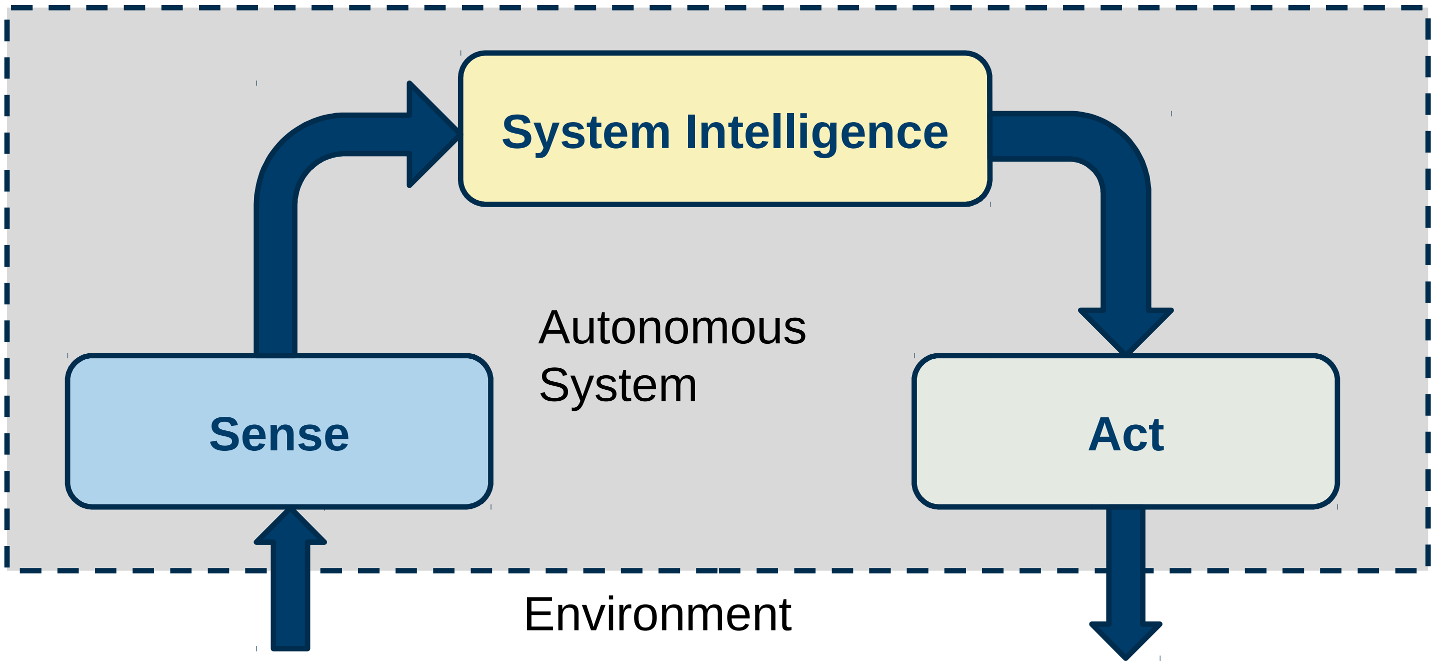
Forsvarets forskningsinstitutt

Deep Learning for Control in Robotics

Narada Warakagoda

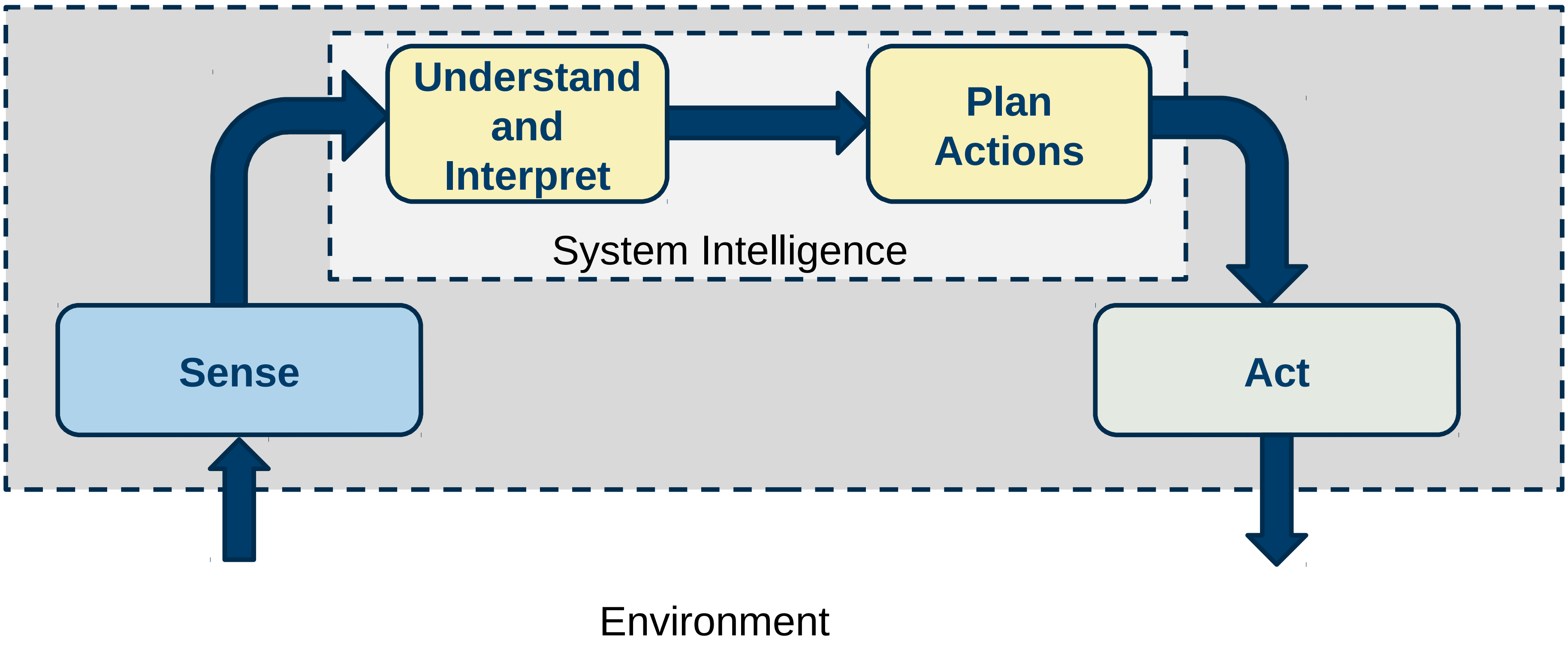
Robotics = Physical Autonomous Systems

- An autonomous system is a system that can auotomatically perform a predefined set of tasks under real world conditions
- Examples:
  - Autonomous vehicles (navigation)
  - Autonomous manipulator systems (manipulation)



