2021 The 21st International Conference on Control, Automation and Systems (ICCAS 2021)
Ramada Plaza Hotel, Jeju, Korea, Oct. 12∼15, 2021

Model-Based Reinforcement Learning with LSTM Networks for Non-Prehensile Manipulation Planning

Jeffrey Fong[1*], Domenico Campolo[2], Cihan Acar[3], Keng Peng Tee[3]

[1]School of Computing, National University of Singapore,
117417, Singapore (jfong@comp.nus.edu.sg)
[2]School of Mechanical and Aerospace Engineering, Nanyang Technological University,
639798, Singapore (d.campolo@ntu.edu.sg)
[3]Institute for Infocomm Research, Agency for Science, Technology and Research (A*STAR) ,
138632, Singapore ({acar_cihan, kptee}@i2r.a-star.edu.sg) * Corresponding author

Abstract: Solving non-prehensile manipulation tasks requires domain knowledge involving various interactions such as switching contact dynamics between the robot and the object, and the object-environment interactions. This results in a switched nonlinear dynamic system governing the physical interactions between the object and the environment. In this paper, we propose an interactive learning framework that allows a robot to autonomously learn and model an unknown object's dynamics, as well as utilise the learned model for efficient planning in completing re-positioning tasks using non-prehensile manipulation. First, we model the overall object dynamics using a Long Short-Term Memory (LSTM) neural network. We then assimilate the learned model into the Monte Carlo Tree Search (MCTS) algorithm with a dense reward function to generate an optimal sequence of push actions for task completion. We demonstrate the framework in both simulated and real robot that pushes objects on a table.

Keywords: Manipulation planning, non-prehensile, reinforcement learning

1. INTRODUCTION

Non-prehensile manipulation is the manipulation of objects without grasping, and encompasses a wide range of actions including pushing, tilting, flipping, and even throwing. By leveraging on the environment (e.g. for weight support, or gravity-assisted motion), non-prehensile manipulation can be more efficient and less restrictive than conventional prehensile manipulation. In particular, object manipulation by pushing allows large, heavy objects to be moved expeditiously, and can be applied in many industrial tasks, such as material transfer, sorting, cleaning, and rearrangement.

Non-prehensile manipulation, particularly for pushing an object, is challenging due to the presence of switching/sliding contact dynamics and strong friction effects, which are difficult to model and control. Moreover, uncertainty in the center of mass location of the object can introduce rotational instability when pushed at the wrong locations. Our paper focuses on push planning to reach a target object pose, which amounts to finding an optimal finite sequence of actions on the object, in terms of the contact locations and the push velocities. This can be formulated as a Markov Decision Process (MDP) and solved through reinforcement learning (RL) methods. Most research works on non-prehensile manipulation planning focus on model-free RL methods, including deep Q-Network [1], heuristic value function with receding horizon planner [2], and natural policy gradient method [3]. Although these methods have been shown to produce positive results, they often exhibit low sample efficiency as a model-free agent needs to learn from scratch [4]. Due to high sampling cost involving contact-rich interactions between robot and environment, we focus on

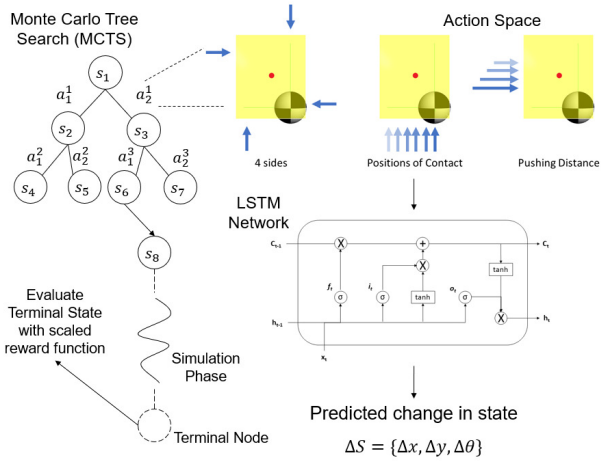


Fig. 1.: Graphical representation of the system diagram. Push planning is done by an MCTS algorithm. Each action is illustrated with the action space which serves as the input to the LSTM network that outputs the predicted change in state Δs. The gray and black circles represent the center of mass of the object while the red dots represent the spatial center point of the object.

model-based RL, which has higher sample efficiency and thus more practical compared to model-free RL.

Model-based RL requires learning a model of the object dynamics, which allows prediction of the state and reward resulting from an action. Some works focused on physics-based parameter estimation techniques for estimating center of mass positions [5] and friction parameters [6] in pushing operations. Other works estimate

parameters using model-fitting techniques such as least squares [7], assuming that the model of the dynamics is linear-in-the-parameters (LIP) and time-invariant. However, this does not hold well for the pushing problem where nonlinear-in-the-parameters friction and discontinuous contact switches are present. More recent works utilised machine learning and artificial intelligence techniques to model the object's dynamics [8]-[11]. Using the model of the object dynamics, simulated look ahead can be performed to plan the optimal action. Existing methods include model-based RL with model predictive control [10]-[12], sampling-based motion planning with generative model [13], and extended Monte Carlo tree search in unobservable domains for re-arrangement tasks [14].

