RoboCup 2D Half Field Offense Technical Manual

Matthew Hausknecht

March 4, 2016

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`. 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`.

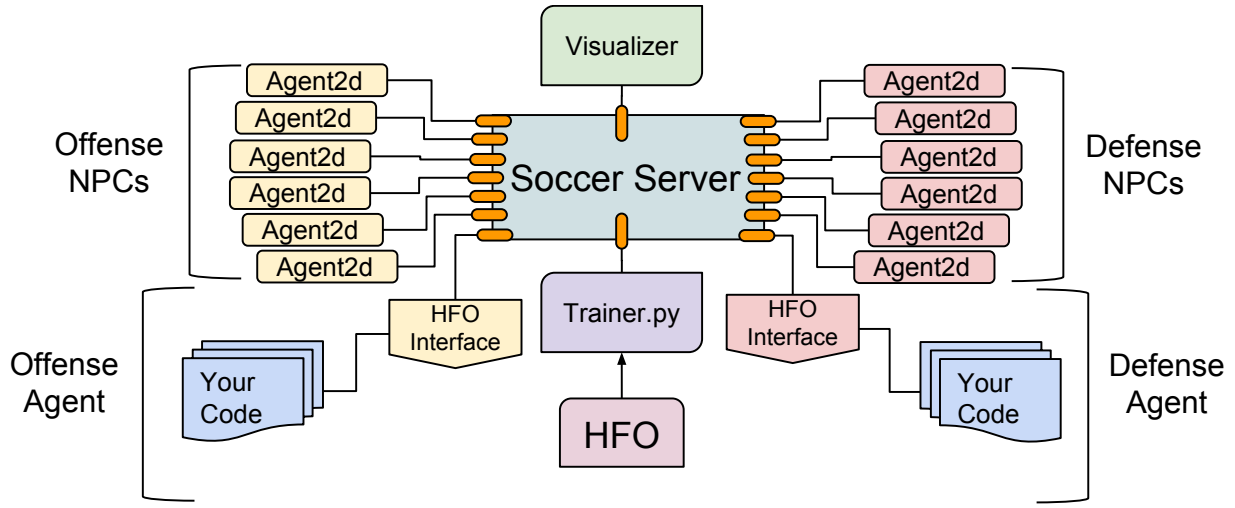


Figure 1: HFO is comprised of several components which communicate over the network. Network connections are depicted with orange ovals. Calling the HFO executable starts the trainer, visualizer, and all the offensive and defensive npcs (Agent2d) as well as the offensive and defensive agent servers. Your code then uses the HFO interface to connect your agent to the server. Once all agents are connected, the game begins. The trainer oversees the game.

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 server. This is done through the call:

```
> hfo.connectToServer(FEATURE_SET, port, etc);
```

The arguments to this function are provided by the HFO executable for each player upon starting the game. For example `./bin/HFO --offense-agents=1` prints all the information needed to connect the offensive player:

```
Waiting for player-controlled agent base_left-0:
  config_dir=HFO/bin/teams/base/config/formations-dt,
  uniform_number=11, server_port=6000, server_addr=localhost,
  team_name=base_left, play_goalie=False
```

By default, the server starts on port 6000, but 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 logs are rcg files: `log/*.rcg`. These 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 Making Videos

It is possible to make videos from logs by saving frames from SoccerWindow2. It helps to full-screen SoccerWindow2 before making a video as it will save higher quality images. There are also several display options under View → View Preference → Show that toggle what will be displayed. Saving frames can be done by File → Save Image. To convert the saved pngs into a movie:

```
avconv -r 10 -start_number 0 -i 3v3/image-%05d.png -f mp4 -c:v libx264
-s 1024x768 -vf "crop=iw/2.5:8.38*ih/10:iw/2:ih/10,transpose=1"
-pix_fmt yuv420p test.mp4
```

This command autocrops offensive half of the playfield and rotates it 90 degrees. Avconv can be replaced by ffmpeg. Start number specifies the number of the first frame. yuv420p pix format for OSX compatibility.

8 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. Note that for player controlled agents, it is necessary to specify a `record_dir` in the `connectToServer` function:

```
std::string record_dir = "log/";
hfo.connectToServer(features, config_dir, unum, port, server_addr,
                    team_name, goalie, record_dir);
```

9 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, 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.

10 Player On Ball

By default, episodes begin with the ball being randomly positioned in the offensive half of the playfield. Typically the first task for the offense is to send a player to collect the ball. It is possible to instead request that a certain offensive player is given the ball at the start of each episode. This is accomplished as follows:

```
> ./bin/HFO --offense-on-ball 1
```

The above command will always give the ball to the first offensive player (e.g. uniform number 11). If an offense-on-ball number is specified that is larger than the number of offensive players, the ball will be given to a random offensive player at the start of each episode.

11 Teams

By default, offensive and defensive NPCs use the base Agent2D policy. It is possible to use policies from different teams as follows:

```
> ./bin/HFO --offense-team helios
> ./bin/HFO --defense-team base
```

This would take offense NPCs from Helios' 2013 Eindhoven release and defensive NPCs from the default Agent2D-base. Currently the only supported teams are Helios and Base.

