Robot Model Specifications v1.02

This document presents the constraints for creating robot models allowed to compete in the virtual humanoid soccer competition. From now on, this document will only be updated in cases of serious concern.

If you developed or are using a tool which makes the process easier for you, feel free to share it with the TC. In case you feel that your team requires elements which are not described in this document (e.g., specific contact properties between the robot feet and the ground), contact the TC as soon as possible and we will examine your request.

Introduction

This document aims at providing teams with clear specifications regarding what is allowed in their robot models but also advices on how to write their models and review the one provided by other teams.

The robot models for the competition will have to be written in the PROTO format. This format generally allows for a wider variety of sensors than what can be used for the Humanoid League, therefore we will present additional constraints in this document.

The main concerns of those rules and the reasons for these restrictions are the following:

- From a **research** point of view, this competition aims at providing an environment for transfer learning through sim2real experiments.
- In order to achieve **realism**, the characteristics and design of the robots should resemble robots used in regular competitions.
- To ensure a **fair** competition, teams who try to model their robots accurately by taking into account noise models and flaws of their hardware should not be penalized with respect to others.
- Using an **automatic referee** implies a few additional constraints on the robot models.
- Since the **resources** available for the simulation are limited, restrictions on the architecture of robots and on the accuracy of meshes may be required.

Teams can upload multiple robot models if they would like to play with different robots. In KidSize a maximum of 4 different robots is allowed. In AdultSize a maximum of 2 different robots is allowed.

Besides the robot model itself, a documentation similar to the robot specification in previous years, but more extensive, needs to be provided. This documentation must include all relevant datasheets of the sensors used in the robot (e.g. motors, acclerometers, gyroscopes, cameras) and a description of how the parameters in the proto were derived from the datasheets.

For custom developed hardware, the test procedure of how the performance values were measured must be described. When experiments have been run by the teams, sharing the raw data is encouraged.

While an example of values for the Dynamixel Motors MX28, MX64, MX106, and XH540-W270 are provided in the appendix, teams can use custom values if they provide a proper benchmark of the hardware as justification.

Model constraints

All sensor and actuator nodes that are not explicitly mentioned in this section are forbidden during the competition.

Exported parameters

As the robot model must be in the PROTO syntax, the following fields have to be exposed:

- `translation`
- `rotation`
- `name`
- `controllerArgs`

All the exposed field are automatically filled by the referee. Other parameters may be exposed, but they will be ignored (e.g. set to their default value).

It is encouraged to expose key parameters of the model such as the parameters for each type of motor used.

```
PROTO MyRobot [
  field  SFVec3f     translation         0 0 0
  field  SFRotation  rotation            0 1 0 0
  field  SFString    name                ""
  field  MFString    controllerArgs      []
  field  SFFloat      MX64-torque-12V    6.00
  field  SFFloat      MX64-vel-12V       6.60
  field  SFFloat      MX64-damping-12V   1.51
  field  SFFloat      MX64-friction-12V  1.42
  field  SFFloat      MX106-torque-12V   8.40
  field  SFFloat      MX106-vel-12V      4.71
  field  SFFloat      MX106-damping-12V  0.79
  field  SFFloat      MX106-friction-12V 2.14
]
{
  Robot {
    translation IS translation
    rotation IS rotation
    name IS name
    controllerArgs IS controllerArgs
    ...
  }
}
```

As mentioned in the README provided by Cyberbotics, you should parse the `name` field and ensure your robot adapts to it to display its team color and player number:

```
  if fields.name.value ~= '' then
    -- name is supposed to be something like "red player 2" or "blue player 1"
    local words = {}
    for word in fields.name.value:gmatch("%w+") do table.insert(words, word) end
    local color = words[1]
    local number = words[3]
```

Then, the `color` and `number` variables should be used by your PROTO file to display the requested color and player number. This can be achieved by forging a texture name from these variables or using them directly to assign material colors, create shapes, etc.

Finally, `selfCollision` should be activated on all your models.

Sensors

For several sensors, a Look up Table (LUT) is used to specify the response of the sensor. This information is also used to specify the limits (min and max) for each sensor and the noise profile. This table should always be set according to the hardware specifications of the sensor you are modelling.