In this paper, we propose a novel approach using long short-term memory (LSTM) to learn the complex object dynamics during object pushing, and integrating this learnt LSTM model with Monte Carlo Tree Search (MCTS)-based reinforcement learning to plan near-optimal sequence of push actions to move the object close to the target pose. LSTMs are capable of modeling time-varying non-linear non-homogeneous ordinary differential equations governing the object dynamics during pushing, and are free from LIP and time-invariance assumptions. To solve the push planning problem, we model it as a finite-size finite-horizon MDP, and use MCTS to plan the optimal set of push actions based on iterative simulations with the LSTM model predictions. Obstacle avoidance is achieved through penalizing actions that lead to collisions during MCTS simulations. Inspired by the shaped reward function design in [15], we propose a dense nonlinear reward function providing dense reward signals with enhanced sensitivity near the goal. Thanks to the dense rewards providing sufficient guiding signals for the planner, a shallow simulation rollout depth can be fixed to reduce computational burden. We performed extensive simulations to evaluate the effectiveness of the LSTM model under different center of mass and friction conditions. A real robot experiment was also conducted to demonstrate feasibility in real world conditions.

2. NON-PREHENSILE RE-POSITIONING TASK

The environment consists of an object of interest that the robot will re-position and a set of obstacles $\mathcal{O}$ that must not have contact with the object. In this paper, we consider objects that are *regular convex polygons with non-uniform mass density*. The state space of the environment is denoted by $\mathcal{S}$= $\{x, y, \theta\}$, where $(x, y) \in \mathbb{R}^2$ represents the object position on a plane (e.g. floor or tabletop), and $\theta \in [-\pi, \pi]$ the object orientation, or angle of rotation, about the axis perpendicular to the plane. We are concerned with completing the task $\mathcal{T}$, which involves pushing the object from the start state s_0 to a goal region

$$\Omega_g := \{s \in \mathcal{S} \mid |x-x_g| \leq \delta_x, |y-y_g| \leq \delta_y, |\theta-\theta_g| \leq \delta_\theta\} \quad (1)$$

surrounding a goal state s_g, with $\delta_x, \delta_y, \delta_\theta$ being threshold numbers, while avoiding any contact with obstacles $\mathcal{O}$.

During task planning, we generate a possible action sequence $\zeta = \{a_1, a_2, \cdots, a_n\}$ from a start state $s_0 \in \mathcal{S}$ to a terminal state $s_T \in \mathcal{S}$. The objective is to produce an optimal action sequence ζ^* among all possible ζ that creates a path to the goal region $\Omega_g \subset \mathcal{S}$ with the least actions needed. This objective can be modeled as a model-based reinforcement learning problem applied to a finite-size and finite-horizon Markov Decision Process (MDP). A finite size means that the action space is discrete while finite horizon refers to the finite length of the sequence of transitions to the terminal state, i.e. $s_k, a_k, s_{k+1}, a_{k+1}, s_{k+2}, \cdots, s_T$. All the episodes from s_k to s_T can be considered as getting to the same terminal state but with different experiences and rewards according to the reward function $R(s, a)$. Theoretically, the learned model is able to act as a stochastic transition model $P(s'|s, a) \rightarrow [0, 1]$ which attempts to map actions a to next states s' as accurately as possible. Throughout this process, the Markov property is strictly adhered to in which the planner decides on each action based only on the current state (represented as the root node in the tree) and not on the sequence of history of events which preceded it.

3. MODELING OBJECT-PUSHING DYNAMICS USING AN LSTM NETWORK

3.1 Dynamics of Object Pushing

In non-prehensile manipulation, the motion of the object is governed by the physical interactions between the object and the environment as well as the contact dynamics between the object and the robot's end-effector. This involves simultaneous modeling of various kinematic and inertial parameters, as well as identifying the relationships between each of these parameters. The underlying dynamics for single-continuous-contact object pushing are nonlinear and hybrid [16]:

$$\begin{aligned} \dot{p} &= \begin{cases} b_1(p, u) & \text{if} \quad u \in \mathcal{MC} \\ b_2(p, u) & \text{if} \quad u > \mathcal{MC} \\ b_3(p, u) & \text{if} \quad u < \mathcal{MC} \end{cases} \\ s &= h(p) \end{aligned} \quad (2)$$

where $s(t) = [x(t), y(t), \theta(t)]^T$ is the object pose at time t, $p(t)$ the internal state of the dynamics, $b_i(\cdot)$ nonlinear functions modeling the effects of friction and inertia for different conditions on the motion cone $\mathcal{MC}$ at the point of contact, $h(\cdot)$ the mapping from the internal state to the output object pose, and $u(t)$ the pusher velocity.