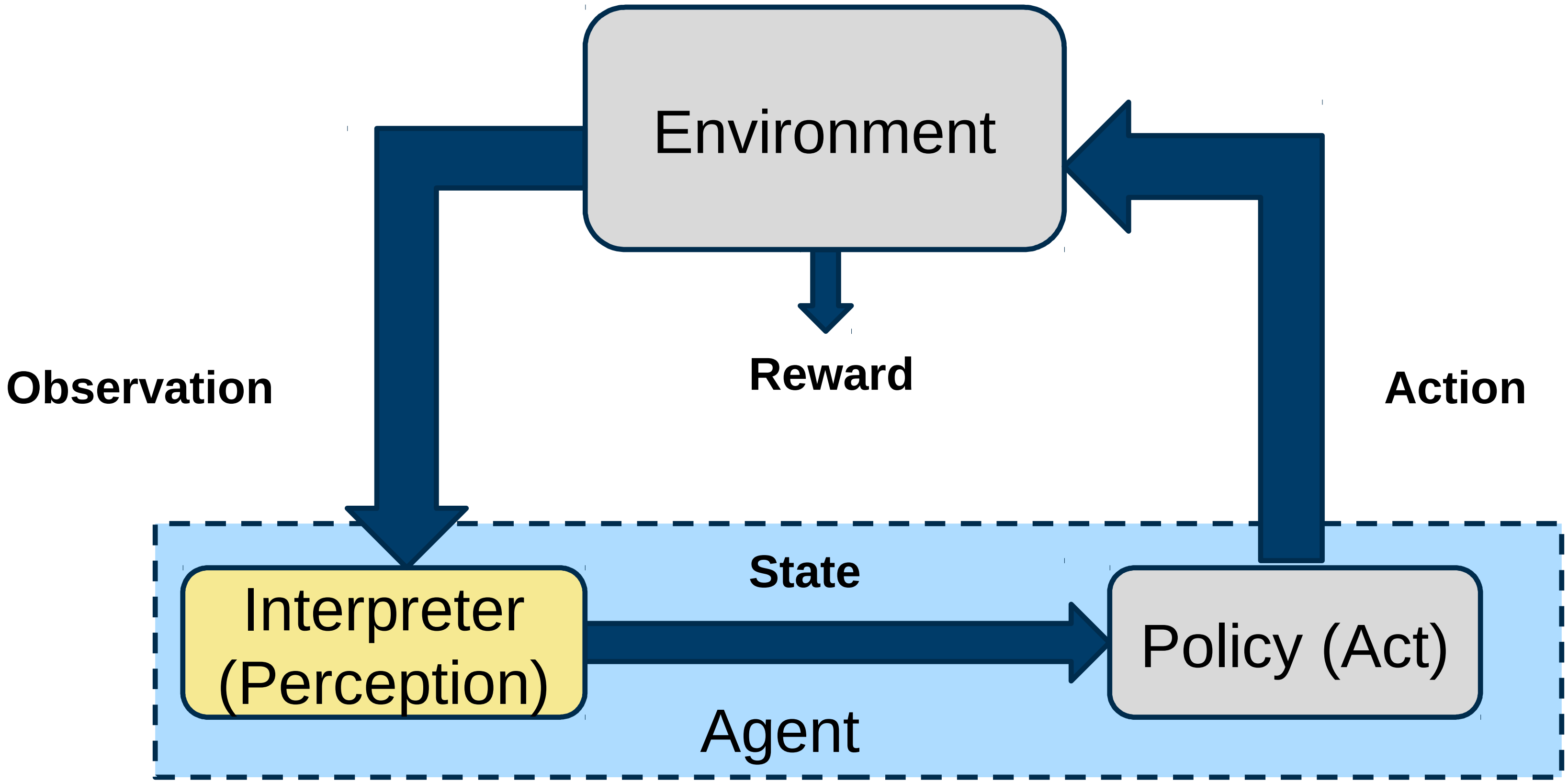
Designing Autonomous System Intelligence

- Main components
  - Understand/Interpret the sensor signals
  - Plan appropriate actions
- Going from manual design to automatic learning



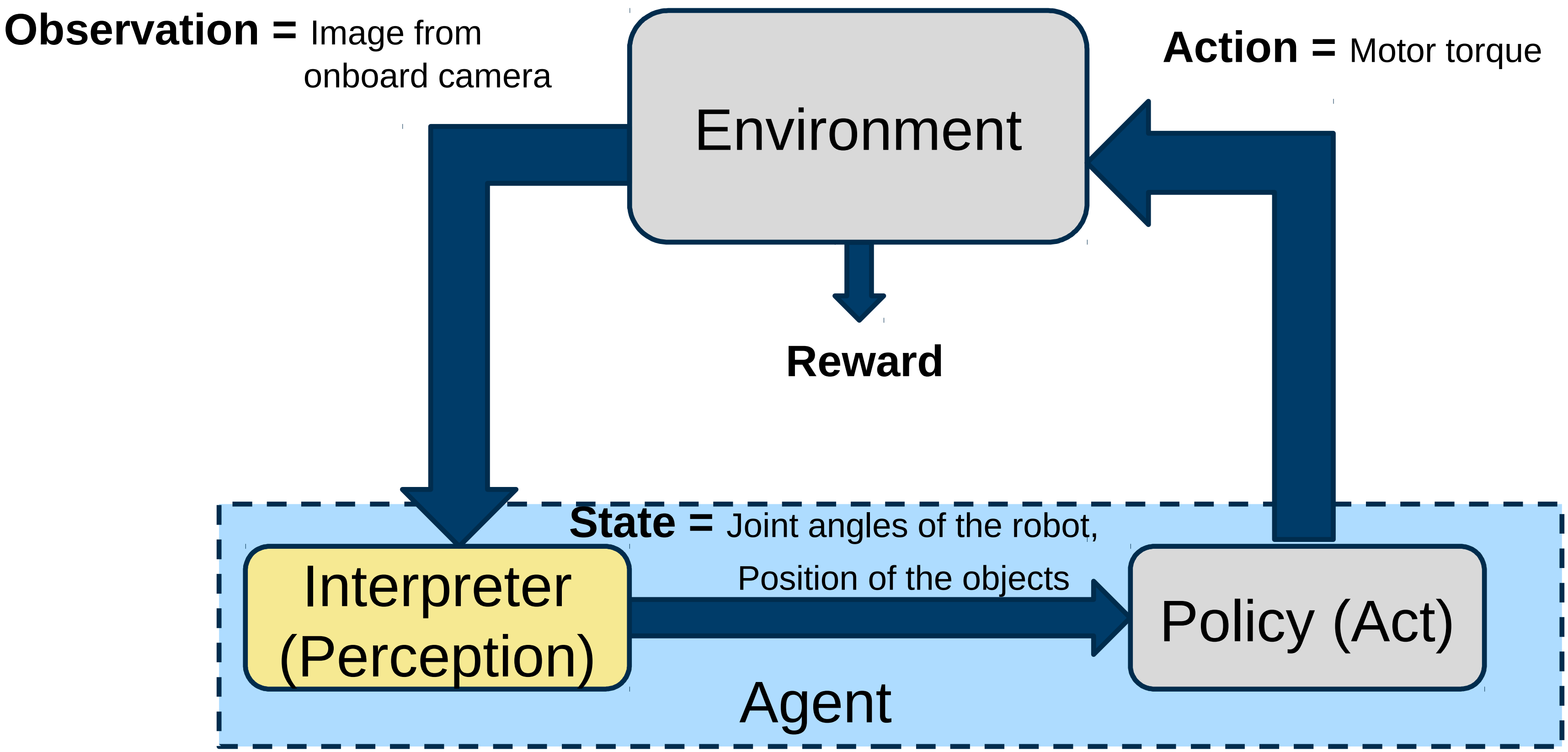
Reinforcement Learning

- We can cast the learning problem as a reinforcement learning problem



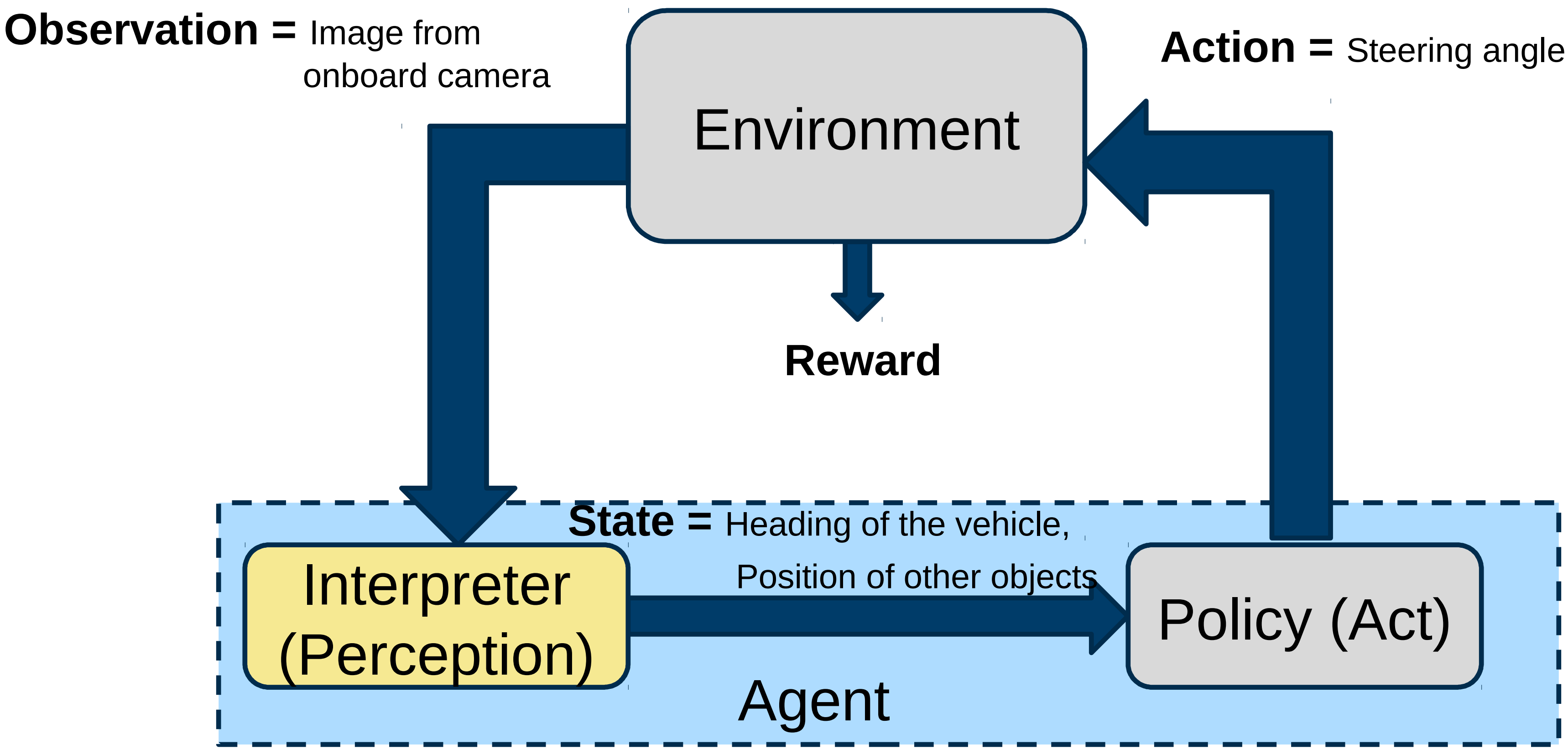
Example 1 (Manipulation)

