Applying Methods in RL to Character Animation

Unsupervised RL IW Seminar

COS IW: Unsupervised RL Seminar

Contact:
Shivam Kak | 6307700880 | sk3686@princeton.edu

Prof. Benjamin Eysenbach

Topic Agenda

(Approx 9 min.)

- Motivation & Goal
- Problem Background
- Related Work
- Approach
  - AMP Mod 1
  - AMP Mod 2
  - Custom Dataset
- Implementation
  - Code Review
- Results
- Conclusion
  - Future Work

Contact:
Shivam Kak | 6307700880 | sk3686@princeton.edu

Prof. Benjamin Eysenbach

Motivation & Goal

Motivation & Goal

- Generation of assets for animated movies / games takes a lot of work! Involves creation of:
  - 3D Mesh
  - Textures
  - Rigging
  - Animation of Rigging
- Prior methods in using motion tracking to generate animation sequences have proven to be data-intensive and expensive
- Unsupervised RL is well suited to learning distributional characteristics of certain motions
- Personal Interest

Contact:
Shivam Kak | 6307700880 | sk3686@princeton.edu

Prof. Benjamin Eysenbach

Goal

"Can we reduce pose error for Adversarial Motion Planning (AMP) agents for specific tasks (walking) by using custom training data or applying modifications to the base algorithm?

Problem & Background

Diving into AMP

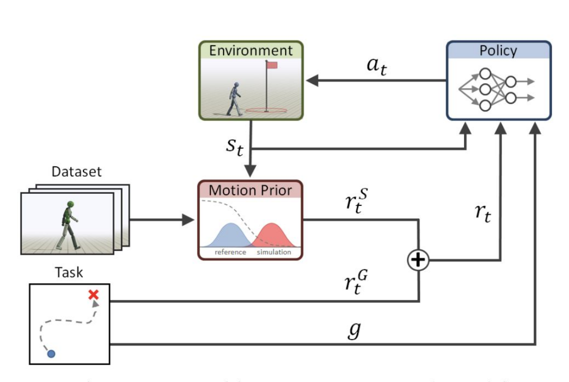
- Enables physics-based characters to learn natural movements from unstructured motion data without manual reward engineering or explicit motion planning
- Uses adversarial training to develop a motion prior that guides character behavior to match reference motion styles
- Combines task objectives with learned motion priors

- **Learns motion prior D(st, st+1) as discriminator trained to differentiate between real motion transitions from dataset vs those generated by policy π**

The discriminator is trained by solving a least-squares regression problem to predict a score of 1 for samples from the dataset and −1 for samples recorded from the policy. The reward function for training the policy is then given by

$$r(s_t, s_{t+1}) = \max\left[0,\ \ 1 - 0.25(D(s_t, s_{t+1}) - 1)^2\right]. \tag{7}$$

The additional offset, scaling, and clipping are applied to bound the reward between [0, 1], as is common practice in previous RL frameworks [Peng et al. 2018a, 2016; Tassa et al. 2018]. Mao et al. [2017] showed that this least-squares objective minimizes the Pearson χ^2 divergence between $d^{\mathcal{M}}(s, s')$ and $d^{\pi}(s, s')$.



Contact:
Shivam Kak | 6307700880 | sk3686@princeton.edu

Prof. Benjamin Eysenbach

Diving into AMP

ALGORITHM 1: Training with AMP

```
input M: dataset of reference motions
D ← initialize discriminator
π ← initialize policy
V ← initialize value function
B ← ∅ initialize reply buffer

while not done do
    for trajectory i = 1, ..., m do
        τ^i ← {(s_t, a_t, r_t^G)_{t=0}^{T-1}, s_T^G, g} collect trajectory with π
        for time step t = 0, ..., T − 1 do
            d_t ← D(Φ(s_t), Φ(s_{t+1}))
            r_t^S ← calculate style reward according to Equation 7 using d_t
            r_t ← w^G r_t^G + w^S r_t^S
            record r_t in τ^i
        end for
        store τ^i in B
    end for

    for update step = 1, ..., n do
        b^M ← sample batch of K transitions {(s_j, s'_j)}_{j=1}^K from M
        b^π ← sample batch of K transitions {(s_j, s'_j)}_{j=1}^K from B
        update D according to Equation 8 using b^M and b^π
    end for

    update V and π using data from trajectories {τ^i}_{i=1}^m
end while
```