For (2), it is highly difficult to model the functions and conditions analytically, perform system identification, or find the solution to the set of differential equa-

tions, even for single-continuous-contact pushing. For single-discontinuous-contact pushing considered in this paper, more switchings in the dynamics are involved, so the model becomes even more complex. With non-uniform mass density, the mass center does not necessarily coincide with the geometric center of the polygon, so it is difficult to predict, from object geometry, how the object will move when pushed at different points. To overcome these difficulties, an approach is to use Recurrent Neural Networks (RNN), such as Long-Short Term Memory (LSTM) networks, to approximate such hybrid nonlinear dynamics and predict the steady-state object pose given a force or velocity profile of a robot's end-effector.

3.2 LSTM Neural Network Dynamics Model

LSTM networks are widely used to model long time series and it resolves the vanishing gradient problem inherent in classical RNNs [17]. Compared to a single activation function in an RNN unit, each LSTM cell contains a special internal gating mechanism, consisting of the forget, input and output gates, which regulates the flow of information by retaining or forgetting certain information selectively.

We parameterize the learned dynamics model as an LSTM neural network, where the parameter vector λ represents the weights of the network. The parameters of the dynamics model learn to predict the change in the object pose as a result of the push action:

$$\Delta s_k := s_{k+1} - s_k = \begin{bmatrix} x(t_{k+1}) - x(t_k) \\ y(t_{k+1}) - y(t_k) \\ \theta(t_{k+1}) - \theta(t_k) \end{bmatrix} \tag{3}$$

where t_k and t_{k+1} are the initial times of the kth and t_{k+1} actions, respectively, with t_{k+1} indicative of the end of the kth action. Therefore, we can predict the new object state $\hat{s}_{k+1}$, from the current one s_k, as a result of the action a_k:

$$\hat{s}_{k+1} = s_k + \hat{f}_\lambda(s_k, a_k) \tag{4}$$

where $\hat{f}_\lambda$ is the function approximator which makes the predictions using model parameter λ and input a_k. The specific architecture that we use in this study is described in Section 5.1

To collect data for training the LSTM network, the robot performed N random push interactions with the object of interest on a tabletop. Each push action is sampled uniformly from an action space $\mathcal{A}$ which comprises all possible combinations of object edges, contact points, and push magnitudes (see Section 4.1). The input data for the LSTM consists of the measured object poses s_j before each push, as well as a *minimum-jerk* [18] time-series for the push speed. The output data consists of the measured difference in object poses $\Delta s_j = s_{j+1} - s_j$ before and after each push.

Based on the Mean Squared Error (MSE) loss function:

$$\mathcal{L} = \frac{1}{N} \sum_{j=0}^{N-1} \| \Delta s_j - \hat{f}_\lambda(s_j, a_j) \|^2 \tag{5}$$

we train the LSTM using stochastic gradient descent with adaptive learning rate and initialization bias correction [19].

4. OPTIMAL PUSH PLANNING

Monte Carlo methods are well suited for model-based reinforcement learning as they are used for planning in finite-horizon sequential decision-making problems. Such methods rely on repeated random sampling and simulations to obtain numerical results and continuously generate new successor states given an action and a current state: $s_{k+1} = F(s_k, a_k)$.

4.1 Action Space

We decompose the full action space $\mathcal{A}$ as a Cartesian product of 3 sets of action components $E \times C \times D$:

$$E = \left\{ 0, \frac{2\pi}{N_E}, \frac{4\pi}{N_E}, ..., \frac{2(N_E-1)\pi}{N_E} \right\} \tag{6}$$

$$C = \left\{ \frac{1}{N_C}, \frac{2}{N_C}, ..., \frac{N_C - 1}{N_C} \right\} \tag{7}$$

$$D = \{D_i\}_{i=1,2,\cdots,N_D} \tag{8}$$

where E is a list of edges for an N_E-sided polygonal object expressed in terms of the angular displacement from a reference edge, C a list of $N_C - 1$ contact locations along an edge expressed as a fraction of the edge length, and D a list of N_D push distances at a contact location. For simplicity, we consider that the push is applied only in the direction normal to the edge.

The edge $E_i, i \in [1, N_E]$ determines the pushing direction, which we consider to be along the edge normal. The push distance $D_i, i \in [1, N_D]$ affects the extent of the motions generated by the specific pushing direction. For each push distance D_i, given push duration τ, we generate a minimum-jerk time-series for the push speed.

In other words, selecting a particular action instance a_k from action space $\mathcal{A}$ involves selecting an edge on the object from E, a contact location along the selected edge from C, and a push distance at the selected contact location from D. Then, given push duration τ, we generate the push speed trajectory $v(t)$ for action a_k, starting at the selected contact location and acting in the direction of the normal to the selected edge. This speed trajectory, along with the current object pose, are fed into LSTM model to yield a prediction of the final object pose following the push. The upper right part of the system diagram in Figure 1 provides an illustration of the action space for a rectangular object.

