A Reinforcement Learning Approach to Multi-Robot Planar Construction

Caroline Strickland[1], David Churchill[1], Andrew Vardy[2]

***Abstract*— We consider the problem of shape formation in a decentralized swarm of robots trained with reinforcement learning. Shapes are formed from ambient objects which are pushed into a desired pattern. The shape is specified using a projected scalar field that the robots can locally sample. This scalar field plays a similar role to the pheromone gradients used by social insects such as ants and termites to guide the construction of their sophisticated nests. The overall approach is inspired by our previously developed *orbital construction* algorithm. In this paper, we use reinforcement learning to automatically learn policies that accomplish shape formation without the need for hand-coding algorithmic solutions for each desired shape. The particular research questions addressed in this paper are as follows: (1) The performance of learned policies versus the original hard-coded orbital construction algorithm; (2) The performance of the system on more shapes than were considered for the original algorithm. (3) The impact of the number of robots used in training and then subsequently in testing; We provide experimental results using a custom two-dimensional physics simulator of an environment containing circular robots and objects.**

I. Introduction

We are interested in the capacity of swarms of robots to form shapes using objects in their environment. This has direct application to topics such as cleaning (merging all waste objects into a single cluster) and recycling (segregating objects by type and moving them to desired collection points). It may also be a useful capacity in construction-related tasks such as forming walls and enclosures. In [10] a new algorithm which could form various enclosed shapes based on sensing a scalar field which acting as a template to specify the shape. This algorithm, called *orbital construction* (OC) is guided by the scalar field to orbit the growing structure, pushing objects inward or outward to join the structure. The OC algorithm is intentionally minimalistic which allows for easy implementation and reduces the need for parameter tuning. But it represents only one point in the space of possible solutions. In this paper we employ reinforcment learning (RL) to more fully explore the space of possible control policies for this problem. RL provides a robust and natural means for the coordination of action choices between agents in multi-agent systems [2]. We compare the performance of learned policies using RL to the OC algorithm and consider the problem of learning to construct different shapes. While this is a multi-robot system approach, we also examine whether policies learned on a small number of robots are applicable on a larger number. Our experimental results were obtained on a custom 2d physics simulator optimized for collisions between circular robots and objects.

Collective construction by swarms of robots is a topic which takes inspiration from social insects such as ants, bees, and termites who create very sophisticated structures in a decentralized manner [1]. One proposed mechanism to explain some aspects of social insect construction is the use of a template, often consisting of some sensed environmental parameter or pheromone emitted by the insects themselves [9]. A well-known example of such a template is the pheromone emitted by the queen in a colony of termites (genus *Macrotermes*). The concentration of this pheromone is used to guide the construction of a "royal chamber" enclosing the queen. In a recent review of collective robot construction, numerous different organizing principles were identified, including the use of such templates [4]. Experiments have been conducted on physical robots showing the construction of walls where the template is sensed as a light field [6] or through sensing landmarks combined with odometry [5]. The approach taken in this paper assumes a set of robots can sense such a template, which we refer to as a scalar field. A desired shape is built by pushing objects inwards or outwards to lie within a specific contour of the scalar field.

The results we obtain here come from a custom simulation environment designed to allow efficient policy learning using reinforcement learning (RL). So this paper, like [10], assumes that the scalar field can be sensed in a future hardware realization. This might be achieved by performing on-board localization and then consulting a stored scalar field image, or by using a projected light field [6], or perhaps by reading the scalar field from a printed surface.

The orbital construction algorithm [10] is based on this ability to locally sample the scalar field. A fixed threshold value specifies a contour line of the scalar field. If two thresholds are specified, we can define a region which is to be filled in with objects. In the simplest case, the scalar field is defined by a single seed point at scalar value 1 with every other point having a value of $1 - d$ where d is the Euclidean distance from the seed point. In this case, specifying two thresholds defines an annulus. Figure 1 shows the performance of our RL approach in constructing such an annulus. For the original OC algorithm, two different types of agents were defined—*innies* which nudge objects outwards from the seed point, and *outies* which nudge objects inwards. This algorithm combines the aspect of using the scalar field

