Learning Robust Policies for Object Manipulation with Robot Swarms

Gregor H.W. Gebhardt[1] and Kevin Daun[1] and Marius Schnaubelt[1] and Gerhard Neumann[2]

***Abstract*— Swarm robotics investigates how a large population of robots with simple actuation and limited sensors can collectively solve complex tasks. One particular interesting application with robot swarms is autonomous object assembly. Such tasks have been solved successfully with robot swarms that are controlled by a human operator using a light source. In this paper, we present a method to solve such assembly tasks autonomously based on policy search methods. We split the assembly process in two subtasks: generating a high-level assembly plan and learning a low-level object movement policy. The assembly policy plans the trajectories for each object and the object movement policy controls the trajectory execution. Learning the object movement policy is challenging as it depends on the complex state of the swarm which consists of an individual state for each agent. To approach this problem, 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 in the swarm as well as to the allocation of an agent to its position in the swarm. These invariances make the learned policy robust to changes in the swarm and also reduce the search space for the policy search method significantly. We show that the resulting system is able to solve assembly tasks with varying object shapes in multiple simulation scenarios and evaluate the robustness of our representation to changes in the swarm size. Furthermore, we demonstrate that the policies learned in simulation are robust enough to be transferred to real robots.**

I. INTRODUCTION

Nature provides us with a multitude of examples that show how swarms of simple agents are much richer in their abilities than a single individual. This synergy effect is also called superadditivity, which means that “the entire team should be able to achieve much more than individual robots working alone” [1]. Termites, for example, are insects with a body size in the low millimeter range and with simple sensors for their local environment. Still, termite colonies, which consist of up to millions of individual insects, are able to build mounds exceeding the physical properties of a single termite by several orders of magnitude. These synergy effects of swarms are a main principle that swarm robotics aims to exploit. In contrast to traditional robotics, which usually is based on robust machines with sufficient sensory equipment, swarm robots are often simple agents with limited actuation and sensors. Instead, robotic swarms leverage from the 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. However, learning a policy that guides a swarm with simple control rules by an external signal to achieve a complex behavior is a feasible way to overcome this limitation.

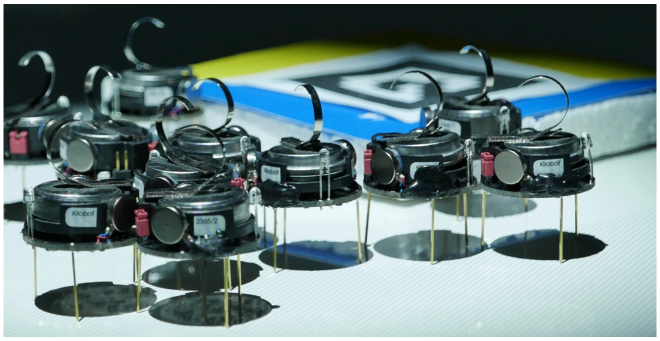

Fig. 1. Kilobots pushing an object in an assembly task. The robots have a diameter of 3.3cm, the object in the background is a square of 15cm width.

In this paper, we will consider autonomous object assembly with a large robot swarm. Recently, large affordable robot swarms such as the Kilobots [2] have become available, which allows for new interesting applications. The Kilobots can sense the ambient light and—using the phototaxis algorithm—they can follow the gradient of the intensity towards a light source. A swarm of Kilobots has been used in an object assembly experiment [3], where a human operator controls multiple stationary light sources to guide the swarm. Formulating this task with control rules to automate the assembly process is, however, a hard task.

Motivated by this application, we present an approach based on policy learning to find a control strategy for the light source. 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 waypoints 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.

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

[1] G.H.W. Gebhardt, K. Daun and M. Schnaubelt are with CLAS, Department of Computer Science, Technische Universität Darmstadt, 64289 Darmstadt, Germany, gebhardt@ias.tu-darmstadt.de

[2] G. Neumann is with LCAS, University of Lincoln, Brayford Pool, LN6 7TS Lincoln, UK, geri@robot-learning.de

we embed into a reproducing kernel Hilbert space [4]. 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 [5] 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. A short overview over this method has been presented previously by the authors in [6].

II. 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, [7], [8] 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, [9] 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 [10], 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 [11]. 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 phototaxis behavior runs on each agent and a desired swarm behavior is achieved by learning the controller for the light source. 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. This makes the evaluation of an executed policy on the agent or a constant communication between a global critic and the agents extremely difficult. In [12], a swarm of flagellated magnetotactic bacteria was used to to build a pyramid out of building blocks in the micrometer range. The bacteria have a flagatella-based propulsion motor and their direction can be 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. Similarly, in [13] a set of PD controllers is presented to control a swarm of phototactic agents. 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).

III. PRELIMINARIES

In this section, we will briefly introduce the policy search method we use for learning the low-level object movement policy. Furthermore, we shortly discuss the embeddings of distributions which we later use to represent the state of the swarm. And finally, we will also depict the planning strategies which we apply to generate trajectories from the high-level object assembly policy.

A. Actor-Critic Relative Entropy Policy Search

We use actor-critic relative entropy policy search (AC-REPS) [5] 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) [14] and consists of three steps: First, we estimate the Q-function using the observed state transitions with least-squares temporal difference learning (LSTD) [15], [16]. Second, we improve the policy by maximizing the expected Q-values for the sampled data using REPS. And third, we obtain a continuous policy by matching a weighted Gaussian process [17].

