Player Behavior and Team Strategy for the Robocup 3D Simulation League

Georgios Methenitis

Department of Electronic and Computer Engineering
Technical University of Crete

Thesis Committee
Assistant Professor Michail G. Lagoudakis
Assistant Professor Georgios Chalkiadakis
Professor Minos Garofalakis

Chania, August 2012

Abstract

This thesis presents a complete team design for the RoboCup 3D Simulation League focusing on player behavior, team strategy, and team coordination.

Presentation Outline

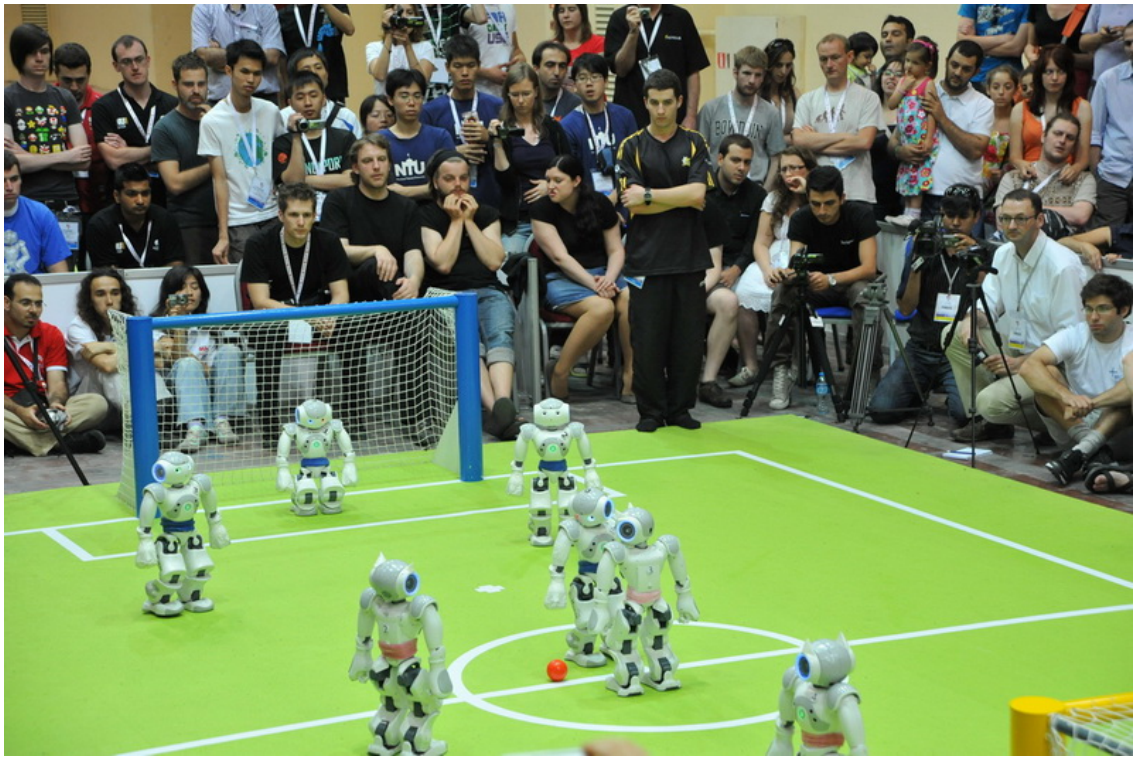
Presentation Outline

Robocup Competition

- RoboCup is an international robotics competition.
- Founded in 1997.
- The official goal of the project is stated as an ambitious endeavor: "By the year 2050, a team of fully autonomous humanoid robot soccer players shall win the soccer game, complying with the official rule of the FIFA, against the winner of the most recent World Cup".

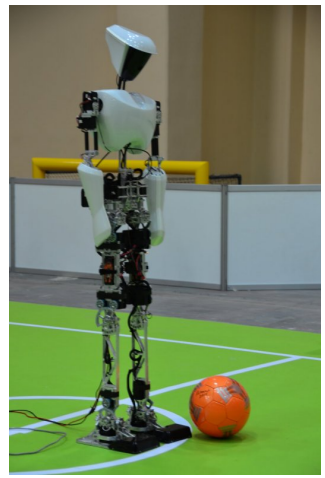
Robocup Soccer Competition

Standard Platform League



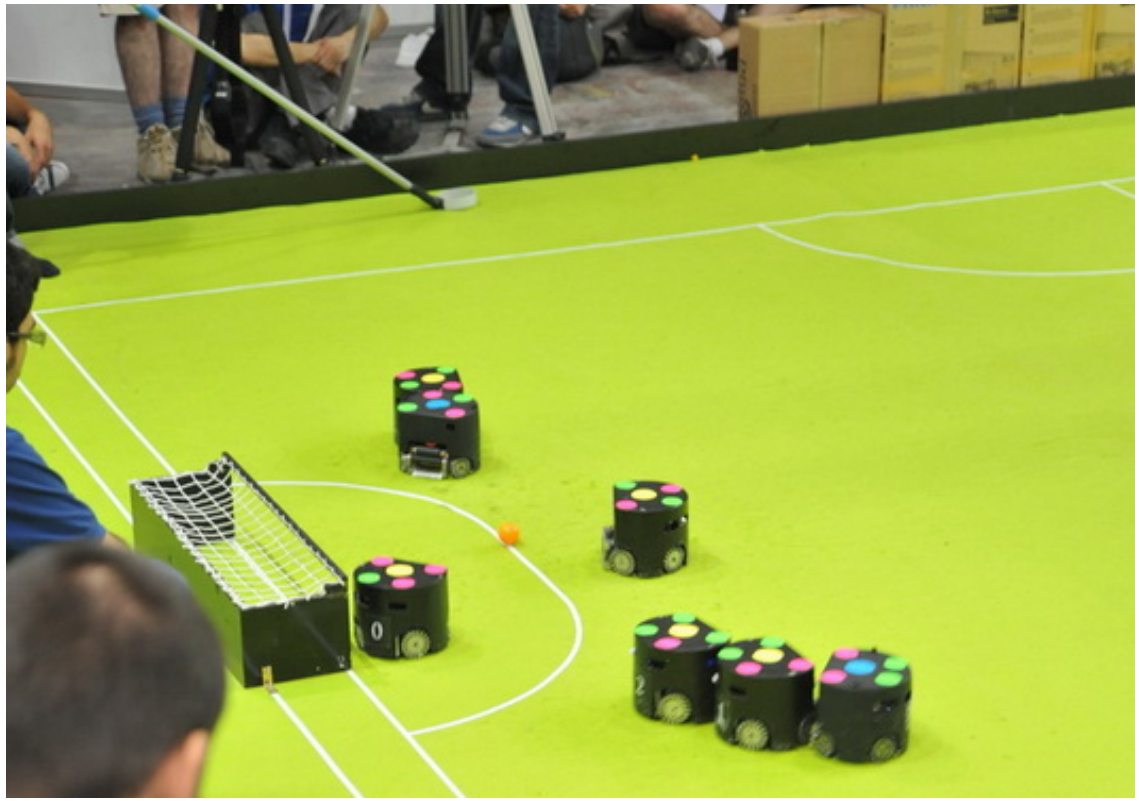
Robocup Soccer Competition

Humanoid League



Robocup Soccer Competition

Small-Size League

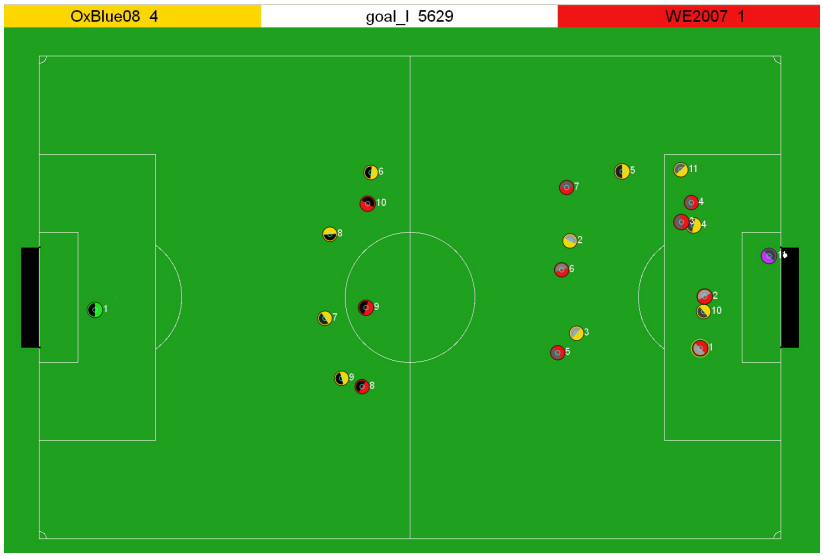


Robocup Soccer Competition

Middle-Size League

Robocup Soccer Competition

Simulation League



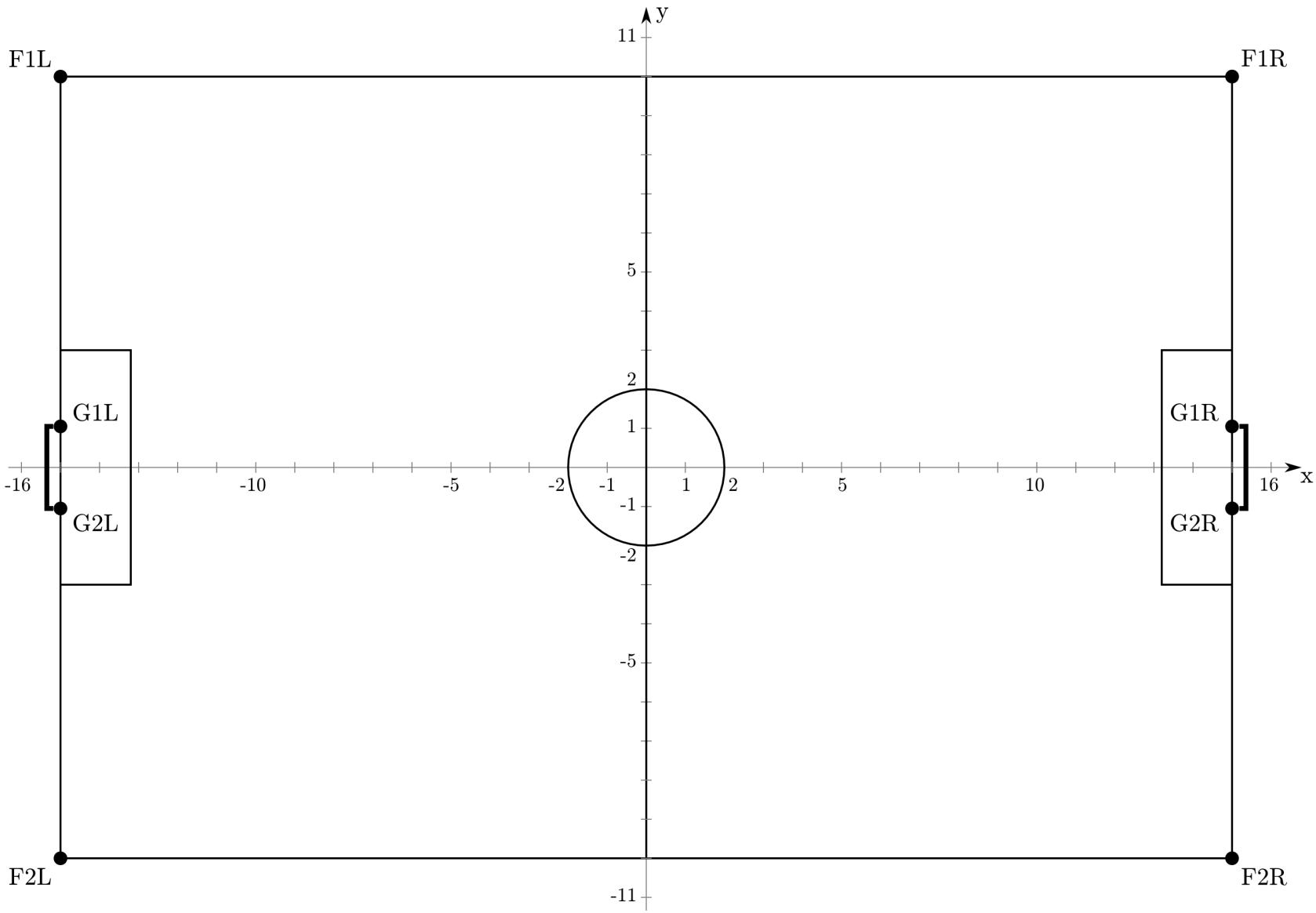
- Virtual games inside a computer
- Independent software agents
- 2D League and 3D League

Presentation Outline

SimSpark

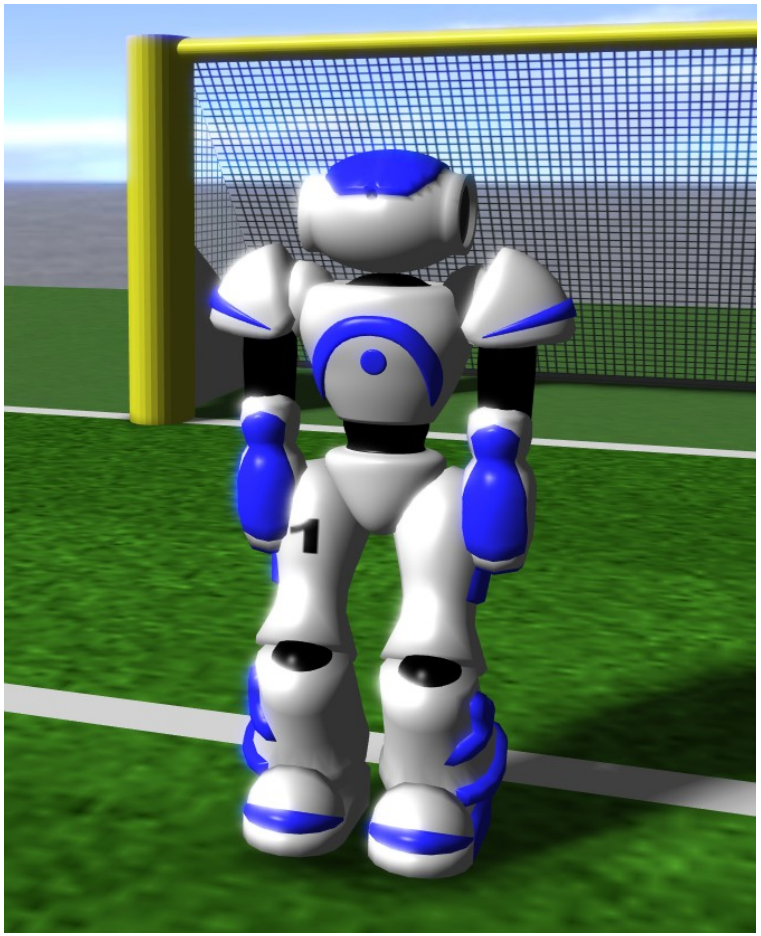
- **SimSpark** is used as the official Robocup 3D simulator.
- Generic physics simulator.
- Multiple agents in three-dimensional environment.
- *Rcssserver3d* is the official competition environment.
- Version 0.6.5: 9 Players, 21m x 14m Field
- Version 0.6.6: 11 Players, 30m x 20m Field

