Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

AUTOMATIC
CONTROL
LABORATORY

Mattia Risiglione

RoboCup: Coordinated defensive strategy

Semester Thesis

Supervision

Prof. Dr. John Lygeros
Dr. Benjamin Flamm
Dr. Alexandros Tanzanakis

June 2019

Contents

Abstract

In the context of the five-a-side Standard Platform League of RoboCup, this project proposes and implements a coordinated defensive strategy to improve our current behavior.
The chosen strategy aims to reduce the chance for attacking opponents to score a goal. Considering current capabilities of competitors in the league, this main objective has been split into two tasks: repossessing the ball when lost, and protecting the goal.
In order to reduce computational complexity, a heuristic approach dynamically finds reference positions to place defending players. Role substitutions between defenders and attackers are introduced to allow attackers to participate in defense when needed.
The proposed strategy is compared to the previous defensive strategy to motivate improvements.

Chapter 1

Introduction

RoboCup (Robot Soccer World Cup) is an international robotics competition in the contest of soccer.
The focus of the competition is to push forward the barrier of technology, promoting research in the main area of robotics.
The different teams involved need to build and/or program robots to make them achieving certain tasks fully autonomously.
The *Nomadz* team from ETHZ plays in the Standard Platform League, a specific section of the competition created to promote software, rather than mechanics, design.
For this purpose, all teams need to be equipped with same hardware: specifically the robots used are small scale humanoids called NAO, produced by Softbank Robotics.
The v6.0 is the most recent version of the robot available on the market and currently used by all teams.
Although the available computation power has increased during the years, it still represents a barrier due to the huge amount of operations running in parallel.
The embedded processor is a 4 core ATOM at 1.91GHz.

Figure 1.1: Soccer match

1.1 B-Human Framework

Every year the best four teams in the competition should publish their code and the corresponding code release.
This allows newcomers to start from an already working platform and being able to give a quick contribution to the community.
For what regards our team, the starting point was the work released in 2013 by the German team B-Human from University of Bremen, winner that year.

The choice of using the work from this team is mainly due to its modularity.
The whole control program is composed by different modules, each designed for a certain task, e.g motion, image processing, self-localization, communication.
Modules require input and produce a certain output the so-called representations.
These representations are shared between modules (see figure 1.3). In this way, every module can work with the representations of every other module if required.
Therefore, the modules have to be executed in a specific order to make the whole system working, this is done by a module scheduler of the framework.
The function update (the representation of the module) of each module is called at any iteration of the robot controller and updates their representations.
When a new module is declared, it has to have specified all its dependencies of other modules in order to use them.
The framework combines all these dependencies such that every module works only with the up-dated representations of the other modules.

1.2 CABSL

Amongst all other things, in 2013 release [1], B-Human published their C++ framework to encode behavior called CABSL, *C-based Agent Behavior Specification Language*.
In CABSL language, the base elements used are: options, states, transitions, actions.
Each option is composed by states in which specific actions can be executed or lower options can be called.
Every options have the success state (called *target_ state*) and the failure state (called *aborted_ state*).
Higher level options can make use of these two special states to check what happened in the called options.
Inside each state, state transitions can be defined.
The state machine stays in one state until one of the condition is not satisfied.
A special class of transition is represented by the common transitions. They are checked every time and can be defined once and being valid for all states.

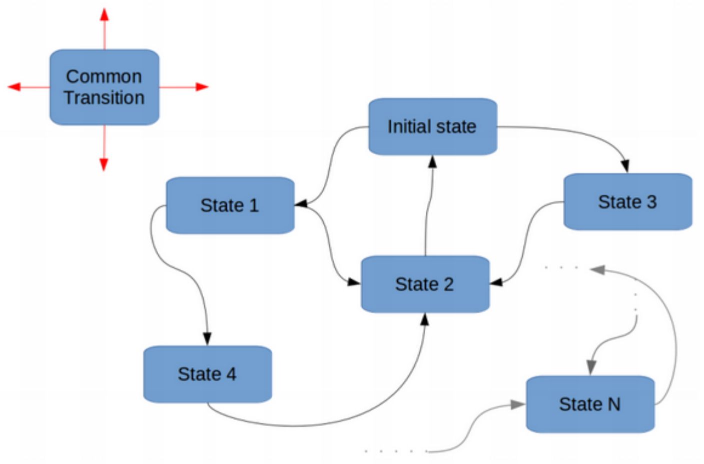


Figure 1.2: Example of transition inside option

Chapter 2

Previous Strategy

The first step of the project was to analyze the previous strategy implemented.
The main structure of the defense behavior came from work pubblished by B-Human with some improvements done along the years.
To refer to the implemented code, check the header files into the folder: (*Src/Modules/Behaviour-Control2013/Roles/*).

