Supplementary Materials

Appendix A. Full Proofs

A.1. Information Gain via Random Features

Poof of Proposition 2 Using the random feature approximation $\mathbf{K}_n \approx \mathbf{\Phi}_n\mathbf{\Phi}_n^T$, the posterior variance becomes

$$\sigma_n^2(x_*) = \boldsymbol{\phi}(x_*)^T\boldsymbol{\phi}(x_*) - \boldsymbol{\phi}(x_*)^T(\mathbf{\Phi}_n\mathbf{\Phi}_n^T + \sigma^2\mathbf{I}_n)^{-1}\boldsymbol{\phi}(x_*)$$

Applying the Woodbury identity:

$$(\mathbf{\Phi}_n\mathbf{\Phi}_n^T + \sigma^2\mathbf{I}_n)^{-1} = \frac{1}{\sigma^2}\mathbf{I}_n - \frac{1}{(\sigma^2)^2}\mathbf{\Phi}_n(\mathbf{\Phi}_n^T\mathbf{\Phi}_n + \sigma^2\mathbf{I}_D)^{-1}\mathbf{\Phi}_n^T$$

Substituting and simplifying:

$$\sigma_n^2(x_*) = \boldsymbol{\phi}(x_*)^T\boldsymbol{\phi}(x_*) - \frac{1}{\sigma^2}\boldsymbol{\phi}(x_*)^T\boldsymbol{\phi}(x_*) + \frac{1}{\sigma^2}\boldsymbol{\phi}(x_*)^T(\mathbf{\Phi}_n^T\mathbf{\Phi}_n + \sigma^2\mathbf{I}_D)^{-1}\boldsymbol{\phi}(x_*) \tag{12}$$

$$= \sigma^2\boldsymbol{\phi}(x_*)^T(\mathbf{\Phi}_n^T\mathbf{\Phi}_n + \sigma^2\mathbf{I}_D)^{-1}\boldsymbol{\phi}(x_*) \tag{13}$$

$$\frac{\sigma_n^2(x_*)}{\sigma^2} = \boldsymbol{\phi}(x_*)^T(\mathbf{\Phi}_n^T\mathbf{\Phi}_n + \sigma^2\mathbf{I}_D)^{-1}\boldsymbol{\phi}(x_*) \tag{14}$$

Substituting back into the IG formula of GP yields $\frac{1}{2}\log\left(1 + \boldsymbol{\phi}(x_*)^T(\mathbf{\Phi}_n^T\mathbf{\Phi}_n + \sigma^2\mathbf{I}_D)^{-1}\boldsymbol{\phi}(x_*)\right)$. We can finally reinterpret the observation noise variance σ^2 as a regularization parameter λ, giving the desired result. ■

A.2. Posterior Variance Error Bound

Proof of Proposition 6 Let consider the true posterior variance, $\sigma_n^2(x) = k(x,x) - \mathbf{k}^\top\mathbf{K}^{-1}\mathbf{k}$, with $\mathbf{k} = \mathbf{k}_n(x)$ and $\mathbf{K} = \mathbf{K}_n + \lambda I_n$, considering the approximation $k(x,x') \approx \boldsymbol{\phi}(x)^T\boldsymbol{\phi}(x')$, we can consider our approximated posterior variance $\hat{\sigma}_n^2(x)$ as a perturbation of the true one and define $\hat{\mathbf{k}} = \mathbf{k} + \mathbf{\Delta_k}$ and $\hat{\mathbf{K}} = \mathbf{K} + \mathbf{\Delta_K}$. Let us expand the following difference

$$\hat{\mathbf{k}}^\top\hat{\mathbf{K}}^{-1}\hat{\mathbf{k}} - \mathbf{k}^\top\mathbf{K}^{-1}\mathbf{k} = (\mathbf{k}+\mathbf{\Delta_k})^\top(\mathbf{K}+\mathbf{\Delta_K})^{-1}(\mathbf{k}+\mathbf{\Delta_k}) - \mathbf{k}^\top\mathbf{K}^{-1}\mathbf{k} \tag{15}$$

$$= \mathbf{k}^\top\hat{\mathbf{K}}^{-1}\mathbf{k} + \mathbf{k}^\top\hat{\mathbf{K}}^{-1}\mathbf{\Delta_k} + \mathbf{\Delta_k}^\top\hat{\mathbf{K}}^{-1}\mathbf{k} + \mathbf{\Delta_k}^\top\hat{\mathbf{K}}^{-1}\mathbf{\Delta_k} - \mathbf{k}^\top\mathbf{K}^{-1}\mathbf{k} \tag{16}$$