- Controlling robotic arm



Example 2 (Navigation)

- Controlling an autonomous vehicle



Learnable Modules

- Policy/Control (state-to-action)
- Perception (observations-to-state)
- Policy+Perception (observations-to-action)
- Environment model (action+ current state -to- next state)
- Reward function (action+ current state -to- reward/cost)
- Expected rewards (Value functions Q, V)

Learning Perception vs. Control

- Data distribution
  - Perception learning uses iid assumption and it is reasonable
  - Control learning cannot use iid assumption, because data are correlated.
    - Errors can grow: compounding errors

- Supervision signal
  - Perception learning can be based on supervised learning
  - Control learning with direct supervision is not straight-forward.

- Data collection
  - Perception learning can use offline data
  - Control learning with offline data is difficult
    - Simulators
    - Can lead to realty gap

Weaknesses of Reinforcement Learning

- Learning through mostly trial and error
  - High cost in terms of time and resources
- Need a suitable reward function (manually designed)
  - In many cases designing reward function difficult

Try to exploit other information in learning instead of or in addition to reinforcement learning

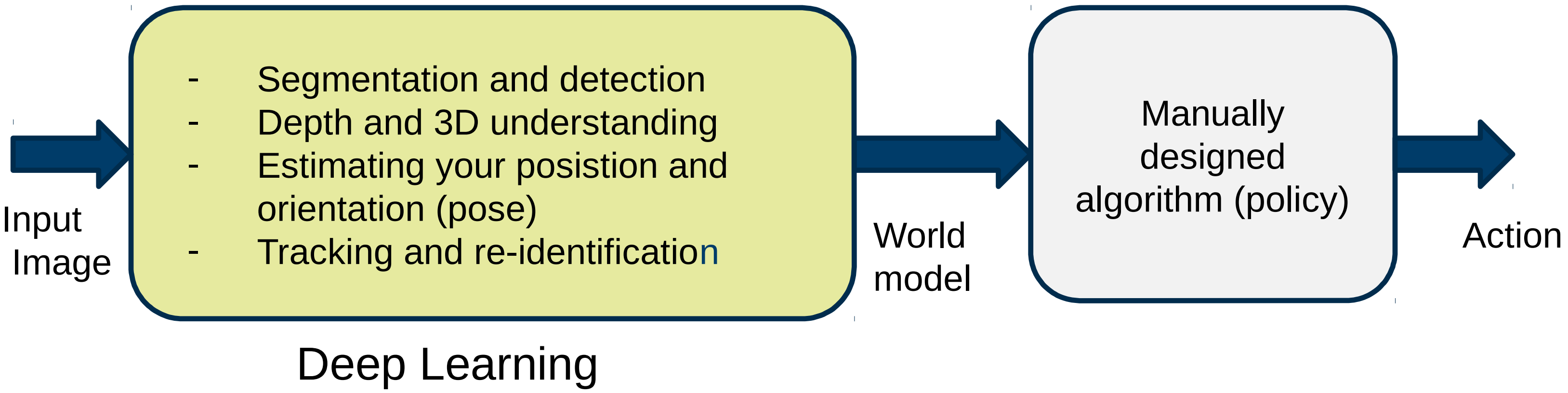
- Expert demonstrations
- Optimal control

Main Approaches

- **Manual design of actions (Learn perception only)**
  - Mediated Perception
  - Direct Perception
- **Learn actions (policy)**
  - Pure reinforcement learning
    - DQN (Deep Q-Network)
    - DDPG (Deep Deterministic Policy Gradient)
    - NAF (Normalized Advantage Function)
    - A3C (Asynchronous Advantage Actor Critic)
    - TRPO (Trust Region Policy Optimisation)
    - PPO (Proximal Policy Optimization)
    - ACKTR (Actor Critic Kronecker Factored Trust Region)
  - Optimal control and reinforcement learning
    - GPS (Guided Policy Search)
  - Pure expert demonstration based learning
    - Behavior cloning/Behavioural reflex
  - Combined expert demonstration and reinforcement learning
    - Maximum entropy deep Inverse reinforcement learning
    - Guided Cost Learning (GCL)
    - Generative Adversarial Imitation Learning (GAIL)