4.2 Monte Carlo Tree Search

The Monte Carlo Tree Search (MCTS) algorithm [20] is one of the Monte Carlo methods. It is an *anytime* algorithm and it functions on the basis of iterative episodic simulations starting from the current initial state to a terminal state. The execution of the algorithm progressively builds an asymmetric tree whose structure is influenced by the selection process following the *Upper Con-*

Algorithm 1 MCTS for Optimal Push Planning

1: $s_0, S_g \leftarrow$ `ObserveStates()`
2: **while not** `CheckGoalReached`(s_0, S_g) **do**
3: $\quad\zeta, \hat{S} \leftarrow$`MonteCarlo`$(s_0, S_g)$
4: $\quad$**for** $i = 1$ to N_ζ **do**
5: $\quad\quad$`PerformPushAction`(ζ_i)
6: $\quad\quad s_i \leftarrow$ `ObserveStates()`
7: $\quad\quad$**if** `CheckThreshold`$(s_i, \hat{s}_i)$ **then**
8: $\quad\quad\quad$**break**

fidence Bound for Trees (UCT) equation [21] given by $UCT_i = \bar{R}_i + C\sqrt{2\ln n/n_i}$, where $\bar{R}_i$ is the average reward received over the number of visits to node n_i, n the total number of expansions, and C a constant controlling the exploration-exploitation tradeoff.

A good strategy to complete the task can be inferred as the tree grows until a node whose state falls within the goal region. We evaluate the value estimations of the action sequences through tracking the nodes with the highest number of visits. In this case, we consider the sequence of actions which lead to the shortest path from the root node n_0 to the goal node n_g in the tree to be ζ^*.

4.3 MCTS for Push Planning

The iterative simulation feature of the MCTS algorithm presents a synergistic effect when combined with the LSTM neural network dynamics model developed in Section 3.The learned model is able to act as a simulator which performs the black-box predictions needed during the simulation phase of the MCTS algorithm, providing an automatic process in expanding the tree and producing the optimal sequence of actions.

Algorithm 1 illustrates the MCTS algorithm modified for optimal push planning. Each node in the tree stores the following information:

- n_i - the number of times the node has been visited
- s_i - the state in state space S that describes the node
- a_i - the preceding action that created the node
- $\bar{R}_i$ - average reward of the node

With an initial state s_0 and a goal state S_g, the MCTS algorithm is executed to generate a feasible action sequence ζ to perform, as well as the predicted states $\hat{S} = [\hat{s}_1, \hat{s}_2, \cdots, \hat{s}_{N_\zeta}]$, where N_ζ denotes the cardinality of ζ (Line 3). The number of actions in the sequence will differ depending on how far apart s_0 and S_g are. As the robot performs each action a_i in the sequence physically (Line 5), it observes the next state s_{i+1} (Line 6) and checks if the error between the network prediction $\hat{s}_{i+1}$ and ground truth s_{i+1} exceeds a threshold ϵ (Line 7). If ϵ is exceeded, the robot will abort the execution of the current ζ and plans for another action sequence ζ_{new} again with s_{i+1} as the new s_0. Otherwise, the robot will proceed to perform a_{i+1} and the ϵ check repeats again. The procedure stops when the robot successfully pushes the object to the goal, i.e. $s_T \in \Omega_g$.

4.4 Reward Function

One of the key to implementing efficient MCTS algorithms is to formulate a clear and accurate reward function. Reward functions provide reward signals which encourage and guide the MCTS planner to evaluate terminal states succinctly while maximising goal-related information gain, thereby expanding the tree as directly as possible. We design a dense reward function with quadratically increasing reward as the terminal state approaches the goal state. To allow different scaling in position and orientation, we consider 2 reward component functions $R_X : \mathbb{R}^2 \to \mathbb{R}$ and $R_\theta : \mathbb{R} \to \mathbb{R}$.

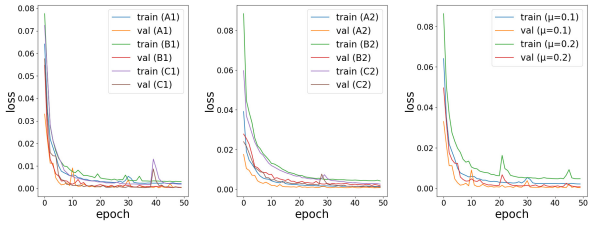


Fig. 2.: Mean squared error loss vs number of epochs for different COM positions in Object_1 (left) and Object_2 (middle), as well as different friction coefficients for Object_1 (right).