1) Least-Squares Temporal Difference Learning: 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 $\boldsymbol{\phi}(\boldsymbol{s}, \boldsymbol{a})$ (we will explain in Section IV-A.3 how we design the feature function), the Q-function can be approximated as a linear function in the features $Q(\boldsymbol{s}, \boldsymbol{a}) = \boldsymbol{\phi}(\boldsymbol{s}, \boldsymbol{a})^\intercal \boldsymbol{\theta}$, where $\boldsymbol{\theta}$ are the parameters of the function. We estimate these parameters by least-squares temporal difference learning [15] with L_{22} penalization [16]:

$$\begin{aligned} \boldsymbol{\theta} &= (\boldsymbol{X}^\intercal \boldsymbol{X} + \beta' \boldsymbol{I})^{-1} \boldsymbol{X}^\intercal \boldsymbol{y}, & \boldsymbol{X} &= \boldsymbol{C}(\boldsymbol{A} + \beta \boldsymbol{I}), \\ \boldsymbol{C} &= (\boldsymbol{\Phi}(\boldsymbol{\Phi}^\intercal \boldsymbol{\Phi} + \beta \boldsymbol{I})^{-1}, & \boldsymbol{y} &= \boldsymbol{C}\boldsymbol{b}, \\ \boldsymbol{b} &= \boldsymbol{\Phi}^\intercal \boldsymbol{R}, & \boldsymbol{A} &= \boldsymbol{\Phi}^\intercal(\boldsymbol{\Phi} - \gamma \boldsymbol{\Phi}'). \end{aligned} \tag{1}$$

Here, $\boldsymbol{\Phi} = [\boldsymbol{\phi}(\boldsymbol{s}_0, \boldsymbol{a}_0), \ldots, \boldsymbol{\phi}(\boldsymbol{s}_T, \boldsymbol{a}_T)]^\intercal$ is the feature matrix, $\boldsymbol{R} = [r_0, \ldots, r_T]^\intercal$ is the reward vector, and $\boldsymbol{\Phi}'$ denotes the feature matrix of the next states. Two regularization coefficients are introduced. The coefficient β' adds a l_2 penalty to the projection step while the coefficient β applies a regularization to the fixed-point step to avoid over-fitting. The parameter γ is the discount factor. We found that this L_{22}-regularized LSTD version could deal better with the high-dimensional feature spaces that we will use in this work than standard LSTD due to the improved regularization method.

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 [14]. Inspired by the episodic REPS algorithm [18], 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).

Given the state distribution $\mu(\boldsymbol{s})$ from the current set of samples, we want to maximize the expected Q-value

$$\mathbb{E}[Q(\boldsymbol{s},\boldsymbol{a})] = \int \mu(\boldsymbol{s}) \int \pi(\boldsymbol{a}|\boldsymbol{s}) Q(\boldsymbol{s},\boldsymbol{a}) \,\mathrm{d}\boldsymbol{s}\,\mathrm{d}\boldsymbol{a}. \tag{2}$$

However, as we do not want to solve this optimization problem for each state independently, we maximize instead over the joint state-action distribution $p(\boldsymbol{s},\boldsymbol{a}) = p(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s})$. As the state distribution is not allowed to change, it is required that the 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 $\int p(\boldsymbol{s})\boldsymbol{\phi}(\boldsymbol{s})\, d\boldsymbol{s} = \hat{\boldsymbol{\phi}}$ with the average feature vector $\hat{\boldsymbol{\phi}}$ of all state samples. The resulting constraint optimization problem is now given by

$$\begin{aligned} \arg\max_{p} & \iint p(\boldsymbol{s},\boldsymbol{a}) Q(\boldsymbol{s},\boldsymbol{a}) \,\mathrm{d}\boldsymbol{s}\,\mathrm{d}\boldsymbol{a}, \\ \text{s.t. } & \mathrm{KL}\left[p(\boldsymbol{s},\boldsymbol{a}) || q(\boldsymbol{s},\boldsymbol{a})\right] \leq \epsilon, \\ & \int p(\boldsymbol{s})\boldsymbol{\phi}(\boldsymbol{s})\, d\boldsymbol{s} = \hat{\boldsymbol{\phi}}, \\ & \int p(\boldsymbol{s},\boldsymbol{a})\, d\boldsymbol{s}\, d\boldsymbol{a} = 1. \end{aligned} \tag{3}$$

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

$$p(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s}) \propto q(\boldsymbol{s},\boldsymbol{a}) \exp\left[(Q(\boldsymbol{s},\boldsymbol{a}) - V(\boldsymbol{s}))/\eta\right], \tag{4}$$

where $V(\boldsymbol{s}) = \boldsymbol{v}^{\intercal}\boldsymbol{\phi}(\boldsymbol{s})$ is a state dependent baseline similar to a value function. The Lagrangian multipliers η and $\boldsymbol{v}$ can be obtained efficiently by minimizing the dual function on the samples.

