Robotics Notes

Huu Duc Nguyen M.Sc.

07 May 2022

Contents

Abbreviations

ML	Machine Learning
DL	Deep Learning
RL	Reinforcement Learning
prob.	probability
params.	parameters
info.	information
a.k.a.	also known as
func.	function
vs.	versus
DOF	degrees of freedom
EE	end-effector
D-H	Denavit–Hartenberg
KF	Kalman Filter
EKF	Extended Kalman Filter
IF	Information Filter
EIF	Extended Information Filter
MHEKF	Multi-Hypothesis Extended Kalman Filter
MDP	Markov Decision Process
POMDP	Partially Observable Markov Decision Process

1 Introduction

This is my personal learning notes for robotics.

1.1 Learning Resources

- Springer Gandbook of Robotics [SKK08]
- Probalistic Robotics [TBF06]

[TODO:]

2 Robotics Foundations

2.1 Introduction

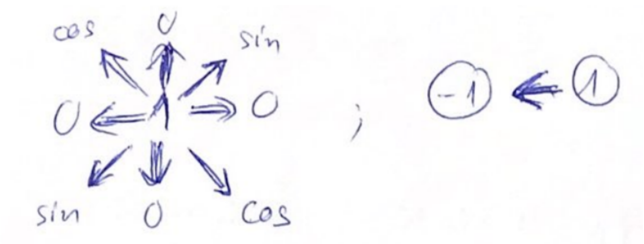
Types of robots:

Types	Articulated robot	Gantry	Scara	Delta	Hexapod
Workspace	Large			Small	Small
Structure	6 rotational	3 linear	1 translational, 2-3 parallel rotational	3 parallel kinematics chains	6 parallel kinematics chains
Market share	68%	19%	11%	1%	1%

2.2 DH Convention

Denavit–Hartenberg (D-H) Convention: [**TODO:**]

Elementary rotation matrices:



2.3 Position & Orientation

- Euler angles: **12** different sets of angles $R_{xy'z''}(\Phi) = R_z(\varphi)R_{y'}(\theta)R_{z''}(\psi)$
- Roll-Pitch-Yaw: fixed frame XYZ $R(\varphi, \theta, \psi) = R_z(\varphi)R_y(\theta)R_x(\psi)$
- Gimbal lock: loss of 1 degrees of freedom (DOF) in a 3-dimensional, 3 gimbal mechanism when 2/3 gimbals are driven into a parallel configuration
  NOTE: It exists no matter what rotation sequence you choose

- Angle & axix representation: solves gimbal lock problem
  However, does not guarantee an unique solution. When doing inverse, has problem when $\sin\theta = 0$
- Unit quaternion: ***CAN*** describe any rotation without ambiguities!

Intuition: angle & axis representation, unit quaternion are ways to avoid dealing with/or deal better with gimbal lock

2.4 Actuators and Joints

2.5 Workspace

2.6 Kinematics

2.6.1 Direct Kinematics

- Homogeneous coordinate representation, transformation matrix
- $\vec{e}_{x,{}^iO} = \vec{e}_{z,{}^{i-1}O} \times \vec{e}_{z,{}^iO}$

2.6.2 Inverse Kinematics

- Kinematic redundancy: No $\rightarrow$ Functional $\rightarrow$ hyper
- 3 ways to solve:
  - Analytical: if analytical solution exists, ***ALWAYS*** use it. Non redundant
  - Numerical: If closed-form solution does not exist / too hard to find analytically. If redundant
    $$err = r_d - f(q_e)$$
    $$q_{k+1} = q_k + \frac{1}{J_A(q)} adj(J_A(q)) err_k$$
  - Algebraic: solve polynomials equations

2.6.3 Differential Kinematics

$\boxed{\text{joint velocity} \Rightarrow \text{end-effector (EE) velocity}}$

Using Geometric Jacobian

$$q = [q_1 \dots q_n]^T \quad \text{– joint positions} \tag{2.1}$$

$$v_E = \begin{bmatrix} \dot{p}_E \\ \omega_E \end{bmatrix} \quad \text{– EE velocity} \tag{2.2}$$

$$\dot{p}_E = [\dot{p}_x \quad \dot{p}_y \quad \dot{p}_z]^T \quad \text{– EE linear velocity} \tag{2.3}$$

$$\omega_E = [\omega_x \quad \omega_y \quad \omega_z]^T \quad \text{– EE angular velocity} \tag{2.4}$$

$$\Rightarrow v_E = J_G(q).\dot{q} \tag{2.5}$$

$$J_G(q) = \begin{bmatrix} J_{GP}(q) \\ J_{GO}(q) \end{bmatrix} \quad \begin{array}{l} \text{–Geometric Jacobian} \\ J_{GP}(q) \text{ – geometric position Jacobian} \\ J_{GO}(q) \text{ – geometric orientation Jacobian} \end{array} \tag{2.6}$$

Using Velocity Composition Rule

- Linear velocity composition rule

$$^{O}\dot{r}_{p,{}^{0}O} = \tag{2.7}$$

- Angular velocity composition rule
- Strictly follow D-H conventions, then $\Rightarrow \delta, d, l, \alpha, \theta$
  Prismatic joints, revolute joints

2.6.4 Kinematics Singularities

Kinematics singularities occurs if the Jacobian is rank-deficient. There are 2 types:

- Boundary singularities: manipulator is out stretched or retracted $\Rightarrow$ does not drive the robot to the edge
- Internal singularities: caused by alignment of 2/more axes

Singularity affects the kinematics solution:

- Mobility of robotic structure is reduced
- Inverse kinematics $\Rightarrow$ yields infinite solutions
- Close to a singularity, small velocity in operational space **CAUSES** large velocity in joint space

[TODO: Singularities decoupling $\star$]

2.6.5 Kinematic Redundancy

2.6.6 Differential Mapping

2.6.7 Inverse Differential Kinematics

2.6.8 Statics

2.6.9 Manipulability Ellipsoids

2.7 Dynamics Model

2.8 Dynamic Parameter Identification

2.9 Direct & Inverse Dynamics

2.10 Parallel Kinematics

3 Trajectory Planning

3.1 Introduction

3.1.1 Definitions

Goal: compute a function of time that describes the motion of robot's joints. Depending on the task and time constraints, there will be different restrictions on the trajectory.

- A **Path** is a set of points in the joint/operational space, as geometrical description of the motion
- A **Trajectory** is a path through space as a function of time
  $\boxed{\textbf{Trajectory = Path + Time Law}}$

3.1.2 Joint space vs. Operational Space Trajectory Planning

- Joint space: planning is done in terms of joint positions, velocities and accelerations: $q, \dot{q}, \ddot{q}$. $\Rightarrow$ trajectory of the EE should NOT be important.
- Operational space: planning is done in terms of EE positions, orientation and their derivatives: $\mathbf{x}_E, \dot{\mathbf{x}}_E, \ddot{\mathbf{x}}_E$. This gives more control over the EE path, e.g., for melding, gluing applications.

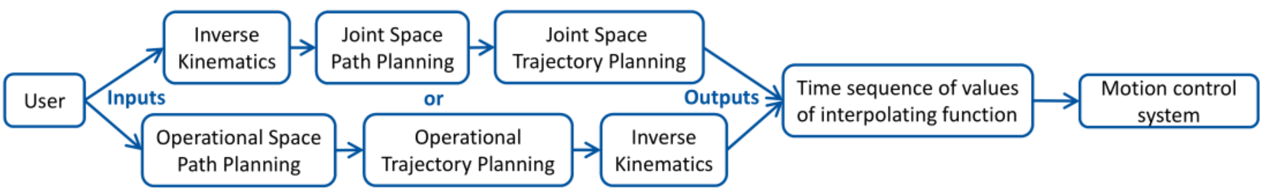


Figure 3.1: Trajectory planning procedure.

- Inputs: path description, geometric / kinematic constraints (usually in operational space), constraints resulting from the manipulator dynamics
- Outputs: A time sequence of values for the joints' positions, velocities and accelerations as needed by the motion control system
- The paths and trajectories are not defined by the user explicitly in each point, but as a set of relevant parameters (params.): start, end points, possible intermediate points, geometric primitives, total time, maximum accelerations and velocities

3.2 Joint Space Trajectories

Given start and end EE poses $\mathbf{P}_{start}, \mathbf{P}_{end}$ (and possible intermediate poses), apply inverse kinematics to get the joint poses (Fig. 3.1).

$$\left.\begin{matrix} \mathbf{P}_{start} \Rightarrow \mathbf{q}(t_{start}) \\ \mathbf{P}_{end} \Rightarrow \mathbf{q}(t_{end}) \end{matrix}\right\} \Rightarrow \mathbf{q}_t, \dot{\mathbf{q}}_t, \ddot{\mathbf{q}}_t \quad \text{for} \quad t \in [t_{start}, t_{end}]$$

3.2.1 Point-to-Point Motion

Possible ***timing laws*** for point-to-point motion:

- Cubic polynomial for the joints motion: can impose start, end positions and velocities

$$\begin{aligned} q_i(t) &= a_{i,3}t^3 + a_{i,2}t^2 + a_{i,1}t + a_{i,0} && \text{– cubic polynomial for joint } i \text{ motion} \\ \dot{q}_i(t) &= 3a_{i,3}t^2 + 2a_{i,2}t + a_{i,1} && \text{– parabolic velocity profile} \\ \ddot{q}_i(t) &= 6a_{i,3}t + 2a_{i,2} && \text{– linear acceleration profile} \end{aligned}$$

  Problem: infinite jerk at start and end positions

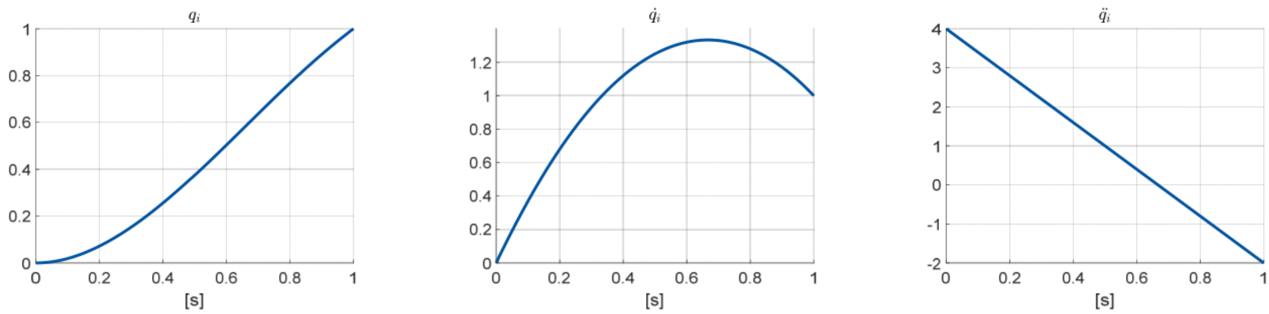


Figure 3.2: Example of cubic polynomial.