Simulation Soccer Field



Robot Model

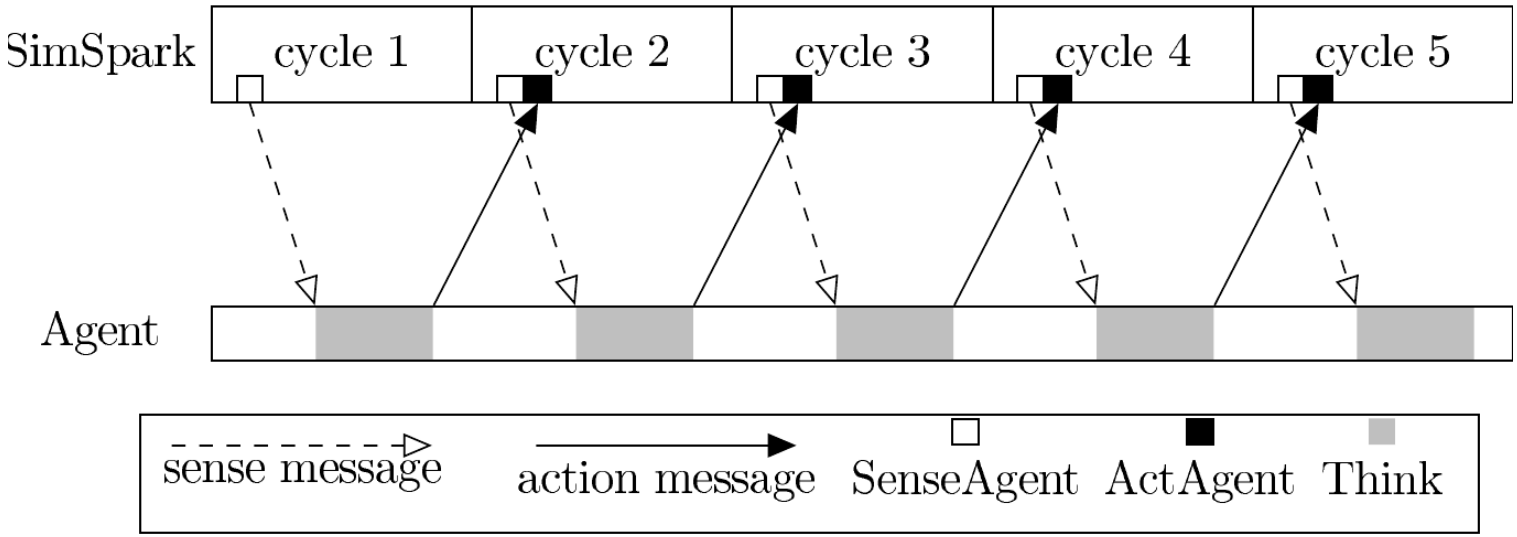
The Nao humanoid robot manufactured by Aldebaran Robotics. Its height is about 57cm and its weight is around 4.5kg.



Server

The SimSpark server hosts the process that manages and advances the simulation.

- Simulation Cycle, 20ms



Agent Perceptors

Perceptors are the senses of an agent.

1. HingeJoint Perceptor
2. ForceResistance Perceptor
3. GyroRate Perceptor
4. Accelerometer Perceptor
5. Vision Perceptor
6. Hear Perceptor
7. GameState Perceptor

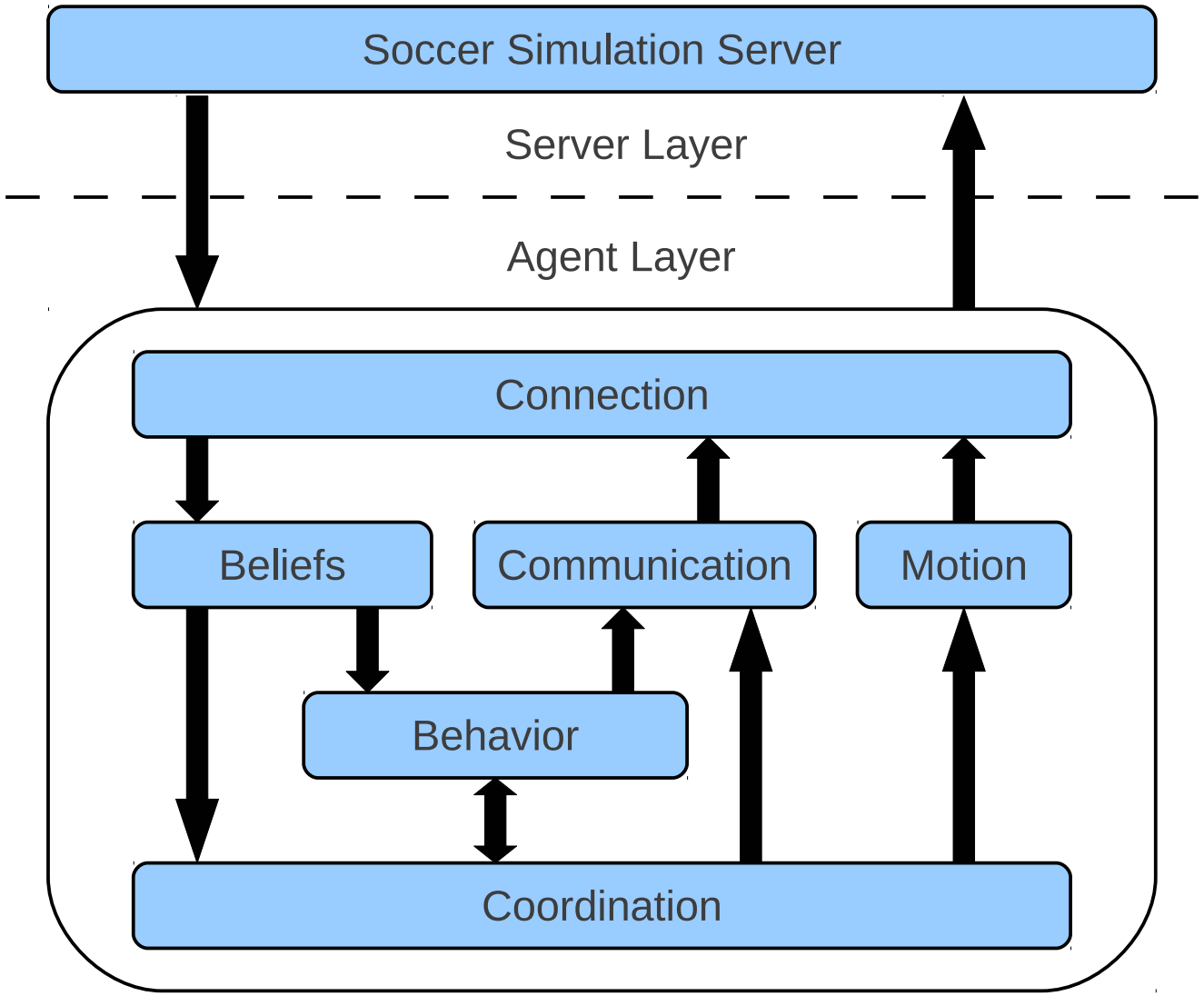
Agent Effectors

Effectors allow agents to perform actions within the simulation.

1. Create Effector
2. Init Effector
3. Beam Effector
4. Synchronize Effector
5. HingeJoint Effector
6. Say Effector

Presentation Outline

Agents Architecture



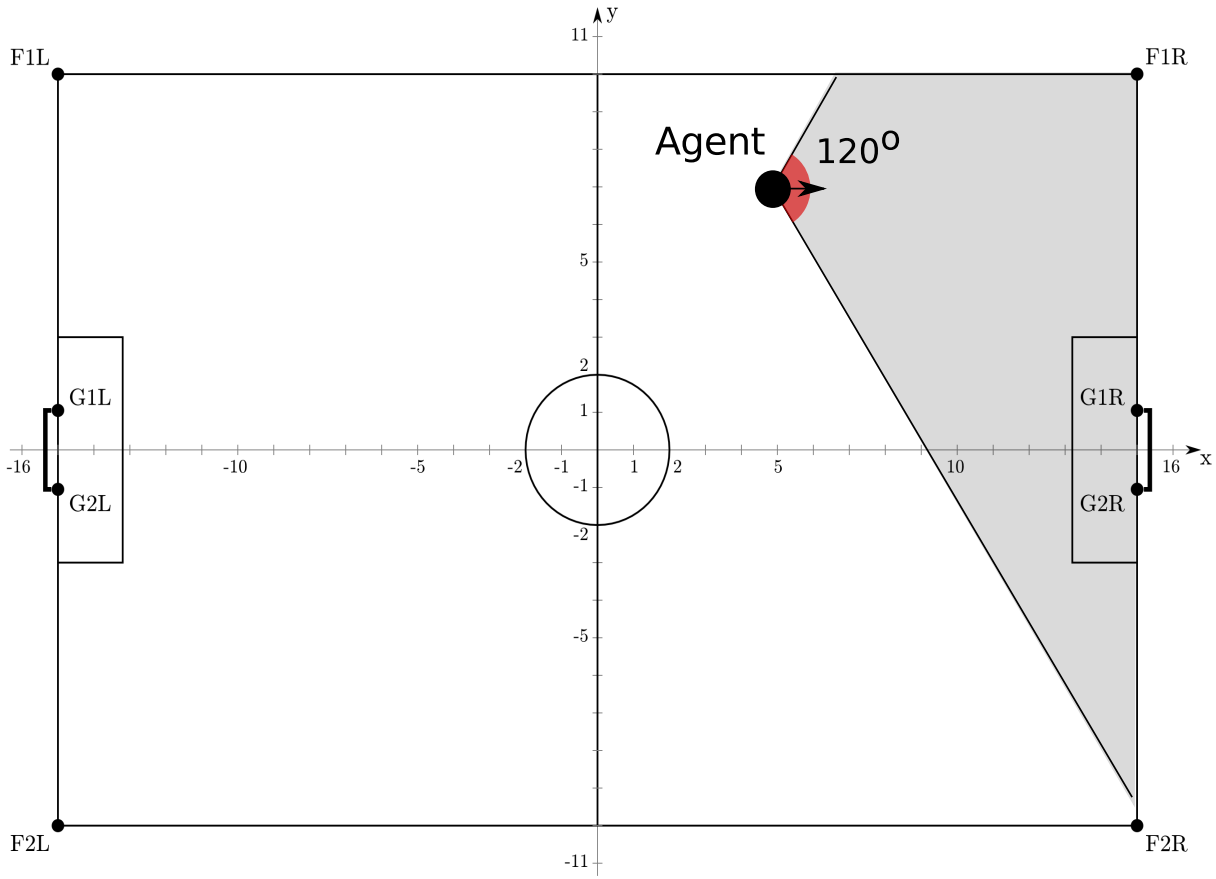
Perception

- Different from real robot soccer games
- Processed but noisy data from sensors
- Perceptor Messages
- Agents update their beliefs by parsing these messages.

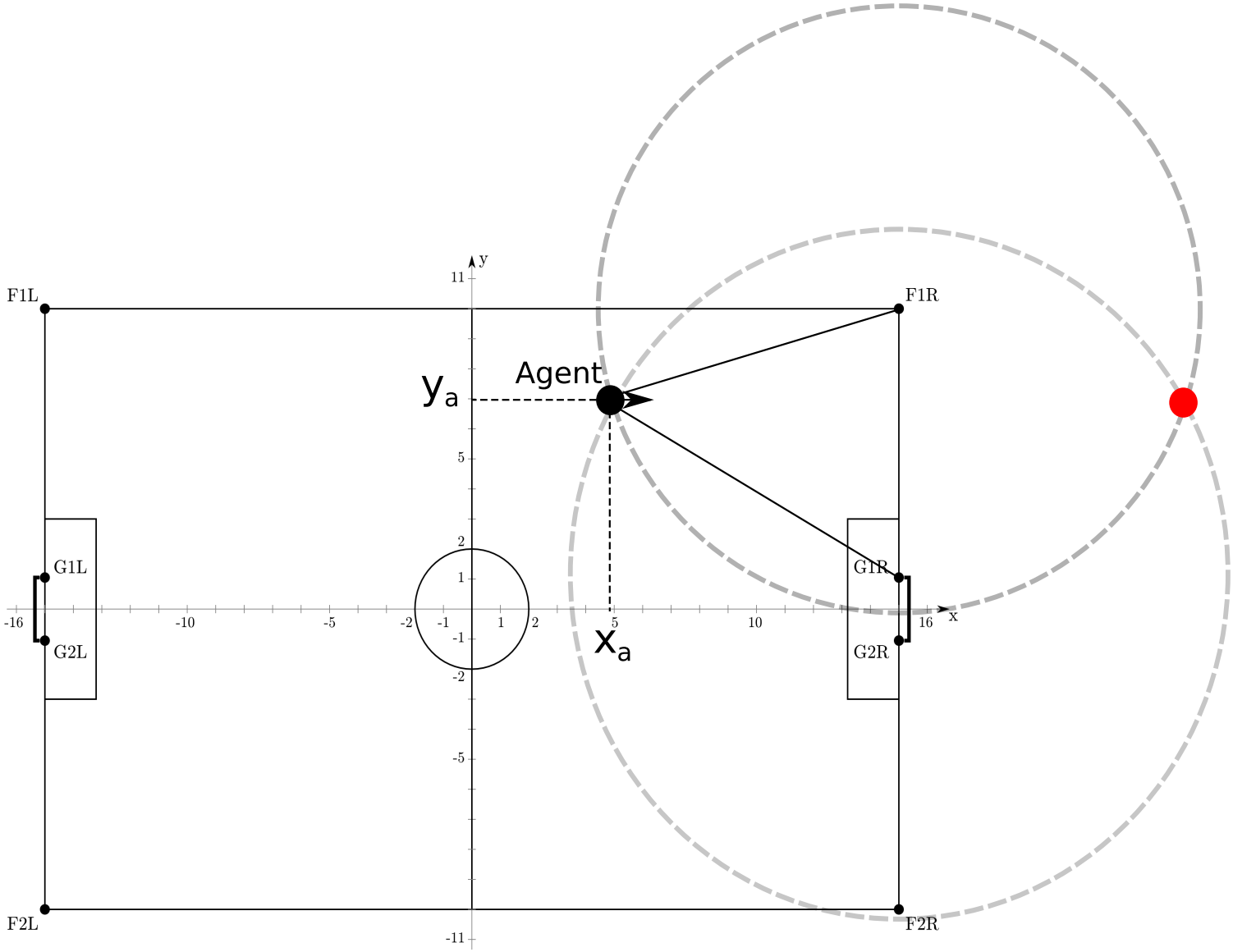
```
(time (now 46.20))(GS (t 0.00) (pm BeforeKickOff))(GYR (n torso)
(rt 0.00 0.00 0.00))(ACC (n torso) (a 0.00 -0.00 9.81))(HJ (n hj
1)(ax 0.00))(HJ (n hj2) (ax 0.01))(See (G2R (pol 14.83 -11.81 1.
08))(G1R (pol 14.54 -3.66 1.12)) (F1R (pol 15.36 19.12 -1.91))(F
2R (pol 17.07 -31.86 -1.83)) (B (pol 4.51 -26.40 -6.15)) (P (tea
m AST_3D)(id 8)(rlowerarm (pol 0.18 -35.78 -21.65)) (llowerarm (
pol 0.19 34.94-21.49)))(L (pol 8.01 -60.03 -3.87) (pol 6.42 51.1
90 -39.13 -5.17))(L (pol 5.91 -39.06 -5.11) (pol 6.28-29.26 -4.8
8)) (L (pol 6.28 29.34 -4.95)(pol 6.16 -19.05 -5.00)))(HJ(n raj1
) (ax -0.01))(HJ (n raj2) (ax -0.00))(HJ (n raj3)(ax -0.00))(HJ(
n raj4) (ax 0.00))(HJ (n laj1) (ax 0.01))(HJ (n laj2) (ax 0.00)) ...
```

