FULL PAPER

Learning Policies for Object Manipulation with Robot Swarms

Gregor H. W. Gebhardt[a], Kevin Daun[a] and Marius Schnaubelt[a] and Gerhard Neumann[b]

[a]Technische Universitat Darmstadt, Hochschulstr. 10, 64289 Darmstadt, Germany; [b]LCAS, University of Lincoln, Brayford Pool, LN6 7TS, Lincoln, UK

ARTICLE HISTORY
Compiled October 22, 2018

ABSTRACT
The manipulation of objects with a swarm of simple agents is a challenging task. The simplicity of the robots usually prohibits to learn such manipulations skills directly on the agents. In this paper, we assume a swarm that is guided by a common input signal such as the gradient of a light source. We propose a new method for learning control policies for such input signals that lead to the desired object manipulation by the swarm. Therefore, we introduce a representation of the swarm which is based on Hilbert space embeddings of distributions. This representation is invariant to the number of agents and to the allocation of the agents to the agent positions. Furthermore, the learned policy are robust to changes in the swarm and the search space for the learning method is significantly reduced. We learn control policies for several object shapes and show to which extend a policy learned on a simple shape generalizes to more complex shapes. We further, apply the learned control policies in a hierarchical controller to solve the task of autonomous object assembly. Finally, we demonstrate that the policies learned in simulation are robust enough to be transferred directly to the real robots.

KEYWORDS
swarm robotics; multi-agent systems; policy search; reinforcement learning

1. Introduction

An important characteristic of swarms is that the abilities of the collective are usually much larger than the abilities of the individual. Examples are large structures built by ants or termites or the defense behavior of fish schools or bees. This synergy effect which is also called superadditivity ('the entire team should be able to achieve much more than individual robots working alone' [1]), is a main principle that swarm robotics aims to exploit. The field of nano-robotics which makes use of biological or synthetic molecular machines is one particularly interesting area of research. An example for such biological molecular machines are flagellated bacteria, i.e., bacteria that are propelled by a filament attached to a rotating molecular motor [2]. While the flagella motors provide propulsion, magnetotactic bacteria, for example, have a chain of ferromagnetic particles in their body that serves as an interface for controlling their orientation using a weak magnetic field [3].

In contrast to traditional robotics, which usually is based on robust machines with sufficient sensory equipment, swarm robots are usually simple agents with limited actuation and sensors. Instead, robotic swarms leverage from the high redundancy and the distributed nature of their hardware. Because state-of-the-art learning algorithms usually rely on computational powerful machines, their application to learn a behavior directly on swarm robots is limited. Learning

CONTACT G. Gebhardt. Email: gebhardt@ias.tu-darmstadt.de

a policy that externally guides a swarm to achieve a complex behavior is a feasible way to overcome this limitation. In this paper, we consider the task of autonomous object assembly with robot swarms. A swarm of Kilobots [4] has been used in an object assembly experiment [5], where a human operator controls a light source as global input signal to the swarm. Formulating the control rules to automate the assembly process is, however, a hard task.

Motivated by this application, we present an approach based on policy search to find a control strategy for the external signal. We split the assembly process in two subtasks: generating a top-level assembly plan using simple planning strategies, and learning a low-level object movement policy. The assembly plan encodes way points for each object while the object movement policy controls the trajectory execution by guiding the Kilobots with the light source. In this study, we treat the assembly plan as given and only learn object movement policies through policy search.

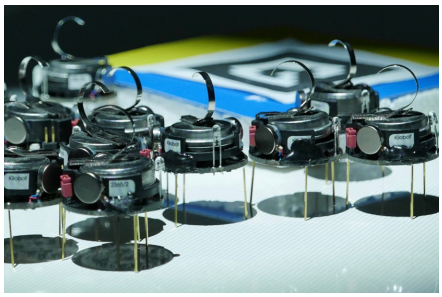

Figure 1. Kilobots pushing an object in an assembly task. The robots have a diameter of 3.3cm, the object in the background is a square of 15cm width.

Learning to push an object is a complex task as we have to coordinate a large number of agents, which results in many state variables. While we need information about the configuration of the swarm (e.g., the positions of the agents) in our state representation, it is of no importance which individuals of the swarm are at which positions. Furthermore, our policy should be independent of the number of agents participating in pushing the object. Hence, instead of representing the state by the positions of every agent, we represent it as a distribution over the agent locations which we embed into a reproducing kernel Hilbert space [6]. This allows us to compare the swarm configurations independently of the number of individuals and of their specific locations. Our reinforcement learning algorithm is based on the recently introduced actor-critic relative entropy policy search (AC-REPS) algorithm [7] and learns a non-parametric Gaussian Process (GP) policy for controlling the light source. We evaluate our approach in simulation on different assembly tasks with different object shapes. Additionally, we demonstrate that the learned policies can be transferred to the real Kilobots which shows that the learning process is robust enough to allow a direct transfer from simulation to the real world. Parts of this work have been presented by the authors in [8,9].

2. Related Work

While swarm robotics have been studied over the last three to four decades, using machine learning techniques to control robot swarms is a very recent field of research. In contrast to our approach, related work often directly learns the policies of the agents instead of a policy for a common control signal. For example, [10,11] both learn actor and critic functions based on feature mappings using fuzzy-nets. The authors assume that the state is fully observable to the critic and the actor. In contrast, [12] proposes a multi-agent learning approach based on deep Q-learning in which only the critic has access to the full information about the state, while the actor has local observations. In [13], a method for multi-robot learning based on particle swarm optimization is presented. Each robot acts as a particle that rolls out a certain set of controller parameters. After each iteration the best performing parameters are shared with the robots in the neighborhood. The particle swarm approach is furthermore compared to genetic algorithms in [14]. However, this approach requires that each agent is able to asses the quality of the action it has performed and communicate the result with its neighbors.

The significant difference of our method is that we do not learn the policies for the individual agents. Instead, we assume that the same behavior runs on each agent and a desired swarm behavior is achieved by learning the controller for the global input. This setup—simple policy on the robots, complex control using an external signal—allows to use a much simpler hardware for the agents. The Kilobots, for example, can only sense the ambient light and communication is limited to robots in the close neighborhood. Evaluating a policy on the agent or communicating values between a global critic and the agents would be very difficult. In [15], a swarm of flagellated magnetotactic bacteria was used to to build a pyramid of building blocks in the micrometer range. The bacteria have a flagatella-based propulsion motor and their direction is controlled by a magnetic field that acts on nanoparticles in the cellular body. In the pyramid-building experiment the magnetic lines are controlled according to a planned trajectory to move the swarm. In [16] a control method for a swarm of phototactic agents is presented. The paper proposes a set of different PD controllers for manipulating an object with different goals (i.e., rotating, or translating the object, or a combination of both).

3. Preliminaries

This section provides a short discussion of the policy search method we use for learning the low-level object movement policy and gives a quick overview of the embeddings of distributions which we use to represent the state of the swarm. Additionally, we will depict the planning strategies for generating swarm and object trajectories from the high-level assembly policy.

3.1. Actor-Critic Relative Entropy Policy Search

We use actor-critic relative entropy policy search (AC-REPS) [7] to learn a continuous, non-parametric, probabilistic policy $\pi(\boldsymbol{a}|\boldsymbol{s})$ from samples. This model-free reinforcement learning algorithm is based on relative entropy policy search (REPS) [17] and consists of three steps:

(1) Least-squares temporal policy iteration (LSPI) is used to estimate the Q-function from the observed samples [18–20].
(2) REPS optimizes the policy with respect to the expected Q-function while staying close to the old policy [17]. For the sample-based case this results in a discrete distribution over the state-action samples.
(3) A sparse weighted Gaussian process learns the new continuous policy by regressing from states to actions using the weights obtained from REPS [21].

Additionally, these three steps can be iterated until convergence of the resulting policy for a given SARS sample set similar to a policy iteration approach. In the next paragraphs, we will discuss these three steps in more detail. Note that they assume some feature mapping of either a state or of a state-action pair. While AC-REPS is applicable to a wide range of problems, naturally, the feature mapping is very specific to the problem domain. We will discuss the feature mapping that we have used for the domain of swarm robotics later in Section 4.1.3.

3.1.1. Least-Squares Policy Iteration

We want to estimate the Q-function from a data set of SARS tuples $\mathcal{D} = \{(\boldsymbol{s}_t, \boldsymbol{a}_t, r_t, \boldsymbol{s}'_t)\}_{t=0}^{T}$ sampled from the environment. These tuples consist of the state $\boldsymbol{s}_t$ and the action $\boldsymbol{a}_t$ taken by the agent at time t which results in a reward r_t and the transition to the next state $\boldsymbol{s}'_t$. Given a feature mapping $\boldsymbol{\phi}(\boldsymbol{s}, \boldsymbol{a})$, the Q-function can be approximated as a linear function in feature

space $Q(\boldsymbol{s}, \boldsymbol{a}) = \boldsymbol{\phi}(\boldsymbol{s}, \boldsymbol{a})^\intercal \boldsymbol{\theta}$, where $\boldsymbol{\theta}$ are the parameters of the function. We write the features of the state-action pairs as matrix $\boldsymbol{\Phi} = [\boldsymbol{\phi}(\boldsymbol{s}_0, \boldsymbol{a}_0), \ldots, \boldsymbol{\phi}(\boldsymbol{s}_T, \boldsymbol{a}_T)]^\intercal$ and the rewards r_t as vector $\boldsymbol{R} = [r_0, \ldots, r_T]^\intercal$. We can obtain the parameters $\boldsymbol{\theta}$ as [18,19]

$$\boldsymbol{u} = \left(\boldsymbol{\Phi}^\intercal (\boldsymbol{\Phi} - \gamma \boldsymbol{\Phi}')\right)^{-1} \boldsymbol{\Phi}^\intercal \boldsymbol{R} \tag{1}$$

$$\boldsymbol{\theta} = \left(\boldsymbol{\Phi}^\intercal \boldsymbol{\Phi}\right)^{-1} \boldsymbol{\Phi}^\intercal \left(\boldsymbol{R} + \gamma \boldsymbol{\Phi}' \boldsymbol{u}\right) \tag{2}$$

with discount factor γ and the feature matrix $\boldsymbol{\Phi}'$ that consists of the expected feature mappings $\mathbb{E}_{a'_t \sim \pi(s'_t)}[\boldsymbol{\phi}(\boldsymbol{s}'_t, a'_t)]$ with the next states $\boldsymbol{s}'_t$ from the SARS samples and the policy π. In practice, estimating the expected feature mapping with a single sample from the policy or, in case of a Gaussian process policy, with the mean action is usually sufficient. Here, Equation 1 is the fixed-point equation of the Bellman operator and Equation 2 is an orthogonal projection into the space spanned by the features.