[1]Department of Computer Science, Memorial University of Newfoundland, St. John's, Canada, cts321@mun.ca, dave.churchill@gmail.com

[2]Department of Computer Science, Department of Electrical and Computer Engineering, Memorial University of Newfoundland, St. John's, Canada, av@mun.ca

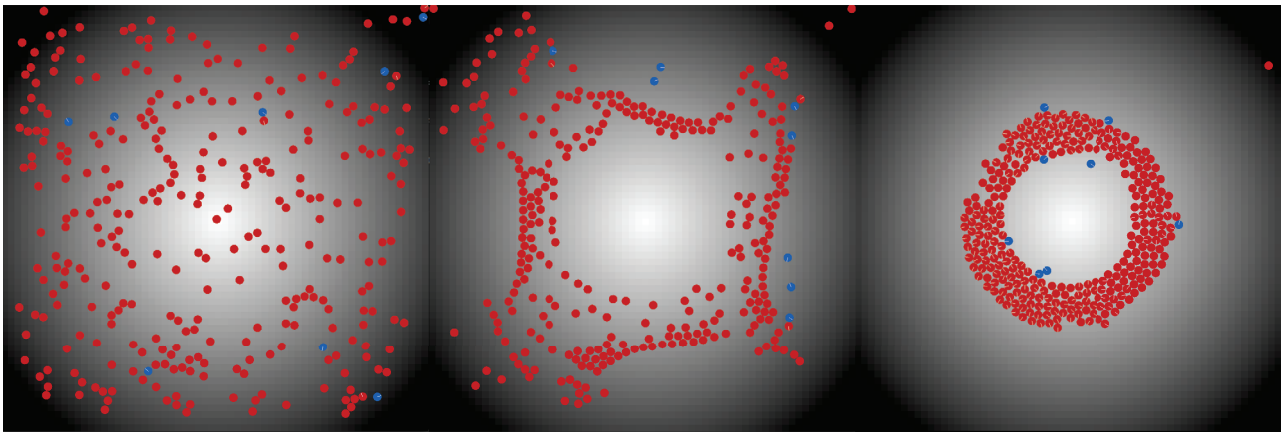


Fig. 1: Three states of our environment demonstrating the progression of RL annulus construction using 8 robots (blue circles) and 250 pucks (red circles). Shown are an initial randomized configuration of pucks (left), an intermediary state during learning/construction (middle), and a successfully constructed shape (right). The background color of each state shows the value of the environment's projected scalar field, ranging from a value of 0 (black) to 1 (white), which guides shape formation.

for guidance with the minimalistic approach to clustering objects discovered by Gauci et al. [3]. In this approach, robots alternate between veering to the left in response to a sensed object and a default behaviour of veering to the right when the object is not sensed. The oscillation between these two responses leads a robot to move towards the outer edge of an object, thus nudging it inwards. The OC variant of this concept uses the scalar field to define the directionality of these movements so that robots can nudge objects inwards or outwards depending upon what is desired. For *outies* objects are nudged inwards until the threshold value of the scalar field is reached. This algorithm has a number of attractive properties: (1) it is minimalistic and requires only a coarse sensing of whether objects lie in the left or right field-of-view; (2) the shape constructed can be varied just by changing the scalar field; (3) robots tend to circumnavigate clockwise about the growing structure and stay out of each other's way; (4) no special grasping capability is required—the robots just bump into the objects to move them (assuming sufficient mass).

However, we identified a need to further explore the space of algorithms which aim to construct a desired shape using guidance from a scalar field. OC does not perform well in defining the interior of concave shapes. Also, it has no built-in strategy to fully explore the environment for objects not yet included in the structure. While there are a number of techniques that we could have implemented (for example, evolutionary algorithms), an RL approach was felt to be promising — it is straightforward to define a reward function for this problem, and our environment lends itself nicely to an efficient state representation.

