INVERSE REINFORCEMENT LEARNING DEFINITION:

**Inverse Reinforcement Learning (IRL)** does, in fact, **use expert's state-action pairs**. The goal of IRL is to infer the **reward function** that the expert is implicitly optimizing based on the expert's **behavior** (which is represented by the state-action pairs). Here’s how it works:

**How IRL Uses Expert's State-Action Pairs:**

* **State-action pairs** are the key data in IRL. These pairs represent the **decisions** that the expert made in various situations (states) and what action they took in those states.
* IRL **does not directly learn the policy** (the mapping from states to actions) as imitation learning does, but it uses the state-action pairs to infer the reward function that could have led the expert to take those actions.

Once the reward function is inferred from these state-action pairs, the agent can use reinforcement learning (RL) to **optimize** its own behavior based on the learned reward function.

Paper: A Survey of Imitation Learning: Algorithms, Recent Developments, and Challenges

 **IRL Overview:**

* IRL involves an **apprentice agent** that tries to **infer the reward function** underlying observed expert demonstrations. The agent then optimizes its policy using **Reinforcement Learning (RL)**, by interacting with its environment and adjusting based on observed rewards, which helps it learn better over time.

 **Challenges in IRL:**

* **Computational Expense:** The computational requirements for IRL are typically higher compared to other methods, which can be a challenge.
* **Ambiguity Between Policy and Reward Function:** One significant challenge in IRL is the **ambiguity in the relationship** between the policy and the reward function. A single optimal policy can correspond to **multiple reward functions**, making it challenging to infer the exact reward function from demonstrations.

 **Categories of IRL Algorithms:** The paper introduces three major categories of IRL methods that aim to address the ambiguity in reward inference:

* **Maximum-Margin Methods:** These methods aim to find a reward function that best explains the expert's behavior by a margin. This ensures that the learned reward function is distinct and optimal for the demonstrated behavior.
* **Maximum Entropy (MaxEnt) Methods:** These approaches try to resolve ambiguity by maximizing the **entropy** of the resulting policy. This adds a level of randomness to avoid overfitting to a single policy and helps generalize the behavior learned from demonstrations.
  + **MaxEntIRL** and **Maximum Entropy Deep IRL** use deep neural networks to model complex, nonlinear reward functions, enabling the model to handle more intricate tasks.
* **Bayesian Methods:** Bayesian IRL methods use the expert's actions to estimate a **posterior distribution** over possible reward functions. This allows the algorithm to incorporate uncertainty in the reward function and update it dynamically as more data becomes available.

 **Real-World Considerations:**

* **Transition Model and Expert Policy Estimation:** In practice, most IRL algorithms rely on the assumption that the **transition model** and **expert's policy** are known, which is often unrealistic in real-world settings. In most cases, the agent must **estimate these models** from observed samples, which can introduce errors into the reward function inference.

Paper: Algorithms for Inverse Reinforcement Learning

**1. Understanding the IRL Problem**

* The paper defines **Inverse Reinforcement Learning (IRL)** as the process of inferring a reward function from observed behavior, rather than manually specifying one.
* In a typical IRL setting:
  + The **expert** is an agent (human, robot, etc.) demonstrating behavior.
  + The **trajectories** are sequences of states and actions observed from the expert.
  + The **goal** is to recover the reward function that explains why the expert behaves a certain way.
* This aligns with many applications, including **robot learning, autonomous driving, and human behavior modeling from videos**.

**2. The Challenge of Degeneracy in IRL**

* **Degeneracy** means that **multiple reward functions** can produce the **same optimal behavior**.
* This is problematic because IRL solutions might infer **arbitrary rewards** rather than the true underlying intent.
* The paper addresses this by **favoring reward functions that make deviations from optimal behavior costly**, which **encourages the most plausible explanation for the expert’s actions**.

**5. Selecting the Best Reward Function**

* Since multiple reward functions can explain the same behavior, the paper proposes a **heuristic**:
  + **Prefer rewards where deviations from optimal actions are costly.**
  + This **encourages stability and prevents arbitrary solutions**.
* In **video-based IRL**, this could help compare different representations:
  + A **better representation** is one where **small deviations in behavior cause significant differences in inferred rewards**.

GRAPH BASED REPRESENTATION:

Paper: Action Scene Graphs for Long-Form Understanding of Egocentric Videos

 **Egocentric Action Scene Graphs (EASGs)**

* A **structured representation** of actions over time, used for **long-term understanding** of egocentric videos.
* **Nodes** represent the camera wearer, action verb, and objects involved, while **edges** capture relationships between them.
* **Temporal Evolution**: EASGs model interactions in three key frames (Precondition, Point of No Return, Postcondition).

