Server Specification of the Virtual Soccer Competition [Preliminary]

This is a preliminary document describing the server infrastructure of the virtual RoboCup Humanoid League soccer competition. The next major update is planned for April 19th. In case you feel that your team requires elements to run the robot controller which are not described in or in conflict with this document, please contact the TC as soon as possible and we will examine your request. Details on where exactly the tournament will be hosted are still under exploration and details are expected in the next major update.

Computing

Each robot has its own, dedicated Linux VM ("robot VM"). The robot VMs will have the following processing power:

- **CPU:** 4 vCPU based on Intel Cascade Lake CPUs or above
- **GPU:** NVIDIA T4 or above

Inside each robot VM a Docker image with the robot control software for exactly one robot is running. There will be other processes running on the robot VM besides the ones in the Docker container but the footprint of these processes should be negligible.

Docker Images

Teams are expected to provide Docker images for their robot control software. It is possible to configure a different Docker image for each robot in the `config.json`. The arguments passed when running a Docker image (the `CMD`) can be configured via the `config.json`. Furthermore, there will be the following environment variables available in the Docker containers:

- `ROBOCUP_ROBOT_ID`: The id of your robot as specified in the `config.json`. Valid values are 1 - 4 (KidSize) or 1 - 2 (AdultSize).
- `ROBOCUP_TEAM_COLOR`: Your team color as announced in the game schedule. It is equal to either `red` or `blue`.

Networking

The network consists of the following participants:

- **Robot VMs:** Each team has 4 (KidSize) or 2 (AdultSize) robot VMs.
- **Simulator VM:**
  - **Webots:** The simulator itself with 4 (KidSize) or 2 (AdultSize) robots per team.
  - **API server:** Robot VMs access their virtual robots via the API server.
  - **GameController:** The GameController sends information about the game state to the robot VMs.
  - **AutoRef:** The AutoRef (Automatic Referee Service) replaces the human referee and communicates its decision to the GameController.

Both the API server and the AutoRef are implemented as Webots supervisors.

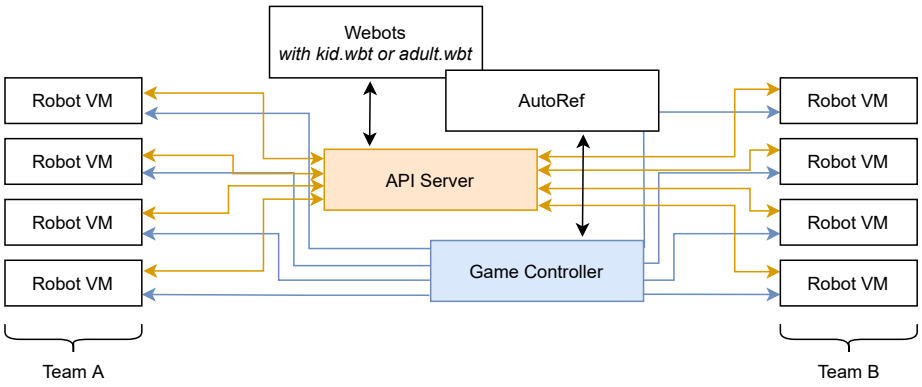

Figure 1: The network communication of the virtual soccer competition. Information from the Webots simulator are only accessible via the API server. The AutoRef only interacts with the GameController and has no direct communication to the robot VMs.

The total bandwidth between all robot VMs from a single team and the simulator API is limited to 50 MB/s and is evenly distributed among all robots playing. For example, if a team from KidSize has provided a `config.json` with 4 robot ids, each robot VM has a bandwidth of 12.5MB/s. The total bandwidth between robot VMs (of the same team) is limited to 1MB/s. Robot VMs of opposing teams are not allowed not communicate.

The Docker image is run using the host network (think of `docker run --network host`). Thus all ports from within the Docker container will be reachable from outside the robot VM.

