Player Behavior and Team Strategy in Robocup Simulation 3D League

Georgios Methenitis

Technical University of Crete

Chania, August, 2012

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

Abstract

In this thesis we present...

Outline of Topics

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 Leagues

Soccer Popular, well-known Rules, cooperative multi-agent systems in dynamic adversarial environments.

Rescue Prompt support for planning disaster mitigation, search and rescue.

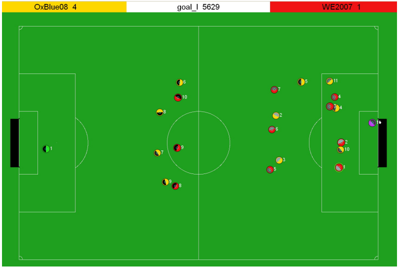
@Home Aims to develop service and assistive robot technology with high relevance for future personal domestic applications.

Junior It is designed to introduce RoboCup to primary and secondary school children

Soccer Simulation League

One of the oldest leagues in RoboCup's Soccer. The Simulation League focus on artificial intelligence and team strategy. Independently moving software players (agents) play soccer on a virtual field inside a computer. There are two sub-leagues: 2D and 3D.

2D vs 3D



3D Simulation Soccer

- At its beginning, the only available robot model was a spherical agent.
- In 2006, a simple model of the Fujitsu HOAP-2 robot was made available, being the first time that humanoid models were used in the simulation league.
- In 2008, the introduction of a Nao robot model to the simulation gave another perspective to the league.
- **SimSpark** is used as the official Robocup 3D simulator.

SimSpark

- **SimSpark** is a generic physics simulator system for multiple agents in three-dimensional environments.
- *Rcssserver3d* is the official competition environment for the RoboCup 3D Simulation League. It implements a simulated soccer environment, whereby two teams of up to nine, and in the latest version up to eleven, humanoid robots play against each other.

Server's Versions

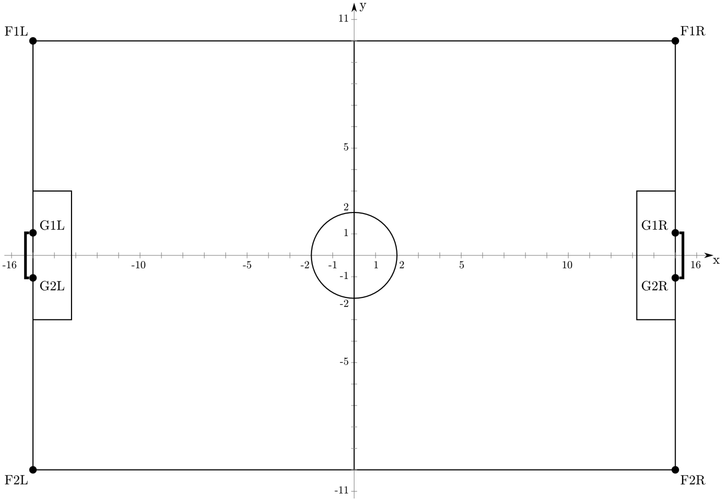
- Version 0.6.5

Players	9
Length	21m
Width	14m

- Version 0.6.6

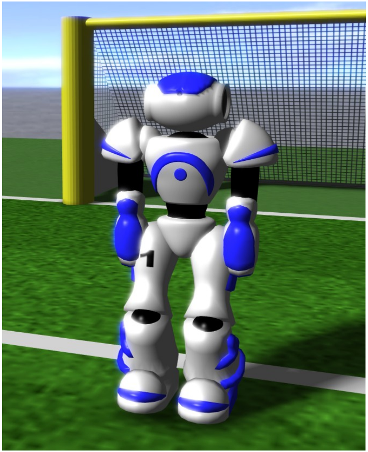
Players	11
Length	30m
Width	20m

Soccer Field



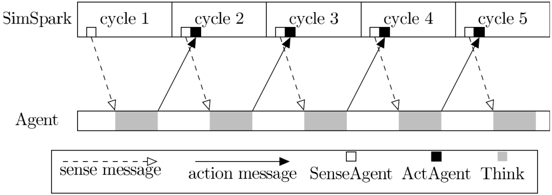
Robot Model

The Nao humanoid robot manufactured by Aldebaran Robotics. Its height is about 57cm and its weight is around 4.5kg. The simulated model comes with 22 degrees of freedom.