 **Graph-Based Representation of Actions**

* EASGs are **dynamic, time-varying graphs** where nodes represent actions, objects, and actors.
* The **temporal evolution** of the graph tracks interactions over time, providing a structured view of actions.
* Suitable for **long-form video understanding** and modeling **human behavior**.

 **Predicting Future Scene Graphs**

* EASGs can predict future scene graphs by analyzing past action sequences.
* The task is to predict future relationships (e.g., action-object interactions) based on observed actions.

 **EASGs for Inverse Reinforcement Learning (IRL)**

* **Temporally-aware representation** that helps in **inferring reward functions** for IRL tasks.
* Captures action-object relationships over time, providing richer learning signals than traditional methods.
* IRL agents can learn to **replicate human behavior** more effectively.

 **Graph vs. Image-Based Representations**

* **EASGs** provide both **spatial and temporal relationships**, tracking interactions over time.
* Unlike static image-based representations, EASGs model **long-term dependencies** in human behavior.

 **Benefits of EASGs in IRL**

* **Structured learning signal**: Enables agents to learn complex behaviors based on observed interactions.
* **Prediction capability**: Agents can predict future actions and adjust behavior accordingly, improving long-term behavior modeling.

Paper: RoboEXP: Action-Conditioned Scene Graph via Interactive Exploration for Robotic Manipulation

**Interactive Scene Exploration in Robotics**

* **RoboEXP** enables a robot to autonomously explore environments by interacting with them and constructing an **Action-Conditioned Scene Graph (ACSG)**. This graph encodes both **spatial relationships** and **action effects**, allowing the robot to understand how its interactions influence the world.
* This approach is valuable for **behavior learning**, as the robot actively builds structured knowledge about objects and actions rather than passively observing its surroundings.

**Action-Conditioned Scene Graphs (ACSGs) and Their Role in IRL**

* ACSGs integrate **both actions and objects** into a graph where nodes represent objects or actions, and edges define **how actions affect objects**.
* In **Inverse Reinforcement Learning (IRL)**, such a structured representation is advantageous because it **models causal relationships** between actions and their effects, aiding in **reward inference** from expert demonstrations.
* By leveraging ACSGs, an IRL agent can **better infer the underlying reward function** by understanding not just what actions occur but how they influence the environment.

**Temporal Evolution and Incremental Learning**

* The **RoboEXP** system continuously updates the ACSG as it interacts with the environment, adding new objects and actions.
* This aligns with IRL’s goal of **incremental learning**, where an agent refines its policy based on observed human behaviors over time.

**Reward Structure for Exploration and Behavior Replication**

* RoboEXP rewards the robot for **discovering new objects** and **exploring efficiently**.
* This principle can be applied in IRL by shaping a reward function that **guides an agent toward behaviors that closely replicate human actions**.
* For instance, an IRL agent could receive **positive rewards** for accurately mimicking expert behavior and **penalties** for deviating from expected actions.

**Zero-Shot Learning and Generalization in IRL**

* RoboEXP is capable of **zero-shot generalization**, meaning it can explore and manipulate objects in novel environments without prior exposure.
* In IRL, this translates to an agent **learning human behavior in one environment (e.g., simulation)** and successfully **transferring that knowledge to new, unseen scenarios** (e.g., real-world settings or video-based learning).

**Scene Graph Construction for Robotic Manipulation and IRL**

* The ACSG structure allows a robot to **interact with objects** and **perform goal-directed manipulation tasks**.
* In IRL, this structured representation can help agents **model human interactions with the environment** and predict **how different actions lead to specific outcomes**.

Paper: Graph Inverse Reinforcement Learning from Diverse Videos

**1. Diversity in Videos for IRL**

* This paper emphasizes **leveraging diverse video sources**, particularly **third-person videos**, to help IRL scale and learn more **generalized reward functions**. In real-world settings, **first-person demonstrations** are often unavailable, so **third-person videos provide a broader, more scalable way** to train IRL models.

**2. Graph Abstraction of Videos**

* Instead of processing **raw pixel data**, this paper proposes using **graph-based representations** to **abstract interactions** within a video. Raw images contain **irrelevant textures, lighting variations, and background noise** that don’t contribute to learning task-relevant rewards. A **graph-based approach** represents **objects as nodes** and **interactions as edges**, allowing for a **structured understanding** of task dynamics.
* This is **highly relevant to IRL**, as reward functions must capture **how actions influence the environment over time**, which is naturally modeled by **spatial and temporal relationships in a graph**.

