Planning-integrated Policy for Efficient Reinforcement Learning in Sparse-reward Environments

Christoper Wulur
Universität Hamburg
8wulur@informatik.uni-hamburg.de

Cornelius Weber
Universität Hamburg
weber@informatik.uni-hamburg.de

Stefan Wermter
Universität Hamburg
wermter@informatik.uni-hamburg.de

***Abstract*—Model-free reinforcement learning algorithms can learn an optimal policy from experience without requiring prior knowledge. However, model-free agents require vast amounts of samples, particularly in sparse reward environments where most states contain zero rewards. We developed a model-based approach to tackle the high sample complexity problem in sparse reward settings with continuous actions. A trained world model is queried by a particle swarm optimization (PSO) planner and employed as the action selection mechanism, hence taking the role of the actor in an actor-critic architecture. Parameters of the PSO regulate the agent's exploration rate. We show that the planner aids the agent to discover rewards even in regions with zero value gradient. Our simple planning integrated policy architecture learns more efficiently with fewer samples than continuous model-free algorithms.**

***Index Terms*—Reinforcement Learning, Planning, Particle Swarm Optimization**

I. Introduction

Reinforcement learning (RL) is a branch of machine learning that enables agents to learn through trial and error by receiving a reward signal from the environment as feedback on their actions. An agent will try to select among the best possible actions to maximize these rewards, given its current state at the time. Unlike supervised learning, reinforcement learning does not require labeled examples. With the recent breakthrough of deep RL algorithms, various challenging tasks could be solved.

RL algorithms can be classified into model-based and model-free algorithms. Model-based RL utilizes a world model, which is a function approximator of the environment's dynamics. By employing the world model, the agent is capable of generating look-ahead plans [1], which can reduce the amount of actual environment exploration. However, in complex environments, an accurate world model is often unavailable. Moreover, planning compels to foresee multiple states, given a series of actions to reach a goal, yielding imprecision as the error accumulates. Consequently, model-based reinforcement learning is susceptible to a model bias, which interferes with the agent's performance [2], [3].

Model-free RL, on the other hand, does not require any knowledge of the environment. The agent learns directly from the environment through experience. In its trial and error approach, model-free RL demands collecting vast amounts of samples [4]–[6]. This situation is aggravated if the agents learn in sparse reward environments, where most environment states yield zero reward feedback. An agent may need to discover a long sequence of correct actions that leads to these sparse rewards, highly unlikely to be found through random exploration [7]. While finding the goal is already a difficult task, even more, to learn the optimal policy from many different states [8].

In general, providing a sparse, binary reward is straightforward to model [7], [9]. For example, in a grasping task, a robot is rewarded for grasping a given object. A sensor can be used to detect whether the agent successfully grasps the object.

One could attempt to employ techniques such as reward shaping to help an agent learn in sparse settings. Reward shaping uses domain expertise to transform a sparse reward signal into a continuous dense reward. The goal is broken down into subgoals, and the reward function is modified to return a continuous reward in larger state space regions. For instance, in the pendulum environment, where the goal is to balance the pendulum, the pendulum's angle could guide the agent. It indicates the agents' progress; the closer the pendulum moves upward, the more reward the agent receives. Unfortunately, reward shaping is not a trivial task. It requires thorough experimentation [10], [11]. Failure to design a suitable reward function leads to premature convergence due to local optima [9], [12]. Often, RL agents exploit the reward signals from the environment, producing unexpected behavior [8], [13].

We are interested in solving problems in continuous spaces and sparse reward settings. In practice, many problems are in these settings, especially in robotics. For example, in grasping tasks, the robot must utilize its joint motors in continuous space. The agent must discover an optimal policy given a large space to explore, where the rewards are mostly zero. This kind of environment leads to high sample complexity.

Inspired by the recent success of model-based algorithms, we propose a Planning-Integrated Policy (PIP) algorithm to reduce the sample complexity of an RL agent in sparse reward settings. The main idea is to utilize planning to support the agent to look beyond any gradient-free region. We realize the planner in a continuous environment by a particle swarm optimizer (PSO). Based on one trained predictive world model, the

PSO implements multiple plans, where diversity emerges from probing different actions. The found optima of the iterative PSO algorithm correspond to locally optimal plans. The PSO parameters allow regulating the planner's optimization, where minimal complexity, i.e., one plan with a single step look-ahead, yields the standard model-free actor-critic model.

II. Related Work

Recent model-based approaches, such as MuZero [14] Dreamer [15], and PlaNet [16], have demonstrated their efficiency compared to model-free approaches. Nevertheless, none of these algorithms focus on the continuous domain under sparse reward settings. Dreamer and PlaNet focus on solving problems in image-based observations. Besides, PlaNet's performance showed high variance under sparse reward settings. MuZero is intended to solve discrete actions by using a Monte Carlo tree search variation and only recently has been extended to a continuous version [17].

Hein et al. [18] introduced Particle Swarm Optimization Policy (PSO-P) to solve problems in the continuous domain. In PSO-P, model predictive control utilizes PSO to optimize a plan with respect to a reward function. The optimizer employs a world model to predict future states. Hence, it produces a series of actions that are optimized. However, PSO-P requires a dense reward function to optimize in order to plan.

