Autonomous Mobile Robots

Cheat Sheet by Janis Hutz
ETHZ, FS2026

1 Introduction

1.1 Probability

Def **Sum rule** $\mathbb{P}(X) = \sum \mathbb{P}(X,Y) = \sum \mathbb{P}(X \cap Y)$

Def **Prod** $\mathbb{P}(X \cap Y) = \mathbb{P}(X|Y)\mathbb{P}(Y) = \mathbb{P}(Y|X)P(X)$

Thm **Bayes** $\mathbb{P}(Y_i|X) = \frac{\mathbb{P}(X|Y_i)\mathbb{P}(Y_i)}{\sum_{j=1}^{n}\mathbb{P}(X|Y_j)\mathbb{P}(Y_j)}$

Def **Cont. Var** Sums become integrals (and vice-versa) e.g. $\sum_X \mathbb{P}(X) = 1$ becomes $\int \mathbb{P}(x)\,\mathrm{d} = 1$

Def **Indep.** x, y indep. iff $\mathbb{P}(\mathcal{X} \cap \mathcal{Y}) = \mathbb{P}(\mathcal{X})\mathbb{P}(\mathcal{Y})$

Def **Cond. Indep.** iff $\mathbb{P}(\mathcal{X} \cap \mathcal{Y}|\mathcal{Z}) = \mathbb{P}(\mathcal{X}|\mathcal{Y})\mathbb{P}(\mathcal{Y}|\mathcal{Z})$

Def $\mathbb{E}[\boldsymbol{x}] = \int_{-\infty}^{\infty} \boldsymbol{x}\mathbb{P}(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}$, also for $\boldsymbol{x} = \boldsymbol{f}(\boldsymbol{x})$

Def $\mathrm{Cov}[x] = \mathbb{E}[\boldsymbol{x}\boldsymbol{x}^\top] - \mathbb{E}[\boldsymbol{x}]\mathbb{E}[\boldsymbol{x}]^\top = \boldsymbol{\Sigma}$

Def **Gauss. Dist.** $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ ($\boldsymbol{\mu}$ mean, $\boldsymbol{\Sigma}$ cov.), PDF: $f(\boldsymbol{x}) = \frac{1}{\sqrt{(2\pi)^k \det(\boldsymbol{\Sigma})}} \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right)$

Σ^{-1} for Σ diagonal, inverse of diag els (e.g. σ^{-1})

Always Normalize (i.e. sum of all probabilities is 1)

1.2 Measurement models

$\boldsymbol{z} = \boldsymbol{b}_C + s\boldsymbol{M}_S\boldsymbol{\omega} + \boldsymbol{b} + \boldsymbol{n} + \boldsymbol{o}$: $\boldsymbol{b}_C$ const bias, $\boldsymbol{b}$ time bias, $\boldsymbol{M}$ missal., $\boldsymbol{n} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{R})$ noise, ${}_S\omega$ corr. meas., $\boldsymbol{o}$ other infl.

Finding: Is in W-frame: may need $\boldsymbol{T}_{BW}$ or $\boldsymbol{R}_{BW}$. Also see Sec. 3

1.3 Trigonometry & Linear Algebra

Def **Rule of cosines** $c^2 = a^2 + b^2 - 2ab\cos(\gamma)$

Def **Orthogonal vec** $v^\top w = 0$; **Def** **Vec Line Eq** $\boldsymbol{p} + \lambda\boldsymbol{d}$

Def **Determinant** $ad - bc$ for mat $[a, b; c, d]$.

Def **Eigenvalues** Solve $\det(M - \lambda I) = 0$.

Rem **Matrix Inverse** For diagonal, inverse of diag els

1.4 Error Propagation

For functions $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x}$, the ***linear error propagation*** is given by $\Sigma^f = A\Sigma^x A^\top$, with Σ^x the uncertanty of $\boldsymbol{x}$ (covariance mat.), typically $\mathtt{diag}(\sigma^2)$, with σ^2 the variance of all variables involved

2 Locomotion & Kinematics

2.1 Positioning

Def **Pos Vec.** ${}_{\boxed{W}}\boldsymbol{t}_{\boxed{B}} = {}_{\boxed{W}}\boldsymbol{t}_{\boxed{W}\boxed{B}}$, Relative to Frame, P. in other Frame, Start P. of vec in first F., $\sin = s$, $\cos = c$

Def **State vector** x_R: x, v of rob in W, pos of sensors

Def **Rot. Mat.** $\boldsymbol{R}_z(\psi)$ (Yaw), $\boldsymbol{R}_y(\theta)$ (Pitch), $\boldsymbol{R}_x(\varphi)$ (Roll)

$$\begin{bmatrix} c(\psi) & -s(\psi) & 0 \\ s(\psi) & c(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}; \begin{bmatrix} c(\theta) & 0 & s(\theta) \\ 0 & 1 & 0 \\ -s(\theta) & 0 & c(\theta) \end{bmatrix}; \begin{bmatrix} 1 & 0 & 0 \\ 0 & c(\varphi) & -s(\varphi) \\ 0 & s(\varphi) & c(\varphi) \end{bmatrix}$$

Rem Application: ${}_W\boldsymbol{a} = \boldsymbol{R}_{WB}{}_B\boldsymbol{a}$

Lem $\boldsymbol{R}_{BW} = \boldsymbol{R}_{WB}^{-1} = \boldsymbol{R}_{WB}^\top$, $\det(\boldsymbol{R}_{WB}) = 1$ (orth.)

Rem Cols of $\boldsymbol{R}_{WB}$ are basis vec. of Frame $\vec{\mathcal{F}}_B$ in $\vec{\mathcal{F}}_W$

Def **Euler Ang** (Tait-Brian) $\boldsymbol{R}_{EB} = \boldsymbol{R}_z(\psi) \cdot \boldsymbol{R}_y(\theta) \cdot \boldsymbol{R}_x(\varphi)$.