RL is a subfield of machine learning that is focused around the environment rewarding agents taking desired behaviours and punishing undesired ones. Such systems are well-suited to robotics, as agents often interact with complex environments through unique sensors, actuators, and techniques to reach a state or objective. RL applied to a multi-agent system is called MARL (multi-agent reinforcement learning). A main feature that MARL offers is flexibility — a multi-agent decentralised system can have agents added or removed without the need for rewriting control algorithms. This allows for more robust solutions due to its ability to cope with failure. Using simulators paired with MARL, we can learn policies that result in efficient actions for pattern formation. This would be an improvement over hard-coded algorithms, as it removes dependence from often flawed control algorithms. In addition, RL eliminates the need for writing control algorithms in the first place — which tend to be extremely task-specific and time-consuming. Learning a policy takes into account a variety of variables and representation techniques that must be predetermined, which grants the user control of specific aspects before learning begins.

In society, learning is an essential component of intelligent behavior. However, each individual agent does not need to learn everything from scratch or by their own discovery. Instead, they can exchange information and discoveries with each other and learn from their peers. The same can be said for MARL systems [8]. Policy transfer between simulation runs involves agents following a previously learned policy trained either in a different environment or on a different number of agents. We expect that policy transfer will be able to help us in a number of ways, particularly if we can learn a policy on a single agent quickly and then apply that policy to an n-agent environment, where learning would otherwise have taken longer.

Three environment states are shown in Figure 1, demonstrating the construction of an annulus over time, with the scalar values of the tiles representing distance from the center. However, in this paper we discuss how RL can be used to create more complex shapes that may not be possible via the orbital construction algorithm - for example, non-symmetrical shapes containing right angles. RL methods should be able to learn to construct a variety of shapes without algorithmic changes or parameter tuning, furthering the argument that RL methods offer a more robust solution than hand-coded algorithms.

II. Orbital Construction in a Reinforcement Learning Framework

To eliminate the need for hand-coding algorithms such as the OC algorithm previously described, we have chosen to use RL in an attempt to automatically learn the robot actions necessary to construct desired shapes. In order to apply RL methods to the OC problem, we must frame it in an RL context. An RL problem is defined by several key elements: environment, state, action, reward, policy, and values. The solution method to that problem is some algorithm which carries out the process of generalized policy iteration to learn a policy for the agent(s) to follow. We will now describe each of these elements within the context of the OC problem.

A. Environment

The environment of an RL problem is the physical or simulated world in which the agent(s) operate. The environment used for our OC experiments is the same as described by Vardy in [10], an example of which can be seen in Figure 1. In this environment, circular agents are free to move around in a rectangular enclosed space which is also occupied by circular *puck* objects whose position can be changed by applying a force to them via an agent collision. In addition to the rectangular boundary, it contains a *scalar field* grid projected onto the 'floor' surface, where the grid values range between 0 and 1. The scalar field is illustrated in Figure 1 with the grid being represented by a discrete gray-scale background image. The lightest spaces have a scalar value of 1, while the darkest have a scalar value of 0. This scalar field encodes information relating to the final shape to be constructed by the algorithm. The simulation of this environment is implemented via the CWaggle open-source software project, described later.

B. State

A *state* of an environment is the instantaneous perception of the current configuration of the environment from some viewpoint. MARL methods typically use one of two forms of state representation: a *global* state recording the positions of all agents and objects in the environment, or a *local* state storing just the sensory information available to any given agent at some time step. As global state information is rarely available in a real-life robotics setting, we chose the latter local representation for our OC RL implementation. One key result of this choice is that our OC RL method will learn a single policy that will be followed by each agent in the environment, a much simpler task than using global states to learn a separate policy for each agent.

In most RL problems, the smaller the state representation, the easier the learning task becomes [7]. Also, the less complex the sensor configuration, the more feasible it becomes to implement as a real-world swarm robot system. For both of these reasons, we wish to implement the smallest possible set of sensors that can facilitate the OC RL task. This minimum functionality requires that our agent a) be able to sense pucks in its vicinity, and b) detect its scalar field value and heading. The sensory configuration that was used for our experiments can be seen in Figure 2.

