The Kilobot Gym

Gregor H.W. Gebhardt[1] and Gerhard Neumann[2]

***Abstract*— Simulation is a crucial tool when learning control policies for robotic systems. The evaluation of a control policy is a recurring task in most learning algorithms which can be significantly sped up when using a simulation instead of the real system. This improvement in learning speed becomes even more significant when working with robot swarms which usually operate rather slow and need to be tediously initialized by hand. In this paper, we present a novel simulator for swarm robotics. Our simulator is inspired by the Kilobot platform and builds on the OpenAI gym. This allows to evaluate a wide range of learning algorithms using a unified interface for controlling the swarm. The code of our simulation framework is available at [3].**

I. INTRODUCTION

Learning the parameters of a control policy is usually an iterative process of evaluating the parameters on the system and improving the parameters based on the evaluation results. In robotics, the most time consuming task is often the evaluation of the parameters on a real robotic system. Using a simulation of the system can speed up this process by several orders of magnitude. In swarm robotics this improvement is even more significant as each evaluation requires to bring the system into some initial state. For a swarm of robots this task would be especially cumbersome compared to a single system which can often do the initialization on its own. In this paper, we present a new simulation framework for swarm robotics inspired by the Kilobot platform [7]. Our simulation environment is based on the *OpenAI gym* [1] which aims to provide a wide range of benchmark tasks for policy learning algorithms. By using a unified interface for all environments, the gym framework facilitates the repeatability and reusability of research results. Our simulation framework is implemented in Python and thus easy to use with a wide range of machine learning libraries. We use the 2D physics library *Box2d* [2] for simulating the movements and the game engine *pygame* [9] for visualization. The simulation of the Kilobot robots in our framework borrows from the simulation of the movements in Kilombo [4], a C-based simulator specifically targeted for the Kilobot platform. It was written with the aim to simulate the movements of the Kilobots and the noise in the IR communication very realistically. The experiments in [8] were conducted on a simulator which is also based on Box2d. The Kilobots are simulated as discs that get an impulse in the direction of the goal state at each time step. The implementation uses JavaScript embedded in HTML files which results in code that is hard to use for programmatic evaluations and, furthermore, the reusability of JavaScripts without any modularity goes towards zero. Another approach to simulate a swarm of Kilobots is to use the robot simulator V-REP [6]. The simulations are performed in 3D, where the user can select between different physics engines. However, as the movement simulations are performed via physical interactions of the Kilobot legs with the floor, the simulation speed reduces significantly with the number of agents in the environment. This leads already for a small number of Kilobots to poor performances. The swarm simulator ARGoS [5] uses a multi-physics approach, i.e., it allows to apply multiple physics engines in the same environment depending on the region in which the robot operates.

[1]Gregor H.W. Gebhardt is with the Computational Learning for Autonomous Systems (CLAS) group, Institut für Informatik, Technische Universität Darmstadt, Germany gebhardt@ias.tu-darmstadt.de

[2]Gerhard Neumann is with the Lincoln Centre for Autonomous Systems (L-CAS), School of Computer Science, University of Lincoln, UK gneumann@lincoln.ac.uk

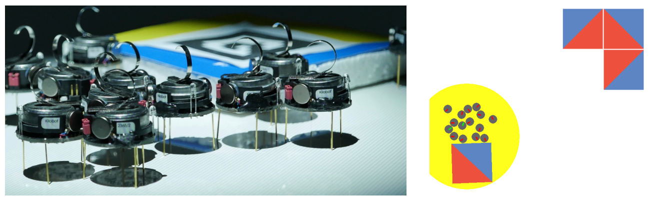

Fig. 1. Left: a scene with a small swarm of Kilobots with an object in the background. Right: a swarm of Kilobots in our simulation framework with with four square objects.

II. PRELIMINARIES

Our simulation framework is inspired by the Kilobot platform and based on the OpenAI gym simulation framework. We will shortly introduce both in the following sections.

A. The Kilobots

The Kilobots are an affordable and open source platform developed specifically for the evaluation of collective algorithms on large swarms of robots. Each disc-shaped robot is approximately 3 cm in diameter and 3 cm tall, examples are depicted in Figure 1. The body consists of a circuit board with a battery on top and three rigid legs. The legs are mounted equiangular, where one leg is at the front of the Kilobot and two at the rear. Using two vibration motors mounted above the two rear legs, the Kilobots can move up to 1 cm/s. This locomotion technique, which is based on the slip-stick principle, restricts the Kilobots to flat surfaces with low friction. When activating just one of the motors, the Kilobot rotates around the opposing rear leg at about $45°$/s. By setting appropriate values for both motors the Kilobot can move similar to differential drive vehicles at the full range between only rotating in any direction to moving straight forward. Additionally, the Kilobots can sense their ambient light and communicate with other Kilobots in their vicinity using IR transmitters.