$\psi = \arcsin\left(R_{21} \div \sqrt{1-R_{31}^2}\right)$, $\theta = \arcsin(-R_{31})$, $\varphi = \arcsin\left(R_{31} \div \sqrt{1-R_{31}^2}\right)$

$$[\boldsymbol{n}]^\times = \begin{bmatrix} 0 & -n_3 & n_2 \\ n_3 & 0 & -n_1 \\ -n_2 & n_1 & 0 \end{bmatrix}$$

For pitch axis $\theta = \pm 90\,\mathrm{deg}$, ***Gimbal Lock***, Jacobian ***singular***.

Def **Angle-Axis** $\boldsymbol{\alpha} = \alpha\boldsymbol{n}$ ($\boldsymbol{n}$ normal); To ***rotation matrix***: $\boldsymbol{R}(\alpha, \boldsymbol{n}) = \boldsymbol{I}_3 + \sin(\alpha)[\boldsymbol{n}]^\times + (1 - \cos(\alpha))([\boldsymbol{n}]^\times)^2$

To ***quaternion***: $\boldsymbol{q} = [\boldsymbol{n}, \alpha]$

Def **Quaternions** $q = q_w + q_x i + q_y j + q_z k$ with $i^2 = j^2 = k^2 = -1$, ($ij = -ji = k$, same for jk and ki).

$\boldsymbol{q} = \left[\boldsymbol{v}(\boldsymbol{q}), a(\boldsymbol{q})\right]^\top$, $a(\boldsymbol{q}) = q_w$, Add like $\mathbb{C}$ (by "groups")

Mult $\boldsymbol{q} \otimes \boldsymbol{p} = \begin{bmatrix} a(\boldsymbol{q})\boldsymbol{v}(\boldsymbol{p}) + a(\boldsymbol{p})\boldsymbol{v}(\boldsymbol{q}) + \boldsymbol{v}(\boldsymbol{q}) \times \boldsymbol{v}(\boldsymbol{p}) \\ a(\boldsymbol{q})a(\boldsymbol{p}) - \boldsymbol{v}(\boldsymbol{q})^\top \boldsymbol{v}(\boldsymbol{p}) \end{bmatrix}$

To Rot Mat $R_{AB} = \boldsymbol{I}_3 + 2a(\boldsymbol{q}_{AB})[\boldsymbol{v}(\boldsymbol{q}_{AB})]^\times + 2([\boldsymbol{v}(\boldsymbol{q}_{AB})]^\times)^2$

Def **Transf. M** $\boldsymbol{T}_{AC} = \boldsymbol{T}_{AB}\boldsymbol{T}_{BC}$ Right-Handed typically; Aero: Left-H

$$\boldsymbol{T}_{AB} = \begin{bmatrix} \boldsymbol{R}_{AB} & {}_A\boldsymbol{t}_B \\ \boldsymbol{0}_{1\times 3} & 1 \end{bmatrix}; \boldsymbol{T}_{BA} = \boldsymbol{T}_{AB}^{-1} = \begin{bmatrix} \boldsymbol{R}_{AB}^\top & -\boldsymbol{R}_{AB}^\top {}_A\boldsymbol{t}_B \\ \boldsymbol{0}_{1\times 3} & 1 \end{bmatrix}$$

2.2 Forward Kinematics (FK)

$\boldsymbol{T}_{WB_n}(\boldsymbol{\theta}) = \boldsymbol{T}_{WB_0}\boldsymbol{T}_{B_0B_1}(\theta_1)\cdots\boldsymbol{T}_{B_{n-1}B_n}(\theta_n)$.

For 2R system: ${}_W\boldsymbol{t}_{WE} = \begin{bmatrix} L_1\cos(\theta_1) + L_2\cos(\theta_1+\theta_2) \\ L_1\sin(\theta_1) + L_2\sin(\theta_1+\theta_2) \end{bmatrix}$

Workspace W: $\theta_1, \theta_2 \in [-\pi, \pi]$. ***Computation*** Similar for nR sys (more angles, more lengths). Last dim is typically sum of angles (or equiv). ***For velocity kinematics***: See 2.7.

Def **Singularity** Loss of deg of Freedom $\det(\boldsymbol{J}(\boldsymbol{\theta})) = 0$

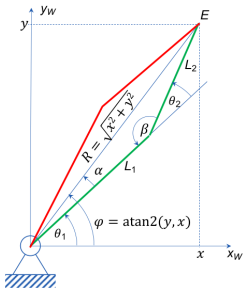
2.3 Inverse Kinematics (IK)

Option: Solve FK for angles.

Better: Law of cosine with polar coordinates. Compute angle using cosine rule, $\theta_1 = \varphi \pm \alpha$, $\theta_2 = \pm(\pi - \beta)$ (Positive for Elbow Down, Neg. for Elbow Up)

Extension to 6R: 1. Waist: spherical coords (2 sol.)
2. 2 sols from 2R for shoulder + elbow
3. Solve for wrist joints (no infl. on pos)

For Inv. velocity: $\dot{\boldsymbol{\theta}} = J_S^{-1}(\boldsymbol{\theta}){}_W\boldsymbol{t}_{WE}$



Def **Underactuated** workspace < 6D; IK solvable

Def **Overactuated** ∞ IK sol; Reduced singularities; Num. sol.

2.4 Temporal Models

Notation Linearization typically written as δX.

