Apprenticeship Learning for Reinforcement Learning

with application to RC helicopter flight

Ritwik Anand, Nick Haliday, Audrey Huang

Table of Contents

- Introduction
  - Autonomous helicopter control problem
  - Reinforcement Learning
  - Apprenticeship Learning
- Theory
  - Learning dynamics [Abbeel, Ng 2005]
  - Learning reward function [Abbeel, Ng 2004]
  - Combining to get policy using control theory
- Putting into practice [Abbeel, Ng 2006]
- Conclusion

Autonomous Helicopter Flight

- Autonomous flight has important applications
  - Wildland fire fighting
  - Fixed-wing formation flight
- Helicopters have unique capabilities
  - In-place hover
  - Low-speed flight
- But it’s a challenging control problem
  - High-dimensional
  - Asymmetric
  - Noisy
  - Nonlinear, non-minimum phase dynamics

It's a Reinforcement Learning (RL) Problem

Learn an optimal decision policy by interacting with an environment and observing delayed or partial rewards.

- **Learner** (helicopter controller) performs **actions** (motion controls)
- Actions influence the **state** of the environment (helicopter)
  - Position
  - Orientation
  - Velocity
  - Angular velocity
- Environment returns **reward**
  - How closely we are following desired trajectory

Markov Decision Process (MDP)

Describes the reinforcement learning problem as a sextuple:

$$(\mathcal{S}, \mathcal{A}, \mathcal{T}, H, \mathcal{I}, \mathcal{R})$$

$$\mathcal{S} = \text{Set of states}$$
$$\mathcal{A} = \text{Set of actions}$$
$$\mathcal{T} = \text{Dynamics model}$$
$$H = \text{Time Horizon}$$
$$\mathcal{I} \subset \mathcal{S} = \text{Set of initial states}$$
$$\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R} = \text{Reward function}$$

Objective: Find Optimal Policy

- A policy π maps states $x \in S$ to actions $u \in A$
- Yields sum of rewards or value function $\mathbb{E}[\sum_{t=0}^{H} \mathcal{R}(x_t, u_t|\pi)]$

We want to:

- Find optimal policy π^* that maximizes value function:

$$\pi^* = \arg\max_\pi \mathbb{E}[\sum_{t=0}^{H} \mathcal{R}(x_t, u_t|\pi)]$$

MDP dynamics are unknown

$$(\mathcal{S}, \mathcal{A}, \mathcal{T}, H, \mathcal{I}, \mathcal{R})$$

- states (*S*) and actions (*A*) are physical properties of system, thus easily specified
- Time horizon (*H*) and initial state (*I*) and are given
- Assume for the moment that reward function (*R*) is known
- **Dynamics (*T*) must be learned**
  - Discrete
  - Continuous (linearly parameterized)

Discrete Dynamics

- Set of states *S* and actions *A* are **finite**
  - Games such as Go and Atari
- Discrete transition probabilities $P(x_{t+1} \mid x_t, u_t)$

Continuous Dynamics: Linearly Parameterized

- Set of states *S* and actions *A* are **continuous**
  - Suited for autonomous helicopter control problem
- Linearly parameterized dynamics given by:

$$x_{t+1} = A\phi(x_t) + Bu_t + w_t$$

  - $x_{t+1} \in S$ is next state
  - $x_t \in S$ is current state
  - Φ is feature mapping of state space
  - $u_t \in A$ is current action
  - w_t is process noise, I.I.D. multivariate Gaussian with known variance

We need to estimate coefficient matrices *A* and *B*.

Learning dynamics: Exploration vs Exploitation

Exploration

- Sufficiently visit all relevant parts of MDP
- Collect accurate transition probabilities