**3. Modeling Interactions with Interaction Networks**

* The paper utilizes **Interaction Networks (INs)**, a deep learning method that explicitly models how **objects interact and evolve over time**.
* By structuring these interactions as a **graph**, the IRL model can infer how different **actions contribute to task completion**, improving reward learning.

**4. Temporal Matching and Reward Function**

* To learn an effective reward function, the paper introduces **Temporal Cycle Consistency (TCC)**, which ensures that **actions align correctly across different video sequences**.
* **Why is this important for IRL?**
  + IRL requires **consistent temporal alignment** of observed behaviors to correctly infer **progression toward a goal**.
  + TCC helps match **similar action sequences across different videos**, improving the model’s ability to **generalize learned rewards**.

**5. Embedding Space and Task Progression**

* The paper constructs an **embedding space** where video sequences are mapped based on **task progression**.
* The agent’s reward is defined based on **how close its current state is to expert demonstrations in this embedding space**.
* This provides a **continuous measure of progress**, helping IRL models better define **rewards for intermediate steps**, rather than relying solely on **final task success**.
* This approach improves **robustness**, as it allows an agent to infer **reward signals even when exact matches to training demonstrations are unavailable**.

IMAGE BASED REPRESENTATION:

Paper: TOWARDS UNIVERSAL VISUAL REWARD AND REPRESENTATION VIA VALUE-IMPLICIT PRE-TRAINING

**1. Value-Implicit Pretraining (VIP) for General Reward Learning**

* **VIP enables generalizable visual reward learning** by training on large, unlabeled **human video datasets**, making it useful for **robotic tasks without task-specific fine-tuning**.
* Unlike traditional **task-specific RL models**, VIP **learns from human behavior passively**, making it more **scalable** and **widely applicable**.
* This is important for **Inverse Reinforcement Learning (IRL)** because it allows agents to **derive reward functions that generalize across multiple tasks**, similar to how **human behaviors can be generalized across different contexts**.

**2. Learning from Human Videos Without Action Labels**

* Instead of requiring **explicit robot action labels**, VIP uses **goal-conditioned reinforcement learning** on **unlabeled human video data**.
* This makes VIP a **self-supervised learning approach**, where the agent extracts **task-relevant reward signals** from human goal-directed behavior.

**3. Goal-Conditioned Learning for Robotic Tasks**

* **How does VIP define rewards?**
  + VIP **measures the distance between the robot’s current state and a goal state** in an **embedding space**.
  + The **closer the robot is to the goal representation, the higher the reward**, guiding the agent’s behavior **without needing explicit action labels**.

**4. Self-Supervised Learning and Scalability**

* **VIP’s self-supervised learning approach** means it can train on **massive unlabeled datasets** of human behavior, making it **much more scalable** than traditional **hand-annotated** IRL datasets.

**5. Generalizable Reward Model Across Tasks**

* **VIP’s learned reward model** can be applied to **unseen tasks**, making it a **universal reward model**.
* **Why does this matter for IRL?**
  + **IRL seeks to extract reusable reward functions** that can generalize to **new tasks** and **different environments**.
  + VIP **provides a structured way** to **transfer learned reward functions** to new settings, improving **task adaptation and generalization**.

The key difference between **Offline Goal-Conditioned Reinforcement Learning (Offline GCRL)** and **Inverse Reinforcement Learning (IRL)** lies in their objectives and learning paradigms:

**1. Objective:**

* **Offline GCRL**: Learns a policy to achieve specific goals using a fixed dataset of past interactions.
* **IRL**: Infers the underlying reward function from expert demonstrations and then typically trains an RL agent using this inferred reward.

**How VIP obtains the reward function:**

1. **Learning from Unlabeled Human Videos:** VIP trains on human behavior in video form, using **goal-conditioned reinforcement learning** to infer rewards from changes in visual data rather than explicit action labels.
2. **Goal-Conditioned Rewards:** The key idea in VIP is to measure the **distance between a current state** (visual embedding) and a **goal state**. The agent learns to navigate in an embedding space where the goal is defined as a **desired state** (e.g., a human performing a task in a video). The closer the agent gets to the goal state, the higher the reward.
3. **Self-Supervised Learning Approach:** VIP works in a self-supervised manner:
   * It doesn't need hand-annotated labels or explicit goals.
   * The rewards emerge from **visual progress** toward the goal. The agent learns what constitutes progress through observation.
   * As the agent processes video data, it learns to correlate visual features with the progression toward a goal, effectively **deriving a reward function** from passive, unlabeled human demonstrations.
4. **Emergent Rewards:** Through the optimization process, VIP learns to predict the **value** (reward) of each state based on visual features, meaning that the agent **implicitly learns the reward structure** needed to achieve a task.

