RoboCup 2D Half Field Offense
Technical Manual

Matthew Hausknecht

October 2, 2015

Contents

1 Overview

This document describes the installation, usage, state, and action spaces of the HFO domain.

2 Installation

Installation with CMake:

```
> mkdir build && cd build
> cmake -DCMAKE_BUILD_TYPE=Release ..
> make -j4 # Replace 4 with the number of cores on your machine
> make install # This just copies binaries to the HFO directory; no sudo required
```

HFO installation has been tested on Ubuntu Linux and OSX. Successful installation depends on `CMake, Boost-system, Boost-filesystem, Flex`. By default, the soccerwindow2 visualizer is also built and requires `Qt4`. Experimentally speaking, HFO is fully-functional without the visualizer. To disable this component, use the following cmake command:

```
> cmake -DCMAKE_BUILD_TYPE=Release -DBUILD_SOCCERWINDOW=False ..
```

2.1 Python Interface

The Python interface is required for interfacing Python agents to the HFO domain. To install this interface, from the main HFO directory:

```
> pip install .
```
  or
```
> pip install --user .
```
  if you have limited permissions on the machine.

3 Uninstall

The install is completely contained in the HFO directory. Simply delete this directory to uninstall. If you have installed the python interface, uninstall it as follows: `pip uninstall hfo`.
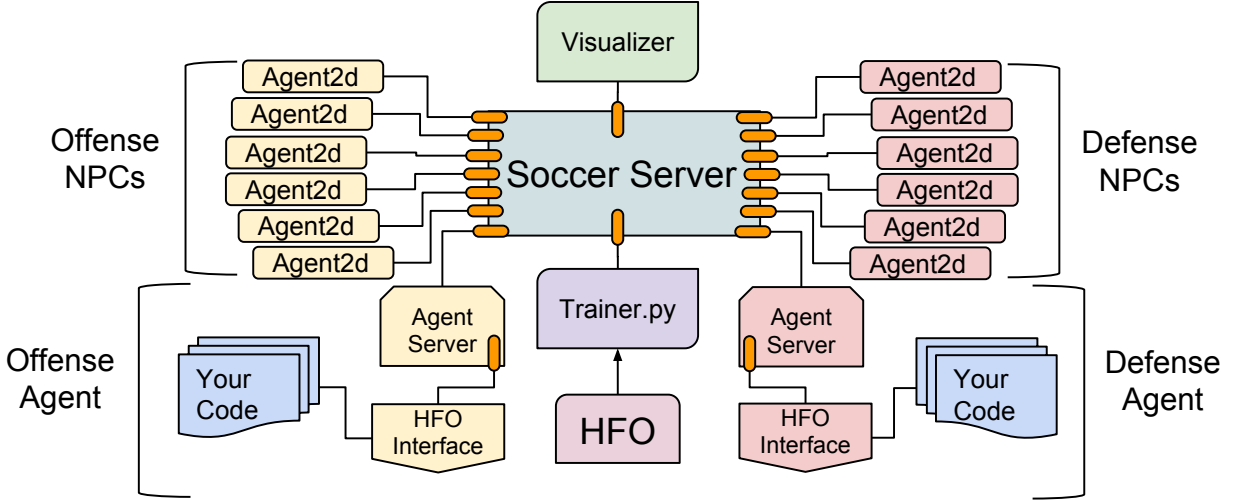
Figure 1: HFO is comprised of several components which communicate over the network. Network connections are depicted with orange ovals. Calling the HFO executable starts the trainer, visualizer, and all the offensive and defensive npcs (Agent2d) as well as the offensive and defensive agent servers. Your code then uses the HFO interface to connect your agent to an agent server. Once all servers have connected agents, the game begins. The trainer oversees the game and is responsible for resetting the players between episodes.