and 7 illustrate examples of motions learned by the characters. Performance is evaluated using the average pose error, where the pose error e_t^{pose} at each time step t is computed between the pose of the simulated character and the reference motion using the relative positions of each joint with respect to the root (in units of meters),

$$e_t^{\text{pose}} = \frac{1}{N^{\text{joint}}} \sum_{j \in \text{joints}} \left\| (\mathrm{x}_t^j - \mathrm{x}_t^{\text{root}}) - (\hat{\mathrm{x}}_t^j - \hat{\mathrm{x}}_t^{\text{root}}) \right\|_2 . \quad (10)$$

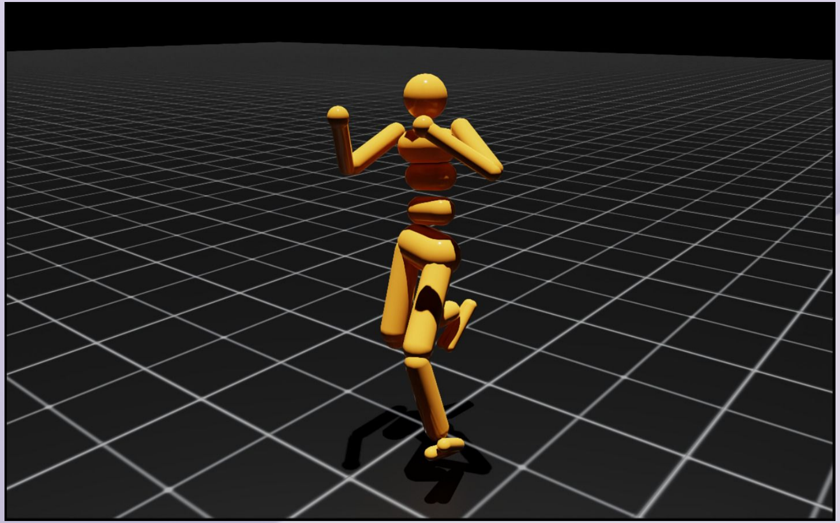
x_t^j and $\hat{\mathrm{x}}_t^j$ denote the 3D Cartesian position of joint j from the simulated character and the reference motion, and N^{joint} is the total number of joints in the character's body. This method of evaluating