Model ***robot dyn*** as ***Cont-time non-lin. system of ODE***: $\dot{\boldsymbol{x}} = \boldsymbol{f}_C(\boldsymbol{x}(t), \boldsymbol{u}(t), \boldsymbol{w}(t))$, measurement $\boldsymbol{z}(t) = \boldsymbol{h}(\boldsymbol{x}(t)) + \boldsymbol{v}(t)$.
With: $\frac{\partial}{\partial t}\boldsymbol{x}(t) = f_C(\boldsymbol{x}(t), \boldsymbol{u}(t))$ the model for the robot state update and $\boldsymbol{h}(\boldsymbol{x}(t))$ the model for the measurements (e.g. for IMU)

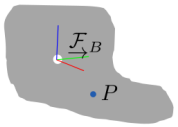
Linearised at $\boldsymbol{f}_C(\overline{\boldsymbol{x}}, \overline{\boldsymbol{y}}) = 0$, at ***equilibrium*** (if $\neq 0$, then add it):
$\delta\dot{\boldsymbol{x}}(t) = \boldsymbol{F}_C\delta\boldsymbol{x}(t) + \boldsymbol{G}_C\delta\boldsymbol{u}(t) + \boldsymbol{L}_C\boldsymbol{w}(t)$; $\delta\boldsymbol{z}(t) = \boldsymbol{H}\delta\boldsymbol{x}(t) + \boldsymbol{v}(t)$.
$\boldsymbol{F}_C$ system (often a Jac. / Taylor, vars are x_i. If no x in $\boldsymbol{F}$, then linear), $\boldsymbol{G}$ input gain (often Jac / Taylor, vars are u_i, if no u_i, then lin.), $\boldsymbol{w}$ process noise, $\boldsymbol{H}$ is measurement, $\boldsymbol{v}$ measurement noise, both zero-mean ***Gauss. White Noise Proc.***. $\boldsymbol{u}_k$ inputs at time k.

Discretized $\boldsymbol{x}_k = \boldsymbol{f}(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k, \boldsymbol{w}_k)$; $\boldsymbol{z}_k = \boldsymbol{h}(\boldsymbol{x}_k) + \boldsymbol{v}_k$
Linearised: $\delta\boldsymbol{x}_k = \boldsymbol{F}\delta\boldsymbol{x}_{k-1} + \boldsymbol{G}_k\delta\boldsymbol{u}_k + \boldsymbol{L}_k\boldsymbol{w}_k$; $\delta\boldsymbol{z}_k = \boldsymbol{H}_k\delta\boldsymbol{x}_k$
For discretized, iterative integral from t_{k-1} to t_k
Linearization happens typically with one of the below:

Def **Euler-Forward** $\boldsymbol{x}_k = \boldsymbol{x}_{k-1} + \Delta t\boldsymbol{f}_C(\boldsymbol{x}_{k-1}, \boldsymbol{u}_{k-1}, t_{k-1})$

Def **Trapezoidal num. int** $\Delta\boldsymbol{x}_1 = \Delta t\boldsymbol{f}_C(\boldsymbol{x}_{k-1}, \boldsymbol{u}_{k-1}, t_{k-1})$
$\Delta\boldsymbol{x}_2 = \Delta t f_C(\boldsymbol{x}_{k-1} + \Delta\boldsymbol{x}_1, \boldsymbol{u}_k, t_k)$, then:
$\boldsymbol{x}_k = \boldsymbol{x}_{k-1} + 0.5 \cdot (\Delta\boldsymbol{x}_1 + \Delta\boldsymbol{x}_2)$

2.5 Rigid body & IMU kinematics



Velocity ${}_I\boldsymbol{v}_{IB} = {}_I\dot{\boldsymbol{t}}_B$
Rot. Velocity ${}_I\boldsymbol{\omega}_{IB} = \dot{\alpha} \cdot {}_I\boldsymbol{n}$
Vel. point P ${}_B\boldsymbol{v}_{IP} = {}_B\boldsymbol{v}_{IB} + {}_B\boldsymbol{\omega}_{IB} \times {}_B\boldsymbol{t}_P$
Rotation Matrices

- For left pertubing $\dot{\boldsymbol{R}}_{IB} = [{}_I\omega_{IB}]^\times \boldsymbol{R}_{IB}$
- For right pertubing $\dot{\boldsymbol{R}}_{IB} = \boldsymbol{R}_{IB}[{}_I\omega_{IB}]^\times$
- Constant angular velocity ($\exp[\Delta\boldsymbol{\alpha}]^\times = \delta\boldsymbol{R}(\Delta\boldsymbol{\alpha}) \quad \boldsymbol{\alpha} = {}_I\boldsymbol{\omega}_{IB}\delta t$)
  $\boldsymbol{R}_{IB}(t + \Delta t) = \exp[\Delta\boldsymbol{\alpha}]^\times \boldsymbol{R}_{IB}(t)$

Quaternions

- For left pertubing $\dot{\boldsymbol{q}}_{IB} = \frac{1}{2}\begin{bmatrix} {}_I\boldsymbol{\omega}_{IB} \\ 0 \end{bmatrix} \otimes \boldsymbol{q}_{IB}$
- For right pertubing $\dot{\boldsymbol{q}}_{IB} = \frac{1}{2}\boldsymbol{q}_{IB} \otimes \begin{bmatrix} {}_B\boldsymbol{\omega}_{IB} \\ 0 \end{bmatrix}$

IMU (Outputs ${}_S\widetilde{\boldsymbol{a}}$ (accel.), ${}_S\widetilde{\boldsymbol{\omega}}$ (rot. accel.))
${}_W\dot{\boldsymbol{t}}_S = {}_W\boldsymbol{v}$; $\dot{\boldsymbol{q}}_{WS} = \frac{1}{2}\boldsymbol{q}_{WS} \otimes \begin{bmatrix} {}_S\widetilde{\boldsymbol{\omega}} + w_g - b_g \\ 0 \end{bmatrix}$
${}_W\dot{\boldsymbol{v}} = \boldsymbol{R}_{WS}({}_S\widetilde{\boldsymbol{a}} + w_a - b_a) + {}_W\boldsymbol{g}$ where gray parts only IRL (in theor. models, leave out), with $\dot{\boldsymbol{b}}_g = \boldsymbol{w}_{b_g}$ and $\dot{\boldsymbol{b}}_a = \boldsymbol{w}_{b_a}$
IMU Sensor Model: $\widetilde{\boldsymbol{z}} = \boldsymbol{b}_C + s\boldsymbol{M}\boldsymbol{z} + \boldsymbol{b} + \boldsymbol{n} + \boldsymbol{o}$ where bias $\boldsymbol{b}$ and scale s often modelled time-varying $\dot{b}(t) = \sigma_C n(t)$. $\boldsymbol{b}_C$ const. calibration / bias; $\boldsymbol{M}$ Misalignment; $\boldsymbol{n}$ noise; $\boldsymbol{o}$ other infl.