Let $\eta = X \oplus \theta$. Hence, $d_X := \sqrt{(x - x_g)^2 + (y - y_g)^2}$ and $d_X^* := \epsilon_X + \Delta_X$, where (x_g, y_g) is the goal position and ϵ_X, Δ_X positive constants representing the size of the goal region and the spread of the reward function, respectively. Contrarily, $d_\theta := |\theta - \theta_g|$ and $d_\theta^* := \epsilon_\theta + \Delta_\theta$, with θ_g the goal yaw angle and $\epsilon_\theta, \Delta_\theta$ positive constants. Then, the component reward function is defined as:

$$R_\eta(\eta) = \begin{cases} 1 & d_\eta \leq \epsilon_\eta \\ -1 & d_\eta > d_\eta^* \\ 2\left(\frac{d_\eta - d_\eta^*}{\Delta_\eta}\right)^2 - 1 & otherwise \end{cases} \tag{9}$$

which provides dense reward signals for both the position and orientation components, reaching a maximum value of 1 within a ϵ-neighborhood of the goal, and quadratically decreasing to -1 with increasing distance from the goal.

We combine the position and orientation components by a weighted sum to obtain the dense reward function $R_{dense} : \mathbb{R}^2 \times \mathbb{R} \to \mathbb{R}$. Specifically, $R_{dense}(x, y, \theta) = \rho R_X(x, y) + (1 - \rho) R_\theta(\theta)$ where $\rho \in [0, 1]$ is a constant which can be adjusted to weigh the relative importance of position versus orientation error based on the task requirements.

In order to find an action sequence that not only pushes the object into the goal region, but also minimizes the number of actions, we add a cost term that penalizes long action sequences, yielding the overall reward function:

$$R_{overall} = \varrho R_{dense} + (1 - \varrho)\left(\frac{2}{N_\zeta} - 1\right) \tag{10}$$

where $\varrho \in [0, 1]$ is a constant.

4.5 Obstacle Avoidance

In this study, we are dealing with static obstacles. The obstacle avoidance feature is achieved by applying the

Table 1.: Statistical measure of the network prediction performance for all 6 COM positions and 2 Friction settings. All objects are trained with $\mu = 0.2$ unless stated otherwise. Each network was trained 20 times and the statistical measures values averaged.

Performance Index	**COM**$_{A1}$ $\mu = 0.1$	**COM**$_{A1}$ $\mu = 0.2$	**COM**$_{B1}$	**COM**$_{C1}$	**COM**$_{A2}$	**COM**$_{B2}$	**COM**$_{C2}$
Average MSE for test set	3.91×10^{-3}	3.22×10^{-3}	2.81×10^{-3}	1.32×10^{-3}	3.23×10^{-3}	2.65×10^{-3}	2.85×10^{-3}
Average MSE for training set	2.18×10^{-3}	1.16×10^{-3}	7.28×10^{-4}	6.45×10^{-4}	1.24×10^{-3}	9.95×10^{-4}	1.04×10^{-4}
Average MPE for test set (displacement) [m]	5.95×10^{-3}	4.56×10^{-3}	2.35×10^{-3}	2.51×10^{-3}	4.45×10^{-3}	3.83×10^{-3}	3.63×10^{-3}
Average MPE for test set (orientation) [rad]	0.07634	0.039447	0.03035	0.0357	0.04283	0.04014	0.03871

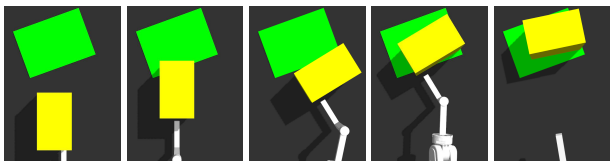
Gilbert-Johnson-Keerthi (GJK) distance algorithm [22]. We also add a heuristic that penalises the planner if a collision occurs during the simulation phase. If collision occurs during the expansion phase, the corresponding child node in the tree will be removed and no longer considered. On the other hand, if collision occurs during the simulation phase, the current rollout will be terminated and a reward of -1 will be given instead to penalise the planner for causing a collision. A large negative reward was not preferred in this case as it was deemed too harsh since R_{max} is only kept at unit reward and a harsh penalty will cause confusion for the UCT algorithm, limiting exploration.

5. EXPERIMENTS AND RESULTS

We evaluated the prediction performance of the LSTM network and the efficiency of the MCTS implementation on both simulated and real 7-DOF arm of the DRC-Hubo robot. The task was to push a tabletop object to a desired goal region (1) with $\epsilon_x = 0.04$, $\epsilon_y = 0.04$, and $\epsilon_\theta = \pi/18$. The obstacle avoidance feature described in Section 4.5 was also tested for both the simulation and real robot experiment.

For simplicity, we considered a rectangular object, although the proposed method is valid for any regular convex polygon. We attached an L-bar to robot hand for better contact with the object surface as well as for ease of trajectory planning. The L-bar was defined as a revolute joint in simulation to increase the flexibility of the robotic arm. In the real robot experiment, the L-bar was firmly grasped in the robot's hand with known frame transformations.