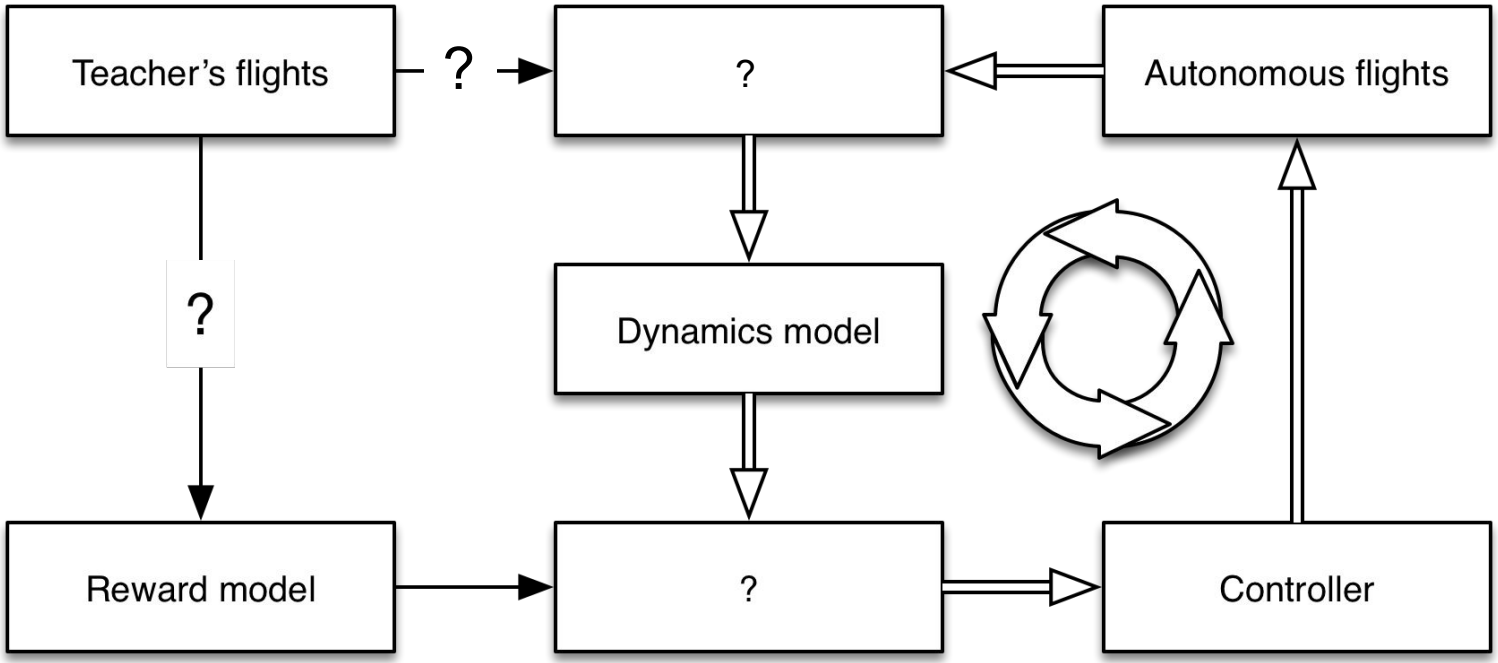
Exploitation

- Given current MDP, maximize sum of rewards over time

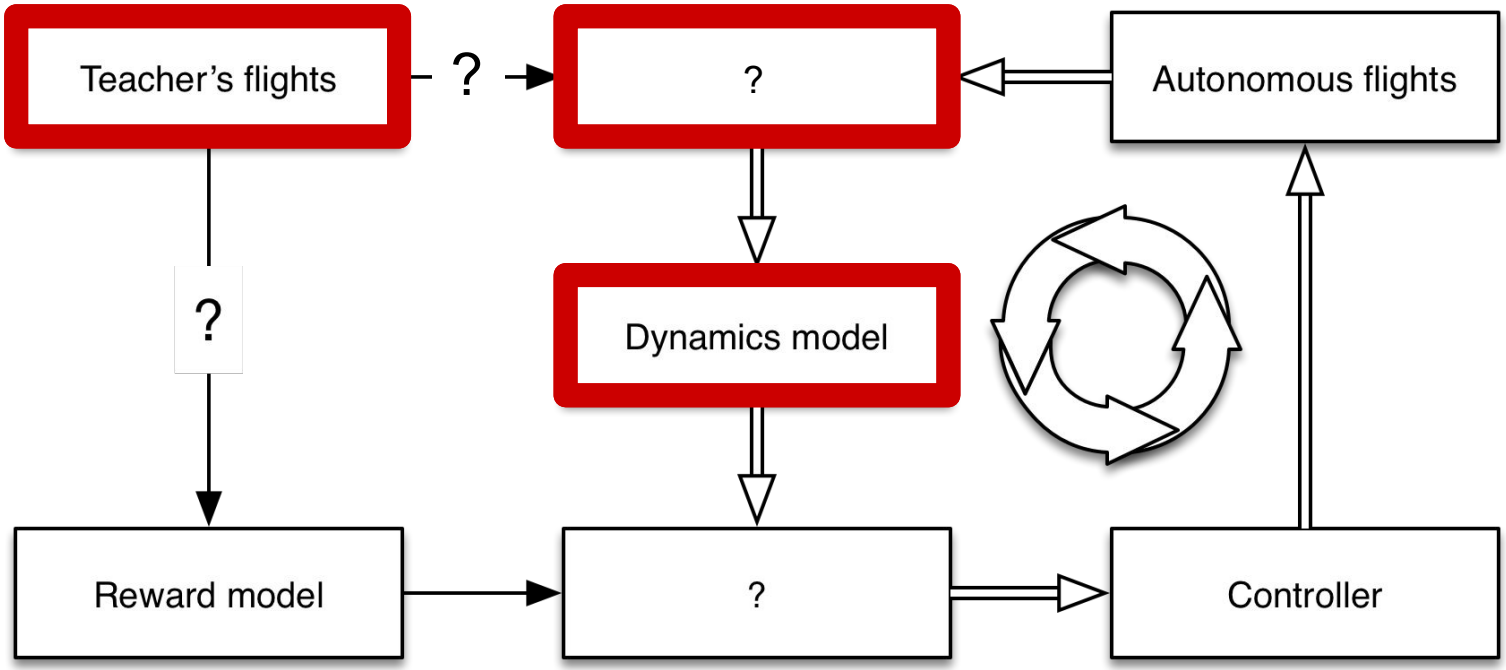
Exploration Policies Are Impractical

- Example exploration policy - **E^3 algorithm**
  - Strong bias for exploration policies
    - Generate accurate MDP model
  - Then uses exploitation policy to maximize reward
- Too aggressive for real system
  - Computationally intractable
  - Unsafe trajectories

Solution: Apprenticeship Learning



Learns MDP dynamics from initial expert data.



Apprenticeship Learning: MDP for Helicopters

$$(\mathcal{S}, \mathcal{A}, \mathcal{T}, H, \mathcal{I}, \mathcal{R})$$

1. Only **system dynamics (*T*)** are unknown.
2. But we do know system dynamics are **linearly parameterized by**:

$$x_{t+1} = A\phi(x_t) + Bu_t + w_t$$

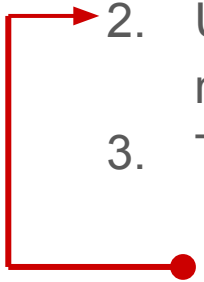
Apprenticeship Learning: High-Level Algorithm

INITIALIZE:

1. Expert demonstrates task. Record state-action trajectories.

MAIN LOOP:

2. Use all state-action trajectories thus far to learn a dynamics model. Find a near-optimal policy using RL algorithm.
3. Test policy on real system.
   a. If as good as expert, RETURN policy.
   b. If not as good, add state-action trajectories to training set. GO to #2.