Server

The SimSpark server hosts the process that manages and advances the simulation. The simulation state is constantly modified through the simulation update loop. Each simulation step corresponds to 20ms of simulated time.



Monitor

- SimSpark Monitor
  - Responsible for rendering the current simulation
  - Default SimSpark monitor
- RoboViz Monitor
  - Designed to assess and debug agent behaviors in the RoboCup 3D Simulation League
  - Allow drawing
  - Official monitor of the RoboCup 3D simulation Soccer League

Monitors Comparison

Agent Perceptors I

Perceptors are the senses of an agent, allowing awareness of the agent's model state and the environment.

HingeJoint Perceptor A hinge joint perceptor receives information about the angle of the corresponding single-axis hinge joint.

- **Message format:** `(HJ (n <name>) (ax <ax>))`
- **Frequency:** Every cycle

ForceResistance Perceptor This perceptor informs about the force that acts on a body.

- **Message format:** `(FRP (n <name>) (c <px> <py> <pz>) (f <fx> <fy> <fz>))`
- **Frequency:** Only in cycles where a body collision occurs

GyroRate Perceptor The gyro-rate perceptor delivers information about the change in orientation of a body.

- **Message format:** `(GYR (n <name>) (rt <x> <y> <z>))`

Agent Perceptors II

Frequency: Every cycle

Accelerometer Perceptor This perceptor measures the proper acceleration a body ex- periences relative to free fall.

Message format: `(ACC (n <name>) (a <x> <y> <z>))`

Frequency: Every cycle

Vision Perceptor Vision perceptor delivers information about seen objects in the environment, where objects are either others players, the ball, field lines, or markers on the field.

Message format:
```
(See +(<name> (pol <d> <a1>
<a2>))
+(P (team <name>) (id <ID>)
+(<bodypart> (pol <d> <a1> <a2>)))
+(L (pol <d> <a1> <a2>)(pol <d> <a1>
<a2>)) )
```

Frequency: Every third cycle (60ms)

Agent Perceptors III

Hear Perceptor Hear perceptor serves as an aural sensor and receives messages shouted by other players.

Message format: `(hear <time> self/<direction> <message>)`

Frequency: Only in cycles, where a message is heard

GameState Perceptor The game state perceptor delivers information about the ac- tual state of the soccer game environment.

Message format: `(GS (t <time>) (pm <playmode>))`

Frequency: Every cycle

Agent Effectors I

Effectors allow agents to perform actions within the simulation. Agents control them by sending messages to the server and the server changes the game state accordingly.

Create Effector An agent uses this effector to advice the server to construct the physical representation and all further effectors and perceptors of the agent in the simulation environment according to a scene description file it passes as a parameter.

Message format: `(scene <filename>)`

Frequency: Only once

HingeJoint Effector Effector for all axes with a single degree of freedom.

Message format: `(<name> <ax>)`

Frequency: Once per cycle maximum

Agent Effectors II

Synchronize Effector Agents running in Agent Sync Mode must send this command at the end of each simulation cycle.

Message format: `(syn)`
Frequency: Every cycle

Init Effector The init effector registers the agent as a member of a team with a specific player number.

Message format:
```
(init (unum <playernumber>)
      (teamname <teamname>) )
```
Frequency: Only once

Beam Effector The beam effector allows a player to position itself anywhere on the field only before any kick-off.

Message format: `(beam <x> <y> <rot>)`
Frequency: Once before each kick-off

Agent Effectors III

Say Effector: The say effector permits communication among agents by broadcasting messages in plain ASCII text (20 characters maximum).