Self-Localization

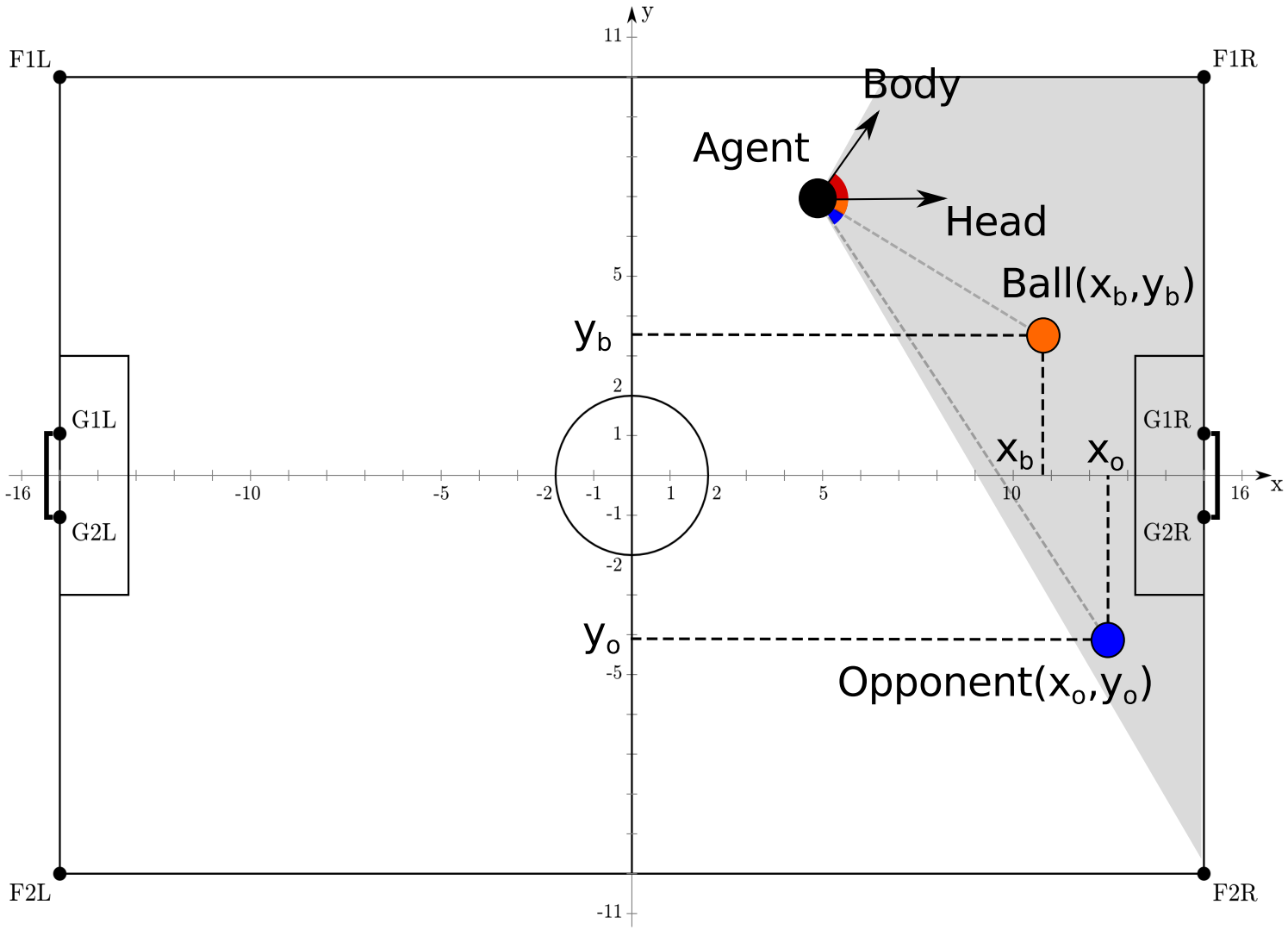
- Self-Localization uses visible landmarks.
- Executed every three cycles (60ms).
- Restricted by field of view (120 Degrees).



Self Localization



Object Localization



Localization Filtering

Why is filtering needed?

- Temporary loss of observations
- Noisy observations

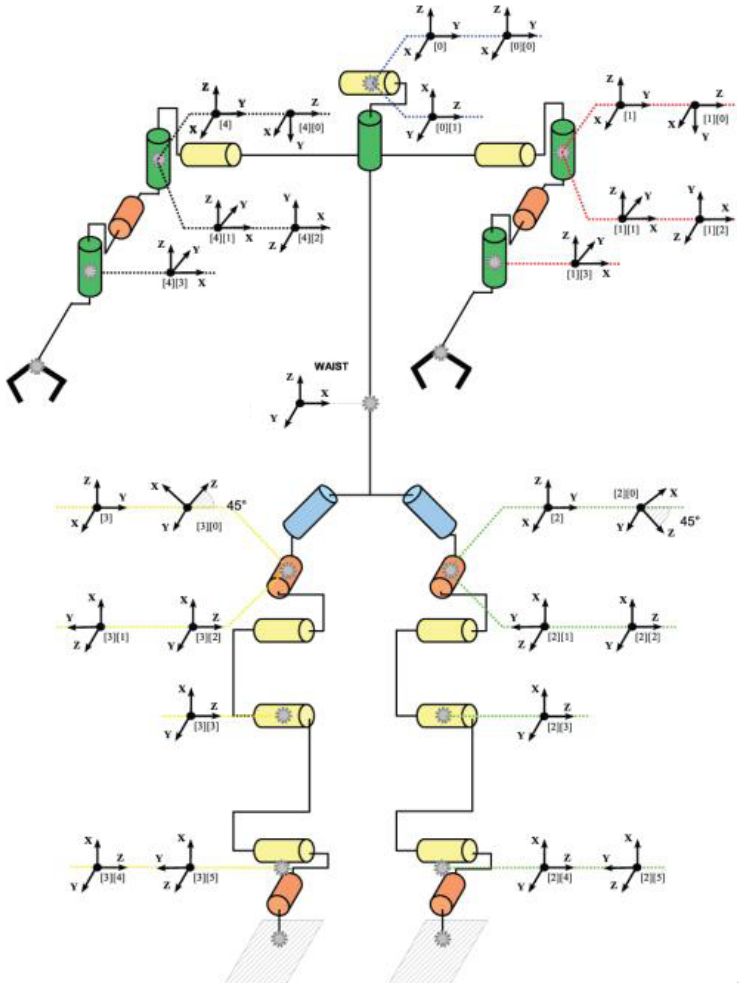
Filtering is applied to ball and self localization.

Algorithm 1 Localization Filtering

```
1:  Input: LastEstimate
2:  Output: FilteredLocation
3:  Queue: a FIFO queue storing the MaxSize (default=10) most recent estimates
4:
5:  if size(Queue) = 0 then
6:      Queue.Add(LastEstimate)
7:  else if LastEstimate ≉ AverageLocation(Queue) then
8:      Queue.Remove()
9:  else
10:     if size(Queue) = MaxSize then
11:         Queue.Remove()
12:     end if
13:     Queue.Add(LastEstimate)
14: end if
15: return AverageLocation(Queue)
```

Nao's Anatomy

The simulated Nao robot comes with 22 degrees of freedom, corresponding to 22 hinge joints.



Motion and Movement

- A motion is commonly defined as a sequence of timed joint poses.
- A pose is a set of values for a set of joints at a given time.

$$
\begin{aligned}
Pose(t) &= \{J_1(t), J_2(t), \ldots, J_n(t)\} \\
Pose(t+1) &= \{J_1(t+1), J_2(t+1), \ldots, J_n(t+1)\} \\
Pose(t+2) &= \{J_1(t+2), J_2(t+2), \ldots, J_n(t+2)\} \\
&\ldots
\end{aligned}
$$

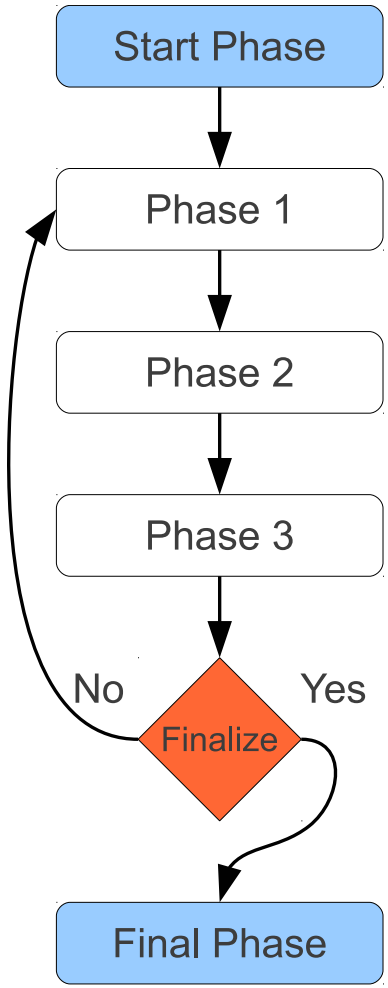
XML-Based Motion Files

```
<phase name="Start" next="Phase1">
   <effectors>
      Joint Values
   </effectors>
   <duration>duration</duration>
</phase>

<phase name="Phase1" next="Phase2">
   <effectors>
      Joint Values
   </effectors>
   <duration>duration</duration>
</phase>

<phase name="Phase2" next="Phase1">
   <effectors>
      Joint Values
   </effectors>
   <duration>duration</duration>
   <finalize>Final</finalize>
</phase>

<phase name="Final">
   <effectors>
      Joint Values
   </effectors>
   <duration>duration</duration>
</phase>
```



XML-Based Motion Controller

- Responsible for executing motions.
- Velocity values are required for control.
- This velocity is computed at the beginning of each phase:

$$JointVelocity = \frac{DesiredJointValue - CurrentJointValue}{PhaseDuration}$$

Text-Based Motion Files

```
#WEBOTS_MOTION,V1.0
LHipYawPitch,LHipRoll,LHipPitch,LKneePitch,LAnklePitch,...
00:00:000,Pose1,0,-0.012,-0.525,1.05,-0.525,0.012,0,...
00:00:040,Pose2,0,-0.011,-0.525,1.05,-0.525,0.011,0,...
00:00:080,Pose3,0,-0.009,-0.525,1.05,-0.525,0.009,0,...
00:00:120,Pose4,0,-0.007,-0.525,1.05,-0.525,0.007,0,...
```

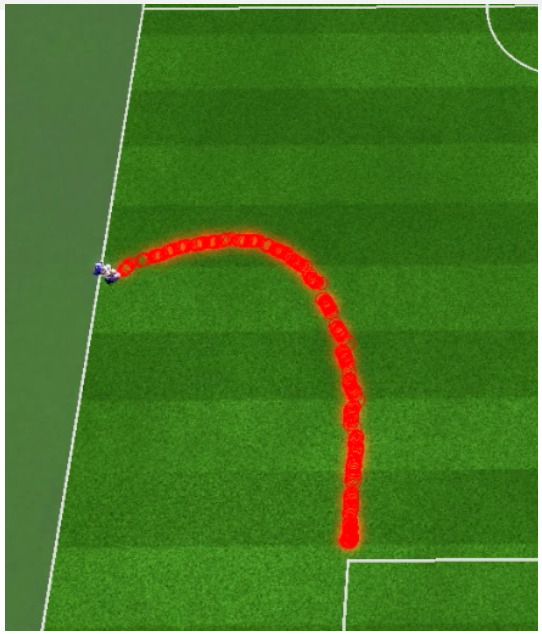
Parameters that can be modified:

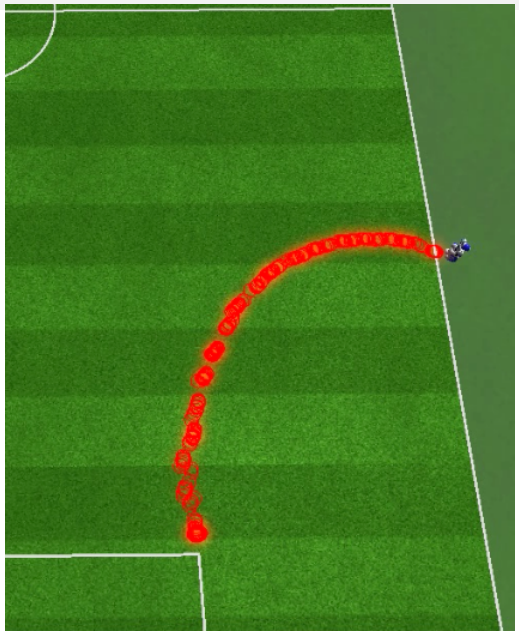
- Duration, time between poses in simulation cycles.
- PoseStep, step for advancing from pose to pose.
- The desired velocity of each joint i is computed by:

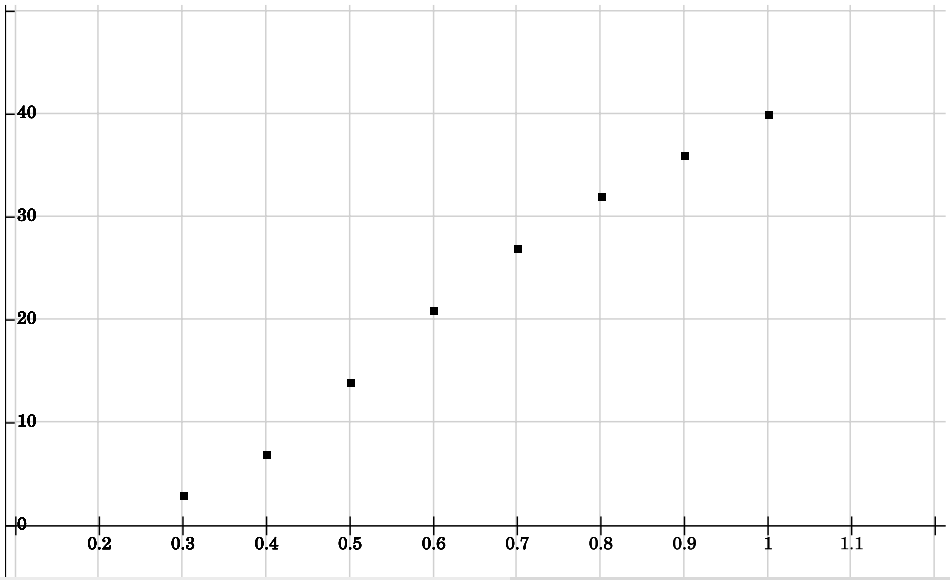
$$JointVelocity_i = \frac{DesiredJointValue_i - CurrentJointValue_i}{Duration \times CycleDuration}$$

Dynamic Motion Elements

- Walk Leaning
- Walk Slowdown
- Dynamic Turn







Basic Actions

Basic actions combine perception and motion files in simple ways.