2.1 State machine

We currently have an initial option, called *PlayingState*, that works as a role provider.
At the start of the game, through a configuration files roles are assigned based on the number of robot. For defenders, robots number 2 and 3 are assigned to this role.
After a player gets assigned to a role, it run afterwards a specific state machine.
Defenders run individually the following state machine:

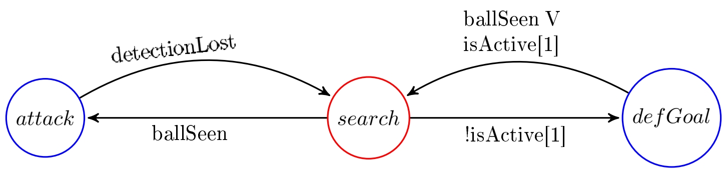


Figure 2.1: Simplified version of defender state machine

The initial state is called *search* and it is used by every defender to scan environment around and give a contribution to the team in detecting the ball.
In this state, the robot turns the head left and right, while rotating around itself.
If a ball is seen during scanning, the robot will go to the success state and it waits for the above option to take over.
When the goalie becomes inactive, meaning it is penalized (look official rules), the defenders retreat to protect the goal. Some default position close to the penalty area are assigned for this emergency case.
Finally, when the ball is detected, the defenders decide to transition to the attack mode to engage the ball. Actions here are taken accordingly to where strikers are, but mainly the ball is cleared away or kicked into the opponent goal. There is also a limit for the robot to go in order to have a robot too far away from its goal.
Vice-versa, when localization of ball is lost, defenders go back to the *search* state.

2.2 Considerations

The first observation that can be done is the lack of a coordination.
For example, both defenders can transition to the *attack* state at the same time; there is no check to avoid task overlapping in this sense.
Moreover, defenders engage the ball as soon as they detect it and that happens really fast from the kick-off because every player is able to localize the ball within 3 meters far from its place.
In most of the matches, we were ending up with no players able to defend; this behavior resulted to be too aggressive.
Additionally, it is not very efficient to keep defenders in fixed position when they do not engage. A dynamic placement should be introduced to make defenders faster when they need to engage.
As a final consideration, the previous strategy does not try to optimize any efficient criteria and does not manage situations in which defenders are lost, e.g when they get penalized or a piece of hardware breaks.

2.2.1 Drawbacks summary

For the sake of completeness and to really highlights the last considerations done, it is reported here below a list of the actual drawbacks.

- Lack of coordination
- Aggressive conditions for engaging
- No dynamic placement
- Static assignment of role
- No optimality
- No fault management

Chapter 3

Defense Strategy

In many team sports, defense is a term referred to the action of preventing an opponent from scoring.
In order to do so, being inspired by what in real soccer is done, we created a tactics to be played by a group of our players.
The tactics has two main goals:

- protecting the goal
- repossessing the ball when lost

In the following section we describe a procedure that we would have liked to adopt and we tried to use at the beginning of the project.
However, the nature of the problem makes the final model complicated and some difficulties were met in trying to use this approach. We will discuss all of them here.

3.1 Optimal plan

One possible way for fulfilling the goals is to find a global policy solving a centralized optimization problem that has the following structure:

$$
\begin{aligned}
\max_{U} \quad & (\mathit{goal\ protection}) + (\mathit{ball\ interception}) \\
\text{s.t} \quad & defender\ dynamics \\
& \mathit{opponent\ dynamics} \\
& interception\ constraints \\
& collision\ avoidance \\
& state\ bounds \\
& control\ bounds \\
& \mathit{initial\ conditions}
\end{aligned}
$$

By using the output control sequence, the agents would be placed in space to maximize the cost function taken.