B. OpenAI Gym

The OpenAI gym [1] is a collection of simulation environments for benchmarking reinforcement learning algorithms and with the aim to make research in the field of machine learning more reproducible. The gyms have a standardized interface which makes them easily interchangeable. The idea is to decouple the reinforcement learning from the simulation environment. The functions `reset()` and `step(action)` hide the simulation functionality, where both functions return the observation of the current state and `step` additionally returns the reward that was obtained for the given action as well as information about the state of the simulation. Additionally, the environment can be visualized by calling the function `render()`. The variables `observation_space` and `action_space` in the gym environment provide the learning algorithm with information about which kind of observations it can expect and what kind of actions are accepted by the environment.

The OpenAI gym comes with a wide range of implemented environments based on different simulation engines. Examples are Atari video games, a lunar lander based on Box2d, classical control problems such as the cart-pole or a pendulum, or more complex environments based on MuJoCo. To the authors knowledge there exist to date no other gym environment that targets a robot swarm setup.

III. THE KILOBOT GYM

The Kilobot Gym is a simulation framework based on the OpenAI gym which allows to evaluate reinforcement learning algorithms for swarm robotics. In its current state, the Kilobot Gym is designed such that the individual robots have a simple logic, e.g. a phototactic behavior, and the complex control of the swarm is done via a global control input, e.g. changing the direction of a light source. An extension for learning the policies of the individual robots would be straight forward. However, as for now we have chosen to stay close to the real Kilobot platform which is not designed for this learning setup (i.e., exchanging information with individual Kilobots is not possible during execution).

A. The Library

We have implemented a library which features each of the entities that we could also find at the Kilobot setup in a lab: a table which is the environment for the Kilobots, objects to manipulate and light sources. We will explain these components in the next paragraphs.

1) The Environment: The `KilobotsEnv` is the base class for creating environments in which we want to test our learning approaches. The `KilobotsEnv` has a table: a rectangular shape of a certain width and height to which the movements of all Kilobots and objects are bounded. Via the method `_configure_environment`, we can add Kilobots, objects and a light source to the table. Furthermore, we need to implement `get_reward(state, action, new_state)` which defines how well we value a transition from a given `state` to a `new_state` caused by a certain `action`.

2) The Kilobots: The `Kilobot` class is a base class for implementing the behavior of the agents. The base class features a similar API as the Kilobot c-library for the real robots. The ambient light can be measured via the function `get_ambientlight`, where the actual measurement is calculated by the light source of the environment. The movement of the Kilobot can be controlled with the function `set_motors` and a color which will be used in the visualization can be set via `set_color`. Additionally, we provide the auxilliary functions `turn_left`, `turn_right`, and `switch_direction`. The behavior needs to be implemented in the abstract methods `_setup` and `_loop`. Besides this abstract base class, we have implemented a `PhototaxisKilobot`, which has a phototactic behavior similar to the algorithm described in [7]. The `SimplePhototaxisKilobot` is a much more simplified version of the `PhototaxisKilobot` as it takes directly the gradient to the light source to compute its movement direction. This latter implementation follows more the behavior of nano-robots which move along the lines in a magnetic field.

3) Objects: To let the Kilobot swarm interact with objects in their environment, we have created a set objects, which is easily extendible to more shapes. In our library, we have objects with the following basic shapes: `Quad`, `Circle`, and `Polygon`. From the base class `Polygon`, we have furthermore implemented the shapes `Triangle`, `LForm`, `TForm`, and `CForm`, where the latter three are interesting for assembly tasks. All objects are parameterized by their width and height.

4) Light: The light source is the common input signal for the Kilobot swarm. In our simulation environment, we provide an abstract base class `Light` that defines a common interface for all light sources. This interface comprises the methods `step(action)` which executes the passed action on the light, e.g., moving the light source, `get_value(position)` and `get_gradient(position)` compute the value and gradient, respectively, at the given position, and `get_state()` returns the state of the light. Furthermore, we have implemented two light classes. First, a `GradientLight`, which simulates a uniform gradient on the whole table and accepts as actions the angle to which direction the gradient should point. And second, a `CircularGradientLight` which simulates a circular gradient of a certain radius that can be moved around the table. Here, the actions are the translations of the center position of the circular gradient. A `CompositeLight` allows arbitrary combinations of light sources.

B. Physical Simulation

We simulate the physics using the 2D simulation framework Box2d [2]. During the initialization of the

TABLE I

PHYSICAL PROPERTIES OF THE SIMULATED OBJECTS AND ROBOTS.

