# From rocks to walls: a machine learning approach for lunar base construction

André Tomaz de Menezes
andre.menezes@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

September 2020

**Abstract**

*In-situ* resource utilization is a key aspect for an efficient human exploration of extraterrestrial environments. A cost-effective method for the construction of preliminary structures is dry stacking with locally found unprocessed rocks, which is a challenging task. This thesis focus on learning this task from scratch. Former approaches rely on previously acquired models, which may be hard to obtain in the context of a mission. In alternative, we propose a model-free, data-driven approach. We formulate an abstraction of the problem as the task of selecting the position to place each rock, presented to the robot in a sequence, on top of the currently built structure. The goal is to assemble a wall that approximates a target volume, given the 3D perception of the currently built structure, the next object and the target volume. An agent is developed to learn this task using reinforcement learning. The Deep Q-networks algorithm is used, where the Q-network outputs a value map corresponding to the expected return of placing the object in each position of a top-view depth image. The learned q-function is able to capture the goal and dynamics of the environment. The emerged behaviour is, to some extent, consistent with dry stacking theory. The learned policy outperforms engineered heuristics, both in terms of stability of the structure and similarity with the target volume. Despite the simplification of the task, the policy learned with this approach could be applied to a realistic setting as the high level planner in an autonomous construction pipeline.

**Keywords:** reinforcement learning, dry stacking, *in-situ* resource utilization, autonomous construction, model-free

## 1. Introduction

For a long term human exploration of extraterrestrial environments, such as the Moon, it is essential to use native materials as replacement to resources otherwise brought from Earth, at great expense. This is commonly referred to as *in-situ* resource utilization and, amongst other applications, is useful for the construction of planetary infrastructures [1]. Initial settlement infrastructures, such as roads, platforms and shade walls, may be built with unprocessed or minimally processed local rocks using the ancient method of dry stacking [2]. Although rudimentary, this technique has been proven to produce robust and long lasting structures, while requiring very low pre-processing time and energy. Due to the increased risk and limitations imposed by human missions [3], it is important to have systems with the capability of autonomously setting up infrastructure.

However, assembling a structure from a set of irregularly shaped rocks is a difficult task, usually performed by experienced humans and requiring some amount of intuition. It is not clear how to translate this into an autonomous system, specially in a context in which no models of the environment are available. An increasingly successful approach to autonomously learning such difficult tasks is reinforcement learning. Concretely, the field of deep reinforcement learning has seen a number of breakthroughs in recent years, such as the DQN algorithm [4].

With this work, we propose a formulation for the problem of autonomously building a stable dry stack wall with irregular 3D blocks that is compatible with a model-free reinforcement learning approach. We aim to show that it is possible for an agent to learn this complex task from scratch, without relying on previously acquired object models or internal physical simulation. To accomplish this, a deep neural network architecture is developed to learn a mapping of the state representation to action values using DQN [4].

### 1.1. Related work

Some previous works exploit a physics engine to plan a stable structure from a set of pre-scanned

irregular rocks [5–11]. These works use the physics engine to generate a set of possible poses, from which the best one is chosen according to a defined set of criteria. These criteria usually prioritize lower positions and larger support polygons with normal closer to the vertical direction. Reinforcement learning has been applied to learning such a criterion [9] in the form of a value function that evaluates the possible poses, learned using the DQN algorithm. Some of these works have been applied in an autonomous construction pipeline, used to build vertical towers with up to four stones [7] or complete walls with four courses [11].

To the best of our knowledge, there is no previous work that explores a completely model-free approach to the problem of dry stacking with irregular rocks.

## 2. Background

In a reinforcement learning setting, an agent interacts with an environment in order to learn a behaviour that maximizes a reward signal [12].

At each time step $t$, the agent observes the environment's state $S_t$ and takes an action $A_t$ accordingly. This causes the environment to shift to state $S_{t+1}$, which comes with the reward $R_{t+1}$. The tuple $(S_t, A_t, R_{t+1}, S_{t+1})$ represents the transition at time step $t$. The transition probability distribution $p(s', r|s, a)$ is a property of the environment.

The agent must learn a policy that maximizes the cumulative reward $G_t = \sum_{k=t+1}^{\infty} \gamma^{k-t-1} R_k$, where the discount factor $\gamma$ defines the importance given to delayed rewards. This policy may be obtained implicitly from an estimate of the action-value function $q_\pi(s, a) = \mathbb{E}[G_t \mid S_t = s, A_t = a, \pi]$, by selecting the action that maximizes the expected cumulative reward for a given state. From the definition of $G_t$, it is possible to derive Bellman optimality equation $q_*(s, a) = \mathbb{E}[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \mid S_t = s, A_t = a]$, where $q_*$ is the action-value function for the optimal policy.

### 2.1. Deep Q-networks

The DQN algorithm uses a deep neural network to learn a function approximation $Q$ of the optimal action-value function $q_*$. This is done by minimizing the temporal difference error $\delta_t = R_{t+1} + \gamma \max_a Q(S_{t+1}, a|\mathbf{w}^-) - Q(S_t, A_t|\mathbf{w})$, derived from the Bellman optimality equation. $\mathbf{w}$ and $\mathbf{w}^-$ are the parameters of the current and target networks, used to estimate the action-values. The target network is updated with the parameters of the current network at a defined period and kept constant otherwise for stability reasons. At each interaction with the environment, the agent selects an action according to an exploratory policy (e.g. $\varepsilon$-greedy) based on the current action-value estimates and stores the transition in a replay memory. The network update is then performed by sampling a minibatch of transitions from the replay memory and performing a stochastic gradient descent step (or any improved optimization algorithm) to minimize $\sum_{i \in \text{batch}} \text{loss}(\delta_i)$, with some defined loss function (e.g. quadratic).