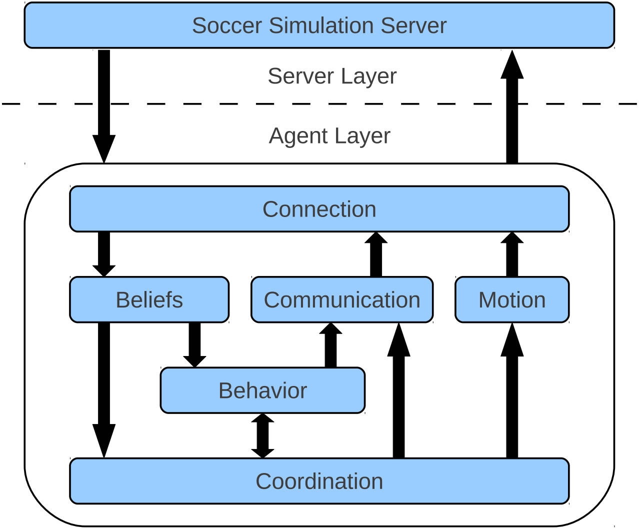
Message format: `(say <message>)`

Frequency: Once per cycle maximum

Architecture

- Connection with Server
- Update Beliefs and Sensors' Data
- Localization Process
- Locomotion
- Actions
- Agent Behavior
- Communication
- Team Coordination

Architecture



Connection

- Agent has to be connected to the server at all times during a simulated game.

Connection

- Agent has to be connected to the server at all times during a simulated game.
- Agent receives sense messages from the server every 20ms at the beginning of each simulation cycle.

Connection

- Agent has to be connected to the server at all times during a simulated game.
- Agent receives sense messages from the server every 20ms at the beginning of each simulation cycle.
- Agents willing to send action messages, can do so at the end of their think cycles, which may or may not coincide with the simulation cycles.

Perception

- Perceptions in simulated soccer are quite different compared to those in real soccer games.

Perception

- Perceptions in simulated soccer are quite different compared to those in real soccer games.
- Agents do not have to process raw data coming directly from sensors, but rather listen to sensor and higher-level observation messages sent by the server at each cycle.

Perception

- Perceptions in simulated soccer are quite different compared to those in real soccer games.
- Agents do not have to process raw data coming directly from sensors, but rather listen to sensor and higher-level observation messages sent by the server at each cycle.
- Agents update their beliefs and store sensors' data parsing these messages.

