Sensor and actuator reference for the foot-bot

The foot-bot is a differential-drive wheeled robot specifically designed to conduct experiments with robot swarms. The robot is endowed with sensors and actuators that enable it to interact with other foot-bots and with the environment. The foot-bot, its sensors and actuators are shown below.

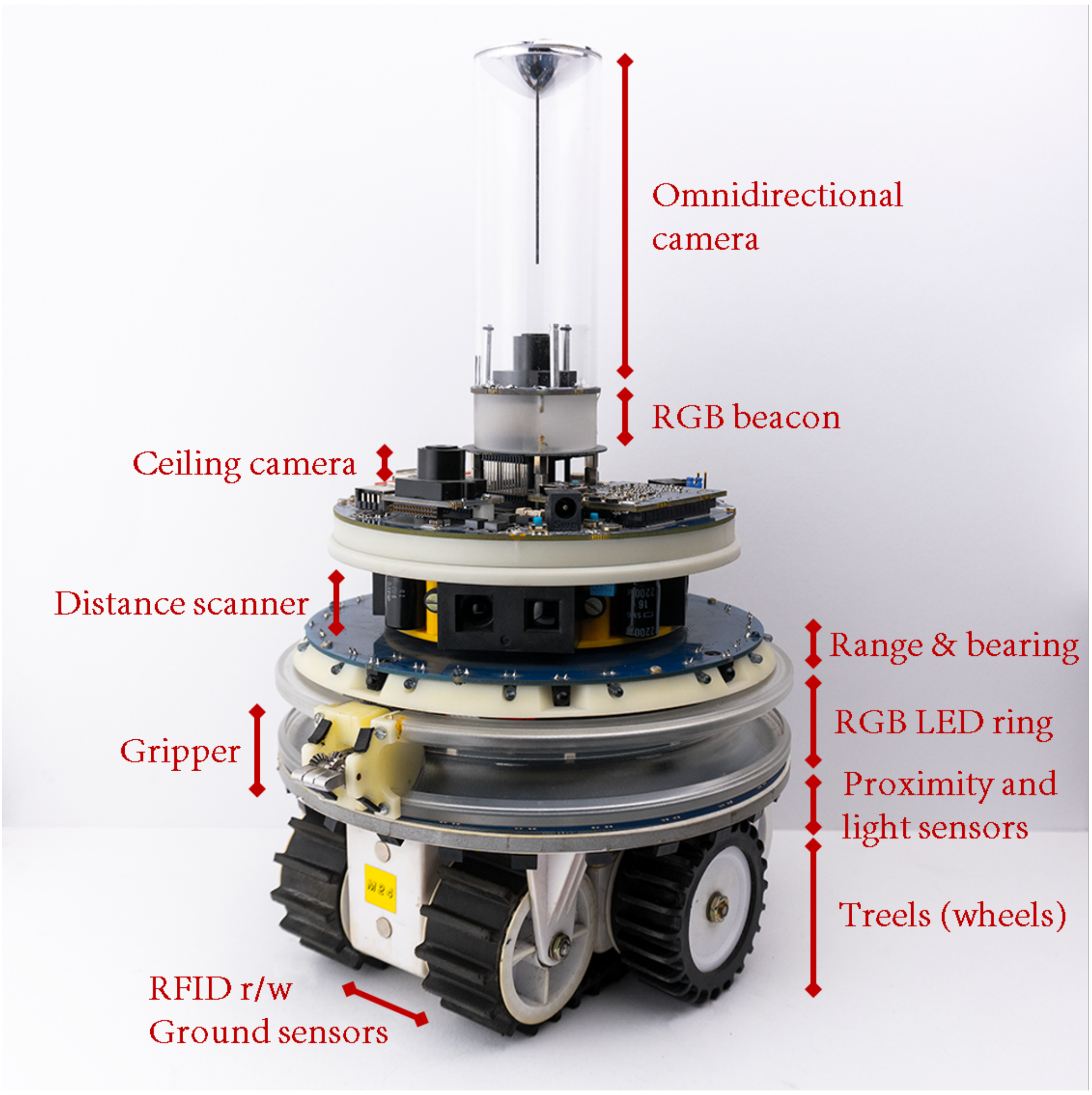


Figure 1: The foot-bot and its sensors and actuators. Picture by David Garzón Ramos.