2.6 Rigid Body Dynamics

Def **Newton II** For fin. body w/ mass m and inertia mat. I, with force $\boldsymbol{F}$ and torque $\boldsymbol{T}$ on ***Centre of Mass*** (CoM), in body frame:

$$ {}_B\boldsymbol{F} = m({}_B\dot{\boldsymbol{v}}_{CoM}) + m_B\boldsymbol{\omega} \times {}_B\boldsymbol{v}_{CoM} $$
$$ {}_B\boldsymbol{T} = \boldsymbol{I}({}_B\dot{\boldsymbol{\omega}}) + {}_B\boldsymbol{\omega} \times {}_I\boldsymbol{B}\boldsymbol{\omega} $$

${}_B\boldsymbol{v}_{CoM}$ vel. of CoM, ${}_B\omega$ rot. speed; both w.r.t. inert. frame
To determine dynamics: ***(1)*** Define control inputs (e.g. wheel speeds), ***(2)*** Assumptions / Simplifications about state & inputs ***(3)*** List forces, torques, etc needed, ***(4)*** Express as func of states / inp ***(5)*** Reduce to the CoM ***(6)*** Plug into Newton-Euler eq combined with rigid body kinematics

2.7 Manipulator Velocity Kinematics

Differentiate FK $x = f(\boldsymbol{\theta})$, with $\dot{\boldsymbol{x}} = [{}_W\boldsymbol{v}, {}_W\boldsymbol{\omega}]^\top = \boldsymbol{J}(\boldsymbol{\theta})\dot{\boldsymbol{\theta}}$, with $\boldsymbol{J} = \frac{\partial f}{\partial \boldsymbol{\theta}}$ manipulator Jacobian. Then $\dot{\boldsymbol{\theta}} = J^{-1}\dot{\boldsymbol{x}}$ or $\dot{\boldsymbol{\theta}} = J^+\dot{\boldsymbol{x}}$, if over-actuated, with $J^+ = J^\top(JJ^\top)^{-1}$ (Moore Penrose-Inverse)

2.8 Legged robots

Legged robots don't need to maintain stability!

Def **Static Walking** 3 or more legs on ground, stable if frozen, safe, slow, inefficient.

Def **Dynamic Walking** $<$ 3 legs on ground, falls when not moving, fast, efficient, hard.

2.9 Wheeled robot Kinematics

Non-holonomic systems **not integrable**, no inst. move in every direct. ***Wheel constraints*** $v_i = \omega_i r_i$ (r_i constraints, (wheel radii))

- *Driving straight* all $\boldsymbol{v}$ equal (ICR: R.Cent.)
- *Turning* Wheel axis must intersect the ***Instant Centre of Rotation*** (ICR) of vehicle, speeds: $v_i \div R_i = \Omega$ (R_i = dist. wheel-ICR; Ω: vehicle rotation rate (around ICR))

To compute ICR, use $v_i \div R_i = \Omega$ and similarity.

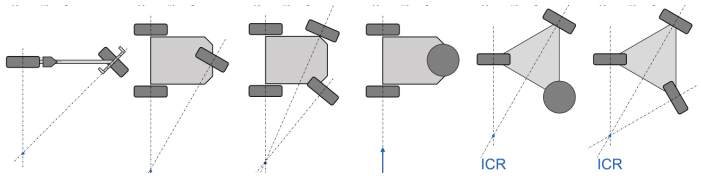
Below: α, l pos in frame, β rot at that pos (z-ax). To compute $\boldsymbol{c}$ in $\boldsymbol{c} \cdot {}_B\boldsymbol{v}_{WB} = \omega$ (For multiple wheels, construct mat. from this)

$c_i = [\sin(\alpha+\beta) \div r, -\cos(\alpha+\beta) \div r, -\cos(\beta) \cdot l \div r]$

Maneuverability

- Deg. of Mobility: $\delta_m = 3 - \#\text{constrained directions}$
- Deg. of Steerability: $\delta_s = \#\text{steerable wheels}$
- Deg. of Maneuverability: $\delta_M = \delta_m + \delta_s$

Wheel Configurations



Bicyle	Tricycle	Ackermann	Diff. Drive	Two-Steer	Three-Steer
$\delta_m = 1$	$\delta_m = 1$	$\delta_m = 0$	$\delta_m = 2$	$\delta_m = 1$	$\delta_m = 0$
$\delta_s = 1$	$\delta_s = 1$	$\delta_s = 2$	$\delta_s = 0$	$\delta_s = 2$	$\delta_s = 3$
$\delta_M = 2$	$\delta_M = 2$	$\delta_M = 2$	$\delta_M = 2$	$\delta_M = 3$	$\delta_M = 3$

2.9.1 Differential Drive Kinematics

State vec $\boldsymbol{x} = [x_1, x_2, \theta]^\top$, ***Inputs*** $\boldsymbol{u} = [\omega_l, \omega_r]^\top$, wheel radii r_l, r_r, w width of robot

Gen. eq. of Motion $\dot{x}_1 = v\cos(\theta)$, $\dot{x}_2 = v\sin(\theta)$, $\dot{\theta} = \Omega$, with $v = 0.5 \cdot (\omega_l r_l + \omega_r r_r)$, $\Omega = \frac{\omega_r r_r - \omega_l r_l}{w}$

Straight: $v = \omega_l r_l = \omega_r r_r$, $\Omega = 0$, $D = v\Delta t$.

Turning: $\Omega = (\omega_l r_l)/R_l = (\omega_r r_r)/R_r$, $R = v/\Omega$, $\Delta\theta = \Omega\Delta t$

Discretized: $\boldsymbol{x}_k = \boldsymbol{x}_{k-1} + b_i$ with $i \in \{s, t\}$, respectively, with $\boldsymbol{b}_s = \begin{bmatrix} D\cos(\theta) \\ D\sin(\theta) \\ 0 \end{bmatrix}$ $\boldsymbol{b}_t = \begin{bmatrix} R(\sin(\Delta\theta+\theta) - \sin(\theta)) \\ -R(\cos(\Delta\theta+\theta) - \cos(\theta)) \\ \Delta\theta \end{bmatrix}$; R dist ICR

Swedish Wheels: Have 3 DoF, typ. (3 pcs) equally spaced on circle. Typ. Param. each: α, β, γ (z, x, y axes, oriented along x, z up)

Wheel rot speed from movement Typ. comp: $J \cdot \boldsymbol{R}(\theta) \cdot \boldsymbol{x} = \dot{\varphi}$, with $\boldsymbol{R}(\theta) \cdot \boldsymbol{x}$ transform from base frame to robot frame.