- Look Straight
- Scan
- Pan Head
- Track Object
- Track Moving Object
- Find Opponent's Goals
- Look For Ball
- Turn To Ball
- Turn To Localize
- Stand Up
- Prepare for Kick

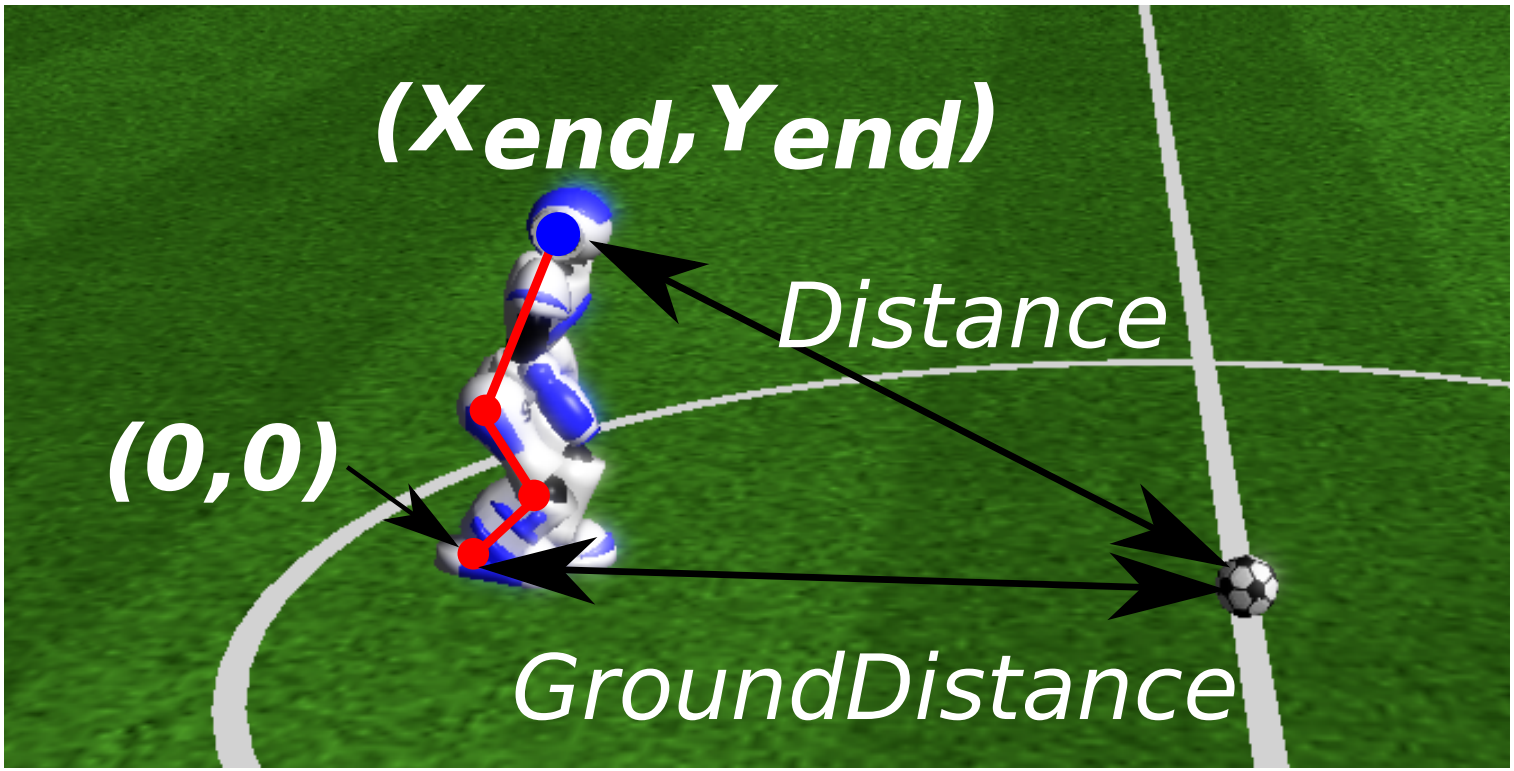
Complex Actions

Complex actions combine perceptual information, motion files, and basic actions. They have a more complicated structure and aim to achieve specific goals.

- Walk To Ball
- On Ball Action
- Avoid Obstacles
- Walk To Coordinate
- Walk To Direction
- Dribble Ball To Direction

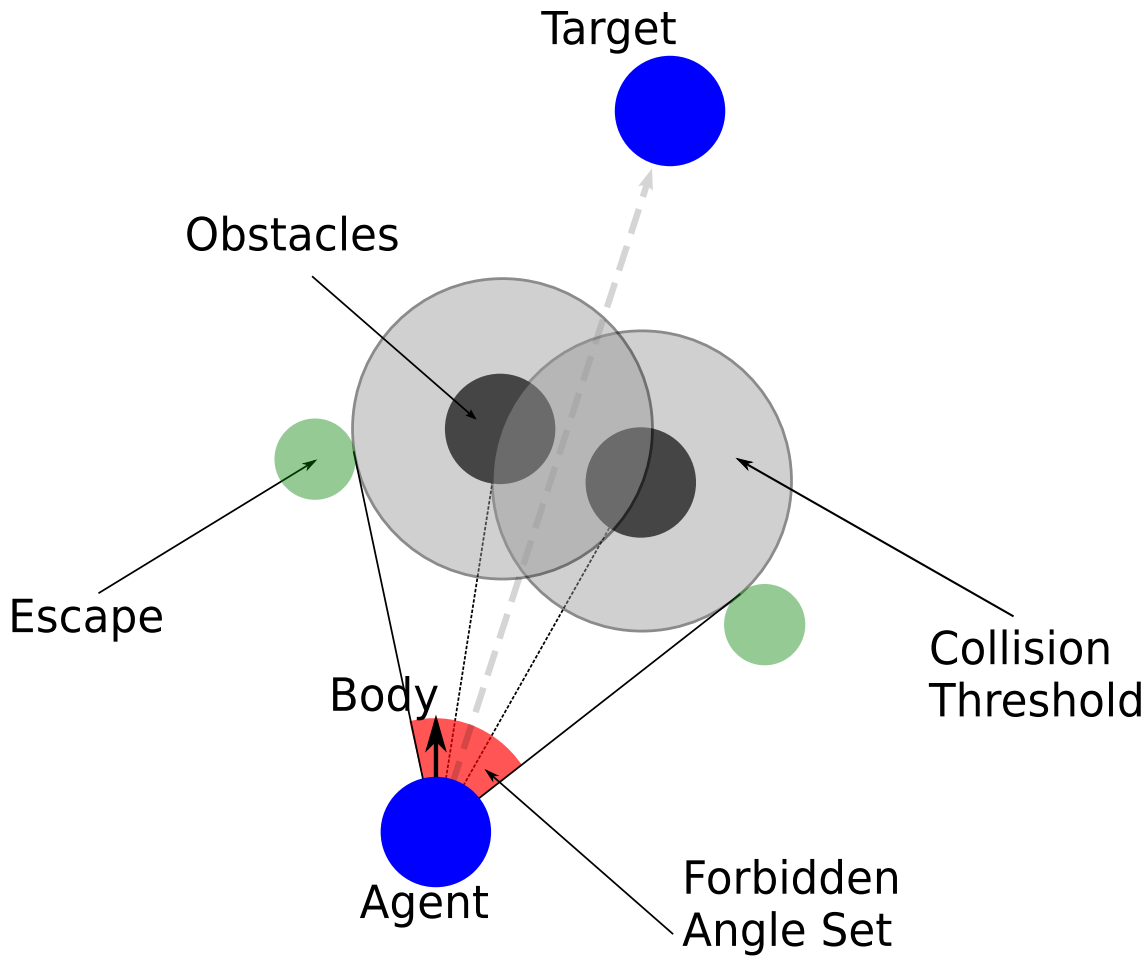
Complex Actions

- Walk to Ball



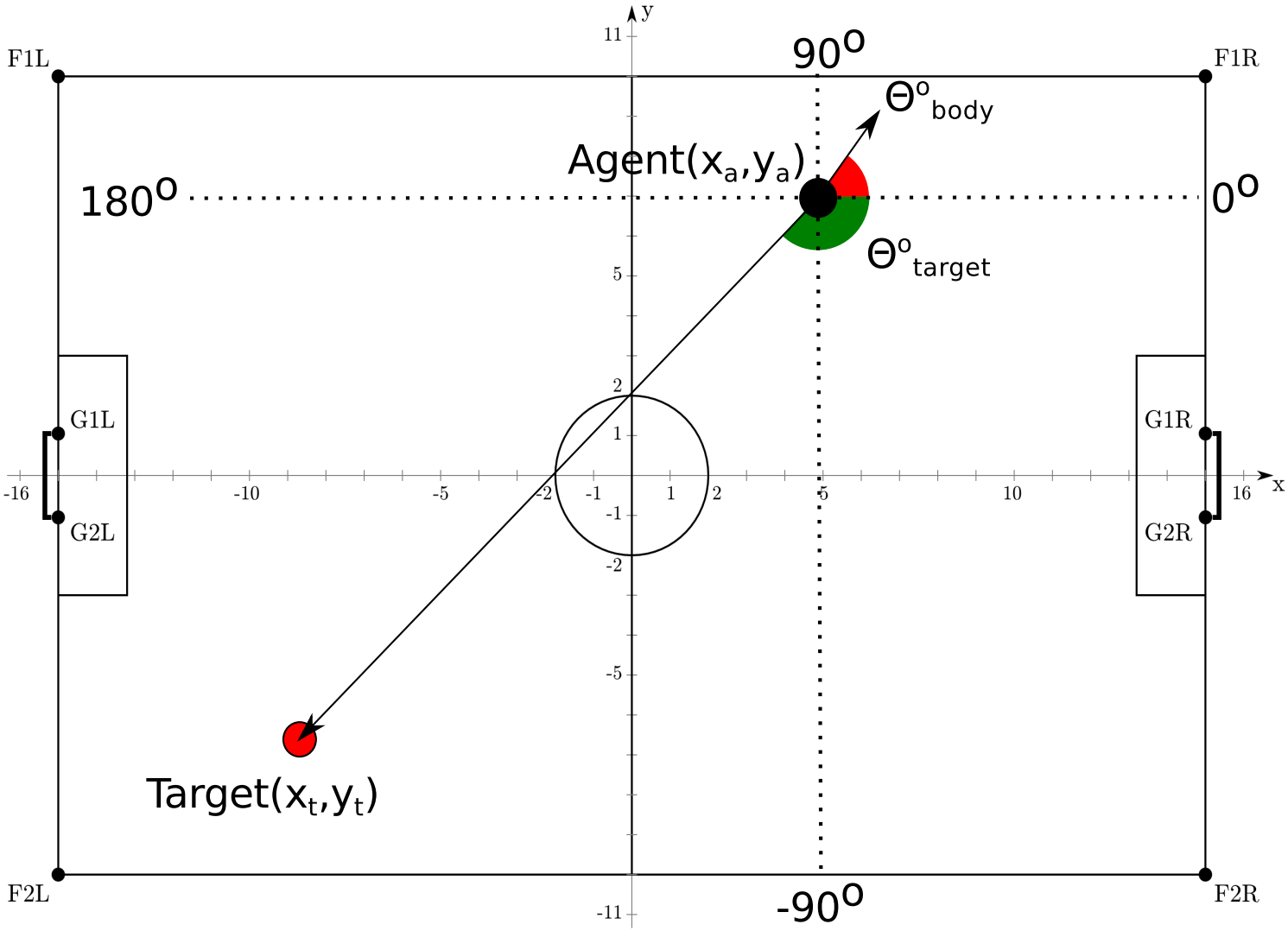
Complex Actions

- Avoid Obstacles



Complex Actions

- Walk to Coordinate



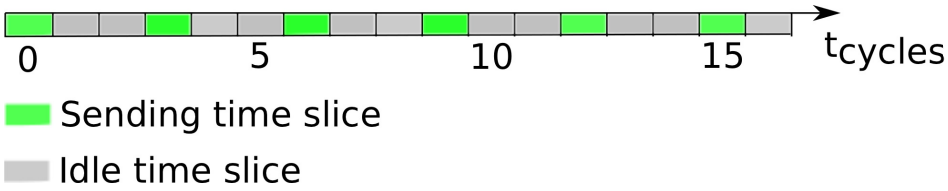
Communication

Restrictions

- Maximum distance of 50 meters.
- Maximum length of 20 ASCII characters.
- Only one message can be heard at any given time.
- Messages from the same team can be heard only every other cycle.

Protocol

- Time slices, with integer labels indicate uniform numbers.

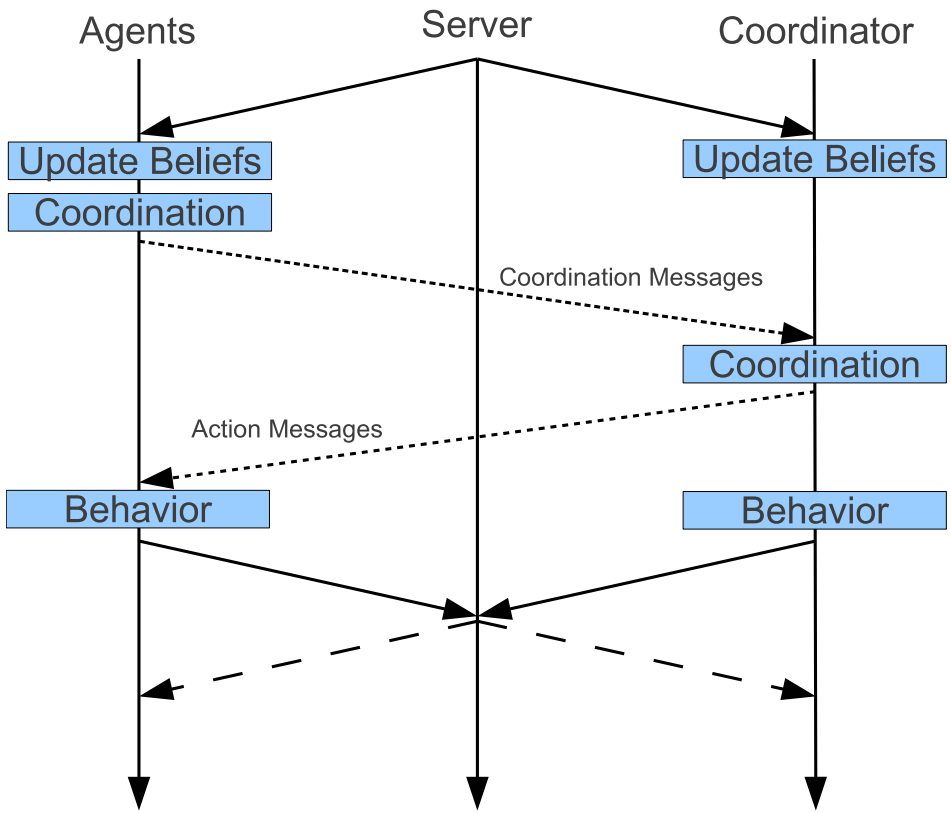


- The label starts at 1 and grows by 1 every time a player sends a message.
- Every player can "hear" all his teammates every 540ms (9 players) and 660ms (11 players).

Presentation Outline

Team Coordination

- Dynamic determination of individual behaviors for each agent.
- Executed only by one agent, the coordinator.
- All players communicate their beliefs to the coordinator.
- Coordinator sends back to the players the computed actions.