Paper: Unsupervised Perceptual Rewards for Imitation Learning

**1. Unsupervised Discovery of Task Sub-goals**

* The proposed method **automatically discovers intermediate steps (sub-goals)** in a task from a **few human demonstration videos**.
* Instead of **manually defining rewards**, the model learns **which steps matter** by training classifiers on **high-level representations from a pre-trained deep model**.
* These **sub-goals provide more structured feedback** compared to traditional reward learning, helping the agent understand **not just the final goal but also the steps to get there**.

**2. Learning Reward Functions from Video Demonstrations**

The classifiers trained on intermediate steps are **combined into a single step-wise reward function**, ensuring that the agent gets **continuous feedback throughout the task**.

**3. Relation to Inverse Reinforcement Learning (IRL)**

* This method **extends IRL** by focusing on **task decomposition**—instead of learning a single reward function, it learns a **sequence of sub-goal rewards**.
* It is similar to **SWIRL (Sub-goal Weighted IRL)** and **other segmentation-based IRL methods**, which also **break down tasks into smaller steps** for improved learning.

**4. Simplified IRL Model for Low-Data Learning**

* Instead of solving the full **MaxEnt IRL optimization problem**, this method **approximates it using a simplified assumption**:
  + It assumes **independence between time steps and features** (similar to **naïve Bayes**), which makes the model **computationally efficient**.

Paper: Model-Based Inverse Reinforcement Learning fromVisual Demonstrations

**1. Model-Based IRL Approach**

* **Why model-based?** Unlike **model-free IRL**, which learns policies directly from experience, **model-based IRL** explicitly learns a **dynamics model** of the environment.
  + This allows **planning over predicted future states**, reducing the need for excessive trial-and-error.
  + However, **real-world robotics often prefers model-free IRL** due to challenges in learning **accurate** dynamics models.
* The framework involves **two nested optimizations**:
  + **Inner optimization** – Learns a policy given a **cost function** and a **transition model**.
  + **Outer optimization** – Adjusts the **cost function parameters** to match expert demonstrations.

**2. Key Components of the System**

The system consists of three main modules:  
1️⃣ **Keypoint Detector** – Extracts **low-dimensional** visual features (keypoints) from RGB images.  
2️⃣ **Dynamics Model** – Predicts **state transitions** based on keypoints and actions.  
3️⃣ **Gradient-Based Model Predictive Planner (MPC)** – Optimizes actions using the learned **cost function and dynamics model**.

* This setup enables **visual model-predictive control (MPC)**, where the agent uses its learned model to predict and **optimize future actions**.

**3. Learning Visual Representations with Keypoints**

* Instead of processing **high-dimensional images**, the model extracts **2D keypoints** using an **autoencoder with a structural bottleneck**.
* **Why keypoints?**
  + They **reduce input complexity** while preserving task-relevant information.
  + They help in **generalization**, as the system focuses on **essential motion cues** rather than raw pixel values.
* The dynamics model then predicts **how these keypoints evolve over time**, allowing the agent to anticipate future states.

**4. Inverse Reinforcement Learning (IRL) Optimization**

* The **inner loop** optimizes the **policy** based on a given cost function.
* The **outer loop** adjusts the **cost function parameters** so that the predicted trajectory **matches expert demonstrations** in latent space.
* The **IRL loss** measures the difference between the robot’s predicted trajectory and the human demonstration, ensuring that the cost function correctly **captures task-relevant behaviors**.

**5. Time-Dependent Cost Function for Task Adaptation**

* A **time-dependent cost function** is introduced, meaning different keypoints **are weighted differently at different time steps**.
  + **Why?** Some aspects of a task (e.g., grasping an object) **matter more at specific stages**, so the model **prioritizes them dynamically**.
* The cost function measures the **distance in the latent keypoint space** between the predicted and expert states, ensuring smooth learning from visual demonstrations.

**6. Learning from Relative Demonstrations**

* Instead of assuming that expert demonstrations are **absolute** (fixed start and end positions), the model **learns from relative trajectories**.
  + This allows for **better generalization** across different initial conditions.
  + The robot can start from **varied positions** and still learn to reach the goal effectively.

This approach is **directly based on images**, but it does **not** process full raw images. Instead, it extracts **low-dimensional representations** using a **keypoint detector**. The extracted **keypoints** serve as a compact, structured representation of the scene, which makes learning and planning more efficient.