3 Sensors & Actuators

Meas. Model: $\boldsymbol{z} = \boldsymbol{h}(\boldsymbol{x}) + \boldsymbol{v} + \boldsymbol{o}$, with $\boldsymbol{h}(\boldsymbol{x})$ determ. mean, $\boldsymbol{v}$ zero-mean noise, $\boldsymbol{o}$ unmodelled effects, $\boldsymbol{x}$ true state. $\boldsymbol{h}(\boldsymbol{x})$ describes how to compute x from known values (e.g. Intercept-Theorem). Probabilistic M.M: use random variable in definition, and s. 1.4

Motor encoders Typ. 64-2048 increments per rev; Estimate rot

Rolling-Shutter Most CMOS sensors don't take full image at once, need time stamp for each row

Def **Proprioceptive** Robot-internal states (e.g. IMU, encoders)

Def **Exteroceptive** Environment (e.g. LIDAR, Cameras)

3.1 GNSS

Need ultra-precise time sync ($c \approx 0.3\ m/ns$). ***Errors***

- Multipath problem (signal bounce) (0.5 - 100m)
- Ionosphere delays (10m)
- Satellite pos. err, trop. delay (1m)

3.2 Actuators

Hydraulic acc., easy control, power; maint., speed, price

Pneumatic price, shock abs., speed; acc., loud, maint.

3.2.1 DC Motor

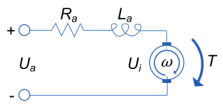
(Kirchoff) $U_a = L_a\dot{I}_a + R_a I_a + U_i$

(Torque, Lorenz Force) $T = k_T I_a$

(Induced V, Faraday) $U_i = k_i\omega$

(Mech. pow. = electric power)

$U_i I_a = k_i\omega I_a = T_\omega = k_T I_a \omega \Rightarrow k_i = k_T =: k$



3.3 Cameras

Def **Pinhole projection** $\begin{bmatrix} u & v \end{bmatrix}^\top = \frac{f}{z}\begin{bmatrix} x & y \end{bmatrix}^\top$ with f the distance to the lens and z the distance from object

$u = c_u + f \cdot x'$ and $v = c_v + f \cdot y'$ where $x' = t_x \div t_z$ and $y' = t_y \div t_z$ where u, v are the pixel x, y coords, $\boldsymbol{c} = [c_u, c_v]^\top$ is optical centre of cam in pixel coords, f scale factor.

The full proj: $\boldsymbol{u} = \begin{bmatrix} \lambda u \\ \lambda v \\ \lambda \end{bmatrix} = \begin{bmatrix} f & 0 & c_u \\ 0 & f & c_v \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} = \boldsymbol{K}\ {}_C\boldsymbol{t}_P$

If p. in diff frame, e.g. W-frame then $\boldsymbol{u} = \boldsymbol{K}[\boldsymbol{R}_{CW}\ {}_C\boldsymbol{t}_{CW}]{}_W\boldsymbol{t}_P$

3.3.1 Pinhole Camera Projection with distortion

Def Model: $\boldsymbol{u} = \boldsymbol{k}(\boldsymbol{d}(\boldsymbol{p}({}_C\boldsymbol{t}_P)))$, with $\boldsymbol{c}$ as above, k_i radial distortion params, optional for $i > 2$, p_i tangential distortion parameter, f_u, f_v x/y focal lengths in pixels. These should be given by task

(Projection) $\boldsymbol{x'} = \boldsymbol{p}({}_C\boldsymbol{t}_P) = t_z^{-1} \cdot [t_x, t_y]^\top$

(Distortion model) $r^2 = x'^2 + y'^2$, $\boldsymbol{x''} = \boldsymbol{d}(\boldsymbol{x'})$

$\boldsymbol{x''} = \frac{1+k_1 r^2 + k_2 r^4 + k_3 r^6}{1+k_4 r^2 + k_5 r^4 + k_6 r^6} \cdot \boldsymbol{x'} + \begin{bmatrix} 2p_1 x'y' + p_2(r^2 + 2x'^2) \\ p_1(r^2 + 2y'^2) + 2p_2 x'y' \end{bmatrix}$

(Scale and Centre) $\boldsymbol{u} = \boldsymbol{k}(\boldsymbol{x''}) = \text{diag}([f_u, f_v]) \cdot \boldsymbol{x''} + \boldsymbol{c}$

(To unit plane) $\boldsymbol{x''} = \boldsymbol{k}^{-1}(\boldsymbol{u}) = [f_u^{-1}, f_v^{-1}]^\top(\boldsymbol{u} - \boldsymbol{c})$

(Un-distort) $\boldsymbol{x'} = \boldsymbol{d}^{-1}(\boldsymbol{x''})$ (usually comp. numerically)

(Compute ray) ${}_C\boldsymbol{r} = [\boldsymbol{x'}, 1]^\top$

3.3.2 Undestorting a whole image

$\boldsymbol{u}_i = \boldsymbol{k}(\boldsymbol{d}(\boldsymbol{k}_{\text{new}}^{-1}(\boldsymbol{u}_{i,\text{new}})))$, where $\boldsymbol{u}_{i,\text{new}}$ is the place of pixel in output, $\boldsymbol{u}_i$ is the input, with k_{new} the new focal lengths and image centre

Omnidir. Cam undistortion model with $f(u, v) = \sum_{i=0}^{N} a_i \rho^i$ with $\rho = \sqrt{(u - c_u)^2 + (v - c_v)^2}$, $N = 4$ accurately describes it for most fisheye and catadioptric cameras