The following list of sensors is allowed with the restrictions mentioned here:

- Position sensors
  - The resolution of sensors should match the hardware specifications. For example, a 12 bit rotary encoder (like in the Dynamixel) has a resolution of $\frac{2\pi}{2^{12}} \approx 0.0015$.
- Accelerometer
  - LUT should be specified
- Gyro
  - LUT should be specified
- Touch Sensor
  - The 3 different options allowed are `Bumper`, `Force`, `Force-3d`.
  - Since the calculation of forces in the simulation is inherently noisy, no extra noise needs to be modeled for these sensors.
- Cameras

  - The maximal amount of raw data that a team is allowed to create through cameras is set to 100MB/s (MegaByte/second) *per team* (based on raw RGB images). This requirement ensures that the rendering of images is not slowing down the simulation. An example of a valid configuration with 4 robots is the use of 640*480 images at 27 FPS.
    * Note: This limit is above the global network limit. Therefore, in order to use all of this bandwidth, you will need to use `JPEG` compression on the network which is not implemented yet in the Webots API yet.

Note: for the accelerometer and Gyro sensors, an offset drifting slowly will be added inside the simulator. This will model the fact that those sensors are highly sensitive to parameters such as the temperature. These parameters, however, will not be accessible in the PROTO file.

Actuators

Active joints can be implemented in two different ways:

- HingeJointWithBacklash for angular articulations, with the following child:
  - A RotationalMotor
  - The `backlash` set according to the real hardware.
  - Parameters `maxTorque` and `maxVelocity` are set according to the detailed procedure below
  - The `jointParameters` should be set according to the real hardware parameters.
    * `springConstant`: If your robot has a Parallel Elastic Actuator (PEA) a spring force matching the real robots PEA may be specified.
      · Parameters `dampingConstant` and `staticFriction` are set according to the detailed procedure below
- Hinge2JointWithBacklash for angular articulations around two axes. This is preferred when possible as it speeds up the simulation.
  - Parameters are the same as for the HingeJoint but need to be set for each motor individually
- SliderJoint for linear articulations, with the following child
  - A linear motor with the field `maxForce` set to a value matching the hardware specifications.
  - `JointParameters` should be set similar to the joint parameters of a HingeJoint

Both types of joints need to have the following child as well:

- A position sensor matching the constraints expressed in the previous section

Actuator and Joint Parameters

The following parameters need to be derived from the Motor specifications: - `maxTorque` - `maxVelocity` - `dampingConstant` - `staticFriction`

The following procedure should be used to determine these values from the datasheet:

`maxTorque` = *stallTorque*

`maxVelocity` = *maxVelocity*

To generate the other two parameters, the NT-Curve of the motor is used. It is assumed to be linear for simplicity. The damping constant is the negative value of the slope of the NT-Curve. To calculate it, the end points of the curve are read and the slope is calculated.

Static friction is calculated via the intersection of the NT-Curve with the x-axis to get the *effectiveTorque*. Then `staticFriction` = *stallTorque* - *effectiveTorque*

When multiple operating voltages are available, but only an NT-Curve for one of them (as is the case for the Dynamixel motors), the values are scaled up accordingly.

To illustrate the whole procedure we will look at the MX28 datasheet.

At 12V: - `maxTorque` = *stallTorque* = 2.5 Nm - `maxVelocity` = *maxVelocity* = 55 rev/min = 5.76 rad/s

We read the two endpoints of the NT-curve as *Point1 = (0.112 Nm, 49 rev/min = 5.13 rad/s)*, *Point2 = (1.288 Nm, 12 rev/min = 1.26 rad/s)*. To get the `dampingConstant` we calculate the negative slope (with flipped axes):

vel_diff = (Point2.y - Point1.y) *torque_diff* = (Point2.x - Point1.x)

`dampingConstant` = -(torque_diff / vel_diff) = -((1.288 - 0.112) / (1.26 - 5.13)) = 0.30

To get the static friction, the *effectiveTorque* is calculated as the intersection of the NT-curve with the y-axis:

effectiveTorque = -(Point1.y / (*vel_diff*/*torque_diff*)) - Point1.x = 1.67

`staticFriction` = *stallTorque* - *effectiveTorque* = 2.5 - 1.67 = 0.83

When a voltage different from 12V is used, the points read in the graph are scaled in the x-axis by the torque scaling factor and in the y-axis by the velocity scaling factor.

velocityScalingFactor = *maxVelocityAt14_8V* / *maxVelocityAt12V*

torqueScalingFactor = *maxTorqueAt14_8V* / *maxTorqueAt12V*

To ease the the process of creating your `PROTO` files, two elements are provided in the appendix:

- The parameters computed for classical dynamixel motors.
- A script allowing to compute these values easily from the datasheet of your own motors.