### 2.2. Dueling DQN

The action-value function can be factorized into the state-value function and an advantage function as $q_*(s, a) = v_*(s) + a_*(s, a)$. This factorization is motivated by the fact that the state by itself may be good or bad and sometimes the action taken has little influence on the value. With separate representations, the state-value can be learned faster, instead of implicitly learning it for each state-action pair. Wang *et al.* [13] introduced this factorization into the network architecture in the DQN algorithm. The network produces two outputs $V(s|\mathbf{w}_0, \mathbf{w}_v)$ and $A(s, a|\mathbf{w}_0, \mathbf{w}_a)$ that are combined to obtain $Q(s, a|\mathbf{w})$. The weights vector $\mathbf{w}_0$ is common to both estimators and $\mathbf{w} = (\mathbf{w}_0, \mathbf{w}_v, \mathbf{w}_a)$.

### 2.3. Universal Value Functions

The transition probability distribution can be factorized as $p(s', r|s, a) = p(r|s, a, s')p(s'|a, s)$. While the state-transition probability $p(s'|a, s)$ is purely a function of the environment's "physics", the reward distribution $p(r|s, a, s')$ is defined by the underlying goal. Therefore, a more general representation of the environment may be given by $p(s', r|s, a, g) = p(r|s, a, s', g)p(s'|a, s)$, where $g$ represents the goal.

Accordingly, a general value function [14] can be defined as $q_{g,\pi}(s, a) = \mathbb{E}[G_t \mid S_t = s, A_t = a, \pi, g]$, the expected return given the goal. For an environment with a defined state-transition probability, many different general value functions may be learned according to different goals. Each goal is associated with the optimal policy $\pi_{g*}$, corresponding to value function $q_{g*}$.

Schaul *et al.* [15] proposed to unify the set of possible general value functions, $\{q_{g*} : g \in \mathcal{G}\}$, into an universal value function approximator. The idea is to use a deep neural network that takes the $g$ as an additional input, such that $Q(s, a, g\|\mathbf{w})$ is an estimate of $q_{g*}(s, a)$. With this formulation, the function approximation allows to generalize not only over a potentially large state space, but also over a potentially large set of goals.

## 3. Methodology

The proposed approach to learn a dry stacking policy using model-free reinforcement learning is divided in two parts. First, a simplified version of the task is formulated as a RL environment with discrete state and action spaces. Then, an agent architecture is developed to learn a value function for the environment, using DQN.

### 3.1. Environment

The environment must capture the problem of building a dry stack structure with irregular blocks in a target volume. In order to frame it as a learnable RL environment, the problem is simplified to choosing the position for each object in a random sequence of irregular 3D blocks.

At each time step, a new object from the sequence is presented to the agent, as well as the currently built structure and the target volume. The agent must choose a position $\mathbf{x} \in \mathbb{R}^3$ to place the new object, with freedom restricted to the horizontal coordinates $x_0$ and $x_1$. The object's vertical coordinate $x_2$ is defined so that the object is laid on top of the current structure, and with the same orientation as it was presented. To simulate the placement, the velocity of the object is artificially controlled until a support polygon is achieved (at least three contact points). The simulation is then run freely until all objects stabilize. This process is repeated until all objects in the sequence were placed. Figure 1 shows a visualization of the environment, which is simulated using PyBullet[1].

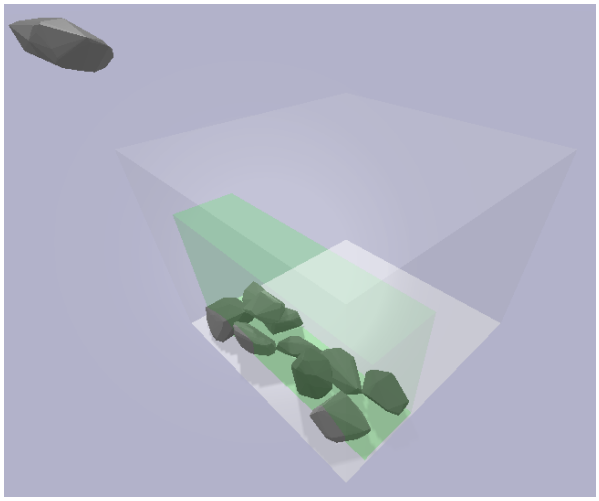
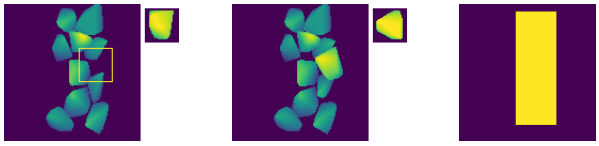


Figure 1: Visualization of the environment. The white box represents the region where the agent can observe the currently built structure, which corresponds to the region where placements are allowed. The green box represents the target volume. The new object is presented at the top left corner of the image. The boxes are represented for visualization and are not physically present in the environment.