However, as this solution often overfits to 'the noise of the system rather than the underlying system itself', [20] propose to regularize both, the fixed-point equation and the orthogonal projection. Using l_2 regularization on both terms, they arrive at

$$\boldsymbol{\theta} = (\boldsymbol{X}^\intercal \boldsymbol{X} + \beta' \boldsymbol{I})^{-1} \boldsymbol{X}^\intercal \boldsymbol{y}, \tag{3}$$

$$\boldsymbol{X} = \boldsymbol{\Phi} - \gamma \boldsymbol{\Sigma} \boldsymbol{\Phi}', \tag{4}$$

$$\boldsymbol{y} = \boldsymbol{\Sigma} \boldsymbol{R}, \tag{5}$$

$$\boldsymbol{\Sigma} = \boldsymbol{\Phi} (\boldsymbol{\Phi}^\intercal \boldsymbol{\Phi} + \beta \boldsymbol{I})^{-1} \boldsymbol{\Phi}^\intercal. \tag{6}$$

Here, $\boldsymbol{\Sigma}$ is the orthogonal projection into the space spanned by the features, regularized by β, and Equation 3 is the fixed-point equation of the Bellman operator regularized by β'.

3.1.2. Policy Improvement

In the policy improvement step, we want to optimize the policy such that the Q-function is maximized. However at the same time, large changes in the policy might lead to loss of information [17]. Inspired by the episodic REPS algorithm [22], AC-REPS solves this problem by using information-theoretic constraints. AC-REPS updates the policy by optimizing the expected Q-values of the samples with the constraint that the new policy $\pi(\boldsymbol{a}|\boldsymbol{s})$ is close to the old policy $q(\boldsymbol{a}|\boldsymbol{s})$ in terms of the Kullback-Leibler divergence (KL). Assuming a state distribution $\mu(\boldsymbol{s})$, we want to maximize the expected Q-value

$$\mathbb{E}_{\mu,\pi}[Q(\boldsymbol{s}, \boldsymbol{a})] = \int \mu(\boldsymbol{s}) \int \pi(\boldsymbol{a}|\boldsymbol{s}) Q(\boldsymbol{s}, \boldsymbol{a}) \, \mathrm{d}\boldsymbol{s} \, \mathrm{d}\boldsymbol{a}. \tag{7}$$

However, instead of optimizing the policy for each state independently, we resort to maximize over the joint state-action distribution $p(\boldsymbol{s}, \boldsymbol{a}) = p(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s})$. As we don't want to optimize the state distribution, we need to ensure that the optimized marginal state distribution $p(\boldsymbol{s}) = \int p(\boldsymbol{s}, \boldsymbol{a}) \mathrm{d}\boldsymbol{a}$ is the same as the state distribution $\mu(\boldsymbol{s})$ that has generated the data. This constraint is implemented by matching feature averages of the distributions $p(\boldsymbol{s})$ and $\mu(\boldsymbol{s})$ as

$$\int p(\boldsymbol{s}) \boldsymbol{\phi}(\boldsymbol{s}) \, d\boldsymbol{s} = \hat{\boldsymbol{\phi}}, \tag{8}$$

where $\hat{\boldsymbol{\phi}}$ is the average feature vector of all state samples. Similarly, we bound the KL between the new and the old joint state-action distributions instead of the new and old policy, i.e.,

$$\mathrm{KL}\left[p(\boldsymbol{s},\boldsymbol{a})||q(\boldsymbol{s},\boldsymbol{a})\right] \leq \epsilon \tag{9}$$

With a final constraint that ensures that $p(\boldsymbol{s},\boldsymbol{a})$ is a probability distribution, the resulting constraint optimization problem is

$$\begin{aligned} \arg\max_{p} & \iint p(\boldsymbol{s},\boldsymbol{a})Q(\boldsymbol{s},\boldsymbol{a})\,\mathrm{d}\boldsymbol{s}\,\mathrm{d}\boldsymbol{a}, \\ \text{s.t. } & \mathrm{KL}\left[p(\boldsymbol{s},\boldsymbol{a})||q(\boldsymbol{s},\boldsymbol{a})\right] \leq \epsilon, \\ & \int p(\boldsymbol{s})\boldsymbol{\phi}(\boldsymbol{s})\,d\boldsymbol{s} = \hat{\boldsymbol{\phi}}, \\ & \iint p(\boldsymbol{s},\boldsymbol{a})\,d\boldsymbol{s}\,d\boldsymbol{a} = 1. \end{aligned} \tag{10}$$

The upper bound ϵ for the KL divergence is a parameter of REPS that controls the exploration-exploitation trade-off by restricting the greediness of the method. This parameter is usually chosen heuristically. The constraint optimization problem can be solved in closed form with the method of Lagrange multipliers, yielding

$$p(\boldsymbol{s},\boldsymbol{a}) = q(\boldsymbol{s},\boldsymbol{a})\exp\left[\frac{Q(\boldsymbol{s},\boldsymbol{a}) - V(\boldsymbol{s})}{\eta}\right]Z^{-1} \tag{11}$$

where $V(\boldsymbol{s}) = \boldsymbol{\nu}^\intercal\boldsymbol{\phi}(\boldsymbol{s})$ is a state dependent baseline similar to a value function, $\boldsymbol{\nu}$, η, and λ are the Lagrangian multipliers, and $Z = \exp[(\eta + \lambda)/\eta]$ is a normalizing therm. Using $Z = \iint p(s,a)\mathrm{d}s\mathrm{d}a$, we can obtain the dual function by substituting $p(s,a)$ in the Lagrangian with Equation 11. The Lagrangian multipliers η and $\boldsymbol{\nu}$ can then be obtained efficiently by minimizing the dual

$$g(\boldsymbol{\nu},\eta) = \eta\log\left(\frac{1}{N}\sum_{i=1}^{N}\exp\left(\frac{Q(\boldsymbol{s}_i,\boldsymbol{a}_i) - \boldsymbol{\nu}^\intercal\phi(s_i)}{\eta}\right) + \eta\epsilon + \boldsymbol{\nu}^\intercal\hat{\phi}\right) \tag{12}$$

using the Broyden-Fletcher-Goldfarb-Shannon algorithm.

3.1.3. Matching a Continuous Stochastic Policy

Solving the optimization problem obtained from REPS gives the desired probabilities $p(\boldsymbol{s}_i,\boldsymbol{a}_i) = p(\boldsymbol{s}_i)\hat{\pi}(\boldsymbol{a}_i|\boldsymbol{s}_i)$ only for the discrete samples in $\mathcal{D}$. For sample $(\boldsymbol{s}_i,\boldsymbol{a}_i)$ we want to take action $\boldsymbol{a}_i$ in state $\boldsymbol{s}_i$ with probability

$$\hat{\pi}(\boldsymbol{a}_i|\boldsymbol{s}_i) \propto \exp\left(\frac{Q(\boldsymbol{s}_i,\boldsymbol{a}_i) - \boldsymbol{\nu}^\intercal\boldsymbol{\phi}(\boldsymbol{s}_i)}{\eta}\right) \tag{13}$$

We can ignore the old policy $q(\boldsymbol{a}|\boldsymbol{s})$ here because the actions in $\mathcal{D}$ have been samples from $q(\boldsymbol{a}|\boldsymbol{s})$ and the state distributions $p(\boldsymbol{s})$ and $q(\boldsymbol{s})$ are equal due to the constraint in Equation 8.

By fitting a weighted linear model in the feature space of the states, we obtain a continuous policy for the whole state space. With the assumptions of Gaussian exploration noise and of a Gaussian prior over the actions we arrive at a weighted linear Gaussian model which turns

into a weighted Gaussian process by using a kernel function to compute the inner product of the features, i.e.,

$$\begin{aligned} \pi(\boldsymbol{a}|\boldsymbol{s}) &= \mathcal{N}(\boldsymbol{\mu}(\boldsymbol{s}), \boldsymbol{\Sigma}(\boldsymbol{s})), \quad \text{where} \\ \boldsymbol{\mu}(\boldsymbol{s}) &= \boldsymbol{k}(\boldsymbol{s})^\intercal \left(\boldsymbol{K} + l\boldsymbol{W}\right)^{-1} \boldsymbol{A}, \\ \boldsymbol{\Sigma}(\boldsymbol{s}) &= k(\boldsymbol{s}, \boldsymbol{s}) + \boldsymbol{k}(\boldsymbol{s})^\intercal \left(\boldsymbol{K} + l\boldsymbol{W}\right)^{-1} \boldsymbol{k}(\boldsymbol{s}) + l. \end{aligned} \tag{14}$$

Here, $k(\boldsymbol{s}_i, \boldsymbol{s}_j)$ denotes the kernel function of the two states $\boldsymbol{s}_i$ and $\boldsymbol{s}_j$, $\boldsymbol{k}(\boldsymbol{s})$ is the kernel vector of the states $\boldsymbol{s}_i \in \mathcal{D}$ and state $\boldsymbol{s}$ and $\boldsymbol{K}$ is the kernel matrix of the states in $\mathcal{D}$. Note that the prior over the actions is a scalar in the kernel function $k(\boldsymbol{s}_i, \boldsymbol{s}_j) = bK(\boldsymbol{s}_i, \boldsymbol{s}_j)$, we will discuss the kernel function K in Section 4.1.3. Furthermore, l is the variance of the exploration noise, $\boldsymbol{W}$ is a diagonal matrix of the weights $W_{ii} = \hat{\pi}(\boldsymbol{a}_i|\boldsymbol{s}_i)$, and $\boldsymbol{A}$ is a matrix with the actions as row vectors.

A common drawback of kernel functions is that they do not scale well with the number of samples used for training. Several sparsification approaches have been proposed to overcome this issue, e.g., the method of projected latent variables by [23], the sparse Gaussian processes (SPGP) by [24], or the sparse greedy Gaussian process regression by [25]. They use a set of *pseudo inputs* (also called latent variables, inducing inputs or active set) to approximate the full covariance of the Gaussian process. Effectively, given a set of m pseudo inputs, the kernel function in the predictive mean and covariance of a sparse GP is substituted by

$$k(\boldsymbol{s}_i, \boldsymbol{s}_j) = \boldsymbol{k}_m(\boldsymbol{s}_i)^\intercal \boldsymbol{K}_{mm}^{-1} \boldsymbol{k}_m(\boldsymbol{s}_j), \tag{15}$$

where $\boldsymbol{k}_m(\boldsymbol{s}_i)$ is a kernel vector of state $\boldsymbol{s}_i$ with the pseudo inputs and $\boldsymbol{K}_{mm}$ is the kernel matrix of the pseudo inputs. Following the derivations in [21], this allows to define a sparse, weighted Gaussian processes as

$$\boldsymbol{\mu}(\boldsymbol{s}) = \boldsymbol{k}_m(\boldsymbol{s})^\intercal \boldsymbol{Q}^{-1} \boldsymbol{K}_{mn} (l\boldsymbol{W} + \boldsymbol{\Lambda})^{-1} \boldsymbol{A} \tag{16}$$

$$\boldsymbol{\Sigma}(\boldsymbol{s}) = k(\boldsymbol{s}, \boldsymbol{s}) - \boldsymbol{k}_m(\boldsymbol{s})^\intercal \left(\boldsymbol{K}_{mm}^{-1} + \boldsymbol{Q}^{-1}\right) \boldsymbol{k}_m(\boldsymbol{s}) + l \tag{17}$$

$$\boldsymbol{Q} = \left(\boldsymbol{K}_{mm} + \boldsymbol{K}_{mn} (l\boldsymbol{W} + \boldsymbol{\Lambda})^{-1} \boldsymbol{K}_{nm}\right), \tag{18}$$

with diagonal matrix $\Lambda_{ii} = k(\boldsymbol{s}_i, \boldsymbol{s}_i) - \boldsymbol{k}_m(\boldsymbol{s}_i)^\intercal \boldsymbol{K}_{mm}^{-1} \boldsymbol{k}_m(\boldsymbol{s}_i)$.