- Fifth order polynomial for the joints motion: allows imposing start, end positions, velocities and accelerations with finite jerk.

$$\begin{aligned} q_i(t) &= a_{i,5}t^5 + a_{i,4}t^4 + a_{i,3}t^3 + a_{i,2}t^2 + a_{i,1}t + a_{i,0} && \text{– fifth order polynomial} \\ \dot{q}_i(t) &= 5a_{i,5}t^4 + 4a_{i,4}t^3 + 3a_{i,3}t^2 + 2a_{i,2}t + a_{i,1} && \text{– forth order velocity profile} \\ \ddot{q}_i(t) &= 20a_{i,5}t^3 + 12a_{i,4}t^2 + 6a_{i,3}t + 2a_{i,2} && \text{– cubic acceleration profile} \end{aligned}$$

- Trapezoidal acceleration profile: given time period, max acceleration, velocity and position. **NOTE:** Just know that the below formulas and constraints exist. The derivation is

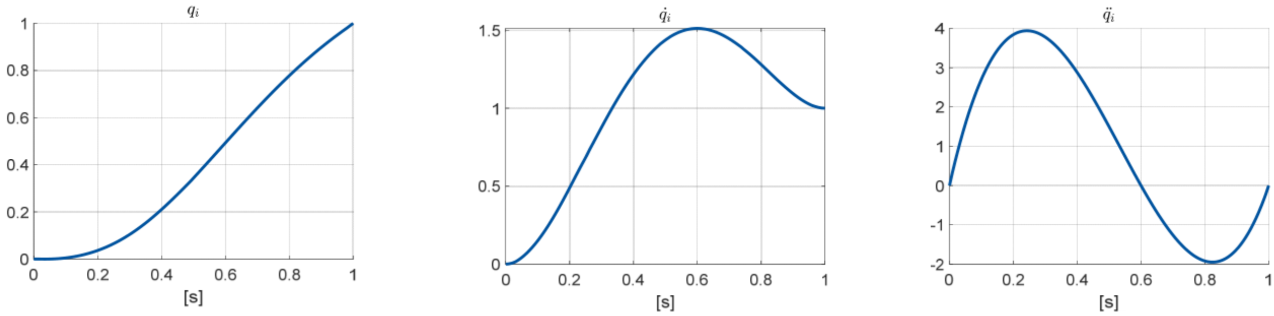


Figure 3.3: Example of fifth order polynomial.

complicated and time-consuming.

$$\Delta t_{i,I} = T - \frac{\dot{q}_{\max}}{\ddot{q}_{\max}} - \frac{q_{\max}}{\dot{q}_{\max}} \qquad \frac{q_{\max}}{T} \leq \dot{q}_{\max} \leq \frac{2q_{\max}}{T}$$

$$\Delta t_{i,II} = 2\frac{\dot{q}_{\max}}{\ddot{q}_{\max}} + \frac{q_{\max}}{\dot{q}_{\max}} - T \qquad \frac{\dot{q}_{\max}}{T - \frac{q_{\max}}{\dot{q}_{\max}}} \leq \ddot{q}_{\max} \leq \frac{2\dot{q}_{\max}}{T - \frac{q_{\max}}{\dot{q}_{\max}}}$$

$$\Delta t_{i,IV} = T - 4\Delta t_{i,I} - 2\Delta t_{i,II} \qquad \dddot{q}_{\max} = \frac{\ddot{q}_{\max}}{\Delta t_{i,I}} = \frac{\ddot{q}_{\max}}{T - \frac{\dot{q}_{\max}}{\ddot{q}_{\max}} - \frac{q_{\max}}{\dot{q}_{\max}}}$$

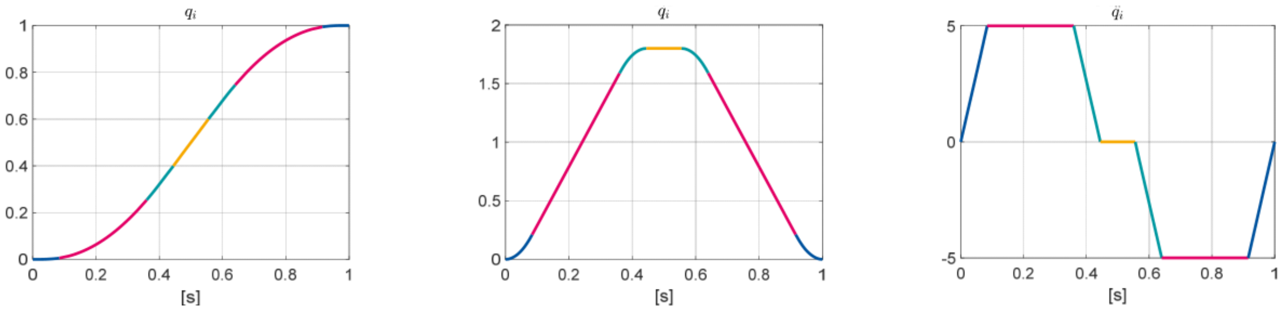


Figure 3.4: Example of trapezoidal acceleration profile.

3.2.2 Motion through a Sequence of Points

- This can be viewed as extension of point-to-point motion
- The path is defined through a **sequence of N path points**
- The time instances when these points must be reached should also be given
- Instead of using an $(N+3)$-order polynomial through N joint configurations, we can use low-order interpolating polynomials (Fig. 3.5).
  - Each joint variable q_i corresponds to a set of $N-1$ cubic polynomials $\Pi_{i,k}(t), k = 1, \ldots, N-1$

$$q_i(t_k) = q_{i,k} \qquad \text{– intermediate poses}$$
$$q_{i,1} = q_{i,start}, q_{i,N} = q_{i,end}$$
$$t_{i,1} = t_{i,start}, t_{i,N} = t_{i,end}$$
$$\Pi_{i,k}(t_k) = \Pi_{i,k+1}(t_k) = q_{i,k} \qquad \text{– path continuity}$$

- To impose **velocity continuity**, additional information is needed
  $$\dot{\Pi}_{i,k}(t_{i,k+1}) = \dot{\Pi}_{i,k+1}(t_{i,k+1}) = \dot{q}_i(t_{i,k+1})$$
  - Arbitrary values for the velocity at the path points $\dot{q}_i(t_k)$ **or**
  - Values for the velocity at the path points $\dot{q}_i(t_k)$ according to some criteria **or**
  - Impose continuous acceleration at the path points

  NOTE: Probably leads to acceleration jerks
- Continuity for acceleration: imposed by 4 equations
  NOTE: Required using 2 additional virtual points for 2 additional polynomials $\Rightarrow 4(N+1)$ equations for $4(N+1)$ unknowns. It is computationally expensive. With some tricks, the problem can be reduced to N variables ...

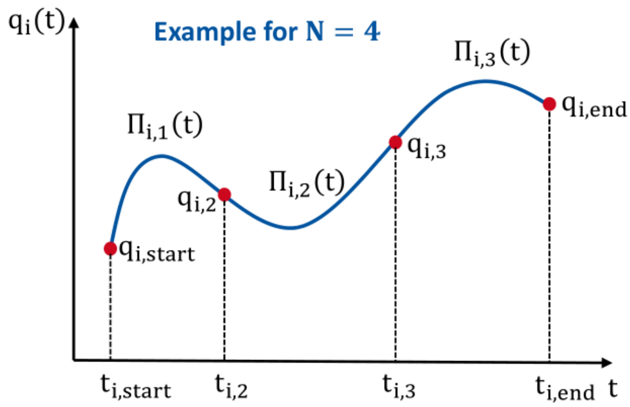


Figure 3.5: Example of motion through sequence of four points using low-order (cubic) interpolating polynomials.