Structure of the robot

Teams should provide in a dedicated file named `postures.json` the configurations to be used:

- For upright posture (with fully extended knee)
- Longest extension posture

The format of the file should be as follows (values in rad):

```
{
    "robot_name" : "MyRobotName",
    "upright" : {
        "motor_1_name": 0.123,
        "motor_2_name": 0.456,
        ...
    },
    "extension": {
        "motor_1_name": 0.123,
        "motor_2_name": 0.456,
        ...
    }
}
```

Each robot needs to comport body parts with the following annotations as suffixes on the Solid names:

- `[foot]`: for all the body parts that can be in contact with the ground when walking (e.g. `Solid.name "left foot [foot]"`).
- `[arm]`: for all body parts between shoulder and hand. Those parts are not allowed to touch the ball (e.g. `Solid.name "left elbow [arm]"`).
- `[hand]`: each robot must have two solids declared as hands. Those solids might have a very low mass and volume, but the center of the solid should be at the end of the arm. Those annotations are used to compute the convex hull of the robot.

Some joints (i.e., motors) should also be annotated with the following suffixes:

- `[shoulder]`: axis of the first joint of the arm (e.g. `name "right shoulder pitch [shoulder]"`).
- `[hip]`: for the first leg joint with an axis lying parallel to the ground plane (e.g. `name "left hip roll [hip]"`).

Annotations of the joints should be made in the name field of the device, as seen in the following example:

```
HingeJointWithBacklash{
      jointParameters HingeJointParameters {
        ...
```

```
        }
        device [
            RotationalMotor {
                name "LeftHipRoll [hip]"
                ...
            }
            PositionSensor {
                ...
            }
        ]
}
```

Guidelines for writing your models

For general guidelines on how protos should be written please view the Cyberbotics documentation.

Usage of the `DEF/USE` mechanism as well as splitting up the `PROTO` into multiple subfiles (especially each visual or more complex collision model in its own file) is encouraged.

If you already have a `URDF` of your robot, using urdf2webots can give you a good starting point but manual adjustments are probably required.

If you agree that your robot model is made available to other teams and the public, we suggest adding a license on top of the PROTO file.

Collision model complexity

Since simulation has to be performed at a reasonable real time factor, collision models (e.g., boundingObject) should be constructed from geometric primitives (box, capsule, cylinder, sphere). If your robot is using cleats under the feet, they must be modeled using capsules or spheres rather than cylinders. This will reduce the computational complexity of calculating the contact between the grass and the cleats.

To manually create these collision models we suggest a tool based on onshape-to-robots pure shape approximation which will be released in the near future.

If you are already using a `URDF` model of your robot, rotated bounding boxes can be generated using the `simplify_urdf_collision` tool.

Visual model

While visual model complexity (i.e., the number of triangles) is a lesser issue than collision model complexity, it should still be set reasonably low (i.e. below 50 000 vertices in total).

When exporting a model from a CAD software, the resolution of the mesh approximation can usually be set. When only a mesh model is available, sofware such as Meshlab's Quadratic Edge Collapse Decimation can be used.

We suggest to apply an Appearance or PBRAppearance matching the real robot. Several appearances are provided by Cyberbotics and described here.

Model inspections

The inspection of the models will be based on the following elements:

- The content of the `PROTO` model and the associated files.
- The documentation provided describing how you obtained the different characteristics for your actuators and your sensors.
- The `postures.json` file.

The inspection will be performed based on 3 different parts:

- *Semi-automated validation*: to automatically check geometric proportions and provide a summary of the main characteristics for each robot.

- *Peer-reviewed validation*: manual check that the provided `PROTO` files respect the rules for some elements that cannot be checked automatically.
- *TC Validation*: manual verification of the issues risen by the teams and additional checks on submission materials.

Semi-automated validation

The following properties of the robot are extracted automatically:

- `Htop`: In upright posture, the amplitude along the z component.
- `Hleg`: In upright posture, z component between `[hip]` and the minimal value of the robot along z-axis.
- `Hhead`: In upright posture, z component between `[shoulder]` and the maximum value along z-axis.
- `M`: the total mass of the robot.
- `Hcom`: In upright posture, the height of center of mass.
- `BMI = M/Htop^2`
- `Width`: In upright posture, the diameter of the smallest cylinder in which the robot can fit.
- `FootWidth, FootLength`: The size along y (resp x) axis of the bounding box for one foot.
- `MaxLength`: The maximal distance between two points of the robot in longest extension posture.