Coordination Phases

- Update Coordination Beliefs
- Determine Coordination Subsets
  - *Goalkeeper*: one player, the goalkeeper
  - *Inactive*: players fallen on the ground or players with lost self-location
  - *Active*: three players, the ones closest to the ball
  - *Support*: all remaining players
- Generate Team Formation
- Assign Team Roles
- Determine Active Positions
- Coordinate Active Players
- Determine Support Positions
- Coordinate Support Players

Coordination Modes

Coordination is not a static procedure and may change dynamically during different game states. There are three modes of team coordination:

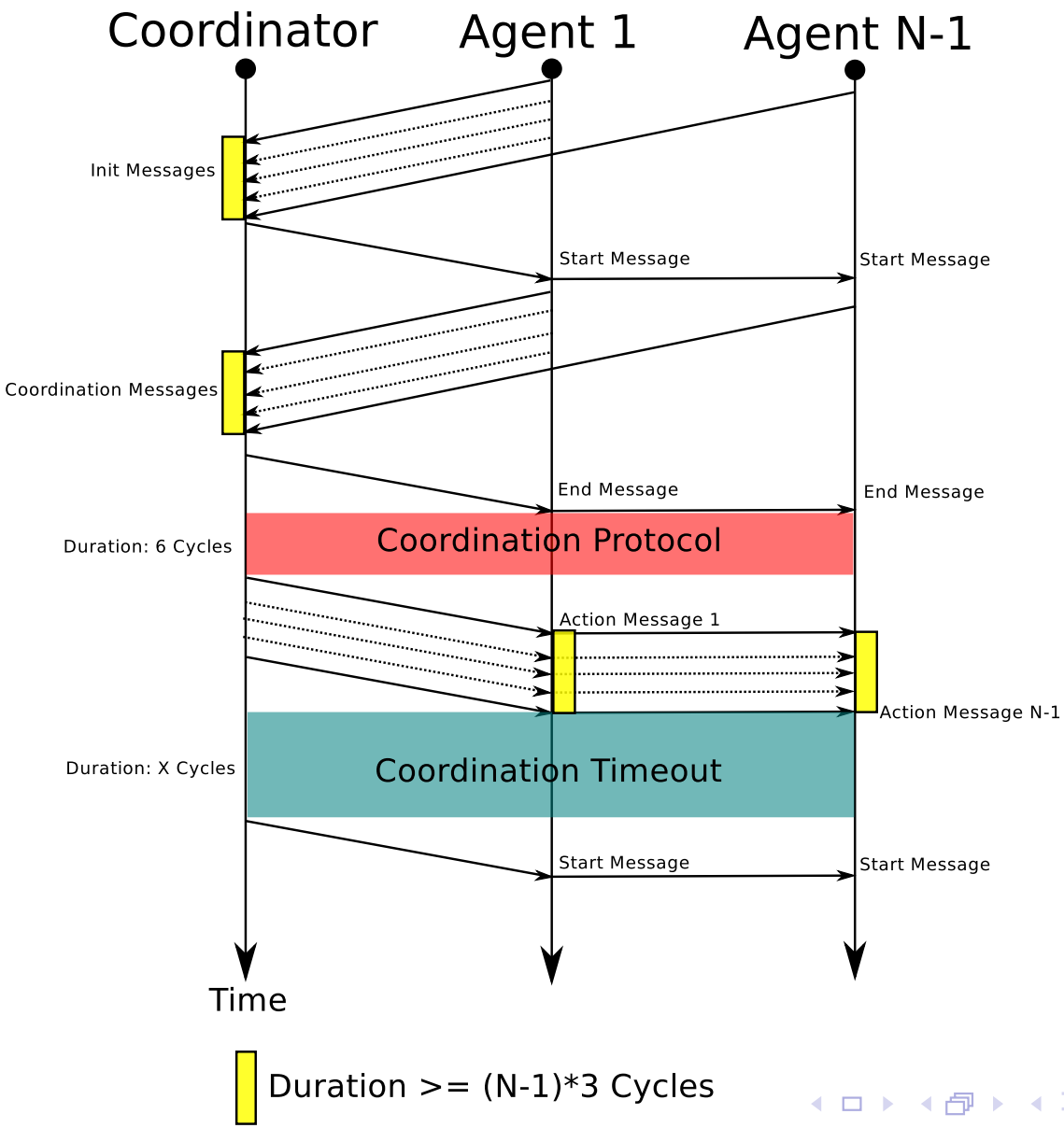
1. Active Mode
2. Support Mode
3. Wait Mode

Message Types and Communication

There are several types of messages:

- Init Message
- Start Message
- Coordination Message
  - Type C, position, ball position.
  - Type L, position.
  - Type B, ball position in relation to body.
  - Type X, empty messages.
- End Message
- Action Message

Messaging Protocol

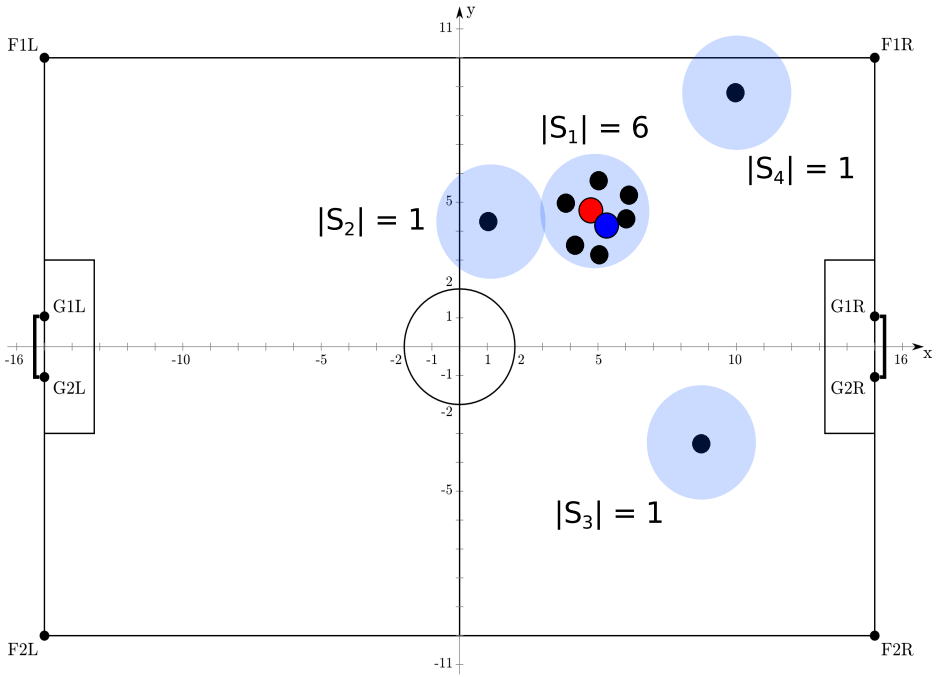


Coordination Beliefs

Coordination need to have a good knowledge about the game state.

- Global ball position
- Agents' distances from ball
- Agents' positions

Estimated Ball Position



- Ball position observations
- Real ball position
- Distance threshold
- Estimated ball position

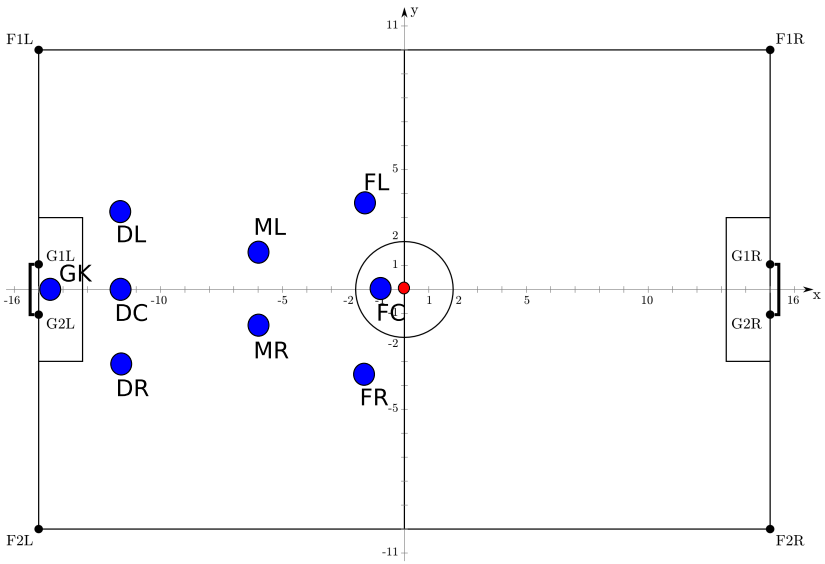
$$w(s_i) = |s_i|^3$$

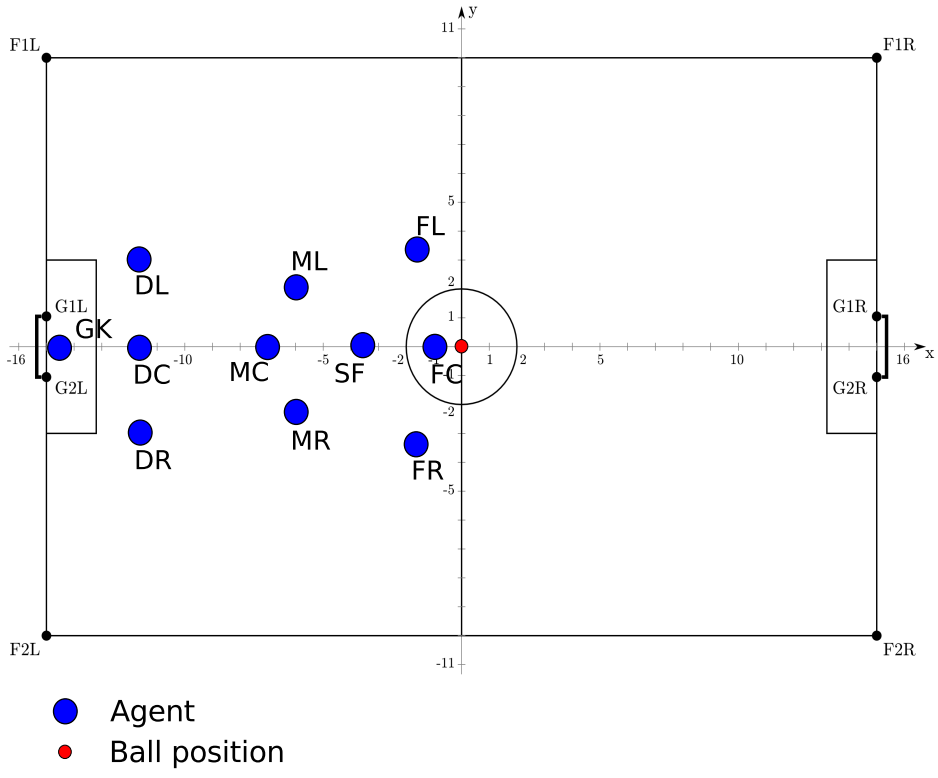
$$\textbf{GlobalBallBelief} = \sum_{i=1}^{m} \frac{w(s_i)}{\sum_{k=1}^{m} w(s_k)} \sum_{o_{ij} \in s_i} \frac{o_{ij}}{|s_i|}$$

Generation of Team Formation

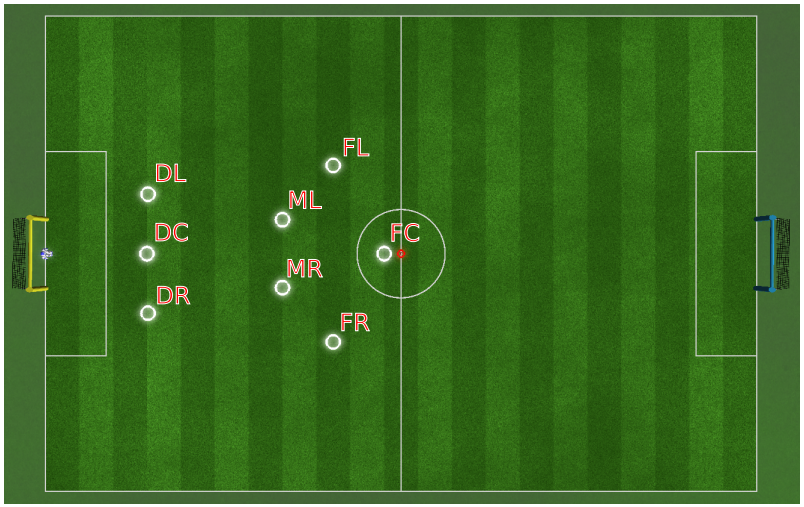
Team formation is dynamically generated based on the position of the ball.

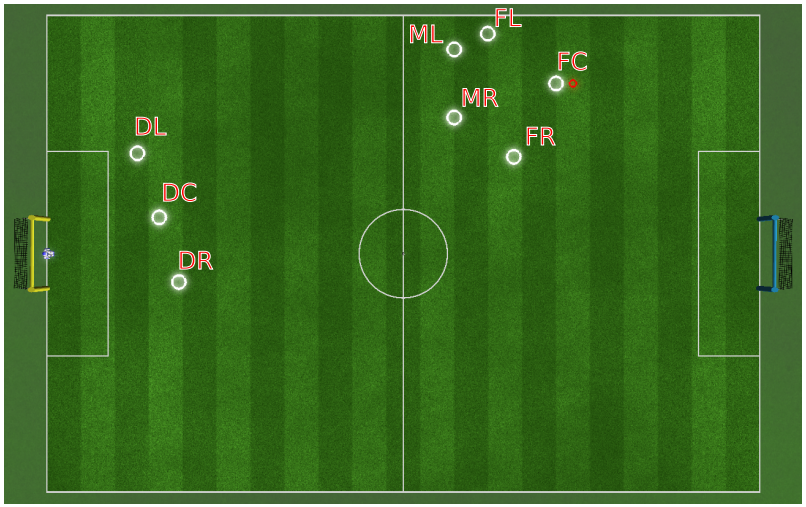
- 9 Players Version
- 11 Players Version