The robotic arm was position-controlled in joint space using desired joint trajectories generated by MoveIt!. Each push action was based on the minimum-jerk push velocity trajectories described in Section 4.1. Perception of the environment in the real robot experiment was done by using ArUco markers and OpenCV detection algorithms. Different markers are placed on the object, obstacle and the goal to allow the robot to identify the types and estimate the poses.



(a) Initial (b) Action 1 (c) Action 2 (d) Action 3 (e) Action 4

Fig. 3.: Representative action sequence planned and produced by MCTS to push object to the goal area in simulated robot study.

5.1 Evaluating LSTM networks for modeling object dynamics

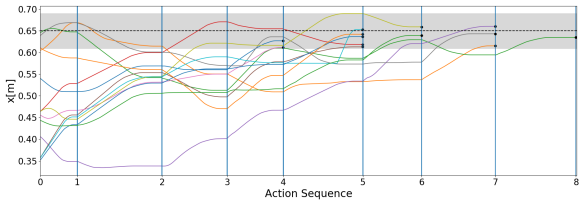
For training of the LSTM network, we collected data of 5000 instances of tabletop object pushing by the robot (uniformly distributed over action set $\mathcal{A}$) in Gazebo simulation environment, on 2 objects: Object$_1$ with dimensions 12×10×20 cm, and Object$_2$ 20×12×20 cm. Our network consists of two stacked LSTM hidden layers of sizes 50 and 25 respectively and each layer is topped with a dropout layer with rate 0.2. The Adam optimizer [19] was used for optimization while the Mean Squared Error (MSE) was used as our loss function. To test for generalisability of the network, a test set with unseen position of contact and distance pushed values was used.

We conducted hyperparameter optimization using a Grid Search algorithm [23] for the learning rate and batch size to fine-tune the LSTM network. All the networks in the grid search algorithm were each trained with 50 epochs. The optimal learning rate and batch size were found to be 0.005 and 64 (79 iterations per epoch) respectively, and used in the remaining of this study.

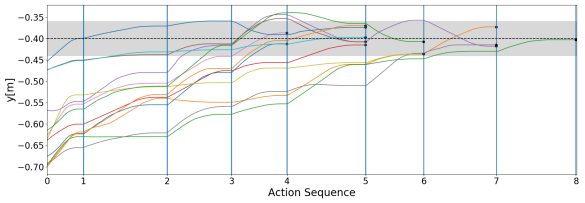
To further verify the versatility of the LSTM network in modeling different dynamics, we conducted a series of tests with variants of the objects mentioned in this section previously. The variants included varying Center of Mass (COM) positions, as well as the friction coefficients μ. In both tests, all the datasets are of size 5000.

5.1.1 Varying Center of Mass

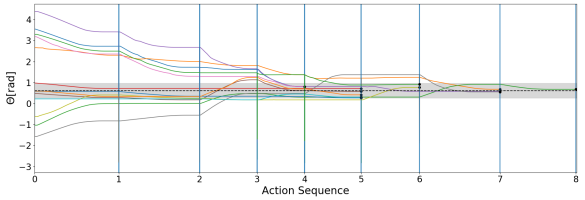
In our experiment, we considered three different COM positions: COM$_A$ $= (0.05, -0.03)$, COM$_B$ $= (0, 0)$, and COM$_C$ $= (-0.05, 0.03)$. All other object properties as well as the environment were kept the same. Object-specific LSTM networks with the optimal archi-



(a) Trajectory in x



(b) Trajectory in y



(c) Trajectory in θ

Fig. 4.: Object trajectories in x, y and θ sampled from 13 different initial poses. The common goal region is denoted by the shaded region and the goal state the black dashed line. Black dots indicate the end of the action sequence.

tecture were trained. Early stopping was implemented to prevent overfitting. An evaluation of the three networks is shown in Figure 2 and Table 1. All six networks showed promising prediction performances with encouraging rates of convergence within 50 epochs. In addition, we evaluated the networks statistically with the MSE (5) and mean prediction errors (MPEs) given by $\mathcal{L}_{P,X} = \frac{1}{N}\sum_{j=1}^{N}\|X_j - \hat{X}_j\|$ and $\mathcal{L}_{P,\theta} = \frac{1}{N}\sum_{j=1}^{N}|\theta_j - \hat{\theta}_j|$, where $X_j := [x_j, y_j]^T$ and $\hat{X}_j := [\hat{x}_j, \hat{y}_j]^T$. The average MPEs for all the six networks with $\mu = 0.2$ showed positive results of $< 5mm$ and 0.05rad for the displacement and orientation errors respectively.