A state of the environment is represented as a pair of elevation maps, containing an overhead view of the current structure and a bottom view of the new object (i.e. the side of the object that will be in contact with the structure). An elevation map is an array whose elements represent the elevation (vertical coordinate) at a given discretized horizontal position. The elements are always greater or equal to zero, with zero corresponding to the maximum distance to the view point. Two consecutive states ($s$ and $s'$) are represented in Figures 2(a) and 2(b). An action is given as a pair of indexes $a = (i, j)$, and can be visualized as overlapping the elevation maps of the object and structure with an offset of $i$ rows and $j$ columns. This visualization is represented in Figure 2(a). The goal is represented with an elevation map with the same dimensions as the overhead view, but representing the target volume. An example of a goal $g$ is provided in Figure 2(c).



(a) State $s$ and action $a$. (b) State $s'$. (c) Goal $g$

Figure 2: Visualization of a transition $(s, a, s')$ and the current goal $g$. The action $a = (i, j)$ corresponds to the pixel indexes of the top left corner of the yellow box in the left figure.

### 3.2. Reward shaping

Reward shaping is a key aspect of RL, as the learned optimal behaviours are critically affected by the way rewards are distributed. It must capture the intended goal clearly, while being distributed in a way that facilitates learning.

As stated before, the goal of the environment is to assemble the objects in a way that approximates the target volume. This may be formalized as achieving the highest intersection over union (IoU) between the volume of the built structure and the target. This can be computed from the overhead elevation map $\mathbf{O}$ and the goal elevation map $\mathbf{G}$ as

$$\mathrm{IoU} = \frac{\sum_{i,j} \min\left(o_{ij}, g_{ij}\right)}{\sum_{i,j} \max\left(o_{ij}, g_{ij}\right)}. \tag{1}$$

This metric lays in the range $[0, 1]$, with 1 corresponding to a fully achieved goal.

Although this metric captures the goal of approximating the target volume, it ignores an implicit aspect of the goal statement: a structure should be a stable assembly of the irregular objects. An example of an undesired behaviour that would be rewarded is to place the objects in a way that makes it likely for them to fall inside the target, rather than not falling. A way to avoid this is to discount the contribution of each object to the reward according to the distance between its original pose, where it

[1] E. Coumans and Y. Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. `http://pybullet.org`, version 2.9.6.

was placed, and its current pose. This way, an object that falls off from its place no longer contributes to the reward, even if it falls inside the target.

As it is not possible to distinguish the contribution of each object to the elevation map $\mathbf{O}$, used to compute the IoU, an approximation must be defined. Although the volume of the objects is variable, a quantity that is correlated with the intersection between current and target volumes (numerator in (1)) is the number of objects inside the target. In order to compute a metric equivalent to the IoU, it is also necessary to define an approximation to the union (denominator in (1)). This can be done by defining the episode length $T$ (the number of objects that would ideally be inside the target volume in the end of an episode) as the approximation to the target volume, and the current number of objects, given by the current time step $t$, as the approximation to the current volume. The union may be obtained using the property $|\mathcal{X} \cup \mathcal{Y}| = |\mathcal{X}| + |\mathcal{Y}| - |\mathcal{X} \cap \mathcal{Y}|$. The equivalent discounted metric may then be defined as

$$\mathrm{DIoU}_t = \frac{\sum_{i=0}^{t-1} b_t^{[i]} \cdot \mu_t^{[i]}}{T + t - \sum_{i=0}^{t-1} b_t^{[i]}}, \tag{2}$$

where $b_t^{[i]}$ is 1 if object $i$ is inside the target at time step $t$ and 0 otherwise, and $\mu_t^{[i]}$ is the applied discount. This metric also lays in the range $[0, 1]$, with 1 corresponding to the ideal case of a terminal state in which all objects are inside the target volume with no displacements from the original poses. The value of $b_t^{[i]}$ can be determined by retrieving the object position (center of mass) from the simulator and checking if it is inside the target volume. The discount can be defined as

$$\begin{aligned} \mu_t^{[i]} = &\max\left(0, 1 - \left(\frac{|\Delta \mathbf{x}_t^{[i]}|}{\Delta \mathrm{x}_{\max}}\right)^{c_{\mathbf{x}}}\right) \times \\ &\max\left(0, 1 - \left(\frac{|\Delta \theta_t^{[i]}|}{\Delta \theta_{\max}}\right)^{c_\theta}\right), \end{aligned} \tag{3}$$

where $\Delta \mathbf{x}_t^{[i]}$ and $\Delta \theta_t^{[i]}$ are the translation and rotation distances between the original pose of object $i$ and its pose observed at time step $t$. The parameters $\Delta \mathrm{x}_{\max}$ and $\Delta \theta_{\max}$ represent the maximum allowable distances. If the object exceeds one (or both) of these distances, its contribution to the reward is zero. The exponents $\mathrm{c_x}$ and $\mathrm{c}_\theta$ control how $\mu$ decreases with the distances. Higher values ($\mathrm{c_x}, \mathrm{c}_\theta > 1$) make it less sensitive to small displacements.

