ASEN 6519 Final Project REPORT

Ben Chupik, Guido Insinger

TOTAL POINTS

(not graded)

QUESTION 1

1 Final Project Report

- 0 pts Correct

David, Goliath and the Dragonball: A DreamerV3 Exploration

Benjamin Chupik
Smead Aerospace
University Of Colorado
Boulder, U.S.A
benjamin.chupik@colorado.edu

Guido Insinger
Smead Aerospace
University of Colorado
Boulder, U.S.A.
guido.insinger@colorado.edu

I. Introduction

For complicated autonomous applications, it is standard practice to use a large autonomous stack consisting of many components that each have their own purpose. Each of these components have to be integrated together, and often tuned so that they work effectively. This integration and tuning requires expert knowledge and ample trial and error, creating time and personnel constraints. One such application is autonomous search and rescue. This is a challenging task that requires exploration, image recognition, motion planning, and control to be integrated together. Each of these modules needs to be tuned for the specific task and integrated together. This creates a complex integration task just to get the autonomy stack working for one application. This paper looks for an "out of the box" solution to the image based search and rescue problem which would allow for non-experts to create autonomous search and rescue systems with the potential to be expanded to many other systems.

To solve the image search and rescue problem in a way that requires little expert knowledge and hand tuning, the standard autonomy stack is disregarded. Instead, a new algorithm's, DreamerV3, capabilities are explored.

II. Background and related work

DreamerV3 [1] is a novel Reinforcement Learning paper that presents an algorithm able to directly learn from visual inputs in a wide range of domains by learning a world model which can predict future states of a 3d environment. Additionally, no hyperparameter tuning is required due to the implementation of symlog reward scaling, allowing direct implementation after choosing the model size.

The first version of the Dreamer algorithm [2] has been previously applied to create a motion planner in pedestrian scenarios [3]. In this paper, a perception layer is added before feeding the data into dreamer and the robot has to reach a pre-specified coordinate. The authors of this paper did also manage to apply their algorithm to a real-life scenario by including LIDAR. Another paper [4] also uses an added perception layer for crowd navigation, in their case before feeding data into SAC [5].

SG-D3QN [6] is an interesting example of using end-to-end training on three separate, but connected, neural networks for crowd navigation. Another article [7] uses a recurrent neural network to pre-process data for crowd navigation before feeding it into PPO [8]. A different approach for crowd navigation [9] was to learn two maps: one to map the environment around the robot and one to track movement of pedestrians.

In addition to exploring the possibilities of DreamerV3, we aim to build on these aforementioned works on autonomous crowd navigation. Crowd navigation is the sole goal in all of these papers. We intend to build on that by making the goal location unknown to the agent which adds image recognition and exploration to the problem.

Besides this, we also attempt to do crowd navigation without adding a custom perception layer. In theory, the world model in DreamerV3 should be able to take over the function of perception layers that have been employed in previous works.

III. Problem formulation

To both explore DreamerV3 and see its effectiveness at search and rescue, multiple test problems are needed. Overall each problem will be discrete in time, and have an action space, state space, reward function, observation space, and a way to propagate dynamics.

A. 2D Pendulum

The 2D Pendulum problem is a pendulum in 2D space hanging down under the effects of gravity. It is moved by applying a torque, and the goal is to get the pendulum upright with the smallest torques possible. More formally, the state space is $[\theta]$, the angle of the pendulum. The action space is $[\tau]$, the torque on the pendulum. The reward function is 1. The observation space is either a vector or image, both are looked at in this paper. The vector is $[x_p, y_p, \omega]$ where x_p and y_p are the x and y location of the end of the pendulum respectively, and ω is the angular velocity of the pendulum. The image observation space is a $64 \times 64 \times 3$ matrix with integer entries in the interval [0,255], which represents a 64×64 image with 3 colors RGB. The termination conditions are 200 steps elapsing. For more info, see gym's pendulum environment documentation [10].

$$r = -(\theta^2 + 0.1 * \dot{\theta}^2 + 0.001 * \tau^2) \tag{1}$$

B. 3D Search and Rescue

The 3D Search and Rescue problem has the general goal of finding a red ball that is randomly placed in a 3D environment with obstacles. There are 2 unique environments created for this paper, the Static Obstacle and Moving Pedestrian environments. These environments share the same agent, observation space, action space, reward function, and state space, all of which will be introduced first. Domain-specific hyperparameters as well as randomization to ensure a robust solution will be discussed in the introduction of each domain.