3.2. Kernel Embeddings of Distributions

We will represent the configuration of the swarm as a distribution where each agent refers to a single sample of that distribution. Using moments of the distribution as representation, e.g., the mean and the variance as in [16], allows for a compact representation of the swarm. However, this representation is also quite limited if the swarm is not distributed similar to a Gaussian distribution. For example, if the swarm is separated into two or more groups, this representation is not able to distinguish between quite different configurations.

The recent technique of kernel embeddings [6] allows a nonparametric representation of distributions with arbitrary shapes. A reproducing kernel Hilbert space (RKHS) $\mathcal{H}$ of functions is uniquely defined by a positive definite kernel function $k(\boldsymbol{x}, \boldsymbol{x}') := \langle \boldsymbol{\psi}(\boldsymbol{x}), \boldsymbol{\psi}(\boldsymbol{x}') \rangle_{\mathcal{H}}$ [26]. Here, the feature mappings $\boldsymbol{\psi}(\boldsymbol{x})$ are often intrinsic to the kernel functions and might map into an infinite dimensional feature space (e.g., the squared exponential kernel.) The embedding of a marginal distribution $p(X)$ is defined as the expected feature mapping of its random variable

$\boldsymbol{\mu}_X := \mathbb{E}_p[\boldsymbol{\psi}(X)] = \int_\Omega \boldsymbol{\psi}(\boldsymbol{x})\, \mathrm{d}p(\boldsymbol{x})$ [6]. In practice we estimate the embedding using samples from $p(X)$ as

$$\hat{\boldsymbol{\mu}}_X = \frac{1}{m}\sum_{i=1}^{m} \boldsymbol{\psi}(\boldsymbol{x}_i) = \frac{1}{m}\sum_{i=1}^{m} k(\boldsymbol{x}_i, \cdot). \tag{19}$$

We will use the mean embedding as representation later to define a kernel function for the state of the swarm. Given infinite dimensional features in the kernel function, such mean embeddings are infinite dimensional as well and cannot be represented explicitly. Still, we can asses the discrepancy of two distributions $p(X)$ and $q(Y)$ using the *maximum mean discrepancy* (MMD) as

$$\begin{aligned}
\mathrm{MMD}(\boldsymbol{\mu}_X, \boldsymbol{\mu}_Y) &= \langle \boldsymbol{\mu}_X - \boldsymbol{\mu}_Y, \boldsymbol{\mu}_X - \boldsymbol{\mu}_Y \rangle && (20)\\
&= \mathbb{E}_{p,p}\left[k(x_i, x_j)\right] - 2\mathbb{E}_{p,q}\left[k(x_i, y_j)\right] + \mathbb{E}_{q,q}\left[k(y_i, y_j)\right] && (21)\\
\widehat{\mathrm{MMD}}(\boldsymbol{\mu}_X, \boldsymbol{\mu}_Y) &= \frac{1}{m^2}\sum_{i=1}^{m}\sum_{j=1}^{m} k(x_i, x_j) - \frac{2}{mn}\sum_{i=1}^{m}\sum_{j=1}^{n} k(x_i, y_j) + \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n} k(y_i, y_j) && (22)
\end{aligned}$$

Note, that the MMD can also be seen as the squared error between two mean embeddings.

3.3. Planning Strategies

A* is a heuristic search algorithm commonly applied for graph search problems [27]. The algorithm selects which node n_s to expand by minimizing the cost $f(n_s) = g(n_s) + h(n_s)$, where $g(n_s)$ is the cost for reaching node n_s from the start and $h(n_s)$ is a heuristic that provides a lower bound to the costs from $\boldsymbol{s}$ to the goal state $\boldsymbol{s}_G$. The cost $g(n_s)$ can be computed by

$$g(n_s) = g(\mathrm{pred}(n_s)) + c(\mathrm{pred}(n_s), n_s), \tag{23}$$

where $\mathrm{pred}(n_s)$ is the parent node of n_s and $c(n_{s_1}, n_{s_2})$ is the cost to get from n_{s_1} to n_{s_2}.

Potential fields [28] are a fast planning method for mobile robots. The robots move along a hypothetical force field, being attracted to the goal position and repulsed from the obstacles. The repulsive potential for an object $\boldsymbol{o}$ is defined as

$$U_{\mathrm{rep}}(\boldsymbol{s}, \boldsymbol{o}) = \begin{cases} \frac{1}{2}\chi\left(\frac{1}{d(\boldsymbol{s},\boldsymbol{o})} - \frac{1}{d_o}\right)^2 & \text{if } d(\boldsymbol{s}, \boldsymbol{s}_G) < d_o \\ 0 & \text{else} \end{cases}, \tag{24}$$

respectively. Here, $d(\boldsymbol{a}, \boldsymbol{b})$ is a measure for the distance between $\boldsymbol{a}$ and $\boldsymbol{b}$, d_o is the maximum distance to the obstacle, and χ is a scaling factor. In our approach, we use the repulsive potential in the cost term for the path segments $c(n_{s_1}, n_{s_2})$ of A* (see Section 4.2 for more details).

4. Learning Control Policies for Object Assembly

We split the task of object assembly into three components (an overview is given in Figure 2): (1) an *assembly policy* that describes how the individual objects should move, (2) a *path planning strategy* to guide the swarm around the objects and to arrange them for the next

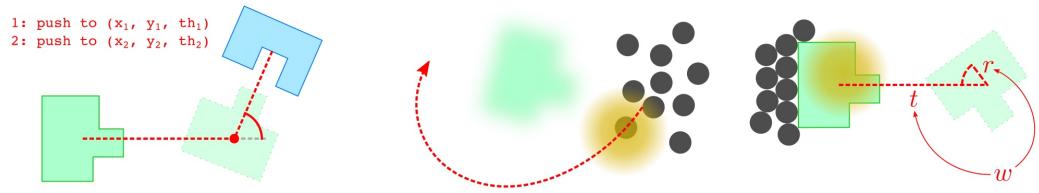

Figure 2. The three components of our approach. Left: the assembly policy defines way points for the objects; middle: a path planning strategy computes collision free paths for the objects but is also used to position the Kilobots for the next push; right: the object movement policy controls the light source when the swarm is pushing the objects.

pushing task, and (3) an *object movement policy* that realizes basic movements of an object by controlling a light source that guides the robot swarm.

4.1. The Object Movement Policy

The object movement policy controls the global input signal, e.g., the position of the light source such that the swarm, which follows this signal, pushes the object along a given trajectory. We reduce the search space for the object movement policies by considering only pushes in positive x-direction or counterclockwise rotations. Later, we apply the learned policies to arbitrary movements by rotating and flipping the state representation accordingly. We further introduce a trade-off parameter $\rho \in [0, 1]$ that weighs between translational and rotational movements. This trade-off is achieved by the design of the reward function which we introduce in Section 4.1.1. In our experiments we usually learned object movement policies for three settings of ρ, i.e., $\rho = 0.0$, $\rho = 0.5$, $\rho = 1.0$.

4.1.1. The Reward Function

The reward function reflects the setting of the trade-off parameter $\rho \in [0, 1]$. The function rewards only rotational movements for $\rho = 1$ and only translational movement for $\rho = 0$. For values in between, ρ trades off the rotational and the translational term. Given the translational movements d_x in x-direction, d_y in y-direction and the rotational movement d_θ, we define the reward as

$$r(\rho) = \rho\, r_{\text{rot}} + (1 - \rho) r_{\text{trans}} - c_y d_y, \tag{25}$$

with the translational and rotational reward terms

$$r_{\text{trans}} = d_x - c_\theta\, d_\theta, \quad \text{and} \tag{26}$$

$$r_{\text{rot}} = d_\theta - c_x\, d_x, \tag{27}$$

respectively. The weights c_x, c_y, and c_θ scale the costs for undesired translational or rotational movements.

4.1.2. States and Actions

We define our state relative to the center of the object part that we want to push. Given the relative light position $\boldsymbol{l} = (x_l, y_l)$ and a swarm configuration with n agents, where agent i has the relative position $\boldsymbol{b}_i = (x_i, y_i)$, the state vector is defined as $\boldsymbol{s} := [\boldsymbol{l}, \boldsymbol{b}_1, \ldots, \boldsymbol{b}_n]$. The action vector $\boldsymbol{a} = (a_x, a_y)$ is the desired displacement of the light source in x- and y-direction.

4.1.3. Features and Kernels

To learn an object movement policy that generalizes to different swarm sizes, we need to employ a feature mapping that abstracts from the number of individuals in the swarm and also from the allocation of the single robots to their positions (i.e., which agent is at which position). Therefore, we represent the state of the swarm as a distribution embedded into a RKHS [6] where we treat each agent as a sample of that distribution, i.e.,

$$\mu_{\boldsymbol{b}}(\cdot) = \frac{1}{n}\sum_{i=1}^{n} k(\boldsymbol{b}_i, \cdot) = \frac{1}{n}\sum_{i=1}^{n} \psi(\boldsymbol{b}_i), \tag{28}$$

where k is a kernel function (e.g., the Gaussian kernel) and ψ is the intrinsic feature mapping of k. This representation is invariant to both, the allocation of the individual agent to the position as well as to the number of agents in the swarm. We can compute the difference between two swarm distributions independently from the number of agents by computing the squared difference of their embeddings

$$d_{\mathrm{b}}(\boldsymbol{b}, \boldsymbol{b}') = \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n} k(\boldsymbol{b}_i, \boldsymbol{b}_j) - \frac{2}{nm}\sum_{i=1}^{n}\sum_{j=1}^{m} k(\boldsymbol{b}_i, \boldsymbol{b}'_j) + \frac{1}{m^2}\sum_{i=1}^{m}\sum_{j=1}^{m} k(\boldsymbol{b}'_i, \boldsymbol{b}'_j). \tag{29}$$

Here, $\boldsymbol{b}$ and $\boldsymbol{b}'$ are two swarm configurations with n and m individuals, respectively. In addition to the state of the swarm, we also need to represent the current relative position of the light $\boldsymbol{l}$ and the desired displacement of the light $\boldsymbol{a}$ (i.e., the action) in the feature vector. For both, we can obtain the squared distance simply by

$$d_{\mathrm{v}}(\boldsymbol{v}, \boldsymbol{v}') = -0.5(\boldsymbol{v} - \boldsymbol{v}')^{\intercal} \operatorname{diag}\left(\sigma_{\mathrm{v}}^{-2}\right)(\boldsymbol{v} - \boldsymbol{v}'), \tag{30}$$

where $\boldsymbol{v}$ can be either the composition of $\boldsymbol{l}$ and $\boldsymbol{a}$ or only the light position $\boldsymbol{l}$, depending if we need a feature function for state-action pairs or only states. We can now combine these two distance measures into a kernel function

$$K(\boldsymbol{s}, \boldsymbol{s}') = \exp\left(-\frac{\alpha}{2} d_{\mathrm{v}}(\boldsymbol{v}, \boldsymbol{v}') - \frac{1-\alpha}{2} d_{\mathrm{b}}(\boldsymbol{b}, \boldsymbol{b}')\right), \tag{31}$$

where $\alpha \in [0, 1]$ weighs the importance of the non-agent dimensions $\boldsymbol{v}$ and the agent dimensions $\boldsymbol{b}$ of the state $\boldsymbol{s}$.