3.4 Depth and Range sensing

Typ. derive D.M. with sensor dir param (as vec line eq)

3.4.1 Triangulation-based

Struct. Light Single cam, single projector: Spatial acc, no worky in bright light, interference with other IR depth cams

Active Stereo 2 cams, 1 proj: worky in bright light, need stereo matching, less accurate, error grows with distance

3.4.2 Classic Stereo

Both images: same plane, focal length f, centre, x-axis. Given corresponding pixels $[u_l, v]$ and $[u_r, v]$: disparity (pixel offset) $d = u_r - u_l$; $z = \frac{b \cdot f}{d}$, then projected into left (right) cam, $u_l = f \cdot \frac{x}{z} + c_u$ or $u_r = f \cdot \frac{x-b}{z} + c_u$, b distance between cameras. Then often apply pinhole camera projection (to e.g. get 3D point coords, comp ${}_C\boldsymbol{r}$, then point $z \cdot {}_C\boldsymbol{r}$). Uncertainty introduced with 1.-order error propag. $\Delta z = \frac{\partial}{\partial d} z \Delta d$, with Δd error of d

3.4.3 Time of Flight, Projection

No occlusions/shadows, Interference with other dev, multipath leading to larger distances sensed

Proj. $\boldsymbol{z} = \begin{bmatrix} \boldsymbol{u} \\ d \end{bmatrix} = \begin{bmatrix} \boldsymbol{k}(\boldsymbol{d}(\boldsymbol{p}({}_C\boldsymbol{t}_P))) \\ [0,0,1]\,{}_c\boldsymbol{t}_P \end{bmatrix}$ ***Back***: ${}_C\boldsymbol{t}_P = \begin{bmatrix} d\boldsymbol{x'} \\ d \end{bmatrix}$ This x' is obtained from pinhole cam projection.

3.4.4 Range Sensors

Ultrasonic Typ. freq: 40kHz - 180kHz, Range: 12cm - 5m, Acc: $\approx$ 2cm, rel error $\approx$ 2% meas. for transp. surf., cheap(ish), Cone wider, reflect. angle dep, wind / currents

LiDAR Time-of-Flight-based, Accuracy, range, works in bright light, Complex, expensive, one timestamp per measurement. Typical Ranges: up to 100m

4 Multi-Sensor Estimation

4.1 Linearization

$\boldsymbol{f}(\boldsymbol{x}) \approx \boldsymbol{f}(\overline{\boldsymbol{x}}) + \boldsymbol{J_f}\big|_{x=\overline{x}}(\boldsymbol{x} - \overline{\boldsymbol{x}})$, f', no vec in 1D; $\overline{\boldsymbol{x}}$ lin. p.

Def **Jacobian** $\boldsymbol{J_f}$ rows for eq of $\boldsymbol{f}$; cols for vars of each eq. It may also be a single value (if just one var in the state)

Def **Gradient** $\nabla \boldsymbol{f}$ is vec, each comp. for par diff of var

Part. diff; Approx. using finite differences $\frac{f(\overline{x}+h) - f(\overline{x})}{h}$,

or central differences (vector of $\frac{\boldsymbol{f}(\overline{\boldsymbol{x}}) + h_i \boldsymbol{e_i}}{h_i}$, with $\boldsymbol{e_i}$ unit vec)

4.2 Linear Least Squares

Goal: $\underset{x \in \mathbb{R}^n}{\text{argmin}} ||\boldsymbol{A}\boldsymbol{x} - b||_2^2$, $\boldsymbol{A}$: rows i-th datap. col c: t_i^{c-1}.

Alt: compute sum of squared errors S, then min. S, i.e. solve system $\frac{\partial}{\partial \alpha} S = 0$, $\frac{\partial}{\partial r} S = 0$ for $\{\alpha, r\}$ the params

Man. sol.: comp. $M = A^\top A$, $b' = A^\top b$, then $Mx = b'$.

Numpy: `numpy.linalg.lstsq`

Prob. sol.: argmax $\mathbb{P}(\boldsymbol{x} \mid \boldsymbol{z})$ with Maximum ...

- ***Likelihood*** $\mathbb{P}(\boldsymbol{x}|\boldsymbol{z}) \propto \mathbb{P}(\boldsymbol{z}|\boldsymbol{x}) = \prod_{i=1}^{N} \mathbb{P}(\boldsymbol{z}_i|\boldsymbol{x})$
- ***a Post*** $\mathbb{P}(\boldsymbol{x}|\boldsymbol{z}) \propto \mathbb{P}(\boldsymbol{z}|\boldsymbol{x})\mathbb{P}(\boldsymbol{x}) = \mathbb{P}(\boldsymbol{x}) \prod_{i=1}^{N} \mathbb{P}(\boldsymbol{z}_i|\boldsymbol{x})$

Gauss-Dist (non)-linear meas. processed in batches, MLE is weighted non-linear Least Squares

4.3 Non-Linear Least Squares

Find $\boldsymbol{x}^* = \text{argmax}\, \mathbb{P}(\boldsymbol{x}|\boldsymbol{z}) = \text{argmin}(-\log(\mathbb{P}(\boldsymbol{x}|\boldsymbol{z})))$

Gauss-Newton Need func F and its Jacobian (or derivative) DF

```
def gauss_newton(x: np.ndarray, F, DF, tol=1e-6):
    s = np.linalg.lstsq(DF(x), F(x))[0] # least sq.
    x = x-s; k = 1
    while np.linalg.norm(s) > tol * np.linalg.norm(x):
        s = np.linalg.lstsq(DF(x), F(x))[0]
        x = x-s; k += 1 # k optional for max iter
    return x, k
```

Levenberg-Marquardt (1) Pick start point $\overline{\boldsymbol{x}}^0$ and start param $\lambda^0 = \max \text{diag}(\boldsymbol{A})$ and v (e.g. $v = 2$).; ***(2)*** Modified GN sys: $\boldsymbol{A} + \lambda \text{diag}(\boldsymbol{A}))\Delta\boldsymbol{x} = \boldsymbol{b}$; ***(3)*** Solve for $\Delta\boldsymbol{x}$; ***(4)*** Update: $\overline{\boldsymbol{x}}^{k+1} = \overline{\boldsymbol{x}}^k + \Delta\boldsymbol{x}$ (if cost reduced), else: $\overline{\boldsymbol{x}}^{k+1} = \overline{\boldsymbol{x}}^k$, $\lambda^{k+1} = \lambda^k v$, go to step 3; ***(5)*** Check convergence, else go to step 2