12 Communication

HFO allows agents to receive and broadcast messages. This is accomplished by the `hear` and `say` functions. The maximum allowed message size is controlled by HFO's `--message-size`

flag. See `examples/communication_agent.cpp` and `examples/communication_agent.py` for examples.

13 Controlling Trials

HFO trials typically end with a goal, the defense capturing the ball, the ball going out of bounds, or running out of time. The trials flag specifies a maximum number of trials `> ./bin/HFO --trials 500`. Instead, a maximum number of frames may be specified: `> ./bin/HFO --frames 1000` will stop the server after 10,000 steps have passed. Each trial is run for a maximum of `--frames-per-trial` steps, but may stop early if no agent approaches the ball within `--untouched-time` steps.

14 State Spaces

The HFO domains provides a choice between a low and a high-level feature set. Selecting between the different feature sets is accomplished when connecting the agent to the server:

```
> hfo.connectToServer(LOW_LEVEL_FEATURE_SET, ...);
> hfo.connectToServer(HIGH_LEVEL_FEATURE_SET, ...);
```

See `examples/hfo_example_agent.cpp` and `examples/hfo_example_agent.py` for examples. As the choice of feature set influences the challenge of learning, it is the responsibility of the user to faithfully report which state space was used. The following sections explain the feature sets.

14.1 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:

14.1.1 High Level State Feature List

Let T denote the number of teammates in the HFO game. Then there are a total of $9 + 5T$ high-level features with an additional $T + 1$ features if at least one opponent is present.

0 **X position** - The agents x-position on the field. See Figure 2.
1 **Y position** - The agents y-position on the field. See Figure 2.
2 **Orientation** - The direction that the agent is facing.

3 **Ball Proximity** - Agent's proximity 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 Proximity** - Agent's proximity 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 3. 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 **Proximity to Opponent** - If an opponent is present, proximity to the closest opponent. This feature is absent if there are no opponents.
T **Proximity from Teammate i to Opponent** - For each teammate i: the proximity 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 3. If teammates are present but not detected, this feature is considered invalid and given the value of -2.
$3T$ **Proximity, Angle, and Uniform Number of Teammates** - For each teammate i: the proximity, angle, and uniform number of that teammate.

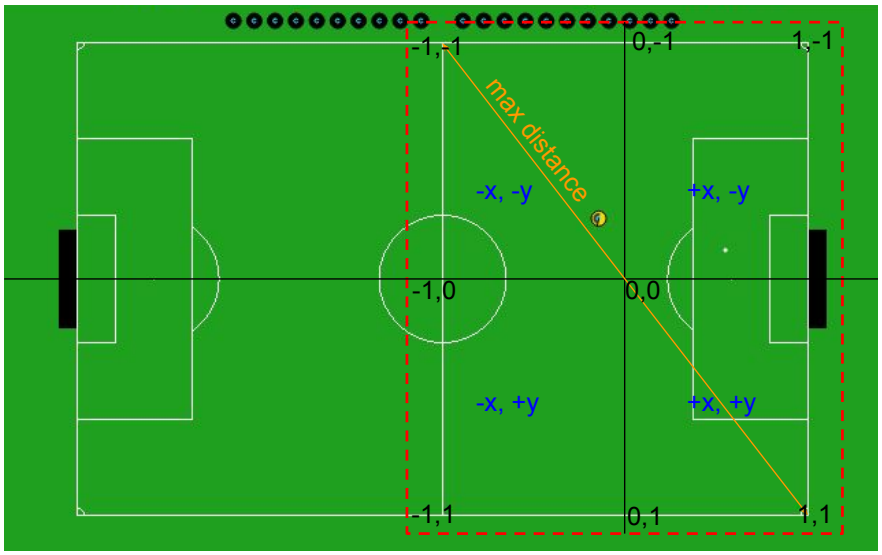


Figure 2: **Normalized Coordinates in the HFO play field**: These coordinates are used for reporting the agent's position in the high-level feature set as well specifying targets for the mid-level actions (Section 15.2). The red-rectangle shows the boundaries of the reported positions, which exceed the play field boundaries by 10% in each direction. Positions exceeding this rectangle are bounded (via min/max) to the edges of the rectangle. All distance features are normalized against the max HFO distance shown in orange.

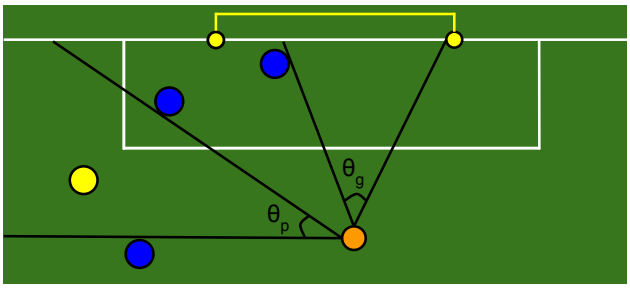


Figure 3: Open angle from ball to the goal θ_g avoiding the blue goalie and the open angle from the ball to the yellow teammate θ_p. Figure reproduced with permission from Samuel Barrett.