3.2 Modeling difficulties

This section explains what makes hard the optimization problem of previous section.
The picture depicted below will be useful to depict some points analyzed.

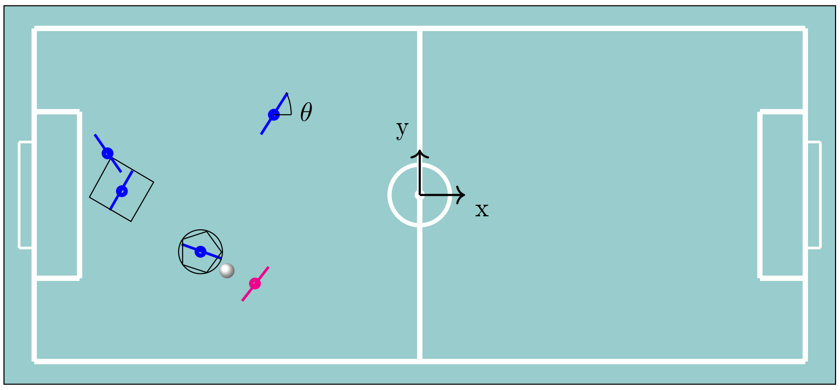


Figure 3.1: How to deal with interception and collision avoidance

3.2.1 Defenders

Kinematic equations with respect to the global reference frame results to be:

$$\begin{bmatrix} x_d[k+1] \\ y_d[k+1] \\ \theta_d[k+1] \end{bmatrix} = \begin{bmatrix} x_d[k] \\ y_d[k] \\ \theta_d[k] \end{bmatrix} + \begin{bmatrix} T_s[k]v[k]\cos(\theta[k]) \\ T_s[k]v[k]\sin(\theta[k]) \\ T_s[k]\omega[k] \end{bmatrix}$$

The orientation term introduces nonlinearities that are hard to be removed since a linearization would require a nominal trajectory to follow and this would be obtained only afterwards.

3.2.2 Inputs

The input of every defender are the linear velocity and the angular velocity.
In order to prevent rotations at high speed or translations at high rotational velocity, one idea, to make the model closer to reality, would be to use as input space an ellipse.

$$\frac{v^2[k]}{v_{max}^2} + \frac{w^2[k]}{w_{max}^2} \leq 1 \tag{3.1}$$

By going for this approach, we would introduce here a quadratic non linear constraint. The rest of constraints are bounds for the control inputs.

3.2.3 Interception

If tasks are not allocated, we need to penalize the whole system, instead of a single player, for ball not intercepted. The reason for which we are talking about interception, and not repossession in general, is because the ball has some dynamics (depending on the opponent's behavior) and the repossession could happen along the ball's trajectory.
In order to express a global penalty, we should firstly define what really means to intercept the ball for a defender.

The idea is to attach a polygon to every player and comes from [2].
Given $(x_d[k], y_d[k])$, the coordinate at time k of the defender, we can define

$$I[k] := \{(x, y) \mid (x - x_d[k]) \sin \frac{2\pi m}{M_i} + (y - y_d[k]) \cos \frac{2\pi m}{M_i} \leq R\} \qquad ,\forall m = 1, ..., M_i$$

as the set describing the interior points of the polygon, inscribed by a circle of radius R.
A binary variable is introduced for every player and becomes true when the ball is inside the set $I[k]$. Meaning:

$$i[k] = 1 <=> (x_b[k], y_b[k]) \in I[k]$$

Additionally we should make sure that when $i[k]$ is true, all the constraints are active.

$$i[k] = 1 <=> i_m[k] = 1 \;\; \forall m = 1, .., M_i \tag{3.2}$$

Working with both sides of eq. 3.2 the following sets of inequalities can be easily derived.

$$\begin{cases} (x_b[k] - x_d[k]) \sin \frac{2\pi m}{M} + (y_b[k] - y_d[k]) \cos \frac{2\pi m}{M} \leq R + H(1 - i_m[k]) \\ (x_b[k] - x_d[k]) \sin \frac{2\pi m}{M} + (y_b[k] - y_d[k]) \cos \frac{2\pi m}{M} \geq \epsilon - i_m[k](H + \epsilon) \end{cases} \tag{3.3}$$

$\forall m = 1, ..., M$, $\forall k = 1, ..., T_f$. $\epsilon > 0$, M maximum of the left hand of 3.3 . Additional inequalities from eq. 3.2 are:

$$\begin{cases} i_m[k] - i[k] \geq 0 \\ i[k] + \sum_{l=1}^{M} (1 - i_l[k]) \geq 1 \end{cases}$$

$\forall m = 1, ..., M$, $\forall k = 1, ..., T_f$.