4 Basic Usage

RoboCup 2D soccer is designed to be played between two teams of autonomous agents who communicate with a game server. Shown in Figure 1, the HFO domain reflects these design choices and allows arbitrary teams to be created consisting of some mix of non-player-controlled agents (agent2d npcs) and player-controlled agents. These options are specified through the following flags:

```
> ./bin/HFO --offense-agents=1 --defense-agents=1 --offense-npcs=2 --defense-npcs=2
```

This would create a 3v3 game with one player-controlled agent on each team. In order for the game to start, you must connect your player-controlled agents to the waiting agent servers. This is done through the call:

```
> hfo.connectToAgentServer(6000, LOW_LEVEL_FEATURE_SET);
```
or in Python:
```
> hfo_env.connectToAgentServer(6000, HFO_Features.LOW_LEVEL_FEATURE_SET)
```

By default, the server for the first agent is allocated to port 6000. Subsequent ports are allocated sequentially backwards (e.g. to connect the second agent, port 5999 would be used). The default port may be changed as follows:

```
> ./bin/HFO --port 12345
```

5 Visualizer

The SoccerWindow2 Visualizer allows a live game to be viewed as it progresses. By default, the visualizer is enabled. However, the game will likely proceed at a pace too fast for meaningful watching. To enforce a standard pace, disable sync-mode:

```
> ./bin/HFO --no-sync
```

To disable visualization altogether, run in headless mode:

```
> ./bin/HFO --headless
```

The visualizer may also be used after the end of a game by replaying logs, as discussed in the next section.

6 Logging

By default, the soccer server generates game logs and stores them in the `log` directory. The main game log is `log/incomplete.rcg`. This log may be replayed using the soccerwindow2 visualizer.

To replay a log:
```
> ./bin/soccerwindow2 -l log/incomplete.rcg
```
To disable logging:
```
> ./bin/HFO --no-logging
```
To change the logging directory:
```
> ./bin/HFO --log-dir /path/to/new/dir
```

7 Recording

It is possible to record the low-level state perceptions, actions, and game status of all players:

```
> ./bin/HFO --record
```

This will produce logs for all the offensive players (`log/left-[1-11].log`) and defensive players (`log/right-[1-11].log`). The first offensive player is left-11, so in the case of single-agent offense, left-11.log will contain the active player's record. The log, `incomplete.rcg` may be used to verify the player numbers on field.

8 Randomness

A seed may be specified as follows:

```
> ./bin/HFO --seed 123
```

This seed will determine the placement of the players and the ball at the beginning of each episode. Due to non-determinism in the player policies and server, it is not sufficient to precisely replicate full games. It *only* replicates the starting conditions for each episode. The player's behavior, observations, and physics all proceed stochastically.

9 State Spaces

The HFO domains provides a choice between a low-level feature set and a higher-level feature set. Selecting between the different feature sets is accomplished when connecting to the agent server. Also see `examples/hfo_example_agent.cpp` and `examples/hfo_example_agent.py` for examples:

```
> hfo.connectToAgentServer(6000, LOW_LEVEL_FEATURE_SET);
> hfo.connectToAgentServer(6000, HIGH_LEVEL_FEATURE_SET);
```

9.1 High Level Feature Set

A set of high-level features is provided following the example given by Barrett et al. pp. 159-160 [1]. Barrett writes "There are many ways to represent the state of a game of half field offense. Ideally, we want a compact representation that allows the agent to learn quickly by generalizing its knowledge about a state to similar states without over-constraining the policy." All features are encoded a floating point values and normalized to the range of [-1,1]. Invalid features are given a value of -2. The features are as follows:

9.1.1 High Level State Feature List

0 **X position** - The agents x position on the field.

1 **Y position** - The agents y position on the field.

2 **Orientation** - The direction that the agent is facing.

3 **Ball Distance** - Distance to the ball.

4 **Ball Angle** - Angle to the ball.

5 **Able to Kick** - Boolean indicating if the agent can kick the ball.

6 **Goal Center Distance** - Distance from the agent to the center of the goal.

7 **Goal Center Angle** - Angle from the agent to the center of the goal.

8 **Goal Opening Angle** - The size of the largest open angle of the agent to the goal, shown as θ_g in Figure 2. Invalid if agent is not playing offense.

T **Teammate i's Goal Opening Angle** - For each teammate i: is goal opening angle. Invalid if agent is not playing offense.

1 **Distance to Opponent** - If an opponent is present, distance to the closest opponent. This feature is absent if there are no opponents.

T **Distance from Teammate i to Opponent** - For each teammate i: the distance from the teammate to the closest opponent. This feature is absent if there are no opponents. If teammates are present but not detected, this feature is considered invalid and given the value of -2.