At each learning iteration of the AC-REPS algorithm, we select a kernel reference set $\mathcal{D}_{\mathrm{ref}} = (\boldsymbol{s}_i, \boldsymbol{a}_i)_{i=1}^{N}$ randomly from the SARS samples. With this, we can define the feature vector $\boldsymbol{\phi}(\boldsymbol{s}, \boldsymbol{a})$ for approximating the Q-function, where the i-th entry of the feature vector

$$\phi(\boldsymbol{s}, \boldsymbol{a})_i = K((\boldsymbol{s}_i, \boldsymbol{a}_i), (\boldsymbol{s}, \boldsymbol{a})), \quad i = 1, \ldots, N \tag{32}$$

is the kernel function evaluated at the reference sample $(\boldsymbol{s}_i, \boldsymbol{a}_i)$. For the policy improvement step, we need a state-dependent feature function which we define as

$$\varphi(\boldsymbol{s})_i = K(\boldsymbol{s}_i, \boldsymbol{s}), \quad i = 1, \ldots, N. \tag{33}$$

4.2. Assembly Policy and Path Planning Strategy

The assembly policy contains the construction information stored as a list of oriented way points with required accuracies for each object. These way points are processed consecutively by applying either the object movement policy or the path planning strategy. When the object movement strategy is applied, we have to minimize the translational error e_{trans} and the rotational error e_{rot} until the next way point is reached. We compute the desired translation-rotation ratio as $\rho_{\text{des}} = \eta e_{\text{rot}}/(e_{\text{trans}} + \eta e_{\text{rot}})$ and choose the object movement policy with closest ration ρ. The parameter η scales the rotational error with the translational error, usually a value of 0.1 leads to good results.

For guiding the swarm from one object to the next, we use a path planning strategy to obtain a collision free path to the target. We use A* in combination with the repulsive potential of the potential fields in the cost term $c(\boldsymbol{s}_1, \boldsymbol{s}_2)$. Naively, we could also simply follow the gradient of the potential field. However, this approach is prone to issues such as local minima, narrow passages, or oscillations around obstacles [29]. Instead, we define the cost function $c(\boldsymbol{s}_1, \boldsymbol{s}_2) = d(\boldsymbol{s}_1, \boldsymbol{s}_2) + U_{\text{rep}}(\boldsymbol{s}_2)$, where $d(\boldsymbol{s}_1, \boldsymbol{s}_2)$ is the distance between $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$, and $U_{\text{rep}}(\boldsymbol{s}_2)$ is the repulsive potential. As heuristic $h(\boldsymbol{s})$ we use the Euclidean distance to the target state.

5. Experimental Setup & Results

We evaluate the proposed learning method in simulation as well as on a robotic platform. As robotic platform, we chose the Kilobot platform [4]. The Kilobots are an affordable and open source platform developed specifically for the evaluation of algorithms on large swarms of robots. Each robot is approximately 3 cm in diameter, 3 cm tall and moves up to 1 cm/s by using two vibration motors.

We have implemented a 2D simulator of the Kilobot platform in Python[1] compatible with the OpenAI gym[30]. To simulate the interactions of the agents and the objects in the world, we use the physics engine Box2D[2]. We use this simulation to evaluate the learning algorithm and to learn the policies that we will later apply directly to the real Kilobots. The simulator internally runs at 10Hz, but only takes action and returns state and rewards at 1Hz.

5.1. Evaluation of the Learning Algorithm

We learn the object movement policy for six object types, i.e., square ($w = 0.15$), rectangle ($w = 0.05, h = 0.3$), triangle ($w = 0.14, h = 0.21$), L-shape, T-shape, and C-shape (each with overall $w = 0.14, h = 0.21$) (c.f. Table 1) and for three ratios $\rho \in [0.0, 0.5, 1.0]$. The object is initialized at $(0, 0)$ with a random orientation uniformly sampled from $[\pi, -\pi]$. To simulate the light source, we use a circular gradient with radius $r = 0.2$. If an agent is within this radius, it senses the gradient towards the center. The initial position of the light and the swarm is sampled normally around the worlds center with standard deviation $\max(w_{world}, h_{world})/3$. The agents are sampled normally around the light position with standard deviation $r/3$.

We learn the object movement policies with 10 agents over 60 iterations. In each iteration we sample 200 episodes with 60 steps/episode. Afterwards, we keep a set of 10000 SARS tuples which we choose randomly from the new samples and the old SARS tuples. To define the feature function for LSTD, we select 1000 samples from the SARS data randomly. We choose 1000 inducing inputs for the sparse GP later by importance sampling using the weights

[1] The Kilobot Gym, `https://github.com/gregorgebhardt/gym-kilobots`

[2] Box2D – A 2D Physics Engine for Games, `http://box2d.org/`

obtained from REPS. After each learning iteration, we evaluate the learned policy on 50 episodes of length 125.

Tables 1–3 show the learned policies, the learned value function, as well as the light and object trajectories of the learning episodes and of the evaluation episodes for the ratios $\rho = 0.0, \rho = 0.5, \rho = 1.0$. The depicted results are taken from the iteration with the highest mean reward of the evaluation episodes. Note that we use artificial configurations in which all agents and the light are at the same position (x, y) to visualize the policies and value functions.

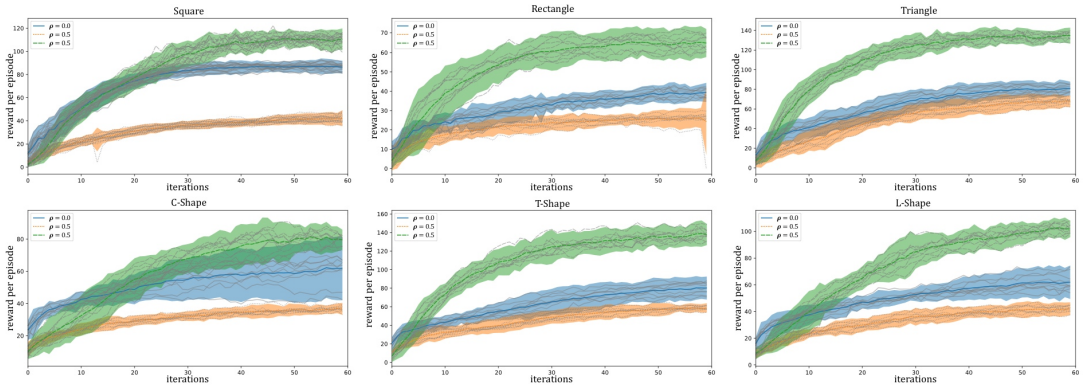

Figure 3. Learning curves for square, rectangle, triangle, C-shape, T-shape, and L-shape for $\rho \in [0.0, 0.5, 1.0]$. Note the different scalings of the y-axis.

Figure 3 shows the learning curves for each object shape and ratios $\rho \in [0.0, 0.5, 1.0]$. Note the differently scaled y-axes which depicts the different difficulties in learning the policies for different object types. It can be seen that the relation of the different policy types varies strongly between the object shapes. This relates to the object geometries which make it harder for some objects to be pushed without rotational movement (e.g. triangle), or which make it easier to be rotated with only little translational movement (e.g. rectangle).

We have evaluated how well each learned set of policies can pushing the objects along a straight line and along a circular path. In addition, we have compared how well a policy learned on the square would generalize to the other shapes. Figure 4 shows the trajectories obtained from the straight-line-task. While we get good results for the square, the rectangle and the T-shape using policies learned for the respective shapes, the task is harder to solve with the triangle, the L-shape and the C-shape. The policies learned for the square perform worse for the rectangle, the L-shape, and the C-shape. For the triangle, the square policies yield similar results as the triangle policies. Except for two outliers, the square policies also yield similar results as the learned policies for the T-shape.

Figure 5 shows the trajectories for the circular-path-task with radii 0.2, 0.4, 0.6, and 0.8. The circular paths are defined by 10 way points in equiangular distance with an orientation accuracy of 1.5 (roughly $\pm\frac{\pi}{2}$) and a position accuracy of 0.1. Except for the last way point which has a position accuracy of 0.05 only in the y coordinate. While the square tracks the circular path nicely, the rectangle only succeeds in tracking the circular paths with radii 0.4, 0.6, and 0.8 adequately. The triangle manages to follow the paths although with a much larger variance than square or rectangle. The L-shape policies seem to be able to track the paths somehow, however, they often fail to meet the quite broad orientation accuracy of the way points. This results in maneuvering around the way points and thus in the loopy trajectories. The same can be observed for the T-shape. In contrast, the C-shape policies manage to follow the circular paths with errors but stay inside the given accuracy windows.

A *pose controller* (PC) for a swarm of simple agents has been proposed in [16]. We compare