3.2.4 Inter-Robot C.A

To avoid collisions between teammates, at each sampling, one of the two dimensional distance must be greater than a certain threshold.
$\forall$ pair of robot i and j, s.t $i > j$, it must be true one of the following equations:

$$|x_i[k] - x_j[k]| \geq h_r \;\; \vee \;\; |y_i[k] - y_j[k]| \geq h_r \tag{3.4}$$

A reasonable value in 3.4 for the lower bound of distance is the height of the robot.
This would prevent a falling robot to drop one of its teammates.
Introducing binary slack variable and rewriting the absolute value as the union of two inequalities:

$$\begin{cases} x_i[k] - x_j[k] \geq h_r - x_{max} h_1 \\ x_j[k] - x_i[k] \geq h_r - x_{max} h_2 \\ y_i[k] - y_j[k] \geq h_r - y_{max} h_3 \\ y_j[k] - y_i[k] \geq h_r - y_{max} h_4 \\ \sum_{l=1}^{4} h_l \leq 3 \end{cases} \tag{3.5}$$

where h_r is the robot's height, x_{max} and y_{max} are respectively the width and the height of the field.
The last constraint, who links the slack binary variables used, makes sure at least one constraint is active.
In eq. 3.5, j must hold for every other teammate and the inequalities above are valid for each time step k.

3.2.5 Computational analysis

Based on procedure previously proposed to model interception and collision avoidance, we would need the following number of binary variables:

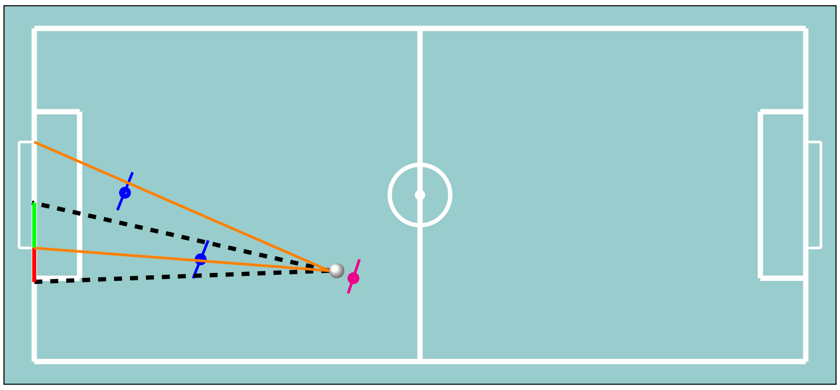
- Inter-collision avoidance in 2D: $\frac{N_D!}{2!(N_D-2)!} \cdot 4$, where N_D is the number of defenders.

- interception: $N_D \cdot (1+s)$, with s number of polygon sides

Furthermore, more effort should be added for formulating the goal protection. Different things should be checked:

- if defender is behind ball
- if its perspective projection intersects goal
- if part of the goal covered was left free by other players

Binary variables are suitable for managing this logic, but this would only increase the binary variables used.



3.3 Heuristics vs Optimality

The resulting formulation is in the form of a MINLP problem.
For the case of three defenders (a reasonable number due to the different goals to achieve) we would have 23 binary variables only in the constraints. Being limited in computational power and modeling accuracy, this approach would not result to be very efficient in real-time.
Using a decentralized approach would be better and the problem has the right structure for it due to a possible task separation, but more computations then would be required for the collision avoidance.
What we are really interested are the reference positions to send defenders, rather than doing trajectory optimization; the main question is to where place defenders and we could have answered

by solving the centralized optimal control problem.
Taking into account these considerations, we tried to replicate goals without necessarily modeling everything.
Especially, an heuristic approach is introduced to find optimal position for the goal protection task, the one looking more demanding.
The other task is treated by sending directly the defender to the current ball's position. By iterating this approach online, we can avoid to calculate offline the ball's trajectory. The approach is then more robust, due to the uncertainties in predicting the ball's trajectory offline.