Therefore, we propose a Planning-Integrated Policy (PIP) algorithm to reduce RL agents' sample complexity in the continuous domain with sparse rewards. PIP's planner is similar to PSO-P, but unlike PSO-P, PIP estimates the reward function by using a critic in an actor-critic architecture.

III. Planning-Integrated Policy Algorithm

Model-free agents excel in finding a goal from a state that is not far from the goal. The reason for this is because over the course of learning, the reward values are propagated from the goal. However, in a sparse setting, most states generate zero rewards. If the agents start from a state far from the goal, they will have difficulty discovering the large-gradient region from the current position. In this situation, planning can support the agent to look beyond a gradient-free region.

Figure 1 sketches the PIP model. PIP consists of three components: a world model, an optimizer, and a value function. PIP follows the actor-critic architecture, where the optimizer extends the actor and, the critic estimates the value function

A world model M is an approximator of the environment's dynamics. In our experiments, we pre-train the world model in an unsupervised fashion from random exploration. It could also be trained during the learning phase of the agent, however, a yet untrained world model during early learning could lead to disturbance, since the agent uses it.

The critic component is implemented as a neural network that estimates the value function $V(s_t)$ of a given state s_t at time t, which is updated via temporal difference learning. The critic loss is $(r_t+\gamma V(s_{t+1})-V(s_t))^2$, where r_t is the current reward, $V(s_{t+1})$ is the value of the next state s_{t+1} following actions from the planner, and γ is a discount factor.

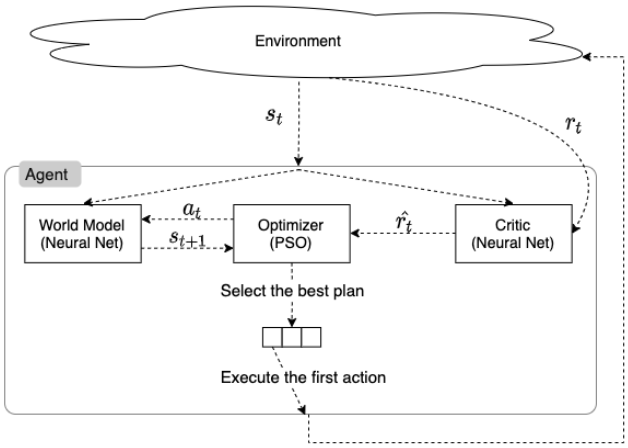

Fig. 1. Planning Integrated Policy Architecture

The procedure of PIP is explained in Algorithm 1. The environment is reset at every new episode. Planning is done anew every step t when the agent executes an action. This replanning keeps the planning error minimal, which arises due to the imprecision between the world behavior and the world model's prediction. This process is repeated until the agent reaches a terminal state or a maximum step T. At the end of an episode, state-action transitions $\{s_t, a_t, s_{t+1}\}$ and their rewards r_t are stored in the replay buffer D. Later, the buffer is sampled with a batch size B to update the critic. The whole process is repeated for E episodes.

Algorithm 2 shows the process of plan optimization with the PSO planner. First, the planner generates a swarm with P particles. The parameters of each particle represent one plan, which consists of a sequence of actions. The optimizer needs to evaluate each plan. With the agent's current state s_t and the first action from the plan a_t, the world model M predicts a new state s_{t+1}. Then, the predicted state s_{t+1} is used together with the next action a_{t+1} from the plan, to predict next future state s_{t+2}. This process continues until the plan length N is reached. Finally, the plan evaluation is done by summing up the estimated returns using the value function V of the predicted states.

The next step is to optimize the swarm. Every particle is evaluated based on its current position in parameter space, and it has a memory of its best position. Also, the swarm is divided into neighborhoods. Each neighborhood selects the best particle among its members. Then, a particle's position is updated according to its velocity, considering the individual best position and its neighborhood best position.

The choice of the optimization method is not limited to PSO. One could use gradient descent or other optimization methods. However, unlike gradient descent, PSO is derivative-free optimization, which means that PSO does not require the objective function to be differentiable. Besides, particle swarm optimization is appealing since it can solve a non-convex optimization problem [18].

Algorithm 1 Planning-Integrated Policy

Input :
Number of Episode E, Number of Max Step T, Batch Size B

Initialize :
World Model M, Planner P, Dataset D, Value function V (Critic) parameterized with θ

for *episode e = 1..E* **do**
  // Data Collection
  $s_t \leftarrow env.reset()$
  for *step t = 1..T* **do**
    $a_t \leftarrow$P(s_t)
    $s_{t+1}, r_t \leftarrow env.step(a_t)$
    $s_t \leftarrow s_{t+1}$
  end
  D.append($\{s_t, a_t, s_{t+1}, r_t\}_{t=1}^{T}$)

  // Update Critic
  Draw samples $\{s_t, a_t, s_{t+1}, r_t\}_{i=1}^{B}$ from D
  $V(s_t) = r_t + \gamma$ V(s_{t+1})
  Compute loss L(θ) $\leftarrow \mathbb{E}[(r_t + \gamma V(s_{t+1}) - V(s))^2]$
  Update parameter $\theta \leftarrow \theta - \alpha \nabla_\theta L(\theta)$
end