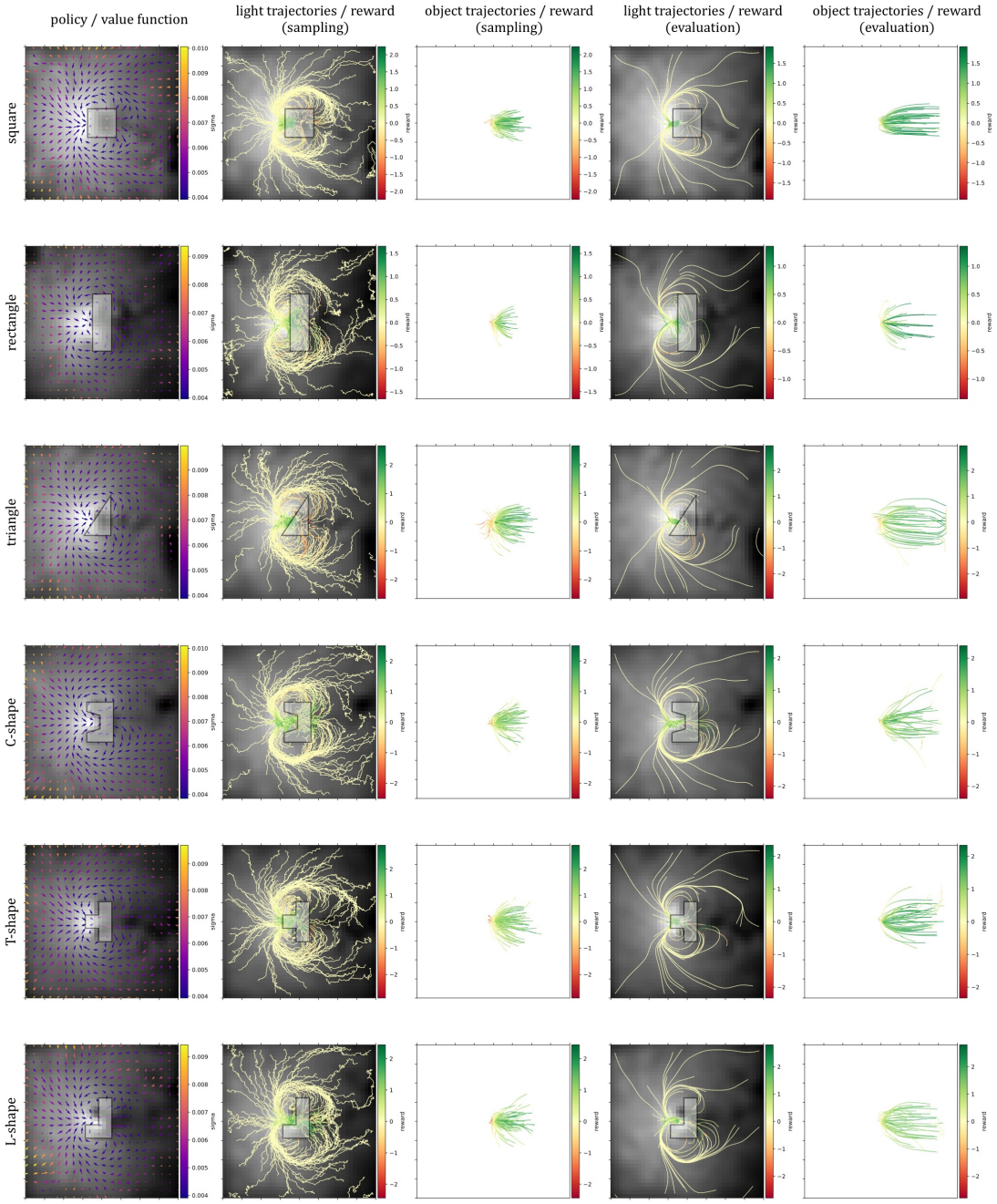

Table 1. Results from learning the object movement policy for $\rho = 0.0$. The first column depicts the learned policy and the value function. The policy is shown as quiver plot where the arrows denote the mean action and the color denotes the variance of the GP. The second column shows the trajectories of the light center during the evaluation episodes (i.e., taking the mean of the GP as action) relative to the object. The third column shows the object trajectories during the evaluation episodes. The color of the trajectories denotes the reward obtained at that time step. Likewise, the fourth and fifth columns depict the trajectories of the light source and the object, respectively, during the sampling episodes (i.e. with actions samples from the GP.)

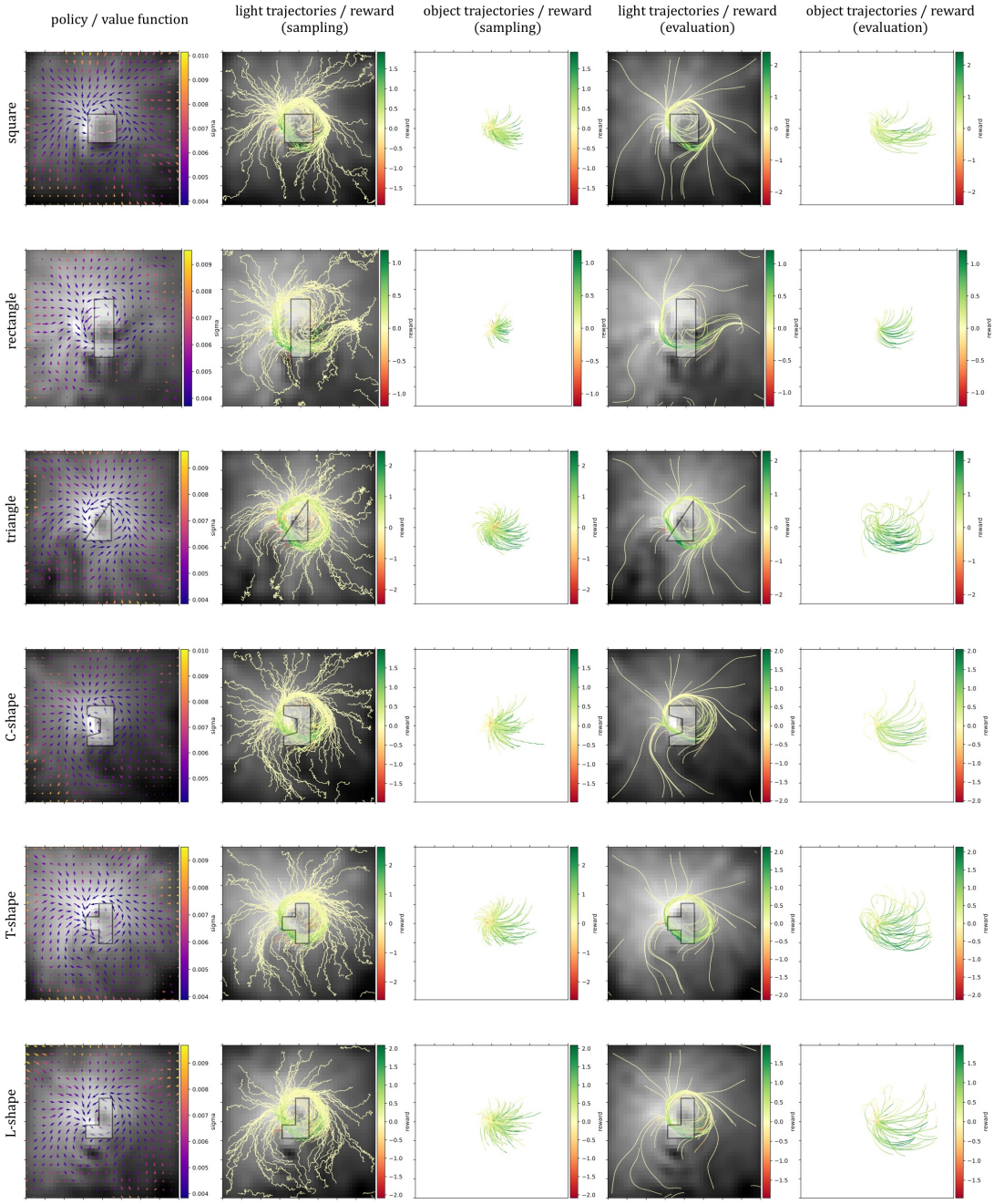

Table 2. Results from learning the object movement policy for $\rho = 0.5$. The first column depicts the learned policy and the value function. The policy is shown as quiver plot where the arrows denote the mean action and the color denotes the variance of the GP. The second column shows the trajectories of the light center during the evaluation episodes (i.e., taking the mean of the GP as action) relative to the object. The third column shows the object trajectories during the evaluation episodes. The color of the trajectories denotes the reward obtained at that time step. Likewise, the fourth and fifth columns depict the trajectories of the light source and the object, respectively, during the sampling episodes (i.e. with actions samples from the GP.)

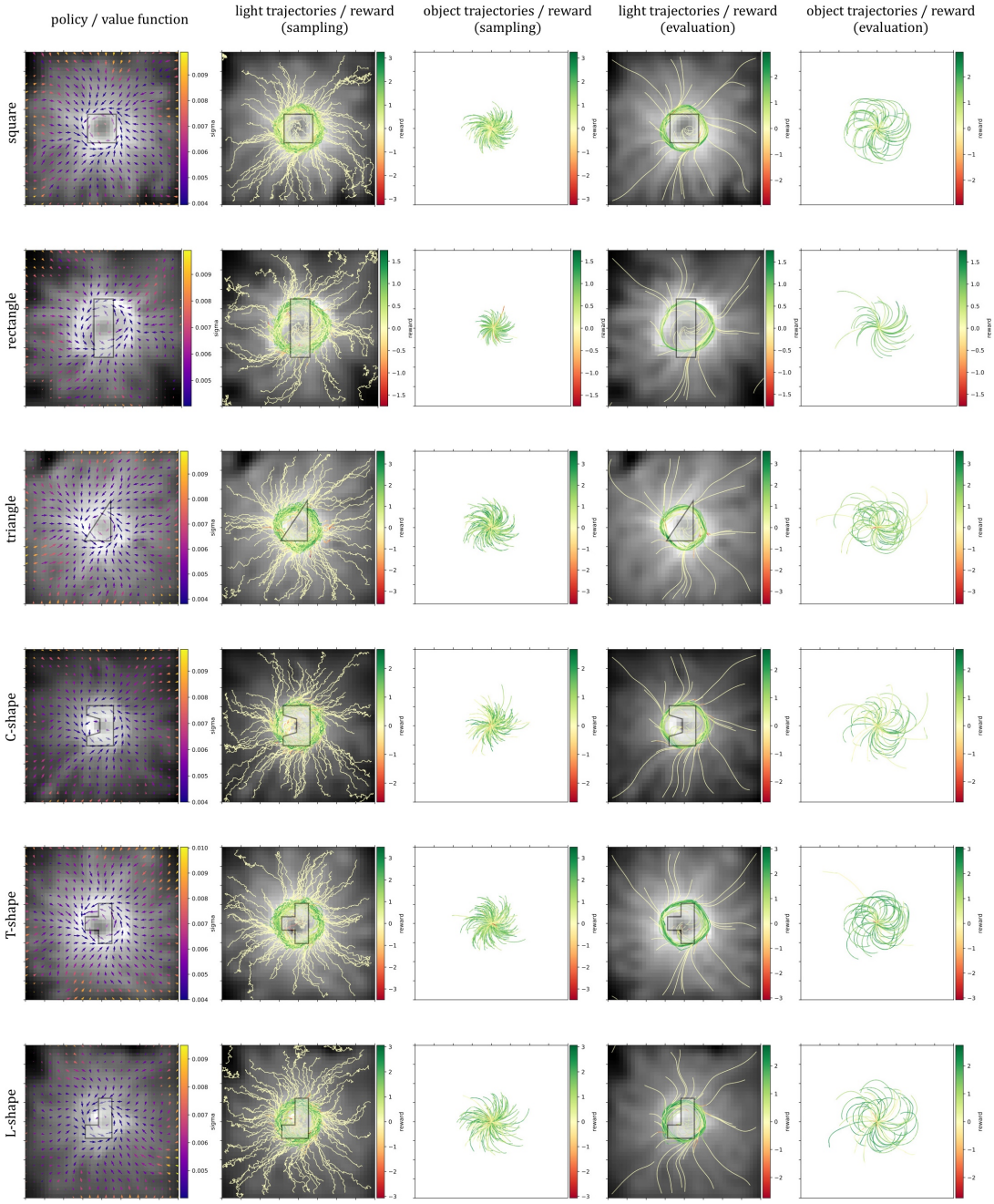

Table 3. Results from learning the object movement policy for $\rho = 1.0$. The first column depicts the learned policy and the value function. The policy is shown as quiver plot where the arrows denote the mean action and the color denotes the variance of the GP. The second column shows the trajectories of the light center during the evaluation episodes (i.e., taking the mean of the GP as action) relative to the object. The third column shows the object trajectories during the evaluation episodes. The color of the trajectories denotes the reward obtained at that time step. Likewise, the fourth and fifth columns depict the trajectories of the light source and the object, respectively, during the sampling episodes (i.e. with actions samples from the GP.)