Contact:
Shivam Kak | 6307700880 | sk3686@princeton.edu

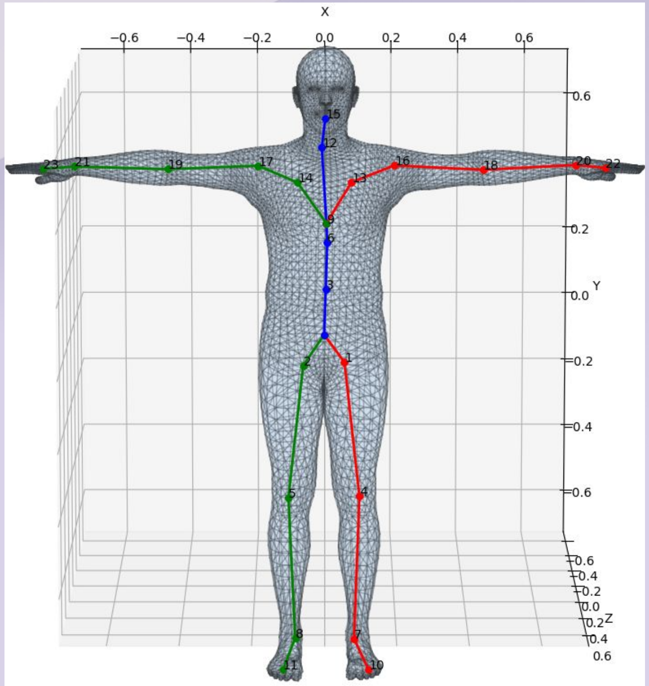
Related Work

Related Work

IsaacSim



HumanML3D



Contact:
Shivam Kak | 6307700880 | sk3686@princeton.edu

Prof. Benjamin Eysenbach

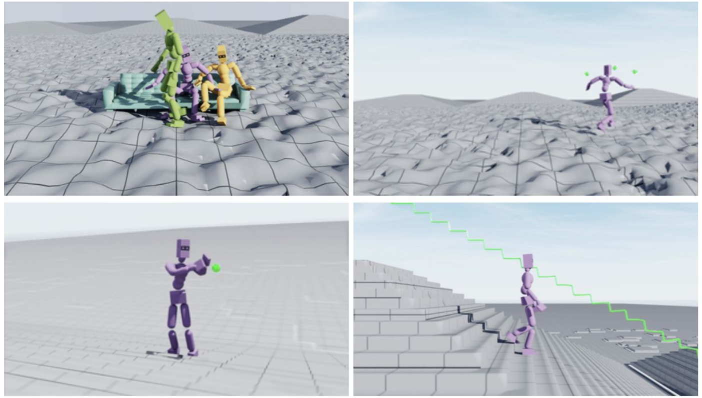
Related Work

ProtoMotions: Physics-based Character Animation

"Primitive or fundamental types of movement that serve as a basis for more complex motions."

What is this?

This codebase contains our efforts in building interactive physically-simulated virtual agents. It supports both IsaacGym and IsaacSim.



Contact:
Shivam Kak | 6307700880 | sk3686@princeton.edu

Prof. Benjamin Eysenbach

Approach

Approach

Algorithm 1 Training with Adaptive AMP

Input: $\mathcal{M}$: dataset of reference motions
$\alpha \leftarrow 0.5$ {initialize adaptive weight}
$D \leftarrow$ initialize discriminator
$\pi \leftarrow$ initialize policy
$V \leftarrow$ initialize value function
$\mathcal{B} \leftarrow \emptyset$ {initialize reply buffer}
while not done **do**
  for trajectory $i = 1, \ldots, m$ **do**
    $\tau^i \leftarrow \{(s_t, a_t, r_t^G)_{t=0}^{T-1}, s_T^G, g\}$ {collect trajectory with π}
    for time step $t = 0, \ldots, T-1$ **do**
      $d_t \leftarrow D(\Phi(s_t), \Phi(s_{t+1}))$
      $r_t^S \leftarrow$ calculate style reward according to Equation 7 using d_t
      $r_t \leftarrow (1-\alpha)w^G r_t^G + \alpha w^S r_t^S$ {adaptive weighting}
      record r_t in τ^i
    end for
    store τ^i in $\mathcal{B}$
  end for
  for update step $= 1, \ldots, n$ **do**
    $b^{\mathcal{M}} \leftarrow$ sample batch of K transitions $\{(s_j, s'_j)\}_{j=1}^K$ from $\mathcal{M}$
    $b^{\pi} \leftarrow$ sample batch of K transitions $\{(s_j, s'_j)\}_{j=1}^K$ from $\mathcal{B}$
    update D according to Equation 8 using $b^{\mathcal{M}}$ and b^{π}
  end for
  update V and π using data from trajectories $\{\tau^i\}_{i=1}^m$
  $\alpha \leftarrow$ update adaptive weight based on relative losses
end while

