MAS Project: Coordination of Soccer Players

Deadline: January 7, 2019

Overview

Robots have to play 5vs.5 matches of 20 minutes (10 minutes per half)[1]. To be effective, the players must coordinate in a multi-robot adversarial environment. Hence, the goal of this project is to implement a coordination routine based on role assignment (see lecture slides).
You will be provided with a void coordination system, empty template of the playing behaviors, and a set of already implemented options that you can exploit. Your solution will be evaluated through a tournament.

Getting the Code

Update the repo that you have used for the homework with the new **project branch**

- git checkout project
- git branch # check that the active branch is the project
- git pull origin project
- compile with *make CONFIG=Release*

Browse the code to find:

- *ContextCoordinator.h/cpp* in
  /Src/Modules/spqr_modules/ContextCoordinator/
  and its representation
- *Role.h*, in */Src/Representations/BehaviorControl/*.
- *UtilityShareProvider.h/cpp* in
  /Src/Modules/spqr_modules/UtilityShareProvider/ and its representation
- *UtilityShare.h* in */Src/Representations/spqr_representation/*.
- options behaviors in */Src/Modules/BehaviorControl/BehaviorControl/*.

[1] more on http://spl.robocup.org/

Role Assignment

The context coordinators determines a set of roles to be triggered depending on the current context of the game. As an example you will find 3 predefined contexts in the code: *playing*, *losing* and *outnumbered*. Each role (in each set) is determined and triggered by the coordinator that assigns them to the players. As for each module, the coordinator has an update function that is invoked at every system cycle. In this function you will find a template with an initial set of roles. You can add other roles and contexts. You have to modify this function to coordinate the team. **Note**. when you download the code, the function is provided with a fixed role assignment and 3 contexts.

In determining the role you can exploit useful information provided to the module. For example, you can read the current robot number in the representation *RobotInfo.h* that contains info about the robot. Modify the update function of the *contextCoordination* module.
Note. When you add/change a role you have to update the *Role.h* representation (requested and included by the coordination module). *Role.h* contains an *enum* with all the roles that can be assigned. The assigned role will be called by the *PlayingState* option and will be executed only during the *Playing* game state.

Utility

Coordination has to be based on utility estimations. Hint: to implement an utility function consider the following elements (you can also add components if you consider them worthy, and/or remove some of them if you consider them useless):

- ball position
- current robot position and orientation
- fallen robots
- opponent robot positions and orientations
- teammate positions and orientations

You have to compute the utilities in the *UtilityShareProvider* module and to read all the utility values, stored in the *UtilityShare* representation, in the *ContextCoordinator* modules. The *UtilityShareProvider* already includes and requests useful representation:

- **OpponentTeamInfo** has information about opponent team color and team score (is part of the *TeamInfo* representation)
- **OwnTeamInfo** has information about your team color and team score (is part of the *TeamInfo* representation)

- **TeamData** has information about the teammates such as their pose and the perceived obstacles.
- **RobotInfo** has information about the local robot (e.g. robot number).
- **RobotPose** has information about the robot position and orientation, and their uncertainty estimation.
- **BallModel** has information about the ball such as its position and the velocity.
- **FallDownState** has information about the state of the robot (e.g. if the robot is fallen).

Project development

Robot coordination with Ground Truth

The first step is to provide a robot coordination that works in the simulation environment by using the SimRobot ground truth. The ground truth is activated whenever a scene with the suffix "Fast" is used to run the simulation. The ground truth provides the ball position, the robot positions and all the perceptions from accurate external observations. In order to use these perceptions use the *FiveRobotsFast.ros2* SimRobot scene. To start the simulation in the SimRobot console type:

- `gc set` #to place robot in their initial positions
- `gc ready` #to trigger the ready game state
- `gc playing` #to trigger the playing game state

Deliverable

Your submission will not be considered if it does not respects the deliverable:

- all global variables and functions that *you* write have to be renamed *lastname_function-name*. Where *lastname* is YOUR lastname.
- rename the *LASTNAME_CONTEXT_Role#NUM* that you will find in the options folder with your lastname and update the Option.h file in: BehaviorControl/BehaviorControl/Options/Option.h
- rename and add the roles in the Role.h representation file. Remember, if you add a context you also have to add the set of roles for that context.
- *LASTNAME_CONTEXT_Role#NUM* have to be completed. You can use existing options (that you find in the options folders) to define your behavior or write your own. One for each context and for each robot in the field.

The deliverable has to be compressed in a single archive that must be named as *LASTNAME_masproject.tar.gz*. It must contain the following files:

- ContextCoordinator.h
- ContextCoordinator.cpp
- UtilityShareProvider.h
- UtilityShareProvider.cpp
- Role.h
- All the roles that you used as *LASTNAME_CONTEXT_ROLE.h*

NOTE: In this project you MUST NOT modify any other representation in the framework.