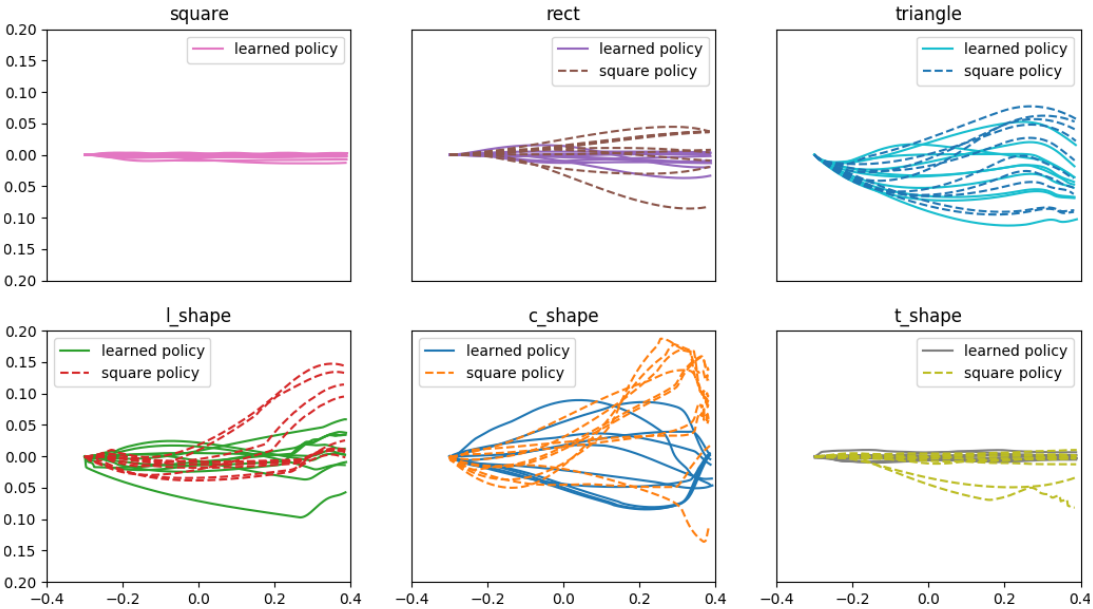

Figure 4. Evaluation of pushing the objects along a straight line with a policy learned for the specific object shape and with a policy learned for the square object.

against this PC by pushing the rectangle object to three target poses. Figure 6, shows the results of this comparison. While both approaches successfully push the object to the first target pose in each of the 10 trials, the PC produces a much longer trajectory as it first pushes the object towards a line through the target pose, before it starts pushing the object towards its final pose. In the second experiment, the PC only succeeds in 4 out of 10 trials to position the object in the target pose. In the third experiment, the PC fails all 10 trials since the target point on the auxiliary line is located outside of the environment. In general, the PC is not able to recover from situations in which the object has been pushed into an undesired position as maneuvering of the object has not been considered in the algorithm.

To evaluate how well our approach generalizes to different swarm sizes, we have applied policies learned with 15 agents on the square object to swarms with 5 to 80 agents. Figure 7 shows the average reward per step for $\rho = 0$ and for $\rho = 1$. Until a swarm size of about 40 agents the reward increases. The more agents are able to push the object the higher is the combined force and, hence, the object moves faster. However, from a swarm size of roughly 50 agents on, the average reward starts to decline. With too many agents in the swarm, the swarm distributes around the object so that the agents push from opposing directions and obstruct the desired motion. Figure 8 depicts this evaluation.

5.2. The Assembly Task in Simulation

We have evaluated the learned policies for triangle, L-shape, C-shape and T-shape on three object assembly tasks in simulation. Furthermore, we have also executed these tasks with policies learned on the square object to asses how good policies learned on a simple shape generalize to more complex shapes.

The first task is to assemble two triangular objects. The assembly policy contains two way points for each of the triangular objects, where the first ensures that the objects are positioned well before they are pushed into the target position. An exemplary execution of the assembly is shown in Figure 9. The swarm pushes the first object to its target position passing through the

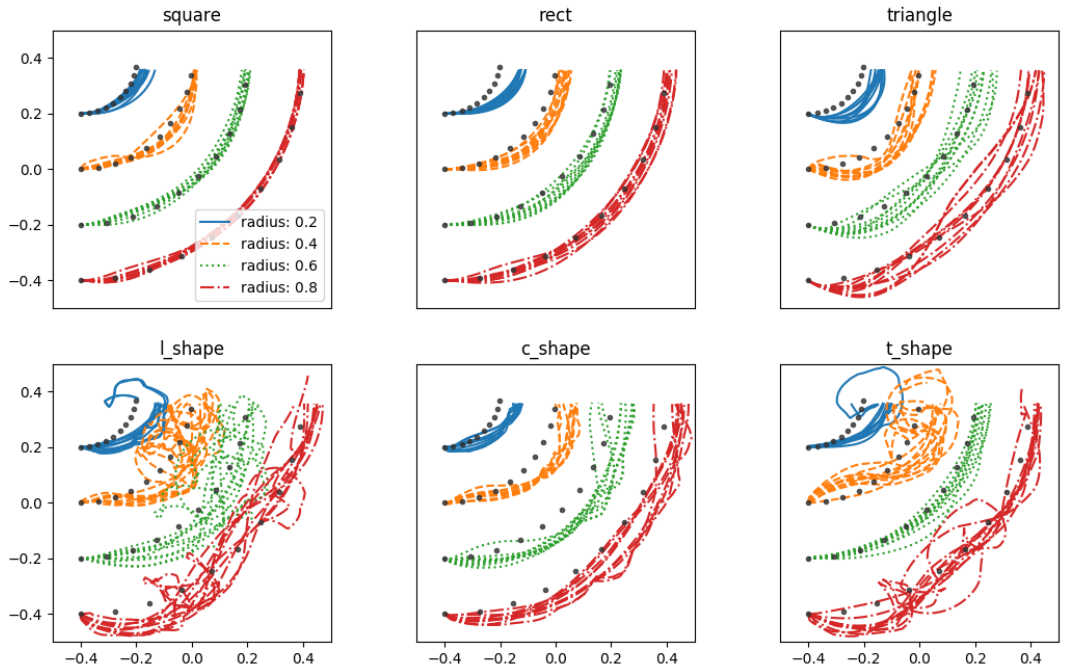

Figure 5. Evaluation of pushing the objects along circular paths with radii $r \in [0.2m, 0.4m, 0.6m, 0.8m]$ with a policy learned for the specific object shape and with a policy learned for the square object. The black dots denote the way points that define the circular path.s Note that some of the trajectories overshoot the target of the circular path since we set the orientation accuracy at the target to ± 1.5 (roughly $\pm\pi/2$) which was not met in these trajectories.

intermediate way point. The swarm is then guided to the second object along a path obtained from the path planning strategy. The positioning of the second object at the intermediate way point requires maneuvering which is done by the learned object movement policy. Finally, the second object is pushed to the target position to assemble it with the first object. In our experiments, the assembly process succeeded in 4 of 5 trials when using the policy learned for the triangle shape and in 0 of 5 trials when using the policies learned with the square shape.

In the second task, the goal is to assemble two L-shapes, exemplary executions of the task are depicted in Figure 10 The first L-shape is pushed directly to the target position. The second L-shape is first rotated to an intermediate way point before pushed against the first L-shape at the target position. The depicted assembly process with square policies nearly succeeds but eventually fails at positioning the second L-shape at the target position. A frequent issue during the execution of this task for both, L-shape policies and square policies, was that the second L-shape could not be positioned adequately at the target way point. During the maneuvering of the object the swarm then pushed the first L-shape away from its target pose and thus breaks the assembly process. In our experiments the assembly process succeeded in 5 of 5 trials for the L-shape policy, however in 2 trials the first L-shape was pushed away during maneuvering the second L-shape. With the square policy, the assembly task succeeded in 2 of 5 trials, although in both successful trials the first L-shape was pushed away from its target position.

The third task is the assembly of a C-shape with a T-shape as depicted in Figure 11. First, the C-shape has to be pushed through a way point to guide the rotation into the final pose. Then the swarm is repositioned at the T-shape which is pushed to an intermediate way point with the right orientation for pushing it into the final position. In our experiments, the CT-shape-assembly task succeeded in 3 of 5 trials with policies learned on C- and T-shapes, respectively. With the square policy, the CT-shape-assembly task succeeded in 0 of 5 trials.

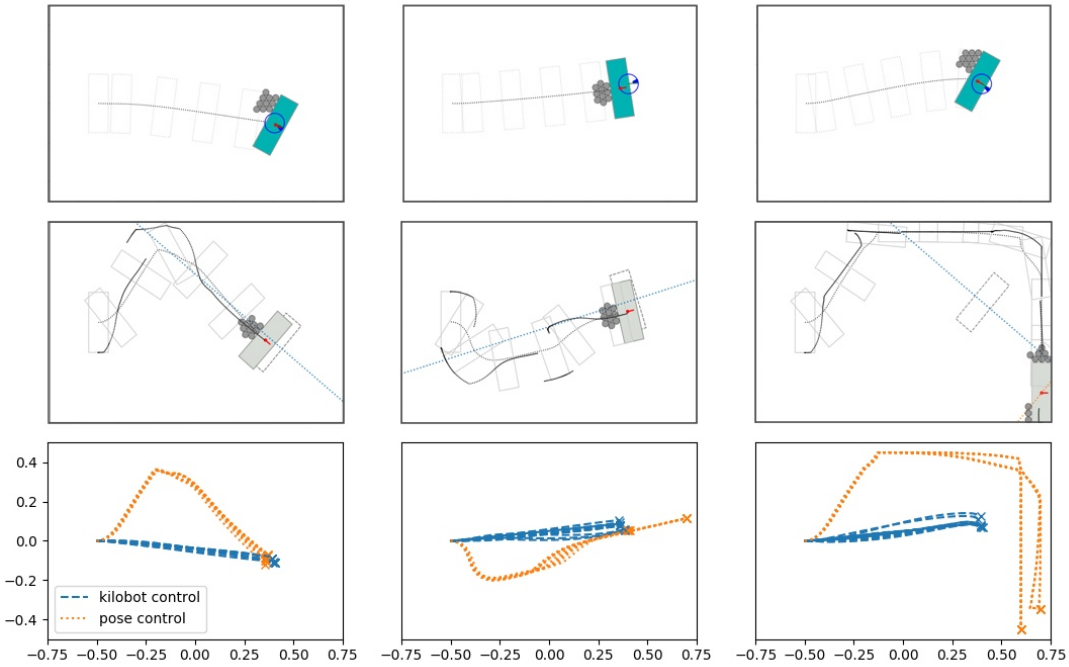

Figure 6. Evaluation of the object movement policy learned on the rectangular object against the pose controller proposed in [16]. The top row shows exemplary runs with the object movement policy for the three target locations. The second row shows exemplary runs with the pose controller [16]. The bottom row shows a comparison of the object trajectories with 10 runs for each controller and each target object position.

