vec{U}\|}{\|\vec{U} \wedge \vec{W}\|}$$

On obtient alors :

Ce rayon est déterminé en pixels sur l'image de la webcam. La distance entre les roues du Khépéra est d'environ 6 cm soit 30 pixels. La vitesse de la roue extérieure est toujours fixée à la vitesse nominale enregistrée dans une constante. On joue donc sur la vitesse de la roue intérieure pour réaliser la courbe. Si V est la vitesse de la roue extérieure, la vitesse de la roue intérieure est donnée par $V_{intérieure} = V ((R-l/2) / (R+1/2))$

En théorie, cette méthode permet d'atteindre directement la cible. En pratique, le rayon de courbure est recalculé tout au long du déplacement pour être plus précis. Si la direction du robot balaye la position de la cible, le robot arrête de tourner pour poursuivre tout droit et atteindre la cible.

Le robot s'arrête lorsqu'il se trouve dans une zone resserrée autour de la cible. Cette zone est un carré et la tolérance peut être modifiée.

Communication

Dans le projet, nous avons distingué 2 types de communication :
- La communication interprocessus : Les échanges de messages et d'information ainsi que la synchronisation entre les différents processus de l'application.
- La communication entre l'application et les robots : L'envoi des commandes depuis l'application ainsi que la réception de données provenant des robots.

Communication interprocessus

Elle consiste en l'utilisation de dispositifs multitâches tels que les sémaphores et les mémoires partagées. Les sémaphores nous ont permis la synchronisation entre les processus ainsi que la protection des données. La mémoire partagée a été préféré pour le partage d'informations sur l'état du système aux boîtes aux lettres car :
- l'information devait être disponible à tous les processus
- L'information devait encore être présente lorsqu'un processus lisait la mémoire

La mémoire partagée contenait :
- La position et la direction des 2 robots
- La position de la balle

Ainsi chaque processus pouvait lire l'état des robots et de la balle à tout moment (tant qu'un autre processus n'accéder pas à cette mémoire protégée par un sémaphore). Seul le processus chargé de la capture et de l'exploitation vidéo est autorisé à écrire dans la mémoire.

Afin de faciliter tous les appels systèmes, nous avons utilisé une classe avec des méthodes statiques. Il s'agit de la classe "BobbyUtils".

Communication avec les robots

La communication avec un robot s'effectue à l'aide d'une librairie fournie avec les robots. L'utilisation de ses fonctions d'assez bas niveau demande de créer des chaînes de caractères qui sont ensuite envoyés par l'intermédiaire de la connexion USB au robot. Les méthodes chargées de l'écriture et la lecture sur le dispositif USB ayant déjà été créées par un groupe des années précédentes, nous les réutilisons. Seulement, afin de faciliter tous les appels (ne pas avoir à taper les chaînes de caractères avant chaque appel), on a extrait toutes les méthodes de communication avec le robot dans une classe à part que l'on a nommé "CCom". Les méthodes créent elle-même les chaînes de caractères à partir ses paramètres (Plus besoin de se demander si la lettre D correspond à la commande pour établir la vitesse des roues du robot, il suffit d'appeler la méthode SetSpeed qui s'en chargera).

Par exemple : Pour envoyer la commande (chaîne de caractères) "D,20,20", il suffit appeler la méthode SetSpeed(20,20).

La classe correspondant aux robots (classe "Rob") appelle donc les méthodes de la classe "CCom" pour envoyer les ordres au robot après initialisation de la communication (qui est aussi une méthode de la classe "CCom").

Améliorations possibles

L'ensemble de ces méthodes sont bas niveau, on a juste rajouté une légère sur-couche afin de faciliter les appels et donc la lisibilité du code (entraînant une certaine aisance lors de modification de code et de correction). Cependant, il y a certainement des améliorations possibles à effectuer dans ses segments de codes mais celles-ci seraient d'ordre mineur par rapport aux autres modules que sont l'acquisition video et les mouvements du robot.
On peut imaginer un très léger gain de performance en optimisant le code.

IA

Pour ce projet nous avons mis en place une IA très simpliste afin de se concentrer sur les problèmes de déplacement.
L'IA est composé de 3 stratégies. La stratégie est définie par une analyse du déplacement de la balle et sa position (l'état du jeu).

Le moteur de l'IA

L'IA analyse l'état du jeu et adopte alors une stratégie. La stratégie est une tâche fille qui est créé à la fin de l'évaluation de l'état. Si la stratégie courante n'est plus d'actualité, on relâche un jeton pour tuer la tâche fille. La tâche fille prendra alors le jeton et se terminera. On pourra alors lancer une nouvelle stratégie.