14.2 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]. Several different types of features exist:

14.2.1 Boolean Features

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

14.2.2 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.

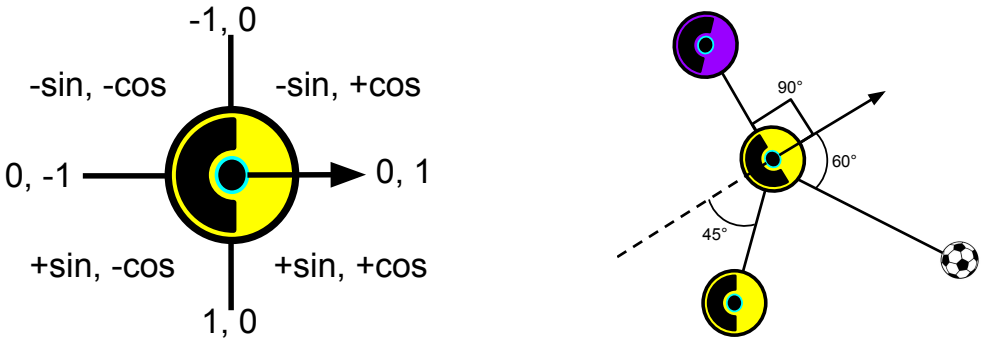
14.2.3 Angular Features

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 4 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.



(a) Angular Encoding

(b) Additional Examples

Figure 4: **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. $\langle sin(\theta) = -1, cos(\theta) = 0 \rangle$.

14.2.4 Proximity Features

Proximity features encode the proximity of the agent to an object 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 maximum proximity of 1 indicates the agent is co-located with the object of interest, while a minimum proximity of -1 indicates that the agent is across the field from the object of interest.

14.2.5 Landmark Features

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

14.2.6 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.

14.2.7 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* and are normalized in the range $[-1, 1]$.

14.2.8 Low Level State Feature List

Let T denote the number of teammates and O denote the number of opponents in the HFO game. Then there are a total of $58 + 8T + 8O$ low-level features:

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 the agent's velocity.
5–6	**Self_Ang** [Angle] Agent's Global Body Angle.
7	**Stamina** [Other] Agent's Stamina: Low stamina slows movement.
8	**Frozen** [Boolean] Indicates if the agent is Frozen. Frozen status can happen when tackling or being tackled by another player.
9	**Colliding_with_ball** [Boolean] Indicates the agent is colliding with the ball.
10	**Colliding_with_player** [Boolean] Indicates the agent is colliding with another player.
11	**Colliding_with_post** [Boolean] Indicates the agent is colliding with a goal post.
12	**Kickable** [Boolean] Indicates 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.
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** [Proximity] Proximity 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** [Proximity] Proximity to the right field line.
48	**OOB Top Dist** [Proximity] Proximity to the top field line.

49 **OOB Bot Dist** [Proximity] Proximity to the bottom field line.
50 **Ball Pos Valid** [Valid] Indicates the ball position estimate is valid.
51–52 **Ball Angle** [Angle] Agent's angle to the ball.
53 **Ball Dist** [Proximity] Proximity to the ball.
54 **Ball Vel Valid** [Valid] Indicates the ball velocity estimate is valid.
55 **Ball Vel Mag** [Other] Magnitude of the ball's velocity.
56–57 **Ball Vel Ang** [Angle] Global angle of ball velocity.
8*T* **Teammate Features** [Player] One teammate feature set (8 features) for each teammate active in HFO, sorted by proximity to the agent.
8*O* **Opponent Features** [Player] One opponent feature set (8 features) for each opponent present, sorted by proximity to the player.

15 Action Space

The HFO domain provides support for both low-level primitive actions, mid-level, and high-level strategic actions. Low-level, parameterized actions are provided for locomotion and kicking. Mid-level actions are still parameterized by capture high level activities such as dribbling. Finally, high-level discrete, 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. Low, medium, and high level actions are available through the same interface. As the choice of action spaces greatly influences the challenge of learning, it is the responsibility of the user to faithfully report which action spaces were used.

15.1 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.

15.2 Mid Level Actions

- **Kick_To**(target_x, target_y, speed): Kicks the ball to the specified target point with the desired speed. Valid values for $\text{target}_{x,y} \in [-1, 1]$ and speed $\in [0, 3]$.
- **Move_To**(target_x, target_y): Moves to the specified target point using the max dash speed. Valid values for $\text{target}_{x,y} \in [-1, 1]$.

- **Dribble_To**(target$_x$, target$_y$): Dribbles the ball to the specified target point. Attempts to fetch the ball if the agent doesn't already possess it. Performs some checks to avoid opponents and keeps good control of the ball. Valid values for target$_{x,y} \in [-1, 1]$.
- **Intercept**(): Moves to intercept the ball, taking into account the ball velocity. More efficient than chasing the ball.

15.3 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.

15.4 Special Actions

- **NO-OP**: Indicates that the agent should take no action.
- **Quit**: Indicates to the agent server that you wish to terminate the HFO environment.

16 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.