3.3 Operational-space Trajectories

- Planning in the Joint Space may result in unexpected EE motion, resulting from the non-linear direct kinematics.
- Planning in Operational Space ensures that the EE motion follows exactly the geometrically specified path by
  - interpolating a sequence of prescribed poses
  - generating analytical motion primitives (e.g. lines, circular arcs etc.)
  - but it requires **real time** inverse kinematic computation, which is more computationally expensive

- $\boxed{\text{Path primitives + timing laws = trajectory planning}}$
  analyzed and computed separately

3.3.1 Path primitive

- The word *primitive* means a basic component, from which other is derived.
- **Path primitives** are the analytical geometrical descriptions for paths.
- Parametric representation of the path Γ in space is a function of a general parameter σ describe the position vector for point $\mathbf{P}$ (Fig. 3.6)

  $$\mathbf{r}_{\mathbf{P},{}^jO} = \mathbf{r}_\mathbf{P} = \mathbf{r}_\mathbf{P}(\sigma), \quad \sigma \in [\sigma_{start}, \sigma_{end}]$$

- Usually it makes sense to replace general params. σ by the current path length s

  $$\mathbf{r}_\mathbf{P}(s) = f(s) \quad \text{with} \quad \sigma_{start} = 0, \sigma_{end} = s_{end}$$

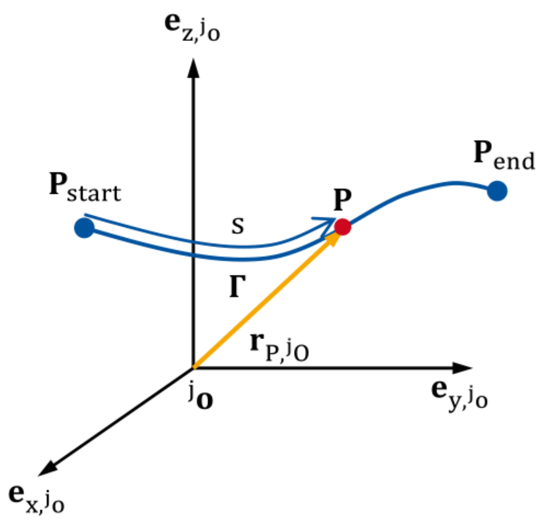


Figure 3.6: Example for parametric representation of path Γ

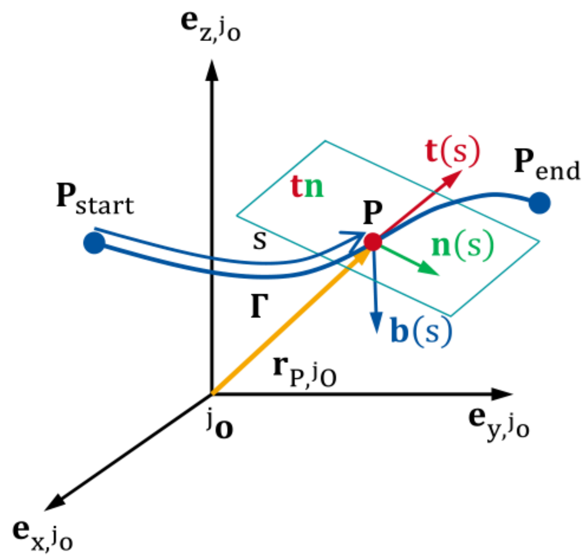


Figure 3.7: Frenet frame with 3 unit vectors.

3.3.2 Frenet Frame

Frenet Frame is used to define the coordinate orientation along the path Γ. The three unit vectors establish the Frenet frame (Fig. 3.7):

- Tangent unit vector $\mathbf{t}(s) = \dfrac{d\mathbf{r}_P(s)}{ds}$ along the direction of the path
- Normal unit vector $\mathbf{n}(s) = \dfrac{1}{\left\|\frac{d^2\mathbf{r}_P(s)}{ds^2}\right\|}\dfrac{d^2\mathbf{r}_P(s)}{ds^2}$ perpendicular to $\mathbf{t}(s)$
- Binormal unit vector $\mathbf{b}(s) = \mathbf{t}(s) \times \mathbf{n}(s)$ complements a right handed moving frame

3.3.3 Path Primitive: Rectilinear Path

The simplest parametric path representation is a linear segment between $\mathbf{P}_{start}$ and $\mathbf{P}_{end}$

$$\mathbf{r}_{P,{}^jO}(s) = \mathbf{r}_{P_{start},{}^jO} + \frac{s}{\left\|\mathbf{r}_{P_{end},{}^jO} - \mathbf{r}_{P_{start},{}^jO}\right\|}\left(\mathbf{r}_{P_{end},{}^jO} - \mathbf{r}_{P_{start},{}^jO}\right) \tag{3.1}$$

$$\mathbf{t}(s) = \frac{d\mathbf{r}_P(s)}{ds} = \frac{1}{\left\|\mathbf{r}_{P_{end},{}^jO} - \mathbf{r}_{P_{start},{}^jO}\right\|}\left(\mathbf{r}_{P_{end},{}^jO} - \mathbf{r}_{P_{start},{}^jO}\right) \tag{3.2}$$

$$\mathbf{n}(s) = \frac{1}{\left\|\frac{d^2\mathbf{r}_P(s)}{ds^2}\right\|}\frac{d^2\mathbf{r}_P(s)}{ds^2} \quad \text{with} \quad \frac{d^2\mathbf{r}_{P,{}^jO}(s)}{ds^2} = 0 \tag{3.3}$$

$\Rightarrow$ **CAN NOT** define the Frenet frame uniquely. The plane $\mathbf{tn}$ can be rotated arbitrarily around $\mathbf{t}(s)$

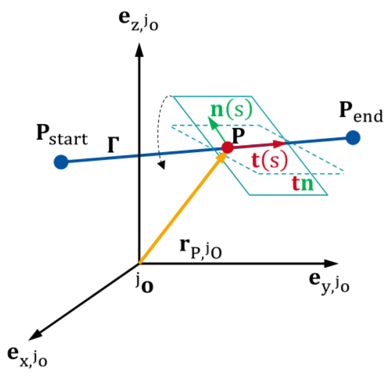
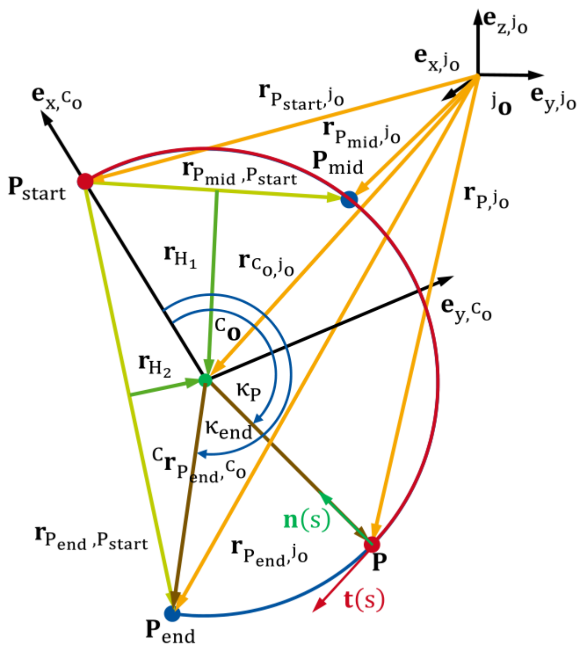


Figure 3.8: Rectilinear path primitive.

3.3.4 Path Primitive: Circular Arc

The circular arc is defined through three points $\mathbf{P}_{start}$, $\mathbf{P}_{mid}$ and $\mathbf{P}_{end}$.

- Compute center ${}^j\mathbf{r}_{C_O,{}^jO}$
  - Determine ${}^j\mathbf{r}_{P_{mid},P_{start}}$
  - Determine ${}^j\mathbf{r}_{P_{end},P_{start}}$
  - Determine unit vector ${}^j\mathbf{e}_{z,{}^CO}$
  - Determine helper vectors $\mathbf{r}_{H_1}, \mathbf{r}_{H_2}$
  - Determine ${}^j\mathbf{r}_{C_O,{}^jO}$
- Determine unit vectors ${}^j\mathbf{e}_{x,{}^CO}, {}^j\mathbf{e}_{y,{}^CO}$
- Determine radius
- ...



3.3.5 Path Primitive: Spline Path

Given a set of points $\mathbf{P}_i$ for $i \in \{1, n\}$, a smooth curves is generated by interpolating method

- Between two consecutive points $\mathbf{P}_i$ and $\mathbf{P}_{i+1}$, there is path segment $\mathbf{S}_i(\lambda)$
- To ensure smoothness between path's segments, the path must be continuous up to its second derivative at path points
- Use polynomials of degree five

$$\mathbf{S}_i(\lambda) = a_{i,5}\lambda^5 + a_{i,4}\lambda^4 + a_{i,3}\lambda^3 + a_{i,2}\lambda^2 + a_{i,1}\lambda + a_{i,0}, \qquad 0 \le \lambda \le \lambda_i$$

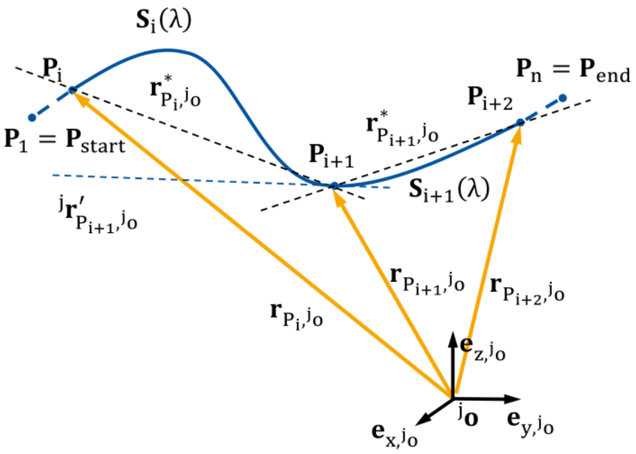


Figure 3.9: Spline path primitive.