T **Pass Opening Angle** - For each teammate i: the open angle available to pass to teammate i. Shown as θ_p in Figure 2. If teammates are present but not detected, this feature is considered invalid and given the value of -2.

$3T$ **Distance, Angle, and Uniform Number of Teammates** - For each teammate i: the distance, angle, and uniform number of that teammate.

There are a total of $9+5*$num_teammates features with an additional $1+$num_teammates features if at least one opponent is present.

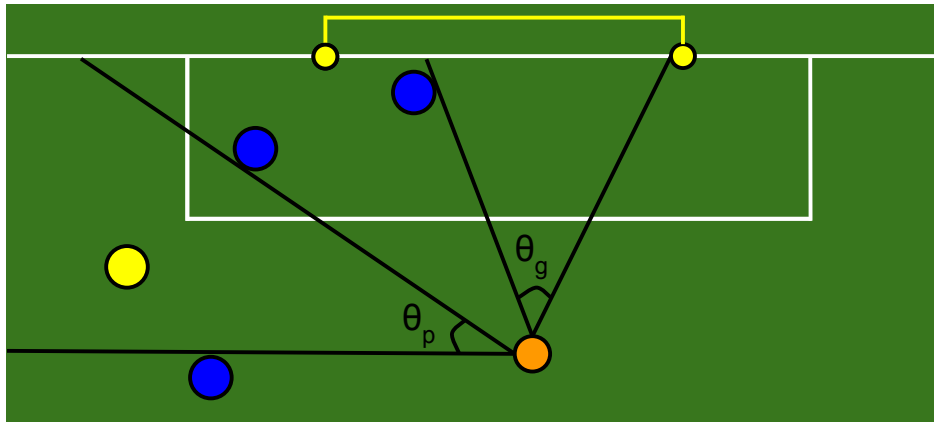Figure 2: Open angle from ball to the goal θ_g avoiding the blue goalie and the open angle from the ball to the yellow teammate θ_p. Figure reproduced with permission from Samuel Barrett.

9.2 Low Level Feature Set

The state features used by HFO are designed with the mindset of providing an overcomplete, basic, egocentric viewpoint. The features are basic in the sense that they provide distances and angles to relevant points of interest, but do not include higher level perceptions such as the largest angle between a goal post and goalkeeper.

All features are encoded as floating point values normalized to the range of [-1,1]. Different types of features are discussed next.

9.2.1 Boolean Features

Boolean features assume either the minimum feature value of -1 or the maximum feature value of 1.

9.2.2 Valid Features

Since feature information is attained from the Agent's world-model, it is possible that, the world model's information may be stale or incorrect. *Valid features* are boolean features indicating consistency of world model predictions. For example, if the world model's estimate of the agent's position is known to be flawed, the *valid feature* for self position would assume the minimum value of -1. Otherwise it will assume the maximum value of 1.

The features associated with a valid feature are given the value of zero if an inconsistency is detected. For example, if the world model detects that the agent's velocity is invalid, the feature that encodes the magnitude of self velocity will be set to zero.

9.2.3 Angular Features

Angular features (e.g. the angle to the ball), are encoded as two floating point numbers – the $sin(\theta)$ and $cos(\theta)$ where θ is the original angle in radians. Figure 3 provides examples of the angular encoding.

This encoding allows the angle to vary smoothly for all possible angular values. Other encodings such as radians or degrees have a discontinuity that when normalized, could cause the feature value to flip between the maximum and minimum value in response to small changes in θ.

Given an angular feature $\langle \alpha_1, \alpha_2 \rangle$ we can recover the original angle θ (in radians) by taking the $cos^{-1}(\alpha_2)$ and multiplying by the sign of α_1.