5.3. The Kilobot Setup

We use a horizontally mounted 2m × 1.5m whiteboard as environment for the Kilobots. The whiteboard provides a reflective surface with low friction which is beneficial for the slip-stick motion of the robots. We further emulate a light source using a projector mounted vertically to the ceiling. To control the swarm, we project a circular gradient with radius 0.2m and use the phototaxis algorithm on the Kilobots [31]. Figure 12 depicts the setup.

In contrast to the original design developed at Harvard [32], the commercially manufactured Kilobots[3] have a surface-mounted device (SMD) light sensor at the side of the battery instead of the through hole (TH) diode at the back. However, this change in the design significantly

[3]Distributed by K-Team, `http://www.k-team.com/`

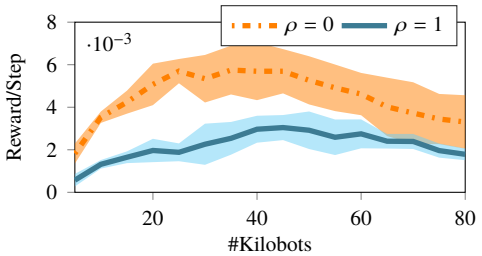

Figure 7. Average reward per time-step of a pure translation ($\rho = 0$) and a pure rotation ($\rho = 1$). All policies are learned with 15 agents and evaluated with 5 to 80 agents.

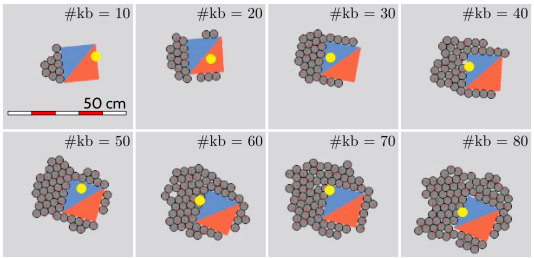

Figure 8. With a size of 50 agents and more, the swarm distributes around the object and obstructs the intended push. A video including these evaluations is available at `https://youtu.be/kuU8wsR9dD4`.

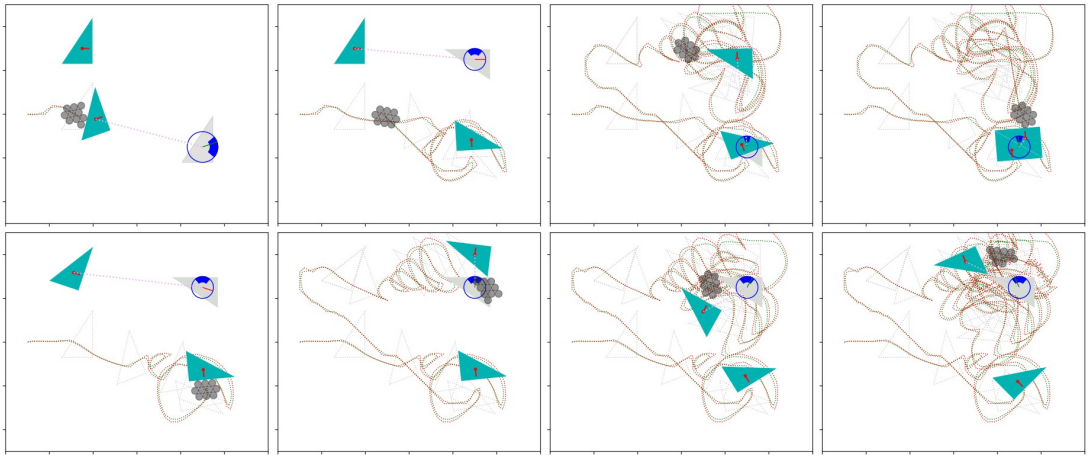

Figure 9. Assembly task with two triangle objects. In the top row, the task is executed successfully with a policy learned on the triangle shape, in the bottom row the task fails when executed with a policy learned with a square object. The red line is the trace of the light source, the green line is the trace of the swarm. The blue circles depict the way points with the required accuracy for position and orientation, the green/red line inside of the way points depicts the current orientation of the object.

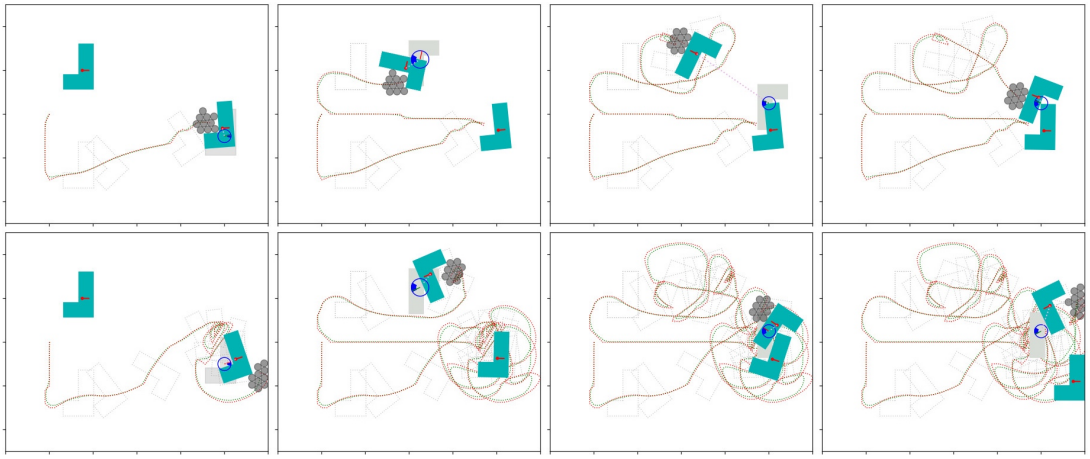

Figure 10. Assembly task with two L-shape objects objects. In the top row, the task is executed successfully with a policy learned on the L-shape, in the bottom row the task fails when executed with a policy learned with a square object.

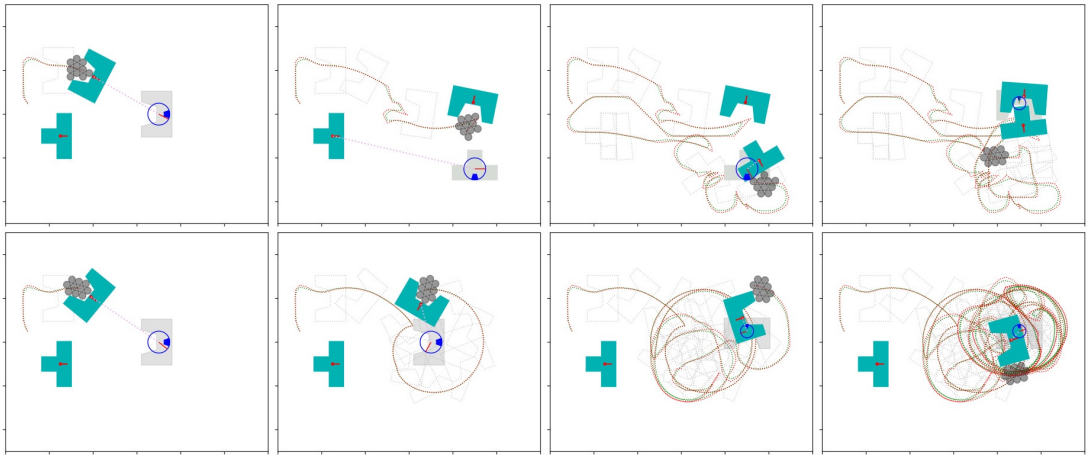

Figure 11. Assembly task of a C-shape with a T-shape. In the top row, the task is executed successfully with polices learned on C- and T-shape. In the bottom row the task fails when executed with a policy learned with a square object.

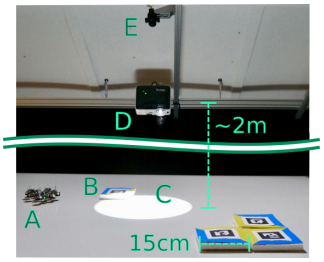

Figure 12. The Kilobot swarm (A) pushes the assembly objects (B) on a 2m × 1.5m whiteboard. The circular light gradient (C) is projected onto the table by a video projector (D). The scene is observed with an RGB camera (E).

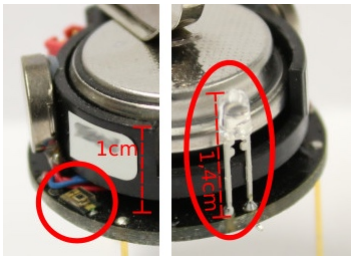

Figure 13. Modification of the Kilobot hardware, to achieve a good phototaxis behavior. Left: commercially available Kilobot with an SMD light sensor. Right: modified Kilobot with a through-hole diode as in the original design.

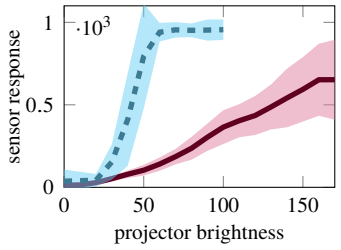

Figure 14. Sensor response curves of SMD sensor and TH diode. The TH diode has a much greater dynamic range. The plots show mean and average over 5 and runs for SMD sensor and TH diode, respectively.

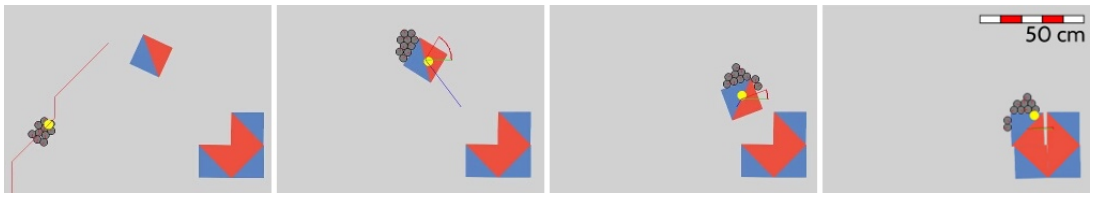

Figure 15. The assembly task of four squares into a big square in simulation. The Kilobots are depicted by gray circles and the light position by a yellow circle. A video of both experiments is available at `https://youtu.be/kuU8wsR9dD4`.

decreases the performance of the phototaxis algorithm. Additionally, the chosen SMD sensor has a roughly three-times-reduced dynamic range in comparison to the TH sensor which we chose as replacement (see Figure 14).

To obtain the positions of the Kilobots and the objects in the scene, we apply simple detection and tracking algorithms. However, the low illumination of the scene (which is required for the phototaxis behavior of the Kilobots) and the bright circular gradient projected onto the table exceeds the dynamic range of the RGB camera. To overcome this problem, we generate HDR images from images with different exposure times.