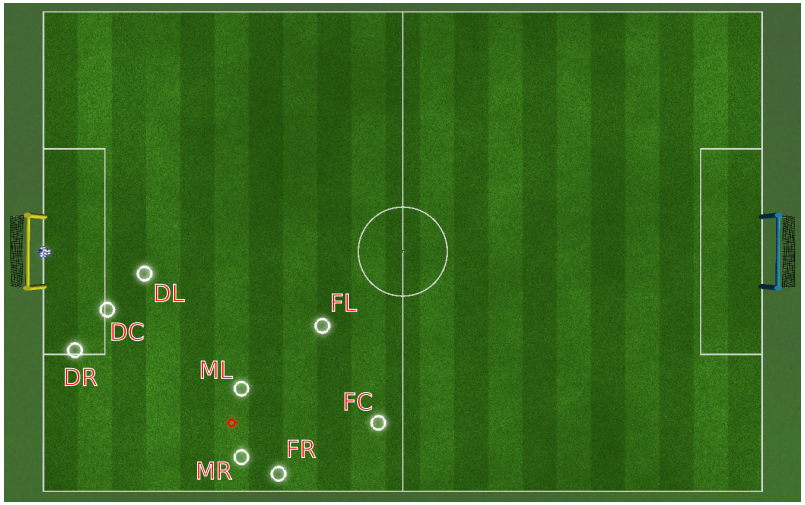




Team Formation Examples

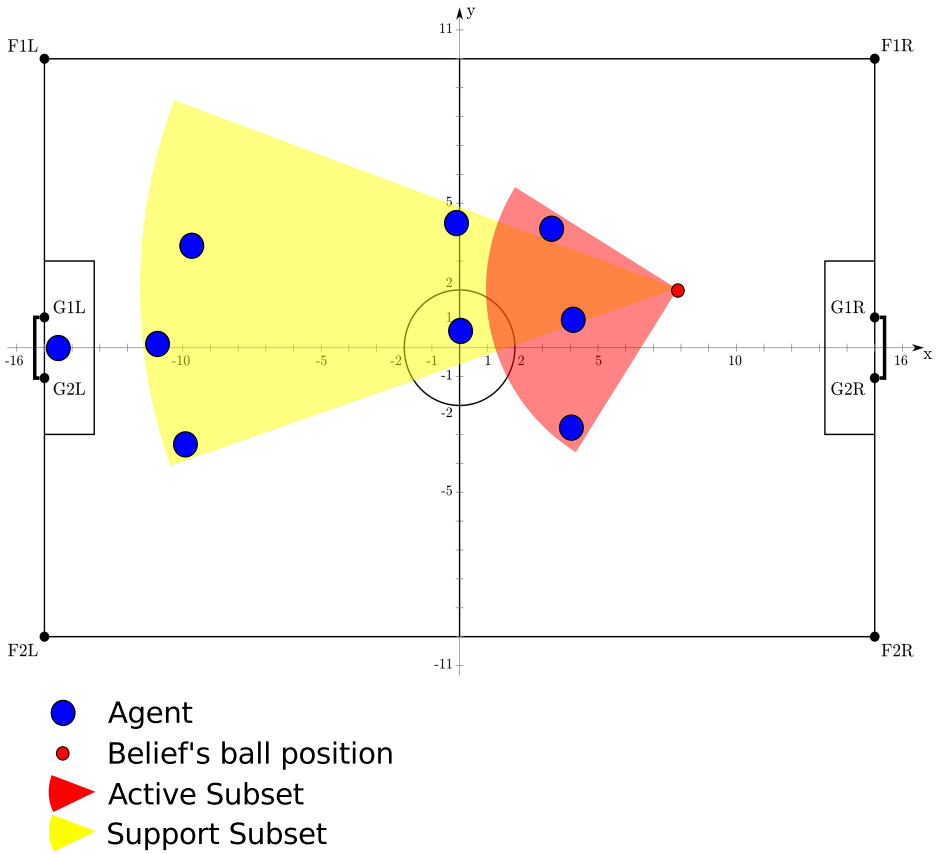






Coordination Splitter

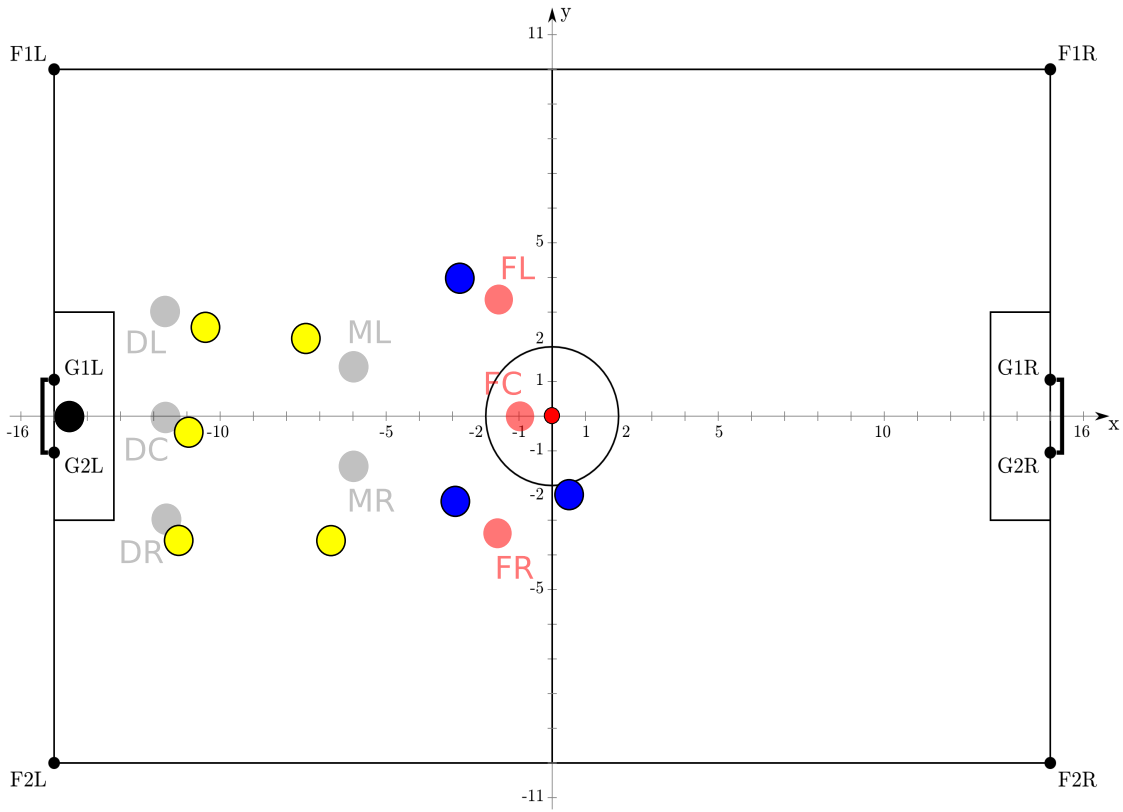
- Same number of positions and agents
- Computationally expensive problem
- Exhaustive algorithm (worst case): 40320 - 3628800 mappings



Team Roles Assignment

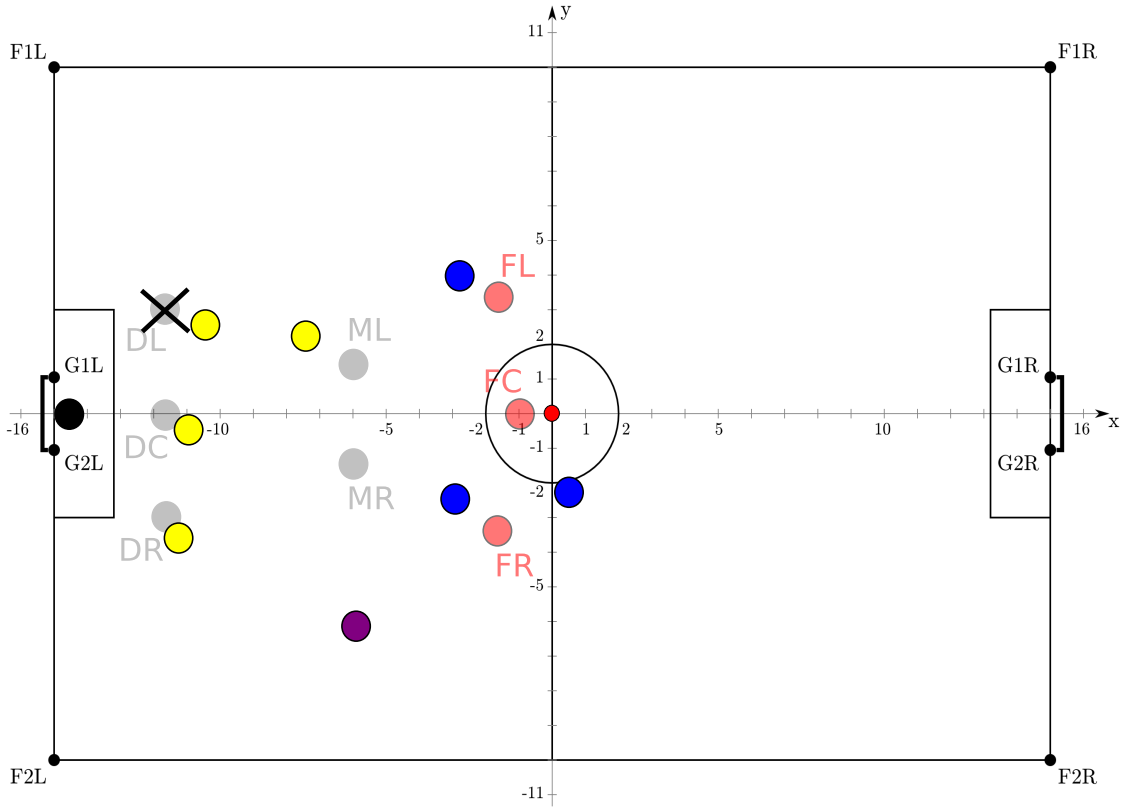
- Assigns roles to all agents.
- Roles close to ball, active players.
- Remaining roles will be assigned to support players.
- Inactive agents.
- Same number of roles will be discarded.

Team Roles Assignment Example 1



- Formation role positions
- Formation role positions near to ball
- Ball position
- Support player
- Active player

Team Roles Assignment Example 2



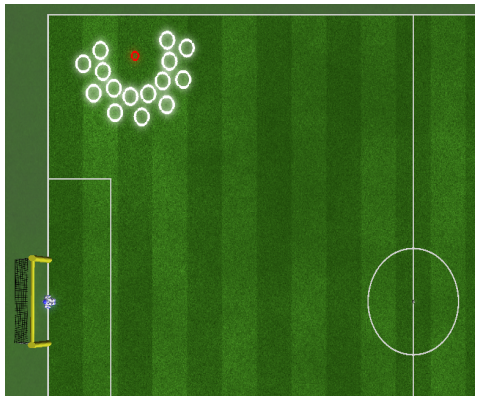
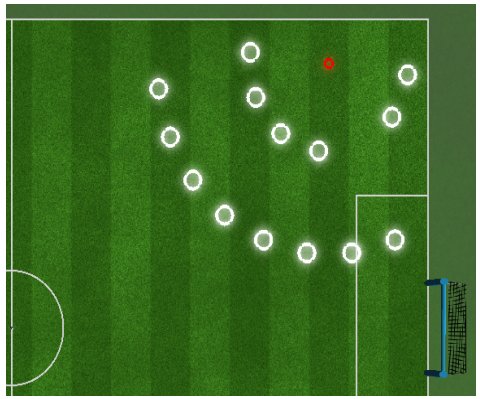
- Formation role positions
- Support player
- Active player
- Inactive player

Coordination

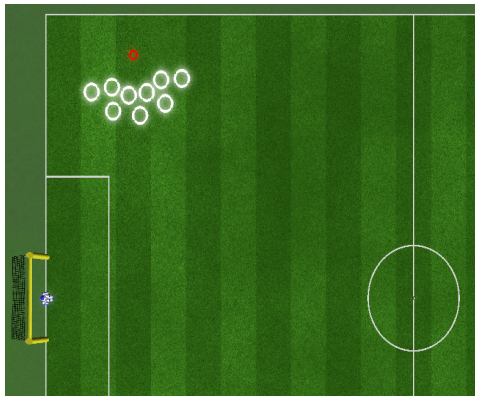
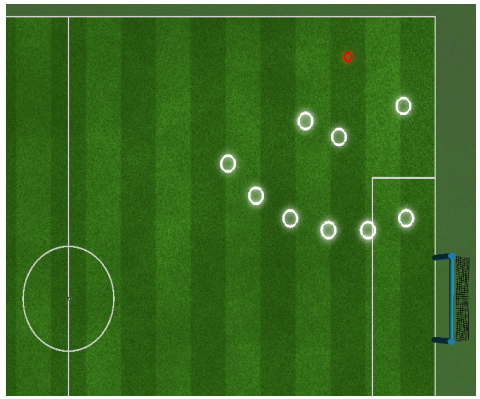
- Active Players (up to 3): exhaustive optimal algorithm
- Support Players (up to 8 or 10): dynamic programming algorithm
- Goalkeeper: own behavior
- Inactive Players: fixed predefined action

Determination of Active Positions

- Dynamic generation of active positions

- Position elimination using the field utility value
- Up to 9 candidate positions

Active Players Coordination

- On Ball player.
- Agent closest to the ball.
- Angle from goal is also taken into consideration.
- Set of ≤ 9 positions.
- Set of ≤ 2 agents.
- Exhaustive algorithm.
- Every combination of mappings is checked.
- Cost function to determine which of them is optimal.

Active Coordination Evaluation Function

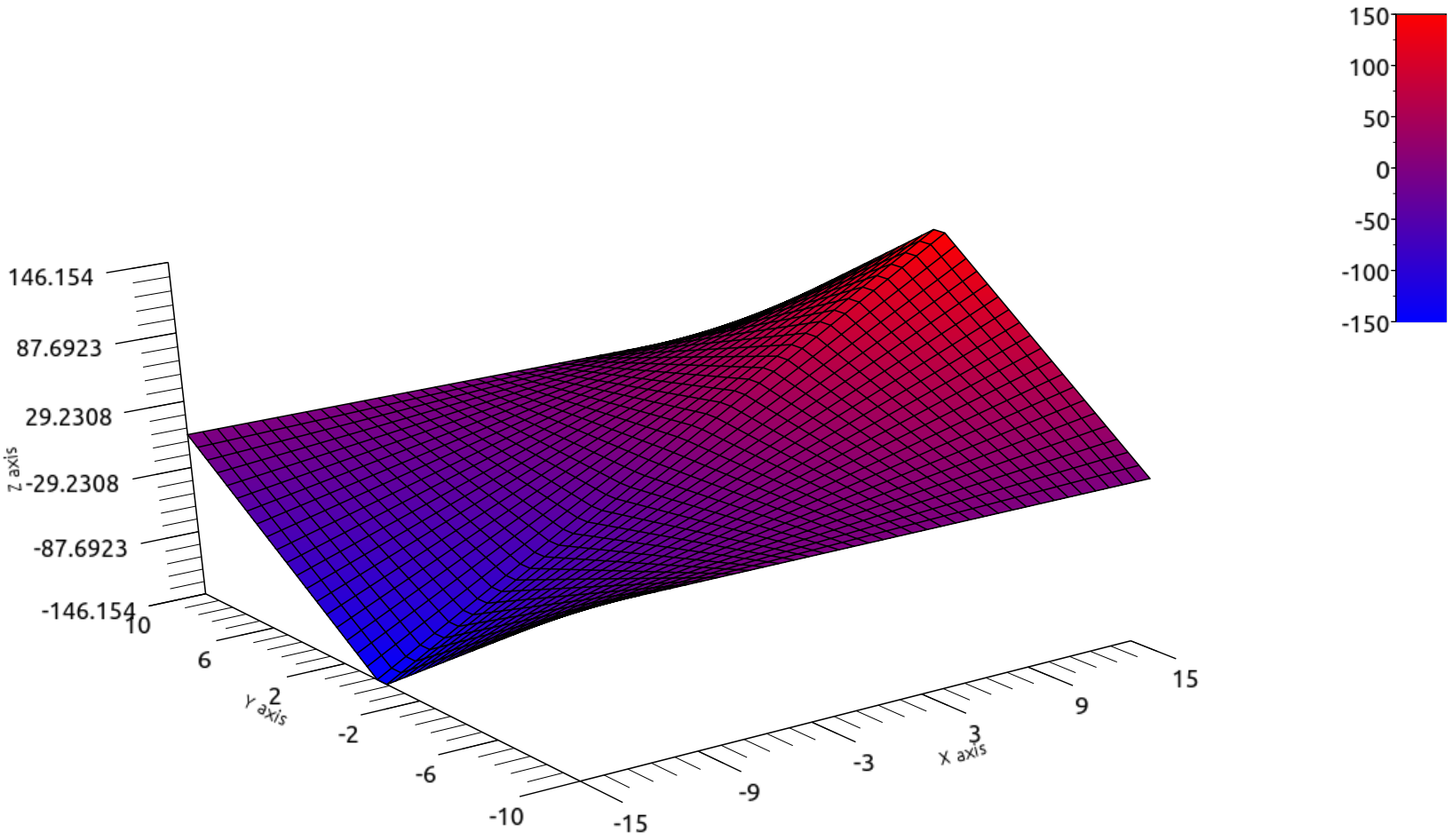
The evaluation function scores each possible mapping using the following features defined for each agent i:

1. **Distance** $C_{d,i}$
2. **Potential Collisions** $C_{c,i}$
3. **Field Utility** $C_{u,i}$
4. **Close Targets** $C_{t,i}$
5. **Horizontal Stretch** $C_{h,i}$

$$ActiveCost(ActiveMapping) = \sum_{i=1}^{3} w_d C_{d,i} + w_c C_{c,i} - w_u C_{u,i} - w_t C_{t,i} - w_h C_{h,i}$$

Where $(w_d, w_c, w_u, w_t, w_h)$ are the weights of the features, currently set at $(1, 1, 1/7, 1, 1)$.