3) Matching a Weighted Gaussian Process: Solving the optimization problem obtained from REPS only gives the desired probabilities $p(\boldsymbol{s}_i,\boldsymbol{a}_i) = p(\boldsymbol{s}_i)\pi(\boldsymbol{a}_i|\boldsymbol{s}_i)$ for the samples in $\mathcal{D}$. To generalize this sample-based policy representation to a representation $\tilde{\pi}$ that generalizes for the whole state-action space, the expected KL divergence between π and $\tilde{\pi}$ can be minimized. This step is equivalent to a weighted maximum likelihood estimate of $\tilde{\pi}$ with sample weights [18]

$$w_i = \exp\left((Q(\boldsymbol{s}_i,\boldsymbol{a}_i) - V(\boldsymbol{s}_i))/\eta\right). \tag{5}$$

With these weights, $\tilde{\pi}$ can be approximated with a weighted Gaussian Process (GP) [17], where the action $\boldsymbol{a}_i$ for state $\boldsymbol{s}_i$ is weighted by w_i. A sparse formulation of the Gaussian process [19] allows to keep the computations tractable.

B. Kernel Embeddings of Distributions

Kernel embeddings allow a nonparametric representations of distribution with arbitrary shapes. We will use this representation later as a feature mapping for the state of the swarm. Let $\mathcal{H}$ be a reproducing kernel Hilbert space (RKHS) of functions, uniquely defined by a positive definite kernel function $k(\boldsymbol{x},\boldsymbol{x}') := \langle \boldsymbol{\psi}(\boldsymbol{x}), \boldsymbol{\psi}(\boldsymbol{x}')\rangle_{\mathcal{H}}$ [20]. Here, the feature mappings $\boldsymbol{\psi}(\boldsymbol{x})$ are often intrinsic to the kernel functions and might map into an infinite dimensional feature space. We can embed a marginal distribution $p(X)$ as the expected feature mapping of its random variable $\boldsymbol{\mu}_X := \mathbb{E}_X\left[\boldsymbol{\psi}(X)\right] = \int_{\Omega} \boldsymbol{\psi}(\boldsymbol{x})\,\mathrm{d}p(\boldsymbol{x})$ [4]. In practice we estimate the embedding from samples as

$$\hat{\boldsymbol{\mu}}_X = \frac{1}{m}\sum_{i=1}^{m} \boldsymbol{\psi}(\boldsymbol{x}_i) = \frac{1}{m}\sum_{i=1}^{m} k(\boldsymbol{x}_i, \cdot). \tag{6}$$

C. Planning Strategies

A* is a heuristic search algorithm commonly applied for graph search problems [21]. 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 cheapest costs from $\boldsymbol{s}$ to the goal state $\boldsymbol{s}_G$. The cost $g(n_s)$ for reaching a node n_s can be computed by tracing the path from n_s back to the start node and recursively summing up the sub-path costs, i.e.,

$$g(n_s) = g(\mathrm{pred}(n_s)) + c(\mathrm{pred}(n_s), n_s), \tag{7}$$

where $\mathrm{pred}(n_s)$ is the parent node of n_s and $c(n_{s_1}, n_{s_2})$ is the cost of the path between nodes n_{s_1} and n_{s_2}.

Potential fields [22] 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 attraction potential and the repulsive potential for an obstacle $\boldsymbol{o}$ are defined as

$$U_{\text{att}}(\boldsymbol{s}) = \tfrac{1}{2}\chi d(\boldsymbol{s},\boldsymbol{s}_G)^2, \text{ and} \tag{8}$$

$$U_{\text{rep}}(\boldsymbol{s},\boldsymbol{o}) = \begin{cases} \frac{1}{2}\chi\left(\frac{1}{d(\boldsymbol{s},\boldsymbol{o})} - \frac{1}{d_o}\right)^2 & \text{if } d(\boldsymbol{s},\boldsymbol{s}_G) < d_o \\ 0 & \text{else} \end{cases}, \tag{9}$$

respectively. Here, d is a measure for the distance to the target state $\boldsymbol{s}_G$ or the obstacle $\boldsymbol{o}$, d_o is the maximum distance to the obstacle, and χ is a scaling factor. Summing up both potentials yields the total potential $U(\boldsymbol{s}) = U_{\text{att}}(\boldsymbol{s}) + \sum_{\boldsymbol{o}} U_{\text{rep}}(\boldsymbol{s},\boldsymbol{o})$. The path of the robot can be computed by following the gradient $\nabla U(\boldsymbol{s})$. 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 IV-B for more details).

IV. Towards Solving the Assembly Task

To fulfill the task of assembling multiple parts of an object into a whole unit, we use three components; an overview is given in Figure 2. First, an assembly policy that describes

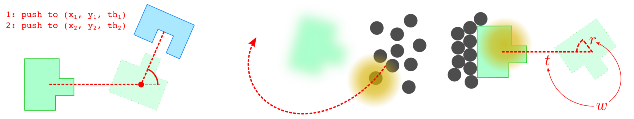
1: push to (x_1, y_1, th_1)
2: push to (x_2, y_2, th_2)
r
t
w

Fig. 2. The three components of our approach. Left: the assembly policy defines waypoints 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.

how the parts should move such that they form a whole unit in the end. Second, a path planning strategy to guide the swarm around the objects and to arrange them for the next pushing task. And third, an object movement policy that realizes basic movements of an object part indirectly by controlling a light source that guides the robot swarm. In the following section, we will describe these components in detail.