Executes purely greedy, exploitative policies.

Algorithm: Details

Given: $\alpha > 0$, parameters N_E and k_1, and expert policy π_E.

INITIALIZE:

1. Run N_E trials under **expert policy π_E**.
2. Record state-action **trajectories $(x_{curr}, u_{curr}, x_{next})$**. Add to training set.
3. Compute average sum of rewards over all N_E trials, the **utility $U(\pi_E)$**.

Algorithm: Details

Given: $\alpha > 0$, parameters N_E and k_1, and expert policy π_E.

INITIALIZE:

1. Run N_E trials under **expert policy π_E**.
2. Record state-action **trajectories (s_{curr}, a_{curr}, s_{next})**. Add to training set.
3. Compute average sum of rewards over all N_E trials, the **utility $U(\pi_E)$**.

MAIN LOOP:

1. Use **regularized linear regression** on all state-action trajectories thus far to estimate A and B. Call the **new dynamics $T^{(i)}$**.
2. From new dynamics $T^{(i)}$, use RL to derive **optimal policy $\pi^{(i)}$**.
3. Run k_1 trials under $\pi^{(i)}$ to get new trajectories and **compute utility $U(\pi^{(i)})$**.
4. If $U(\pi^{(i)}) \geq U(\pi_E) - \alpha/2$, return $\pi^{(i)}$ and exit. Otherwise, **add new trajectories** to training set and loop again.

Algorithm: Regularized Linear Regression

Recall the linear dynamics predicts the next state using:

$$x_{t+1} = A\phi(x_t) + Bu_t + w_t$$

Then the *k*th rows of *A* and *B* are estimated by:

$$\underset{A_{k,:},B_{k,:}}{\arg\min} \sum_j \left(x_{next}^{(j)} - (A_{k,:}\phi(x_{curr}^{(j)}) + B_{k,:}u_{curr}^{(j)}) \right)^2 + \frac{1}{\lambda^2}(\|A_{k,:}\|_2^2 + \|B_{k,:}\|_2^2)$$

j indexes over all state-action-state triples $\{(x_{curr}^{(j)}, u_{curr}^{(j)}, x_{next}^{(j)})\}_j$ occurring after each other in the system's observed trajectories.

Algorithm: Analysis

Given dynamics $\mathcal{T}$ and $\alpha, \delta > 0$, then for

$$U(\pi) \geq U(\pi_E) - \alpha,$$ optimal policy

$$N = O(\text{poly}(\frac{1}{\alpha}, \frac{1}{\delta}, \mathcal{T}))$$ polynomial iterations

to hold with probability at least $1 - \delta$, it suffices that

$$N_E = \Omega(\text{poly}(\frac{1}{\alpha}, \frac{1}{\delta}, \mathcal{T}))$$ polynomial expert trials

$$k_1 = \Omega(\text{poly}(\frac{1}{\alpha}, \frac{1}{\delta}, \mathcal{T}))$$ polynomial policy trials

The returned policy will be approximately optimal after a polynomial number of iterations and trials N_E & k_1, with high probability.

High-Level Proof: Basis

Lemmas:

1. Simulation Lemma: Accurate transition probabilities are sufficient for accurate policy evaluation.

Facts:

1. After we have sufficient expert data, the estimated model is accurate for evaluating the utility of the expert's policy in every iteration.
2. Inaccurately modeled state-action pairs can only be visited a "small" number of times until all state-action pairs are accurately modeled.

High-Level Proof: Details

1. Consider policy $\pi^{(i)}$ in iteration i of the algorithm. **$\pi^{(i)}$ is the optimal policy for the current training set/model.**

High-Level Proof: Details

1. Consider policy $\pi^{(i)}$ in iteration *i* of the algorithm. We chose it because it's the optimal policy for the current training set/model.
2. **$\pi^{(i)}$ utility is valued as lower than the π_E's.**

*<u>Recall Termination Condition:</u> If **$U(\pi^{(i)}) \geq U(\pi_E) - \alpha/2$**, return $\pi^{(i)}$ and exit. Otherwise, add new trajectories to training set and loop again.*

High-Level Proof: Details

1. Consider policy $\pi^{(i)}$ in iteration *i* of the algorithm. We chose it because it's the optimal policy for the current training set/model.
2. $\pi^{(i)}$ utility is valued as lower than the π_E's. Fact #1 says $\boldsymbol{\pi_E}$ **must be evaluated correctly.**

Recall Fact #1: After we have sufficient expert data, the estimated model is accurate for evaluating the utility of the expert's policy in every iteration.

High-Level Proof: Details

1. Consider policy $\pi^{(i)}$ in iteration *i* of the algorithm. We chose it because it's the optimal policy for the current training set/model.
2. $\pi^{(i)}$ utility is valued as lower than the π_E's. Fact #1 says π_E must be evaluated correctly. **Then $\pi^{(i)}$ must be evaluated incorrectly.**

High-Level Proof: Details

1. Consider policy $\pi^{(i)}$ in iteration *i* of the algorithm. We chose it because it's the optimal policy for the current training set/model.
2. $\pi^{(i)}$ utility is valued as lower than the π_E's. Fact #1 says π_E must be evaluated correctly. Then $\pi^{(i)}$ must be evaluated incorrectly.
3. From the Simulation Lemma, **$\pi^{(i)}$ must be visiting inaccurately modeled state-action pairs.**

Recall Simulation Lemma: Accurate transition probabilities are sufficient for accurate policy evaluation.