This metric may be used to distribute rewards at each step by immediately rewarding each contribution to the final value. This is accomplished using

$$R_t = \begin{cases} \mathrm{DIoU}_t - \mathrm{DIoU}_{t-1} & \text{if } 0 < t \leq T \\ 0 & \text{if } t = 0, \end{cases} \tag{4}$$

This way, an action that increases the metric (e.g. successfully places an object inside the target) is immediately rewarded, while an action that makes it decrease (e.g. causes part of the existing structure to collapse) is penalised. At the same time, the accumulated episode reward is the final value of the metric, which corresponds to an evaluation of the final result:

$$\sum_{t=1}^{T} R_t = \sum_{t=1}^{T} \mathrm{DIoU}_t - \sum_{t=0}^{T-1} \mathrm{DIoU}_t = \mathrm{DIoU}_T. \tag{5}$$

### 3.3. Model generation

In order to allow the agent to learn a generalization, the experience provided by the environment must be diverse. This includes the object models used. As it is hard to find a large number of models of real irregular objects (e.g. natural rocks), these models are synthetically generated. This allows the generation of a dataset of models from which the sequences used in each episode are sampled.

The model generation process is based on the method presented by Thangavelu *et al.* [8] to generate datasets of irregular convex polygons, used for 2D construction. It can be defined as generating blocks deformed according to an irregularity parameter $\varsigma$. As such, the starting point is a perfect brick: a box with extents $1 \times \frac{1}{2} \times \frac{1}{3}$, scaled to be inscribed in a sphere with radius $\mathrm{r}_{\max}$. The parameter $\mathrm{r}_{\max}$ is defined to bound the size of the objects (e.g. to create a set of objects that fit in a given gripper). A random displacement $\Delta \mathbf{x}$ is then applied to each of the eight vertices of the box, with each component $\Delta x_i$ sampled from a truncated normal distribution with zero mean, standard deviation $\sigma = \varsigma\, \mathrm{r}_{\max}$ and support $\Delta x_i \in [-\mathrm{r}_{\max}, \mathrm{r}_{\max}]$. A sequence of n subdivisions and random displacements of the additional vertices is then applied, were the parameter n controls the level of detail of the objects. A subdivision consists in dividing each face in four smaller faces, such that a new vertex is added in the middle of each edge and in the center of each face. This reduces the length of each edge by a factor of $\frac{1}{2}$. The distribution from which the displacements of the new vertices are sampled is scaled accordingly: after the $i^{th}$ subdivision, the truncated normal distribution has standard deviation $\sigma = \varsigma\, 2^{-i} \mathrm{r}_{\max}$ and support $\Delta x_i \in \left[-2^{-i}\mathrm{r}_{\max}, 2^{-i}\mathrm{r}_{\max}\right]$. With this process, the displacement of the original vertices greatly affects the overall shape of the object, while the following subdivisions and displacements successively increase the detail of the irregular shape. In the end of the "irregularization" process, the

model is converted to its convex hull. This is done for two reasons: first, the usage of convex shapes makes the collision computations more efficient in the physics engine; secondly, the convex hulls actually resemble more natural rocks (often subject to erosion) than the obtained irregular shapes. If any of the extents of the oriented bounding box (i.e. the box with minimum volume that bounds the model) exceeds the diameter $2\,\mathrm{r}_{\max}$ due to the random displacements, the model is scaled down to fit in a bounding box with maximum extent $2\,\mathrm{r}_{\max}$.

The meshes of the models are generated according to the described process using Trimesh[2]. Each mesh is positioned and oriented in the model's frame so that its oriented bounding box is centered at the origin and aligned with the axes, with the largest extent aligned with the first axis and the smallest with the third (vertical). A value for the density is uniformly sampled from an interval $[\rho_{\min}, \rho_{\max}]$, which simulates diverse materials or materials with variable composition or porosity. Some examples of models generated with different values of $\varsigma$ are presented in Figure 3.

(a) $\varsigma = 0$. (b) $\varsigma = 0.1$. (c) $\varsigma = 0.25$. (d) $\varsigma = 0.5$.

Figure 3: Models generated with different values of the irregularity parameter $\varsigma$ (3 subdivisions performed).

### 3.4. Agent

At each time step, the agent observes the $\mathrm{m}_{\mathbf{O}} \times \mathrm{n}_{\mathbf{O}}$ overhead elevation map $\mathbf{O}$, the $\mathrm{m}_{\mathbf{N}} \times \mathrm{n}_{\mathbf{N}}$ object elevation map $\mathbf{N}$ and the $\mathrm{m}_{\mathbf{O}} \times \mathrm{n}_{\mathbf{O}}$ goal elevation map $\mathbf{G}$ (same size as $\mathbf{O}$). The overhead and target views are stacked to form the $\mathrm{m}_{\mathbf{O}} \times \mathrm{n}_{\mathbf{O}} \times 2$ array $\mathbf{M}$ with elements given by

$$m_{ijk} = \begin{cases} o_{ij} & \text{if } k = 0 \\ g_{ij} & \text{if } k = 1, \end{cases} \tag{6}$$

which makes the spatial relation between the current structure and target volume clear to the agent.

As previously stated, an action $a = (i, j)$ can be visualized as overlapping the object elevation map on the overhead elevation map with the corresponding offsets. From this observation, a natural candidate to map the state to action values is a cross-correlation like operation that slides $\mathbf{N}$ through $\mathbf{M}$, outputting a value for each possible offset. Such an operation provides the intended codomain and captures the underlying spatial relations in the observed state. Figure 4 provides a visualization of the agent architecture, where the value estimator performs cross-correlation like operation.

The value estimator is translated into a neural network with an architecture based on a fully convolutional Siamese network [16]. The developed architecture is represented in Figure 5.