Sense Message Example

```
(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

- Localization is executed every three cycles (60ms).

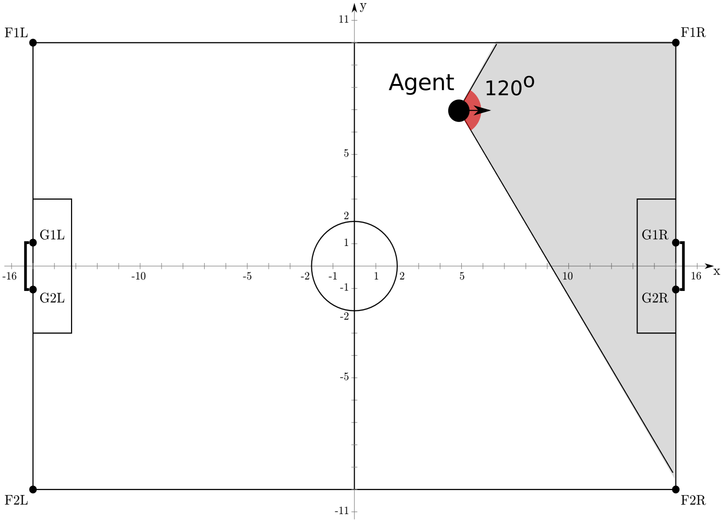
Self-Localization

- Localization is executed every three cycles (60ms).
- Localization uses the eight visible landmarks into the field.
  - G1R, G2R
  - G1L, G2L
  - F1R, F2R
  - F1L, F2L

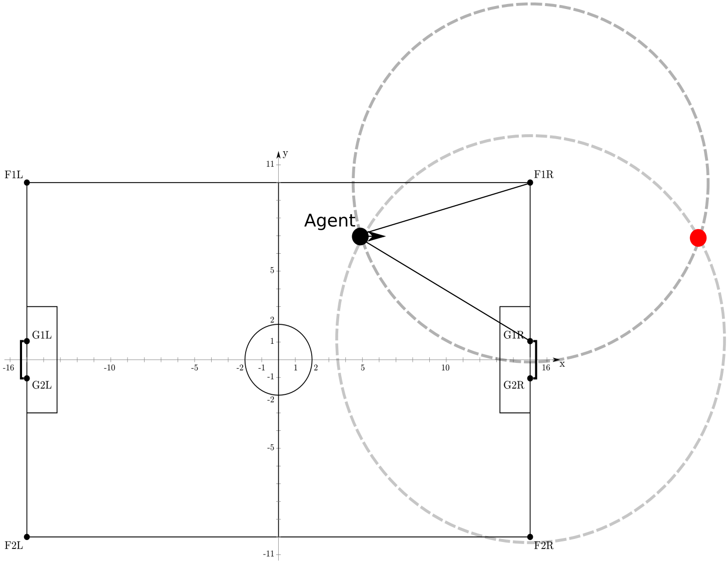
Self-Localization

- Localization is executed every three cycles (60ms).
- Localization uses the eight visible landmarks into the field.
  - G1R, G2R
  - G1L, G2L
  - F1R, F2R
  - F1L, F2L
- A key restrictive factor is that the agents are equipped with a restricted vision perceptor which limits the field of their view to 120 degrees.

Simulated Nao's Field of View



Self-Localization Technique



Object Localization

- Computing the position of other visible objects.

Object Localization

- Computing the position of other visible objects.
  - Other Players
  - Ball

Object Localization

- Computing the position of other visible objects.
  - Other Players
  - Ball
- Information about Visible Objects

Object Localization

- Computing the position of other visible objects.
  - Other Players
  - Ball
- Information about Visible Objects
  - Distance
  - Horizontal, Vertical Angles

Object Localization

- Computing the position of other visible objects.
  - Other Players
  - Ball
- Information about Visible Objects
  - Distance
  - Horizontal, Vertical Angles
- Enough information to compute those objects' positions if we know our exact position.

Localization Filtering

- Absence of a more sophisticated probabilistic localization scheme.

Localization Filtering

- Absence of a more sophisticated probabilistic localization scheme.
- Temporary absences of landmarks.

Localization Filtering

- Absence of a more sophisticated probabilistic localization scheme.
- Temporary absences of landmarks.
- Noisy observations.

Localization Filtering Algorithm

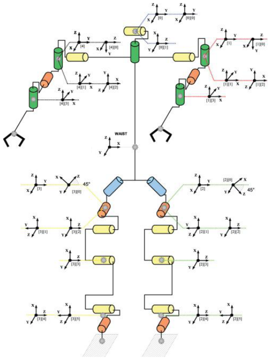
Algorithm 1 Localization Filtering

Input: *LastEstimate*
Output: *FilteredLocation*
Queue: a FIFO queue storing the *MaxSize* (default=10) most recent estimates

if $size(Queue) = 0$ **then**
    $Queue.Add(LastEstimate)$
else if $LastEstimate \not\approx AverageLocation(Queue)$ **then**
    $Queue.Remove()$
else
    if $size(Queue) = MaxSize$ **then**
        $Queue.Remove()$
    end if
    $Queue.Add(LastEstimate)$
end if
return $AverageLocation(Queue)$

Nao’s Anatomy

The simulated Nao robot comes with 22 degrees of freedom, corresponding to 22 hinge joints. Figure 4.6 shows Naoćs anatomy with all joints, split in five kinematic chains (head, left arm, right arm, left leg, right leg).



Motion and Movement

In robotics, a complex motion is commonly defined as a sequence of timed joint poses. A pose is a set of values for every joint in the robot's body or in a specific kinematic chain at a given time. For any given set of n joints a pose at time t is defined as:

$$Pose(t) = \{J_1(t), J_2(t), \ldots, J_n(t)\}$$

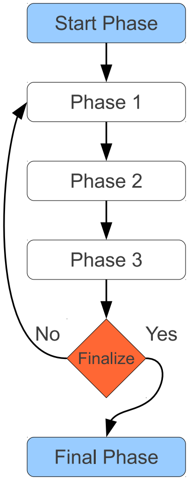
XML-Based Motion Files

```
<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

To generate motions for our agent we need to create a motion string, which encloses information about each joint's velocity. This velocity is computed as follows:

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

A velocity value is calculated for each joint involved in the motion and the final output of the motion controller is sent to the server. In addition, zero velocity is set for every joint not included in the `effector` field of each phase, so that they stop moving.

