

FIRA Smart Home Simulation League

Student Programming Guide (U14 / U19)

Student Edition | Draft v0.2 | smarthome_robot 0.1.6 | 17 October 2026

FIRA Smart Home Simulation League Technical Committee

Welcome! Let's Program a Smart Vacuum Robot

This practical guide is written for U14 and U19 students. You will make a virtual vacuum move, read its sensors, avoid obstacles and test cleaning strategies. Each example is based on the public Webots smarthome_robot Python helper, not the Chogan controller API.

IMPORTANT: Educational draft. The examples have been checked for Python syntax, but not run in the event's live simulator. The newer candidate rulebook may describe a separate two-robot contest; do not assume this helper is that contest's approved SDK.

1. Set Up Webots and the Python Library

The public simulation repository describes Webots R2025a and recommends Python 3.12. The league website recommends Python 3.10; the smarthome_robot package supports Python 3.8+. Your event organizers must publish one verified working combination. Install Webots, download github.com/smart-home-league/simulation and install the library in the Python environment selected inside Webots.

```
python -m pip install smarthome_robot==0.1.6
```

Open worlds/Smart-Home-U14.wbt in Webots and load the U14 example from the examples folder through the robot window. U19 teams should use the official U19 world and example. No hardware robot is needed.

2. First Goal: Make Your Robot Move

The helper class is RobotU14. Wheel commands are assignments to left_motor and right_motor; unlike the Chogan sample, these are not getDevice()/setVelocity() calls.

```
from smarthome_robot import RobotU14
```

```
robot = RobotU14(team_name="Learning Team")
```

```
SPEED = 8.0
```

```
elapsed = 0.0
```

```
while robot.step(robot.time_step) != -1:
```

```
    if elapsed < 2.0:
```

```
        robot.left_motor = SPEED
```

```
        robot.right_motor = SPEED
```

```

else:
    robot.left_motor = 0.0
    robot.right_motor = 0.0
    elapsed += robot.time_step / 1000.0

```

The example requests equal wheel speeds for about two simulation seconds, then requests zero. Confirm forward direction in the real world. The units and limits of wheel commands must be specified by the event SDK; 8.0 is only a conservative example.

3. Speed, Reverse and Turn

```

robot.left_motor = 0.0
robot.right_motor = 0.0 # Stop request

```

```

robot.left_motor = 8.0
robot.right_motor = 8.0 # Equal wheel speeds

```

```

robot.left_motor = -5.0
robot.right_motor = -5.0 # Reverse command

```

```

robot.left_motor = 6.0
robot.right_motor = -6.0 # Rotate in place

```

These are individual commands, not a complete program. Robot motion requires a working `robot.step(robot.time_step)` loop. A differential-drive robot generally goes straight when the wheels match and turns when they differ. Calibrate the sign convention before using left/right turn labels.

4. Understand the Distance Readings

The public helper exposes `distance_front_left`, `distance_front_right`, `distance_left` and `distance_right`. These four convenient properties do not rule out eight underlying Webots sensors described by the older league site. They are different abstraction levels. The public controller example treats a raw value below 0.5 as near an obstacle; your threshold requires calibration.

```

from smarthome_robot import RobotU14

```

```

robot = RobotU14(team_name="Learning Team")
while robot.step(robot.time_step) != -1:
    print(robot.distance_front_left,
          robot.distance_front_right,
          robot.distance_left,
          robot.distance_right)

```

Try it: record all four values in an empty area and near a wall. A raw reading is not automatically measured in centimetres.

5. Stop or Turn Before a Collision

```
from smarthome_robot import RobotU14
```

```
robot = RobotU14(team_name="Learning Team")
NEAR = 0.5
SPEED = 8.0
```

```
while robot.step(robot.time_step) != -1:
    blocked = (robot.distance_front_left < NEAR or
               robot.distance_front_right < NEAR)
    if blocked:
        robot.left_motor = -SPEED
        robot.right_motor = SPEED
    else:
        robot.left_motor = SPEED
        robot.right_motor = SPEED
```

Expected behavior: go forward while the front sensors report open space; otherwise rotate. This simple policy can turn endlessly in a corner. Add a recovery state in Lesson 8.