$$= \mathbf{k}^\top(\hat{\mathbf{K}}^{-1} - \mathbf{K}^{-1})\mathbf{k} + 2\mathbf{k}^\top\hat{\mathbf{K}}^{-1}\mathbf{\Delta_k} + \mathbf{\Delta_k}^\top\hat{\mathbf{K}}^{-1}\mathbf{\Delta_k} \tag{17}$$

$$\leq |\underbrace{\mathbf{k}^\top(\hat{\mathbf{K}}^{-1} - \mathbf{K}^{-1})\mathbf{k}}_{\text{Matrix perturbation } t_1}| + |\underbrace{2\mathbf{k}^\top\hat{\mathbf{K}}^{-1}\mathbf{\Delta_k}}_{\text{Cross term } t_2}| + |\underbrace{\mathbf{\Delta_k}^\top\hat{\mathbf{K}}^{-1}\mathbf{\Delta_k}}_{\text{Vector term } t_3}|. \tag{18}$$

where we used the symmetry property $\mathbf{k}^\top\hat{\mathbf{K}}^{-1}\mathbf{\Delta_k} = \mathbf{\Delta_k}^\top\hat{\mathbf{K}}^{-1}\mathbf{k}$ and triangle inequality.

Bounding t_1: Since the smallest eigenvalue of $\mathbf{K}^{-1}$ $\hat{\mathbf{K}}^{-1}$ is λ, $\|\mathbf{k}\|_2 \leq \sqrt{n}\kappa$, and using the inverse matrix perurbation bound for $\hat{\mathbf{A}} = \mathbf{A} + \mathbf{E}$, $\|\hat{\mathbf{A}}^{-1} - \mathbf{A}^{-1}\|_2 \leq \|\mathbf{A}^{-1}\|_2 \cdot \|\hat{\mathbf{A}}^{-1}\|_2 \cdot \|\mathbf{E}\|_2$, we have

$$|\mathbf{k}^\top(\hat{\mathbf{K}}^{-1} - \mathbf{K}^{-1})\mathbf{k}| \leq \|\hat{\mathbf{K}}^{-1}\|_2 \cdot \|\mathbf{\Delta_K}\|_2 \cdot \|\mathbf{K}^{-1}\|_2 \cdot \|\mathbf{k}\|_2^2 \leq \frac{\epsilon\kappa^2 n^2}{\lambda^2}. \tag{19}$$

Bounding t_2 and t_3: Similarly to t_1, we can bound the two other terms with

$$|2\mathbf{k}^\top \hat{\mathbf{K}}^{-1}\boldsymbol{\Delta}_\mathbf{k}| \leq \frac{2\epsilon n\kappa}{\lambda}, \qquad |\boldsymbol{\Delta}_\mathbf{k}^\top \hat{\mathbf{K}}^{-1}\boldsymbol{\Delta}_\mathbf{k}| \leq \frac{\epsilon^2 n}{\lambda}. \tag{20}$$

Finally, $|\hat{\sigma}_n^2(x) - \sigma_n^2(x)| \leq \epsilon + \frac{\epsilon\kappa^2 n^2}{\lambda^2} + \frac{2\epsilon n\kappa}{\lambda} + \frac{\epsilon^2 n}{\lambda} = \epsilon + \frac{\epsilon\kappa^2}{\lambda_0^2} + \frac{2\epsilon\kappa}{\lambda_0} + \frac{\epsilon^2}{\lambda_0}$, using Assumption 4. ■

A.3. RFIG Error Bound

To bound RFIG, we can directly use the established bound for posterior variance, by using the following lemma:

Lemma 10 (Shifted Logarithmic Difference bound) *For any $a, b > 0$, we have*

$$|\log(1+a) - \log(1+b)| \leq |a-b| \tag{21}$$

Proof On the interval between a and b, there exists c between a and b such that $\log(1+a) - \log(1+b) = f'(c)(a-b) = \frac{a-b}{1+c}$. Since $\min(a,b) \leq c \leq \max(a,b)$, we have $\frac{1}{1+\max(a,b)} \leq \frac{1}{1+c} \leq \frac{1}{1+\min(a,b)}$. Therefore, $\left|\frac{a-b}{1+c}\right| \leq \frac{|a-b|}{1+\min(a,b)}$, we have $a, b > 0$, which completes the proof. ■

Now, we have all the elements to obtain a deterministic upper bound on RFIG.