Algorithm 2 Particle Swarm Optimization Planner

Input :
Plan Length N, Discount factor γ, Number of PSO's Iterations I, Number of PSO's Particles P, Cognitive Coefficient c_1, Social Coefficient c_2, Weight Inertia w, World Model M, Value function V

Initialize :
Particles' Position x, Particles' Velocity v
$r_1, r_2 \sim$ *Uniform*$(0, 1)$

Function *evaluate(plan)* **is**
  $r \leftarrow 0$
  for *length n = 1..N* **do**
    $a \leftarrow plan[n]$
    $s_{t+1} \leftarrow M(s, a)$
    $r \leftarrow r + V(s_{t+1})$
    $s_t \leftarrow s_{t+1}$
  end
  return r
end
for *iteration i = 1..I* **do**
  for *particle p = 1..P* **do**
    // Set personal best position
    if $evaluate(x_p(i)) < evaluate(y_p(i))$ **then**
      $y_p = x_p$
    // Set neighbourhood best position
    if $evaluate(y_p(i)) < evaluate(\hat{y}_p(i))$ **then**
      $\hat{y}_p = y_p$
  end
  for *particle p = 1..P* **do**
    // Update position and velocity

$$v_p(i+1) = wv_p(i) + c_1 r_1[y_p(i) - x_p(i)] + c_2 r_2[\hat{y}_p(i) - x_p(i)]$$

$$x_p(i+1) = x_p(i) + v_p(i+1)$$

  end
end
$a = \arg\min V(x)$
return x_a

An attractive aspect of using PSO is that the particles can be illustrated as an agent's mental exploration. A particle's initial position is randomly generated, reflecting a random plan. In the beginning, the value function does not contain information about the reward; optimizing the particles is not useful. Not optimizing these particles is essentially similar to generating random plans, which corresponds to considerable exploration noise. Changing the PSO's parameters, i.e., increasing the number of particles and iterations, reduces the exploration noise level since plans nearer to an optimum will be found. Vice versa, reducing either the number of particles or iterations, or both, has the effect of increasing exploration noise.

We can adjust these parameters dynamically, e.g., by employing simulated annealing [19] so that the particles are exploring in the early episodes. While after discovering a reward, agents can start optimizing the plans.

IV. Methodology

A. Continuous Grid World

First, the performance of the agent is analyzed in a sparse-reward continuous grid world problem. Both action and observation are in the form of coordinates (X, Y) where their value ranges from 0 to 1. The agent movement is limited to a change ΔX and ΔY within the interval $[-0.1, 0.1]$. The goal is a square region of side length of 0.1 located at coordinate (0.45,0.45). When the agent arrives inside the goal state, it will receive a reward +1. Otherwise, it will gain 0 rewards.

Visualization of Planning

Figure 2 visualizes selected plans, which constitute the PSO particles. The red dots denote initial position, where the optimizer was analyzed in 8 different initial states. Each color represents a particle. Among the actually used 50 particles, only the best 3 particles are plotted for clarity. The dotted lines indicate the imagined plans, while solid lines show the corresponding actual executed action. The background color represents the critic's value for that state; the brighter the color, the higher its value. In order to depict the hills and valleys of the critic's function, the state space was discretized with a resolution of 0.01.

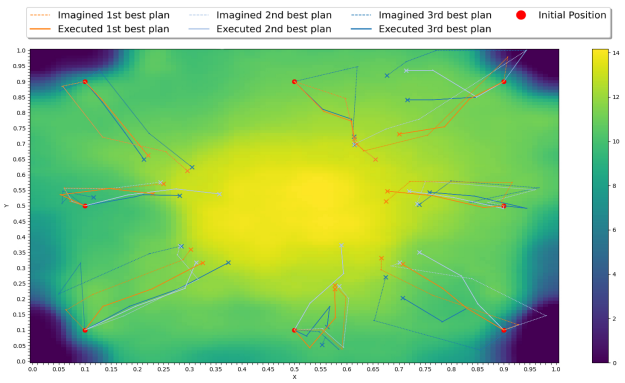

Fig. 2. Planning Visualization in the Continuous Grid World

From the visualization, we can investigate the planning behavior. The particles were attracted towards the rewards, despite imprecision between the actual execution and the imagined plan. For example, in the bottom left state, the first particle imagined to turn up, but it turned diagonally straight to the rewards in reality. This is because the trained world model used for planning may deviate from the actual world behaviour. Nevertheless, it provides a good heuristic for the planner.

B. Swing-up Balancing Pendulum

The second environment is a pendulum problem where the agent's objective is to swing up and balance the pendulum. The environment state consists of two observations: the pendulum's angle θ and the pendulum's velocity $\dot{\theta}$. The angle θ is in form of radians which is normalized to have values ranging from $-\pi$ to π. As for the velocity, $\dot{\theta}$ is limited to have a value from -8 to 8. The agent obtains three observations: the sine and cosine of θ; and the velocity $\dot{\theta}$.

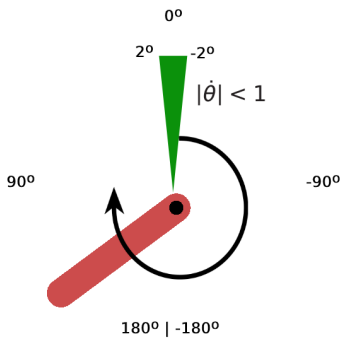

Fig. 3. Sparse swing-up pendulum environment

There is only one continuous action in the pendulum environment. The action represents the angular torque of the pendulum. Its value ranges from -2 to 2, where the sign indicates to move to the right or left. With this maximum torque, it limits the agent to reach the top by solely using its motor. The agent must swing and exploit the momentum and gravity to reach the top.