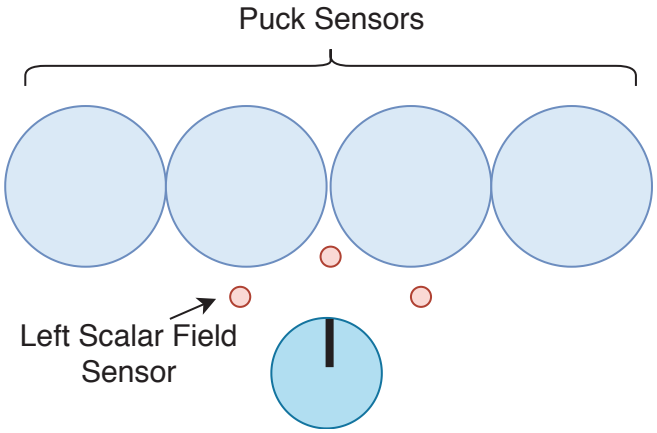


Fig. 2: The sensor configuration for each agent. Agent shown on bottom as a teal circle with its current heading (black line). Sensors are rigidly fixed relative to the position and heading of the agent, moving with the agent as it moves.

There are 2 types of sensors for each agent, each positioned symmetrically about forward heading of the robot. The first sensors are the 4 circular *puck sensors* which give a reading of 1 or 0 whether representing whether a puck in the environment currently intersects with the radius of the sensor. For our experiments, we combine the 2 left of center puck sensors as the 'left puck sensor' and the two right of center sensors as the 'right puck sensor', which approximates the left and right rectangular puck sensors described in [10]. Then there are 3 *scalar field sensors* which can read the floating point value of the scalar field directly below the center of each sensor. For our experiments we require each state have a finite length binary representation, and so we discretize this real number scalar field value svr into an integer svi, based on dividing the range of [0, 1] into n equally sized areas using the equation $svi = \lfloor svr * n \rfloor$. Next, instead of simply encoding all 3 field sensor values in the state, we employ a manipulation to both minimize state representation and determine the relative agent orientation. We use the middle field sensor to obtain the agent's scalar field position value, and the relative values to the middle sensor of the left and right field sensors to determine the agent's heading within the scalar field. For example, if both the left and right field sensors have a value less than the middle sensor, we know the agent is heading toward a 'lighter' area of the scalar field. Our final state representation was then formed as a binary string of length $4+log_2(n)$ with the following bits:

PL	PR	FM	FL	FR

- **PL** (puck left): 1 bit representing if either of the 2 left puck sensors is active (1) or inactive (0)
- **PR** (puck right): 1 bit representing if either of the 2 right puck sensors is active (1) or inactive (0)
- **FM** (field mid): an integer of $log_2(n)$ bits representing the svi of the middle field sensor
- **FL** (field left): 1 bit representing if the left field sensor is less than the field mid sensor (1) or not (0)
- **FR** (field right): 1 bit representing if the right field

sensor is less than the field mid sensor (1) or not (0)

For example if we use $n = 16$ field divisions, this would yield $2^{4+log_2(16)} = 2^8 = 256$ possible states for an agent. If we then have an agent with: left puck sensor active (PL=1), right puck sensor inactive (PR=0), field mid sensor value of 0.4 (FM=$\lfloor 0.4 * 16 \rfloor$=6=0110), field left sensor value of 0.35 (FL=1), field right sensor value of 0.45 (FR=0), this would yield a state presentation of 10011010 (154 of 256 possible). The state observed by an agent at time step t of the environment simulation is denoted as s_t.

C. Actions

In our OC environment, each agent moves forward each time step at a constant speed. The only action that an agent can take is to rotate a given angle for the current time step. This means that the decision making process for OC RL involves only deciding on which angle to rotate on every time step. For our RL framework, we discretize the real-valued space of actual turn angles into a finite set of turning angles that can be chosen to guide the agent.

D. Reward

The reward signal is arguably the most important element of any RL problem, as it defines the objective measure that is to be optimized by the learning process. Intuitively, the reward for our OC RL framework should become more positive as the pucks in the environment get closer to the desired formation. The desired formation for our OC environment is defined by a given scalar field, along with two threshold values: an inner threshold T_i, and an outer threshold T_o. Due to the nature of the scalar field, we must be sure to choose these thresholds such that $T_i > T_o$. We can therefore construct our reward signal as a function of the distances of the pucks in the environment from the desired location within the scalar field.