A. The Object Movement Policy

The object movement policy controls the position of the light source such that the swarm, which follows the intensity of the light, pushes the object along the trajectory that was generated from the assembly policy. We reduce the search space for the object movement policies by considering only pushes in positive x-direction or counterclockwise rotations. By assuming symmetric objects, we can later apply the learned policies to arbitrary movements by rotating and flipping the state representation accordingly. We further introduce a trade-off parameter ρ that weighs between translational and rotational movements. This trade-off is achieved by the design of the reward function which we introduce in Section IV-A.1. We learn multiple policies to control the light source for discrete settings of the parameter ρ: one policy for pure translation ($\rho = 0$), one for pure counterclockwise rotation ($\rho = 1$), and an arbitrary number of policies for different ratios ρ of combined translational and rotational movements. In our experiments we usually learned object movement policies for five settings of ρ.

1) The Reward Function: The reward function reflects the setting of the trade-off parameter $\rho \in [0, 1]$. The function rewards a purely rotational movement for $\rho = 1$ and a purely translational movement for $\rho = 0$. For values in between, ρ trades off the rotational against 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

$$r(\rho) = \rho\, r_{\text{rot}} + (1 - \rho) r_{\text{trans}} - c_y d_y, \tag{10}$$

with the translational and rotational reward terms

$$r_{\text{trans}} = d_x - c_\theta\, d_\theta, \quad \text{and} \tag{11}$$
$$r_{\text{rot}} = d_\theta - c_x\, d_x, \tag{12}$$

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

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 $\boldsymbol{s} := [\boldsymbol{l}, \boldsymbol{b}_1, \dots, \boldsymbol{b}_n]$. The action vector $\boldsymbol{a} = (a_x, a_y)$ is the desired displacement of the light source in x- and y-direction.

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 actual positions. Therefore, we represent the state of the swarm as a distribution embedded into a RKHS [4], where we treat each agent as a sample of a distribution. This representation as distribution is invariant to both, the allocation of the individual agent to the position (i.e., which agent is at which position), as well as to the number of agents in the swarm. Thus, the state of the swarm is represented as

$$\mu_{\boldsymbol{b}}(\cdot) = \frac{1}{n} \sum_{i=1}^{n} k(\boldsymbol{b}_i, \cdot) = \frac{1}{n} \sum_{i=1}^{n} \psi(\boldsymbol{b}_i), \tag{13}$$

where k is a kernel function (e.g., the Gaussian kernel) and ψ is the intrinsic feature mapping of k. With this state representation, we can compute the difference between two swarm distributions independently from the number of agents by computing the squared difference of their embeddings

$$\begin{aligned} d_{\text{b}}(\boldsymbol{b}, \boldsymbol{b}') =& \frac{1}{n^2} \sum_{i=1}^{n} \sum_{j=1}^{n} k(\boldsymbol{b}_i, \boldsymbol{b}_j) - \frac{2}{nm} \sum_{i=1}^{n} \sum_{j=1}^{m} k(\boldsymbol{b}_i, \boldsymbol{b}'_j) \\ &+ \frac{1}{m^2} \sum_{i=1}^{m} \sum_{j=1}^{m} k(\boldsymbol{b}'_i, \boldsymbol{b}'_j). \end{aligned} \tag{14}$$

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

$$d_{\text{v}}(\boldsymbol{v}, \boldsymbol{v}') = -0.5(\boldsymbol{v} - \boldsymbol{v}')^{\intercal} \operatorname{diag}\left(\sigma_{\text{v}}^{-2}\right) (\boldsymbol{v} - \boldsymbol{v}'), \tag{15}$$

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 just states. We can now combine these two distance measures into a kernel function

$$K(\boldsymbol{s}, \boldsymbol{s}') = \exp\left(-\frac{\alpha}{2} d_{\text{v}}(\boldsymbol{v}, \boldsymbol{v}') - \frac{1 - \alpha}{2} d_{\text{b}}(\boldsymbol{b}, \boldsymbol{b}')\right), \tag{16}$$

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 $\mathcal{D}_{\text{ref}} = (\boldsymbol{s}_i, \boldsymbol{a}_i)_{i=1}^{N}$ 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

$$\phi(\boldsymbol{s}, \boldsymbol{a})_i = K((\boldsymbol{s}_i, \boldsymbol{a}_i), (\boldsymbol{s}, \boldsymbol{a})), \quad i = 1, \dots, N \tag{17}$$

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

$$\varphi(\boldsymbol{s})_i = K(\boldsymbol{s}_i, \boldsymbol{s}), \quad i = 1, \dots, N \tag{18}$$

with the same kernel function as in Equation 17.

B. Assembly Policy and Path Planning Strategy

The assembly policy contains the construction information stored as a list of oriented waypoints with required accuracies for each object. These waypoints 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 waypoint is reached. We compute the desired translation-rotation ratio as $\rho_{\text{des}} = e_{\text{rot}}/(e_{\text{trans}} + e_{\text{rot}})$ and choose the closest learned object movement policy to be executed.