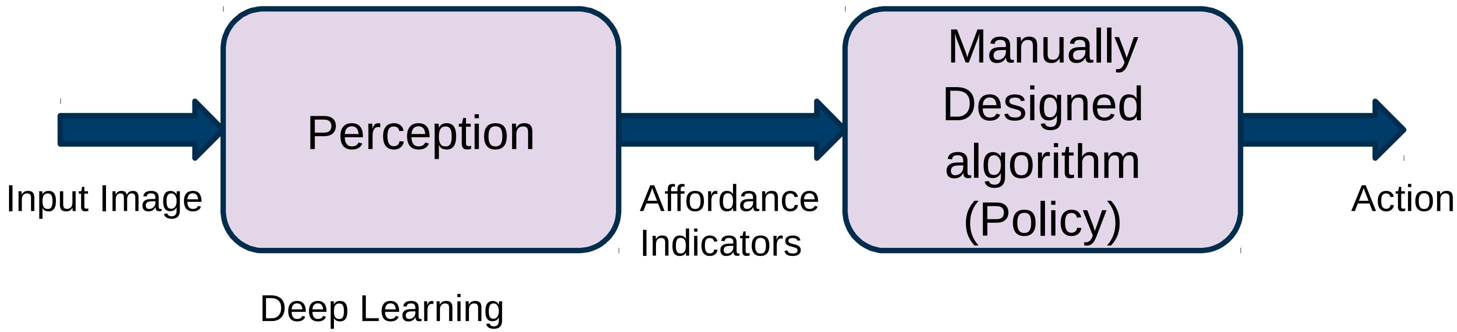
Manual Design of Control/Actions

Mediated Perception



Direct Perception

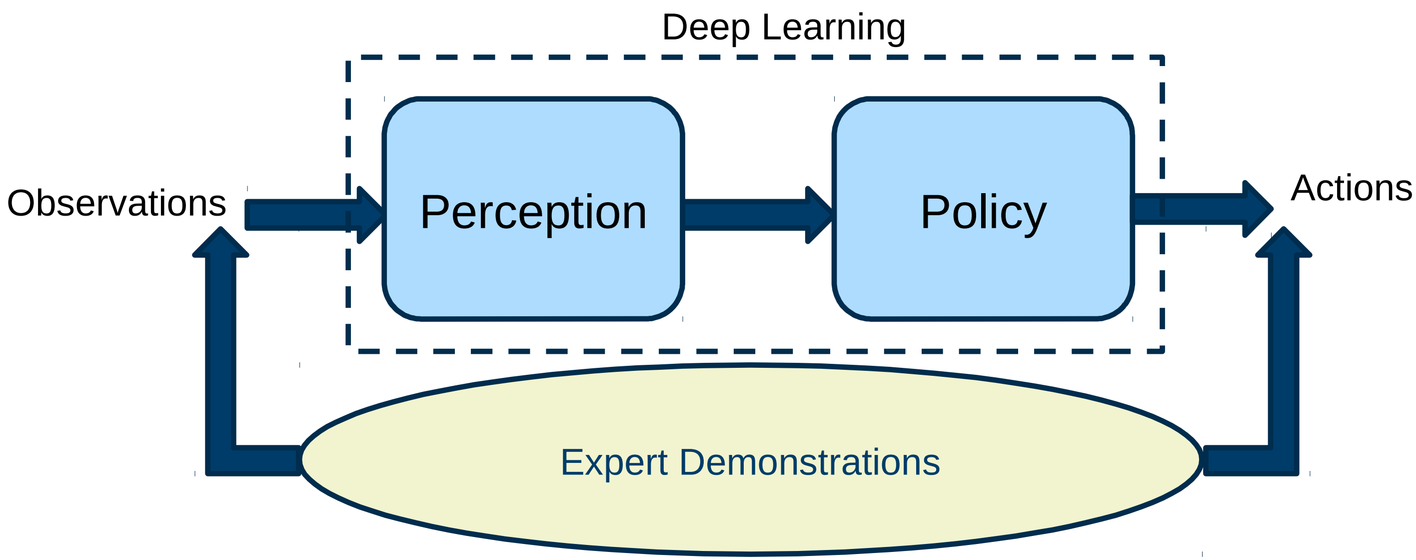
- Learn «Affordance Indicators» from input image
  - Eg: Distance to the left lane/right lane, distance to the next car
- Use a manually designed algorithm to convert affordance indicators to actions.



Expert Demonstrations Only

Behaviour Cloning

- A type of imitation learning
- Direct learning of the mapping between input observations and actions
- Supervised learning problem with training data given by the expert demonstrations
- Mostly applied in controlling autonomous vehicles



Issues of Behavioral Cloning

- Compounding Errors
  - Due to supervised learning assuming iid samples

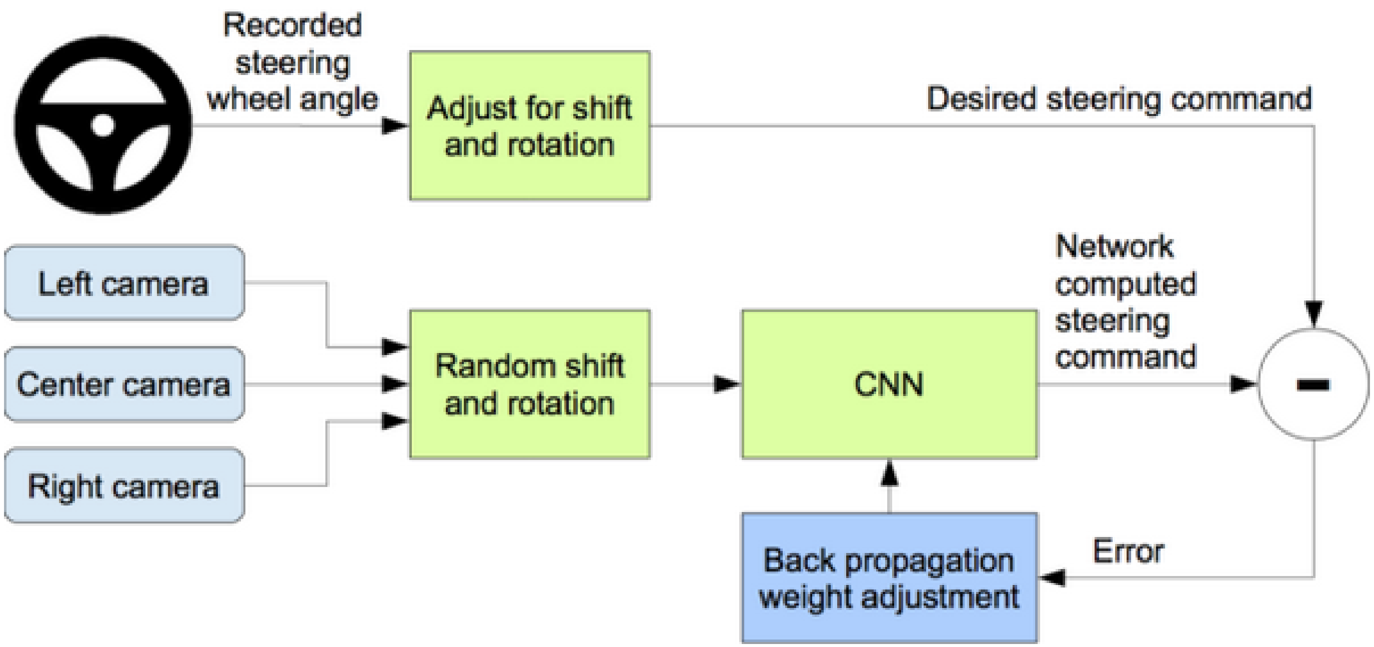
- Reactive Policies
  - Ignore temporal dependencies (long term goals are not considered)
  - Blind imitation of the expert demonstrations

DAgger (Dataset Aggegation)

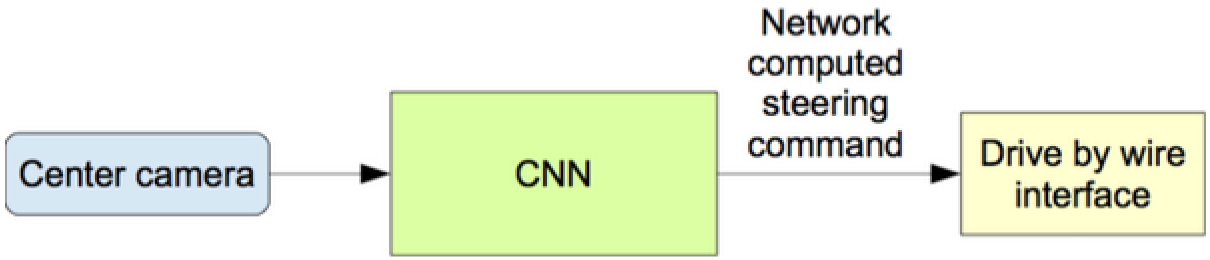
- Algorithm proposed to combat «compounding errors»
- Iteratively interleaves execution and training.

1. Use the expert demonstrations to train a policy
2. Use the policy to gather data
3. Label data using the expert
4. Add new data to the dataset
5. Train a new policy on new data (supervised learning)
6. Repeat from step 2

NVIDIA Deep Driving (Training)

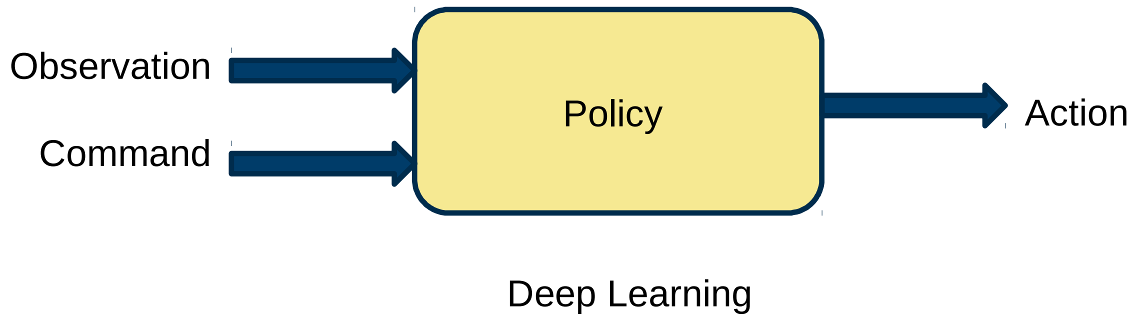


NVIDIA Deep Driving (Testing)