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,...
00:00:160,Pose5,0,-0.004,-0.525,1.05,-0.525,0.004,0,...
00:00:200,Pose6,0,0.001,-0.525,1.051,-0.525,-0.001,0,...
00:00:240,Pose7,0,0.006,-0.525,1.05,-0.525,-0.006,0,...
00:00:280,Pose8,0,0.012,-0.525,1.05,-0.525,-0.012,0,...
00:00:320,Pose9,0,0.024,-0.525,1.05,-0.525,-0.024,0,...
```

Text-Based Motion Controller

The motion controller could be customized easily to perform these motions in different ways. The following parameters can be modified:

Duration The time between poses in simulation cycles. By default, $Duration = 2$.

PoseStep The step for advancing from pose to pose. By default, $PoseStep = 1$, but we can subsample the motion with other values, e.g. for $PoseStep = 2$, we execute pose1, pose3, pose5, ...

The desired velocity of each joint is computed by:

$$JointVelocity = \frac{DesiredJointValue - CurrentJointValue}{Duration \times CycleDuration}$$

A velocity value is calculated for each joint involved in the motion and the final output of the motion controller is sent to the server.

Dynamic Motion Elements

Walk Leaning The XML-based walk motion can be dynamically modified to lean to the right or to the left. This is accomplished by altering the joint values of the (left or right) HipPitch and AnkePitch joints in specific phases of the walk motion.

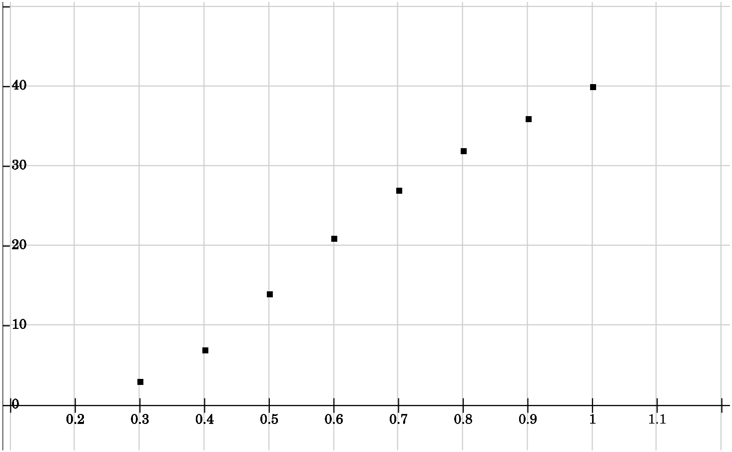
Walk Slowdown Increasing the phase durations dynamically by about 35% yields a smooth approach to a stopping position.

Dynamic Turn The text-based turn motion can be dynamically modified using a gain value for scaling the resulting velocities in order to perform the motion in a smoother or rougher way. By dynamically changing this value between 0.3 and 1.0, the agent is able to turn its body anywhere between 3 and 40 degrees.

Dynamic Turn Example

X-Axis Gain factor

Y-Axis Agent turn



Actions

Actions are split into groups in terms of their complexity:

Basic Actions Basic actions combine perceptual information and motion files in simple ways to achieve something useful.

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

Simple Actions I

Look Straight Straight Moves the head to its nominal position. Both head joints are set to 0.

Scan Moves the head to perform periodic panning and tilting.

Pan Head Moves the head to perform periodic panning at zero tilt.

Track Object Moves the head to bring a particular object to the center of the field of view. This action is applicable only when the object being tracked is visible, but is limited by the joint ranges.

Simple Actions II

Track Moving Object This action estimates the direction and the speed of a moving object using a small number of observations, obtained while performing the Track Object action. It records a set of five consecutive observations and another set of five consecutive observations delayed by a fixed time period (the default is 5 cycles). The difference between the average positions of each set gives a vector that reveals the direction of motion. Taking the ratio of the magnitude of this vector and the time delay yields the speed of the moving object.

Find Opponentćs Goals This action estimates the direction of the opponent's goal with respect to the agent by performing the Scan action.

Look For Ball Turns the body of the agent, while performing the Scan action, until the ball appears within the field of view.

Simple Actions III

Turn To Ball Turns the body of the agent towards the direction of the ball, while performing the Track Ball action. It can applied only when a ball is visible.