Algorithm 2 Training with Hierarchical AMP

Input: $\mathcal{M}$: dataset of reference motions
$D_L, D_H \leftarrow$ initialize local and global discriminators
$\pi \leftarrow$ initialize policy
$V \leftarrow$ initialize value function
$\mathcal{B} \leftarrow \emptyset$ {initialize reply buffer}
while not done **do**
  for trajectory $i = 1, \ldots, m$ **do**
    $\tau^i \leftarrow \{(s_t, a_t, r_t^G)_{t=0}^{T-1}, s_T^G, g\}$ {collect trajectory with π}
    for time step $t = 0, \ldots, T-1$ **do**
      $d_t^L \leftarrow D_L(\Phi_L(s_t), \Phi_L(s_{t+1}))$ {local features}
      $d_t^H \leftarrow D_H(\Phi_H(s_{t:t+k}))$ {global features}
      $r_t^S \leftarrow$ calculate combined style reward using d_t^L, d_t^H
      $r_t \leftarrow w^G r_t^G + w^S r_t^S$
      record r_t in τ^i
    end for
    store τ^i in $\mathcal{B}$
  end for
  for update step $= 1, \ldots, n$ **do**
    $b^{\mathcal{M}} \leftarrow$ sample batch of K transitions $\{(s_j, s'_j)\}_{j=1}^K$ from $\mathcal{M}$
    $b^{\pi} \leftarrow$ sample batch of K transitions $\{(s_j, s'_j)\}_{j=1}^K$ from $\mathcal{B}$
    update D_L, D_H according to Equation 8 using $b^{\mathcal{M}}$ and b^{π}
  end for
  update V and π using data from trajectories $\{\tau^i\}_{i=1}^m$
end while

Contact:
Shivam Kak | 6307700880 | sk3686@princeton.edu

Prof. Benjamin Eysenbach

Approach

- AMASS dataset is comprised of the following subsets:
    - ACCAD, BMLhandball, BMLmovi, BMLrub, CMU, CNRS, DanceDB, DFaust, EKUT, EyesJapanDataset, GRAB, HDM05, HUMAN4D, HumanEva, KIT, MoSh, PosePrior, SFU, SOMA, SSM, TCDHands, TotalCapture, Transitions, WEIZMANN

- Selected Subset: HumanEva Dataset
    - http://humaneva.is.tue.mpg.de/
    - Focuses on basic human actions
    - Walking, jogging, gesturing
    - Reasoning: try to avoid overfitting for the simple task of walking humanoid

Contact:
Shivam Kak | 6307700880 | sk3686@princeton.edu

Prof. Benjamin Eysenbach

Implementation

Implementation (Algo 1)

```python
def calculate_extra_reward(self):
    # Get task reward from parent class
    task_rew = super().calculate_extra_reward()

    if self.disable_discriminator:
        return task_rew

    # Get style reward
    discriminator_obs = self.experience_buffer.discriminator_obs
    style_rew = self.calculate_discriminator_reward(
        discriminator_obs.view(self.num_envs * self.num_steps, -1)
    ).view(self.num_steps, self.num_envs)

    # Update adaptation tracking
    self.style_reward_history.append(style_rew.mean().item())
    self.task_reward_history.append(task_rew.mean().item())

    # Adaptive weighting
    combined_reward = (1 - self.alpha) * task_rew + self.alpha * style_rew
    return combined_reward
```

New

```python
def calculate_extra_reward(self):
    rew = super().calculate_extra_reward()

    if self.disable_discriminator:
        return rew

    discriminator_obs = self.experience_buffer.discriminator_obs
    disc_r = self.calculate_discriminator_reward(
        discriminator_obs.view(self.num_envs * self.num_steps, -1)
    ).view(self.num_steps, self.num_envs)

    self.experience_buffer.batch_update_data("discriminator_rewards", disc_r)

    extra_reward = disc_r + rew
    return extra_reward
```

Old

Contact:
Shivam Kak | 6307700880 | sk3686@princeton.edu

Prof. Benjamin Eysenbach

Implementation (Algorithm 2)