In an environment with P pucks, we define $SV_t(P_i)$ as the scalar field value located at the center of puck $1 \leq i \leq P$ at simulation time step t. Next we define a distance function $D_t(P_i)$ which yields 0 if the puck is inside the desired thresholds T_i and T_o at time step t, or the difference from the closest threshold if outside it:

$$D_t(P_i) = \begin{cases} SV_t(P_i) - T_i, & \text{if } SV_t(P_i) > T_i \\ T_o - SV_t(P_i), & \text{if } SV_t(P_i) < T_o \\ 0, & \text{otherwise} \end{cases}$$

We then define a global evaluation function $Eval_t(E)$ on an environment E with which averages $D_t(P_i)$ for all pucks:

$$Eval_t(E) = 1 - \frac{1}{P} \sum_{i=1}^{P} D_t(P_i)$$

which ensures $0 \leq Eval_t(E) \leq 1$. An ideally constructed shape will therefore yield a reward $Eval_t(E) = 1$. Our final RL reward function R_t for an environment at time step t then simply subtracts the current evaluation from the evaluation of the previous time step $t-1$. If the environment has come closer to the desired shape then the reward will be positive:

$$R_t = Eval_t(E) - Eval_{t-1}(E)$$

E. Q-Learning

For our experiments, we chose to implement one of the most basic forms of RL: the tabular Q-Learning algorithm [7]. The value function for Q-Learning is stored as a mapping of state-action pairs $Q(s, a) = v$, the expected future return (sum of rewards) of taking action a at state s. The Q-Learning policy $P(s) = a$ maps states to actions, with action a being chosen as the one which maximizes the value $Q(s, a)$. One iteration of Q-Learning happens with each time step t of the environment simulation. For a given state s_t an action is chosen separately for each agent from the current policy and carried out, advancing the current state s_t to s_{t+1}. Then, the reward R_t is calculated, and the Q-values are updated with the following rule:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(R_t + \gamma max_a Q(s_{t+1}, a_t) - Q(s_t, a_t))$$

Using the new Q-values the policy is updated, and the process repeats. This update is applied once for every agent at every time step using the global reward function R_t. In the above equation, α is the *learning rate*, and γ is the *discount factor*. For all experiments, we fix $\gamma = 0.9$, with α varying between experiments.

F. Environment & RL Implementation

The simulation of this OC RL environment for our experiments was implemented in CWaggle[1], an open-source software robotics simulator specifically designed for the efficient simulation of circular-based physics agents. CWaggle supports static and dynamic circle-circle and circle-line collision resolution, with each agent and puck having mass, radius, and velocity, with constant deceleration. Scalar fields can be defined in CWaggle manually, or loaded from an image saved in jpg or png format. CWaggle is written in C++, and is based on the existing Waggle JavaScript simulator described in [10]. The Q-Learning algorithm was implemented within the CWaggle framework from scratch, and supports the saving and loading of learned values and policies to disk.

III. Experiments And Discussion

Three experiments were carried out to demonstrate the effectiveness of RL for orbital construction. These experiments tested: (1) the overall performance of reinforcement learned policies versus the original OC algorithm, (2) the performance of the system on a variety of projected scalar grids versus the annulus shape that was considered originally by the OC algorithm, and (3) The impact of the number of robots used in training and testing using RL. In this section we will first describe the metrics used to compare the different methods, followed by descriptions of the experiments and discussion of results.