In this case, the inputs of the two branches are semantically different: one contains the overhead view of the current structure and target volume and the other is the view of the object to be placed. This differs from the usual utilization of Siamese networks, where both inputs have the same meaning and must be matched (e.g. if the goal was to find an instance of the new object in the current structure). Therefore, a modification of the Siamese architecture is used, where the branches do not share weights and may even have different architectures. The only restriction is that each branch outputs an array with the same height and width as its input, and that both branches output arrays with the same depth. This means that each branch must be a fully convolutional network providing a dense (pixelwise) feature extraction. Concretely, the U-net [17] architecture is used for this effect. As the object elevation map is smaller and contains less semantic information, a shallower network is used for its branch. The network sizes used are 5 levels in $\varphi_l$ and 3 in $\varphi_r$, and both branches use convolutional layers with 16 channels at the first level (see [18]).

The network architecture is extended with one additional fully convolutional module, placed at the output of the cross-correlation. This is motivated by the fact that the network must learn a specific value function, estimated from the collected rewards. In opposition to learning similarity, in which a metric must be as high as possible for a good match and low otherwise, this network should fit the specific values corresponding to the expected return. With the additional module, the output of the cross-correlation may be mapped to the intended values, giving the branches more freedom to express the features without the restriction of the cross-correlation fitting the value function. This module consists of two $3 \times 3$ convolutional layers with 16 channels and ReLU activation followed by a $1 \times 1$ single channel convolution, with no activation, that maps the 16 channels to the output value function.

Additionally, the network is adapted to match the dueling architecture. In this case, it is intuitive that the intrinsic value of a state is greatly influenced by the current structure: not only in terms of the availability of potentially good positions for the new object, but also in the stage of the episode (i.e. an advanced episode, indicated by a structure

[2] M. Dawson-Haggerty *et al.* Trimesh. `https://trimsh.org/`, version 3.8.1.

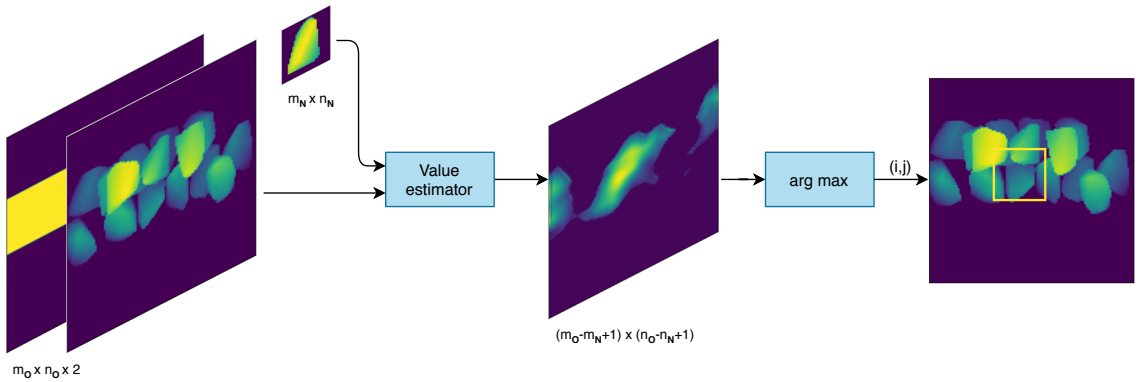



Figure 4: Process of choosing an action from the state. The image in the right is a visualization of the action, where $a = (i, j)$ gives the pixel indexes of the top left corner of the yellow box.

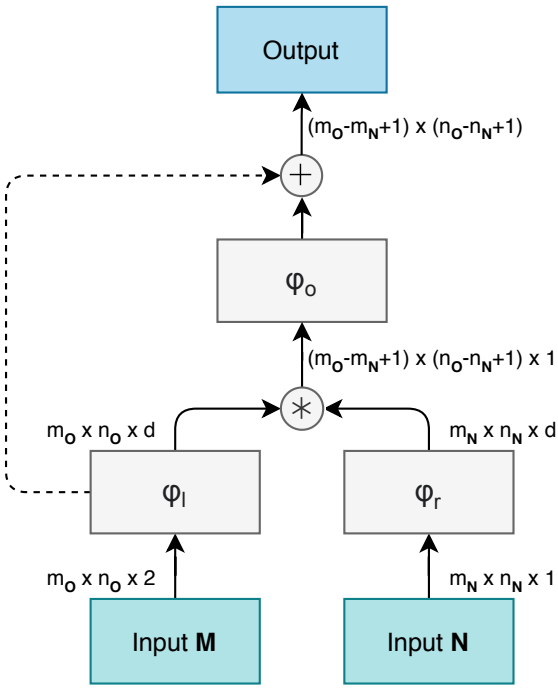



Figure 5: Network architecture. The modules $\varphi_l$ and $\varphi_r$ are the left and right branches of the network, that perform a dense feature extraction to the inputs **M** and **N**. The module $\varphi_o$ performs a fully convolutional transformation to the result of the cross-correlation, in order to estimate action advantages $A(s, a)$. A scalar is extracted by the module $\varphi_l$ to estimate the state value $V(s)$, that is then combined with the action advantage estimates.

with many objects, means lower expected return because the terminal state is closer). Although the shape of the object by itself may also be indicative of the expected quality of the available positions, this can be seen as part of the action advantage. This observation is translated into the architecture by extracting a scalar value from the branch of the overhead view, which is then combined with the output of the network as in [15]. This value is extracted by applying a global average pooling layer to the higher level, lower resolution feature maps in the branch (i.e. the end of the contracting path of the U-net), and then applying one fully connected layer with 256 units and ReLU activation and a single fully connected unit (no activation) to map the feature vector to a scalar.