Old

```python
def discriminator_forward(self, obs: Tensor, return_norm_obs=False) -> Tensor:
    args = {"obs": obs}
    return self.discriminator(args, return_norm_obs=return_norm_obs)
```

```python
def discriminator_forward(self, obs: Tensor, return_norm_obs=False) -> Tensor:
    # Split into local and global features
    local_features = obs[:, :self.local_feature_dim]
    global_features = obs  # Full sequence

    # Get predictions from both discriminators
    local_pred = self.local_discriminator({"obs": local_features},
                                          return_norm_obs=return_norm_obs)
    global_pred = self.global_discriminator({"obs": global_features},
                                            return_norm_obs=return_norm_obs)

    # Combine predictions
    combined_pred = 0.5 * local_pred + 0.5 * global_pred

    if return_norm_obs:
        return {
            "outs": combined_pred,
            "norm_obs": obs
        }
    return combined_pred
```

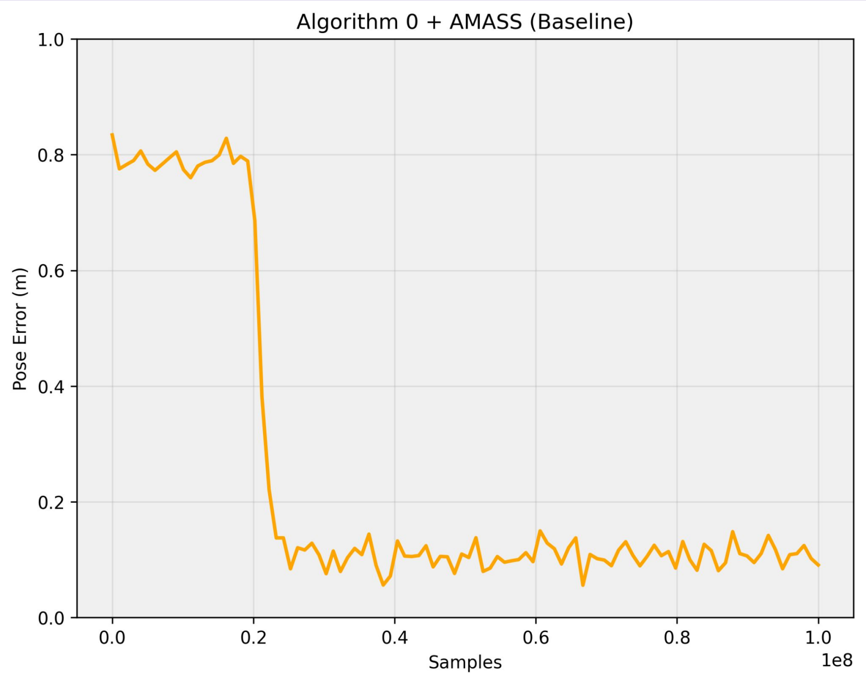
New

Contact:
Shivam Kak | 6307700880 | sk3686@princeton.edu

Prof. Benjamin Eysenbach

Results

Results

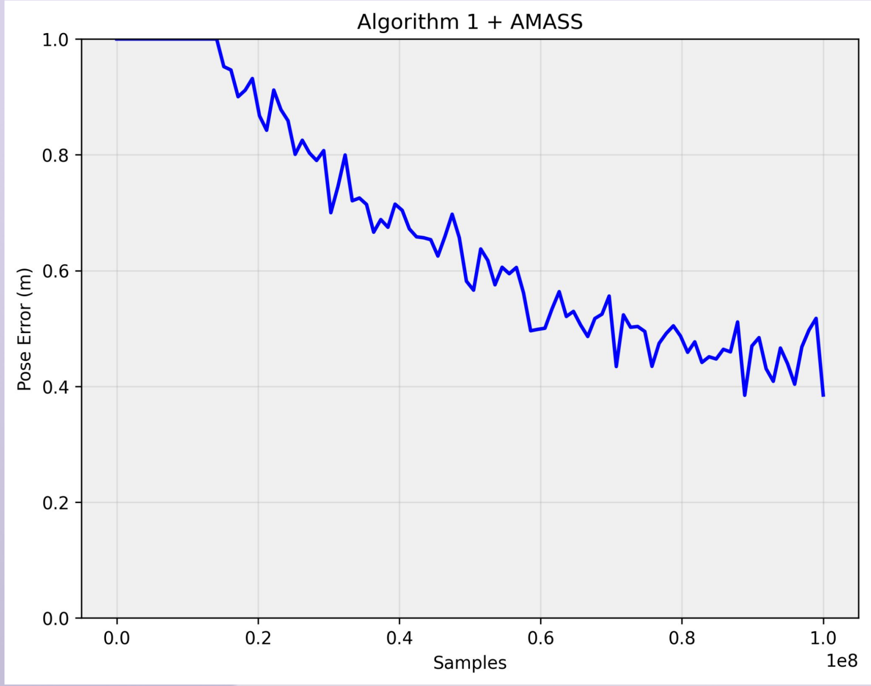


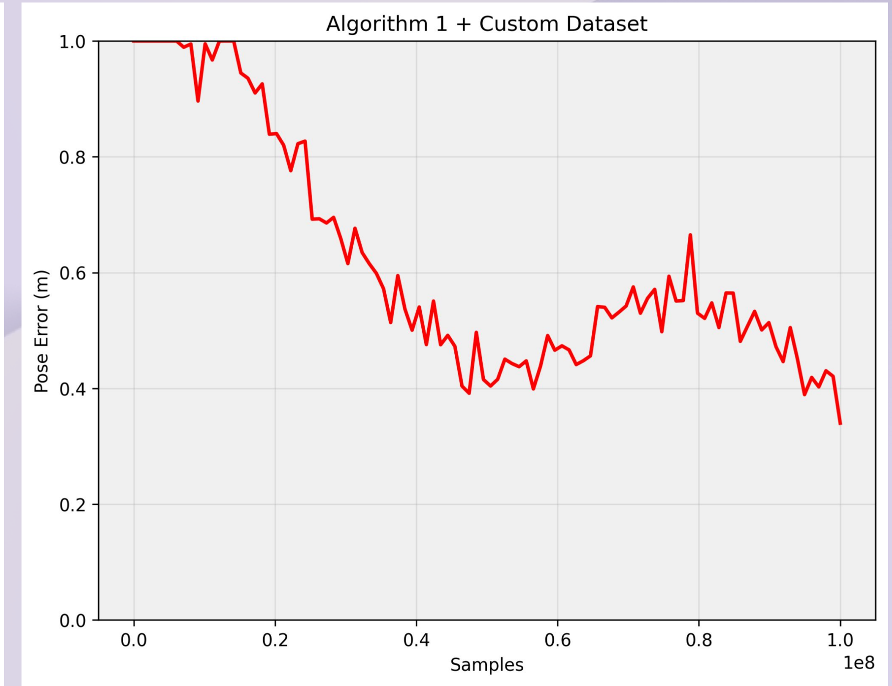
$$e_t^{\text{pose}} = \frac{1}{N^{\text{joint}}} \sum_{j \in \text{joints}} \left\| (\mathrm{x}_t^j - \mathrm{x}_t^{\text{root}}) - (\hat{\mathrm{x}}_t^j - \hat{\mathrm{x}}_t^{\text{root}}) \right\|_2 .$$

(Pose Error compared to motion file provided in ProtoMotions repo for basic humanoid walking)