Not all sensors and actuators will be used during each exercise. The exercise sheet will specify the sensors and actuators available to you.

Lua interface

The robot-related functions and data are stored in the robot table. For instance, to set the robot wheel speed, you need to call

```
robot.wheels.set_velocity(2,3)
```

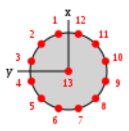
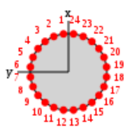
Analogously, to access the reading of the 4th proximity sensor and store it in a variable named r, you type

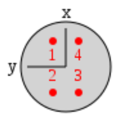
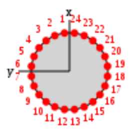
```
r = robot.proximity[4].value
```

Do not modify the values of the `robot` attributes! More specifically:

- Never write directly into the `robot` attributes:
  `robot.proximity[4].angle = 1.67 -- NO!`
- Never apply operations such as `table.sort()` to the `robot` table:
  `table.sort(robot.proximity, function(a,b) return a.value > b.value end) -- NO!`

Table 1: Reference for the sensors and actuators on the foot-bot

Sensor/actuator	Description	Placement
`robot.colored_blob_omnidirectional_camera`	This device returns a list of colored blobs, along with their position with respect to the robot center. A colored blob in ARGoS corresponds to an LED. The list of blobs varies in size over time, depending on what the robots sees. To know the number of messages received in a given time step, use `#robot.colored_blob_omnidirectional_camera`. To start collecting blobs, you need to call `enable()`. To stop, call `disable()`. Each blob is stored in a table composed of distance (the distance of the blob source in cm), angle (the angle between the robot local x axis and the position of the blob source; the angle is in radians), and color (a table composed of red, green, and blue, which are three RGB components of the blob's color in the range [0,255]). For example, to access the red component of the first blob seen by the robot you can write `robot.colored_blob_omnidirectional_camera[1].color.red`	
`robot.gripper`	The gripper allows a robot to connect to objects such as boxes and cylinders, or other robots. A robot attached to a passive object can transport it, if it is light enough. To lock the gripper, you use `lock_positive()` or `lock_negative()`. They both do the same job, so you can call the one you prefer. To unlock the gripper (thus releasing the attached object), use `unlock()`.	
`robot.id`	A string containing the id of the robot.	
`robot.leds`	Sets the color of the robot LEDs. The robot has a total of 13 RGB LEDs. 12 of them are arranged in a ring around the robot body, and one (also called the beacon) is positioned at the top of the robot body. To set the colors of a single LED, use `set_single_color(idx, color)`. idx is the number of the LED to set (1-12 for the body LEDs, 13 for the beacon). color can be expressed as a string, such as "red", "green", "blue", etc., or as a triplet of numbers r,g,b. To set all colors at once, use `set_all_colors(color)`. The color parameter works like `set_single_color(idx, color)`.	
`robot.light`	The light sensor allows the robot to detect light sources. The robot has 24 light sensors, equally distributed in a ring around its body. Each sensor reading is composed of an angle in radians and a value in the range [0,1]. The angle corresponds to where the sensor is located in the body with respect to the front of the robot, which is the local x axis. Regarding the value, 0 corresponds to no light being detected by a sensor, while values >0 mean that light has been detected. The value increases as the robot gets closer to a light source.	 x y 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24