Both 3D environments are implemented in Webots [11], and the agent that is used in both environments is the standard Webots e-puck robot. The e-puck robot has a camera attached to the front of the robot and is differential drive.

The observation space for these environments is, just like for the 2D Pendulum image case, a $64 \times 64 \times 3$ matrix with integer entries in the interval $[0, 255]$ representing a 64×64 RGB image that is taken from the e-puck camera.

The action space is a simple vector of size 2 with entries in the continuous interval $[-2\pi, 2\pi]$, where each entry corresponds to a velocity input to one of the wheels of the e-puck.

The reward function used for both Search and Rescue environments is given in Equation 2 and consists of an obstacle avoidance term R_o, goal finding term R_g, shaping potential function F and constant term ν. This constant term gives a small negative reward for each step, rewarding episodes that terminate early.

$$R(s, s') = R_o(s') + \kappa * R_g(s') + \rho * F(s, s') - \nu \quad (2)$$

The state vector in these environments is defined in Equation 3.

$$s = (\vec{p}_a, \vec{p}_g, \vec{P}_o) \quad (3)$$

This state vector is a concatenation of the agent position $\vec{p}_a$, goal position $\vec{p}_g$ and obstacle vector $\vec{P}_o$.

The number of obstacles in these environments will be denoted as n, after which we can define the obstacle vector $\vec{P}_o$ itself to be a concatenation of the all of the n individual position vectors p_{o_i} as shown in Equation 4.

$$\vec{P}_o = (\vec{p}_{o_1}, \vec{p}_{o_2}, ..., \vec{p}_{o_n}) \quad (4)$$

Equation 5 shows how these obstacles show up in the reward function. Effectively, R_o returns -1 when the agent is within ϵ_o of any obstacle and 0 otherwise. It should be noted that all distance functions $d(\vec{p}, \vec{p})$ in this paper refer to the euclidean x, y distance between position vectors.

$$R_o(s) = \begin{cases} -1 & \text{if } \exists \vec{p}_{o_i} \in \vec{P}_o : d(\vec{p}_a, \vec{p}_{o_i}) < \epsilon_o \\ 0 & \text{otherwise} \end{cases} \quad (5)$$

If the agent is within range ϵ_g of the goal, Equation 6 sets the value of R_g to 1 and the episode is terminated. Again, in all other cases 0 is returned.

$$R_g(s) = \begin{cases} 1 & \text{if } d(\vec{p}_a, \vec{p}_g) < \epsilon_g \\ 0 & \text{otherwise} \end{cases} \quad (6)$$

Finally, a potential-based shaping function is added. The addition of a shaping function in this form has been proven [12] to not affect the optimal policy of the problem. In this equation, $\gamma = 0.997$ which is taken from DreamerV3 [1].

$$F(s, s') = \gamma \Phi(s') - \Phi(s) \quad (7)$$

Φ is chosen as Equation 8 in order to incentivize movement towards the goal.

$$\Phi(s) = -d(\vec{p}_a, \vec{p}_g) \quad (8)$$

For both environments, the same reward hyperparameters were used as given in Equation 9

$$\begin{cases} \kappa = 100 \\ \rho = 20 \\ \nu = 0.2 \end{cases} \quad (9)$$

1) Static obstacles: Figure 1 shows the static obstacle environment. In this environment the obstacles are always in the same position. At the start of an episode, the goal is randomly put in one of the corners and the agent starts in the middle with a random yaw.

Fig. 1. Static obstacle environment. The domain-specific parameters are $\epsilon_g = 0.15$ and $\epsilon_o = 0.08$. Episodes are limited to $T = 60s$ and a timestep of $dt = 0.2s$ is used.