The swing-up pendulum is most challenging in a sparse reward setting, which does not permit inclusion of expert knowledge into the reward function. A reward is only given when the agent reaches a specific angle (from -2 to 2 degrees) while the velocity is low, i.e., $|\dot{\theta}| < \dot{\theta}^{thres}$ with a small threshold $\dot{\theta}^{thres} = 1$, as the green region indicates in Figure 3. The agent must satisfy both constraints and stay within this range for at least five steps to receive a reward. After receiving a reward, the agent keeps receiving rewards until it cannot hold the angle and velocity constraints.

C. Preprocessing by Radial Basis Function

Both continuous grid-world and pendulum environments are XOR-like problems in the sense that there is no monotonous relation between the input and output variables. It is hard to train neural networks for such linearly non-separable problems. Therefore, we use radial basis functions for preprocessing, which transform the observation space into higher dimensionality. Thus, the input space is easier separable compared to the original state representations.

Some studies [20], [21] have reported that RBF networks yielded faster execution time than neural networks. This transformation should aid the neural networks by making the problem easier so that the agent could learn faster.

As an example, we used the pendulum environment to show the advantages of the radial basis function. The pendulum environment has three observation variables: the pendulum position in Cartesian coordinates (x,y) and its velocity.

To explore the number of RBF units needed for the pendulum environment, we used various parameters. We used the combinations of 3,5,7 for the position space and 5,11,17 for the velocity space. CACLA agents were used to evaluating the number of RBF units.

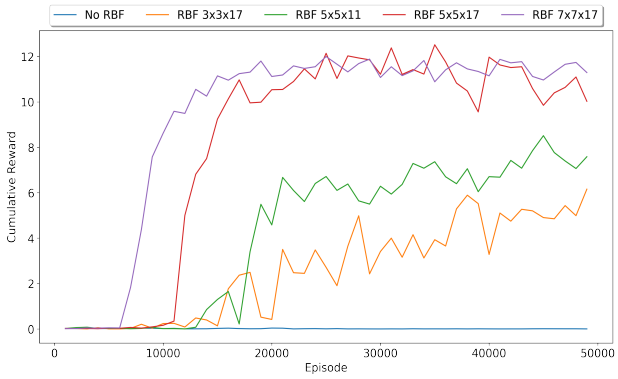

Fig. 4. Training results of CACLA agents with various RBF units

We used CACLA since it has a fast training time and fewer hyperparameters configuration than PIP and TD3. The agents are trained in the pendulum environment with 50 steps. Other parameters of CACLA are set to be the same among all agents. Thus, we can compare the effect of RBF units by observing the agents' cumulative rewards.

Labels in Figure 4 denote the number of radial basis units used in each dimension. For example, 5_5_17 means that the agents used five radial basis units each for x and y space and 17 units for the velocity space. This configuration resulted in 425 dimensions (5 x 5 x 17). An odd number of units were used to have a unit that could represent the center value.

As we can see from the result, the agent without radial basis units was unable to learn. It is because it had difficulties to perform the nonlinear separation in low-dimensional space. The agent was unable to learn. As we increased the number of radial basis units, the performance was improved.

From the result, we can see the benefit of using a radial basis function representation as the input. However, the radial basis function comes with the cost of more computation as the input dimensions increased.

We used 5x5x17 radial basis function units for the pendulum environment. For the continuous grid world environment, we used $10x10$ radial basis function units.

D. Hyperparameter Configuration

To ensure CACLA, TD3, and PIP behaved optimally in both environments, we tuned hyperparameters using a grid search. All agents were set up in the same environment settings, including the network architectures and the replay buffer. The final configuration is shown in Table I.

	Hyperparameter	Continuous Grid World	Pendulum
Common	Episodes	2e4	2e4
	Max steps	15	100
	Critic layers	64-64-32	1024-512-256
	Critic learning rate	1e-3	1e-3
	Critic activation function	relu	relu
	Replay buffer size	1.5e5	1e6
	Training batch size	256	256
	Radial basis units	10x10	5x5x17
	Discount factor γ	0.95	0.95
CACLA	Actor layers	64-64-32	1024-512-256
	Actor learning rate	1e-3	1e-3
	Actor activation function	tanh	tanh
	Exploration noise	0.01	0.5
TD3	Actor layers	64-64-32	1024-512-256
	Actor learning rate	1e-3	1e-3
	Actor activation function	tanh	tanh
	Exploration noise	0.01	0.5
	Policy Noise	0.0	0.0
	Tau	0.05	1.0
PIP	PSO particles	50	50
	PSO iterations	3	3
	PSO c1	0.5	1.4
	PSO c2	0.5	1.4
	PSO w	0.3	0.7
	PSO neighbourhood	20	20
	Plan length	5	5

TABLE I
HYPERPARAMETER CONFIGURATIONS USED IN CONTINUOUS GRID WORLD AND PENDULUM

V. RESULTS

All the agents were trained with the same number of episodes and steps. The number of steps is the number of interactions between agents and the environment in one episode. We reset the environment and assigned the agents in a different random initial state every new episode. Each algorithm was run three times to ensure the result was not produced by chance.