Sensor/actuator	Description	Placement
`robot.motor_ground`	The motor ground sensor reads the color of the floor. It is a list of 4 readings, each containing a table composed of value and offset. The value goes from 0 or 1, where 0 means black, and 1 means white. The offset corresponds to the position read on the ground by the sensor. The position is expressed as a 2D vector stemming from the center of the robot. The vector coordinates are in cm.	
`robot.proximity`	The proximity sensors detect objects around the robots. The sensors are 24 and are equally distributed in a ring around the robot body. Each sensor has a range of 10cm and returns a reading composed of an angle in radians and a value in the range [0,1]. The angle corresponds to where the sensor is located in the body with respect to the front of the robot, which is the local x axis. Regarding the value, 0 corresponds to no object being detected by a sensor, while values >0 mean that an object has been detected. The value increases as the robot gets closer to the object. For example, to access the angle of the first proximity you can write `robot.proximity[1].angle` and to access the value of the first proximity you can write `robot.proximity[1].value`.	 x y 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
`robot.random`	This table offers a set of functions to draw random numbers from a distribution. Use bernoulli() to get either 0 or 1 from a Bernoulli distribution with p=0.5. You can also write bernoulli(p) to set a different value for p. Use exponential(m) to get a random number from an exponential distribution with mean m. Use gaussian(s) to get a random number from a Gaussian distribution with standard deviation s and zero mean. You can also write gaussian(s,m) to set a non-zero mean. Use uniform() to get a random number from a uniform distribution in the range [0,1). Alternatively, you can use uniform(max) to get a number between 0 and max, or uniform(min,max) to get a number between min and max. If you want integer numbers, use the functions `uniform_int(max)` and `uniform_int(min,max)`.	

Sensor/actuator	Description	Placement
`robot.range_and_bearing`	The range-and-bearing system allows robots to perform localized communication. Localized communication means that a robot, upon receiving data from another robot, also detects the position of the sender with respect to its local point of view. It is important to notice that the range-and-bearing system is not like WiFi. First, because two robots can exchange data only if they are in direct line of sight - if an object is between two robots, the robots can't communicate. Second, because robots that send data can only broadcast it in a limited area - you can't pick who you talk to as you would with an IP address. Third, the robots can exchange only 10 bytes of data. To set the data to broadcast, use `set_data()`. This function accepts input in two forms. You can write `set_data(idx, data)`, and this means that you set the idx-th byte to the value of data. data must be a number in the range [0,255]. Alternatively, you can write `set_data(data)`, where data must be a table containing exactly 10 numbers in the range [0,255]. At each time step, a robot receives a variable number of messages from nearby robots. To know the number of messages received in a given time step, use `#robot.range_and_bearing`. Each message is stored in a table composed of data (the 10-bytes message payload), `horizontal_bearing` (the angle between the robot local x axis and the position of the message source; the angle is on the robot's xy plane, in radians), `vertical_bearing` (like the horizontal bearing, but it is the angle between the message source and the robot's xy plane), and range (the distance of the message source in cm). For example, to access the first byte of data of the first message received you can write `robot.range_and_bearing[1].data[1]`. To access the range and the `horizontal_bearing` of the same message you can write `robot.range_and_bearing[1].range` and `robot.range_and_bearing[1].horizontal_bearing`.	
`robot.turret`	The foot-bot gripper is attached to a rotating device called the gripper. You can control the gripper by either setting its rotation, or its rotational speed. To set its rotation, you must first call the method `set_position_control_mode()` to switch the gripper to position control mode, and then call `set_rotation(angle)` to rotate the gripper at angle angle. Alternatively, you can set the rotational speed by calling `set_speed_control_mode()` first, and then `{set_rotation_speed(speed)}. With set_passive_mode()` you instruct the gripper to be in a state in which, as the robot moves with an object gripped, the turret rotates due to the weight of the gripped object.	
`robot.wheels`	The real robot moves using two sets of wheels and tracks called treels. For simplicity, we treat the treels like normal wheels. To move the robot, use `set_velocity(l,r)` where l and r are the left and right wheel velocity, respectively. By 'wheel velocity' we mean linear velocity. In other words, if you say `set_velocity(5,5)`, the robot will move forward at 5cm/s. You can get some information about robot motion and wheels, too. `axis_length` is the distance between the two wheels in cm. `velocity_left` and `velocity_right` store the current wheel velocity. `distance_left` and `distance_right` store the linear distance covered by the wheels in the last time step.