CARLA- Car Learning to Act

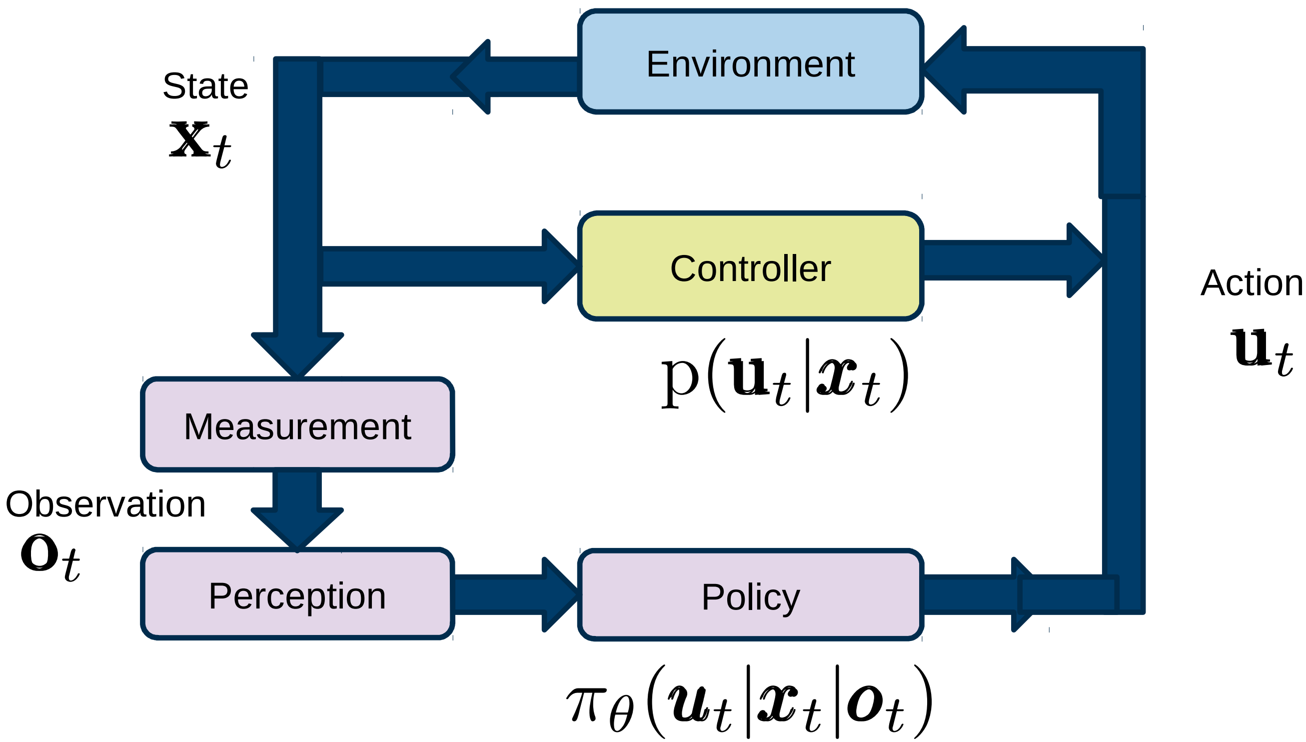
- Conditional Imitation Learning.
- More than driving straight
- Supervised training with expert demonstrations
  - Observertion = Forward Camera Image
  - Command = follow the lane, straight, left, right
  - Action= Steering parameters



Reinforcement Learning with Optimal Control

Guided Policy Search (GPS)

- Reinforcement learning algorithm
- Use optimal control to find optimal state-action trajectories
- Use optimal-state action trajectories to guide policy learning.



GPS Problem Formulation

- Consider an episode, of length T:

$$\tau = [\mathbf{x}_1, \boldsymbol{u}_1, \boldsymbol{x}_2, \boldsymbol{u}_2, \boldsymbol{x}_3, \boldsymbol{u}_3, \ldots, \boldsymbol{x}_T, \boldsymbol{u}_T]$$

- Controller $p(\boldsymbol{u}_t|\boldsymbol{x}_t)$ and environment dynamics $p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t)$ can define the trajectory τ
- Assume that each state-action pair is associated with a reward (cost) $c_t = c(\boldsymbol{x}_t, \boldsymbol{u}_t)$
- We want to optimize the total cost

$$c(\tau) = \sum_{t=1}^{T} c(\boldsymbol{x}_t, \boldsymbol{u}_t)$$

GPS Problem Formulation

- We want to optimize the total cost

$$c(\tau) = \sum_{t=1}^{T} c(\boldsymbol{x}_t, \boldsymbol{u}_t)$$

with respect to τ

- We also want that policy should give us the correct action:

$$\boldsymbol{u}_t = \pi_\theta(\boldsymbol{x}_t)$$

- We can formulate the problem with Lagrange multipliers

$$\min_{\tau,\theta} c(\tau) \text{ s.t. } \mathbf{u}_t = \pi_\theta(\mathbf{x}_t)$$

$$\bar{\mathcal{L}}(\tau, \theta, \lambda) = c(\tau) + \sum_{t=1}^{T} \lambda_t(\pi_\theta(\mathbf{x}_t) - \mathbf{u}_t)$$

How to Solve this Optimization?

$$\min_{\tau,\theta} c(\tau) \text{ s.t. } \mathbf{u}_t = \pi_\theta(\mathbf{x}_t)$$

$$\bar{\mathcal{L}}(\tau,\theta,\lambda) = c(\tau) + \sum_{t=1}^{T} \lambda_t(\pi_\theta(\mathbf{x}_t) - \mathbf{u}_t)$$

- Use dual gradient descent:

1. $\text{Find } \tau \leftarrow \arg\min_\tau \bar{\mathcal{L}}(\tau,\theta,\lambda)$
2. $\text{Find } \theta \leftarrow \arg\min_\theta \bar{\mathcal{L}}(\tau,\theta,\lambda)$
3. $\lambda \leftarrow \lambda + \alpha \frac{dg}{d\lambda} \qquad \text{where } g = \bar{\mathcal{L}}(\tau^\star,\theta^\star,\lambda)$
4. Repeat from 1

Dual Gradient Descent (DGD) Steps

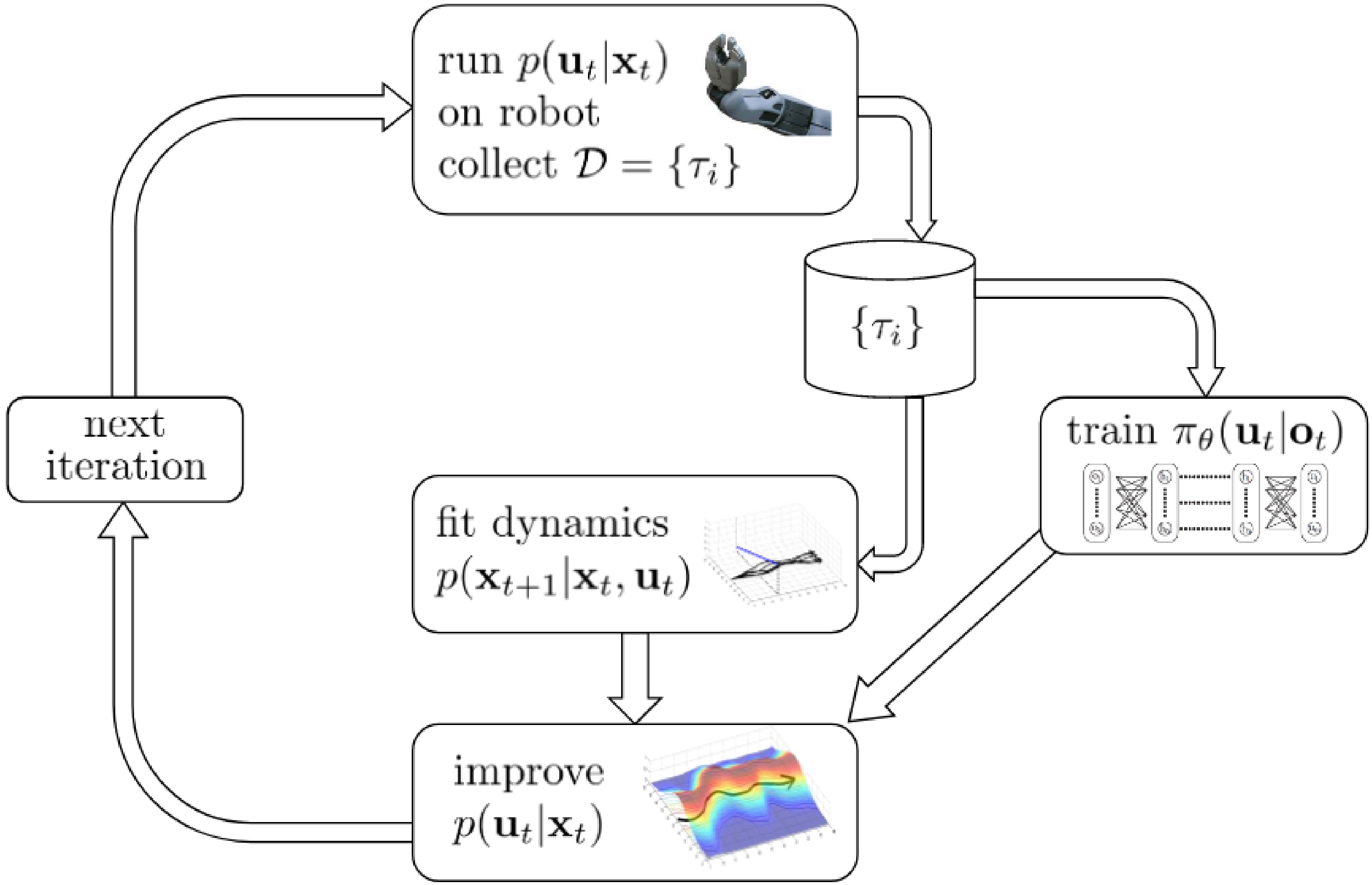
- Step 1: $\text{Find } \tau \leftarrow \arg\min_{\tau} \bar{\mathcal{L}}(\tau, \theta, \lambda)$
  - This is a typical optimal control problem.
  - Algorithms such as LQR (Linear Quadratic Regulator) can be used.
  - Using the current values of λ, θ we can find the optimal trajectory τ