3.4 Nonlinear Dynamics Motor Primitives

This *nonlinear dynamics motor primitives* are NOT the same as path primitives mentioned above. Let's examine two second-order dynamic systems (z, y) and (v, x) as follows: [INS02]

$$\dot{z} = f_z(y, z) = \alpha_z(\beta_z(g - y) - z) \tag{3.4}$$

$$\dot{y} = f_y(z, x, v) = z + \frac{\sum_{i=1}^{N} \Psi_i w_i}{\sum_{i=1}^{N} \Psi_i} v \tag{3.5}$$

$$\dot{v} = f_v(x, v) = \alpha_v(\beta_v(g - x) - v) \tag{3.6}$$

$$\dot{x} = f_x(v) = v \tag{3.7}$$

$$\Psi_i = \exp\left(-\frac{1}{2\sigma_i^2}(\tilde{x} - c_i)^2\right), \quad i = 1, \dots, N \qquad \text{– Gaussian kernel functions} \tag{3.8}$$

$$\tilde{x} = \frac{x - x_0}{g - x_0} \tag{3.9}$$

- The 2nd system dynamic behavior is simply analogous to value tracking: variable x goes to target value g, the velocity v starts and ends at 0 (Fig. 3.11).
- Compared to the 2nd one, the 1st system is just different in the extra nonlinear term in $\dot{y}$

- This is similar as with Fourier series. Fourier series is used to approximate an arbitrary time series data via a weighted sum of multiple sinusoidal functions. Here, we approximate the different between the velocity profile for trajectory tracking and for value tracking by a weighted sum of Gaussian-like functions.
- The above dynamical system is spatially invariant in the sense that scaling of the goal g does not affect the topology of the attractor landscape

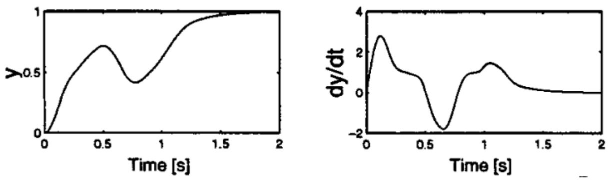


Figure 3.10: Trajectory tracking: y as joint position and velocity profile $\dot{y}$.

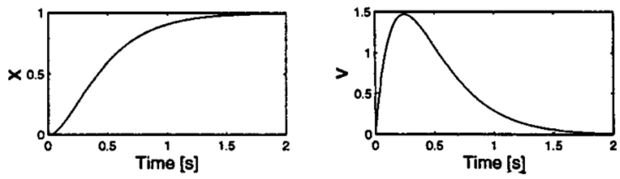


Figure 3.11: Value tracking: x as joint position and velocity profile $\dot{x}$.

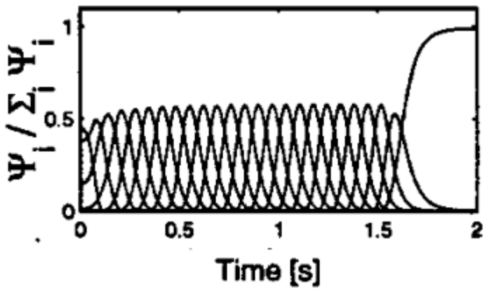


Figure 3.12: Basis functions from Gaussian kernels.

With regards to trajectory planning, long story short, we manage to parameterize a trajectory y to params. w_i

- q - current joint position
- g - desired joint position
- x - a timing signal
- v - a scaling signal

Given a new trajectory:

- Shift the given trajectory to 0 start position
- Scale the time constants of the dynamical system such that (v, x) reaches $(0, g)$ at approximately the same time $\Rightarrow$ a time trajectory for $v(t), x(t)$
- Given $v(t), x(t)$, calculate $u_{des} = \dot{y}_{demo} - z$ as desired output
- Learn w_i via recursive least squares to minimize the locally weighted error criterion for each local model $u_{i,t} = w_i v_t$

$$J_i = \sum_t \Psi_{i,t}(u_{des,t} - u_{i,t})^2$$

This parameterization approach satisfies following requirements:

- Able to fit a demonstrated trajectories with high precision (fitting all DOF and all points of the trajectories)
- Able to generate similar movements to different goals. If we fix the weights w_i, change the goal g, we will receive a similar trajectories. **<u>NOTE:</u>** good with close targets, better with variation in height, not with left and right.
- Robust against perturbations (e.g., in scenarios there is interaction with external objects)

$$\tilde{y} \qquad\qquad \text{– the actual position} \tag{3.10}$$

$$\dot{y} = \alpha_y \left(\frac{\sum_{i=1}^{N} \Psi_i, w_i}{\sum_{i=1}^{N} \Psi_i} v + z \right) + \alpha_{py}(\tilde{y} - y) \tag{3.11}$$

$$\dot{x} = v \left(1 + \alpha_{px}(\tilde{y} - y)^2 \right)^{-1} \tag{3.12}$$

- Good basis for comparing movements, as a metric for measuring differences between movements. Similar movements will have similar weights $w_i \Rightarrow$ can be used to classify movements (simple nearest neighbors)

[**TODO: The effect of hyper-params.**]

- α_v
- β_v
- α_z
- β_z
- c_i
- ...

4 Robotics System Engineering

4.1 Structural Synthesis

4.2 End-Effector Technology

4.3 Components of Robotics System

4.4 Control Architecture

4.5 Mobile Manipulators

5 Robotics Sensor Systems

This chapter gives an overview on different types of sensors used in robotic systems, the physics behind them and more ...

5.1 Overview

Two classes of sensors:

Proprioceptive sensors	Exteroceptive sensors
Proprioception, also referred to as *kinaesthesia*, is the sense of self-movement, force, and body position (src).	*Exteroception* is sensitivity to stimuli that are outside the body, resulting from the response of specialized sensory cells to objects and occurrences in the external environment. Exteroception includes the five senses of sight, smell, hearing, touch, and taste (src).
Proprioceptive sensors provide internal perception about the state of the robot, i.e., joint positions, velocities, accelerations, torque.	*Exteroceptive* sensors provide external perception about the measured force, tactile input, proximity range, vision, etc.

5.1.1 Future Applications

- Industrial robots: more complex tasks, wider / broader range of manufacturing
- Service robots: excluding industrial automation application
- Application: logistics, medical, field, defense robots.

5.1.2 Key abilities

(CIMMPC)

- Configurability
- Interaction ability
- Motion ability
- Manipulation ability
- Perception ability: suitable choice of sensing modality
- Cognitive ability: reduction of programming and configuration effort

5.1.3 Key technologies

$$\text{Advances in sensors: } \begin{cases} -\text{Control theory} \\ -\text{Safety} \\ -\text{Navigation} \end{cases}$$

[**TODO: add graph**]

5.2 Control & Feedback Control Systems

5.2.1 Introduction to industrial robots

5.2.2 Internal metrology of industrial robot

5.2.3 External metrology of industrial robot

5.2.4 Communication between sensors & robots via industrial Ethernet & 5G

5.3 Electromagnetic Sensor

Proprioceptive sensors:

	Encoders	Tachometer	Gyroscope
Position/Pose	✓		
Velocity	✓	✓	
Acceleration	✓	✓	✓

NOTE: If you can measure x, you can derive $\dot{x}, \ddot{x}$

5.3.1 Principles of Electromagnetism

Maxwell's equations:

- Gauss's law

$$\Phi_E = \oint \vec{E}.d\vec{A} = \frac{Q_{encl}}{\varepsilon_0}; \varepsilon_0 = \text{const} \quad \Leftrightarrow \quad \vec{\nabla}\vec{E} = \frac{\rho}{\varepsilon_0} \tag{5.1}$$

$$\Leftrightarrow \oint \vec{D}.d\vec{A} = \int \rho.dV = Q_{encl}; \vec{D} = \varepsilon\vec{E} \quad \Leftrightarrow \quad \boxed{\vec{\nabla}\vec{D} = \rho} \tag{5.2}$$

with: Φ_E – electric flux

- Gauss's law for magnetism: $\vec{B}$ is solenoidal vector fields. There is no magnetic monopole [**TODO:**] **divergence, no source, no sink**

$$\Phi_B = \oint \vec{B}.d\vec{A} = 0 \quad \Leftrightarrow \quad \boxed{\vec{\nabla}\vec{B} = 0} \tag{5.3}$$

with: Φ_B – magnetic flux

- Maxwell-Faraday equation
- Ampere's circuital law
- Magnetic field
- Hall effect

5.3.2 Position Sensors

- Potentiometer: works according to Ohm's law
  $\begin{cases} \text{Turning} \\ \text{Sliding} \end{cases}$ potentiometer: when the joint rotates, the gear turns the potentiometer
- Induction-based sensors:
  - Resolver ...
  - Inductive transducer
  - Tranverse armature transducer: based on Maxwell's bridge
  - Differential transducer
  - Inductosyn
- Rotary encoders:
  - Absolutely Encoder versus (vs.) Incremental Encoder [**TODO: images**]

Absolutely Encoder	Incremental Encoder
maintain information (info.) when power is removed	immediately report position changes
position info. is available immediately	doesn't keep track
relationship between encoder value & physical position set at assembly	have to go back to fixed point to initialize measurement

  - Magnetic encoder vs. Optical encoder
  - Number of track $\Rightarrow$ resolution
    n tracks $\Rightarrow$ resolution $\frac{2\pi}{2_n}$

NOTE: Current robots use encoders (magnetic / optical).