Chapter 4

Approach proposed

In previous chapter we discussed the main difficulties faced in trying to find a strategy solving an optimal control problem.
This chapter shows the steps performed to tackle the issue.

4.1 Formulating tasks

The tactics is composed by simple actions that should allow the team to achieve the main goal. To deny other team's forwards to penetrate or send the ball in our goal, we create the following tasks:

- create a moving barrier to protect our goal from kicks
- attacking the ball to repossess it

These two tasks will be assigned to our players and we make sure that the team is able to fulfill these two tasks.

4.2 Team structure

Each team is composed by five players. In terms of roles, the main assumption is to keep one goalie, two attackers and two defenders. This would balance the offense and defense behavior.
The attackers are two players who coordinate with each other to bring as quickly as possible the ball into opponent goal.
While the defenders are the two left players on the field that coordinate with each other to defend our goal creating a barrier.

4.3 Task allocation

Based on the reformulated tasks and the considerations done on the number of players involved for achieving them, we can list the job as:

1. go to ball
2. join first position in barrier
3. join second position in barrier

In our case, we have three jobs to be assigned and each robot can execute at most one work at any given time.
First of all, the ball-attacking task has an higher priority because repossessing the ball prevents completely from any possible opponent's threat.
Now the question is how to distribute the three jobs to three players. In literature, different ways for solving task allocation problems have been proposed.
One common approach is based on assigning a score to every job and find the best candidate for it. However, an utility function combines quantities having different meaning and unit of measure. Additionally, we need to take care about computation and that could grow very quickly. For lowering done the number of combinations, we firstly analyze how our team is composed.
Since attackers, striker and supporter, are in charge of going to the ball at any given instant of time and have already an internal coordination for it, we decide to assign, at the initial instant of time, the repossession of the ball to its players.
Then the logic is to improve the performance of attack and assigning always this job to the fastest player in the team.

4.3.1 Interception

The reason behind the choice of sending one agent to the ball rather than trying to intercept the ball is due to mainly lack of time and available resources. We do not have any way to forecast the ball trajectory and we still have a noisy observation of the ball's velocity. Moreover, the actual mean position of the ball is not accurate yet.
Sending the robot to the ball makes the task more efficient.

4.4 Role Substitutios

The logic is to substitute players in attack with any other player faster in reaching the ball.
However, since we do not want to leave any free job, we need to substitute a defender with an attacker, in charge of replacing its teammate leaving defense.
We want to minimize overall The algorithm is the following:

Algorithm

```
1) Calculate time to reach ball for each player
2) Select fastest player and check role
3) If its role is defender, select fastest attacker to substitute him.
   Execute substitution
```

4.4.1 Communication

In terms of communication, who is playing as defender announces when a role substitution is needed and communicates with the player (striker or supporter) in charge of substituting him.

4.4.2 Delay

To avoid any delay at the beginning of the team, robots are assigned to their roles by default trough a configuration file.
After a certain amount of time, when robots have a better localization and identification of the ball, the dynamic role assignment is allowed. delays at the beginning, players are assigned their immediate roles by default.

4.4.3 Lips

Another issue found is an oscillating role swaps. Due to noisy observation of the ball or a fastest direction change of the opponent, a role substitution cannot be redone within a certain threshold of time. This makes the whole set of transitions more stable.

4.4.4 Time estimate

The criteria used for assigning positions is the time.
In continous time, the problem to solve for finding the shortest time to go to get a certain pose looks:

$$\begin{aligned} \min_{u} \quad & t_F \\ \text{s.t} \quad & \dot{x}_R = f(x_R, u) \\ & x^R_{MIN} \le x_R \le x^R_{MAX} \\ & u_{MIN} \le u_R \le u_{MAX} \\ & x_R(t_F) = x^*(t_F) \end{aligned}$$

However, real motion is managed by a sequence of rotations and translations. We firstly rotate to point the target location and then we go straight to reach it; a final rotation then adjusts the heading to get the desired one.
This means that the overall time for reaching the target pose can be computed as sum of three terms (two rotations and one translation).

