**A ROBOTICS JOURNAL**

**General Terms**

**Inductive transfer learning** - ability of a learning mechanism to improve performance on the current or target task after having learned a different but related concept or skill on a previous source task.

**Transductive transfer learning** - denotes the transfer learning setting in which the only set of target documents that we are interested in classifying is known and available at training time.

**Deadreckon** – estimate future robot state using measured states at current time and letting them evolve forward in time

**Kalman filter** – A filter that uses joint probability distributions of sensor data to improve localization of the system.

**Particle filter** – used to estimate an autonomous vehicle pose (i.e. location and orientation) if we have a map of the environment. The filter operates by first generating a randomized distribution of possible robot poses and locations. Then, given a measurement using a sensor, it filters out the states which are unlikely to have returned this sensor data – so a model of the sensor is necessary here. The filtered particles are then upsampled to the original number of particles with original states acting as probability distributions from which new particle states are created. The robot then performs an action and using odometry data each of the states evolves to a new approximated states of the robot. These states are checked against the sensor data and filtered. The process continues until the true state of the robot is guessed.

**SLAM** – Simultaneous localization and mapping. Composed of two steps/problems solved simultaneously: 1) Localization: estimating where the robot is (i.e. Kalman filter / Particle filter) 2) Mapping: building a map of the environment. Pose graph optimization is the state of the art used in robotics to do this. We use optimization techniques that solve the SLAM problem.

**Safet - constrained learning** –

**Distributed learning –**

**Parameters –** descriptors of the environment that help in posing an optimization function.

**Models –** generalize parameters.

State equations

**State space of dynamic systems**

<https://x-engineer.org/state-space-model-dynamic-system/>

[**LQR**](https://jonathan-hui.medium.com/rl-lqr-ilqr-linear-quadratic-regulator-a5de5104c750)

The objective: Given initial state and final state at arbitrary time we find a sequence of controls by minimizing where is the dynamics model (basically A matrix from state equations).

The cost function is typically:

The control gain K from full state feedback , and with the above cost function can we solved to be: where is a solution to algebraic Ricatti equation:

Note that under full state feedback control, the dynamics of the system reduce to where the open loop state equations are: and

Once we compute using solution to Ricatti eqn. we plug it into and find stable solutions

**Euler – Lagrange**

https://en.wikipedia.org/wiki/Inverted\_pendulum

**Robotic manipulation**

**Joint space** – the space in which joint angles live. Represented as **q**.

**Pseudo Inverse** – necessary for over-actuated systems.

Jacobian pseudo-inverse equation is:

**Jacobian** – a matrix which maps join space velocities onto end effector velocity, i.e. where is of the form

**Direct/Forward Kinematics** – a computational step which takes the form where theta is joint space and x is end effector space

* representations:
  + **Product of exponentials:** Final transformation T is expressed as , where M is the home positions of the end effector in the world coordinates, and S are joint twist angles, represented as and
  + are found by identifying the axis of rotation of a given frame in the coordinates of the previous. is found by finding the velocity of the link end in the coordinates of the previous frame.
  + **M**matrix is built by identifying how the end effector frame can be achieved from base frame.
  + Each screw vector **S** is computed by identifying the effect of joint motion in the base frame while the robot is in home position from which **M** was found.
  + Jacobian is found in the same manner as each of the **S** but arbitrary angles need to be imposed

**Inverse Kinematics** – a computational step which takes the form where theta is joint space and x is end effector space

* Problems in IK are :
  + Multiple/infinite solutions
  + No solutions
  + No analytical solutions
* Analytical solutions to exist only if number of constraints is the same as number of dof
* Numerical approaches are used when analytics solutions cannot be found
* **Jacobian transpose method:**
  + update rule 🡺
  + Pros: simple, no inversion
  + Cons: unpredictable joints, needs many iterations to converge when Jacobian entries are small
* **Pseudo Inverse method**
  + update rule 🡺 **OR**
  + Pros: second order method, thus shortest path in q-space and computationally fast
  + Cons: matrix inversion causes problems, unpredictable joint configurations
* **Pseudo inverse with explicit optimization criterion**
  + update rule 🡺
  + Pros: computationally fast and explicit control over arm configurations (pay attention not the additional term that relates to concurrent joint angles with respect to some fixed set)
  + Cons: Numerical problems are singularities
* **The extended Jacobian Method**
  + : operates by adding null space basis vectors to the cost functions
  + Pros: fast, numerically robust, conservative in control action
  + Cons: matrix inversion in SVD is necessary

[**Markov Decision Process**](https://en.wikipedia.org/wiki/Markov_decision_process) – a mathematical framework for modeling decision making.

**Reinforcement Learning** - a learning in which an agent is taught to make decisions as to maximize points earned from an environment.

**Value function –** estimate of the reward the agent will receive at the end of the episode starting from some state **s**.

**Q learning** – an algorithm which creates a state/action table, Q(s,a), from which optimal actions can be taken given the agent state.

* The table is populated by iteration on the Bellman’s equation: Q(s,a) = r +
* Where: r is immediate reward of taking an action, and is the largest believed reward that can be achieved in the next state after taking a given action and normalized by discount factor setting how much we want to value short versus long term rewards.

**Deep Q Networks** – an algorithm which replaces q