### 3.5. Baselines

Three baseline levels of performance are considered for comparison with the learned policies. The first corresponds to the policy that randomly samples actions from the complete action space, corresponding to the observable region. The second is given by the policy that randomly samples actions inside the target volume, which corresponds to capturing the notion of the goal of the environment. For the third level, a position choice criterion is introduced in the form of an heuristic function. This level corresponds to capturing an understanding of the environment dynamics.

The considered heuristic is given by the cross-correlation between **O** and **N**, given by

$$\mathrm{C}_{ij}(\mathbf{O}, \mathbf{N}) = \sum_{k,l} o_{i+k,j+l} \cdot n_{kl}. \tag{7}$$

By definition, the values in the elevation maps are always greater or equal to zero. For this reason, each element of the cross-correlation output $\mathrm{C}_{ij}(\mathbf{O}, \mathbf{N})$ can be here interpreted as a weighted sum of the elements $o_{i+k,j+l}$, with the weights given by $n_{kl}$. Elements $o_{i+k,j+l}$ that do not coincide with the object (i.e. $n_{kl} = 0$) are not considered in the sum, while the remaining are weighted according to the object shape. Effectively, this results in a blurring operation, with the filter shape corresponding to the shape of the object. The local minima of the cross-correlation output are likely to correspond to concavities in the structure where the ob-

ject fits, because sharper local minima in the elevation map **O** would have been blurred out. Additionally, the global minimum is also the lowest point in the blurred surface, which is consistent with the height criterion used in related work. The action choice is performed in two steps. First, the cross-correlation is computed and the local minima are calculated. Then, the lowest of the local minima that are located in the target region is selected as the action to be performed.

## 4. Results

A set of experiments is performed on the described environment. The set of models used is constituted by 500 models generated with each value of $\varsigma \in \{0.5, 0.55, \ldots, 1\}$, resulting in a complete set of 5500 models. The number of subdivisions performed in the generation process is 3. The size of the overhead and object elevation maps are $128 \times 128$ and $32 \times 32$, respectively. The episode length is set as 30. Two metrics are used to evaluate the obtained policies. The first is the IoU defined in (1), which is a measurement of the similarity between the built and target volumes. The second is the average of the discounts defined in (3), given by

$$\mathrm{AD}_t = \frac{1}{t} \sum_{i=0}^{t-1} \mu_t^{[i]}, \tag{8}$$

which is a metric of the similarity between the planed structure (given by the poses were the objects were placed) and the actual structure (given by the current poses of the objects). The later evaluates the effectiveness and stability of the placements.

The training sessions are performed using the DQN algorithm with a replay memory of size 400000, minibatch size of 32, and 2 transitions collected between network updates. The network optimization uses Adam [19] with learning rate $6.25 \cdot 10^{-5}$ and exponential decay rate for the first and second moment estimates 0.95.

Two training sessions are run with rewards generated with the IoU (1) and four with rewards generated by the DIoU (2). At intervals of 10000 iterations, the current greedy policy is evaluated with 100 episodes. Figures 6 and 7 show the evolution of the evaluation results during training, compared with the results of the three baseline levels. The curves highlighted with the stronger colour are considered the best run for each reward shape, and correspond to the same network initialization.

From the learning curves in Figures 6 and 7, it is observable that the notion of the goal is easily (and quickly) learned. This corresponds to reaching the second level of baseline performance, given by the policy that samples actions from the target region. Afterwards, the dynamics of the environment should be gradually captured by the value functions. It is verified in Figure 7 that the rewards generated by the DIoU result in returns that are difficult to predict, which explains the frequent breakdowns of the training sessions. These breakdowns correspond to one of the layers in the network stopping to produce activations for any input, which results in the prediction of equal values for all actions and the consequent performance breakdown. The layer that doesn't produce activations blocks back-propagation, which stops the learning process. This outcome could be prevented with the usage of different hyperparameters (e.g. smaller learning rate) or network architecture modifications (e.g. residual connections). Despite this, for one of the training sessions a placing criterion is learned and the performance surpasses the third baseline level, given by the cross-correlation heuristic. On the other hand, the policies learned with the IoU are more consistent, but do not significantly outperform the baseline.

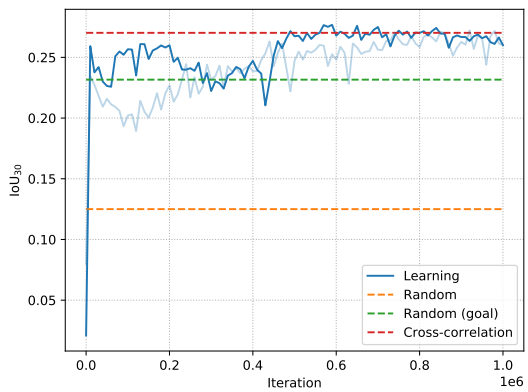



Figure 6: Evolution of the evaluation results during training with rewards generated by the IoU.