After reaching a waypoint, the swarm has to be arranged at the next object. Using the object movement policy for this task is not feasible since the swarm might collide with other objects and the policy is only valid nearby the object to move. Hence, in this case we use a path planning strategy to guide the swarm around objects and arrange them for the next pushing task. This strategy is a combination of A* with potential fields, where we use the potential field in the cost term $c(\boldsymbol{s}_1, \boldsymbol{s}_2)$. Naively, we could also simply follow the gradient of the potential field. This would however bring up several issues such as getting stuck in local minima, avoiding narrow passages, or oscillations around obstacles [23]. Instead, we define the cost function as $c(\boldsymbol{s}_1, \boldsymbol{s}_2) = d(\boldsymbol{s}_1, \boldsymbol{s}_2) + U_{\text{rep}}(\boldsymbol{s}_2)$, where $d(\boldsymbol{s}_1, \boldsymbol{s}_2)$ is a measure of the distance between $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$, and $U_{\text{rep}}(\boldsymbol{s}_2)$ is the repulsive potential from the obstacle in the potential field. As heuristic $h(\boldsymbol{s})$ we use the Euclidean distance to the target state. We plan a single path which we follow with the center of the circular gradient. To avoid that the swarm gets out of reach of the light gradient, the center of the gradient has to stay within a certain range from the mean position of the swarm.

V. Experimental Setup & Results

To evaluate the proposed learning method, we apply it on the Kilobot platform [2]. 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. However, the locomotion technique based on the slip-stick principle restricts the Kilobots to flat surfaces with low friction and furthermore lacks proper odometry. We use phototaxis to control a swarm of robots with a single light source [24]. In order to perform phototaxis, the robot activates one of the two vibration motors and thereby moves slowly forward while rotating. Whenever the ambient light sensor, which is placed at the back of the robot, perceives an increasing light intensity, the rotation direction is switched. This causes the robot to move along the light gradient towards the light source. We learn and evaluate the proposed method first on a 2D Kilobot simulator and learning framework. For this, we have implemented a Kilobot simulator in Python based on the physics engine Box2D[1] in a highly parallelizable fashion to speed up the learning phase on a computing cluster. Later we apply the policies that we have learned in simulation directly to the real Kilobots.

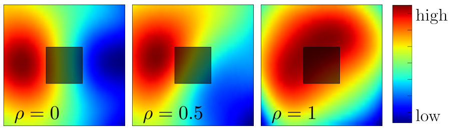

Fig. 3. The value function plots around the object depict how our approach successfully adapts to different settings of ρ.

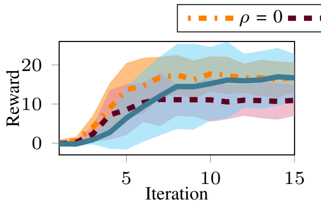

Fig. 4. Mean and 2σ interval of reward for pure translation, pure rotation, and combined movement (ρ = 0, ρ = 1, ρ = 0.5) over 10 runs after 0 to 15 learning iterations.

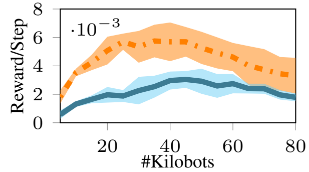

Fig. 5. Average reward per time-step of a pure translation (ρ = 0) and a pure rotation (ρ = 1). All policies are learned with 15 Kilobots and evaluated with 5 to 80 Kilobots.

A. Learning the Object Movement Policy

For learning the object movement policy, we simulate a world with a single square object of 15cm width which is initialized at $(0, 0)$, the world is simulated at 10Hz, however, we only take one SARS sample per second. We sample the initial positions of the Kilobots and the light with two different strategies to obtain a stable learning process. In the first, we initialize the light above the object and the Kilobots uniformly around the object. In the second, we choose the initial position of the swarm and the light by sampling polar coordinates from a a Gaussian with increasing variance for the angle and uniformly in the interval [0.1, 0.75] for the radius. By increasing the variance of the Gaussian, we ensure that the policy is first learned in regions that are more important for the task (i.e., behind the object) and later in regions that are further away from the object. We learn the object movement policies with 15 Kilobots for a square object over 20 iterations. In each iteration we sample 100 episodes with 125 steps per episode. Afterwards, we maintain a set of SARS tuples which we choose randomly from the new samples and the old SARS tuples. To define the feature function for LSTD, we further select 1000 samples randomly from the SARS data set just as another 500 samples as sparse subset for the GP. Figure 3 depicts the value function after 15 iterations for three different translation-rotation ratios ρ. For this visualization we use artificial configurations where all Kilobots and the light are at the same position (x, y).

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

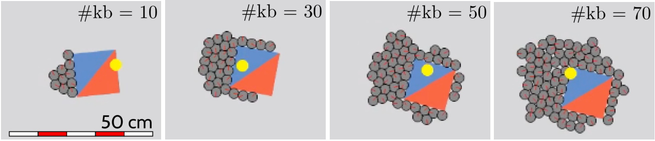

Fig. 6. The pure translation policy learned with 15 Kilobots evaluated on different swarm sizes. With a size of 50 agents and more, the swarm distributes around the object which obstructs the intended push.

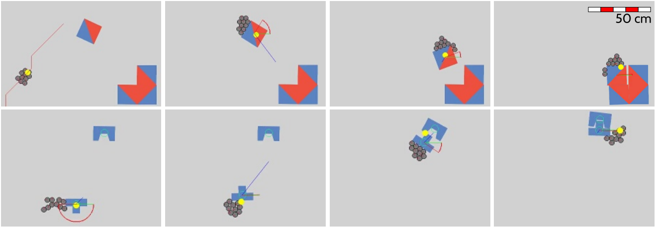

Fig. 7. Two examples for a successful assembly task. The Kilobots are depicted by gray circles and the light position by a yellow circle. A video of both experiments is available at `https://youtu.be/kuU8wsR9dD4`.