2) Pedestrian obstacles: In the pedestrian obstacle environment the agent has to avoid moving pedestrians instead of static blocks. Again, at the start of an episode the goal location is randomized to one of four possible locations as shown in Figure 2. In addition to a randomized goal location, the pedestrians are also given random straight line trajectories within the shown zones. Pedestrians always start on opposite sides of the environment.

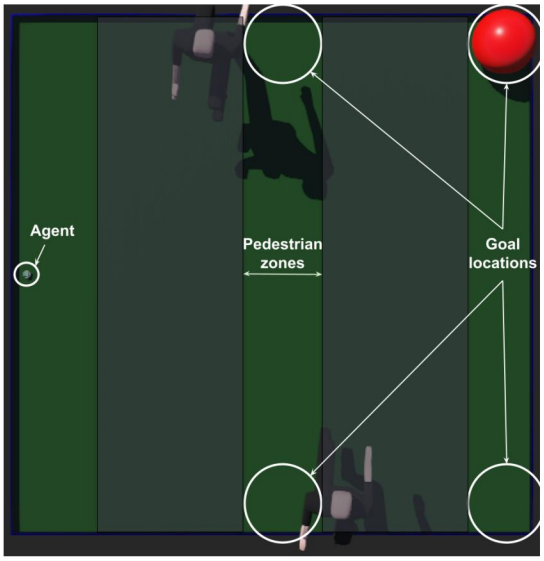

Fig. 2. Pedestrian obstacle environment. The domain-specific parameters are $\epsilon_g = 0.5$ and $\epsilon_o = 0.75$. Episodes are limited to $T = 180s$ and a timestep of $dt = 0.2s$ is used.

IV. Solution Approach

We chose to use the official DreamerV3 codebase https://github.com/danijar/dreamerv3 to solve all of our environments.

A. DreamerV3-WeBots Integration

To the best of our knowledge, DreamerV3 has not been integrated with Webots before. Due to this novelty, in addition to the lack documentation for the DreamerV3 codebase, the integration turned out to be quite a challenge.

Figure 3 shows an overview our final integration.

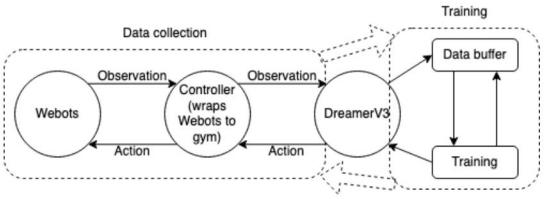

Fig. 3. Overview of our integration of DreamerV3 in Webots. Code is available on https://github.com/Benjamin-Chupik/dreamerv3_webots

This integration is coded as a Webots controller, in which the DreamerV3 training process controls controls the Webots simulation.

When collecting data, Webots simulates the environment based on the actions chosen by DreamerV3, after which the observations from Webots are passed back to DreamerV3. Once enough data is collected to train, the simulation pauses while DreamerV3 updates its parameters. After parameters are updated, the simulation and concurrently data collection resumes. The communication between DreamerV3 and Webots is facilitated through our own custom wrapper. It encodes the observations from Webots into a gym environment which DreamerV3 interacts with and also decodes actions chosen by DreamerV3 back into Webots inputs.

B. DreamerV3 Parameters

DreamerV3 only has a couple tunable parameters, mainly the model size, train to run ratio, and the batch size. For all the experiments use a 'large' model that has 77 million parameters. For the vector observation space a batch size of 16 is used, however this takes too much memory for the image observation space so a batch of 8 is used. The train to run ratio is always 1:64 meaning that there are 64 interactions with the environment for every training step. These parameters seem standard for image processing and similar to what is done in the DreamerV3 paper.

C. Computational Setup

Everything was trained on an NVIDIA GeForce RTX 3070 Ti. This is a reasonably priced ($600) graphics card with 8GB ram.

V. Results

Every experiment was trained on the prior computational setup. The training times are listed in Table I.