* **Not Graph-Based:** Unlike **graph-based IRL** (which uses graph structures like scene graphs or relational graphs to model spatial and semantic relationships), this method operates on **latent visual representations** learned from images.
* **Why Keypoints?** Keypoints provide a **simplified, structured representation** of the scene without needing explicit graph construction. They serve as **intermediate features** between raw pixels and full scene graphs.

**How Does This Model-Based IRL Learn the Reward?**

The reward (cost function) is learned **through an optimization process** that aligns the agent’s behavior with expert demonstrations. Here’s how it works:

**1. Learning a Latent Space Representation**

* The model **extracts keypoints** from images to create a **low-dimensional latent representation** of the state.
* A **dynamics model** is trained to predict future keypoint positions based on actions.

**2. Inverse Reinforcement Learning (IRL) Optimization**

* The IRL process involves **two levels of optimization**:  
  1️⃣ **Inner Loop (Policy Optimization):** The agent finds an optimal **policy** (sequence of actions) given a current cost function and the learned transition model.  
  2️⃣ **Outer Loop (Cost Function Learning):** The cost function (reward function) is adjusted so that the agent’s behavior aligns with the expert demonstrations.
* **Loss Function:** The model minimizes the difference between:
  + The **latent trajectories** predicted by the agent’s policy.
  + The **latent trajectories** observed in expert demonstrations.
* **Time-Dependent Cost Function:** The cost function is weighted differently at different time steps to emphasize important stages of the task (e.g., grasping is more important early, while placing is more important later).

Paper: Time-Contrastive Networks: Self-Supervised Learning from Video

**1) Viewpoint-Invariant Representation for Imitation Learning**

* **TCNs learn viewpoint-invariant representations** in a self-supervised manner, focusing on motion-relevant features rather than raw appearance.
* This is particularly useful for **learning human behaviors from videos**, where camera angles may vary.

**2) Learning a Structured Metric Space for Imitation**

* TCNs use **triplet loss** to learn a structured metric space:
  + **Pulls together** temporally aligned frames from different viewpoints.
  + **Pushes apart** visually similar frames that are temporally distant.
* This results in a feature space where:
  + **Task-relevant actions cluster together**, independent of viewpoint.
  + **Unrelated motions remain distinguishable**, avoiding confusion due to similar-looking but unrelated frames.

**3) Leveraging Pretrained Features (e.g., ImageNet) to Reduce Data Requirements**

* TCNs **do not train from scratch**; instead, they use **pretrained deep networks** (e.g., ResNet, ImageNet features).
* This enables **semantically meaningful feature extraction with fewer labeled demonstrations**, reducing the need for extensive data collection.

**4) Self-Supervised Learning Without Labels**

* Unlike traditional imitation learning, which requires **explicit trajectory labels or state-action pairs**, TCNs use **self-supervised learning** to discover meaningful representations.
* This enables **unsupervised behavior learning from raw video demonstrations**, reducing dependence on expensive annotations.

**5) Constructing a Reward Function for Reinforcement Learning**

* One application of TCNs is to **define a reward function based on embedding distances**:
  + The distance between a **TCN-encoded human demonstration** and a **robot’s observation** can serve as a **similarity-based reward signal**.
  + This allows learning from **video demonstrations without explicit state-action pairs**.
* The method transitions from:
  + **Strong learning signals** (Euclidean distance).
  + **Fine-tuning phase** (Huber loss for smooth optimization).

📌 **Clarification:**

* **TCNs do not directly infer IRL reward functions**; instead, they provide a **similarity measure**, which can be **used as a reward signal** for reinforcement learning.

**6) Practicality in Real-World Settings: Single-View Demonstration, Multi-View Training**

* TCNs **are trained on multi-view videos** but can **generalize to single-view demonstrations**.
* This means that at **test time, only a single video from an unseen viewpoint is needed**.
* This is crucial for **real-world IRL applications**, where multiple synchronized cameras are often unavailable.

**Alternative to Graph-Based Approaches:**

* Instead of modeling **state transitions as graphs**, TCNs extract **task-relevant embeddings** directly from raw video pixels.

**Time-Contrastive Networks (TCNs) can support Inverse Reinforcement Learning (IRL)** by learning **a structured embedding space** where task-relevant behaviors are well-separated from irrelevant visual variations. However, TCNs do not directly perform IRL—they provide an intermediate representation that makes IRL more effective.