A. Continuous Grid World

Figure 5 shows the cumulative rewards of the three algorithms. The x-axis shows the episode, and the y-axis the cumulative rewards. The lines indicate the mean of cumulative rewards, which are smoothed with a window size of 1000, while the areas show the percentiles 5 to 95 over 3 different runs. The grey dotted line shows an expected optimal solution as a comparison for the agents' performance, which was approximated by taking the average of the best and worst-case scenario of the continuous grid world environment.

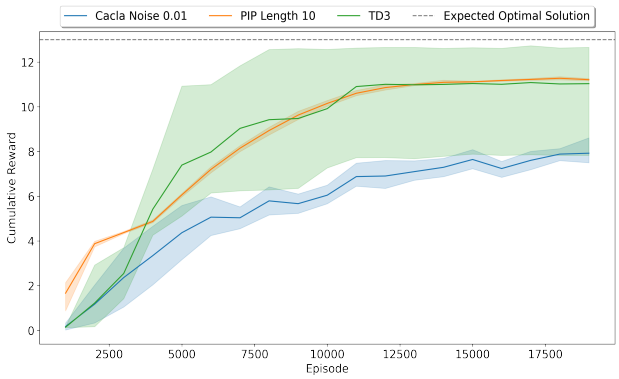

Fig. 5. Cumulative rewards in the training phase of CACLA, TD3 and PIP agents in the continuous grid world

In the best-case scenario, the agents started inside the reward area in the center. Assuming that agents had an optimal policy, where it could stay inside until the episode ended, the maximum rewards would be 15.

For the worst-case scenario, the agents started in the corner, for example (0,0). The step was limited to 0.1, both stepping in the x and y-axis. The maximum distance could be achieved by walking diagonally with a distance of 0.14. Hence, the agents required at least four steps to reach the center. Consequently, the maximum reward they could receive was 11. Therefore, we expected the agents, when they behaved optimally, would have rewards converged to 13, as indicated by the grey line.

From the result, TD3 and PIP reached similar mean cumulative rewards above 10, while CACLA received cumulative rewards below 8. However, TD3 had more variance compared to PIP and CACLA.

At the beginning of the training, the PIP agents achieved higher rewards than both TD3 and CACLA. On the other hand, CACLA and TD3 agents were struggling to find the reward. As the episodes progressed, all the agents were able to learn and improve their performance.

B. Swing-up Balancing Pendulum

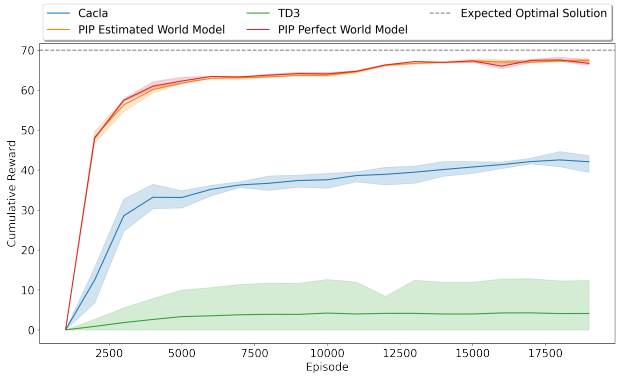

Fig. 6. Cumulative rewards in training phase of CACLA, TD3 and PIP agents in the pendulum environment

Figure 6 shows the benchmark of the three algorithms in the training phase. The dotted-line depicts expected rewards. In the best-case scenario, the agents started already at the top with low speed, and they needed five steps to receive the rewards. Given an episode length of 100, the maximum reward they could get in the best-case scenario was 95. On the other hand, if they started from the bottom, they needed at least 50 steps to reach the top and five steps to stay balanced before receiving the reward. Hence, in the worst case, they would receive up to 45 cumulative rewards. Taking the average of both the best and worst cases, we would expect the average rewards to converge toward approximately 70.

We also experimented to see the performance difference between the approximated world model and the perfect world model, where the environment's dynamics are known and modeled accurately. From the result, PIP seems to be the best among CACLA and TD3. Both, PIP with estimated world model and with perfect world model, had similar performance. CACLA agents had a mean of cumulative rewards above 30 after 5000 episodes. It seems that the CACLA could gather the rewards when it started from the top but failed on the swing-up task. TD3 agents had the worst performance, which had average cumulative rewards below 10 of the entire episodes.

We tested whether the trained agents were able to solve both swing-up and balancing tasks. Unlike in the training phase, where the agents' initial angle was randomly chosen from 360^o, the initial angle was sampled from only the lower half circle and with 0 velocity, to test the challenging swing-up task. Starting from this state, the maximum cumulative rewards that agents could get were 45. In addition, we removed the Gaussian noise for CACLA and TD3 agents during evaluation to ensure they were in full exploitation. The evaluation was done every 1000 episodes.