	Advantages	Disadvantages
Potentiometer	Simple Low cost	Limited range & accuracy Wear out (due to contact)
Resolver	Non-contact	Analog, expensive Requires decoding circuit
Inductive Transducer	Non-contact High accuracy	limited range Analog
Magnetic encoder	Non-contact Robust	Required decoding usually lower resolution
Optical encoder	Non-contact	Required gray decoding Complex wiring

5.3.3 Speed Sensors

Maxwell-Faraday's equation:

$$\frac{\partial \vec{B}}{\partial t} \Rightarrow \frac{\partial \vec{E}}{\partial t}$$

5.3.4 Acceleration Sensors

- Mechanical gyroscopes **(conservation of angular momentum)**
- Micro electro-mechanical system gyroscope (MEMS)

5.4 Capacitive & Piezoelectric Sensors

5.5 Visual Electromagnetic Sensors

5.6 Thermoelectric & Ultrasonic Sensors

5.7 Machine Vision

5.8 Tactile Information

Types of robotic tactile sensors:

- Capacitive tactile sensors (Sec. 5.4)
- Piezoresistive tactile sensors (Sec. 5.4)
- Piezoelectric tactile sensors (Sec. 5.4)
- Inductive tactile sensors
- Optical-based tactile sensors

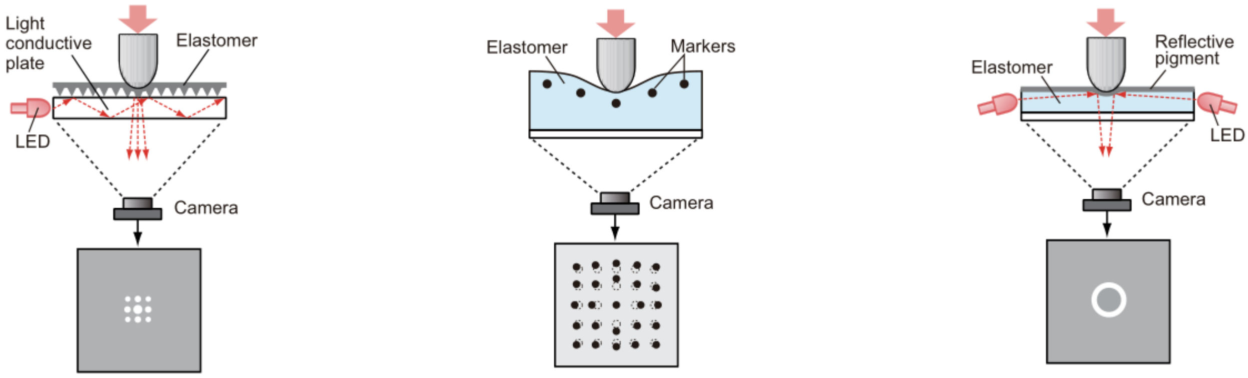


Figure 5.1: Types of optical-based tactile sensors: Light conductive plate, Marker displacement, Reflective membrane (from left to right) [Shi19].

- Strain gauges
- Audio-based tactile sensors: can provide information about surface texture.
- Multi-component tactile sensors

Prior works use tactile sensors in inference of object properties [LBD+17] and robotic manipulation [YA19].

- Grasping
  - Heuristic grasp adaptation strategy
  - Grasp adaptation with slip detection
  - Grasp adaptation with estimating friction coefficient
  - Grasp adaptation with grasp stability estimation
  - Complete grasp process with tactile sensing
- Other robotic manipulation
  - Detecting events with tactile sensors
  - Estimating pose and location with tactile sensors
  - Reinforcement learning with tactile sensors
  - Exploring the workspace without vision
- Tactile perception

Issues of introducing tactile sensors to robotic hands:

- Difficulty to install on robotic hands.
- Wiring, power supply, and processing.
- Low durability, fragility.
- Less compatibility with the other tactile sensors.
- It is unclear what we can do with tactile sensors.
- Disadvantages caused by tactile sensors.
  - Maintenance becomes complicated.

    – Programming becomes complicated.
- Expensive.
- Asking for the moon.

5.9 Data Acquisition & Sensor Fusion

5.10 Signal Preprocessing

5.11 Communication & Signal Transmission

5.11.1 Challenges in Data Transmission

- Different forms (text, speech, images)
- Large distances, limited time
- Certain bandwidth

5.11.2 Information Systems & Data Transmission Principles

5.11.3 Transmission Media

5.11.4 Network Topologies & Data Transmission Protocols

Topologies: −P2P
−Bus
−Fieldbus

Protocols: OSI and TCP/IP

5.11.5 Robot Communication with ROS

6 Probabilistic Robotics

Reference from the great book: [TBF06].

6.1 State Estimation

6.1.1 Bayes Filters

$$
\begin{aligned}
bel(x_t) &= p(x_t|z_{1:t}, u_{1:t}) && \text{belief over a state} \\
\overline{bel}(x_t) &= p(x_t|z_{1:t-1}, u_{1:t}) && \text{a posterior (before adapt to } z_t) \\
\Rightarrow \text{ Calculating } & bel(x_t) \text{ from } \overline{bel}(x_t) && \text{correction/measurement update}
\end{aligned}
$$

Bayes Filter Algorithm:

2 for all x_t do:

3 $\overline{bel}(x_t) = \int p(x_t|u_t, x_{t-1}) bel(x_{t-1}) dx$ prediction step

4 $bel(x_t) = \eta\, p(z_t|x_t)\, \overline{bel}(x_t)$ update step

$\Rightarrow$ Can only be implemented for very simple estimation problems, finite state space

Important assumption: Markov property (each state s a complete summary of the past)

Problem: can not be implemented on digital computers

The next subsections describe two Gaussian filters (Kalman Filter (KF) and Information Filter (IF)) and two extensions of them (Extended Kalman Filter (EKF) and Extended Information Filter (EIF)). The Gaussian filters have major advantage in computational cost, with the disadvantage of having assumption on uni-model distribution.

6.1.2 The Kalman Filter (KF)

- ***Learning Resources:*** www.bzarg.com
- For continuous space, not discrete or hybrid

- **Assumption:** posterior are Gaussians and Markov property

$p(x_t \| u_t, x_{t-1})$	must be linear func. (linear system dynamics)
$p(z_t, x_t)$	also linear
$bel(x_0)$	initial belief must be Gaussian

$$
\begin{aligned}
x_t &= A_t x_{t-1} + B_t u_t + \varepsilon_t \\
p(x_t|u_t, x_{t-1}) &= \det\left(2\pi\Sigma_t\right)^{-\frac{1}{2}} \exp\left[-\frac{1}{2}\left(x_t - A_t x_{t-1} - B_t u_t\right)^T R_t^{-1}\left(x_t - A_t x_{t-1} - B_t u_t\right)\right] \\
z_t &= C_t x_t + \delta_t \quad (= y) \\
p(z_t|x_t) &= \det\left(2\pi Q_t\right)^{-\frac{1}{2}} \exp\left[-\frac{1}{2}\left(z_t - C_t x_t\right)^T Q_t^{-1}\left(z_t - C_t x_t\right)\right] \\
bel(x_0) = p(x_0) &= \det\left(2\pi\Sigma_0\right)^{-\frac{1}{2}} \exp\left[-\frac{1}{2}\left(x_t - \mu_0\right)^T \Sigma_0^{-1}\left(x_t - \mu_0\right)\right]
\end{aligned}
$$

Kalman Filter Algorithm: $(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t)$

2 $\quad \overline{\mu}_t = A_t \mu_{t-1} + B_t u_t$

3 $\quad \overline{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + R_t$

$\Big\}$ incorporate u_t - prediction step - $\mathcal{O}(n^2)$

4 $\quad K_t = \overline{\Sigma}_t C_t^T \left(C_t \overline{\Sigma}_t C_t^T + Q_t\right)^{-1}$ (Kalman gain)

5 $\quad \mu_t = \overline{\mu}_t + K_t(z_t - C_t \overline{\mu}_t)$

6 $\quad \Sigma_t = (I - K_t C_t)\overline{\Sigma}_t$

$\Big\}$ incorporate z_t - correction step - $\mathcal{O}(n^{2,8})$

7 $\quad$ return μ_t, Σ_t $\quad \Rightarrow$ belief at time t

$\Rightarrow$ quite computationally expensive, **everything are Gaussians**

6.1.3 Extended Kalman Filter (EKF)

Overcome the assumption on linearity by only approximate by Gaussians

$$
\begin{aligned}
x_t &= g(u_t, x_{t-1}) + \varepsilon_t \\
z_t &= h(x_t) + \delta_t
\end{aligned}
$$

$$
\begin{aligned}
g(u_t, x_{t-1}) &\approx g(u_t, \mu_{t-1}) + g'(u_t, \mu_{t-1})(x_t - \mu_{t-1}) \\
&= g(u_t, \mu_{t-1}) + G_t(x_t - \mu_{t-1}) \\
&\quad g' \text{ is the Jacobian of state } (n \times n \text{ matrix}) \\
p(x_t|u_t, x_{t-1}) &\approx \det(2\pi R_t)^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}[x_t - g(u_t, x_{t-1})]^T R_t^{-1}[x_t - g(u_t, x_{t-1})]\right\} \\
h(t) &\approx h(\overline{\mu}_t) + h'(\mu_t)(x_t - \mu_{t-1}) \\
&= h(\overline{\mu}_t) + H_t(x_t - \mu_{t-1}) \\
p(z_t|x_t) &= \det(2\pi Q_t)^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}[z_t - h(x_t)]^T Q_t^{-1}[z_t - h(x_t)]\right\}
\end{aligned}
$$

Extended Kalman Filter Algorithm: $(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t)$

2 $\quad \overline{\mu}_t = g(u_t, \mu_{t-1})$

3 $\quad \overline{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t$

4 $\quad K_t = \overline{\Sigma}_t H_t^T \left(H_t \overline{\Sigma}_t H_t^T + Q_t\right)^{-1}$

5 $\quad \mu_t = \overline{\mu}_t + K_t[z_t - h(\overline{\mu}_t)]$

6 $\quad \Sigma_t = (I - K_t H_t)\overline{\Sigma}_t$

7 $\quad$ return μ_t, Σ_t

- Can extend EKF $\Rightarrow$ Multi-Hypothesis Extended Kalman Filter (MHEKF)
- EKF's performance depends on degree of nonlinearities and uncertainty
- Unscented KF and moments matching KF are better

6.1.4 Information Filter (IF)

- Moment representation: $\mu \quad \& \quad \Sigma$
- Canonical representation: $\xi \quad \& \quad \Omega$