Given $\chi_0 = \begin{pmatrix} x_0 \\ y_0 \\ \theta_0 \end{pmatrix}$ and $\chi_f = \begin{pmatrix} x_f \\ y_f \\ \theta_f \end{pmatrix}$, the time to reach final pose is calculated as:

$$t = \frac{\|x_f - x_0\|_2}{v_{MAX}} + \frac{|\theta_p - \theta_0| + |\theta_f - \theta_p|}{w_{MAX}}$$

where $\theta_p = \arctan \frac{y_f - y_0}{x_f - x_0}$

4.5 Barrier

We want to define reference positions to dynamically place the two defenders for creating a barrier. Some assumptions are considered:

1. block shooting must be prioritized criteria for positioning
2. from previous tournament analysis, passes almost never happen
3. $d_{min} \geq h_R$, where d_{min} is minimum distance between players, and h_R is the height of a robot

The second consideration allows to not treat passes and, consequently, we do not mark opponent supporters. The lower bound of third assumption is to prevent one falling robot to not touch nearby teammates.
The main idea for respecting the distance constraint is to assign a bar to every robot and to make sure that bars never intersect with each other.
For having a defensive walls, defenders need to be placed such that the associated bars touch each other and there are no gaps in between.
Additionally we try to place the barrier as possible to the ball and the most important thing is to be sure that the open view angle (indicated by α in picture below) is blocked. In this way, whenever the opponent decides to shot, the barrier will block it.
An idea placement is depicted in the picture below. From now on, the critical triangle refers to the triangle having as vertices the two posts and the ball position at any instant of time.
For what regards the heading, they barrier needs to be perpendicular to the bisector of the critical triangle at the position where is placed..

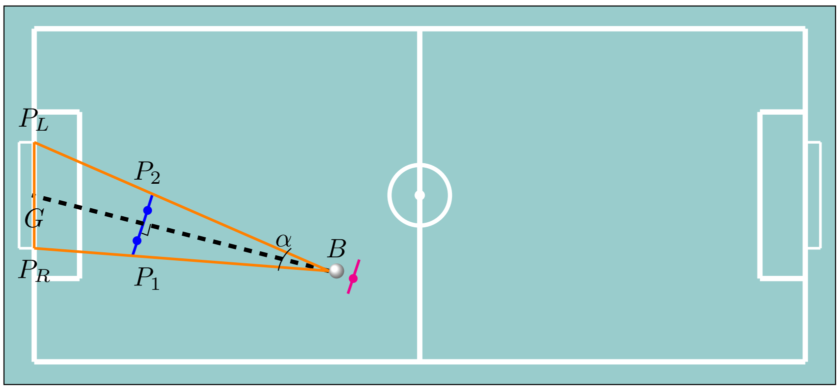


Figure 4.1: Barrier placed on the pitch

The open view angle can be calculated as:

$$\alpha = \arccos \frac{(x_1 - x)(x_2 - y) + (y_1 - y)(y_2 - y)}{\sqrt{(x_1 - x)^2 + (y_1 - y)^2}\sqrt{(x_2 - x)^2 + (y_2 - y)^2}}$$

where (x, y) are coordinates of defender, (x_1, y_1) coordinates of left post, (x_2, y_2) coordinates of right post.
We can now derive the coordinates of the edges in the barrier (P_1 and P_2):

$$P_1 = (x_B - \frac{\frac{h_R}{\cos(\frac{\alpha}{2})}}{\sqrt{1 + m_{P_L B}}}, m_{P_R B} \cdot (-\frac{\frac{h_R}{\cos(\frac{\alpha}{2})}}{\sqrt{1 + m_{P_L B}}}) + y_B) \ )$$

$$P_2 = (x_B - \frac{\frac{h_R}{\cos(\frac{\alpha}{2})}}{\sqrt{1 + m_{P_R B}}}, m_{P_R B} \cdot (-\frac{\frac{h_R}{\cos(\frac{\alpha}{2})}}{\sqrt{1 + m_{P_L B}}}) + y_B) \ )$$