Turn To Localize Turns the body of the agent, while performing the Pan Head action, until the agent's belief about its own location is updated with confidence.

Stand Up Makes the agent stand up on its feet, after a confirmed fall on the ground, whether face-up or face-down. This action monitors the inertial sensors (accelerometers and gyroscopes) to check if our agent has fallen on the ground. Incoming gyroscope and accelerometer values above a specific threshold indicate a possible fall, but this has to be confirmed, because it is not unusual to receive values above threshold due to collisions without a fall. To confirm a fall, the action checks the force resistance perceptors located

Simple Actions IV

under the agent's feet. If these perceptors imply that the legs do not touch the ground, then we are pretty sure that a fall has occurred. In this case, a stand up motion is executed. Foot pressure values are also used to determine whether the stand up motion succeeded or not. The stand up motion is repeated, until it succeeds.

Prepare for Kick Positions the agent to an appropriate position with respect to the ball in order to perform a kick successfully.

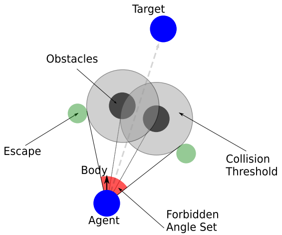
Complex Actions I

Avoid Obstacles Helps agent to avoid possible obstacles.

- Simulated Nao's head can pan from -120° to $+120^\circ$ and the field of view is 120°, we can obtain a complete imaging of all obstacles located close to our agent.
- For each recorded obstacle, we calculate two escape angles that determine the two directions which guarantee avoidance of the obstacle at a safe distance.
- Any escape angle of some obstacle that falls within the forbidden area of some other obstacle is discarded.
- The remaining escape angles, and particularly the escape way points they define (the points closest to the obstacle along the direction of the escape angles), are evaluated in terms of the angle and distance overhead they incur with respect to the agent orientation (for the angle) and the target (for the distance).

Complex Actions II

- The way point that minimizes the total overhead is selected as a temporary target for avoiding the obstacles, while making progress towards the target.



Complex Actions III

Algorithm 2 Escape Angle Set Calculation

Input: $Obstacles = \{O_1, O_2, \ldots, O_n\}$
Output: $EscapeAngleSet$

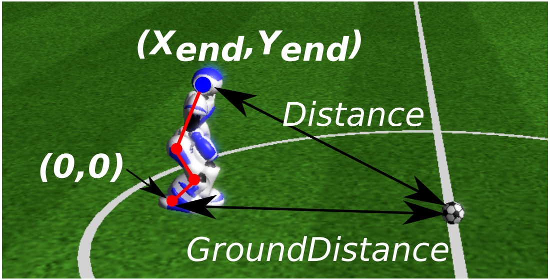
for $i = 1$ **to** n **do**
    find $LeftEscapeAngle_i$ for obstacle O_i
    find $RightEscapeAngle_i$ for obstacle O_i
end for
$EscapeAngleSet = \emptyset$
for $i = 1$ **to** n **do**
    if $LeftEscapeAngle_i \notin [LeftEscapeAngle_j, RightEscapeAngle_j], \forall j \neq i$ **then**
        $EscapeAngleSet = EscapeAngleSet \cup \{LeftEscapeAngle_i\}$
    end if
    if $RightEscapeAngle_i \notin [LeftEscapeAngle_j, RightEscapeAngle_j], \forall j \neq i$ **then**
        $EscapeAngleSet = EscapeAngleSet \cup \{RightEscapeAngle_i\}$
    end if
end for
return $EscapeAngleSet$

Complex Actions IV

Walk to Ball Makes the agent walk towards the ball and stop when the ball is close enough to perform a kick.

- It performs the Turn to Ball action and then walk towards the ball, slowing down when it comes close to the ball.
- Ball distance returned by the vision perceptor is the distance between the camera, which is attached to agent's head, and the ball.
- Forward kinematics along the sagittal plane of the robot to derive the current height of the camera.

Complex Actions V



Having the ball distance and the height of the camera, the ground distance can be easily derived using the Pythagorean Theorem.