6. Use the Bumper Sensors

The helper supplies `bumper_left` and `bumper_right`. A nonzero reading can indicate contact. Bumpers are a backup—not a substitute for distance sensors.

```
from smarthome_robot import RobotU14
```

```
robot = RobotU14(team_name="Learning Team")
while robot.step(robot.time_step) != -1:
    hit = robot.bumper_left != 0 or robot.bumper_right != 0
    if hit:
        robot.left_motor = -6.0
        robot.right_motor = -6.0
    else:
        robot.left_motor = 6.0
        robot.right_motor = 6.0
```

The example may reverse for too short or too long depending on the bumper contact. A robust controller uses a fixed reverse interval followed by a bounded turn.

7. Wheel Encoders and Simulation Time

`left_encoder` and `right_encoder` report wheel rotation, not necessarily metres. Converting them to travel distance requires verified wheel radius and units. `robot.time_step` is a simulation step interval, conventionally measured in milliseconds.

```
from smarthome_robot import RobotU14
```

```
robot = RobotU14(team_name="Learning Team")
while robot.step(robot.time_step) != -1:
    print(robot.left_encoder, robot.right_encoder)
```

8. Stop Getting Stuck: a State Machine

Use three states: MOVE, BACK and TURN. Recovery phases should end after a bounded number of steps so the robot can try a new path.

```
from smarthome_robot import RobotU14
```

```
robot = RobotU14(team_name="Learning Team")
state = "MOVE"
remaining = 0
NEAR = 0.5
```

```
while robot.step(robot.time_step) != -1:
    close = (robot.distance_front_left < NEAR or
             robot.distance_front_right < NEAR)
    hit = robot.bumper_left != 0 or robot.bumper_right != 0
```

```
    if state == "MOVE" and (close or hit):
        state, remaining = "BACK", 12
```

```
    if state == "BACK":
        robot.left_motor, robot.right_motor = -6.0, -6.0
    elif state == "TURN":
        robot.left_motor, robot.right_motor = 6.0, -6.0
    else:
        robot.left_motor, robot.right_motor = 8.0, 8.0
```

```
    if remaining > 0:
        remaining -= 1
    if remaining == 0:
        if state == "BACK":
            state, remaining = "TURN", 14
        else:
            state = "MOVE"
```

This is a learning example; a fixed turn direction can still fail in complex rooms. Measure repeated collisions and change your strategy when recovery fails.

9. Lights and Robot State

The helper lists `led_on`, `led_play` and `led_step` as writable indicator properties. Their appearance depends on the simulated robot model; they do not earn points.

```
from smarthome_robot import RobotU14
```

```
robot = RobotU14(team_name="Learning Team")
robot.led_on = True
while robot.step(robot.time_step) != -1:
    pass
```

10. U19 GPS: Locate Your Robot

The public U19 helper provides `gps` as an `x, y, z` position tuple. These are simulator coordinates, not geographic latitude/longitude or guaranteed tile indices. The map origin and axis convention require confirmation.

```
from smarthome_robot import RobotU19
```

```
robot = RobotU19(team_name="U19 Learning Team")
while robot.step(robot.time_step) != -1:
    x, y, z = robot.gps
    print("position:", x, y, z)
```

11. U19 IMU: Read Orientation

The public U19 helper provides `imu` as roll, pitch and yaw. Check units, direction signs and the simulation coordinate frame before implementing heading control.

```
from smarthome_robot import RobotU19
```

```
robot = RobotU19(team_name="U19 Learning Team")
while robot.step(robot.time_step) != -1:
    roll, pitch, yaw = robot.imu
    print("orientation:", roll, pitch, yaw)
```

12. Battery and Charging: Wait for the Official API

Public league rules describe battery use and green charging areas for U19, but the published helper package's public example does not specify a `stable_charge()` method, charging packet, capacity or drain rate. DO NOT invent `robot.charge()`, hidden world-state access or a message format. The Technical Committee must release a runnable charging example and acceptance test before it is required.