[1]https://github.com/davechurchill/cwaggle

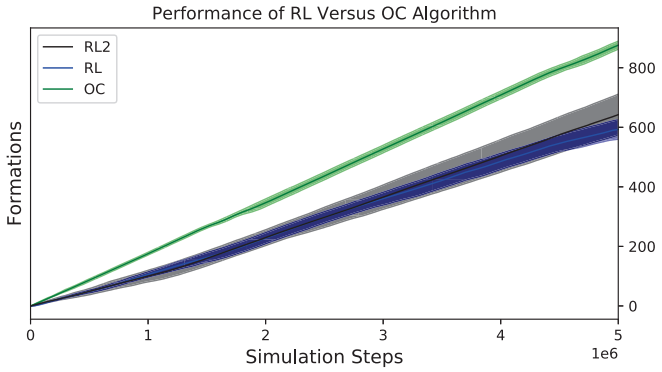


Fig. 3: Experiment 1 Results: OC vs RL Creating Annulus Shape with 8 Agents. OC = Orbital Construction algorithm, RL = Reinforcement Learning from scratch, RL2 = Reinforcement Learning seeded with previously learned policy.

A. Performance Metric

For each experiment, all agents and pucks were set to random positions within the environment. Agents moved forward at a constant speed each time step, with the decision (action) of how much to rotate being given by the various OC methods. Each OC method was carried out to a given maximum number of simulation time steps, and two metrics were kept to determine the effectiveness of each method during that time period: (1) the number of formations that were successfully formed, and (2) the number of times the method got 'stuck' and could not form a formation within specific number of time steps. A formation was deemed successfully formed if the environment E reached $Eval(E) > 0.94$, a threshold value determined by observing repeated shape formation. We noticed that thresholds lower than 0.94 often accept formations dissimilar to the desired shape, while thresholds higher than 0.94 are often challenging to reach and result in the system getting stuck. Once a successful formation was completed, the number of formations was recorded and the environment was 'reset', with each puck and agent being placed in a random position within the environment. A method was deemed to be stuck if no formation reached the threshold evaluation for 150,000 time steps, at which time the environment was reset. RL also had a unique set of parameters which were set to $\alpha = 0.2$, $\gamma = 0.9$, and $\epsilon = 0.0$ (ϵ-greedy) upon configuration of each experiment. Within this framework, we consider one method to be more successful than another if it is able to successfully construct more formations than another within a given amount of time steps. All experiments were run on an Intel i7-7700K CPU processor running at 4.20GHz with 32GB of RAM using Ubuntu 18.04.

B. Experiment 1: RL vs OC Algorithm

We first wanted to test the effectiveness of our RL solution to that of the original OC algorithm on the annulus shape shown in figure 1, to see which method could successfully construct more formations in a given amount of simulation time steps. In this experiment the environment contained 250 pucks and 8 agents, and the maximum number of time steps was set to 5,000,000.

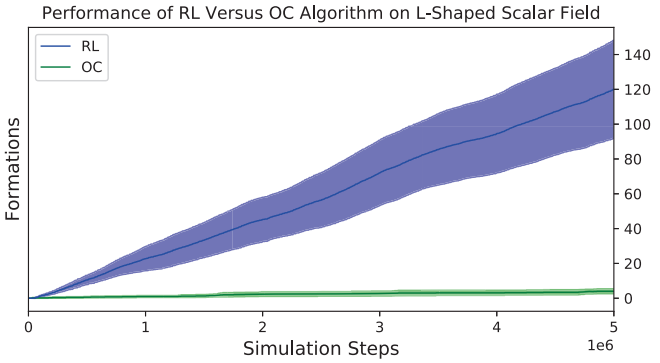
The results of this experiment can be seen in Figure 3, with the x-axis denoting simulation time steps, and the y-axis denoting the number of successfully constructed formations by time step t. Three separate plots are visible, which show the performance of the following solutions over 10 trials: (OC) the original orbital construction algorithm, (RL) the on-line performance of the RL method as it was learning from scratch, and (RL2) the on-line performance of the RL method using a pre-trained policy as a starting point. The pre-trained policy used for RL2 was trained for 5,000,000 time steps using 8 agents. The lines shown are the average performances of each method, with the shaded areas enclosing the best and worst case performances of each method. As expected, the hand-crafted OC algorithm produces formations at a constant rate over time, with its first formations appearing around $t = 5000$. The on-line learning of the RL algorithm requires a longer time period before the first shape is created later at $t = 34000$. The RL2 line demonstrates that seeding the RL algorithm with a pre-trained policy yields better results than seeding it from scratch - an intuitive result which demonstrates the increase in performance of RL as more training steps are allowed. Because the policy used in RL2 had already converged during previous training, RL2 began creating successful formations immediately while the policy used in RL was still converging. Thus, RL2 had the advantage of a running start.