The following constraints are checked automatically

- `5 <= BMI <= 30`
- `(FootWidth*FootHeight) <= (2.2*Hcom)^2/32`
- `1.2 <= max(FootWidth,FootHeight)/min(FootWidth, FootHeight) <= 3.5`
- `Width <= 0.55 Htop`
- `0.35 Htop <= Hleg <= 0.7 Htop`
- `0.1 Htop <= Hhead <= 0.3 Htop`
- Only allowed sensors/actuators are used
  - All required fields are set (e.g., LUT)
- Only geometric primitives used as collision model
- Complexity of visual model

A document containing the following information is generated from the robot model:

- List of the properties of the robot model extracted automatically
- List of all the joints with all parameters specified in the respective section
- List of the sensors and their parameters
- Kinematic structure of the robot

Peer-reviewed validation

The points that should be validated during the review process are the following:

- Are specifications for custom actuators valid?
- Are specifications for sensors valid?
- Are the annotations of the solids (`[foot]` and `[arm]`) and the links (`[shoulder]` and `[hip]`)consistent with the rules?
- Is the collision model adapted to the structure of the robot?
- Are the color markers of appropriate size, color and location?
- Is the upright posture valid?
- Is the extension posture valid?
- Is there any other reason why this model should not be accepted?

TC validation

In order to ensure reasonable performance of the simulation, teams will be required to upload a simple docker image with a walking behavior. The controller needs to connect to a virtual robot model on the field via the robot controller API. As soon as the connection was established, the controller should send the robot commands to let it walk for at least one minute. The robot can walk in any direction on the field as long as it stays within the boundaries of the

artificial turf. The provided controller is still considered valid if the robot is falling down if a reasonable attempt to create a walking behavior can be assumed from the robot's behavior.

The docker image needs to be uploaded to a container registry provided by the Technical Committee before the second review deadline. Team leaders will receive instructions and an access token to the registry by e-mail a few days prior to the second review deadline.

The robot controller and the corresponding behavior is used for benchmarking purpose. It needs to be possible to simulate two full teams (each with four robots in KidSize and two robots in AdultSize) using the provided robot model on the hardware used for the competition in the realtime factor provided by the Technical Committee. In case the robot is too complex teams will be contacted to take appropriate measures to reduce the complexity of the simulation.

KidSize teams will also be requested to provide a link to a YouTube video displaying the robot performing successful stand-up motions.

If teams request an update of their model files after the second review session, the Technical Committee will review the modifications on a case-by-case basis. All modifications made after May 23rd need to be properly justified by the team requesting the change.

Upload format

All robot models for a team must be submitted in a `.zip` archive.

The structure of the folder should be as follows:

```
.
|-- documentation
|   |-- documentation.pdf
|   |-- datasheet1.pdf
|   |-- ...
|-- RobotName1
|   |-- RobotName1.proto
|   |-- postures.json
|   |-- ...
|-- RobotName2
|   |-- RobotName2.proto
|   |-- postures.json
|   |-- ...
```

`...` indicates optional further datasheets or documents in the documentation folder and any other file requested of the robot model in the robot folders. Up to four robot models in KidSize and two robot models in AdultSize may be included in the submission.

Appendix

Values for classical dynamixel motors

The TC provides the following list of parameters that can be used for Dynamixel motors:

```
MX28-torque-12V              2.50
MX28-vel-12V                 5.76
MX28-damping-12V             0.30
MX28-friction-12V            0.83

MX28-torque-14.8V            3.10
MX28-vel-14.8V               7.02
MX28-damping-14.8V           0.31
MX28-friction-14.8V          1.03

MX64-torque-12V              6.00
```

```
MX64-vel-12V                  6.60
MX64-damping-12V              0.66
MX64-friction-12V             1.42

MX64-torque-14.8V             7.30
MX64-vel-14.8V                8.17
MX64-damping-14.8V            0.65
MX64-friction-14.8V           1.73

MX106-torque-12V              8.40
MX106-vel-12V                 4.71
MX106-damping-12V             1.26
MX106-friction-12V            2.14

MX106-torque-14.8V           10.00
MX106-vel-14.8V               5.76
MX106-damping-14.8V           1.23
MX106-friction-14.8V          2.55

XH540W270-torque-12V         10.60
XH540W270-vel-12V             3.14
XH540W270-damping-12V         2.95
XH540W270-friction-12V        1.23

XH540W270-torque-14.8V       12.90
XH540W270-vel-14.8V           3.87
XH540W270-damping-14.8V       2.92
XH540W270-friction-14.8V      1.49
```