Proof of Proposition 7 By applying Lemma X, we have

$$|\hat{\mathrm{IG}}(x|\mathcal{D}_n) - \mathrm{IG}(x|\mathcal{D}_n)| \leq \frac{\boldsymbol{\Delta}_{\sigma_n^2}}{2\lambda} = \frac{\epsilon + \frac{\epsilon\kappa^2}{\lambda_0^2} + \frac{2\epsilon\kappa}{\lambda_0} + \frac{\epsilon^2}{\lambda_0}}{2\lambda} \tag{22}$$

$$= \frac{\epsilon\left[\left(1 + \frac{\kappa}{\lambda_0}\right)^2 + \frac{\epsilon}{\lambda_0}\right]}{2\lambda} \tag{23}$$

$$= \frac{\epsilon\left[\frac{(\lambda_0+\kappa)^2 + \epsilon\lambda_0}{\lambda_0^2}\right]}{2\lambda} \tag{24}$$

$$= \frac{\epsilon(\lambda_0+\kappa)^2 + \epsilon^2\lambda_0}{2n\lambda_0^3} \tag{25}$$

That ends the proof. ■

A.4. High Probability RFIG Bound

Proof of Proposition 9 We aim to find when the information gain error is at least ε:

$$\frac{\epsilon(\lambda_0+\kappa)^2 + \epsilon^2\lambda_0}{2n\lambda_0^3} \geq \varepsilon \tag{26}$$

$$\epsilon(\lambda_0+\kappa)^2 + \epsilon^2\lambda_0 \geq \varepsilon 2n\lambda_0^3 \tag{27}$$

$$\epsilon(\lambda_0+\kappa)^2 + \epsilon^2\lambda_0 - \varepsilon 2n\lambda_0^3 \geq 0 \tag{28}$$

This is a quadratic inequality in ϵ. The quadratic $f(\epsilon) = \lambda_0\epsilon^2 + (\lambda_0 + \kappa)^2\epsilon - 2n\lambda_0^3\varepsilon$ has for root:

$$\epsilon \geq \frac{-(\lambda_0 + \kappa)^2 + \sqrt{(\lambda_0 + \kappa)^4 + 8n\lambda_0^4\varepsilon}}{2\lambda_0}, \tag{29}$$

since $\lambda_0 > 0$, the parabola opens upward. Setting $\kappa = 1$ (Proposition 8), ends the proof. ■

Appendix B. Numerical Experiments

B.1. Newton-Schulz iterations

The Newton-Schulz method provides an iterative approach to matrix inversion that is particularly well-suited for our kernel matrix updates. Due to JAX's compilation and paral-

Algorithm 2: Newton-Schulz Matrix Inversion Update

Input: Previous inverse $\mathbf{X}_{old}$, matrix update $\mathbf{\Phi}_t$, regularization λ
$\mathbf{A} \leftarrow \mathbf{\Phi}_t^T\mathbf{\Phi}_t + \lambda\mathbf{I}$;
$\mathbf{X}_0 \leftarrow \mathbf{X}_{old}$ (warm start);
for $k = 1, 2, \ldots, K$ **do**
  $\mathbf{X}_k \leftarrow \mathbf{X}_{k-1}(2\mathbf{I} - \mathbf{A}\mathbf{X}_{k-1})$;
end
return $\mathbf{X}_K$

lelization constraints, we implement a fixed number of Newton-Schulz iterations ($K = 20$) rather than iterating until convergence. In practice, we observe that 20 iterations provide sufficient accuracy for information gain estimation while maintaining computational efficiency across all experimental environments.

B.2. Hyperparameter Configuration

Table 2 presents the complete hyperparameter configuration used for PPO experiments across all environments. For RND baseline comparisons, we use an embedding size of 256, hidden layer sizes of (256, 256), a bonus learning rate of 1e-4, with ReLU activations, following standard RND implementation practices.

B.3. Milestone Reward Wrapper

We implement a `MilestoneRewardWrapper` that transforms dense reward signals into sparse, milestone-based rewards. This wrapper provides rewards only when the agent reaches specific distance milestones during locomotion, creating challenging exploration scenarios where traditional dense rewards are unavailable. The wrapper operates by tracking the agent's forward displacement from its initial position and providing rewards at fixed distance intervals. Specifically, it:

1. Records the agent's initial position at environment reset
2. Monitors the agent's current position throughout the episode

Table 2: PPO hyperparameters used in all experiments.