9.2.4 Distance Features

Distance features encode the distance to objects of interest. Unless otherwise indicated, they are normalized against the maximum possible distance in the HFO playfield (defined as $\sqrt{l^2 + w^2}$ where l, w are the length and width of the HFO playfield). A distance of zero will be encoded with the minimum feature value of -1 while a maximum distance will be encoded with 1.
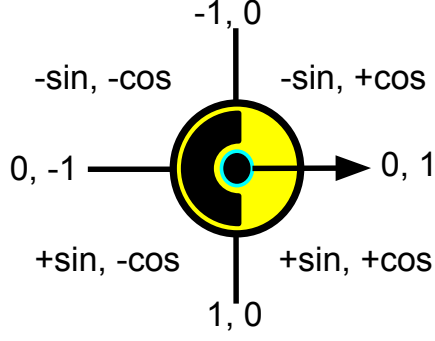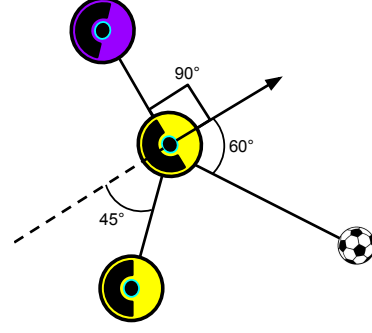
(a) Angular Encoding (b) Additional Examples

Figure 3: **Angular Encoding:** Objects on the agents left/right side result in a negative/positive $sin(\theta)$. $cos(\theta)$ is positive in front of the player and negative behind. For example, an object directly in front of the player would have angular features of $sin(\theta) = 0, cos(\theta) = 1$. Additional examples: **Angle to ball** $\theta = 60°$ or 1.0472 radians. This results in angular features $\langle sin(\theta) = .86, cos(\theta) = .49 \rangle$. **Angle to teammate**: $\theta = 135°, 2.35$ radians. $\langle sin(\theta) = .71, cos(\theta) = -.71 \rangle$. **Angle to Opponent**: $\theta = -90°$ or -1.57 radians. $\langle sin(\theta) = -1, cos(\theta) = 0 \rangle$.

9.2.5   Landmark Features

Landmark features encode the relative angle and distance to a landmark of interest. Each landmark feature consists of three floating point values, two to encode the angle to the landmark and one to encode the distance. Note that if the agent's self position is invalid, then the landmark feature values are zeroed.

9.2.6   Player Features

Player features are used to encode the relationship of the agent to another player or opponent. Each player feature is encoded as 1) a landmark feature of that player's location 2) the global angle of that player's body 3) the magnitude of the player's velocity and 4) the global angle of the player's velocity. Eight floating point numbers are used to encode each player feature.

9.2.7   Other Features

Some features, such as the agent's stamina, do not fall into any of the above categories. These features are referred to as *other features*.

9.2.8   Low Level State Feature List

Basic Features are always present and independent of the number of teammates or opponents. The 32 basic features are encoded using 58 floating point values (*angular features* require two floats, *landmark features* require 3). Additionally a variable number of *player features* are then added. This number depends on the number of teammates and opponents in the

HFO game, but 8 floating point values are required for each player feature. Thus, the total number of features is $58 + 8 * \text{num\_teammates} + 8 * \text{num\_opponents}$.

    0 **Self_Pos_Valid** [Valid] Indicates if self position is valid.

    1 **Self_Vel_Valid** [Valid] Indicates if the agent's velocity is valid.

  2–3 **Self_Vel_Ang** [Angle] Angle of agent's velocity.

    4 **Self_Vel_Mag** [Other] Magnitude of agent's velocity. Normalized against the maximum observed self speed, 0.46.

  5–6 **Self_Ang** [Angle] Agent's Global Body Angle.

    7 **Stamina** [Other] Agent's Stamina: The amount of remaining stamina the agent has. Normalized against the maximum observed agent stamina of 8000.

    8 **Frozen** [Boolean] Indicates if the agent is Frozen. Frozen status can happen when being tackled by another player.

    9 **Colliding_with_ball** [Boolean] Indicates if the agent is colliding with the ball.

  10 **Colliding_with_player** [Boolean] Indicates if the agent is colliding with another player.

  11 **Colliding_with_post** [Boolean] Indicates if the agent is colliding with a goal post.

  12 **Kickable** [Boolean] Indicates if the agent is able to kick the ball.

13–15 **Goal Center** [Landmark] Center point between the goal posts.

16–18 **Goal Post Top** [Landmark] Top goal post.

19–21 **Goal Post Bot** [Landmark] Bottom goal post.