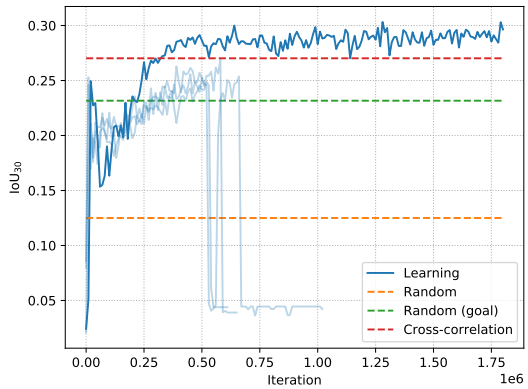



Figure 7: Evolution of the evaluation results during training with rewards generated by the DIoU.

To further analyse the policies obtained from the best run with each metric, a set of 100 episodes is used to evaluate the evolution of the evaluation metrics, IoU and AD, during an episode. The results are shown in Figures 8 and 9, compared to the average evolution obtained with the baselines for the same episodes. Table 1 reports the average and standard deviation of the final values of the metrics obtained with each policy (learned and baselines).

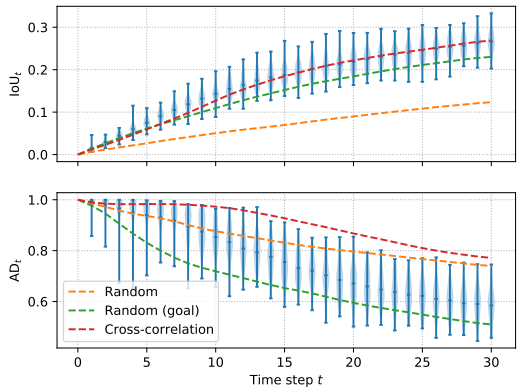



Figure 8: Evolution of the intersection over union and average discount during an episode, for the policy learned with rewards generated by the IoU. Values reported for the baselines are the average over 100 episodes.

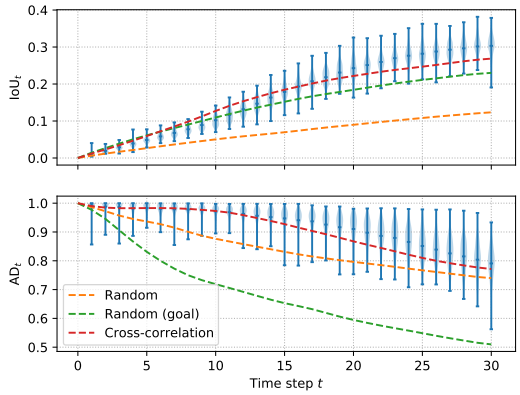


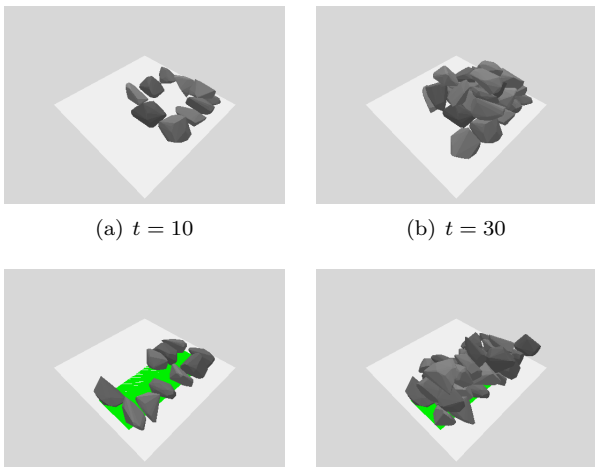

Figure 9: Evolution of the intersection over union and average discount during an episode, for the policy learned with rewards generated by the DIoU. Values reported for the baselines are the average over 100 episodes.

Regarding the policy learned with the IoU, it is possible to observe that the fact that no discounts are applied results in a policy that produces generally unstable placements. The AD curve in Figure 8 shows average values of $AD_t$ closer to the random policy. This ends up undermining the final performance.

Table 1: Evaluation results. Values reported as the average and standard deviation over 100 episodes.

| | $IoU_{30}$ | $AD_{30}$ |
|---|---|---|
| Learned with IoU | 0.266 ± 0.026 | 0.584 ± 0.064 |
| Learned with DIoU | 0.303 ± 0.029 | 0.791 ± 0.058 |
| Random | 0.125 ± 0.027 | 0.738 ± 0.06 |
| Random (goal) | 0.232 ± 0.025 | 0.507 ± 0.064 |
| Cross-correlation | 0.27 ± 0.024 | 0.769 ± 0.046 |

From Figure 9, the evolution of the IoU and AD throughout the episodes allows an interpretation of the behaviour improvement over the baselines. The IoU starts by increasing slower than for the cross-correlation heuristic. This is explained by the fact that this policy starts by packing a tight first course, which also results in keeping the value of AD closer to one for longer. This initial behaviour, along with a suitable understanding of the dynamics of the environment, end up enabling the assembly of structures more similar to the target and with less displacements from the original poses. Figure 10 shows two stages of a structure assembled by the obtained policy, and Figure 11 presents an example of an obtained value map.

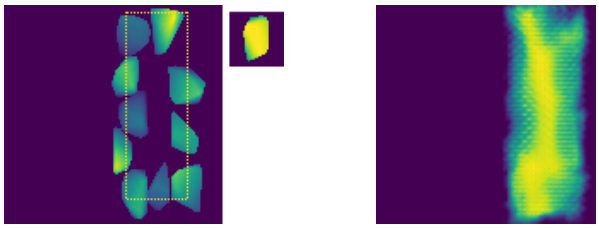


(a) $t = 10$ (b) $t = 30$

(c) $t = 10$ (d) $t = 30$

Figure 10: Example of a structure obtained with the learned policy.