**Ways TCNs Enhance IRL:**

1️⃣ **Feature Representation for Reward Learning**

* TCNs **transform raw video frames into an embedding space** where similar actions are closer, even across different viewpoints.
* Instead of performing IRL on **raw pixel data** (which is inefficient), IRL can use **TCN embeddings as state representations**, making reward learning **more structured and robust**.

2️⃣ **Reward Function Approximation via Embedding Distances**

* IRL requires a reward function, but manually designing one is difficult.
* A **TCN-based reward** can be defined using the **distance between the embedding of an agent's state and an expert’s state**: R(s)=−d(f(s),f(s∗))R(s) = - d(f(s), f(s^\*))R(s)=−d(f(s),f(s∗)) where f(s)f(s)f(s) is the TCN-encoded state representation, and s∗s^\*s∗ is an expert demonstration state.
* This allows IRL to optimize policies based on **task-relevant similarity rather than raw pixel differences**.

3️⃣ **Handling Partial Observability & Viewpoint Variations**

* Traditional IRL assumes **consistent state representations**, which is difficult with video data.
* TCNs ensure that **the same action looks similar regardless of camera viewpoint**, improving **policy generalization** in IRL.

**Is TCN a Graph-Based or Image-Based Representation?**

🔹 **TCN is an Image-Based Representation, NOT a Graph-Based Approach.**  
🔹 It learns **embeddings directly from raw video frames** instead of **modeling transitions as a graph**.

However, **TCN features can be integrated into graph-based IRL** by using the embeddings as **node representations in a state transition graph**.

 **TCNs are image-based**: They work directly with raw pixels and learn **viewpoint-invariant embeddings**.

Paper: Temporal Cycle-Consistency Learning

**1) Problem: Learning Temporal Correspondence from Video Demonstrations**

A major challenge in **Imitation Learning and Inverse Reinforcement Learning (IRL)** is aligning multiple video demonstrations of the same task when:  
❌ **Execution speeds differ** (e.g., one person moves faster than another).  
❌ **Viewpoints change** (e.g., different camera angles).  
❌ **Execution styles vary** (e.g., different hand movements for the same action).

**Why Traditional Methods Fail:**

* **Keypoint tracking**: Fails when body shapes, lighting, or perspectives change.
* **Feature matching**: Struggles with speed variations.
* **Manual annotations**: Expensive and not scalable.

**Solution: Temporal Cycle-Consistency Learning (TCC)**

**TCC learns self-supervised temporal alignment** across multiple demonstrations, ensuring actions are correctly matched in time **without human labels**.

**2) What is Temporal Cycle-Consistency Learning (TCC)?**

TCC is a **self-supervised method** that learns **temporally aligned embeddings** for video frames by enforcing **cycle-consistency across multiple sequences** of the same task.

**Key Properties of TCC:**

✅ **Learns per-frame correspondences** across videos with varying execution speeds.  
✅ **Encodes temporal structure** so corresponding actions align, even with appearance differences.  
✅ **Uses cycle-consistency**—ensuring a frame matched to another video cycles back correctly.

👉 **Core Idea:** If a **frame A in one video** corresponds to **frame B in another**, then B should **cycle back** to A. If this consistency holds, the learned embeddings are **temporally structured**.

**3) Core Mechanism: Temporal Cycle-Consistency Loss**

TCC enforces **temporal consistency** using a **cycle-consistency loss**:  
1️⃣ **Finds the nearest-neighbor frame** in another sequence using learned embeddings.  
2️⃣ **Cycles back to the original sequence** by matching again.  
3️⃣ **Minimizes error** if the cycled-back frame is close to the original frame.

💡 **Why This Matters:**

* **Handles execution order differences** while allowing small variations.
* **Learns time-aware representations** without manual annotations.
* **Generalizes across different people, speeds, and viewpoints.**

**4) How Does TCC Help in IRL and Imitation Learning?**

TCC is **especially useful for IRL** because:  
✅ **Aligns demonstrations temporally**, making it easier to extract expert behavior.  
✅ **Works on unstructured video data**, removing the need for labeled datasets.  
✅ **Preserves execution ordering**, which is critical for defining reward functions in IRL.  
✅ **Handles diverse execution styles**, allowing more robust behavior learning.

**Example in IRL: Robot Learning from Videos**

Temporal Cycle-Consistency Learning (TCC) is **not a direct IRL method**, but it is **highly beneficial** for IRL by improving how demonstrations are processed. Specifically, it helps by **aligning and structuring video demonstrations**, making it easier to infer meaningful reward functions.