Contrapositive: Inaccurate policy evaluation means inaccurate transition probabilities.

High-Level Proof: Details

1. Consider policy $\pi^{(i)}$ in iteration *i* of the algorithm. We chose it because it's the optimal policy for the current training set/model.
2. $\pi^{(i)}$ utility is valued as lower than the π_E's. Fact #1 says π_E must be evaluated correctly. Then $\pi^{(i)}$ must be evaluated incorrectly.
3. From the Simulation Lemma, $\pi^{(i)}$ must be visiting inaccurately modeled state-action pairs.
4. Fact #2 says **we will only visit inaccurate pairs a small number of times** until the dynamics are learned.

Recall Fact #2: Inaccurately modeled state-action pairs can only be visited a "small" number of times until all state-action pairs are accurately modeled.

High-Level Proof: Details

1. Consider policy $\pi^{(i)}$ in iteration *i* of the algorithm. We chose it because it's the optimal policy for the current training set/model.
2. $\pi^{(i)}$ utility is valued as lower than the π_E's. Fact #1 says π_E must be evaluated correctly. Then $\pi^{(i)}$ must be evaluated incorrectly.
3. From the Simulation Lemma, $\pi^{(i)}$ must be visiting inaccurately modeled state-action pairs.
4. Fact #2 says we will only visit inaccurate pairs a small number of times until the dynamics are learned.
5. **The iterations are bounded.**

Apprenticeship Learning: Review

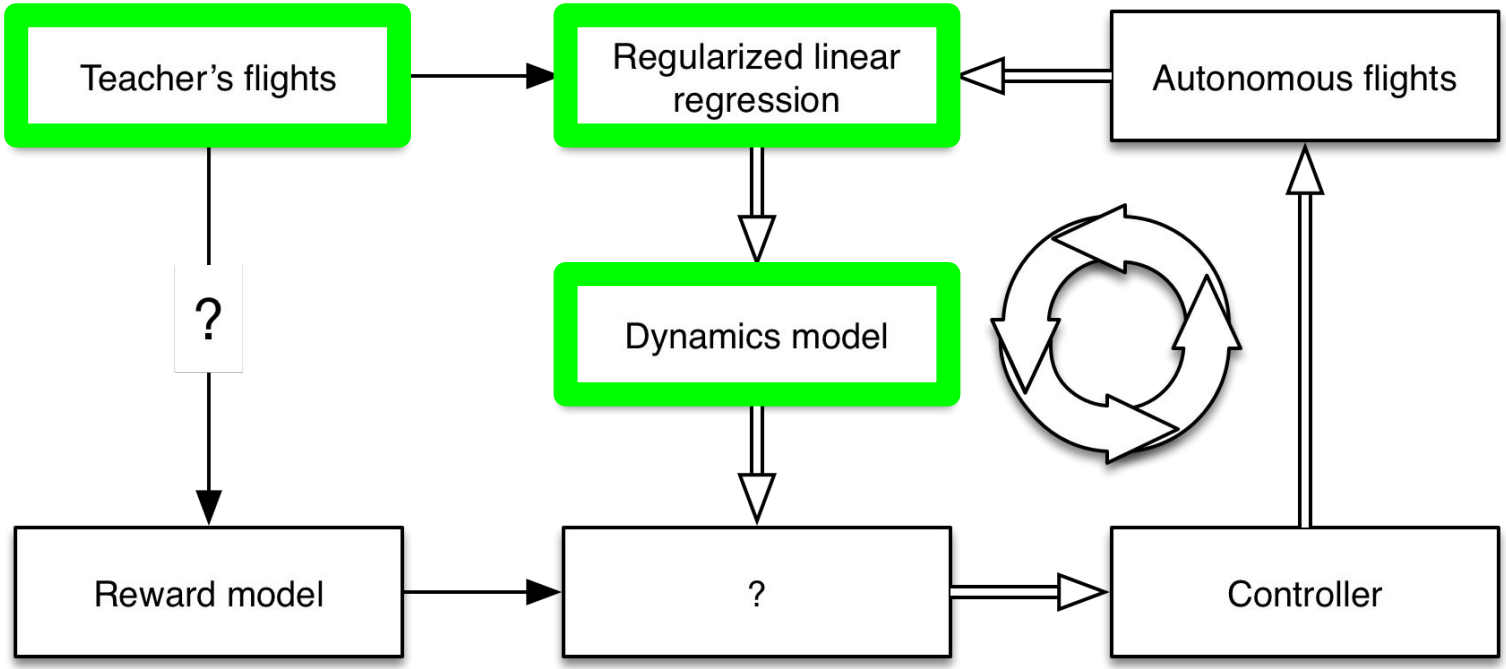
Applications:

- Find unknown dynamics for RL
- Exploration is risky
- Driving cars & aircraft

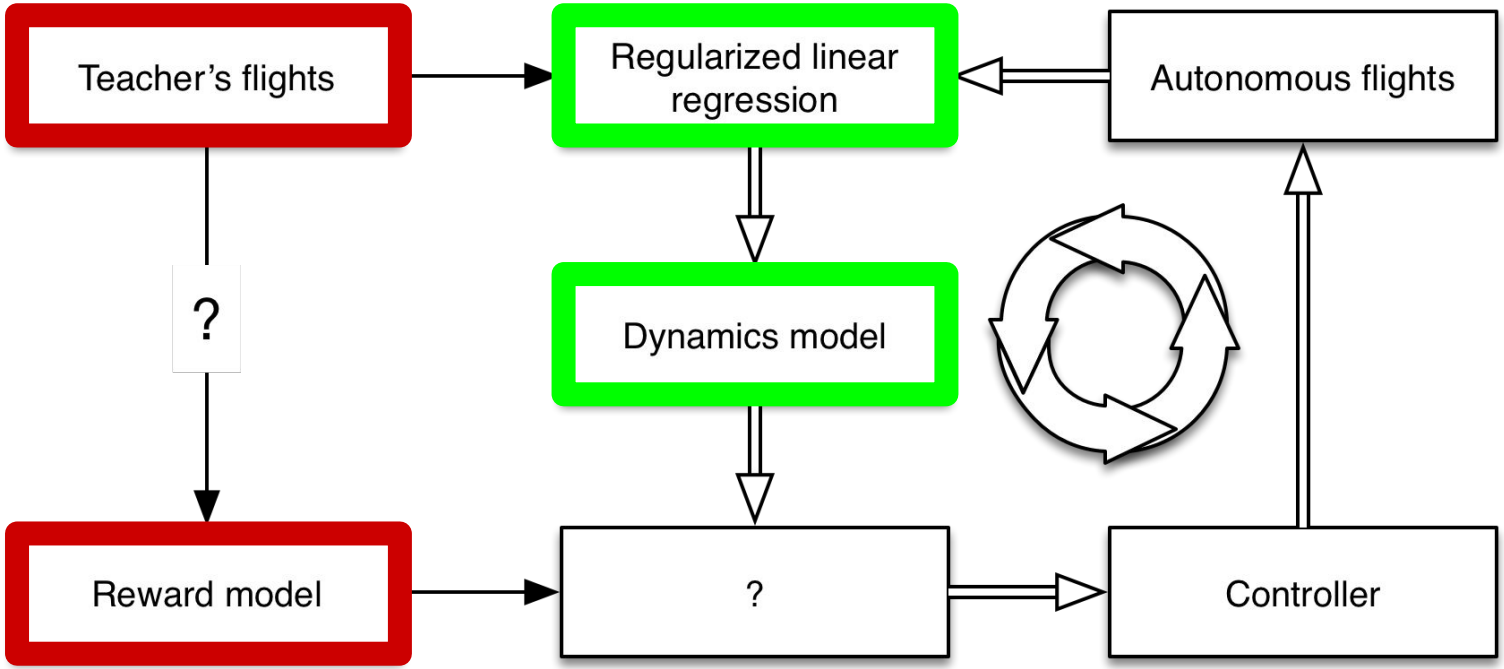
Algorithm:

- Learns dynamics from expert demo
- Greedy, exploitation-only policy improvements
- Polynomial iterations

Now we have a dynamics model.



How can we learn a useful reward function?



Roadmap

- Will solve the problem of deriving a reward function from expert behavior
- **Assume:** expert maximizes some unknown reward function
- **Assume:** reward function is linear combination of known features
- Algorithm indirectly updates a reward function at each iteration

Variables Used

- MDP defined by tuple (S, A, T, γ, D, R)
- S: finite set of states of the system
- A: set of actions which cause a transition from a state to state with a given distribution of probabilities
- T: is the set of state transition probabilities, when taking a given action in a specific state
- γ: is the discount factor
- D: initial state distribution
- Φ: some vector of features which maps a state to $[0, 1]^k$
- R: reward function linear w.r.t. Φ so $R = w^* \times \Phi(s)$ with weights $w^* \in R^k$
    - R is bounded by 1
    - $||w^*|| \leq 1$ for the bound to hold

More Variables

- A policy π is a mapping from states to probability distributions over actions
- The expected value of a policy π is :

$$E_{s_0 \sim D}[V^{\pi}(s_0)] = E[\sum_{t=0}^{\infty} \gamma^t R(s_t)|\pi]$$

$$= E[\sum_{t=0}^{\infty} \gamma^t w * \phi(s_t)|\pi]$$

$$= w * E[\sum_{t=0}^{\infty} \gamma^t \phi(s_t)|\pi]$$

- Where the expectation is taken with respect to a initial start state *s(0)* drawn from distribution *D* and picking actions according to policy π

Feature Expectations

- For a policy π, define $\mu(\pi)$ to be the feature expectations:

$$\mu(\pi) = E[\sum_{t=0}^{\infty} \gamma^t \phi(s_t)|\pi]$$

- R is a linear combination of the known features Φ
- $\mu(\pi)$ completely determine the expected sum of discounted rewards for acting according to π

Initial Simplification

- We assume we are given an Markov Decision Process without the reward function, a feature mapping, ϕ, π_E, $\mu(\pi_E)$
- We want to find a policy whose performance is close to that of the experts on the unknown reward function
- This is equivalent to finding π' s.t. :

$$|E[\sum_{t=0}^{\infty} \gamma^t R(s_t)|\pi_E] - E[\sum_{t=0}^{\infty} \gamma^t R(s_t)|\pi']| \leq \epsilon$$

Initial Simplification

- We use the fact that expectation is linear w.r.t. μ to get:

$$
\begin{aligned}
&E[\sum_{t=0}^{\infty} \gamma^t R(s_t)|\pi_E] - E[\sum_{t=0}^{\infty} \gamma^t R(s_t)|\pi'] \\
&= |w^T \mu(\pi') - w^T \mu_E| \\
&\leq ||w||_2 ||\mu(\pi') - \mu_E||_2 \\
&\leq 1 * \epsilon = \epsilon
\end{aligned}
$$