Contact:
Shivam Kak | 6307700880 | sk3686@princeton.edu

Results

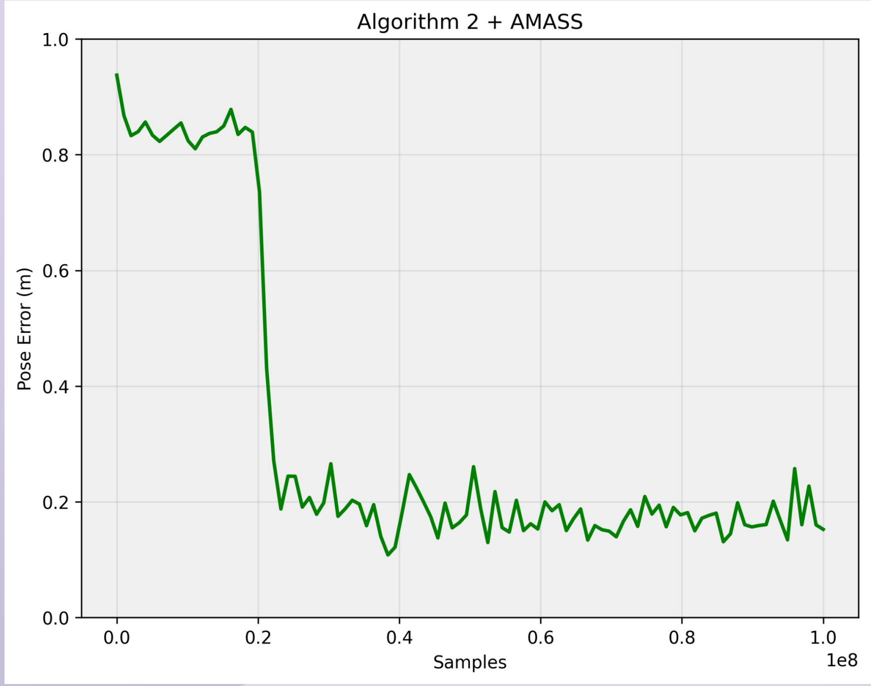


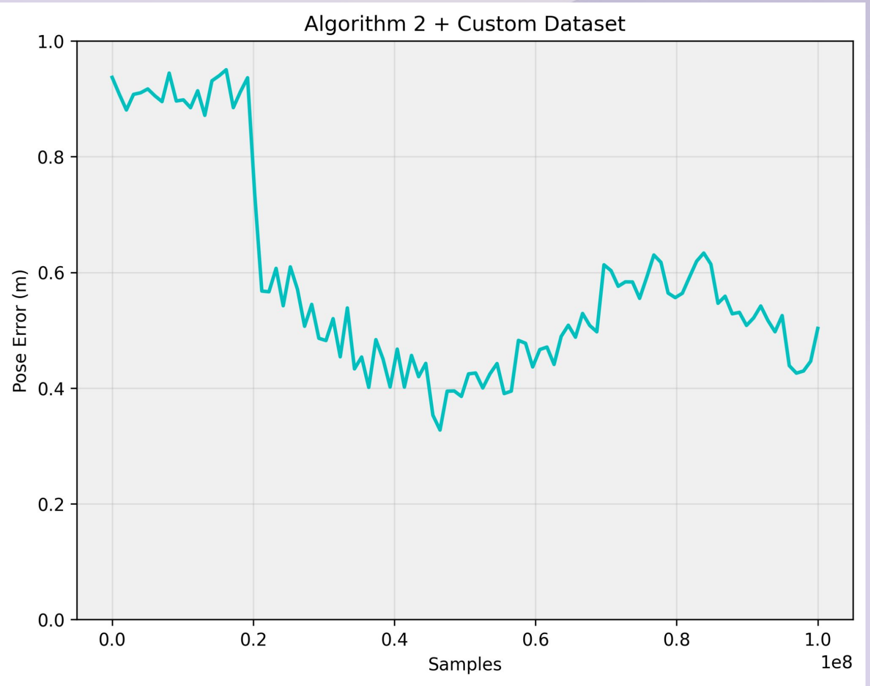


Contact:
Shivam Kak | 6307700880 | sk3686@princeton.edu

Prof. Benjamin Eysenbach

Results

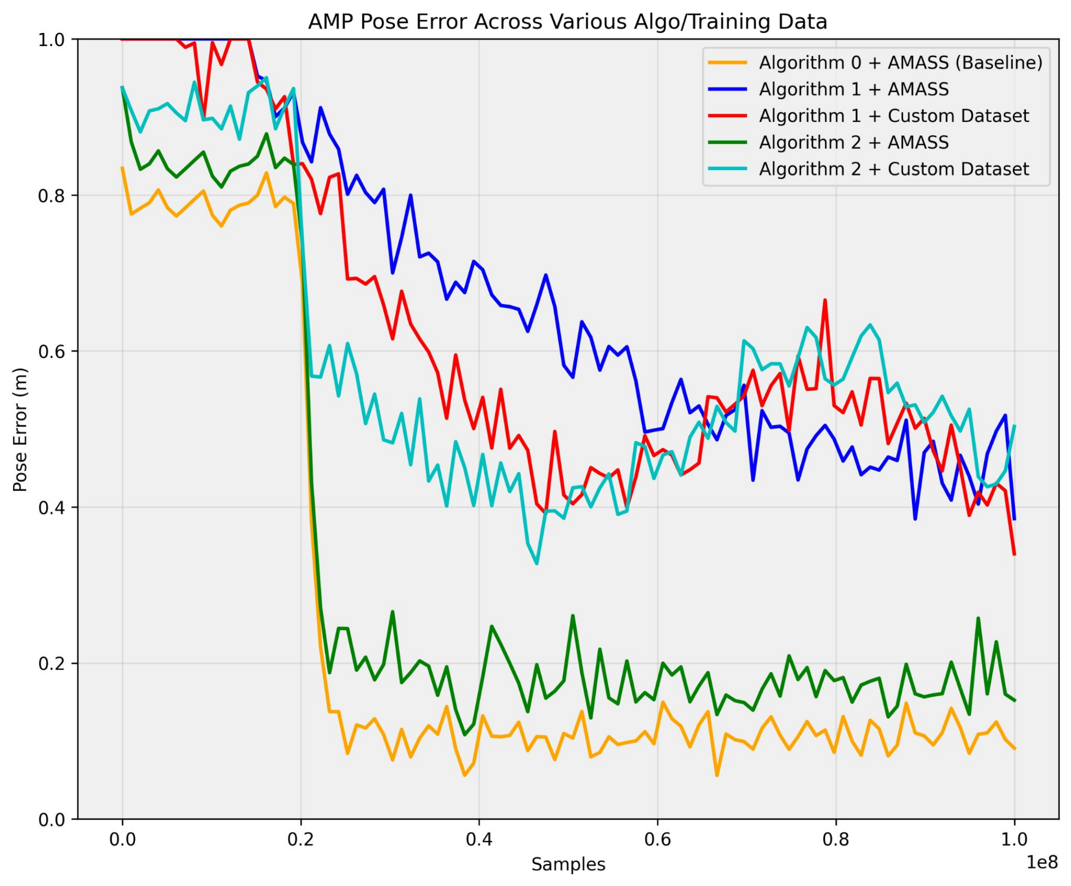




Contact:
Shivam Kak | 6307700880 | sk3686@princeton.edu

Prof. Benjamin Eysenbach

Results



Contact:
Shivam Kak | 6307700880 | sk3686@princeton.edu

Prof. Benjamin Eysenbach

Conclusion

Conclusion

- Future Work
  - Try different subsets of the AMASS dataset
  - Experiment with different task focuses or try a generalized implementation
  - Experiment with different adjustments to AMP base algorithm
- Importance
  - Contribute to reducing expense/labor of animation for movies / games
  - Train agents to understand characteristics of motion from motion capture data

Contact:
Shivam Kak | 6307700880 | sk3686@princeton.edu

Prof. Benjamin Eysenbach