From these two points, knowing the bar's length, we can get the desired position where the two defenders should placed. Then optimal reference poses are:

$$\chi_1^* = \begin{bmatrix} \frac{3P_1^x+P_2^x}{2} \\ \frac{3P_1^y+P_2^y}{2} \\ \arctan(\frac{-1}{m_{BG}}) \end{bmatrix}, \chi_2^* = \begin{bmatrix} \frac{3P_2^x+P_1^x}{2} \\ \frac{3P_2^y+P_1^y}{2} \\ \arctan(\frac{-1}{m_{BG}}) \end{bmatrix}$$

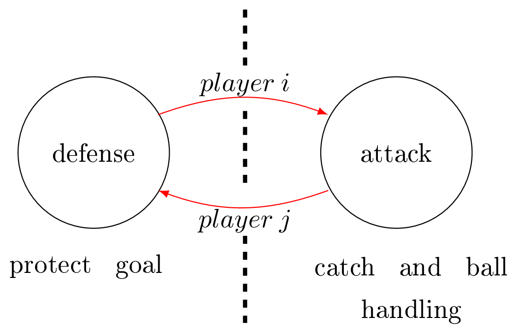
These references are sent to high level motion function *TravelTo*, that deals already the collision avoidance with our teammates.

4.5.1 Considerations

1. Since we do not want goalie to interact with defender, we prevent defenders to get into the goalie area. For this reason, we have the following constraint: $x_i^* \geq x_{goalieArea}$, $\forall i = 1, 2$
2. $X^* = f(X_B)$ => dynamic placement of defenders according to where ball is
3. $\forall t$, $X_1^* \neq X_2^*$ and $y_2^* > y_1^*$ => defenders are spread out
4. $\forall t$, $x_i^* < x_B \quad \forall i = 1, 2$ => always two defender behind ball. If attacker dribbled, always a player available who can engage.

4.6 Task allocation summary

Here is a recap of the whole team structure. We basically created two groups with same number of players, each having a defined role.



These two groups do not exchange ball, but they interact swapping roles.
Just as a remark, as already explained, attack players also join the defense. The distinction above is just based on the type of role.

4.7 Implementation

All behavior regarding defender has been created within CABSL framework.
The new state machine has the following structure:

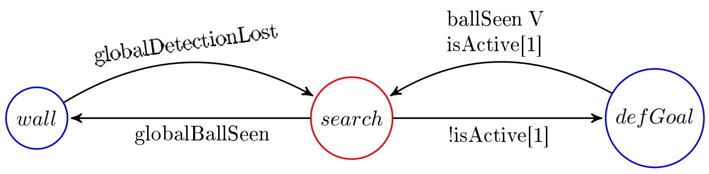


Figure 4.2: actual state machine

The option *attack* has been removed since we do not want to use it anymore. A player who gets assigned defender role can attack the ball only if it is the closest to the ball. In that case, it becomes an attacker and uses the behavior of striker or supporter.
This results to be the best way for a separate design of attack and defense.
The transition to join the wall happens as soon as the combined world model is able to provide us a reliable estimate of the ball's position. Viceversa, when we do not have an available estimate of the position, we transition back to the search state to look for the ball.
Role substitutions have been deployed in the *PlayingState*. Basically, this option so far was only working at the beginning. After the whistle, every agent was initialised to its role, but no common transition were present to change roles.
New variables have been introduced to manage substitutions, meaning a way to communicate to striker or supporter to change role.
The *defGoal* state has been kept since we want to place defenders close to the penalty area in case we cannot use our goalie.

4.8 Simulations

From simulated matches, we investigated how much goal line was covered by the barrier.
In order to answer to this question, a discrete time analysis has been performed, projecting real oriented width of a robot into the goal line and dividing contribution summation with whole goal line.
This analysis has been carried out for the old defensive system and for the new one.
New Defense: percentage of real goal covered is about 73.5 % of the whole goal line.
Old Defense: percentage of real goal covered is about 28.7 % of the whole goal line.