For the pure translation $\rho = 0$ the expected reward has its maximum centered on the left side of the object and the minimum on the right side of the object. The value function for the combined action looks similar to the value function for pure translations but is slightly shifted around the top left corner of the object. The value function for the rotation is a skewed circle around the square. Figure 4 shows the average reward during the learning process. The achieved reward starts converging after 7 iterations.

To evaluate how well our approach generalizes to different swarm sizes, we applied the policies learned with 15 Kilobots to swarms with 5 to 80 Kilobots. Figure 5 shows the average reward per step for a pure translation policy ($\rho = 0$) and a pure rotation policy ($\rho = 1$). Up to a swarm size of about 40 Kilobots 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 Kilobots on the average rewards start to decline. The swarm then distributes around the object so that the Kilobots are pushing it from opposing directions and by that obstructing the desired motion. Figure 6 depicts this evaluation for different swarm sizes.

B. The Assembly Task in Simulation

As a first task, the Kilobots have to assemble a larger square composed of four squares. Three squares are already assembled and a fourth square has to be pushed in the remaining gap by translating and rotating the object. The controller uses five object movement policies with rotation-translation trade-off parameters $\rho \in [0, 0.25, 0.5, 0.75, 1]$. The assembly process is shown in Figure 7. First, the swarm approaches the object by following the path generated by the A* algorithm. Afterwards, the object movement polices are applied and the Kilobots push the object along the path generated from the waypoints until the target configuration is reached. The system solves the task in eight out of ten trials. In both failure trials the controller guides the Kilobots around the currently manipulated square into the other squares which destroys the beforehand assembled configuration.

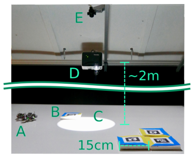

Fig. 8. The Kilobot swarm (A) pushes the assembly objects (B) on a 2m × 1.5m whiteboard. The circular light gradient (C) is projected onto the table by a video projector (D). The scene is observed with an RGB camera (E).

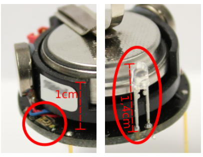

Fig. 9. Modification of the Kilobot hardware, to achieve a good phototaxis behavior. Left: commercially available Kilobot with an SMD light sensor. Right: modified Kilobot with a through-hole diode as in the original design, the SMD sensor is covered with black hot glue.

As a second task, the Kilobots have to assemble a C-shaped object and a T-shaped object. The swarm has to rotate the T-shaped object by 180° before pushing it into the C-shaped object. We use the same five object movement policies learned with a square object as in the previous task. Figure 7 depicts the assembly process. First, the Kilobots rotate the T-shaped object from the starting orientation to the target orientation. Afterwards, the swarm pushes the object close to the final position but rotates the C-shaped object while approaching. Yet, the assembly succeeds after correcting the orientation of the target assembly by rotating the already assembled objects. The system solved the task in six out of ten trials. In all failure trials, the T-shape was pushed inaccurately and disturbed the position of the C-shape. In general, the policies learned with a square were only applicable to a limited extent for pushing the T-shape.

C. The Kilobot Setup

In this section, we will give a short overview of the robotic setup we use to evaluate the proposed method on the real Kilobots. We will also describe how we had to modify the light sensor of the Kilobots and the software we use on the robots as well as the approaches for detecting Kilobots and objects in a camera image. We use a horizontally mounted 2m × 1.5m whiteboard as environment for the swarm. The whiteboard setup is very suitable for the Kilobots as it provides a reflective surface with a low friction which is beneficial for the slip-stick motion of the Kilobots but also for the IR-based communication between the robots. We further emulate a moving light source using a projector mounted vertically on the ceiling and an RGB camera to track Kilobots and objects. The setup is depicted in Figure 8.

1) Light Sensor Adjustment: In contrast to the original design developed at Harvard [25], the commercially manufactured Kilobots[2] have a slightly modified design. The replacement of the through hole (TH) light-sensitive diode at the back with a surface-mounted device (SMD) light sensor at the left side significantly decreases the performance of the phototaxis algorithm. This is mainly caused by two reasons. First, the shifted sensor placement breaks the assumption that the robot should switch its movement direction as soon as the minimum light intensity is detected. Second, as the SMD

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

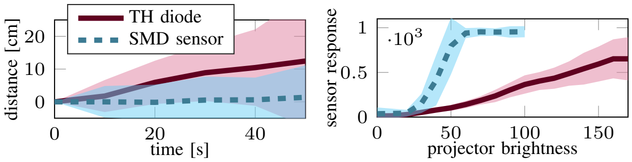

Fig. 10. Left: moved distance in gradient direction with TH diode and SMD sensor. Kilobots with TH diode are about 10 times faster. Right: sensor response curves of SMD sensor and TH diode. The TH diode has a much greater dynamic range. The plots show mean and average over 5 runs with SMD sensor and over 6 runs with TH diode, each with a different Kilobot.

sensor is mounted directly on the printed circuit board (PCB), it is shadowed by the battery. 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 10). This leads to worse performances in scenes with weak gradients as it is the case for a video projector. As the PCB still features the connectors for the TH sensors, restoring the original design is easy. We disabled the SMD sensors by covering them with black hot glue. Figure 10 compares both sensors.

