IT University of Copenhagen August 25, 2025

Advanced Topics in Robotics —Swarm Robotics

Simple Kinematic Simulator for Swarm Robotics

Payam Zahadat
paza@itu.dk

Introduction

This is a kinematic (non-physics) simulator designed to provide a simplified but realistic environment for testing multi-robot behaviors. Many real-world complexities have been abstracted away to help you focus on algorithm design. Keep in mind, however, that for an algorithm to work on real robots, one must consider numerous challenges related to physics, sensing, communication, and noise.
Each robot uses differential-drive motion and is equipped with basic sensors.
You will implement your control algorithm in a dedicated function, using only high-level control commands—as you would when programming a real robot.

Before you begin: Make sure you can run the simulator.

- Requires Python 3.11 or higher, and the packages `pygame` and `numpy`.
- Install Python and the required libraries.
- Run the simulator to ensure everything works.
- Press `P` to pause/resume the simulation.
- Press `Space` to toggle visualization (headless mode).

Note: If you spot any bugs or errors in the simulator, please let me (paza@itu.dk) know.

Robot Model

Each robot is modeled as a differential-drive robot, similar to an e-puck, Thymio, or the robots you built in AI Robotics course.

- **Control Interface:** At every control step, you specify a target rotation angle and a target speed using:

  ```
  robot.set_rotation_and_speed(delta_bearing, target_speed)
  ```

  A proportional controller (P-controller) is implemented inside this function to rotate the robot toward the desired relative rotation.

- **Do not** set `_pos`, `_heading`, `_linear_velocity` or `_angular_velocity`, directly or via `_set_velocity()`.

Sensors

Each robot has four types of sensors:

1. Proximity Sensors

- Simulate infrared or ultrasonic sensing.
- Arranged radially around the robot, covering 360°.
- Each sensor returns the distance to the nearest detected object (if any), and the type: `"wall"`, `"obstacle"`, or `"robot"`.
- Data is available as:

  `robot.prox_readings`

  which is a list of sensor readings ordered clockwise around the robot.

2. RAB Sensors

RAB stands for **Range and Bearing**, and refers to a common sensing and communication modality in swarm robotics. It enables robots to detect nearby peers, estimate their relative distance and direction (bearing), and receive small data payloads from them. This is essential for coordination behaviors like flocking, dispersion, or aggregation. In small-scale robots, RAB is often implemented using **infrared (IR)**: each robot broadcasts a short message while nearby robots detect the angle and strength of the incoming signal using a ring of IR sensors. In larger or more advanced robots, similar functionality is achieved using tools like cameras with visual markers, LIDAR, UWB radios, or GPS-based localization—though they may not be referred to as RAB modules. In this simulator, each robot receives a list of RAB signals from nearby neighbors. Each signal includes:

- `distance`: estimated distance to the sender
- `bearing`: direction to the sender (relative to the robots heading), inferred from the orientation of the receiving sensor.
- `message`: a small message (e.g., the sender's heading)
- `intensity`: signal strength value that decreases with distance
- `sensor_idx`: the index of the receiving sensor

The list of received signals is stored in:

`robot.rab_signals`

Each element is a dictionary:

```
{
  'distance': float,
  'bearing': float,            # radians, relative to robot's heading
  'message': {'heading': ...},
  'intensity': float,          # decays with distance (e.g., ~1 / (d^2 + 1))
  'sensor_idx': int        # the index of the receiving sensor
}
```

3. Light Sensor

Returns the ambient light intensity at the robots location (no directional information).

4. IMU Sensor

Returns the orientation (heading) of the robot. In reality, an IMU (Inertial Measurement Unit) provides acceleration and angular velocity, and orientation is usually obtained from sensor fusion.

Environment

The robots operate in a 2D rectangular arena that is completely surrounded by walls. These walls are detected by the proximity sensors in the same way as obstacles.
You may also add other elements that are already implemented in the simulator to your environment:

- **Obstacles:** Static circular obstacles can be placed in the arena. Robots detect them using proximity sensors.
- **Light Sources:** Light sources can be added to the arena, and robots can sense them using a light sensor. Each light source has a core radius where intensity is maximal, and a defined range where intensity decreases with distance from the center, dropping to zero beyond that range.

Noise Models

To make the simulation more realistic, the simulator includes optional models for imperfections in sensing, communication, and motion. These imperfections can help test the robustness of your algorithms under real-world conditions.
Currently, three types of noise are implemented. Their parameters are set to zero by default, meaning no noise is applied. You are encouraged to experiment with them to evaluate the robustness of your behavior under more realistic conditions. You may also implement additional types of noise if relevant to your experiments.

- **Communication Noise**:
  - **RAB bearing noise** (`RAB_NOISE_BEARING`): Adds angular noise (in radians) to the estimated direction of received RAB signals. This may cause errors in estimating the direction of the other robots.
  - **RAB message dropout** (`RAB_DROPOUT`): Introduces a probability that an incoming RAB message will be randomly dropped, resulting in missed detections of other robots and loss of their transmitted messages.
- **Motion Noise**:
  - **Position noise** (`MOTION_NOISE_STD`): Adds random displacement to the robot's position after each movement step (in pixels).
  - **Heading noise** (`HEADING_NOISE_STD`): Adds small angular noise to the robot's heading update (in radians).

- **Light and Orientation Sensor Noise**:
  - `LIGHT_NOISE_STD` A multiplicative Gaussian noise model is applied directly in the light sensing function to simulate variation in ambient light detection.
  - `ORIENTATION_NOISE_STD` Adds angular noise (in radians) to the robot's actual heading.

Control Function

Implement your control algorithm inside this function:

```
def robot_controller(self)
```

Allowed:

- Use `self.rab_signals`, `self.prox_readings`, `self.light_intensity`, `self.orientation` to read the sensors.
- Use only `self.set_rotation_and_speed(...)` to control motion.

Forbidden:

- Do not modify internal variables like `self._linear_velocity`, `self._angular_velocity`, `self._pos`, or `self._heading`.
- Do not call internal methods like `_set_velocity()`.