Soccer Field Utility Fuction



Active Players Optimal Mapping

Algorithm 2 Active Players Optimal Mapping

Input: $ActivePlayers = ActiveSubset - OnBallPlayer = \{Agent_1, Agent_2\}$
Input: $ActivePositions = \{P_1, P_2, ..., P_N\}, N \leq 9$
Output: $OptimalActiveMapping$

$OptimalActiveMappingCost = +\infty$
for each $(P_i, P_j) \in ActivePositions \times ActivePositions, i \neq j$ **do**
  $ActiveMapping = \{Agent_1 \leftarrow P_i,\ Agent_2 \leftarrow P_j,\ OnBallPlayer \leftarrow Ball\}$
  $ActiveMappingCost = ActiveCost(ActiveMapping)$
  if $ActiveMappingCost < OptimalActiveMappingCost$ **then**
    $OptimalActiveMapping = ActiveMapping$
    $OptimalActiveMappingCost = ActiveMappingCost$
  end if
end for
return $OptimalActiveMapping$

Support Players Coordination

- Set of $\leq$ 8 or 10 positions.
- Set of $\leq$ 8 or 10 agents.
- Dynamic programming algorithm.
- A method proposed by the UT Austin Villa team.
- Able to compute an approximately optimal solution.
- A subset of mappings is checked.
- Cost function to determine which one is good.

Support Coordination Evaluation Function

The evaluation function scores each possible mapping using the following features defined for each agent i:

1. **Distance** $C_{d,i}$
2. **Potential Collisions** $C_{c,i}$

$$ActiveCost(ActiveMapping) = \sum_{i=1}^{3} w_d C_{d,i} + w_c C_{c,i}$$

Where (w_d, w_c) are the weights of the features, currently set at $(1, 1)$.

Support Players Mapping

Algorithm 3 Support Players Mapping

Input: $SupportPlayers = \{A_1, A_2, ..., A_n\}$, $SupportPositions = \{P_1, P_2, ..., P_n\}$
Input: $OAM = OptimalActiveMapping$
Output: $BestSupportMapping$

$BestSupportMapping[s] = \emptyset$, where $s \subseteq SupportPlayers$
$BestSupportMappingCost[s] = +\infty$, where $s \subseteq SupportPlayers$
for $k = 1 \rightarrow n$ **do**
  for each α in $SupportPlayers$ **do**
    $S = \binom{n-1}{k-1}$ sets of $k-1$ agents from $SupportPlayers - \{\alpha\}$
    for each s in S **do**
      $SupportMapping = \{\alpha \leftarrow P_k\} \cup BestSupportMapping[s]$
      $SupportMappingCost = SupportCost(SupportMapping, OAM)$
      if $SupportMappingCost < BestSupportMappingCost[\{\alpha\} \cup s]$ **then**
        $BestSupportMapping[\{\alpha\} \cup s] = SupportMapping$
        $BestSupportMappingCost[\{\alpha\} \cup s] = SupportMappingCost$
      end if
    end for
  end for
end for
return $BestSupportMapping[SupportPlayers]$

Support Coordination Example

$\{P_1\}$	$\{P_1, P_2\}$	$\{P_1, P_2, P_3\}$
$\{A_1 \leftarrow P_1\}$	$\{A_1 \leftarrow P_2\} \cup \arg\min(\{A_2\} \leftarrow \{P_1\})$	$\{A_1 \leftarrow P_3\} \cup \arg\min(\{A_2, A_3\} \leftarrow \{P_1, P_2\})$
$\{A_2 \leftarrow P_1\}$	$\{A_1 \leftarrow P_2\} \cup \arg\min(\{A_3\} \leftarrow \{P_1\})$	$\{A_2 \leftarrow P_3\} \cup \arg\min(\{A_1, A_3\} \leftarrow \{P_1, P_2\})$
$\{A_3 \leftarrow P_1\}$	$\{A_2 \leftarrow P_2\} \cup \arg\min(\{A_1\} \leftarrow \{P_1\})$	$\{A_3 \leftarrow P_3\} \cup \arg\min(\{A_1, A_2\} \leftarrow \{P_1, P_2\})$
	$\{A_2 \leftarrow P_2\} \cup \arg\min(\{A_3\} \leftarrow \{P_1\})$	
	$\{A_3 \leftarrow P_2\} \cup \arg\min(\{A_1\} \leftarrow \{P_1\})$	
	$\{A_3 \leftarrow P_2\} \cup \arg\min(\{A_2\} \leftarrow \{P_1\})$	

Dynamic Algorithm Complexity

- In kth iteration of the algorithm, each agent will be assigned to the P_k position.

Dynamic Algorithm Complexity

- In kth iteration of the algorithm, each agent will be assigned to the P_k position.
- The previous $k-1$ positions will be assigned to the other $n-1$ agents.

Dynamic Algorithm Complexity

- In kth iteration of the algorithm, each agent will be assigned to the P_k position.
- The previous $k-1$ positions will be assigned to the other $n-1$ agents.
- Result in a total of $\binom{n-1}{k-1}$ mappings to be evaluated in each iteration for each agent.

$$n\sum_{k=1}^{n}\binom{n-1}{k-1} = n\sum_{k=0}^{n-1}\binom{n-1}{k} = n2^{n-1}$$

Dynamic Algorithm Complexity

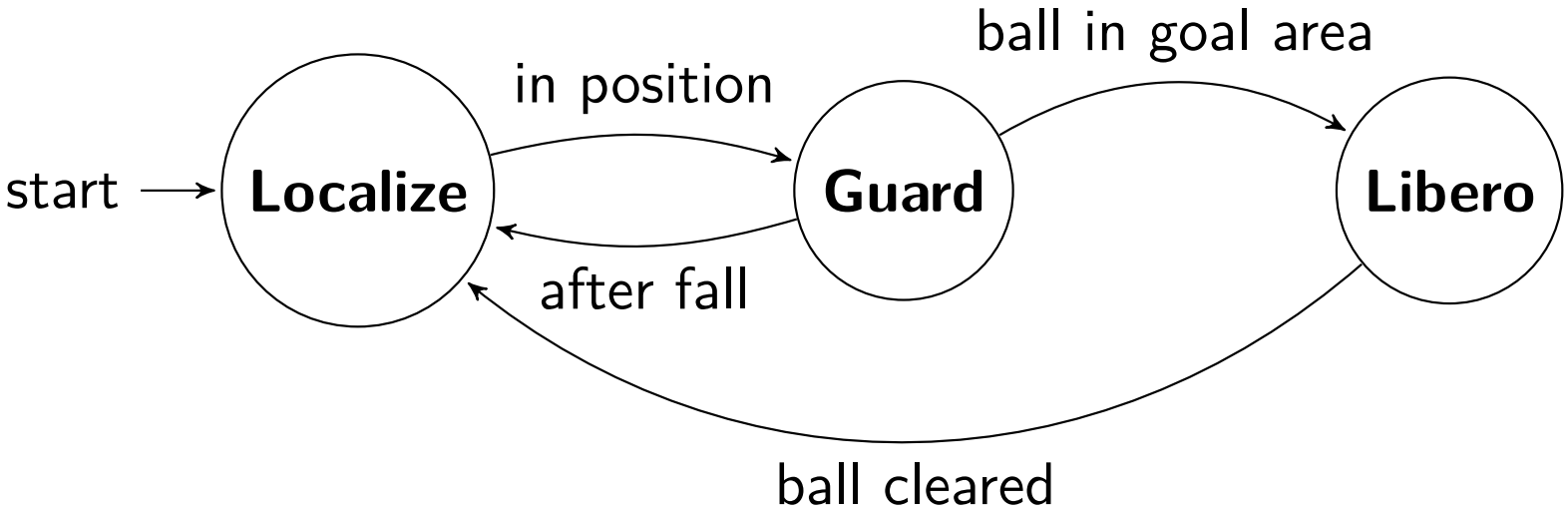
- In kth iteration of the algorithm, each agent will be assigned to the P_k position.
- The previous $k-1$ positions will be assigned to the other $n-1$ agents.
- Result in a total of $\binom{n-1}{k-1}$ mappings to be evaluated in each iteration for each agent.

$$n\sum_{k=1}^{n}\binom{n-1}{k-1} = n\sum_{k=0}^{n-1}\binom{n-1}{k} = n2^{n-1}$$

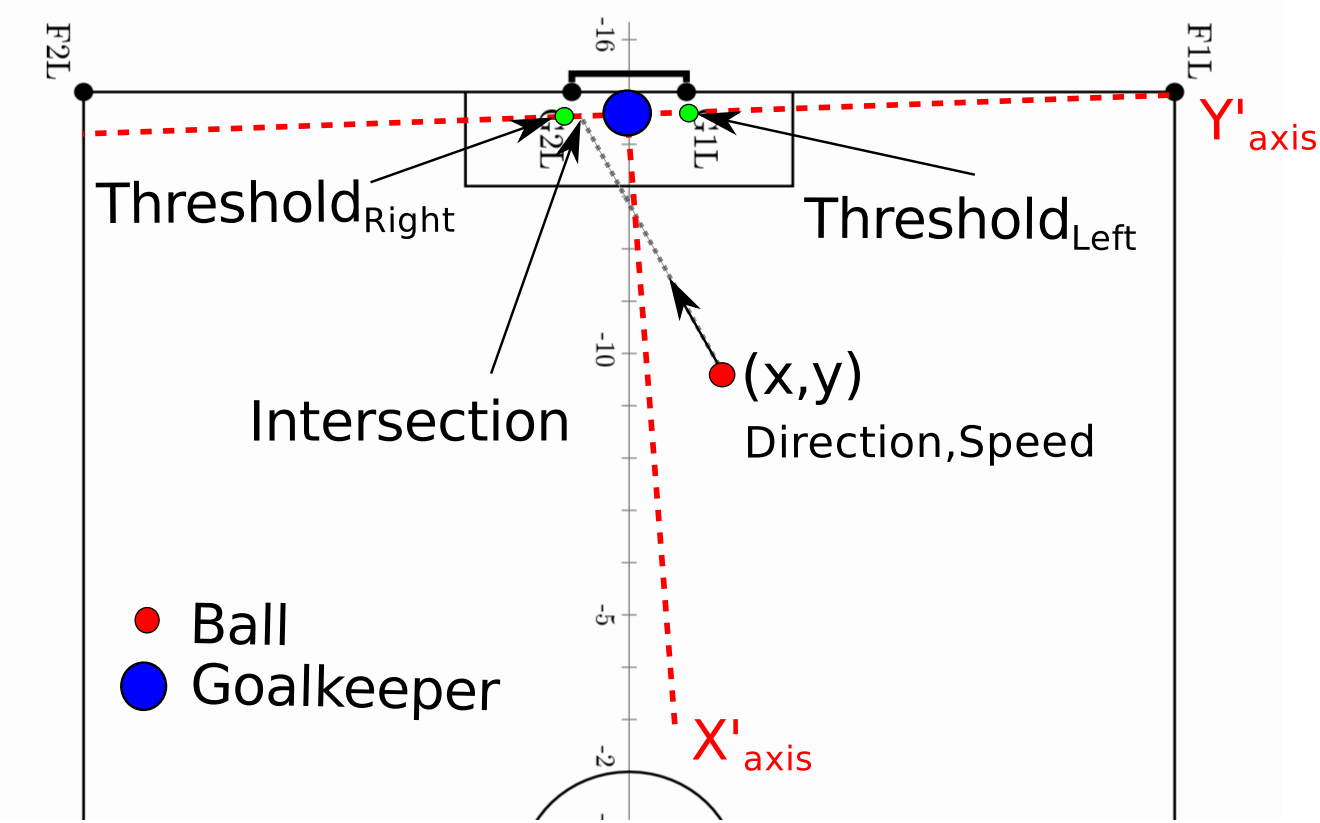
- This represents a reduction to 1024 and 5120 mappings for 8 and 10 agents/positions respectively compared to 40320 and 3628800 mappings of the exhaustive algorithm.

Goalkeeper

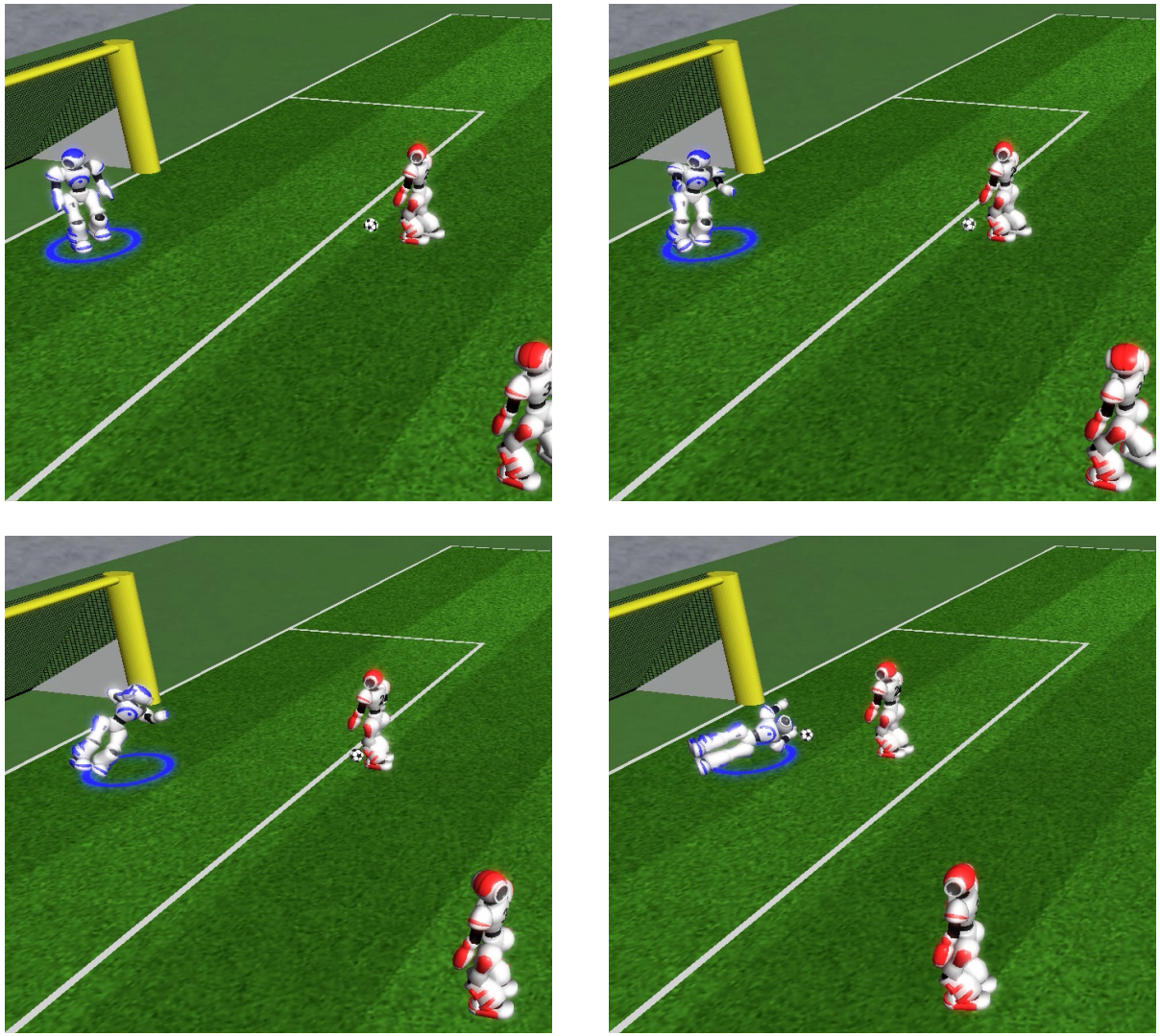
- The only agent in our team who "runs" his own behavior.
- His behavior depends on a finite state machine.