  Information precision matrix: $\Omega = \Sigma^{-1}; \qquad \Sigma = \Omega^{-1}$

  Information vector: $\xi = \Sigma^{-1}\mu; \qquad \mu = \Omega^{-1}\xi$

$$
\begin{aligned}
p(x) &= \det(2\pi\Sigma)^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right\} \\
&= \eta \exp\left\{-\frac{1}{2}x^T\Omega x + x^T\xi\right\}
\end{aligned}
$$

Information Filter Algorithm: $(\xi_{t-1}, \Omega_{t-1}, u_t, z_t)$

$$\left.\begin{array}{ll} 2 & \overline{\Omega}_t = (A_t \Omega_{t-1}^{-1} A_t^T + R_t)^{-1} \\ 3 & \overline{\xi}_t = \overline{\Omega}_t (A_t \Omega_{t-1}^{-1} \xi_{t-1} + B_t u_t) \end{array}\right\} \mathcal{O}(n^{2,8})$$

$$\left.\begin{array}{ll} 4 & \Omega_t = C_t^T Q_t^{-1} C_t + \overline{\Omega}_t \\ 5 & \xi_t = C_t^T Q_t^{-1} z_t + \overline{\xi}_t \end{array}\right\} \mathcal{O}(n^2)$$

6 return ξ_t, Ω_t

6.1.5 Extended Information Filter (IF)

$$\begin{aligned} x_t &= g(u_t, x_{t-1}) + \varepsilon_t \\ z_t &= h(x_t) + \delta_t \\ G_t &= g'(u_t, \mu_{t-1}) \\ H_t &= h'(\mu_t) \end{aligned}$$

Extended Information Filter Algorithm: $(\xi_{t-1}, \Omega_{t-1}, u_t, z_t)$

2 $\mu_{t-1} = \Omega_{t-1}^{-1} \xi_{t-1}$

3 $\overline{\Omega}_t = (G_t \Omega_{t-1}^{-1} G_t^T + R_t)^{-1}$

4 $\overline{\xi}_t = \overline{\Omega}_t \, g(u_t, \mu_{t-1})$

5 $\overline{\mu}_t = g(u_t, \mu_{t-1})$

6 $\Omega_t = \overline{\Omega}_t + H_t^T Q_t^{-1} H_t$

7 $\xi_t = \overline{\xi}_t + H_t^T Q_t^{-1} \left[z_t - h(\overline{\mu}_t) - H_t \overline{\mu}_t\right]$

- ***Global uncertainty:*** set $\Omega = 0$ is better than set $|\Sigma| = \infty$
- IF tends to be numerically more stable than KF
- IF is better for multi-robot problems
- For high dimensional state, EKF is computational better than EIF

6.2 Measurements

6.2.1 Map Representation

Maps: $m = \{m_1, m_2, \ldots, m_N\}$
There are 2 ways to represent a map:

[TODO: Add images]

feature-based	location-based
m_n: properties of a feature and location of feature	a specific location
only the shape of the environment **at the specific locations**	volumetric: **label for any location** in the world
easy to adjust positions of objects $\Rightarrow$ popular in the robotic mapping field	occupancy grid map

6.2.2 Measurement Noise

The 4 types of measurement noise:

- Correct range with local measurement noise
  With z_t^{k*} as the correct distance

$$p_{hit}(z_t^k|x_t, m) = \begin{cases} \eta\, \mathcal{N}(z_t^k|z_t^{k*}, \sigma_{hit}^2) & \text{if } 0 \leq z_t^k \leq z_{\max} \\ 0 & \text{otherwise} \end{cases} \tag{6.1}$$

- Unexpected object

$$p_{short}(z_t^k|x_t, m) = \begin{cases} \eta \lambda_{short} e^{-\lambda_{short} z_t^k} & \text{if } 0 \leq z_t^k \leq z_t^{k*} \\ 0 & \text{otherwise} \end{cases} \tag{6.2}$$

- Failures

$$p_{\max}(z_t^k|x_t, m) = I(z = z_{\max}) \tag{6.3}$$

- Random measurements

$$p_{rand}(z_t^k|x_t, m) = \begin{cases} \frac{1}{z_{\max}} & \text{if } 0 \leq z_t^k < z_{\max} \\ 0 & \text{otherwise} \end{cases} \tag{6.4}$$

[**TODO: Add image, plot**]

$$p(z_t^k|x_t, m) = \begin{bmatrix} z_{hit} \\ z_{short} \\ z_{max} \\ z_{rand} \end{bmatrix}^T \cdot \begin{bmatrix} p_{hit}(z_t^k|x_t, m) \\ p_{short}(z_t^k|x_t, m) \\ p_{max}(z_t^k|x_t, m) \\ p_{rand}(z_t^k|x_t, m) \end{bmatrix} \tag{6.5}$$

6.3 Robot Motion

Pose: $[x, y, \theta]^T$ at location $[x, y]^T$ and orientation θ

6.3.1 Motion Model

Motion Model, also known as (a.k.a.) Probabilistic Kinematic Model: $p(x_t, u_t, x_{t-1})$

Velocity commands	Odometry (distance traveled, angle turned, etc.)
Use for Probabilistic motion planning	**more accurate but post-the-fact (not for motion planning)** Use for estimation

Each has closed form calculation and sampling algorithm.

6.3.2 Velocity Motion Model

Assuming we can control a robot through velocities:

$$u_t = \begin{bmatrix} v_t \\ \omega_t \end{bmatrix}; \qquad x_{t-1} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}; \qquad x_t = \begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix}$$

Motion Model Velocity Algorithm: (x_t, u_t, x_{t-1})

$$\left.\begin{aligned} &\mu = \frac{1}{2}\frac{(x-x')\cos\theta + (y-y')\sin\theta}{(y-y')\cos\theta - (x-x')\sin\theta} \\ &x^* = \tfrac{x+x'}{2} + \mu(y-y') \\ &y^* = \tfrac{y+y'}{2} + \mu(x'-x) \\ &r^* = \sqrt{(x-x^*)^2 + (y-y^*)^2} \end{aligned}\right\}\text{Invert the motion model}$$

$$\Delta\theta = \operatorname{atan2}(y'-y^*, x'-x^*) - \operatorname{atan2}(y-y^*, x-x^*)$$

$$\left.\begin{aligned} &\hat{v} = \tfrac{\Delta\theta}{\Delta t} r^* \\ &\hat{\omega} = \tfrac{\Delta\theta}{\Delta t} \\ &\hat{\gamma} = \tfrac{\theta'-\theta}{\Delta t} - \hat{\omega} \end{aligned}\right\}\text{compared actual velocities with the desired}$$

return $p(v-\hat{v}, \alpha_1|v| + \alpha_2|\omega|) \cdot p(\omega-\hat{\omega}, \alpha_3|v| + \alpha_4|\omega|) \cdot p(\hat{\gamma}, \alpha_5|v| + \alpha_6|\omega|)$

Sample Motion Model Velocity Algorithm: (u_t, x_{t-1})

$$\hat{v} = v + sample(\alpha_1|v| + \alpha_2|\omega|)$$
$$\hat{\omega} = \omega + sample(\alpha_3|v| + \alpha_4|\omega|)$$
$$\hat{\gamma} = sample(\alpha_5|v| + \alpha_6|\omega|)$$
$$x' = x - \frac{\hat{v}}{\hat{\omega}}\sin\theta + \frac{\hat{v}}{\hat{\omega}}\sin(\theta + \hat{\omega}\Delta t)$$
$$y' = y + \frac{\hat{v}}{\hat{\omega}}\cos\theta - \frac{\hat{v}}{\hat{\omega}}\cos(\theta + \hat{\omega}\Delta t)$$
$$\theta' = \theta + \hat{\omega}\Delta t + \hat{\gamma}\Delta t$$
$$\text{return } x_t = [x', y', \theta']^T$$

- Probability normal distribution(a, b): return $\frac{1}{\sqrt{2\pi b}} e^{-\frac{a^2}{2b}}$
- Probability triangular distribution(a, b): return $\begin{cases} 0 & \text{if } |a| > \sqrt{6b} \\ \frac{\sqrt{6b}-|a|}{6b} & \text{else} \end{cases}$
- Sample normal distribution(b): return $\frac{b}{6}\sum_{i=1}^{12} rand(-1,1)$
- Sample triangle distribution(b): return $b.rand(-1,1).rand(-1,1)$

6.3.3 Odometry Motion Model

Only available after the robot has moved
$\Rightarrow$ only use for filter algorithm
not for accurate motion planning and control

Motion Model Odometry Algorithm: (x_t, u_t, x_{t-1})

$$\delta_{rot1} = \text{atan2}(\bar{y}' - \bar{y}, \bar{x}' - \bar{x}) - \bar{\theta}$$
$$\delta_{trans} = \sqrt{(\bar{x} - \bar{x}')^2 + (\bar{y} - \bar{y}')^2}$$
$$\delta_{rot2} = \bar{\theta}' - \bar{\theta} - \delta_{rot1}$$
$$\left.\begin{aligned} &\hat{\delta}_{rot1} = \text{atan2}(y' - y, x' - x) - \theta \\ &\hat{\delta}_{trans} = \sqrt{(x - x')^2 + (y - y')^2} \\ &\hat{\delta}_{rot2} = \theta' - \theta - \hat{\delta}_{rot1} \end{aligned}\right\} \Rightarrow \text{inverse motion model}$$
$$p_1 = prob(\delta_{rot1} - \hat{\delta}_{rot1}, \alpha_1\hat{\delta}_{rot1} + \alpha_2\hat{\delta}_{trans})$$
$$p_2 = prob(\delta_{trans} - \hat{\delta}_{trans}, \alpha_3\hat{\delta}_{trans} + \alpha_4(\hat{\delta}_{rot1} + \hat{\delta}_{rot2}))$$
$$p_3 = prob(\delta_{rot2} - \hat{\delta}_{rot2}, \alpha_1\hat{\delta}_{rot2} + \alpha_2\hat{\delta}_{trans})$$
$$\text{return } p_1.p_2.p_3 \quad (= p(x_t|u_t, x_{t-1}))$$