22–24 **Penalty Box Center** [Landmark] Center of the penalty box line.

25–27 **Penalty Box Top** [Landmark] Top corner of the penalty box.

28–30 **Penalty Box Bot** [Landmark] Bottom corner of the penalty box.

31–33 **Center Field** [Landmark] The left middle point of the HFO play area. True center of the full-field.

34–36 **Corner Top Left** [Landmark] Top left corner HFO Playfield.

37–39 **Corner Top Right** [Landmark] Top right corner HFO Playfield.

40–42 **Corner Bot Right** [Landmark] Bot right corner HFO Playfield.

43–45 **Corner Bot Left** [Landmark] Bot left corner HFO Playfield.

46 **OOB Left Dist** [Distance] Distance to the nearest point of the left side of the HFO playable area. E.g. distance remaining before the agent goes out of bounds in left field.

47 **OOB Right Dist** [Distance] Distance remaining before the agent goes out of bounds in right field.

48 **OOB Top Dist** [Distance] Distance remaining before the agent goes out of bounds in top field.

49 **OOB Bot Dist** [Distance] Distance remaining before the agent goes out of bounds in bottom field.

50 **Ball Pos Valid** [Valid] Indicates if the ball position estimate is valid.

51–52 **Ball Angle** [Angle] Angle to the ball from the agent's perspective.

53 **Ball Dist** [Distance] Distance to the ball.

54 **Ball Vel Valid** [Valid] Indicates if the ball velocity estimate is valid.

55 **Ball Vel Mag** [Other] Global magnitude of the ball velocity. Normalized against the observed maximum ball velocity, 3.0.

56–57 **Ball Vel Ang** [Angle] Global angle of ball velocity.

8T **Teammate Features** [Player] One teammate feature set (8 features) for each teammate active in HFO, sorted by proximity to the agent.

8O **Opponent Features** [Player] One opponent feature set (8 features) for each opponent present, sorted by proximity to the player.

10 Action Space

The HFO domain provides support for both low-level primitive actions and high-level strategic actions. Basic, parameterized actions are provided for locomotion and kicking. Additionally high-level strategic actions are available for moving, shooting, passing and dribbling. Control of the agent's head and gaze is not provided and follows Agent2D's default strategy. Both low and high level actions are available through the same interface. It is the responsibility of the user to faithfully report which action spaces were used.

10.1 Low Level Actions

- **Dash**(power, degrees): Moves the agent with power [-100, 100] where negative values move backwards. The relative direction of movement is given in degrees and varies between [-180,180] with 0 degrees being a forward dash and 90 degrees dashing to the agent's right side. Note, dashing does not turn the agent.

- **Turn**(degrees): Turns the agent in the specified direction. Valid values range between [-180, 180] degrees where 90 degrees turns the agent to directly to its right side.

- **Tackle**(degrees): Contest the ball. Direction varies between [-180, 180].

- **Kick**(power, degrees): Kick the ball with power [0, 100] in relative direction [-180, 180]. Has no effect if the agent does not possess the ball.

- **Quit**: Indicates to the agent server that you wish to terminate the HFO environment.

10.2 High Level Actions

- **Move**(): Re-positions the agent according to the strategy given by Agent2D. The *move* command works only when agent does not have the ball. If the agent has the ball, another command such as *dribble*, *shoot*, or *pass* should be used.

- **Shoot**(): Executes the best available shot. This command only works when the agent has the ball.

- **Pass**(teammate_uniform_number): Passes to the teammate with the provided uniform number. Does nothing if the player does not have control of the ball or the requested teammate is not detected.

- **Dribble**(): Advances the ball towards the goal using a combination of short kicks and moves.

11 Developing a New Agent

New agents may be developed in C++ or Python. In Python, as long as the hfo interface has been installed, the agent needs only to `from hfo import *`. In C++ it is necessary to `#include <HFO.hpp>` and also to link against the shared object library `lib/libhfo.so` when compiling:

```
> g++ example/your_new_agent.cpp -I src -L lib -Wl,-rpath=lib -lhfo
```

References

[1] S. Barrett. *Making Friends on the Fly: Advances in Ad Hoc Teamwork.* PhD thesis, The University of Texas at Austin, Austin, Texas, USA, December 2014.