5.1.2 Varying Friction

Friction is controlled by the friction coefficient μ. Lower μ results in lesser friction. In this experiment, μ values of 0.1 and 0.2 were compared. In both simulations, Object_1 with COM_A position (COM_{A1}) was used with all the other settings kept constant. Similarly to the experiment for varying COM positions, the results for this experiment are shown in Figure 2 and Table 1. The network trained to model the dynamic system with $\mu = 0.1$ also converged within 50 epochs and obtained low test MPEs $L_{P,\theta} < 0.08$ rad and $L_{P,X} < 6$mm. These values are larger than those for the $\mu = 0.2$ friction setting, possibly due to low-friction intermittent sliding between the end-effector and the object during the push.

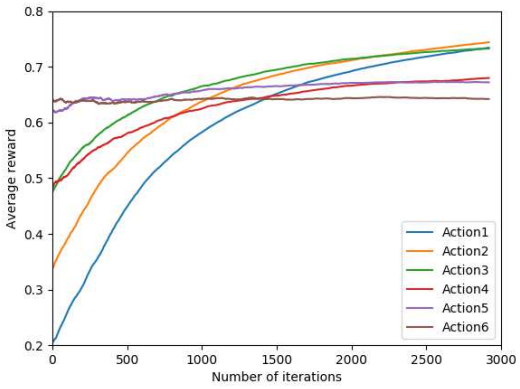


Fig. 5.: Average rewards for all actions in a sample push sequence converge with the number of MCTS iterations.

5.2 Evaluating MCTS for push planning

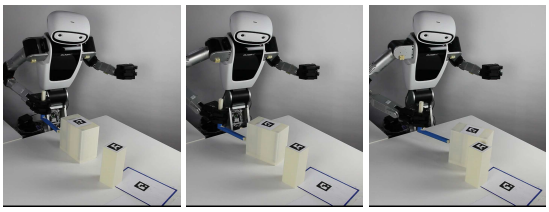
For both simulated and real robot studies, our robot performed the push planning task while avoiding an obstacle, if present. In our implementation of MCTS, a simulation rollout depth of 3 was used to help to ease computational burden. For replanning, we set ε to be 0.1m for both x and y-directions and 0.524rad for orientation. Replanning will be triggered when the accumulated error between the predicted state and ground truth exceeds ε for any state feature. The reward weights are set as $\rho = 0.5$ and $\varrho = 0.7$.

5.2.1 Simulated robot

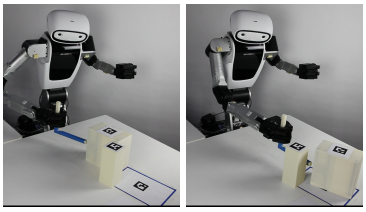
We demonstrate the MCTS implementation in simulation by showing snap shots of a generated representative action sequence in Figure 3. Figure 4 illustrates 13 object trajectories, sampled at different initial poses, converged to the common goal region. Of the 13 sequences, 7 converged within 5 actions, and 6 converged in 6-8 actions. This indicates that both the Euclidean distance and orientation changes were taken into consideration during planning. Thus, the planner is able to recognize and switch between different motions depending on the current state relative to the goal region.

5.2.2 Real robot

For the real robot setup, an obstacle was placed between the object and the goal. We added another heuristic by only considering the object edge that allows for the most number of performable actions during motion planning. This heuristic allowed us to narrow down the action space to an edge E_i. Figure 6 shows snap shots of an action sequence executed by the real robot (more in supplementary video). The robot pushed the object to the goal region while circumventing the obstacle. Note that replanning was triggered more often in the real robot setup due to slight differences in the object properties. However, by re-planning automatically, the robot still succeeded towards the goal most of the time.



(a) Initial (b) Action 1 (c) Action 2



(d) Action 3 (e) Action 4

Fig. 6.: Action sequence planned and executed by a real robot to push an object around an obstacle to the goal (see video accompanying this paper).

5.2.3 Reward Shaping

We evaluated R_{dense} by plotting the average reward of each action generated in a sampled push sequence against the number of MCTS iterations as illustrated in Figure 5. The average rewards for all actions in the push sequence followed an upward trend before converging to an optimal reward. This indicated that the reward function provides accurate evaluations of terminal states with respect to task completion. However, a flattening trend is visible across the actions. By reducing the differences between the current state and the goal region, the measurement error sensitivities for both Euclidean distance and orientation were reduced. Therefore, this resulted in reduced improvements in the average reward of each action in the sequence.

6. CONCLUSION

In this work, a model-based reinforcement learning framework combining learning through an LSTM neural network and planning with MCTS was proposed. We demonstrated that the LSTM network can model the underlying dynamics of an unknown object in pushing operations across various environment and object settings while the MCTS algorithm utilised the learned model as a black-box simulator in generating optimal action sequences for task completion with the use of a dense reward function and various heuristics. The result is a robust framework that can perform push planning with features such as obstacle avoidance. The study provides a proof-of-concept towards a closed-loop planning framework for non-prehensile manipulation. In future work, we plan to extend this framework to explore an online learning approach that can learn incrementally. An initial sub-optimal model can be used and optimized subsequently during the task, reducing the sample complexity of initial model training. Possibilities of transfer learning across different environment and object settings can be explored to provide further versatility. Furthermore, the framework can be generalized to handle irregular polygons.