NOTE:

- Bar $\Leftrightarrow$ measurements

$$\bar{x}_{t-1} = [\bar{x} \quad \bar{y} \quad \bar{\theta}]^T$$
$$\bar{x}_t = [\bar{x}' \quad \bar{y}' \quad \bar{\theta}']^T$$

- Hat $\Leftrightarrow$ estimations
- No bar and hat $\Leftrightarrow$ hypothesized final pose x, y

Sample Motion Model Odometry Algorithm: (u_t, x_{t-1})

$$\delta_{rot1} = \text{atan2}(\bar{y}' - \bar{y}, \bar{x}' - \bar{x}) - \bar{\theta}$$
$$\delta_{trans} = \sqrt{(\bar{x} - \bar{x}')^2 + (\bar{y} - \bar{y}')^2}$$
$$\delta_{rot2} = \bar{\theta}' - \bar{\theta} - \delta_{rot1}$$
$$\hat{\delta}_{rot1} = \delta_{rot1} - sample(\alpha_1 \delta_{rot1} + \alpha_2 \delta_{trans})$$
$$\hat{\delta}_{trans} = \delta_{trans} - sample(\alpha_3 \delta_{trans} + \alpha_4 (\delta_{rot1} + \delta_{rot2}))$$
$$\hat{\delta}_{rot2} = \delta_{rot2} - sample(\alpha_1 \delta_{rot2} + \alpha_2 \delta_{trans})$$
$$x' = x + \hat{\delta}_{trans} \cos(\theta + \hat{\delta}_{rot1})$$
$$y' = y + \hat{\delta}_{trans} \sin(\theta + \hat{\delta}_{rot1})$$
$$\theta' = \theta + \hat{\delta}_{rot1} + \hat{\delta}_{rot2}$$
$$\text{return} \quad x_t = [x' \quad y' \quad \theta']^T$$

6.3.4 Map-based Motion Model

Map-based Motion Model: $p(x_t|u_t, x_{t-1}, m)$

- Occupancy maps: $p(x_t|m) = 0 \Leftrightarrow$ the robot collides
- If the distance from $x_{t-1} \rightarrow x_t$ is small enough ($<$ half the robot's diameter), we can estimate the probability (prob.) $p(x_t|u_t, x_{t-1}, m) \approx \eta p(x_t|u_t, x_{t-1}) p(x_t|m)$, which discards the info relating the robot's path to x_t

Motion Model with Map Algorithm: (x_t, u_t, x_{t-1}, m)
return $p(x_t|u_t, x_{t-1}).p(x_t|m)$

Sample Motion Model with Map Algorithm: (u_t, x_{t-1}, m)

$$
\begin{aligned}
&do: \\
&\qquad x_t = sample\_motion\_model(u_t, x_{t-1}) \\
&\qquad \pi = p(x_t|m) \\
&until \quad \pi > 0 \\
&return < x_t, \pi >
\end{aligned}
$$

7 Markov Decision Process

7.1 Definitions

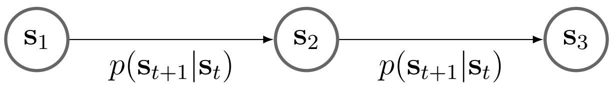
7.1.1 Markov Chain

A Markov Chain $\mathcal{M}$ is a tuple consisting of:

$$\mathcal{M} = \{\mathcal{S}, \mathcal{T}\}$$

$\mathcal{S}$ – state space	$s \in \mathcal{S}$	(discrete or continuous)
$\mathcal{T}$ – transition operator	$p(s_{t+1}\|s_t)$ or $\vec{\mu}_{t+1} = \mathcal{T}\vec{\mu}_t$	($\mathcal{T}$ is a matrix)

Markov chain is a process without memories. In other words, the next state s_{t+1} depends only on the current state s_t, not the previous state s_{t-1}.

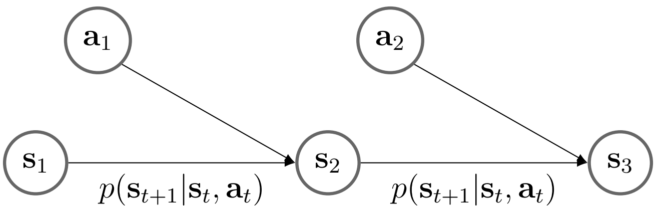


7.1.2 Markov Decision Process (MDP)

A Markov Decision Process (MDP) $\mathcal{M}$ is a tuple consisting of:

$$\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{T}, r, \gamma\}$$

$\mathcal{S}$ – state space	$s \in \mathcal{S}$	(discrete or continuous)
$\mathcal{A}$ – action space	$a \in \mathcal{A}$	(discrete or continuous)
$\mathcal{T}$ – transition operator	$p(s_{t+1}\|s_t)$	(now a tensor)
r – reward function	$r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$,	$r(s_t, a_t)$
γ – discount factor	$\gamma \in [0, 1]$	(optional)



There are definitions relating to MDP:

- Policy: choice of action (at each state): $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$
- Utility: sum of (discounted) rewards

A MDP can also be considered as a Markov chain on the augmented state $(\mathbf{s},\ \mathbf{a})$, a.k.a. Q-state. Knowing the state transition $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ and policy $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$, the transition of these Q-states can be derived as follows:

$$p((\mathbf{s}_{t+1}, \mathbf{a}_{t+1})|(\mathbf{s}_t, \mathbf{a}_t)) = p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)\pi_\theta(\mathbf{a}_{t+1}|\mathbf{s}_{t+1}) \tag{7.1}$$

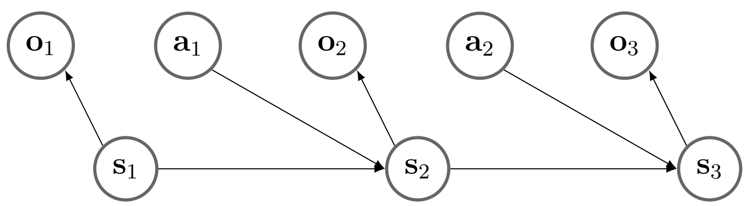
MDP state projects an expectimax-like search tree [**TODO: add image**]

7.1.3 Partially Observable Markov Decision Process (POMDP)

A Partially Observable Markov Decision Process (POMDP) $\mathcal{M}$ is a tuple consisting of:

$$\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{T}, \mathcal{E}, r, \gamma\}$$

$\mathcal{S}$ – state space	$s \in \mathcal{S}$ (discrete or continuous)
$\mathcal{A}$ – action space	$a \in \mathcal{A}$ (discrete or continuous)
$\mathcal{O}$ – observation space	$o \in \mathcal{O}$ (discrete or continuous)
$\mathcal{T}$ – transition operator	$p(s_{t+1}|s_t)$
$\mathcal{E}$ – emission prob.	$p(o_t|s_t)$
r – reward function	$r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$
γ – discount factor	$\gamma \in [0, 1]$ (optional)



Finite vs. infinite horizon: [**TODO:**]

<u>To solve infinite utilities:</u>

- Finite horizon
- Discounting
- Absorbing state (like the fire hole, overheating)

7.2 Bellman equations

- $T(s,a,s') = P(s'|s,a)$ the prob. of reaching state s' from taking action a at state s
- $R(s,a,s')$ the reward of making the transition
- $Q^*(s,a)$: expected utility starting in state s and having taken action a, then act optimally

$$Q^*(s,a) = \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s')\right] \tag{7.2}$$

  It is the sum over possible next state s', because there is uncertainty of which state s' will be reached, even with the same starting state s and action a.
- $V^*(s)$: value of a state - expected utility starting in state s and acting optimally

$$V^*(s) = \max_a Q^*(s,a) \tag{7.3}$$

$$\Rightarrow \quad V^*(s) = \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s')\right] \tag{7.4}$$

- $\pi^*(s)$: optimal action / policy from state s

[**TODO: Explain this??**]

$$\mathbf{V(s) = \mathbb{E}[G_t|S_t = s] = \mathbb{E}\left[R_{t+1} + \gamma V(S_{t+1})|S_t = s\right] = R_s + \gamma \sum_{s'} P_{ss'} V(s')} \tag{7.5}$$

7.3 Partially Observable Markov Decision Process

POMDP is defined with a tuple: $< S, A, O, P, R, Z, \gamma >$:

- S: state
- A: action
- O: **observation**
- P: transition matrix
- R: reward
- Z: **observation func.**
- γ: discount factor (optional)

$$Z^a_{s'o} = P\left[O_{t+1} = o|S_{t+1} = s', A_t = a\right] \tag{7.6}$$

$$\Rightarrow \begin{cases} V^*(b) \leftarrow \max_a \left[R^a_b + \gamma \sum_{s'} P^a_{bb'} V(b')\right] \\ \pi^*(b) = \arg\max_a \left[R^a_b + \gamma \sum_{s'} P^a_{bb'} V(b')\right] \end{cases} \tag{7.7}$$

8 Research Proposal

8.1 Introduction and Background of Interest

[**TODO: ignore for now**]

Robots are complex machines designed to support our lives.