- Step 2: $\text{Find } \theta \leftarrow \arg\min_{\theta} \bar{\mathcal{L}}(\tau, \theta, \lambda)$
  - Use the current values of τ, λ we will optimize

$$\bar{\mathcal{L}}(\tau, \theta, \lambda) = c(\tau) + \sum_{t=1}^{T} \lambda_t (\pi_\theta(\mathbf{x}_t) - \mathbf{u}_t)$$

  - This is just supervised learning $\sum_{t=1}^{T} \pi_\theta(\boldsymbol{x}_t) - \boldsymbol{u}_t$

GPS Summary



Reference: http://rail.eecs.berkeley.edu/deeprlcourse/static/slides/lec-13.pdf

Combining Reinforcement Learning with Expert Demonstrations

Inverse Reinforcement Learning (IRL)

- Motivation
  - In reinforcement learning, we assume that a reward/cost function is known (Manually designed reward function).
  - However, in many real world applications the reward structure is unclear.
  - In inverse reinforcement learning, we learn the reward function based on expert demonstrations.

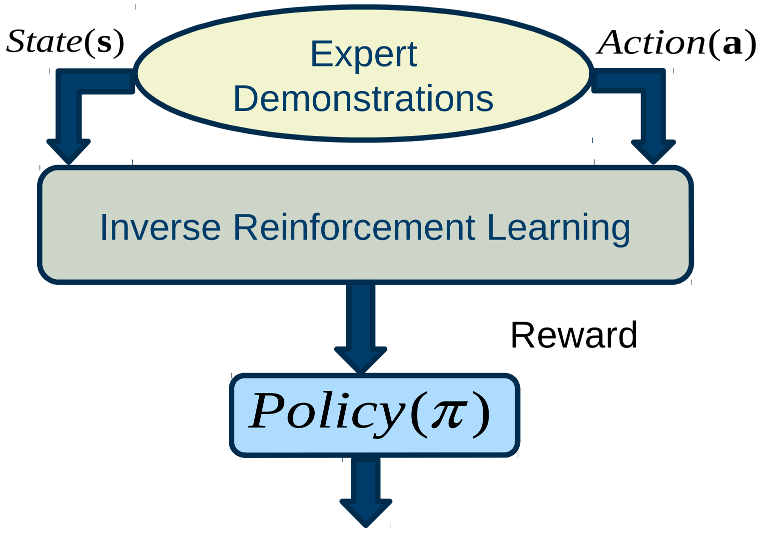
IRL vs. RL

- Reinforcement Learning (RL)
  - States $\boldsymbol{x}$ and actions $\boldsymbol{u}$ are drawn from a given set
  - Direct interaction with the environment or an environment model is known. $p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t)$
  - Reward function $r_\psi(\boldsymbol{x}, \boldsymbol{u})$ is known
  - Learn the optimal policy $\pi^\star(\boldsymbol{u}|\boldsymbol{x})$

- Inverse Reinforcement Learning (IRL)
  - States and actions are drawn from a given set
  - Direct interaction with the environment or an environment model is known
  - Expert demonstrations (state-action pairs generated by an expert) are given $\tau_i = [\mathbf{x}_1, \boldsymbol{u}_1, \boldsymbol{x}_2, \boldsymbol{u}_2, \boldsymbol{x}_3, \boldsymbol{u}_3, \ldots, \boldsymbol{x}_T, \boldsymbol{u}_T]$
  - Assume expert demonstrations are samples from an optimal policy
  - Learn the reward function $r_\psi(\boldsymbol{x}, \boldsymbol{u})$ and then optimal policy $\pi^\star(\boldsymbol{u}|\boldsymbol{x})$.

Challenges of IRL

- Ill-posed problem
- Expert demonstrations are not drawn from the optimal policy



Maximum Entropy IRL

- Trajectory $\tau_i = [\mathbf{x}_1^i, \boldsymbol{u}_1^i, \boldsymbol{x}_2^i, \boldsymbol{u}_2^i, \boldsymbol{x}_3^i, \boldsymbol{u}_3^i, \ldots, \boldsymbol{x}_T^i, \boldsymbol{u}_T^i]$
- Expert demonstrations $\mathcal{D} = \{\tau_i\}$
- Reward $R_\psi(\tau) = \sum_t r_\psi(\boldsymbol{x}_t, \boldsymbol{u}_t)$

- Define the probability of a given trajectory as

$$p(\tau) = \frac{1}{Z} \exp(R_\psi(\tau))$$

where

$$Z = \sum_{\tau \in \mathcal{D}_{all}} \exp(R_\psi(\tau))$$

- Objective of maximum entropy IRL is to maximize the probability of expert demonstrations with respect to ψ

$$\mathcal{L}(\psi) = \sum_{\tau \in \mathcal{D}} \log p_{r_\psi}(\tau)$$

Maxent IRL Optimization with Dynamic Programming

$$\max_{\psi} \mathcal{L}(\psi) = \sum_{\tau \in \mathcal{D}} \log p_{r_\psi}(\tau)$$
$$= \sum_{\tau \in \mathcal{D}} \log \frac{1}{Z} \exp(R_\psi(\tau))$$
$$= \sum_{\tau \in \mathcal{D}} R_\psi(\tau) - M \log Z$$
$$= \sum_{\tau \in \mathcal{D}} R_\psi(\tau) - M \log \sum_{\tau} \exp(R_\psi(\tau))$$

$$\nabla_\psi \mathcal{L}(\psi) = \sum_{\tau \in \mathcal{D}} \frac{dR_\psi(\tau)}{d\psi} - M \frac{1}{\sum_\tau \exp(R_\psi(\tau))} \sum_\tau \exp(R_\psi(\tau)) \frac{dR_\psi(\tau)}{d\psi}$$

Maxent IRL Optimization with Dynamic Programming

$$\nabla_\psi \mathcal{L}(\psi) = \sum_{\tau \in \mathcal{D}} \frac{dR_\psi(\tau)}{d\psi} - M \frac{1}{\sum_\tau \exp(R_\psi(\tau))} \sum_\tau \exp(R_\psi(\tau)) \frac{dR_\psi(\tau)}{d\psi}$$

- But by definition $\frac{1}{\sum_\tau \exp(R_\psi(\tau))} \exp(R_\psi(\tau)) = p(\tau)$
- Therefore the second term becomes $-M \sum_\tau p(\tau) \frac{dR_\psi(\tau)}{d\psi}$
- We can compute this at the state level, rather than at the trajectory level $-M \sum_{\boldsymbol{x}} p(\boldsymbol{x}) \frac{dr_\psi(\boldsymbol{x})}{d\psi}$
- We can use dynamic programming to calculate $p(\boldsymbol{x})$

Maxent IRL Optimization with Dynamic Programming

- We calculate $p(\boldsymbol{x})$ = probability of visiting state $\boldsymbol{x}$
- Assume probability of visiting state $\boldsymbol{x}$ at t=t is $p(\boldsymbol{x}, t) = \mu_t(\boldsymbol{x})$
- Then by the rules of dynamic programming

$$\mu_{t+1}(\boldsymbol{x'}) = \sum_{\boldsymbol{u}} \sum_{\boldsymbol{x}} \mu_t(\boldsymbol{x}) \pi(\boldsymbol{u}|\boldsymbol{x}) p(\boldsymbol{x'}|\boldsymbol{x}, \boldsymbol{u})$$

- Then $$p(\boldsymbol{x'}) = \frac{1}{T} \sum_t \mu_t(\boldsymbol{x'})$$

- This procedure is expensive if the number of states of the system is large.