Reduced the problem to finding $\mu(\pi')$ close enough to $\mu(\pi_E)$

Inverse Reinforcement Algorithm

- Pick some arbitrary policy $\pi^{(0)}$, compute $\mu^{(0)} = \mu(\pi^{(0)})$ and set i = 1
- Compute $$t^{(i)} = \max_{w:\|w\|_2 \leq 1} \min_{j \in \{0,1,\ldots,i-1\}} w^T(\mu_E - \mu^{(j)})$$

- Let $w^{(i)}$ be the value of w that attains the max
- If $t^{(i)} \leq \epsilon$ we terminate
- Using the RL algorithm, compute the optimal policy $\pi^{(i)}$ for the MDP using the Reward function
- $R = w^{(i)T}\phi$
- Set i = i+1 $\mu^{(i)} = \mu(\pi^{(i)})$ and loop back to step 2
- Return the reward function
- Also generates a sequence $R = w^{(i)T}\phi$ esponding to maximizing the reward function at each iteration

Getting the Reward Function

- The maximization can be re-written as :

$$\begin{aligned} \max_{t,w} \quad & t \\ \text{s.t} \quad & w^T \mu_E \geq w^T \mu^{(j)} + t, \; j = 0, \ldots, i-1 \\ & \|w\|_2 \leq 1 \end{aligned}$$

- Thus we can see that the algorithm is trying to find a reward function $R = w^{(i)}\phi$ s.t. :

$$E_{s_0 \; D}[V^{\pi^{(i)}}(s_0)] + t \leq E_{s_0 \; D}[V^{\pi_E}(s_0)]$$

- Essentially a reward function on which the expert does better by a margin of t than any of the set of policies generated by the algorithm thus far

Assumptions of the Algorithm

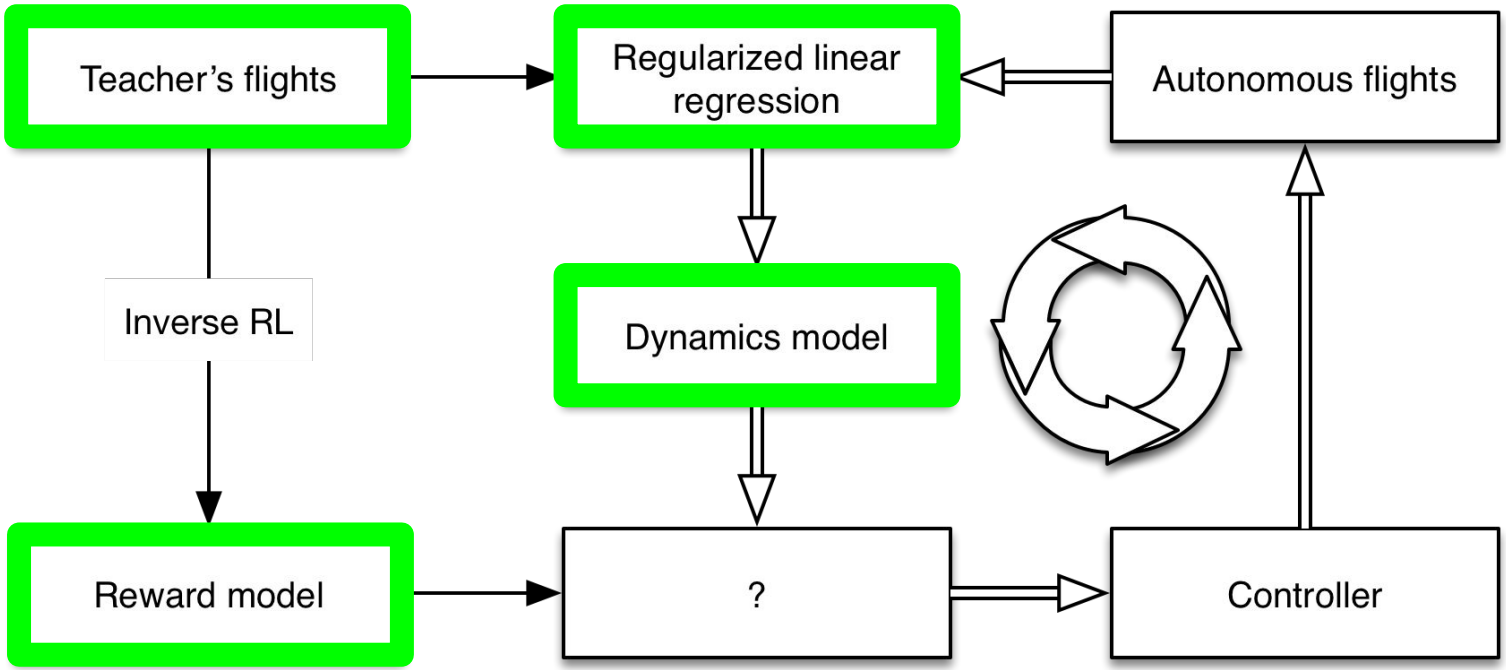
- One key assumption we made was that the algorithm does eventually terminate. From Theorem 1 [Abbeel, Ng 2004] we have that the number of steps before the algorithm terminates with $t^{(i)} \leq \epsilon$ is upper bounded by