Experiment	Training Time (min)	Episodes
2D Pendulum Vector	25	50k
2D Pendulum Image	40	50k
3D Static Obstacle	80	65k
3D Pedestrian	120	85k

TABLE I
Training Statistics

A. 2D Pendulum

The 2D Pendulum Vector and Image were trained for 50k episodes. The resulting rewards over time are shown in Figure 4. Though this is only one trial, the learning rate for the vector is the same as the image. They both have the environment learned and solved at 20K episodes. During training, the frames processed per second (FPS) for the vector environment was 120 FPS and the image environment was 35 FPS. This is partly due to the computation overhead rendering images causes, but some is assumed to be from the algorithm taking longer to train.

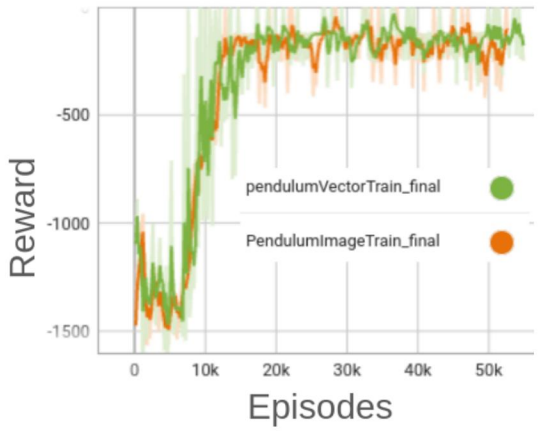

Fig. 4. Pendulum Environment Rewards

As introduced before, DreamerV3 created a world model that can predict the environment. This world model is trained thought the process and gets better with more training. Looking at this world model's predictions can give good insight on what the NN thinks is important and help understand its actions. Figure 5 shows the world model's evolution in the pendulum environment in 6 different example states (the columns). Like introduced in the background section, the world model is given 5 frames and then predicts 45 frames. This prediction is shown in the second row, with the top row being the actual environment and the bottom being the difference. This will be the same for all of the following figures. This figure shows one of the predicted frames around frame 30 and compared to the actual environment. With almost no training time, the world model only predicts static, while after training it can accurately predict the environment and re-render the image.

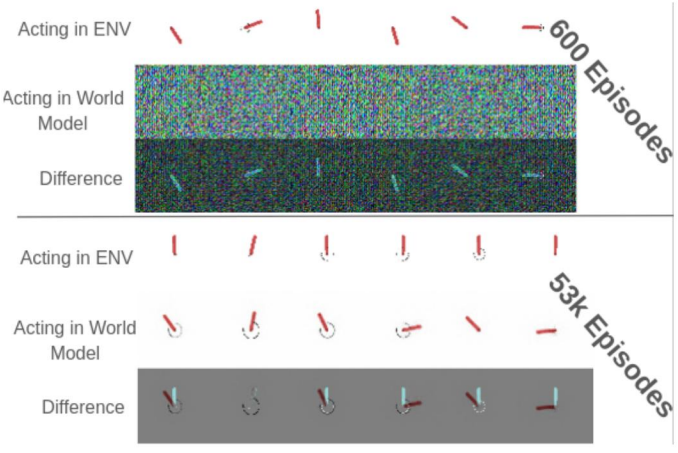

Fig. 5. Pendulum Image Environment World Model Evolution

B. 3D Static Obstacles

The 3D static obstacles environment only trained for 65k episodes. The reward curve shown in Figure 6 shows that the environment is almost solved, but there are still dips in the reward. If trained for longer, the rewards should flatten out more. Also, the reward curve shows large variability at the beginning, which flattens out. This is due to Dreamer exploring the environment and not having a solid world model. The world model evolution throughout training can be seen in Figure 7 where it initially can't predict the environment. Then around 11k episodes it understands that walls exist and starts to predict how the obstacle behave when the robot moves. Then at the end it predicts that the objective ball is in corners, and navigates to those corners. This shows that Dreamer has successfully understood that the balls show up in any of the 4 corners of the environment and understands how to navigate to them.