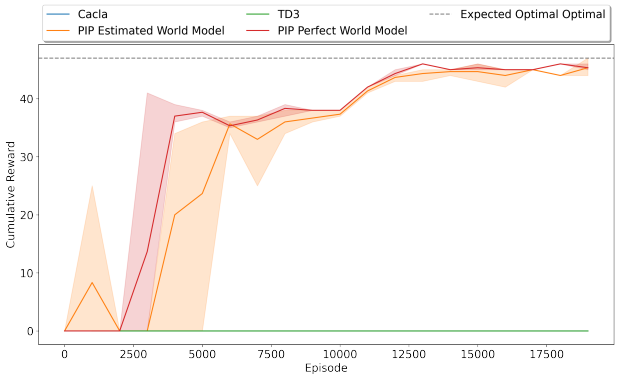

Fig. 7. Cumulative rewards in the evaluation phase of CACLA, TD3 and PIP agents in the pendulum environment

Figure 7 shows the results of cumulative rewards in the evaluation phase. Both CACLA and TD3 agents were not capable of achieving the swing-up tasks. However, they were capable of balancing the pendulum since they could accumulate some rewards, as we have seen in Figure 6

On the other hand, both PIP with/without the perfect world model successfully performed the swing-up balancing tasks. They were not able to perform efficiently in the early episodes. There is a steep performance boost between episode 2500 and 4000 for PIP with the perfect world model. A similar improvement happened for PIP without the perfect world model. At later stages, both converged and produced the most optimal solutions with lower variance.

VI. Discussion

A. Experiment Results

The experimental results demonstrate the advantage of planning. With the help of a world model, a planner could produce actions towards the goal. This mechanism leads the agents to discover rewards more often during the training phase.

The model-free agents had difficulties in both the continuous grid world and swing-up balancing pendulum tasks. Policy and value updates are typically slow and require vast samples to derive policy improvement [22]. The learning is exacerbated when the agents do not find a meaningful signal.

CACLA agents performed better compared to TD3 agents in both tasks. One explanation is that CACLA only updates its policy if the temporal difference is positive, which means the policy is updated only when the action yields a better reward. This behavior is beneficial in a sparse reward environment where many actions generate zero rewards. Thus, CACLA better preserves the knowledge where the goal is. In addition, TD3 estimates a Q-function while learning the Q-values is prone to overestimation bias [23]. Simultaneously, we found that policy noise as a policy regularization method had a disruptive impact on learning in sparse reward settings.

The fact that CACLA agents could collect rewards during the training phase means that they were able to balance the pendulum. However, in the evaluation phase, where the agent's initial state was set to 180 degrees with 0 velocities, they had difficulties performing the swing-up task.

This result demonstrates the limitation of the model-free algorithm. Both CACLA and TD3 rely on Gaussian noise for exploration. To perform swing-up behavior, agents must perform a completely different action. For example, when an agent reaches the point of no return, the maximum action could not move the agent higher due to gravitation. It has to rotate in the opposite direction with maximum torque. With the help of gravity, reaching a higher angle is possible on the other side. This switching rotation behavior is impossible to discover if we use a small amount of Gaussian noise. As we increase the exploration noise, it increases the chance of discovering the swing behavior. However, at the same time, it reduces the chance of the agent receiving rewards since it makes the balancing task more problematic as it requires precision.

PIP exploits its knowledge as soon as a reward is once discovered. Assisted by the world model, it could plan to reach states that are closer to the rewards. Using the same case, in the pendulum environment, when a PIP agent reaches the point of no return, it could foresee a few steps ahead and see a better state on the other side despite the necessity of passing

by low-value states first. It enables the agent to perform swing behavior and eventually reach the top. The balancing task is also achievable since the agent knows the rewards are there and keeps balancing the pendulum to collect them. Hence, during the training time, it collects more meaningful experiences compared to model-free algorithms. As good samples are collected, the critic is getting better, the plan is improved, and the cycle repeats. Consequently, it is more sample efficient than model-free algorithms.

Even without a perfect world model, PIP agents can efficiently discover rewards. As we see from Figures 6 and 7, there was a slight deviation between a planner with a perfect world model and one with an estimated world model. In the evaluation, the impact was apparent where the agents without the learnt world model needed more steps to reach the top because of the world model's imprecision. Nevertheless, as the learning progressed, the agent with the estimated world model reached equal performance with the perfect world model. Despite the world model's inaccuracy, it is capable of aiding the planner towards the goals.

B. Limitation of PIP

With all of its merits, PIP also has some limitations. As PIP recommends to use PSO as the planner, there are some concerns to consider. Firstly, PSO has many hyperparameters to tune. The number of particles, iterations, weight, social, individual coefficients has to be carefully selected. As each environment's space is unique, one should scale the particle's coefficients and weight accordingly. Nevertheless, hyperparameter tuning is needed by other algorithms. One could use a grid search or Bayesian optimization to find the right values.

Secondly, it needs more training time compared to model-free algorithms. For example, in a continuous grid world with 20000 episodes and 15 steps, PIP's training time was 10 hours. In comparison, TD3 and CACLA required 1 and 0.5 hours, respectively. The most time-consuming process happens during planning as the longer the plan's length, the more time it needs since the optimization process is done iteratively. Moreover, during the optimization, each particle needs to be evaluated one by one, contributing more processing time. Furthermore, the agent replans for every action performed to minimize the world model's inaccuracy bias.