Since mid 19th century, its emergence has received research attention in both the academic and industry. The first major application was in the automotive industry. Its presence is entering our lives in different fields is spreading even more and more. [**TODO: move to unstructured environments and challenges**]

8.2 Literature Review

To my best current knowledge, robot arms have made significant improvement, but are still far from delivering general-purpose tasks. Initially, robot arms are programmed explicitly and only capable of working in the known and constrained environment of factories. Progress in different fields of technology has provided the robots more inputs (e.g., vision, audio, haptic) to operate in dynamic and unstructured environments. State-of-the-art robot manipulators can deliver complex tasks, e.g., cooking [Mol], making coffee [Cof], and cleaning around the house [Bot]. However, the behaviors of these models are still awkwardly disruptive and far from the mastery level of the human hand's dexterity. In addition, instead of having the adaptability for a wide range of conditions, most models still operate in designed environments with known tools and settings.

8.2.1 Robotic Grasping

Grasping is one of the critical and unavoidable problems for robot manipulators, especially for logistic and service robots. Overall, most successful grasping approaches are empirical, using deep learning on a large dataset to find the best grasping candidates. Majority of models are supervised learning using single-modal input, i.e. RGB images. A few have considered multi-modal inputs: RGB with depth images or tactile input. Approaches, which take tactile data, use it as the sole input to improve grasp stability, adaptability to object's weight [BLJ+11; LBK+14], a few combine with other modality for improvement [COU+17]. [BMA+13; CRC18; LLC+19; KBK+20]

8.2.2 Reinforcement Learning

Reinforcement Learning (RL) is a branch of Machine Learning (ML) approaches for learning decision making from experiences. Given a sufficient amount of input data, it can potentially surpass human's ability for reasoning, e.g., playing games like Go, chess, Atari. Most works on robot manipulators are currently directed to object picking and hand-over tasks. To meet the need for data, sim-to-real transfer techniques [KBK+20] and collective learning are being studied [YLK+17]. Unlike in other domains of ML, it's difficult to implement transfer learning with RL, given different robot arms, gripper and sensor types. Various directions are being researched to tackle this issue, i.e. offline RL, novel exploration strategies, multi-task learning, meta-learning. [KBP13; Li17; ADB+17; SKS21]

8.3 Approaches and Choice of Methods

8.3.1 Key Points for Improvement

Prior works on robotic manipulators were limited on their utilization of tactile sensors. Because complex hand and arm behaviors need force and pressure feedback, leveraging the manipulators to the dexterity level of human's arms requires these sensors. However, tactile sensors are currently been used mostly for the development of soft robotics [HJL18]. For grasping tasks, despite certain successes from using visual and tactile input separately, there hasn't being a multi-modal approach capable of combining the strengths of both sides, i.e., accurate localization, adaptability to object properties. A model, which has knowledge from both visual and tactile sources, promises to deliver complex manipulation tasks with diverse objects.

It's still challenging to generalize and apply RL to different robotic manipulator's tasks. In general, RL did make a convincing case about its potential to surpass human's ability in some specific applications, e.g., playing Go, chess. However, in other fields, practical applications are yet scattered and limited. For robotic manipulators, current focuses are still around object picking and handling tasks [SKS21]. One of the major causes for this situation is the current gap in generalization and transfer learning. Directions for improvements are meta-learning, multi-task learning, etc. [KBP13].

8.3.2 Approaches and Research Contributions

The progress I wish to work towards concerns robotic collective learning for collaborative tasks. For example, a system of multiple robot arms learning to play 2v2 table tennis, pack an item or cook. Building up towards this progress, I intend to work and gain more experiences on the following problems:

1. A multi-modal approach for robotic picking task: combines visual and tactile input into a Deep Learning (DL) model. Prior works on tactile sensors have proven their potentials in inference of object properties [LBD+17] and improving robotic grasping [YA19]. I want to further extend recent advances in tactile sensors to a multi-modal approach.
2. Visual imitation learning: copies behaviors from visual demonstrations (videos). Prior works are still restricted with certain types of input data, tasks [FYZ+17; SPG19]. Based on previous progress on teleoperation [HVWY+20], I want to extend further to learn in different problem settings, with generalized tasks, and also apply offline RL.
3. A data collection pipeline for robot manipulators: preferably from using the above visual imitation learning, to map human's arm movements to manipulator's. The data would contain both visual and tactile information. It would then be used for the learning of other robotic tasks or in other settings (different types of arm, gripper, etc.).
4. Meta-learning with different manipulation tasks.

Moreover, I'm also interested in exploring the usage of certain DL models, techniques:

- Generative model: Generating expert-like samples.
- Score matching
- Exploration for imitation learning

8.4 Proposed Research Plan

[**TODO:**]

[**TODO: How the lab is related to this research plan?**]

- There is alignment in our directions
- Fit the position

[**TODO: Give compliments to their works!**]

- mention more explicitly regarding the topics, what research they have done
- I read your papers about ... let alone the works on ...

Bibliography

[ADB+17] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath. "Deep reinforcement learning: A brief survey". In: *IEEE Signal Processing Magazine* 34.6 (2017), pp. 26–38.

[BLJ+11] Y. Bekiroglu, J. Laaksonen, J. A. Jorgensen, V. Kyrki, and D. Kragic. "Assessing grasp stability based on learning and haptic data". In: *IEEE Transactions on Robotics* 27.3 (2011), pp. 616–629.

[BMA+13] J. Bohg, A. Morales, T. Asfour, and D. Kragic. "Data-driven grasp synthesis—a survey". In: *IEEE Transactions on robotics* 30.2 (2013), pp. 289–309.

[Bot] *Bot Handy Samsung*. https://research.samsung.com/robot. Accessed: 2022/06/16.

[Cof] *Coffee Master OrionStar*. https://en.orionstar.com/coffeemaster.html. Accessed: 2022/06/16.

[COU+17] R. Calandra, A. Owens, M. Upadhyaya, W. Yuan, J. Lin, E. H. Adelson, and S. Levine. "The feeling of success: Does touch sensing help predict grasp outcomes?" In: *arXiv preprint arXiv:1710.05512* (2017).

[CRC18] S. Caldera, A. Rassau, and D. Chai. "Review of deep learning methods in robotic grasp detection". In: *Multimodal Technologies and Interaction* 2.3 (2018), p. 57.

[FYZ+17] C. Finn, T. Yu, T. Zhang, P. Abbeel, and S. Levine. "One-shot visual imitation learning via meta-learning". In: *Conference on Robot Learning*. PMLR. 2017, pp. 357–368.

[HJL18] S. Haddadin, L. Johannsmeier, and F. D. Ledezma. "Tactile robots as a central embodiment of the tactile Internet". In: *Proceedings of the IEEE* 107.2 (2018), pp. 471–487.

[HVWY+20] A. Handa, K. Van Wyk, W. Yang, J. Liang, Y.-W. Chao, Q. Wan, S. Birchfield, N. Ratliff, and D. Fox. "Dexpilot: Vision-based teleoperation of dexterous robotic hand-arm system". In: *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*. IEEE. 2020, pp. 9164–9170.

[INS02] A. J. Ijspeert, J. Nakanishi, and S. Schaal. "Movement imitation with nonlinear dynamical systems in humanoid robots". In: *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*. Vol. 2. IEEE. 2002, pp. 1398–1403.

[KBK+20] K. Kleeberger, R. Bormann, W. Kraus, and M. F. Huber. "A survey on learning-based robotic grasping". In: *Current Robotics Reports* 1.4 (2020), pp. 239–249.

[KBP13] J. Kober, J. A. Bagnell, and J. Peters. "Reinforcement learning in robotics: A survey". In: *The International Journal of Robotics Research* 32.11 (2013), pp. 1238–1274.

[LBD+17] S. Luo, J. Bimbo, R. Dahiya, and H. Liu. "Robotic tactile perception of object properties: A review". In: *Mechatronics* 48 (2017), pp. 54–67.

[LBK+14] M. Li, Y. Bekiroglu, D. Kragic, and A. Billard. "Learning of grasp adaptation through experience and tactile sensing". In: *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*. IEEE. 2014, pp. 3339–3346.

[Li17] Y. Li. "Deep reinforcement learning: An overview". In: *arXiv preprint arXiv:1701.07274* (2017).

[LLC+19] Y. Li, Q. Lei, C. Cheng, G. Zhang, W. Wang, and Z. Xu. "A review: Machine learning on robotic grasping". In: *Eleventh International Conference on Machine Vision (ICMV 2018)*. Vol. 11041. SPIE. 2019, pp. 775–783.

[Mol] *Moley Robotics*. https://moley.com/. Accessed: 2022/06/16.

[Shi19] K. Shimonomura. "Tactile image sensors employing camera: A review". In: *Sensors* 19.18 (2019), p. 3933.

[SKK08] B. Siciliano, O. Khatib, and T. Kröger. *Springer Gandbook of Robotics*. Vol. 200. Springer, 2008.

[SKS21] B. Singh, R. Kumar, and V. P. Singh. "Reinforcement learning in robotic applications: a comprehensive survey". In: *Artificial Intelligence Review* (2021), pp. 1–46.

[SPG19] P. Sharma, D. Pathak, and A. Gupta. "Third-person visual imitation learning via decoupled hierarchical controller". In: *Proc. of the Conf. on Neural Information Processing Systems (NeurIPS)* 32 (2019).

[TBF06] S. Thrun, W. Burgard, and D. Fox. *Probalistic Robotics*. Emerald Group Publishing Limited, 2006.

[YA19] A. Yamaguchi and C. G. Atkeson. "Recent progress in tactile sensing and sensors for robotic manipulation: can we turn tactile sensing into vision?" In: *Advanced Robotics* 33.14 (2019), pp. 661–673.

[YLK+17] A. Yahya, A. Li, M. Kalakrishnan, Y. Chebotar, and S. Levine. "Collective robot reinforcement learning with distributed asynchronous guided policy search". In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, pp. 79–86.