Robust Cost Functions Account for outliers, by mod. err. terms

4.4 Bayes Filter (in DAG)

$\boldsymbol{x}_k^R$ state at time k, $\boldsymbol{z}_k^p$ dist. meas., $\boldsymbol{u}_k^p$ wheel odometry (= meas.). Typically care about current state: alternate predict & update. Prediction with Product/Sum rule for $\mathbb{P}[\boldsymbol{x}_k^R \mid \boldsymbol{u}_{1:k}^p, \boldsymbol{z}_{1:k-1}^d]$, Update with Bayes' Theorem for $\mathbb{P}[\boldsymbol{x}_k^R \mid \boldsymbol{u}_{1:k}^p, \boldsymbol{z}_{1:k}^d]$

4.5 Particle Filter

Is a Bayes filter approximating the state with S weighted particles $\{x_{k,s}, w_{k,s}\}$. Distribution not necessarily unimodal.

Predict: Push each particle through transition model; Update step:

- Apply Bayes rule $w'_{k,s} = \mathbb{P}[\boldsymbol{z}_i \mid \boldsymbol{x}_{k,s}] w_{k-1,s}$
- Renormalize: $w_{k,s} = w'_{k,s} \div \sum_s w'_{k,s}$
- Resample: rand. sel. S particles acc. to weights, $w_{k,s} = S^{-1}$

4.6 Kalman Filter (KF)

Bayes Filter for Gauss. dist of R.V. & linear meas. model.

Initial state $\boldsymbol{x}_0 \sim \mathcal{N}(\widehat{\boldsymbol{x}}, \boldsymbol{P}_0)$, $\boldsymbol{P}_0$ previous covariance

Prediction With lin. state trans. model ($\boldsymbol{u}_k$ wheel spd, $\boldsymbol{w}_k$ noise, covariance $\boldsymbol{Q}_k$). $\boldsymbol{x}_k = \boldsymbol{F}\boldsymbol{x}_{k-1} + \boldsymbol{G}\boldsymbol{u}_k + \boldsymbol{L}\boldsymbol{w}_k$ with $\boldsymbol{w}_k \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{Q}_k)$:

- ***Mean*** $\widehat{\boldsymbol{x}}_{k|k-1} = \boldsymbol{F}\widehat{\boldsymbol{x}}_{k-1} + \boldsymbol{G}\boldsymbol{u}_k$
- ***Covariance*** $\boldsymbol{P}_{k|k-1} = \boldsymbol{F}\boldsymbol{P}_{k-1|k-1}\boldsymbol{F}^\top + \boldsymbol{L}\boldsymbol{Q}_k\boldsymbol{L}^\top$

Update Lin. meas.: $\widetilde{\boldsymbol{z}}_k = \boldsymbol{H}\boldsymbol{x}_k + \boldsymbol{v}_k$ with $\boldsymbol{v_k} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{R}_k)$:

- ***Measurement residual***: $\boldsymbol{y}_k = \widetilde{\boldsymbol{z}}_k - \boldsymbol{H}\widehat{\boldsymbol{x}}_{k|k-1}$
- ***Residual Covariance***: $\boldsymbol{S}_k = \boldsymbol{H}\boldsymbol{P}_{k|k-1}\boldsymbol{H}^\top + \boldsymbol{R}_k$
- ***Kalman gain***: $\boldsymbol{K}_k = \boldsymbol{P}_{k|k-1}\boldsymbol{H}^\top\boldsymbol{S}_k^{-1}$
- ***Updated mean***: $\widehat{\boldsymbol{x}}_{k|k} = \widehat{\boldsymbol{x}}_{k|k-1} + \boldsymbol{K}_k\boldsymbol{y}_k$
- ***Updated Cov.***: $\boldsymbol{P}_{k|k} = (\boldsymbol{I} - \boldsymbol{K}_k\boldsymbol{H})\boldsymbol{P}_{k|k-1}$

All with G the input model, L noise map, $\widetilde{\boldsymbol{z}}_k$ actual measurement, K trust measurement (R_k small vs $P_{k|k-1}$ small).

4.7 Extended Kalman Filter (EKF)

Non-l. state trans. model $\boldsymbol{x}_k = \boldsymbol{f}(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k, \boldsymbol{w}_k)$ as above:

- ***Mean***: $\widehat{\boldsymbol{x}}_{k|k-1} = \boldsymbol{f}(\widehat{\boldsymbol{x}}_{k-1|k-1}, \boldsymbol{u}_k)$
- ***Covariance***: $\boldsymbol{P}_{k|k-1} = \boldsymbol{F}_k\boldsymbol{P}_{k-1|k-1}\boldsymbol{F}_k^\top + \boldsymbol{L}_k\boldsymbol{Q}_k\boldsymbol{L}_k^\top$

With $\boldsymbol{F}_k = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}$ and $\boldsymbol{L}_k = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{w}}$ (computed as are Jacobians)

Update Non-Linear measurement: $\widetilde{\boldsymbol{z}}_k = \boldsymbol{h}(\boldsymbol{x}_k) + \boldsymbol{v}_k$:

- ***Measurement residual***: $\boldsymbol{y}_k = \widetilde{\boldsymbol{z}}_k - \boldsymbol{h}(\widehat{\boldsymbol{x}}_{k|k-1})$

Difference to KF: $\boldsymbol{H}$ becomes $\boldsymbol{H}_k$, and $\boldsymbol{H}^\top$ is $\boldsymbol{H}_k^\top$. They are linearizations of $\boldsymbol{h}$, see 2.4

Limitations Linearization error (shifts bias or sharpens it), fixed by UKF (by passing charact. points around mean through nonlin. func instead of Jac) or Iterative EKF

E **Diff-Drive Lawn Mower** Known: Radii (r_l and r_r) and track w. Measurements: wheel increments ($\Delta\varphi_l$ and $\Delta\varphi_r$). State $[x, y, \theta]^\top$. Can compute distance, heading change & turn radius.