Maxent IRL Optimization with Dynamic Programming

- The whole algorithm
  1. Gather demonstrations $\mathcal{D}$
  2. Initialize ψ
  3. Find the optimal policy $\pi(\boldsymbol{u}|\boldsymbol{x})$ with the reward function r_ψ (standard RL)
  4. Find state visitation frequency $p(\boldsymbol{x})$ (dynamic programming procedure)
  5. Compute gradient $\nabla_\psi \mathcal{L} = \sum_{\tau \in \mathcal{D}} \frac{dR_\psi}{d\psi}(\tau) - M \sum_{\boldsymbol{x}} p(\boldsymbol{x}) \frac{dr_\psi}{d\psi}(\boldsymbol{x})$
  6. Update ψ with gradient ascent
  7. Repeat from step 3

Maxent IRL Optimization with Sampling

- Dynamic programming approach not suitable for
  - Large state-spaces
  - Unknown dynamics

- The problem is the denominator (Partition function) Z

$$p(\tau) = \frac{1}{Z} \exp(R_\psi(\tau))$$

$$Z = \sum_{\tau \in \mathcal{D}_{all}} \exp(R_\psi(\tau))$$

- Use sampling to estimate Z instead of exact calculation: Guided Cost Learning (GCL).

Guided Cost Learning (GCL)

- Start with the log likelihood (per trajectory) of the expert trajectories $\mathcal{L}(\psi) = \frac{1}{N} \sum_{\tau \in \mathcal{D}} \log p_{r_\psi}(\tau)$
- Substituting $p(\tau) = \frac{1}{Z} \exp(R_\psi(\tau))$ we get $\mathcal{L}(\psi) = \frac{1}{N} \sum_{\tau \in \mathcal{D}} R_\psi(\tau) + \log Z$
- In notation used in paper ($\psi = \theta$ and $R = -c$), $\mathcal{L}_{\mathrm{IOC}}(\theta) = \frac{1}{N} \sum_{\tau_i \in \mathcal{D}_{\mathrm{demo}}} c_\theta(\tau_i) + \log Z$
- Partition function Z is given by $Z = \sum_{\tau \in \mathcal{D}_{all}} \exp(-c_\theta(\tau)) = \sum_{\tau \in \mathcal{D}_{all}} \exp(-c_\theta(\tau)) p(\tau)$ where $p(\tau)$ is a uniform distribution.
- Z is an expectation and therefore, we approximate Z by using M samples drawn from a proposal distribution $q(\tau)$

$$\mathcal{L}_{\mathrm{IOC}}(\theta) \approx \frac{1}{N} \sum_{\tau_i \in \mathcal{D}_{\mathrm{demo}}} c_\theta(\tau_i) + \log \frac{1}{M} \sum_{\tau_j \in \mathcal{D}_{\mathrm{samp}}} \frac{\exp(-c_\theta(\tau_j))}{q(\tau_j)}$$

Guided Cost Learning (GCL)

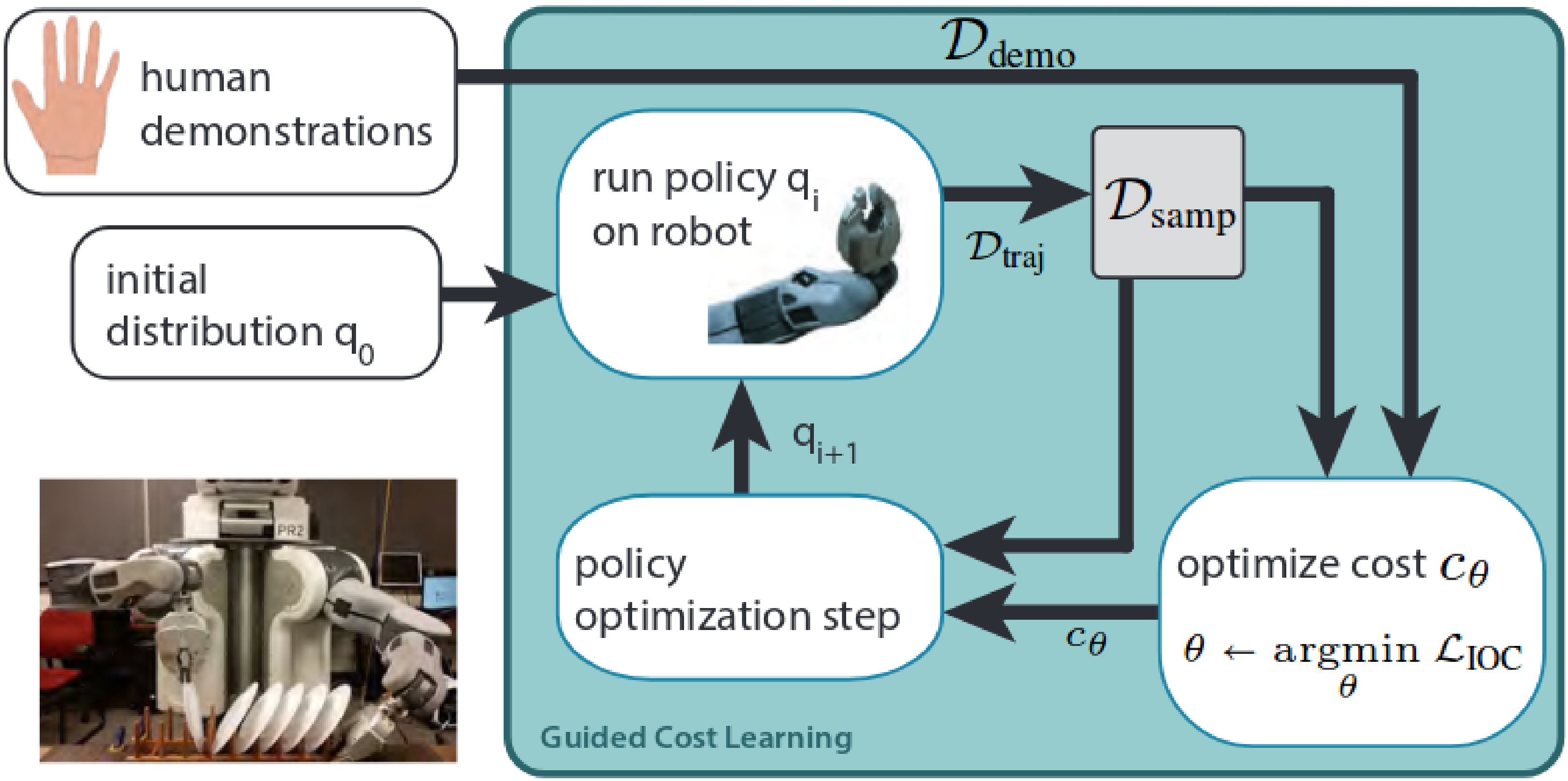
- We obtain gradients of $\mathcal{L}_{\mathrm{IOC}}(\theta)$ wrt θ

$$\frac{d\mathcal{L}_{\mathrm{IOC}}}{d\theta} = \frac{1}{N}\sum_{\tau_i \in \mathcal{D}_{\mathrm{demo}}} \frac{dc_\theta}{d\theta}(\tau_i) - \frac{1}{Z}\sum_{\tau_j \in \mathcal{D}_{\mathrm{samp}}} w_j \frac{dc_\theta}{d\theta}(\tau_j)$$

- Where $w_j = \frac{\exp(-c_\theta(\tau_j))}{q(\tau_j)}$ and $Z = \sum_j w_j$

- If $c_\theta(\tau)$ is implemented using a neural network we can back-propagate
  - $\frac{1}{N}$ If $\tau_i \in \mathcal{D}_{\mathrm{demo}}$
  - $-\frac{w_j}{Z}$ If $\tau_i \in \mathcal{D}_{\mathrm{samp}}$