2) Tracking of Kilobots and Objects: 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.

a) High Dynamic Range Images: An efficient method to compute an HDR image from a series of images with varying exposure times is presented in [26]. Each camera has a unique, unknown response curve $f^{-1}(y_{ij}) = t_i x_j =: I(y_{ij})$ that maps the product $I(y_{ij})$ of the true pixel values (irradiances) x_j (i.e., the light intensity at the pixel) with the exposure time t_i to the observed pixel values y_{ij}. Assuming that we would know the inverse of the response curve $I(y_{ij})$, we could compute the true pixel value as

$$x_j = \frac{\sum_i \omega(y_{ij}) I(y_{ij}) t_i}{\sum_i \omega(y_{ij}) t_i^2}. \tag{19}$$

Here, $\omega(y_{ij})$ is a bell shaped function that puts a higher weight on values in the middle of the camera range which are considered to be less noisy. Fitting a response curve to data from the camera sensor can be achieved by the non-linear least-squares optimization problem

$$\min_I \sum_{i,j} \omega(y_{ij})(I(y_{ij}) - t_i x_j)^2. \tag{20}$$

Since we need to solve for both, $I(y_{ij})$ and x_j, the solution can be found iteratively using the Gauss-Seidel relaxation

$$\forall m : E_m = \{(i,j) : y_{ij} = m\} \tag{21}$$

$$I(m) = \text{Card}(E_m)^{-1} \textstyle\sum_{i,j \in E_m} t_i x_j, \tag{22}$$

where E_m is the index set of the sensor value m and $\text{Card}(E_m)$ its cardinality, i.e., how often m has been observed in all images. Initially, we use a linear mapping $I(m)$. Later, the HDR images can then be computed directly by Equation 19. To remap the image from the HDR space to the 8-bit image space we apply an adaptive logarithmic mapping [27]. Figure 11 depicts the generation of an HDR image.

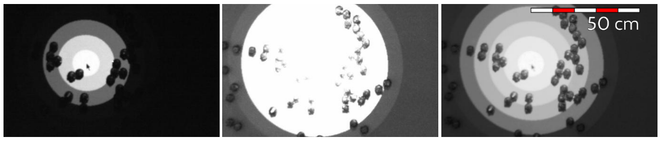

Fig. 11. Images with different exposure times are combined into a HDR image. Left: short exposure; middle: long exposure; right: HDR image.

b) Object Tracking: To achieve a stable and robust tracking of the pose of arbitrary objects, we mark the objects with Chilitags [28]. Chilitags are precise, reliable and illumination tolerant 2D fiducial markers and thus are well suited for the experimental setup.

c) Kilobot Tracking: Due to the small size of the Kilobots in the camera image, it is not feasible to employ the tracking with Chilitags. However, the round geometry of the robots makes this a well-suited problem for a Hough circle transform (HCT). 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. Hence, a rough tracking with HCT is sufficient.

3) Kilobot Phototaxis Control: To control the Kilobot swarm we project a circular gradient (ca. 40cm diameter) into the scene. As long as the Kilobots are within the radius of the gradient, they follow the gradient towards its center using a phototaxis controller. Each Kilobot turns either left or right, however as they turn around the left or right rear leg and not around their center this movement includes always also a translational forward component. The phototaxis controller switches the turn direction, when the sensed light intensity is greater than the previous measurement.

D. The Assembly Task on the Kilobots

We transferred the experiment described in Section V-B to the system introduced in Section V-C using 12, 15, and 24 Kilobots. 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 the five policies learned in simulation (Section V-B) directly to the Kilobot platform. No further optimization on the real robots is required. The system is able to push the fourth quad close to the others with 12 Kilobots. Yet, only around half of the swarm remains in an area where the gradient has an influence when approaching the final position. Hence, the robot swarm is not able to finish the assembly by correcting the orientation of the missing quad. When we used 15 Kilobots the assembly succeeds. Although again many robots fail to follow the light source and diverge in the scene, the amount of Kilobots that remain in the area of the gradient is sufficient to finish the assembly task. Still, the other objects are shifted during the assembly process.

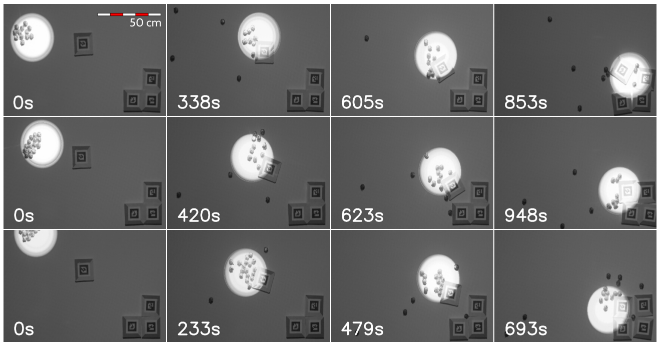

Fig. 12. Assembly of a square part with three similar parts into a big square with different swarm sizes. In the first row: 12 Kilobots, in the second row: 15 Kilobots, in the third row: 24 Kilobots Multiple robots are lost during the run, larger swarm sizes lead to better performances and faster execution.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

A swarm size of 24 Kilobots reduced the time necessary to complete the assembly task from around $950s$ to about $700s$. Four still pictures from the runs with 12, 15, and 24 Kilobots are depicted in Figure 12.