An interesting observation is the behaviour learned for the first steps. The agent learns to start an episode by enclosing the target area with the first objects. This allows the following objects to be supported by an inwards sloping surface, which prevents them from falling out and increases the overall stability. This behaviour, learned from scratch, is



(a) State. (b) Value map.

Figure 11: Example of a value map obtained with the learned policy for the given state. The target is represented in the state with the dotted yellow line.

consistent with the theory of dry stacking (although typically the largest objects would be selected for this purpose, which is not allowed here).

## 5. Conclusions

The major achievement of this work was to develop a setup in which a dry stacking policy can be learned without any previously acquired models of the environment. An agent is able to learn from scratch a policy that captures the goal of the environment and its dynamics. The emerged behaviour is, to some extent, consistent with existent dry stacking theory.

As discussed, the results could be improved upon (or made more consistent) with a better choice of hyperparameters or with some improvements to the network architecture.

The policies learned with the proposed approach could be applied to a realistic setting as the high level planner in an autonomous construction pipeline, where a lower level controller would perform the careful placements.


## Acknowledgements

I would like to thank my supervisors, Alexandre Bernardino and Rodrigo Ventura, and Pedro Vicente for all the support provided during the development of this work. I would also like to acknowledge VisLab for providing the hardware used for the the presented experiments.


## References


[1] K. R. Sacksteder and G. B. Sanders. In-Situ Resource Utilization for lunar and Mars exploration. In *Collection of Technical Papers - 45th AIAA Aerospace Sciences Meeting*, volume 6, 2007.

[2] M. Thangavelu. Living on the Moon. In *Encyclopedia of Aerospace Engineering*. John Wiley & Sons, Ltd, Chichester, UK, 2012.

[3] C. S. Allen, R. Burnett, J. Charles, F. Cucinotta, R. Fullerton, R. Goodman, A. D. Griffith, J. J. Kosmo, M. Perchonok, J. Railsback, S. Rajulu, D. Stilwell, G. Thomas, T. Tri, R. Wheeler, A. Mueller, and A. Simmons. Guidelines and Capabilities for Designing Human Missions. *Nasa/Tm–2003–210785*, (January), 2003.

[4] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[5] M. Lambert and P. Kennedy. Using artificial intelligence to build with unprocessed rock. In *Key Engineering Materials*, volume 517, 2012.

[6] S. A. Nielsen and A. Dancu. Fusing design and construction as speculative articulations for the built environment. In Ajla Aksamija, John Haymaker, and Abbas Aminmansour, editors, *Future of Architectural Research: Proceedings of the Architectural Research Centers Consortium Conference*, Chicago, 2015. Perkins+Will.

[7] F. Furrer, M. Wermelinger, H. Yoshida, F. Gramazio, M. Kohler, R. Siegwart, and M. Hutter. Autonomous robotic stone stacking with online next best object target pose planning. In *Proceedings - IEEE International Conference on Robotics and Automation*. Institute of Electrical and Electronics Engineers Inc., 2017.

[8] V. Thangavelu, Y. Liu, M. Saboia, and N. Napp. Dry Stacking for Automated Construction with Irregular Objects. In *Proceedings - IEEE International Conference on Robotics and Automation*. Institute of Electrical and Electronics Engineers Inc., 2018.

[9] Y. Liu, S. M. Shamsi, L. Fang, C. Chen, and N. Napp. Deep Q-Learning for Dry Stacking Irregular Objects. In *IEEE International Conference on Intelligent Robots and Systems*. Institute of Electrical and Electronics Engineers Inc., 2018.

[10] Y. Liu, M. Saboia, V. Thangavelu, and N. Napp. Approximate stability analysis for drystacked structures. In *Proceedings - IEEE International Conference on Robotics and Automation*, volume 2019-May. Institute of Electrical and Electronics Engineers Inc., 2019.

[11] Y. Liu, J. Choi, and N. Napp. Planning for Robotic Dry Stacking with Irregular Stones. *12th Conference on Field and Service Robotics (FSR19)*, 2019.

[12] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, Cambridge, MA, second edition, 2018.

[13] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. Freitas. Dueling Network Architectures for Deep Reinforcement Learning. In M. F. Balcan and K. Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, New York, New York, USA, 2016. PMLR.

[14] R. S. Sutton, J. Modayil, M. Delp, T. Degris, P. M. Pilarski, A. White, and D. Precup. Horde: A Scalable Real-Time Architecture for Learning Knowledge from Unsupervised Sensorimotor Interaction. In *The 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*, AAMAS '11, Richland, SC, 2011. International Foundation for Autonomous Agents and Multiagent Systems.

[15] T. Schaul, D. Horgan, K. Gregor, and D. Silver. Universal Value Function Approximators. In F. Bach and D. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, Lille, France, 2015. PMLR.

[16] L. Bertinetto, J. Valmadre, J. F. Henriques, A. Vedaldi, and P. H. Torr. Fully-convolutional siamese networks for object tracking. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 9914 LNCS. Springer Verlag, 2016.

[17] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 9351. Springer Verlag, 2015.

[18] O. Ronneberger, P. Fischer, and T. Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. Technical report.

[19] D. P. Kingma and J. L. Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*. International Conference on Learning Representations, ICLR, 2015.