Nevertheless, we did not utilize any process parallelization. PSO could benefit from parallelization, especially while evaluating the particles' performance. Each particle could have separate threads since they are not dependent on each other. Plus, we did not use any GPUs, which could speed up the processing time of PSO. Another option is to reduce plan lengths and the number of iterations. From our experience, the plan's length and optimizer's iteration contribute the most to the training time due to the unrolling plan process. One could decrease the number of plan to accelerate the training process, but must be done carefully as the shorter plan lengths may result in the planner not seeing rewards in the long run.

Another criticism of the model-based approach is that a world model is often not available. The prerequisite of estimating the state-action transitions is necessary. It may not be a trivial task, especially collecting the data, such as robot locomotion tasks, where the agents' objective is to move to a specific location while avoiding collisions. It has large dimension spaces, both for observation and action. Collecting data that covers all possible actions and states is expensive. Furthermore, it is hard to have various initial states.

One solution offered by Plan2Explore [24] is to utilize exploration to create a global world model. It uses an ensemble of world models and a planner whose objective is to reduce the world models' disagreement. Because of this objective, the agents are attracted to states that are difficult to predict until the world models agree with each other regarding the state prediction. Consequently, a robust global world model is produced. Plan2Explore could benefit from the PIP algorithm. The idea of Plan2Explore is to have a global world model to be used in many tasks. However, a reward function is required for task adaptation. As we have argued, designing a reward function is challenging; PIP offers a mechanism to estimate the reward function.

C. Comparison to Other Model-Based RL

PIP has similarities and dissimilarities to other model-based algorithms. One of the signatures of model-based approaches is the application of world models to plan ahead. The fundamental difference among these model-based algorithms lies in how they incorporate the world model.

In Dyna-Q [25], the world model is used to update the Q-function. As it is an off-policy method, the value of a state-action pair is equal to the current reward and the future state's maximum value. The role of the world model in Dyna-Q is to predict this next future state. Unlike Dyna-Q, PIP utilizes the world model to unroll the plan.

The unrolling process also happens in Dreamer. However, unlike PIP, Dreamer uses latent states. The planning in imagination is then used to train the policy network. In contrast, PIP does not approximate a policy; instead, it employs planning as the action selector. Besides, PIP's planning process occurs in the same observation spaces as the environment.

As for MuZero, the planning process is different from PIP. MuZero plans by creating a search tree with Monte Carlo Tree Search. The best action is then selected by following a trajectory in the tree search that leads to the highest value. Due to the nature of its discrete world model, MuZero is limited to solving discrete actions. Recent work from Yang et al. extends MuZero to deal with continuous actions [17]. However, tree search in a continuous domain is difficult because of the exploration-exploitation tradeoff. Moreover, MuZero, which does not predict the state, requires reward feedback for learning and cannot be pretrained unsupervised.

The learning process in PIP is to estimate the reward function. Estimating a reward function could be compared to the concept of inverse reinforcement learning. IRL is coined by Ng, and Russell [26], where a policy is given, and the objective is to estimate a reward function of that policy. The policy is in the form of sample trajectories from expert demonstrations.

Despite its similarity to estimate a reward function, PIP is different from IRL, which is essentially used for imitation learning. It produces a reward function that maps state-action values, which highly rewards a state-action pair similar to what it has seen in the demonstration samples. Later on, a regular RL model is used to derive a policy that mimics the expert. Therefore, the policy produced is as good as the expert. On the other hand, PIP estimates a reward function directly towards the environment's objectives. Hence, it is capable of yielding an optimal policy since it is not constrained by any demonstrations.

VII. Conclusion

We demonstrated that model-free agents had difficulties to solve problems in sparse reward settings. We proposed the PIP algorithm, where an agent utilizes a policy with a planner to aid it towards goal states. This lets the agent receive meaningful experiences in a sparse reward environment.

The PIP algorithm was tested in two environments and benchmarked with CACLA and TD3. The model-free algorithms had difficulty to solve the problem in sparse environment settings in which random exploration alone is not enough to help the agents find the reward.

The PIP algorithm delivers a promising approach to tackle high sample complexity in sparse reward environments. In the future, we would like to see whether the planner can produce a correct solution in more complex tasks, such as a humanoid robot locomotion task. The PIP implementation code is available on https://github.com/ctoto93/pip.

Acknowledgments

The authors gratefully acknowledge support from the German Research Foundation DFG under project CML (TRR 169).

References

[1] Sebastian Thrun, Knut Möller, and Alexander Linden. Planning with an Adaptive World Model. In R. P. Lippmann, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems 3*, pages 450–456. Morgan-Kaufmann, 1991.

[2] Anusha Nagabandi, Gregory Kahn, Ronald S. Fearing, and Sergey Levine. Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning. *arXiv:1708.02596 [cs]*, December 2017. arXiv: 1708.02596.

[3] Vladimir Feinberg, Alvin Wan, Ion Stoica, Michael I. Jordan, Joseph E. Gonzalez, and Sergey Levine. Model-Based Value Estimation for Efficient Model-Free Reinforcement Learning. *arXiv:1803.00101 [cs, stat]*, February 2018. arXiv: 1803.00101.

[4] Yuxi Li. Deep Reinforcement Learning: An Overview. *arXiv:1701.07274 [cs]*, November 2018. arXiv: 1701.07274.