**Key Contributions to IRL:**

✅ **Aligns Temporal Sequences in Demonstrations**

* IRL relies on **trajectory data** (state-action pairs). In videos, actions may happen at different speeds.
* **TCC ensures that different demonstrations align correctly**, allowing the IRL algorithm to extract consistent states and actions.

✅ **Reduces Noise in State Representations**

* Traditional IRL methods struggle with **variations in execution styles**.
* TCC creates a **normalized temporal structure**, reducing ambiguity when inferring rewards.

✅ **Improves Feature Representations for IRL**

* IRL requires **state representations** to estimate rewards.
* TCC provides a **better structured latent space**, making it easier for IRL models to generalize across videos.

**Is TCC a Graph-Based or Image-Based Representation?**

TCC is **neither purely graph-based nor purely image-based**, but it leans more toward an **image-based representation with learned temporal structure**.

1️⃣ **Image-Based Representation (Primary Focus)**

* TCC learns an **embedding space from video frames**.
* It works on raw video data and **does not require explicit graph structures**.
* The embeddings are structured **temporally**, but they remain a form of image-based representation.

**How Would You Use TCC in an IRL Pipeline?**

1️⃣ **Step 1: Preprocess Video Data**

* Collect multiple demonstrations of a task (e.g., humans opening a door).
* Apply TCC to align demonstrations and learn temporally structured embeddings.

2️⃣ **Step 2: Convert Aligned Frames into an IRL-Compatible Format**

* Use the TCC embeddings as **state representations** instead of raw image pixels.
* If using a **graph-based IRL approach**, construct a state-transition graph using TCC embeddings.

3️⃣ **Step 3: Apply IRL to Infer the Reward Function**

* Train an IRL algorithm (e.g., MaxEnt IRL) using the structured TCC representations.
* The learned reward function can now be used to train a policy.

**Final Answer: How TCC Fits with IRL**

* **TCC is primarily an image-based representation** that learns **temporally structured embeddings** from videos.
* It helps IRL by **aligning demonstrations**, **reducing noise**, and **improving state representations**.
* While **not inherently graph-based**, its embeddings **can be used in a graph-based IRL approach** for structured learning.

Paper: XIRL: Cross-embodiment Inverse Reinforcement Learning

**1) Problem: Imitating Tasks Across Different Embodiments**

The key challenge that **XIRL (Cross-Embodiment Inverse Reinforcement Learning)** addresses is:  
✅ Learning policies from video demonstrations where agents have **very different embodiments** (e.g., human hands vs. robot grippers).  
✅ Traditional imitation learning methods struggle with embodiment gaps because:

* **Shape and size** differences between humans and robots.
* **End-effector variations** (e.g., hands vs. robotic claws).
* **Motion differences**, such as different speeds or styles of execution.
* **Viewpoint differences**, as human demonstrations often look different from robot-perspective inputs.

This is **especially important** for human-to-robot imitation learning, where we lack **ground-truth actions** (e.g., torques or joint angles) from human demonstrations.

**2) What is XIRL and How Does It Work?**

XIRL enables an agent (e.g., a robot) to **learn from video demonstrations of humans performing a task**, even though the human's movements are **physically different** from the robot's.

To achieve this, XIRL uses:  
✅ **Temporal Cycle-Consistency (TCC)** to learn embodiment-invariant visual representations from offline videos.  
✅ These learned **visual embeddings** are then used to **define a reward function** for reinforcement learning (RL), allowing the agent to **imitate the task** across different embodiments.

**3) Key Concept: Embodiment-Invariant Reward Function**

**How does XIRL overcome embodiment differences?**

XIRL’s main innovation is that **it does not rely on physical action similarity** (e.g., joint angles, velocities). Instead, it learns **task-level representations** that remain **consistent across different embodiments**.

**How does TCC help?**

✅ **TCC learns a shared feature space** where corresponding time steps across videos of different embodiments are **aligned**, despite differences in speed, motion style, or execution details.  
✅ This allows XIRL to create a **reward function** based on the distance between an agent's current state and a **goal embedding** computed from human demonstrations.

**4) TCC and Reward Generation Process**

The method uses **TCC loss** to train an encoder (**φ**) that **learns embodiment-invariant visual representations** from videos.

**Steps in XIRL’s reward function generation:**

1️⃣ **Training the Encoder (φ):**

* The encoder takes an image frame **I** and outputs an **embedding φ(I)**.
* **TCC loss** aligns similar task states across videos of different embodiments.

2️⃣ **Computing the Goal Embedding (g):**

* The **goal embedding** is computed as the **average embedding** from the final frames of multiple expert demonstrations.
* This represents the **task’s final state** across different embodiments.