VI. Conclusion

In this paper, we have introduced a method to learn assembly tasks for swarm robots with a single global control signal. We have divided this problem into three components: an assembly policy, which we assume as given; a path planning strategy; and a object movement policy which we learn with AC-REPS. For the 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 have shown in simulation that this approach is able to solve an assembly task for objects on which the policy has been learned but also for new object shapes. We have demonstrated that the learned policy can be directly transferred to the real robots without additional learning.

Acknowledgment

This project has received funding from the European Unions 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] L. E. Parker, "Multiple mobile robot systems," in *Springer Handbook of Robotics*, B. Siciliano and O. Khatib, Eds. Springer Berlin Heidelberg, 2008, pp. 921–941.

[2] 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*, 2014.

[3] M. Rubenstein, A. Cabrera, J. Werfel, G. Habibi, J. McLurkin, and R. Nagpal, "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.

[4] A. Smola, A. Gretton, L. Song, and B. Schölkopf, "A hilbert space embedding for distributions," in *International Conference on Algorithmic Learning Theory*, 2007.

[5] C. Wirth, J. Fürnkranz, and G. Neumann, "Model-free preference-based reinforcement learning," in *Proceedings of the 30th AAAI Conference on Artificial Intelligence*, 2016.

[6] G. H. W. Gebhardt, K. Daun, M. Schnaubelt, A. Hendrich, D. Kauth, and G. Neumann, "Learning to assemble objects with a robot swarm," in *Proceedings of the 16th Conference on Autonomous Agents and Multi-Agent Systems*, 2017.

[7] T. Kawakami, M. Kinoshita, M. Watanabe, N. Takatori, and M. Furukawa, "An actor-critic approach for learning cooperative behaviors of multiagent seesaw balancing problems," in *IEEE International Conference on Systems, Man and Cybernetics*, 2005.

[8] T. Kuremoto, M. Obayashi, K. Kobayashi, H. Adachi, and K. Yoneda, "A reinforcement learning system for swarm behaviors," in *IEEE International Joint Conference on Neural Networks*, 2008.

[9] R. Lowe, Y. WU, A. Tamar, J. Harb, O. Pieter Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," in *Advances in Neural Information Processing Systems 30*, 2017.

[10] J. Pugh and A. Martinoli, "Multi-robot learning with particle swarm optimization," in *Proceedings of the 5th International Conference on Autonomous Agents and Multi-Agent Systems*, 2006.

[11] ——, "Parallel learning in heterogeneous multi-robot swarms," in *IEEE Congress on Evolutionary Computation*, 2007.

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

[13] S. Shahrokhi and A. T. Becker, "Object manipulation and position control using a swarm with global inputs," 2016.

[14] J. Peters, K. Mülling, and Y. Altun, "Relative entropy policy search," in *Proceedings of the 24th AAAI Conference on Artificial Intelligence*, 2010.

[15] J. A. Boyan, "Least-squares temporal difference learning," in *Proceedings of the 16th International Conference on Machine Learning*, 1999.

[16] M. W. Hoffman, A. Lazaric, M. Ghavamzadeh, and R. Munos, "Regularized least squares temporal difference learning with nested l2 and l1 penalization," in *European Workshop on Reinforcement Learning*, 2011.

[17] H. van Hoof, G. Neumann, and J. Peters, "Non-parametric policy search with limited information loss," 2017.

[18] A. Kupcsik, M. Deisenroth, J. Peters, L. Ai Poh, V. Vadakkepat, and G. Neumann, "Model-based contextual policy search for data-efficient generalization of robot skills," *Artificial Intelligence*, 2015.

[19] E. Snelson and Z. Ghahramani, "Sparse gaussian processes using pseudo-inputs," in *Advances in Neural Information Processing Systems 18*, 2005.

[20] N. Aronszajn, "Theory of reproducing kernels," *Transactions of the American mathematical society*, 1950.

[21] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, 1968.

[22] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *The International Journal of Robotics Research*, 1986.

[23] Y. Koren and J. Borenstein, "Potential field methods and their inherent limitations for mobile robot navigation," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 1991.

[24] A. Becker, G. Habibi, J. Werfel, M. Rubenstein, and J. McLurkin, "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.

[25] M. Rubenstein, C. Ahler, and R. Nagpal, "Kilobot: A low cost scalable robot system for collective behaviors," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2012.

[26] M. A. Robertson, S. Borman, and R. L. Stevenson, "Dynamic range improvement through multiple exposures," in *Proceedings of the IEEE International Conference on Image Processing*, 1999.

[27] F. Drago, K. Myszkowski, T. Annen, and N. Chiba, "Adaptive logarithmic mapping for displaying high contrast scenes," in *Computer Graphics Forum*, 2003.

[28] Q. Bonnard, S. Lemaignan, G. Zufferey, A. Mazzei, S. Cuendet, N. Li, A. Özgür, and P. Dillenbourg, "Chilitags 2: Robust fiducial markers for augmented reality and robotics." 2013. [Online]. Available: http://chili.epfl.ch/software