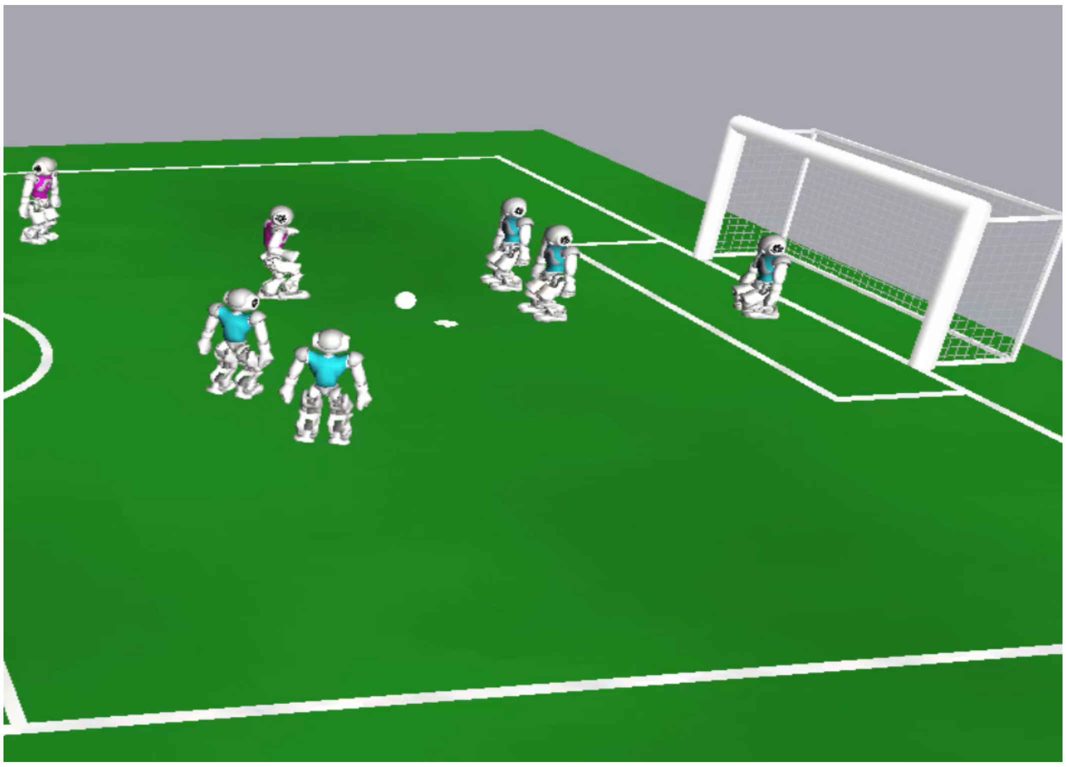


Figure 4.3: Barrier created by players

4.9 Discussion

Results from matches in simulation already show a defense behavior more conservative and always placed in a good position to intercept kicks.
It cannot easily simulate a real scenario due to the a more realiable ground truth on SimRobot. Because of the strict dependencies with the perception module, the whole strategy may be less reliable on real matches.
However, the implementation of a good role assignment allow to reduce the time in repossessing the ball, having essentially the best player assigned to the ball.
Looking to a whole picture of the project, the main goal was to have a coordinated defensive strategy to play when ball was lost. We have now set up specific roles for achieving this objective and we are also able to avoid overlapping of tasks.
The strategy has been tested on a test match with robots running the same code version so far. In the close future, the strategy will be used in the RoboCup World Championships 2019 held in Sidney.
Besides that, in the future a strategy where both defense and attack interact with each other should be implemented. In this way we could take more advantage of the supporters, not involved in defense, and place them in better positions.
Moreover, due to the continuous improvement of locomotion and kicking precision, passes will start to occur and this leads to the necessity of marking opponent supporter. A way for considering opponents to mark would be to assign them a score, based on some parameters like the distance to our own goal, the distance to the ball (to keep into account kicking power limitations) and distance to our own teammates.

Bibliography

[1] B-Human Team *Team Report and Code Release*. University of Bremen, 2013.

[2] Matthew G. Earl and Raffaello D'Andrea *Multi-vehicle Cooperative Control using Mixed Integer Linear Programming*.