[5] Brandon Amos, Ivan Jimenez, Jacob Sacks, Byron Boots, and J. Zico Kolter. Differentiable MPC for End-to-end Planning and Control. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 8289–8300. Curran Associates, Inc., 2018.

[6] Théophane Weber, Sébastien Racanière, David P. Reichert, Lars Buesing, Arthur Guez, Danilo Jimenez Rezende, Adria Puigdomènech Badia, Oriol Vinyals, Nicolas Heess, Yujia Li, Razvan Pascanu, Peter Battaglia, Demis Hassabis, David Silver, and Daan Wierstra. Imagination-Augmented Agents for Deep Reinforcement Learning. *arXiv:1707.06203 [cs, stat]*, February 2018. arXiv: 1707.06203.

[7] Martin Riedmiller, Roland Hafner, Thomas Lampe, Michael Neunert, Jonas Degrave, Tom Van de Wiele, Volodymyr Mnih, Nicolas Heess, and Jost Tobias Springenberg. Learning by Playing - Solving Sparse Reward Tasks from Scratch. *arXiv:1802.10567 [cs, stat]*, February 2018. arXiv: 1802.10567.

[8] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: an introduction*. Adaptive computation and machine learning series. The MIT Press, Cambridge, Massachusetts, second edition edition, 2018.

[9] Mel Vecerik, Todd Hester, Jonathan Scholz, Fumin Wang, Olivier Pietquin, Bilal Piot, Nicolas Heess, Thomas Rothörl, Thomas Lampe, and Martin Riedmiller. Leveraging Demonstrations for Deep Reinforcement Learning on Robotics Problems with Sparse Rewards. *arXiv:1707.08817 [cs]*, October 2018. arXiv: 1707.08817.

[10] Andrew Ng Daishi, Andrew Y. Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *In Proceedings of the Sixteenth International Conference on Machine Learning*, pages 278–287. Morgan Kaufmann, 1999.

[11] M. Seo, L. F. Vecchietti, S. Lee, and D. Har. Rewards Prediction-Based Credit Assignment for Reinforcement Learning With Sparse Binary Rewards. *IEEE Access*, 7:118776–118791, 2019. Conference Name: IEEE Access.

[12] Alexander Trott, Stephan Zheng, Caiming Xiong, and Richard Socher. Keeping Your Distance: Solving Sparse Reward Tasks Using Self-Balancing Shaped Rewards. *arXiv:1911.01417 [cs]*, November 2019. arXiv: 1911.01417.

[13] Pieter Abbeel and Andrew Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Twenty-first international conference on Machine learning - ICML '04*, page 1, Banff, Alberta, Canada, 2004. ACM Press.

[14] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy Lillicrap, and David Silver. Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model. *arXiv:1911.08265 [cs, stat]*, February 2020. arXiv: 1911.08265.

[15] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to Control: Learning Behaviors by Latent Imagination. *arXiv:1912.01603 [cs]*, March 2020. arXiv: 1912.01603.

[16] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning Latent Dynamics for Planning from Pixels. *arXiv:1811.04551 [cs, stat]*, June 2019. arXiv: 1811.04551.

[17] Xuxi Yang, Werner Duvaud, and Peng Wei. Continuous Control for Searching and Planning with a Learned Model. *arXiv:2006.07430 [cs]*, June 2020. arXiv: 2006.07430 version: 1.

[18] Daniel Hein, Alexander Hentschel, Thomas A. Runkler, and Steffen Udluft. Reinforcement Learning with Particle Swarm Optimization Policy (PSO-P) in Continuous State and Action Spaces:. *International Journal of Swarm Intelligence Research*, 7(3):23–42, July 2016.

[19] P. J. M. Laarhoven and E. H. L. Aarts. *Simulated annealing: theory and applications*. Kluwer Academic Publishers, USA, 1987.

[20] Ahmet Yilmaz and Zafer Ozer. Pitch angle control in wind turbines above the rated wind speed by multi-layer perceptron and radial basis function neural networks. *Expert Syst. Appl.*, 36:9767–9775, August 2009.

[21] P Venkatesan and S Anitha. Application of a radial basis function neural network for diagnosis of diabetes mellitus. *CURRENT SCIENCE*, 91(9):5, 2006.

[22] Charles Blundell, Benigno Uria, Alexander Pritzel, Yazhe Li, Avraham Ruderman, Joel Z. Leibo, Jack Rae, Daan Wierstra, and Demis Hassabis. Model-Free Episodic Control. *arXiv:1606.04460 [cs, q-bio, stat]*, June 2016. arXiv: 1606.04460.

[23] Sebastian Thrun and Anton Schwartz. Issues in Using Function Approximation for Reinforcement Learning. page 9, 1993.

[24] Ramanan Sekar, Oleh Rybkin, Kostas Daniilidis, Pieter Abbeel, Danijar Hafner, and Deepak Pathak. Planning to Explore via Self-Supervised World Models. *arXiv:2005.05960 [cs, stat]*, June 2020. arXiv: 2005.05960.

[25] Richard S. Sutton. Integrated Architectures for Learning, Planning, and Reacting Based on Approximating Dynamic Programming. In *In Proceedings of the Seventh International Conference on Machine Learning*, pages 216–224. Morgan Kaufmann, 1990.

[26] Andrew Y. Ng and Stuart J. Russell. Algorithms for Inverse Reinforcement Learning. January 2000.