Script to approximate motor parameters

In case you need to approximate motor parameters for other motors, you can reuse this script to compute their values.

```
import math

VAR_LENGTH = 30

class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

class JointSpecs:
    def __init__(self, name, max_tor_12V, max_tor_14_8V, max_vel_12V_rpm, max_vel_14_8V_rpm,
                 NT_curve_point_1, NT_curve_point_2):
        self.name = name
        self.max_tor_12V = max_tor_12V
        self.max_tor_14_8V = max_tor_14_8V
        self.max_vel_12V_rpm = max_vel_12V_rpm
        self.max_vel_14_8V_rpm = max_vel_14_8V_rpm
        self.NT_curve_point_1_12V = NT_curve_point_1
        self.NT_curve_point_2_12V = NT_curve_point_2

        # convert to rad/s
        self.max_vel_12V = self.max_vel_12V_rpm * math.tau / 60
        self.max_vel_14_8V = self.max_vel_14_8V_rpm * math.tau / 60
```

```
        self.NT_curve_point_1_12V.y = self.NT_curve_point_1_12V.y * math.tau / 60
        self.NT_curve_point_2_12V.y = self.NT_curve_point_2_12V.y * math.tau / 60

        # scale N-T points from 12V to 14_8V
        self.scale_factor_tor = self.max_tor_14_8V / self.max_tor_12V
        self.scale_factor_vel = self.max_vel_14_8V / self.max_vel_12V
        self.NT_curve_point_1_14_8V = Point(
            self.NT_curve_point_1_12V.x * self.scale_factor_tor,
            self.NT_curve_point_1_12V.y * self.scale_factor_vel)
        self.NT_curve_point_2_14_8V = Point(
            self.NT_curve_point_2_12V.x * self.scale_factor_tor,
            self.NT_curve_point_2_12V.y * self.scale_factor_vel)

    def get_values(self, use_14_8V):
        # choose correct values based on voltage
        if use_14_8V:
            stall_torque = self.max_tor_14_8V
            vel = self.max_vel_14_8V
            point_1 = self.NT_curve_point_1_14_8V
            point_2 = self.NT_curve_point_2_14_8V
        else:
            stall_torque = self.max_tor_12V
            vel = self.max_vel_12V
            point_1 = self.NT_curve_point_1_12V
            point_2 = self.NT_curve_point_2_12V

        vel_diff = (point_2.y - point_1.y)
        torque_diff = (point_2.x - point_1.x)
        a = vel_diff / torque_diff
        b = point_1.y - (point_1.x * a)

        # compute torque at vel=0
        # 0 = a*x+b -> x = -b / a
        torque_vel0 = -b / a
        friction = stall_torque - torque_vel0

        damping_constant = -torque_diff / vel_diff

        self.print_value("torque", stall_torque, use_14_8V)
        self.print_value("vel", vel, use_14_8V)
        self.print_value("damping", damping_constant, use_14_8V)
        self.print_value("friction", friction, use_14_8V)
        print("")

    def print_value(self, property_name, value, use_14_8V):
        v_text = "14.8V" if use_14_8V else "12V"
        var_name = (f"{self.name}-{property_name}-{v_text}").ljust(VAR_LENGTH)
        print(f"{var_name}{value:5.2f}")

    def print_all_values(self):
        self.get_values(False)
        self.get_values(True)

# max_tor_12V, max_tor_14_8V, max_vel_12V_rpm, max_vel_14_8V_rpm, NT_curve_point_1, NT_curve_point_2
JointSpecs("MX28", 2.5, 3.1, 55, 67, Point(0.112, 49),
           Point(1.288, 12)).print_all_values()
```

```
JointSpecs("MX64", 6.0, 7.3, 63, 78, Point(0.15, 64),
           Point(2.85, 25)).print_all_values()
JointSpecs("MX106", 8.4, 10, 45, 55, Point(0.7, 42),
           Point(5.6, 5)).print_all_values()
JointSpecs("XH540W270", 10.6, 12.9, 30, 37, Point(0.4, 29),
           Point(8.6, 2.5)).print_all_values()
```