This experiment showed that the RL method is a viable solution for shape construction, yielding 73% as many annulus formations as the OC algorithm, which was specifically designed and tuned by hand to construct only those shapes. Using a single core the experiment ran at 6200 simulation time steps per second, for a total of 13.4 minutes for the 5,000,000 total time steps. While learning from scratch, the RL method took less than 10 seconds to successfully construct its first shape. OC completed **876** successful formations, while RL completed an average of **592**, and RL2 completed an average of **642**. Each of OC, RL, and RL2 had on average one situation where the agents got 'stuck' and the environment was reset.

C. Experiment 2: Scalar Field Variation

Next, we tested how RL compared to OC while sampling an asymmetrical scalar field. Here, we projected an L-shaped scalar field (shown bottom-left in Figure 4) onto the environment, supplying an asymmetrical shape containing a concave feature and right angles. A visualization of this experiment is demonstrated in the bottom center and right images within Figure 4. In the center image, the OC algorithm has difficulty constructing the convex feature and right angles, yielding only 4 successful formations. The RL method was able to construct 120 formations, creating much more pronounced right-angles and concave features.

Demonstrated in the bottom center image of Figure 4, there are often clusters of elements left around the outer ridges of the environment as the 'outies' collect those near the center and push them inwards. With these pucks abandoned, the required accuracy percentage is never met and the desired shape is not formed - there is no way for the agents to venture outside and bring them towards the desired



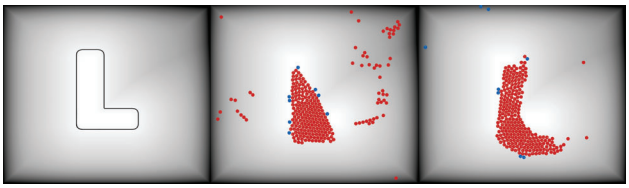


Fig. 4: Experiment 2 Results: Graph of OC versus RL formations on an L-Shaped Projected Scalar Field (top). Below is shown the L-shaped scalar field (left), an OC algorithm construction example (middle), and an RL construction example (right).

scalar value. Our trained RL policies do not have this issue. RL can adjust and recalibrate techniques to suit the scalar field projected underneath. It is not constricted by the rule of clockwise motion and inward rotation around a cluster like the OC algorithm is.

This experiment demonstrates that the RL method is much better suited to construct different shapes than the OC algorithm, which needs to be re-coded and parameters re-tuned to construct each new shape. We performed the same experiment with letters T, I, V, X, Z, and each resulted in a similar outcome, demonstrated in Table I alongside results for the aforementioned L-shape.

	RL		OC	
Shape	Formations	Steps/Form	Formations	Steps/Form
L	120	**41,667**	4	**1,250,000**
T	113.7	**43,975**	7	**714,286**
I	250.2	**19,984**	8	**625,000**
V	209.6	**23,854**	107.9	**64,850**
X	200	**25,000**	28	**178,571**
Z	180	**27,777**	48.7	**102,669**

TABLE I: Averaged results for RL versus OC letter formation on the letters L, T, I, V, X, and Z.

D. Experiment 3: Learned Policy Transfer

Finally, we wanted to test whether a policy learned on a single agent could be applied to multiple agents to construction formations. RL training time per time step increases with the number of agents, for example: training of a single agent resulted in 32,000 simulation time steps/second, while training 8 agents resulted in 6,200 simulation time steps/second. It would be beneficial if we could train a policy on smaller number of agents and then apply it to a larger number. Here, we tested the most extreme case by learning a policy with a single agent ($t = 20,000,000$ simulation steps) and then applied that single pre-learned policy to 2, 4, 8, and 16 agents. The results are shown in Figure 5. The policy transfer was successful, as multiple agents were able to use the single-agent trained policy and complete more formations in fewer time steps than the single agent. While an increase in agents constitutes an increase in formations, it is not a linear increase. With more agents added into the environment, it becomes increasingly congested. There is more potential for collisions between agents (decreasing efficiency), and agents spend more time moving around other agents within the environment.