13. Explore More of the House

Begin with a robust obstacle-avoidance controller, then try to detect repeated paths and find unexplored areas. U14 may need sensor- and encoder-based position estimates. U19 can combine GPS/IMU with its observed obstacles. Both must still handle unexpected changes to the map.

A visit is not always a score: the public Webots simulation evaluates cleanliness, while a proposed new rules version mentions tile ownership and reclaiming by an opponent. These

are NOT interchangeable formulas. Only the scored test cases from the actual event release decide the winner.

14. Practice and Compare Algorithms

Test two controller versions with the same Webots version, world/map, seed if applicable, division and clock limit. Repeat on unfamiliar maps. Record the official score, number of collisions, time spent stuck and any crashes. Make one change at a time.

15. Your Submission Folder

Do not copy Chogan's four-controller ZIP layout: Smart Home is a different league. In the public Webots repository the example is loaded as a robot controller. Final bundle type, main filename, archive limits, README format, allowed modules and delivery deadline must be published and tested by the organizer.

```
team_work/  
  robot.py  
  README.md  
  tests/  
  notes.txt
```

The tree above is an example project layout, not a binding ZIP submission requirement. Do not include private keys, network tokens or the protected supervisor implementation.

16. Five Learning Challenges

1. Move for 2 seconds, then stop. 2. Measure raw readings near different walls. 3. Stop or turn at a calibrated front threshold. 4. Recover after repeated bumper contact. 5. In U19, log GPS and IMU while following a route.

17. Common Errors and Fixes

ModuleNotFoundError: install the package into the Python interpreter Webots actually uses. No motion: verify the world is running, your controller is attached and step() returns updates. Endless spinning: add a bounded TURN state. Unreadable sensor values: calibrate rather than guessing units. Missing U19 attributes: check SDK version and which robot class was constructed.

18. Mini Dictionary

Controller: your robot decision code. SDK: the permitted software interface. Sensor: a value that reports observations. Bumper: contact detector. Encoder: wheel-rotation detector. IMU: orientation reading. GPS: world position coordinates. Scoring engine: the official mechanism that allocates points.

Appendix A. Quick API Reference

Imports: `from smarthome_robot import RobotU14, RobotU19`. Constructors: `RobotU14(team_name='...')` and `RobotU19(team_name='...')`. Simulation: `robot.step(robot.time_step)`. Motor setters: `robot.left_motor`, `robot.right_motor`. Readings: `distance_front_left`, `distance_front_right`, `distance_left`, `distance_right`, `bumper_left`, `bumper_right`, `left_encoder`, `right_encoder`. Indicators: `led_on`, `led_play`, `led_step`. Additional documented U19 readings: `gps` and `imu`.

Appendix B. Before the Committee Can Publish a Final SDK

Release gates: lock and test Python/Webots versions; ship the actual controller SDK and examples; define sensors, units, coordinate axes, allowed wheel speeds, cleaning trigger and scoring formula; document U19 charging and energy data; specify runtime limits, map formats, random seeds, submission structure, error handling and official replay logs. Syntax-checked examples do not equal validated Webots integration.

Appendix C. Reference Sources

Official website: <https://smarthomeleague.ir/> | Public simulation repository: <https://github.com/smart-home-league/simulation> | Public Python package: <https://pypi.org/project/smarthome-robot/> | Candidate PDF requiring direct inspection: <https://smarthomeleague.ir/downloads/SmartHomeLeague-Rulebook-EN.pdf>

End of Student Programming Guide — Draft v0.2; simulator verification pending.