Guided Cost Learning (GCL) Summary



Reference: https://arxiv.org/pdf/1603.00448.pdf

Guided Cost Learning (GCL) Summary

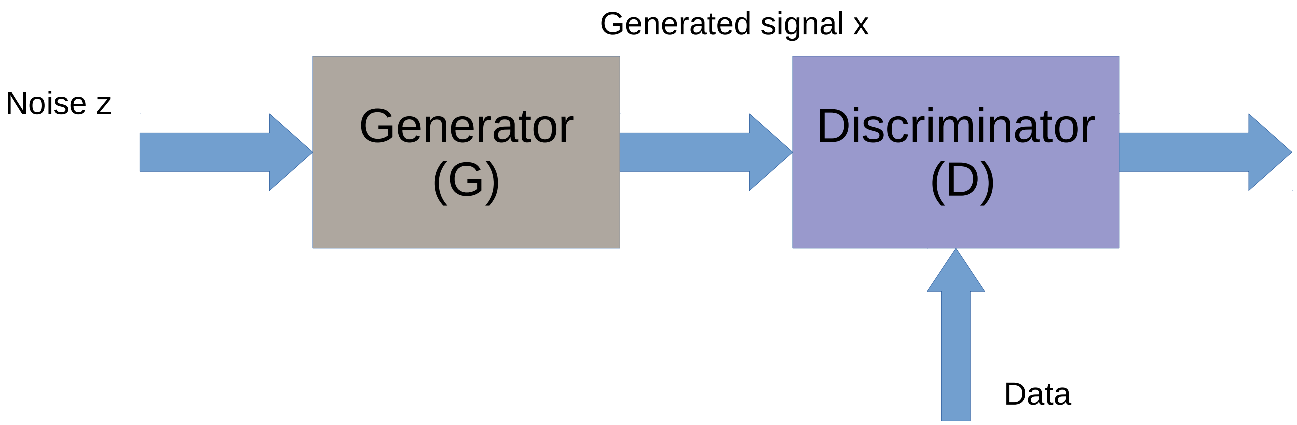
Algorithm 1 Guided cost learning

Initialize $q_k(\tau)$ as either a random initial controller or from demonstrations
for iteration $i = 1$ to I **do**
    Generate samples $\mathcal{D}_{\text{traj}}$ from $q_k(\tau)$
    Append samples: $\mathcal{D}_{\text{samp}} \leftarrow \mathcal{D}_{\text{samp}} \cup \mathcal{D}_{\text{traj}}$
    Use $\mathcal{D}_{\text{samp}}$ to update cost c_θ using Algorithm 2
    Update $q_k(\tau)$ using $\mathcal{D}_{\text{traj}}$ and the method from (Levine & Abbeel, 2014) to obtain $q_{k+1}(\tau)$
end for
return optimized cost parameters θ and trajectory distribution $q(\tau)$

Algorithm 2 Nonlinear IOC with stochastic gradients

for iteration $k = 1$ to K **do**
    Sample demonstration batch $\hat{\mathcal{D}}_{\text{demo}} \subset \mathcal{D}_{\text{demo}}$
    Sample background batch $\hat{\mathcal{D}}_{\text{samp}} \subset \mathcal{D}_{\text{samp}}$
    Append demonstration batch to background batch: $\hat{\mathcal{D}}_{\text{samp}} \leftarrow \hat{\mathcal{D}}_{\text{demo}} \cup \hat{\mathcal{D}}_{\text{samp}}$
    Estimate $\frac{d\mathcal{L}_{\text{IOC}}}{d\theta}(\theta)$ using $\hat{\mathcal{D}}_{\text{demo}}$ and $\hat{\mathcal{D}}_{\text{samp}}$
    Update parameters θ using gradient $\frac{d\mathcal{L}_{\text{IOC}}}{d\theta}(\theta)$
end for
return optimized cost parameters θ

Similarity to Generative Adversarial Networks (GANs)



$$\mathcal{L}_{\text{discriminator}}(D) = \mathbb{E}_{\mathbf{x}\sim p}[-\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{x}\sim G}[-\log(1 - D(\mathbf{x}))]$$

$$\mathcal{L}_{\text{generator}}(G) = \mathbb{E}_{\mathbf{x}\sim G}[-\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{x}\sim G}[\log(1 - D(\mathbf{x}))]$$

Similarity to Generative Adversarial Networks (GANs)

GCL	GAN
Trajectory τ	Sample x
Policy π	Generator G
Reward $r = -c$	Discriminator D
Expert demonstrations	Real data (eg: real images)

- It can be proved that generator and discriminator loss functions for the GCL have a similar form to those of GAN

Generative Adversarial Imitation Learning (GAIL)

- Very similar to GCL
- But does not aim to learn a reward function, instead it uses a classifier (discriminator)
- Trajectory samples are drawn using the TRPO (Trust Region Policy Optimization) algorithm

Algorithm 1 Generative adversarial imitation learning

1: **Input:** Expert trajectories $\tau_E \sim \pi_E$, initial policy and discriminator parameters θ_0, w_0
2: **for** $i = 0, 1, 2, \dots$ **do**
3: Sample trajectories $\tau_i \sim \pi_{\theta_i}$
4: Update the discriminator parameters from w_i to w_{i+1} with the gradient

$$\hat{\mathbb{E}}_{\tau_i}[\nabla_w \log(D_w(s,a))] + \hat{\mathbb{E}}_{\tau_E}[\nabla_w \log(1 - D_w(s,a))] \qquad (17)$$

5: Take a policy step from θ_i to θ_{i+1}, using the TRPO rule with cost function $\log(D_{w_{i+1}}(s,a))$. Specifically, take a KL-constrained natural gradient step with

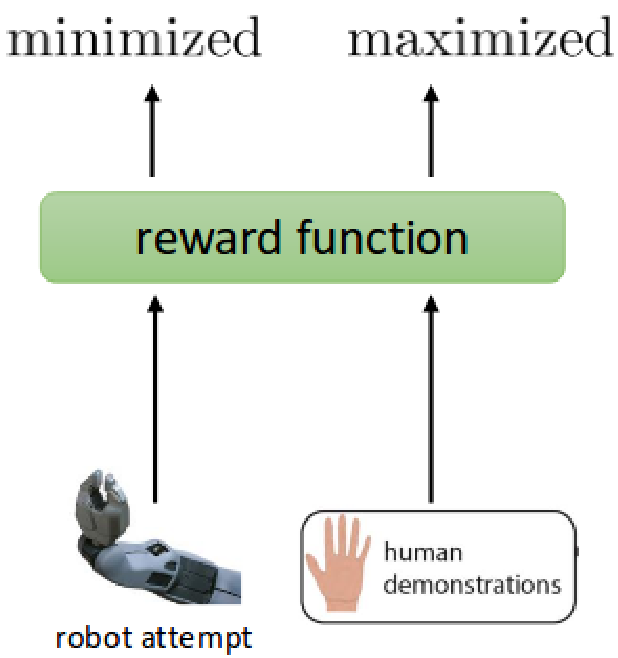
$$\begin{aligned} &\hat{\mathbb{E}}_{\tau_i}\left[\nabla_\theta \log \pi_\theta(a|s) Q(s,a)\right] - \lambda \nabla_\theta H(\pi_\theta), \\ &\text{where } Q(\bar{s},\bar{a}) = \hat{\mathbb{E}}_{\tau_i}[\log(D_{w_{i+1}}(s,a)) \mid s_0 = \bar{s}, a_0 = \bar{a}] \end{aligned} \qquad (18)$$

6: **end for**

GCL vs GAIL

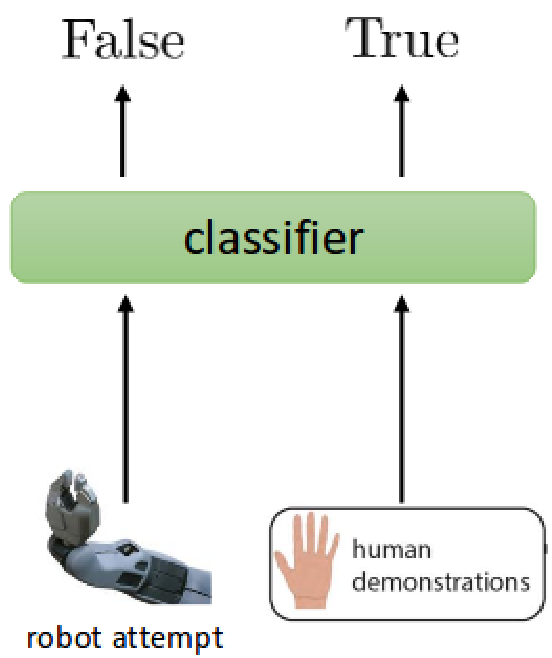
Guided Cost Learning

ICML 2016



Generative Adversarial Imitation Learning

Ho & Ermon, NIPS 2016



Reference: http://rail.eecs.berkeley.edu/deeprlcourse-fa17/f17docs/lecture_12_irl.pdf

Thank You