"Guard" State


F2L
F1L
-16
-10
-5
-2
G2L
G1L
Threshold Right
Threshold Left
Y' axis
(x,y)
Direction,Speed
Intersection
Ball
Goalkeeper
X' axis

Goalkeeper Fall Example



Presentation Outline

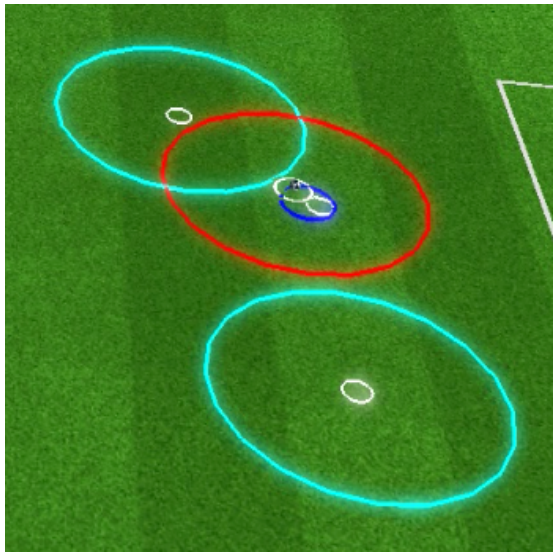
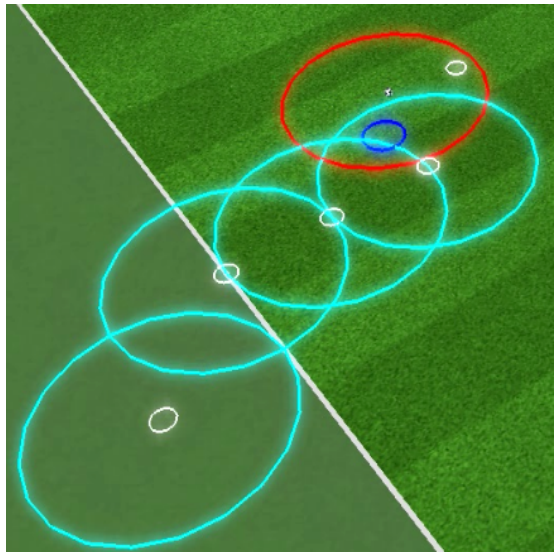
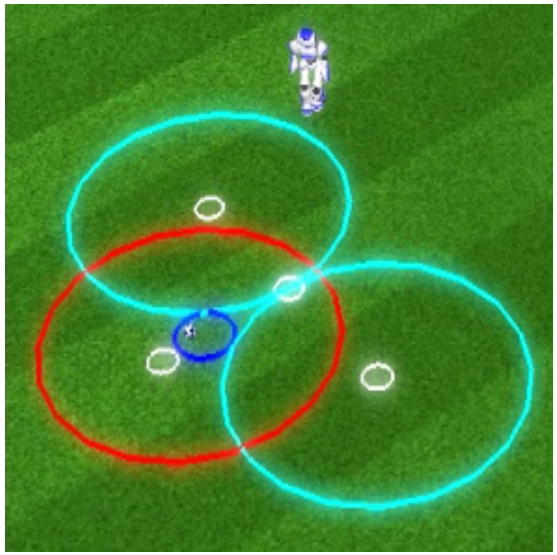
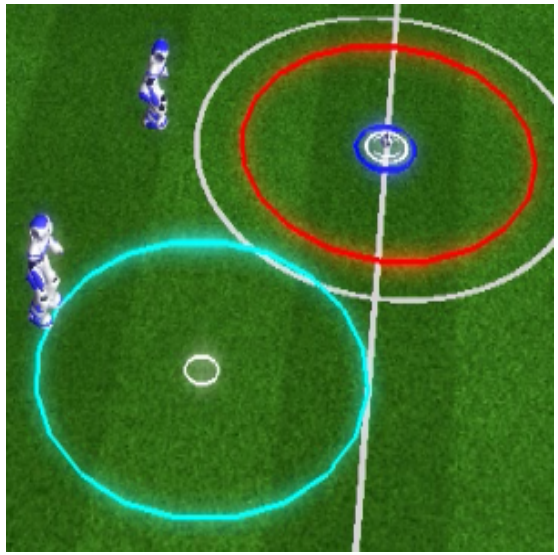
Motion Results and Improvements

Motion Version	Walk(m/s)	Turn(d/s)	Kick(m)	Strong Kick(m)
Webots (Text-Based)	0.11	21	3	-
FIIT (XML)	0.22	25	3 (4 Sec.)	4 (5 Sec.)
Kouretes3D	**0.45**	**30**	**3 (2.5 Sec.)**	**5.5 (2.5 Sec.)**

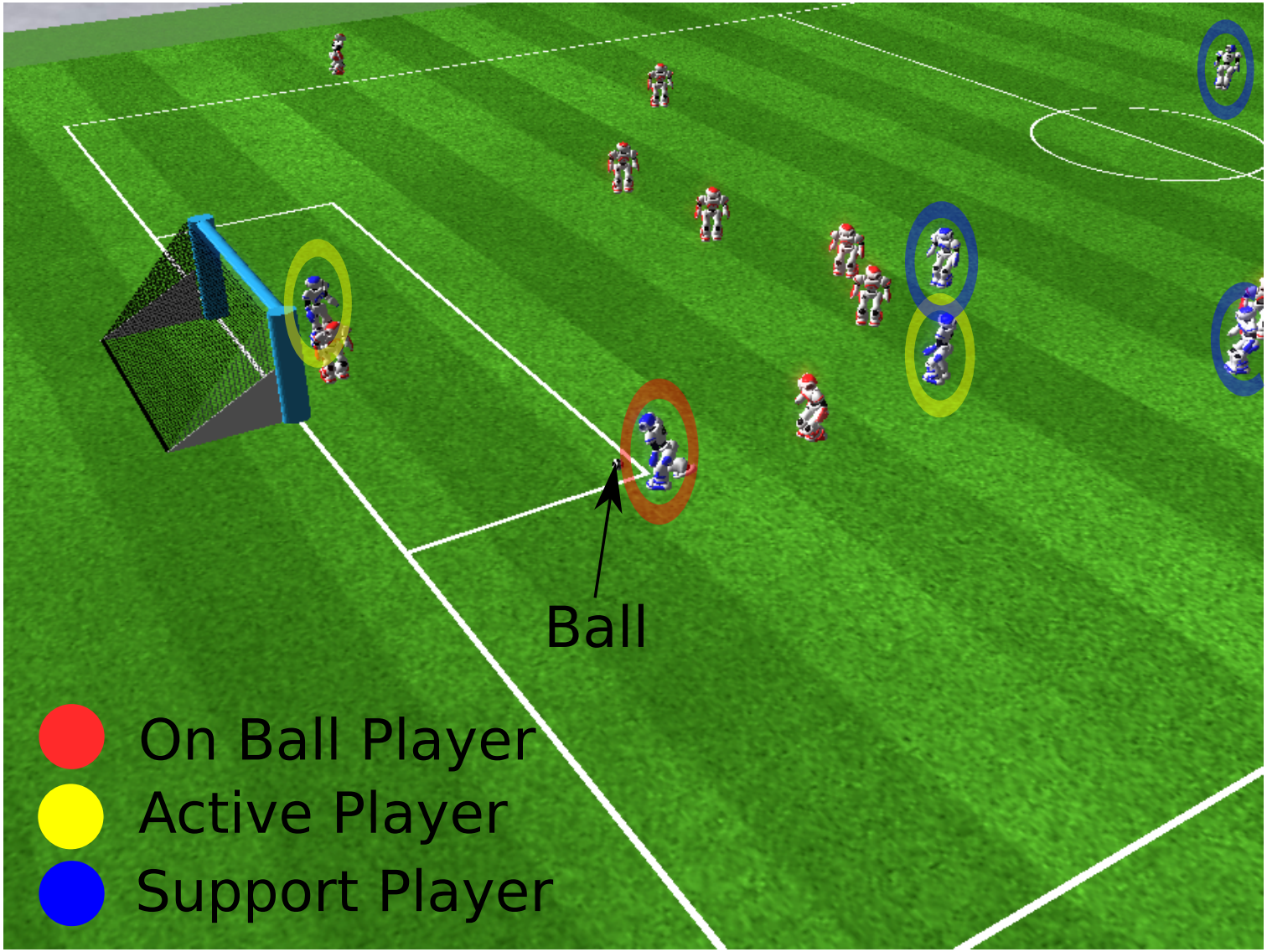
Communication Results

Communication Phase	Ideal (Cycles (Sec.))	During Match (Cycles (Sec.))
Init Messages	24 (0.48)	24 (0.48)
Coordination Messages	24 (0.48)	42.5 (0.85)
Action Messages	24 (0.48)	24 (0.48)

Coordination Beliefs (Global Ball Position)

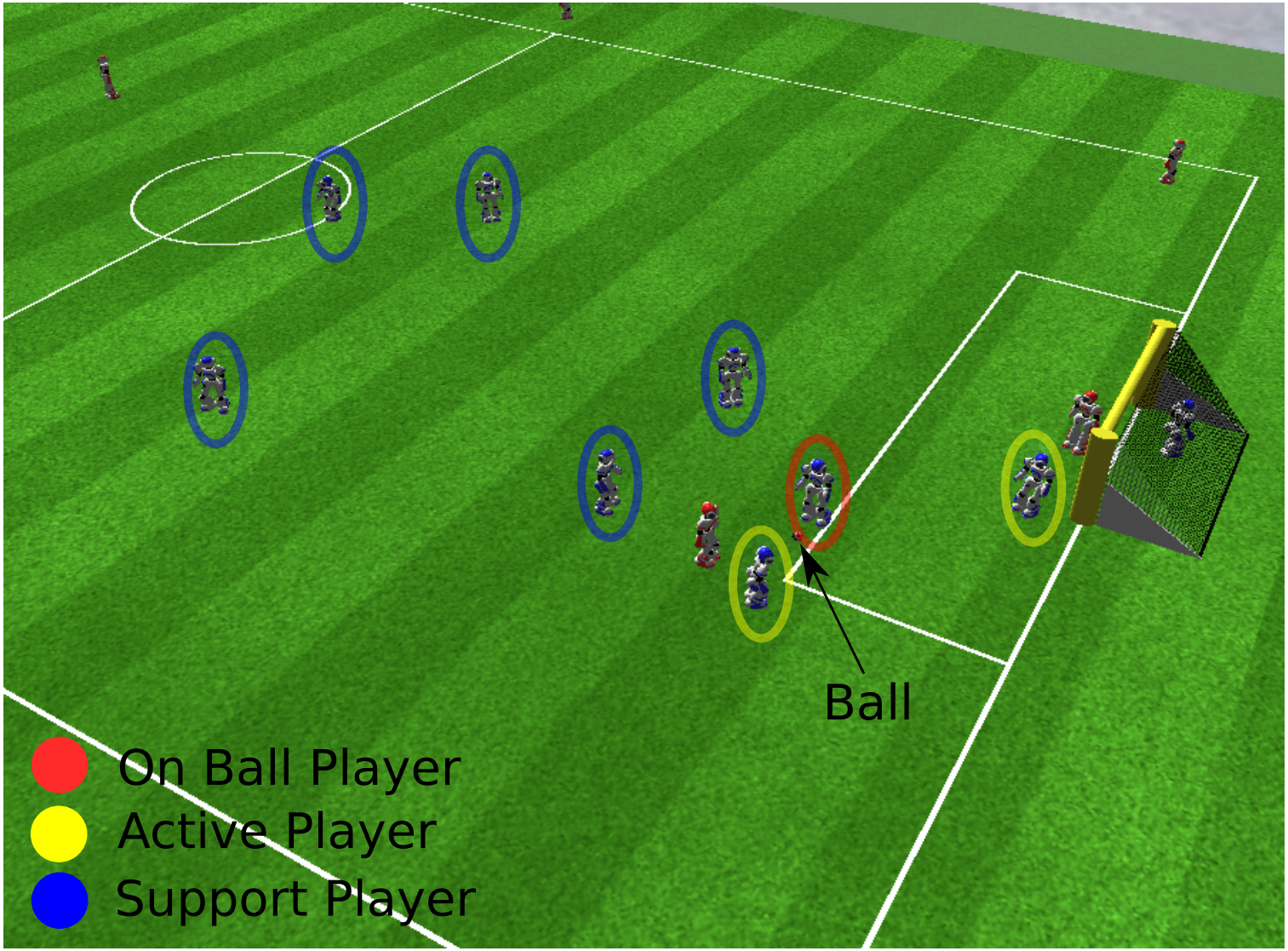





Offensive Positioning 1



Offensive Positioning 2

Defensive Positioning 1



Defensive Positioning 2

Formation Consistency

Goalkeeper Results

GoalKeeper Type	Goals Conceded
No Goalkeeper	~ 7
Goalkeeper with "Empty" Behavior	~ 7
Goalkeeper with "Full" Behavior	~ 3

Full-Games Results

Team	W	D	L	AGD	Games
MAK	2	0	0	+2.0	2
L3M-SIM	3	2	0	+0.6	5
FARZANEGAN	1	1	0	+0.5	2
NomoFC	1	2	0	+0.3	3
Rail	0	4	0	0.0	4
FUTK3D	0	5	0	0.0	5
OxBlue	0	0	2	-1.5	2
BeeStanbul	0	0	3	-4.0	3
UTAustinVilla	0	0	4	-5.2	4
Robocanes	0	0	1	-6.0	1

Full-Games Results

Team	P	W	D	L	F	A	Pts
Kouretes3D	**8**	**2**	**6**	**0**	**2**	**0**	**12**
Farzanegan	8	1	7	0	1	0	10
Rail	8	1	6	1	1	1	9
L3M-SIM	8	1	5	2	1	2	8
NomoFC	8	1	4	3	1	3	7

Presentation Outline

Future Work

- Probabilistic localization system
- Dynamic omni-directional locomotiom
- Passing
- Participation in Robocup

Finally...

Ευχαριστώ πολύ!