Les 3 stratégies

Attaque

Si la balle est dans notre terrain et si elle est immobile ou avance vers le terrain adverse, on considère l'état comme Attaque. A ce moment là, nous plaçons le robot le plus proche derrière la balle et nous lui faisons faire un Boost (on met la vitesse des roues au maximum pendant un cours instant afin de renvoyer la balle dans le terrain adverse).

Défense

Si la balle se dirige vers notre terrain, nous nous plaçons dans une position de défense. Un calcul extrêmement simple permet de définir le point d'intersection de la trajectoire de la balle et de la ligne de défense. On place un des robots (en fonction de leur zone de terrain alloué) à ce point.

Replacement

Si la balle se dirige vers le terrain adverse et est située dans le terrain adverse, nous replaçons les robots. Chaque robot se dirige alors sur la ligne de défense et au milieu de sa zone.

Amélioration possibles

Pour le moment, le déplacement est bloquant donc tant que le robot a un mouvement la tâche fille ne récupérera pas le jeton. Une amélioration serait de faire passer le mouvement en une tâche "killable".
Ensuite si les problèmes de déplacement sont réglés, une IA plus efficace peut être mise en place:

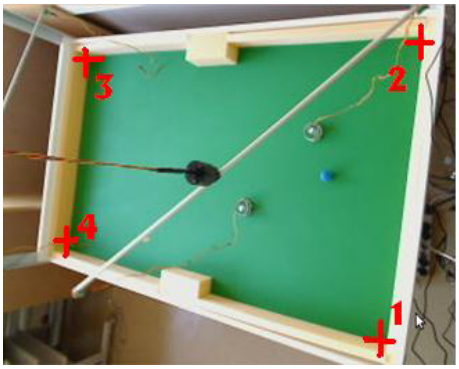
- Attaque prenant en compte les positions des robots adverses
- Replacement constant en fonction des robots adverse
- Ajout d'une stratégie d'interception

L'IA mise en place reste très simpliste mais fonctionne. Les axes d'améliorations sont très nombreux.

Traitement des images

Principe général

On utilise un webcam pour capturer l'image du terrain (fonction ImageProcessor::InitField()). Ensuite à partir de l'image obtenue, il faut extraire la zone intéressant en définissant 4 coins dans l'image.
Pour cela, il faut cliquer dans la fenêtre aux quatre coins du terrain en commençant par le coin inférieur droit puis en tournant dans le sens antihoraire :



L'image de la zone extraite est un parallélogramme, il faut faire une transformation vers un rectangle.
Le module de traitement d'image peut se résumer en des étapes principales suivantes :

- Récupération d'une image depuis la webcam.
- Application d'une transformation à l'image (pour travailler dans un rectangle) via les méthodes **cvGetPerspectiveTransform()** et **cvWarpPerspective()** ;
- Recherche de la balle via un calcul de barycentre.
- Recherche des Kheperas via une méthode de clustering.
- Modifier la mémoire partagée.
- Fin de boucle.

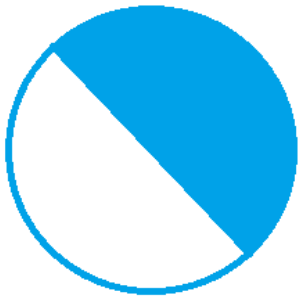
Recherche de la balle

Pour positionner la balle, on utilise les fonctions dans la bibliothèque OpenCV. On applique la méthode de reconnaitre l'image par couleur. Les principes pour retrouver la balle se résume suivant :

- Après avoir l'image de la zone intéressant, on filtre l'image pour garder une seule couleur (ici la couleur rouge, la couleur de la balle)
- Ensuite, on effectue une érosion (par la fonction cvErode()) afin d' éliminer le bruit
- Enfin, on calcule le barycentre des points gardés. Ce point correspond au centre de la balle.

Recherche des Kheperas

Pour chaque Khepera, on met sur son dessus deux demi-cerlces de deux couleur différentes : un blanc et un bleu.



Les principes pour rechercher les kheperas sont :

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

Amélioration possibles

Il y a deux axes d'amélioration pour ce qui est de l'analyse d'image :

- Améliorer l'analyse en réduisant les zones analysées (des rectangles autour des Kheperas)
- Exploiter la bibliothèque OpenCV, il y a des fonctions pour reconnaitre la forme qui est plus rapide pour
  reconnaitre les deux Kheperas au lieu d'utiliser la reconnaissance par couleur

Conclusion

Le projet SCOR a été pour l'ensemble de l'équipe une excellente expérience. D'une part par l'approche qu'il faut prendre du problème (objectif fixé par ses envies plutôt que par des spécifications). D'autre part par son aspect ludique (tout est autour d'un jeu).
Cependant, les problèmes de puissance des moteurs rencontrés ont un peu orienté notre objectif vu qu'au final la grande partie de notre projet a porté sur le déplacement des robots.

Au final le projet SCOR reste fun et très intéressant.