$$O(\frac{k}{(1-\gamma)^2\epsilon^2}\log\frac{k}{(1-\gamma)\epsilon})$$

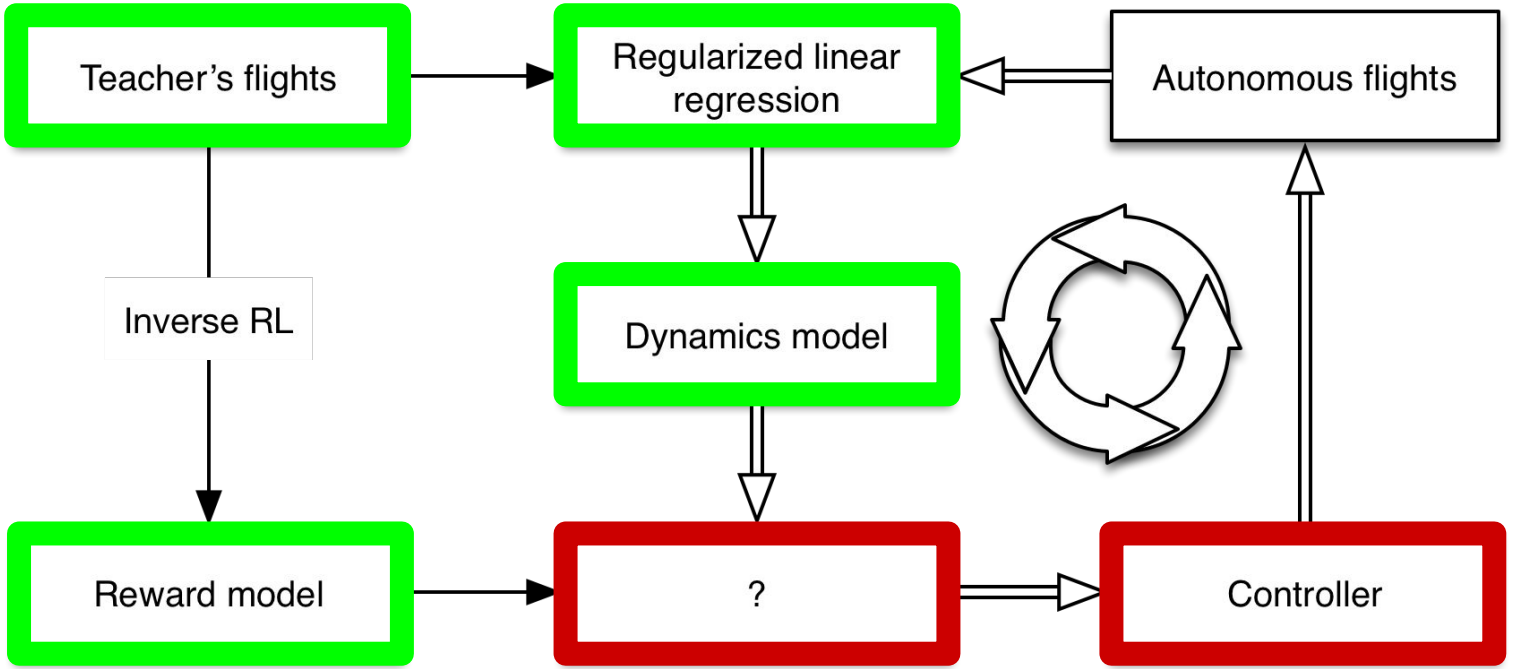
- We also assumed that the value μ_E was known
- In practice must be estimated from monte-carlo samples. Theorem 2 [Abbeel, Ng 2004] states that Theorem 1 holds with probability 1-δ if we use m trajectories as samples. Where

$$\frac{2k}{(\epsilon(1-\gamma))^2}\log\frac{2k}{\delta} \leq m$$

So far...


Teacher's flights
Regularized linear regression
Autonomous flights
Inverse RL
Dynamics model
Reward model
?
Controller

How do we get our policy?



How do we derive optimal actions to maximize reward?

Control Theory

- Once we know the reward function, *R*, and the dynamics, *f*, we are left with a dynamic programming problem.

$$\max_{u} \quad V_H(x,u) = \sum_{t=0}^{H} \mathcal{R}(x_t, u_t)$$
$$\text{s.t.} \quad x_{t+1} = f(x_t, u_t)$$

$$\max_{u} V_H(u) = \max_{u_H} \left(\mathcal{R}(x_H, u_H) + \max_{u_{0:H-1}} V_{H-1}(u_{0:H-1}) \right)$$

- Our reward function will be based on expert trajectory and two parameters (from inverse RL).
- Will build up solution from special case (LQR).

LQR: Linear dynamics with quadratic reward

- Suppose *f* is linear: $f(x, u) = Ax + Bu$
- And *R* is quadratic: $\mathcal{R}(x, u) = -x^T Q x - u^T R u, \text{ for } Q, R \succeq 0$
- Then...

LQR: Linear dynamics with quadratic reward

- We can define the *reward-to-go* function, V_t, and show via induction that it is quadratic and has simple to express policy (Riccati equation):

$$
\begin{aligned}
& & V_t(x) &= \sup_u(-x^TQx - u^TRu + V_{t+1}(Ax+Bu)) \\
V_{t+1}(x) = -x^TP_{t+1}x \implies & & V_t(x) &= \sup_u(-x^TQx - u^TRu - (Ax+Bu)^TP_{t+1}(Ax+Bu)) \\
& & u^* &= -(B^TP_{t+1}B+R)^{-1}B^TP_{t+1}Ax \\
& & V_t(x) &= -x^T\underbrace{(A^TP_{t+1}A + Q - (A^TP_{t+1}B)(B^TP_{t+1}B)^{-1}(B^TP_{t+1}A))}_{P_t \coloneqq}x
\end{aligned}
$$

- Find by setting gradient to zero (valid by concavity).
- Assuming Q positive definite (PD), R positive semidefinite (PSD), P_{t+1} PD, B full rank, everything is well-defined.
- P_t is PD by definition of V_t (negative for all nonzero x).