Parameter	**Value**
Training Configuration	
Total timesteps	1,000,000
Number of environments	32
Steps per environment	128
Evaluation frequency	24,576
Anneal learning rate	True
PPO Algorithm	
Learning rate	0.0003
Number of epochs	4
Number of minibatches	32
Clip ratio (ϵ)	0.2
Value function coefficient	0.5
Entropy coefficient	0.01
Maximum gradient norm	0.5
GAE & Discounting	
Discount factor (γ)	0.99
GAE lambda (λ)	0.95
Normalization	
Normalize observations	True
Normalize intrinsic rewards	True
Network Architecture	
Activation function	Tanh
Hidden layer sizes	(64, 64)

3. Calculates the total distance traveled as the difference between current and initial positions

4. Awards rewards when the agent crosses predefined distance milestones

The milestone reward r_t at timestep t is computed as:

$$r_t = \begin{cases} \alpha \cdot (m_t - m_{t-1}) & \text{if } m_t > m_{t-1} \\ 0 & \text{otherwise} \end{cases} \tag{30}$$

where $m_t = \lfloor d_t/\delta \rfloor$ represents the current milestone, d_t is the distance traveled, δ is the milestone distance interval, and α is the reward scale factor. The wrapper accepts three key parameters:

- **milestone_distance** ($\delta = 1.0$): Distance interval between consecutive milestones
- **reward_scale** ($\alpha = 1.0$): Scale factor applied to milestone rewards

- **position_fn**: Function extracting agent position from environment state (defaults to x-coordinate of the first body)

This design creates environments where agents receive no immediate feedback for small movements but are rewarded for achieving meaningful locomotion progress, making these tasks particularly challenging for exploration strategies. For reproducibility, we provide the complete implementation of the `MilestoneRewardWrapper`:

Listing 1: MilestoneRewardWrapper Implementation

```
from typing import Callable, Optional
from brax.envs import PipelineEnv, State, Wrapper
import jax
from jax import numpy as jp

class MilestoneRewardWrapper(Wrapper):
    """Wrapper that adds milestone-based rewards to any Brax
       environment.
    This wrapper gives a reward whenever the agent reaches specified
        distance
    milestones (e.g., every 1.0 unit of forward movement).
    """
    def __init__(
            self,
            env: PipelineEnv,
            milestone_distance: float = 1.0,
            reward_scale: float = 1.0,
            position_fn: Optional[Callable[[State], jp.ndarray]] =
               lambda state: state.pipeline_state.x.pos[0, 0],
    ):
        """Initializes the milestone reward wrapper.
        Args:
          env: The environment to wrap.
          milestone_distance: Distance between reward milestones.
          reward_scale: Scale factor for milestone rewards.
          position_fn: Function that extracts position from state.
                       Default extracts x position from first body.
        """
        super().__init__(env)
        self._milestone_distance = milestone_distance
        self._reward_scale = reward_scale
        self._position_fn = position_fn

    def reset(self, rng: jax.Array) -> State:
        """Resets the environment and initializes milestone reward
           tracking."""
        state = self.env.reset(rng)
        # Get initial position
        initial_position = self._position_fn(state)
        # Add milestone reward tracking info
        info = state.info.copy()
```

```
        info.update({
            'initial_position': initial_position,
            'last_milestone': 0.0,
            'total_milestones': 0,
            'distance_traveled': 0.0,
            'current_milestone': 0.0,
        })
        return state.replace(info=info)

    def step(self, state: State, action: jax.Array) -> State:
        """Steps the environment and adds milestone rewards."""
        # Get tracking info
        initial_position = state.info.get('initial_position')
        last_milestone = state.info.get('last_milestone', 0.0)
        total_milestones = state.info.get('total_milestones', 0)

        # Step the environment
        next_state = self.env.step(state, action)

        # Get current position and calculate distance traveled
        current_position = self._position_fn(next_state)
        distance_traveled = current_position - initial_position

        # Calculate the current milestone
        current_milestone = jp.floor(distance_traveled / self.
            _milestone_distance)

        # Check if we've reached a new milestone
        new_milestone_reached = current_milestone > last_milestone

        # Calculate milestone reward
        reward = jp.where(
            new_milestone_reached,
            self._reward_scale * (current_milestone - last_milestone
                ),
            0.0
        )

        # Update the total milestones count
        total_milestones = jp.where(
            new_milestone_reached,
            total_milestones + jp.int32(current_milestone -
                last_milestone),
            total_milestones
        )

        # Update the last milestone
        last_milestone = jp.where(new_milestone_reached,
            current_milestone, last_milestone)

        # Update info
```

```
        info = next_state.info.copy()
        info.update({
            'initial_position': initial_position,
            'last_milestone': last_milestone,
            'total_milestones': total_milestones,
            'distance_traveled': distance_traveled,
            'current_milestone': current_milestone,
        })

        return next_state.replace(reward=reward, info=info)
```