$$GroundDistance = \sqrt{BallDistance^2 + CameraHeight^2}$$

Complex Actions VI

On Ball Action This action moves the agent close to the ball and executes an appropriate kick depending on the current state of the game.

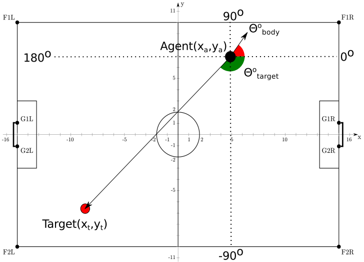
- It first performs the Walk To Ball action in order to reach the ball.
- After the successful completion of the Walk To Ball action, the agent performs the Find Opponent's Goal action.
- Subsequently, it aligns itself with the direction of the opponent's goal.
- The precision of this alignment is inversely proportional to the distance from the opponent's goal.
- Afterwards, it performs the Position for Kick action and finally it executes a kick motion.

Complex Actions VII

Walk to Coordinate This action moves the agent to a specific location (x_t, y_t, θ_t) in the field.

$$\phi = \text{atan2}(x_t - x_a, y_t - y_a)$$

$$d = \sqrt{(x_t - x_a)^2 + (y_t - y_a)^2}$$



Complex Actions VIII

- Distance and direction are recalculated as the agent makes progress toward its target.
- After the position (x_t, y_t) has been reached, a final rotation in place turns the agent towards the desired direction θ_t.
- The action terminates when the agent reaches the desired location (x_t, y_t, θ_t).

Walk To Direction This action makes the agent walk towards a specific direction.

- It employs a turn in-place action to align with the given direction and then a straight walk action to move along the given direction.
- This action is not terminated by itself but there has to be a new action request.

Complex Actions IX

Dribble Ball To Direction This action attempts to dribble the ball towards a specific direction.

- It is quite similar to the Walk To Direction action, however the agent tries to keep the ball in front of its feet at all times.
- This action is not fully functional in our software.

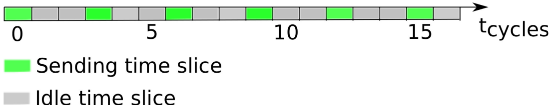
Communication I

Communication in Simspark is not ideal. There are no restrictions about the use of the say effector and every agent can use it at each cycle. However, the hear perceptor comes with some restrictions.

- The agent processes are not allowed to communicate with each other directly, but the agents may exchange messages via the simulation server.
- Messages should not have a length of more than 20 ASCII characters.
- Messages shouted from beyond a maximal distance (currently 50 meters) cannot be heard.
- Only one message can be heard at any given time and messages from the same team can be heard only every other cycle.

Communication II

A simple communication protocol has been created in which time is sliced into pieces lasting one server cycle (20ms) each and repeats every three cycles (60ms).



- Every time slice of the protocol has an associated integer label which indicates the uniform number of the player able to send its message at that slice.
- This label starts at 1 and grows by 1 every time a player sends a message.

Communication III

- Through this simple protocol, every player can receive reliably the messages from all teammates every 540ms (27 cycles) for a team of 9 players or every 660ms (33 cycles) for a team of 11 players.

Messages and Communication

Coordinationćs Beliefs

Subsets in Coordination

Coordination Splitter

Soccer Field Value

Active Positions

Active Coordination

Team Formation

Role Assignment Function

Positions for Support Subset

Support Coordination

Mapping Cost

Movement

Communication

Coordination

Matches

Future Work