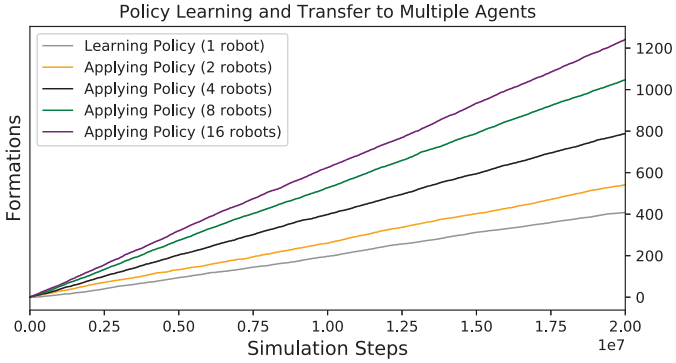


Fig. 5: Experiment 3 Results: Policy Learned on a Single Agent and Transferred to Multiple Agents

IV. Conclusions And Future Work

In this paper we have proposed a system using RL to improve upon previous methods of shape construction. Our system removes dependence on hand-coding algorithms and has the benefit of being able to adapt to unique environments and shape formations. Unlike OC, it does not need distinct types of agents to complete a task, instead agents learn to adapt and fulfill multiple roles based on their current state. The RL solution can also form different types of shapes (such as the L-shape) which the OC algorithm struggles with (likely due to right-angles and concave features). We have also demonstrated the promising initial results for policy transfer, in which policies learned quickly by fewer agents can be carried out by more robots. A video of our results can be seen in https://youtu.be/RzIXQSHFVi4.

For our experiments we used the minimal amount of sensors we deemed necessary to accomplish our task. In doing so, removing the ability of agents to sense other agents. We believe that sensing other agents may be a key element to maximizing cooperation between agents in MARL by reducing costly agent-agent collisions, and so we wish to include such sensors in the future. We believe these experiments have only scratched the surface of using RL for swarm shape formation, as we have shown promising results using the most basic form of tabular Q-learning. In the future we would like to implement deep neural network state representations coupled with deep RL to test the limits of what complex shapes can be formed by more modern learning algorithms.

REFERENCES

[1] Scott Camazine. *Self-organization in biological systems*. Princeton University Press, 2003.

[2] Caroline Claus and Craig Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. *AAAI/IAAI*, 1998:746–752, 1998.

[3] Melvin Gauci, Jianing Chen, Wei Li, Tony J Dodd, and Roderich Gross. Clustering objects with robots that do not compute. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, pages 421–428. International Foundation for Autonomous Agents and Multiagent Systems, 2014.

[4] Kirstin H. Petersen, Nils Napp, Robert Stuart-Smith, Daniela Rus, and Mirko Kovac. A review of collective robotic construction. *Science Robotics*, 4(28), 2019.

[5] T Soleymani, V Trianni, M Bonani, F Mondada, M Dorigo, et al. An autonomous construction system with deformable pockets. Technical report, IRIDIA Technical Report Series (January) 2014-002 (IRIDIA, Université Libre de Bruxelles, Brussels), 2014.

[6] Robert L Stewart and R Andrew Russell. A distributed feedback mechanism to regulate wall construction by a robotic swarm. *Adaptive Behavior*, 14(1):21–51, 2006.

[7] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998.

[8] Ming Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the tenth international conference on machine learning*, pages 330–337, 1993.

[9] Guy Theraulaz, Jacques Gautrais, Scott Camazine, and Jean-Louis Deneubourg. The formation of spatial patterns in social insects: from simple behaviours to complex structures. *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 361(1807):1263–1282, 2003.

[10] Andrew Vardy. Orbital construction: Swarms of simple robots building enclosures. In *2018 IEEE 3rd International Workshops on Foundations and Applications of Self* Systems (FAS*W), Trento, Italy, September 3-7, 2018*, pages 147–153, 2018.