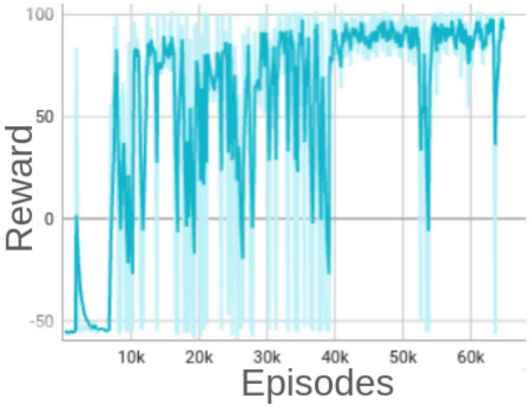

Fig. 6. Static Obstacles Training Rewards

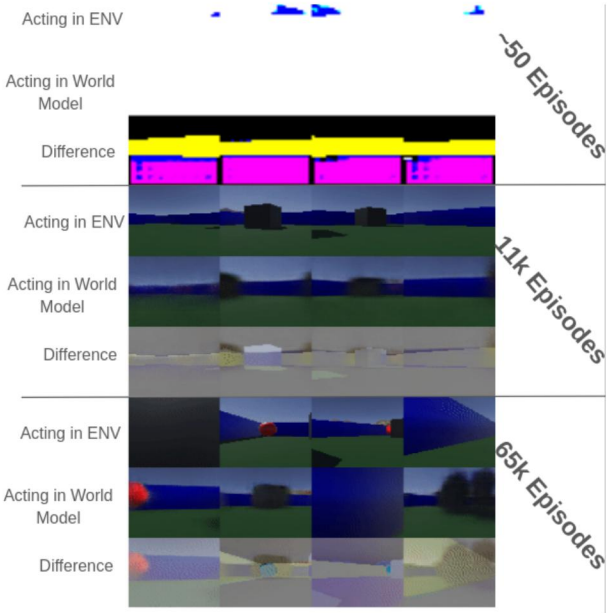

Fig. 7. Static Obstacles World Model

C. 3D Pedestrians

The 3D Pedestrians environment was only trained for 85K episodes due to time constraints. Though the problem is not solved, the results are significant. First, the training curve in

Figure 8 show's a large reward variation until episode 35k when the variation decreases. This is thought to be because Dreamer is still learning the basics of the environment in its world model. Once it learns the basics, it then has to learn the advanced components and the best policy. Looking at Figure 9 taken at the end of training, the world model understands the basics of the environment like walls, how the robot moves, the reward locations, and that pedestrians exist. A harder component to see in a photo is that Dreamer does not understand the behaviors of the pedestrians. In the second column, the pedestrian disappears in a couple frames. This means that Dreamer does not understand how the pedestrians move, and thus struggles to avoid them consistantly. With more training time, Dreamer should accurately predict the pedestrian movement.