To achieve a stable and robust tracking of the pose of arbitrary objects, we mark the objects with Chilitags [33]. Chilitags are precise, reliable and illumination tolerant 2D fiducial markers and thus are well suited for the experimental setup. We track the Kilobots using a Hough circle transform (HCT) which is well suited for the round geometry of the robots. HCT is not as precise and robust as the Chilitag tracking, but since the policy uses a distribution-based state representation, it is less sensitive to noise in the Kilobot state.

5.4. The Assembly Task on the Kilobots

We have evaluated the assembly task on the real Kilobot platform with the modifications described in the previous section. We have learned the object movement policies with ratios $\rho = 0$, $\rho = 0.25$, $\rho = 0.5$, $\rho = 0.75$, and $\rho = 1$ in simulation with a swarm size of 15 Kilobots and evaluated on the real Kilobots using swarms of 12, 15, and 24 agents. For the experiment with the real Kilobots, the swarm size is limited as the area of the circular gradient is limited and the robots outside of the gradient are not controllable anymore. Still, the phototaxis performance is not sufficient to keep all robots reliably in the area of the gradient. We apply policies learned in simulation directly to the Kilobot platform. No further optimization on the real robots is required. Figure 15 depicts a simulation of the assembly task that we later applied

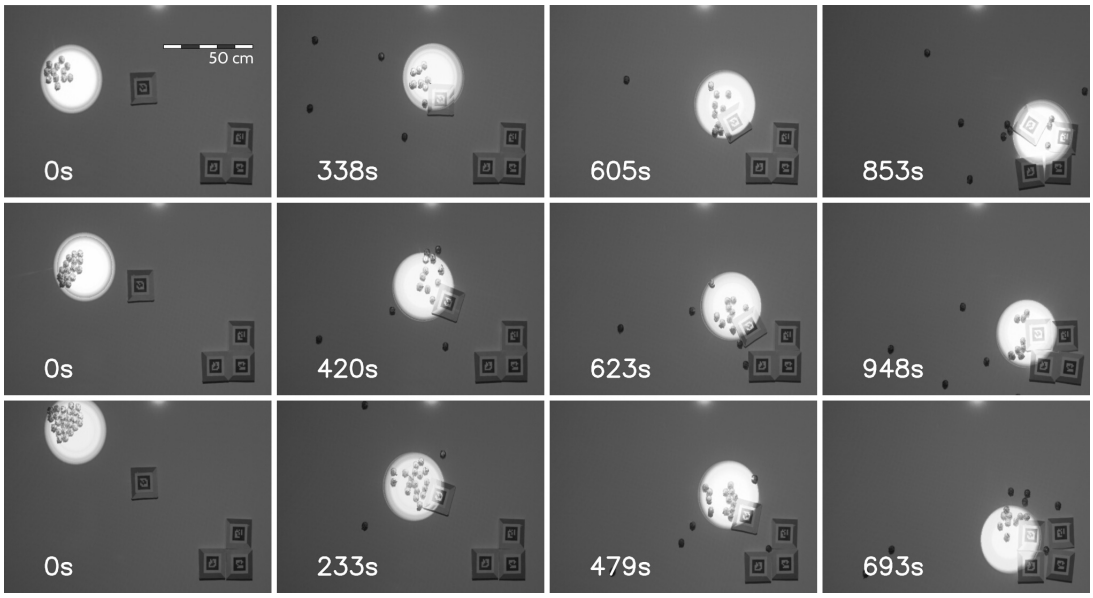

Figure 16. Assembly of a square part with three similar parts into a big square with different swarm sizes. In the first row: 12 Kilobots, in the second row: 15 Kilobots, in the third row: 24 Kilobots Multiple robots are lost during the run, larger swarm sizes lead to better performances and faster execution. A video is available at `https://youtu.be/kuU8wsR9dD4`.

to the real Kilobot platform. The experiment on the real Kilobots is depicted in Figure 16.

With 12 Kilobots, our approach was able to push the fourth square close to the remaining three squares. Yet, only around half of the swarm remains in an area of the gradient when approaching the final position. Consequently, the swarm is not able to finish the assembly by correcting the orientation of the square. With a swarm size of 15 agents the assembly task has succeeded. Although again many robots fail to follow the light source, the number of Kilobots that remain in the area of the gradient is sufficiently large to finish the assembly task. With 24 Kilobots in the swarm, the assembly task has been completed successfully as well and also the time consumption of the task could be reduced to ca. 700s in contras to ca. 950s that were required by the assembly process with 15 Kilobots.

6. CONCLUSION

In this paper, we have presented a novel method for solving the assembly task using a common input signal to a swarm of simple agents. Our method learns policies for the input signal such that the swarm, by following this input signal, pushes an object into a given direction or rotational movement. For this learning method, we have introduced a swarm representation that is invariant to the number of agents in the swarm and their specific locations. This representation simplifies not only the search space for the learning method, it also allows to transfer the learned policy to different swarm sizes. We could show that a policy learned on a simple shape generalizes to a certain extend, still, policies specifically learned for a certain object shape outperforms a general object movement policy. We applied the learned object movement policies in a hierarchical Kilobot controller. We could show in simulation and on the real Kilobot platform, that the Kilobot controller is able to solve the object assembly task. The learned policies could be transfered directly to the real robots without any additional learning.

ACKNOWLEDGMENT

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 645582. Calculations for this research were conducted on the Lichtenberg high performance computer of the TU Darmstadt.

References

[1] Parker LE. Multiple mobile robot systems. In: Siciliano B, Khatib O, editors. Springer handbook of robotics. Springer Berlin Heidelberg; 2008. p. 921–941.

[2] Mavroidis C, Dubey A, Yarmush M. Molecular machines. Annual Review of Biomedical Engineering. 2004;6(1):363–395.

[3] Martel S, Mohammadi M, Felfoul O, et al. Flagellated magnetotactic bacteria as controlled mri-trackable propulsion and steering systems for medical nanorobots operating in the human microvasculature. The International Journal of Robotics Research. 2009;28(4):571–582.

[4] Rubenstein M, Ahler C, Hoff N, et al. Kilobot: A low cost robot with scalable operations designed for collective behaviors. Robotics and Autonomous Systems. 2014;.

[5] Rubenstein M, Cabrera A, Werfel J, et al. Collective transport of complex objects by simple robots: Theory and experiments. In: Proceedings of the International Conferenece on Autonomous Agents and Multi-Agent Systems; 2013.

[6] Smola A, Gretton A, Song L, et al. A hilbert space embedding for distributions. In: International Conference on Algorithmic Learning Theory; 2007.

[7] Wirth C, Fürnkranz J, Neumann G. Model-free preference-based reinforcement learning. In: Proceedings of the 30th AAAI Conference on Artificial Intelligence; 2016.

[8] Gebhardt GH, Daun K, Schnaubelt M, et al. Learning robust policies for object manipulation with robot swarms. In: Proceedings of the 2018 IEEE International Conference on Robotics and Automation (ICRA); Brisbane; 2018.

[9] Gebhardt GHW, Daun K, Schnaubelt M, et al. Learning to assemble objects with a robot swarm. In: Proceedings of the 16th Conference on Autonomous Agents and Multi-Agent Systems; 2017.

[10] Kawakami T, Kinoshita M, Watanabe M, et al. An actor-critic approach for learning cooperative behaviors of multiagent seesaw balancing problems. In: IEEE International Conference on Systems, Man and Cybernetics; 2005.

[11] Kuremoto T, Obayashi M, Kobayashi K, et al. A reinforcement learning system for swarm behaviors. In: IEEE International Joint Conference on Neural Networks; 2008.

[12] Lowe R, WU Y, Tamar A, et al. Multi-agent actor-critic for mixed cooperative-competitive environments. In: Advances in Neural Information Processing Systems 30; 2017.

[13] Pugh J, Martinoli A. Multi-robot learning with particle swarm optimization. In: Proceedings of the 5th International Conference on Autonomous Agents and Multi-Agent Systems; 2006.

[14] Pugh J, Martinoli A. Parallel learning in heterogeneous multi-robot swarms. In: IEEE Congress on Evolutionary Computation; 2007.

[15] Martel S, Mohammadi M. Using a swarm of self-propelled natural microrobots in the form of flagellated bacteria to perform complex micro-assembly tasks. 2010;.

[16] Shahrokhi S, Becker AT. Object manipulation and position control using a swarm with global inputs. 2016;.

[17] Peters J, Mülling K, Altun Y. Relative entropy policy search. In: Proceedings of the 24th AAAI Conference on Artificial Intelligence; 2010.

[18] Lagoudakis MG, Parr R. Least-squares policy iteration. Journal of Machine Learning Research. 2003;.

[19] Boyan JA. Least-squares temporal difference learning. In: Proceedings of the 16th International Conference on Machine Learning; 1999.

[20] Hoffman MW, Lazaric A, Ghavamzadeh M, et al. Regularized least squares temporal difference learning with nested l2 and l1 penalization. In: European Workshop on Reinforcement Learning;

2011.
[21] van Hoof H, Neumann G, Peters J. Non-parametric policy search with limited information loss. 2017;.
[22] Kupcsik A, Deisenroth M, Peters J, et al. Model-based contextual policy search for data-efficient generalization of robot skills. Artificial Intelligence. 2015;.
[23] Seeger M, Williams CK, Lawrence ND. Fast forward selection to speed up sparse gaussian process regression. Workshop on AI and Statistics 9. 2003;9.
[24] Snelson E, Ghahramani Z. Sparse gaussian processes using pseudo-inputs. In: Advances in Neural Information Processing Systems 18. MIT Press; 2006. p. 1257–1264.
[25] Smola AJ, Bartlett PP. Sparse greedy gaussian process regression. In: Advances in Neural Information Processing Systems 13; Vol. 13. MIT Press; 2001. p. 619–625.
[26] Aronszajn N. Theory of reproducing kernels. Transactions of the American mathematical society. 1950;.
[27] Hart PE, Nilsson NJ, Raphael B. A formal basis for the heuristic determination of minimum cost paths. IEEE Transactions on Systems Science and Cybernetics. 1968;.
[28] Khatib O. Real-time obstacle avoidance for manipulators and mobile robots. The International Journal of Robotics Research. 1986;.
[29] Koren Y, Borenstein J. Potential field methods and their inherent limitations for mobile robot navigation. In: Proceedings of the IEEE International Conference on Robotics and Automation; 1991.
[30] Brockman G, Cheung V, Pettersson L, et al. Openai gym. arXiv:160601540 [cs]. 2016;.
[31] Becker A, Habibi G, Werfel J, et al. Massive uniform manipulation: Controlling large populations of simple robots with a common input signal. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems; 2013.
[32] Rubenstein M, Ahler C, Nagpal R. Kilobot: A low cost scalable robot system for collective behaviors. In: Proceedings of the IEEE International Conference on Robotics and Automation; 2012.
[33] Bonnard Q, Lemaignan S, Zufferey G, et al. Chilitags 2: Robust fiducial markers for augmented reality and robotics. ; 2013.