	Body	Kilobot
Density	2.0	1.0
Friction	0.01	0.2
Restitution	0.0	0.0
Linear Damping	8.0	0.8
Angular Damping	8.0	0.8

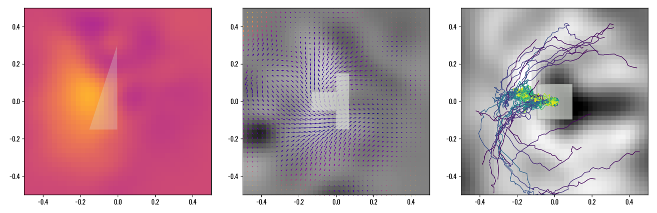

Fig. 2. Exemplary of a plot with different objects from the Kilobot Gym framework.

KilobotEnv, a Box2d world is instantiated and the table is added as a rectangular static Box2d body to define the bounds of the environment. All of the classes in the library, get a reference to this world object during creation such that they can add themselves with their physical properties. The physical properties of the non-actuated objects and the Kilobots in the library are depicted in Table I. Each object and Kilobot creates a Box2d body with these properties and adds an appropriate fixture depending on their shape. For each step of the gym environment, we then perform one update step on the Light, e.g. changing the position, and ten steps of the physical simulation. We use ten velocity and then position iterations for stepping the Box2D world.

C. Graphical Visualization

The Kilobot Gym is visualized using pygame [9]. A wrapper class KilobotsViewer hides details of the implementation such as scaling of the world size to the screen size. This wrapper class provides functions for drawing lines, circles and polygons. The specific code for visualizing objects and Kilobots is implemented in the respective classes. For example, a Kilobot is visualized as circle with a colored line on top indicating the status of the color LED and the direction of the Kilobot. An exemplary visualization of a Kilobot Gym is depicted in Figure 1.

Additionally to the visualization using pygame, we added functionality for plotting directly into matplotlib Axes objects. This functionality has proven to be very helpful during first applications of the Kilobot Gym in policy learning experiments. Exemplary plots with different objects of the library are depicted in Figure 2.

IV. CONCLUSTION & FUTURE WORK

We have presented the Kilobot Gym a new simulator for swarm robotics inspired by the Kilobot platform. Our simulation framework is based on the OpenAI gym which makes it reusable for evaluating many existing implementations of learning algorithms. The choice of Python is furthermore useful as it facilitates the use of our simulation environment in combination with many existing learning frameworks. While the Kilobot Gym does not feature parallel executions inherently, we have been able to add this feature in our experiments externally by implementing a sampler class that manages multiple instances of the gym across multiple processes. We plan to include this feature later into our framework. Another missing feature is the communication between the Kilobots. As we are currently not using this ability of the Kilobots, we did not yet consider to implement this feature into the simulation. However, we do not see big hurdles for adding this functionality.

REFERENCES

[1] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv:1606.01540 [cs]*, 2016.

[2] E. Catto, "Box2d is a 2d physics engine for games," 2018. [Online]. Available: https://github.com/erincatto/Box2D

[3] G. Gebhardt, "gym-kilobots: The kilobots gym," 2018. [Online]. Available: https://github.com/gregorgebhardt/gym-kilobots

[4] F. Jansson, M. Hartley, M. Hinsch, I. Slavkov, N. Carranza, T. S. G. Olsson, R. M. Dries, J. H. Grönqvist, A. F. M. Marée, J. Sharpe, J. A. Kaandorp, and V. A. Grieneisen, "Kilombo: a kilobot simulator to enable effective research in swarm robotics," *CoRR*, 2015.

[5] C. Pinciroli, V. Trianni, R. OGrady, G. Pini, A. Brutschy, M. Brambilla, N. Mathews, E. Ferrante, G. Di Caro, F. Ducatelle, M. Birattari, L. M. Gambardella, and M. Dorigo, "Argos: a modular, parallel, multi-engine simulator for multi-robot systems," *Swarm Intelligence*, vol. 6, no. 4, pp. 271–295, 2012.

[6] E. Rohmer, S. P. N. Singh, and M. Freese, "V-rep: a versatile and scalable robot simulation framework," in *Proceedings of The International Conference on Intelligent Robots and Systems*, 2013.

[7] M. Rubenstein, C. Ahler, N. Hoff, A. Cabrera, and R. Nagpal, "Kilobot: A low cost robot with scalable operations designed for collective behaviors," *Robotics and Autonomous Systems*, vol. 62, no. 7, pp. 966–975, 2014.

[8] S. Shahrokhi and A. T. Becker, "Object manipulation and position control using a swarm with global inputs," *2016 IEEE International Conference on Automation Science and Engineering (CASE)*, pp. 561–566, 2016.

[9] L. Theden, R. Dudfield, and T. Kluyver, "Pygame." [Online]. Available: https://www.pygame.org