Time-dependence

- Same idea with time dependent dynamics and rewards (will need the former).

$$f_t(x, u) = A_t x + B_t u$$
$$\mathcal{R}_t(x, u) = -x^T Q_t x - x^T R_t x$$

- Just substitute time dependent parameters in place of *A, B, Q, R* in Riccati equation.
- Also fine if f_t has zero-mean noise term (maximizing expectation so don't care).

What about non-linear dynamics?

- Our dynamics were only linear parameterized, not linear.
- Instead of optimizing over x_t, u_t, optimize over the error with respect to target trajectory:

$$\bar{x}_t = x_t - x_t^*, \bar{u}_t = u_t - u_t^*$$

- Errors will be small, so we can approximate the dynamics to first order
- Makes sense to design a reward function around deviation from the teacher's trajectory anyway.

Nonlinear optimization problem

$$\begin{aligned} \max_{u} \quad & V = \sum_{t=0}^{H} -(x_t - x_t^*)^T Q (x_t - x_t^*) - (u_t - u_t^*)^T R (u_t - u_t^*) \\ \text{s.t.} \quad & x_{t+1} = f(x_t, u_t) \end{aligned}$$

Linearization

$$x_{t+1} \approx f(x_t^*, u_t^*) + \underbrace{\frac{\partial f}{\partial x}(x_t^*, u_t^*)}_{A_t}(x_t - x_t^*) + \underbrace{\frac{\partial f}{\partial u}(x_t^*, u_t^*)}_{B_t}(u_t - u_t^*)$$

$$\bar{x}_{t+1} = A_t \bar{x}_t + B_t \bar{u}_t$$

Linearized optimization problem

$$\max_{\bar{u}} \quad V = \sum_{t=0}^{H} -\bar{x}_t^T Q \bar{x}_t - \bar{u}_t^T R \bar{u}_t$$

$$\text{s.t.} \quad \bar{x}_{t+1} = A_t \bar{x}_t + B_t \bar{u}_t$$

Iterative LQR

- Problem:
  - We need the errors to be small.
  - In practice this might not be true. So linear approximation will not match actual dynamics well.
- The solution:
  - Linearize around current trajectory instead of target.
  - Still penalize distance from target.
  - Iteratively update toward target trajectory.

Iterative LQR

- Initialize with some guess of $u = \{u_0, \dots, u_H\}$
- With each iteration, given the current u:
  1. simulate the system to find x, using $x_{t+1} = f(x_t, u_t)$
  2. linearize around the current trajectory to obtain a linear system

$$\bar{x}_{t+1} = A_t \bar{x}_t + B_t \bar{u}_t \tag{1}$$

$$\text{where } A_t = \frac{\partial f}{\partial x}(x_t, u_t), B_t = \frac{\partial f}{\partial u}(x_t, u_t) \tag{2}$$

  3. solve time-varying LQR problem with cost

$$V = \sum_{t=0}^{H} (x_t + \bar{x}_t - x_t^*)^\intercal Q (x_t + \bar{x}_t - x_t^*) + (u_t + \bar{u}_t - u_t^*)^\intercal R (u_t + \bar{u}_t - u_t^*) \tag{3}$$

  4. update $u_t \coloneqq u_t + \bar{u}_t$ and repeat

Results

Attempted four difficult maneuvers

- Flip
- Roll
- Tail-in funnel
- Nose-in funnel

Flip and Roll

Flip: 360° rotation in place about lateral axis

- Initial cost matrices (*Q, R*) from algorithm oscillated
- Hand-tweaked matrices in simulator (following “spirit but not letter” of inverse RL algorithm).
- Eventually got a controller that could flip indefinitely.
- Penalty for changes in inputs over consecutive time steps was increased for final controller.

Roll: 360° rotation in place about longitudinal axis

- Same *Q, R* as flips

Tail-in Funnel and Nose-in Funnel

Tail-In Funnel: sideways medium-high speed circle, tail pointed towards center

- Repeatability (looping in place) is desired
- Autonomous funnels more accurate than human expert funnels.
    - error over course of 12 autonomous funnels same as error over 2 human expert funnels

Nose-In Funnel: sideways medium-high speed circle, nose pointed towards center

- Achieves same degree of increased accuracy as tail-in funnel

Video

Conclusion

- Can replace exploration with expert demonstrations and adjustments from repeated exploitation.
- Both reward and dynamics model can be learned from data.
  - Inverse RL
- Good policy via iterative LQR
- Good theoretical guarantees (polynomial).
- Good performance in practice (with some hand-tweaking).
- More details at http://heli.stanford.edu/.
- Simulator at https://sites.google.com/site/rlcompetition2014/domains/helicopter.

References & Questions

- P. Abbeel, A. Coates, M. Quigley, A.Y. Ng, An Application of Reinforcement Learning to Aerobatic Helicopter Flight, NIPS 2006.
- P. Abbeel, A.Y. Ng, Apprenticeship Learning via Inverse Reinforcement Learning, ICML 2004.
- P. Abbeel, A.Y. Ng, Exploration and Apprenticeship Learning in Reinforcement Learning, ICML 2005.
- Later papers:
  - A. Coates, P. Abbeel, A.Y. Ng, Learning for control from multiple demonstrations, ICML. (2008) 144–151. doi:10.1145/1390156.1390175.
  - P. Abbeel, A. Coates, A.Y. Ng, Autonomous Helicopter Aerobatics through Apprenticeship Learning, The International Journal of Robotics Research. 29 (2010) 1608–1639. doi:10.1177/0278364910371999.