3️⃣ **Defining the Reward Function:**

* The reward at any state **s** is based on how close the agent’s current embedding **φ(s)** is to the goal embedding **g**:

r(s)=−1κ∥ϕ(s)−g∥2r(s) = -\frac{1}{\kappa} \|\phi(s) - g\|^2r(s)=−κ1​∥ϕ(s)−g∥2

* **Key Idea:** The agent receives **higher rewards** when its visual representation **matches the goal state**, regardless of embodiment differences.

**5) Is XIRL a Form of IRL? How Does It Relate?**  
🔹 Instead of recovering a reward function from **state-action trajectories**, XIRL extracts **task-relevant representations** that allow RL agents to learn effective policies.

**6) Benefits and Relevance to IRL**

✅ **Cross-Embodiment Learning:**

* XIRL enables **robots to learn from human demonstrations** even when their movement styles are **vastly different**.
* This is crucial for **scaling IRL** to real-world robotics, where robots must learn from human videos.

✅ **Eliminates the Need for Expert Action Labels:**

* Unlike traditional IRL, which **requires expert trajectories**, XIRL learns **from raw video data** without action annotations.
* This makes it **cheaper and more scalable** for real-world applications.

✅ **Enables Reward-Based Imitation Without Explicit IRL Optimization:**

* While **IRL typically involves reward inference**, XIRL **learns a structured embedding space** where reward functions emerge naturally.
* This makes it a **hybrid approach between IRL and self-supervised representation learning**.

✅ **Handles Variability in Execution Styles:**

* XIRL accounts for **differences in speed, motion style, and viewpoints** using **TCC-aligned embeddings**.
* This ensures that robots **understand the high-level task progression** rather than copying exact movements.

**8) Conclusion: Why XIRL Matters for IRL**

🚀 **XIRL offers a new way to learn rewards from video demonstrations without requiring action labels, making it an ideal candidate for scalable IRL applications.**  
🚀 **It generalizes across different embodiments, ensuring that IRL is not limited to settings where the learner and expert share the same physical form.**  
🚀 **By leveraging TCC-based visual embeddings, XIRL provides a flexible and robust way to structure reward learning for RL agents trained from video data.**

**How Can XIRL Be Used with IRL?**

XIRL can be used in **Inverse Reinforcement Learning (IRL)** by providing an **embodiment-invariant reward function** derived from video demonstrations. Instead of recovering a reward function explicitly from expert state-action pairs (as in traditional IRL), XIRL learns a **visual representation space** where rewards are based on how closely the agent's state matches expert demonstrations.

Here’s how XIRL integrates with IRL:

1️⃣ **Learning an Embodiment-Invariant Representation (Self-Supervised Pretraining)**

* XIRL **trains a feature encoder** using **Temporal Cycle-Consistency (TCC)**, ensuring that corresponding task phases from different embodiments (e.g., human vs. robot) are aligned in the **embedding space**.
* The **encoder extracts task-relevant features** from raw video frames, making it possible to compare robot states to human demonstrations **without requiring action annotations**.

2️⃣ **Defining a Reward Function from Video Demonstrations**

* Instead of explicitly learning a reward function through **IRL optimization**, XIRL defines a **reward based on visual similarity** between the agent’s current state and the goal state: r(s)=−1κ∥ϕ(s)−g∥2r(s) = -\frac{1}{\kappa} \|\phi(s) - g\|^2r(s)=−κ1​∥ϕ(s)−g∥2 where:
  + **φ(s)** = embedding of the current state.
  + **g** = embedding of the expert demonstration’s goal state.
* The agent is rewarded when its visual representation aligns with the **goal embedding** extracted from human videos.

3️⃣ **Using the Reward in IRL or RL**

* XIRL can be **plugged into standard RL or IRL frameworks**, using its learned reward function to **guide policy learning**.
* In a **standard IRL setting**, XIRL’s embedding space can be used to estimate **state similarities**, allowing for a **reward function approximation** that generalizes across embodiments.
* In an **RL setting**, the agent optimizes its policy using the XIRL reward, encouraging behaviors that bring its **visual state closer to the expert demonstration**.

**Is XIRL Graph-Based or Image-Based?**

✅ **XIRL is primarily an image-based representation learning approach.**

* **Image-Based:**
  + XIRL learns **task-relevant visual embeddings** from video frames using a CNN-based encoder (ResNet, etc.).
  + It aligns **video frames across different embodiments** based on their **temporal structure** rather than their physical execution.
  + The learned representations capture **task progression** while **ignoring embodiment-specific motion details**.