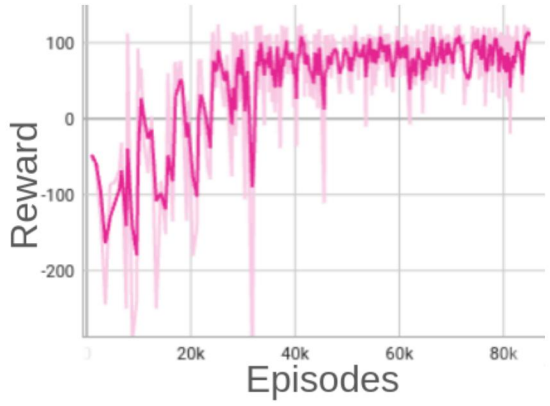

Fig. 8. Pedestrians Training Rewards

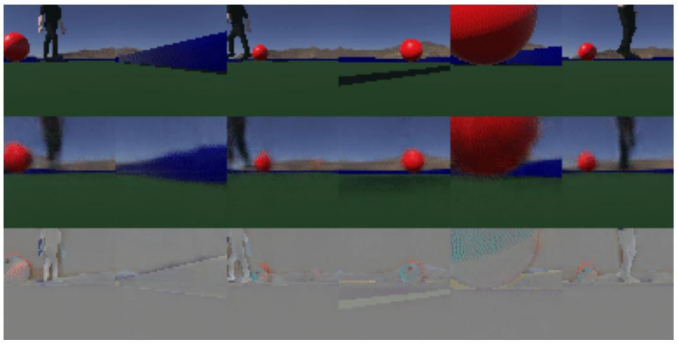

Fig. 9. Pedestrians World Model 85k

VI. Conclusion and Future Work

This paper shows that Dreamerv3 can successfully solve an image search and rescue problem without any additional layers. To show this, DreamerV3 was trained on 3D environments with both static and moving obstacles. Though the training time was limited, the results shows DreamerV3 learning components of the environments and acting in intelligent ways to get the highest reward. This means that given more training time, it is reasonable to assume that DreamerV3 would be able to solve complex real world search and rescue tasks without a lot of expert knowledge and tuning.

The paper also shows how the world model can be used to gauge the progress and understanding of the model. In all the results, there is a significant jump and stability in the rewards once the world model learns the important environmental details. This can be used to understand why a model is underperforming by looking at what the model thinks is important.

Future work that needs to be done is to test the pedestrian environment with longer training time. This will hopefully show more improvement to the world model and the rewards. Also, the behavior with 6 degrees of freedom needs to be explored for any flying application. We predict that the world model will need even more time to understand the environment, even though it is still 3D. We would also like to look at integrating more sensors into the observation layer. Currently, the observation layer is lacking direct depth information ,and we think that this may speed up training and improve the policy.

VII. Contributions and Release

The authors grant permission for this report to be posted publicly

Author Chupik integrated the DreamerV3 code into the codebase and into the 2D environments (modifying the environments for image observation spaces). They also assisted with the integration to WeBots. Additionally, they ran and compiled the results. Both authors wrote the report.

Author Insinger assisted with the integration of the DreamerV3 code into the codebase. They also created and integrated the WeBots test environments into the codebase, creating controllers for the robots and the pedestrians. Both authors wrote the report.

References

[1] D. Hafner, J. Pasukonis, J. Ba, and T. Lillicrap, "Mastering diverse domains through world models," 2023.

[2] D. Hafner, T. Lillicrap, J. Ba, and M. Norouzi, "Dream to control: Learning behaviors by latent imagination," 2020.

[3] W. Zhu and M. Hayashibe, "Autonomous navigation system in pedestrian scenarios using a dreamer-based motion planner," *IEEE Robotics and Automation Letters*, vol. 8, no. 6, pp. 3836–3843, 2023.

[4] T. M. Luu, T. Vu, T. Nguyen, and C. D. Yoo, "Visual pretraining via contrastive predictive model for pixel-based reinforcement learning," *Sensors*, vol. 22, no. 17, 2022.

[5] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," 2018.

[6] Z. Zhou, P. Zhu, Z. Zeng, J. Xiao, H. Lu, and Z. Zhou, "Robot navigation in a crowd by integrating deep reinforcement learning and online planning," *Applied Intelligence*, vol. 52, no. 13, pp. 15600–15616, 2022.

[7] S. Liu, P. Chang, Z. Huang, N. Chakraborty, K. Hong, W. Liang, D. L. McPherson, J. Geng, and K. Driggs-Campbell, "Intention aware robot crowd navigation with attention-based interaction graph," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 12015–12021, 2023.

[8] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017.

[9] S. Yao, G. Chen, Q. Qiu, J. Ma, X. Chen, and J. Ji, "Crowd-aware robot navigation for pedestrians with multiple collision avoidance strategies via map-based deep reinforcement learning," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 8144–8150, 2021.

[10] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.

[11] Webots, "http://www.cyberbotics.com." Open-source Mobile Robot Simulation Software.

[12] A. Y. Ng, D. Harada, and S. J. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping," in *Proceedings of the Sixteenth International Conference on Machine Learning*, ICML '99, (San Francisco, CA, USA), p. 278–287, Morgan Kaufmann Publishers Inc., 1999.

1 Final Project Report

- 0 pts Correct