ACKNOWLEDGEMENTS

The authors are grateful to Minh Khang Pham for the fruitful discussions on the experiment design and implementation, as well as Samuel Cheong, Chong Boon Tan and Jun Li for their assistance with the robot experiments.

This research is supported by grant no. A19E4a0101 from the Singapore Government's Research, Innovation and Enterprise 2020 plan (Advanced Manufacturing and Engineering domain) and administered by the Agency for Science, Technology and Research.

REFERENCES

[1] W. Yuan, J. A. Stork, D. Kragic, M. Y. Wang, and K. Hang, "Rearrangement with nonprehensile manipulation using deep reinforcement learning," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 270–277.

[2] W. Bejjani, M. R. Dogar, and M. Leonetti, "Learning physics-based manipulation in clutter: Combining image-based generalization and look-ahead planning," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019.

[3] K. Lowrey, S. Kolev, J. Dao, A. Rajeswaran, and E. Todorov, "Reinforcement learning for non-prehensile manipulation: Transfer from simulation to physical system," in *IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*, 2018, pp. 35–42.

[4] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, MIT Press, Cambridge, MA, USA, 1st edition, 1998.

[5] M. T. Mason, "Mechanics and planning of manipulator pushing operations," *The International Journal of Robotics Research*, vol. 5, no. 3, pp. 53–71, 1986.

[6] K. M. Lynch, "Estimating the friction parameters of pushed objects," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 1993, pp. 186–193.

[7] D. Kubus, T. Kroger, and F. M. Wahl, "On-line estimation of inertial parameters using a recursive total least-squares approach," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2008, pp. 3845–3852.

[8] Y. Wang, "A new concept using LSTM neural networks for dynamic system identification," in *2017 American Control Conference (ACC)*, May 2017, pp. 5324–5329.

[9] J.K. Li, D. Hsu, and W.S. Lee, "Push-net: Deep

planar pushing for objects with unknown physical properties," in *Robotics: Science & Systems*, 2018.

[10] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine, "Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning," in *IEEE International Conference on Robotics and Automation*, 2018, pp. 7559–7566.

[11] P. Agrawal, A. Nair, P. Abbeel, J. Malik, and S. Levine, "Learning to poke by poking: Experiential learning of intuitive physics," in *Proceedings of the 30th International Conference on Neural Information Processing Systems (NIPS'16)*, 2016, pp. 5092–5100.

[12] F. Ebert, C. Finn, S. Dasari, A. Xie, A. X. Lee, and S. Levine, "Visual foresight: Model-based deep reinforcement learning for vision-based robotic control," *arXiv preprint arXiv:1812.00568*, 2018.

[13] J. A. Haustein, I. Arnekvist, J. A. Stork, K. Hang, and D. Kragic, "Learning manipulation states and actions for efficient non-prehensile rearrangement planning," *arXiv preprint arXiv:1901.03557*, 2019.

[14] J. E. King, V. Ranganeni, and S. S. Srinivasa, "Unobservable monte carlo planning for nonprehensile rearrangement tasks," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 4681–4688.

[15] E. J. Jacobsen, R. Greve, and J. Togelius, "Monte Mario: Platforming with MCTS," in *2014 Annual Conference on Genetic and Evolutionary Computation (GECCO'14)*, 2014, pp. 293–300.

[16] F. R. Hogan and A. Rodriguez, "Feedback control of the pusher-slider system: A story of hybrid and underactuated contact dynamics," *arXiv preprint arXiv:1611.08268*, 2016.

[17] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, November 1997.

[18] T. Flash and N. Hogan, "The coordination of arm movements: an experimentally confirmed mathematical model," *Journal of Neuroscience*, vol. 5, no. 7, pp. 1688–1703, 1985.

[19] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations (ICLR)*, 2015.

[20] G. Chaslot, S. Bakkes, I. Szita, and P. Spronck, "Monte-carlo tree search: A new framework for game ai," in *The Fourth AAAI Conference on Artifical Intelligence and Interactive Digital Entertainment (AIIDE)*. 2008, pp. 216–217, AAAI Press.

[21] L. Kocsis and C. Szepesvári, "Bandit based monte-carlo planning," in *Proceedings of the 17th European Conference on Machine Learning*, Berlin, Heidelberg, 2006, pp. 282–293, Springer-Verlag.

[22] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi, "A fast procedure for com-puting the distance between complex objects in three-dimensional space," *IEEE Journal of Robotics and Automation*, vol. 4, pp. 193–203, April 1988.

[23] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *24th International Conference on Neural Information Processing Systems (NIPS'11)*, 2011, pp. 2546–2554.