For communication between robots, Multicast will not work. This is a limitation

of Docker, see [multicast1] and [multicast2]. We are currently investigating the exact list of the different ways of communicating that will be allowed for teams.

Game Logs

After the video stream of a game is finished, the logs of that game are released. Some logs are released publicly, some are only released to the respective teams.

Public logs:

- **Recording of the game:** The recording will be available both as a video and as a custom format provided by Cyberbotics to generate a 3d-replay version of the game.
- **AutoRef logs:** All decisions taken by the AutoReferee system.
- **GameController logs:** Internal log messages and updates sent to the robot control instances.

Team-internal logs:

- **Docker log:** The output (stdout and stderr) of the robot control software (think of `docker logs`).
- **Custom log folder:** We want to enable teams to record logs for debugging purposes with a limit of 10 GB per robot VM. Thus, each Docker container will have the folder `/robocup-logs` where you are able to write files into. After a game is finished, the files of each of the robot's custom log folder are copied and you will get read-access to that copy. This folder is not shared across the Docker containers (i.e., you are not able to communicate via this folder with the other Docker containers. Each robot has their own folder.). The folder is also not persistent across games, it will be initially empty at the beginning of each game.

Public logs are released to the general public and they will remain accessible at least until the end of the tournament. Team-internal logs will be made available only to the respective team and teams will only receive access to the last three games they played (so that the oldest is deleted when a new one comes in). Team are asked to only download their internal log files once and then distribute it internally among team members to keep downloads from our servers manageable.

We have yet to decide on how logs will be exactly be made accessible.

Storage

Each VM has a HDD or SSD storage with at least 10 GB of free storage. That storage can be accessed via the Docker overlay filesystem (the main filesystem mounted at / in the Docker container). The storage is not persistent across games and is reset after each game.

Note that teams can not pre-populate the filesystem and are expected to store all data they need (including assets such as weights for neural networks) in the Docker image. The filesystem can be only used to store temporary files.

The custom log folder at `/robocup-logs` has an additional 10 GB of storage. The read/write speed of the custom log folder may be slower than the main storage (exact specifications will be released later).

Game Procedure

Prior to the competition, teams submit their `config.json` (the team configuration file, see the API document [apidocument]) and PROTO files (the Webots robot models) via the Humanoid League submission system.

During the competition, teams will be informed of the official streaming schedule for each game. Note that the communicated schedule is the *streaming* time and not when the actual simulated games are played. The simulated games will be played in a two hour time frame before the game is streamed.

A game proceeds as follows:

1. Two hours before the official streaming schedule, the Docker images defined in the `config.json` are pulled (they are pulled before each game so teams can make fixes during the competition).
2. The simulation is started by the Technical Committee and the robot model files are loaded into the simulator.
3. Each robot defined in the `config.json` is assigned a VM, and on each of these robot VMs, the Docker image (which is already pulled) is run with the command provided in the `config.json`.
4. The AutoRef waits at least two minutes between the start of the robot VMs and the start of the match. The start of the match is defined by the game state being set to `READY` for the first half time in accordance with the laws of the game.
5. The GameController announces that the game has ended. The team's Docker containers get a time frame of two minutes for shutting down gracefully before being killed (think of `docker stop --time=120`).
6. The game is streamed on Twitch to the public in accordance with the game schedule.
7. After the game finished streaming, the game logs are released.

Also note that the actual duration of the game will depend on real-time factor of the simulation.

If a team provides an invalid reference to a Docker image, the respective robot model is still spawned in the game in accordance with the `config.json`. In case at least one valid Docker image was provided, the game proceeds normally. If no valid Docker image is provided, the game is counted as a forfeit.

References

apidocument API Specifications for Virtual Soccer Competition, https://cdn.robocup.org/hl/wp/2021/04/v-hsc_simulator_api_draft2.pdf

multicast1 Cannot receive external multicast inside container, https://github.com/moby/moby/issues/23659

multicast2 Multicast in Overlay driver, https://github.com/moby/libnetwork/issues/552