For ***State Transition***: $D = 0.5(r_l\Delta\varphi_l + r_r\Delta\varphi_r)$, rest same as in 2.9.1. F_k is of form $[1, 0, \boldsymbol{d}_1; 0, 1, \boldsymbol{d}_2; 0, 0, 1]$, with $\boldsymbol{d} = b_s$ or $\boldsymbol{d} = b_t$ for straight and turning, respectively.

Updates are linear (GPS: $[x, y]^\top$ and compass: θ, matrix: $\boldsymbol{I}$)

5 SLAM to Spatial AI

SLAM = Simultaneous Localization and Mapping

5.1 Keypoints

Corner det. $SSD(\Delta x, \Delta y) \approx [\Delta_x\ \Delta_y]\boldsymbol{M}[\Delta_x\ \Delta_y]^\top$ λ_i E.V. of M; κ const 0.04-0.15; $\boldsymbol{R} = \det(M) - \kappa \cdot \text{TR}(M)^2 = \lambda_1\lambda_2 - \kappa(\lambda_1 + \lambda_2)^2$; $\boldsymbol{M} = \sum_{x,y\in P} \begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix} = R^\top \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} R$

Blob Detection (I is the image)

Laplacian of Gaussian (LoG): $L = g(x, y, t) \cdot I(x, y)$. Then apply Laplacian Operator $\nabla^2_{\text{norm}} L = t\left(\frac{\partial^2 L}{\partial x^2} + \frac{\partial L}{\partial y^2}\right)$

Diff. of Gaussians (DoG): $\Delta L = L(x, y, t) - L(x, y, kt)$

SIFT Detector (1) Subsample + Blur ***(2)*** DoG on each res. image ***(3)*** Keypoints extrema in DoG pyramid

BRISK / binary descriptors: Compare pixel intensities at fixed sampling pattern around keypoint. Match by Hamming distance (very fast), ***SuperPoint***: CNN learns detector + descriptor

5.2 Bootstrapping

PnP Problem Perspective n-Point Find sol. for camera pose *directly*

RANSAC RANdom SAmpling Consensus Model estimation, for find. outliers & correct. Also great for finding an initial guess for pose. (due to robustness) For N iteration: sample min set, fit model, count inliers (reprojection error $<$ threshold t), keep best, optionally refit on inliers. More outliers $\Rightarrow$ more iterations.

Stereo Triang. Given two rays (known poses for points in 2D). Find good point in 3D. Fast sol: ***Midpoint Method***:

1 Find p. along ray w/ min. dist (Lin. Least Squares)

$\boldsymbol{\lambda} = [\lambda_1\ \lambda_2]^\top = \text{argmin}||({}_W\boldsymbol{t}_{C_2} + \lambda_2{}_W\boldsymbol{e}_2) - ({}_W\boldsymbol{t}_{C_1} + \lambda_1{}_W\boldsymbol{e}_2)||^2$

2 Solve normal equation $\boldsymbol{A}\boldsymbol{\lambda} = \boldsymbol{b}$ with $\boldsymbol{q} = -{}_W\boldsymbol{e}_1^\top{}_W\boldsymbol{e}_2$:

$$\boldsymbol{A} = \begin{bmatrix} 1 & q \\ q & 1 \end{bmatrix} \quad \boldsymbol{b} = \begin{bmatrix} \boldsymbol{e}_1^\top \cdot ({}_W\boldsymbol{t}_{C_2} - {}_W\boldsymbol{t}_{C_1}) \\ -\boldsymbol{e}_2^\top \cdot ({}_W\boldsymbol{t}_{C_2} - {}_W\boldsymbol{t}_{C_1}) \end{bmatrix} \quad C_i \text{ cam}$$

3 Pick midp. ${}_W\boldsymbol{t}_P = 0.5(\tau_1 + \tau_2)$; $\tau_n = {}_W\boldsymbol{t}_{C_n} + \lambda_n{}_W\boldsymbol{e}_n)$

5.3 Place Recognition

Idea: Build vocab from "visual words" (in Training). ***Runtime*** Detect keypoints; Extract descriptors; Build histogram; Query DB for similarity; If no match, insert into DB, else use to compute with e.g. RANSAC

5.4 Mapping

Problem Formulations *Localisation* (always static given map):

$\boldsymbol{x}^*_{R,k} = \text{argmax}\, \mathbb{P}(x_{R,k} \mid \boldsymbol{x}_M, \boldsymbol{z}_{1:k}, \boldsymbol{u}_{1:k})$ (recursive)

$\boldsymbol{x}^*_{R,1:k} = \text{argmax}\, \mathbb{P}(x_{R,1:k} \mid \boldsymbol{x}_M, \boldsymbol{z}_{1:k}, \boldsymbol{u}_{1:k})$ (batch)

SLAM $\{\boldsymbol{x}^*_{R,k}, \boldsymbol{x}^*_M\} = \text{argmax}\, \mathbb{P}(\boldsymbol{x}_R, \boldsymbol{x}_M \mid \boldsymbol{z}_{1:k}, \boldsymbol{u}_{1:k})$ (as ↑)

Mapp.: $\boldsymbol{x}^*_M = \text{argmax}\, \mathbb{P}(\boldsymbol{x}_M \mid \boldsymbol{x}^*_{R,1:k}, \boldsymbol{z}_{1:k}, \boldsymbol{u}_{1:k})$ with given poses $\boldsymbol{x}^*_{R,1:k}$. Thus temp. model $\boldsymbol{u}_{1:k}$ doesn't matter.

Prob. Occ. Grid $\mathbb{P}(f_j) = \mathbb{P}(\neg o_j) = 1 - \mathbb{P}(o_j)$, with $\mathbb{P}(o_j)$ prob. cell j occupied; pairwise independent.

Occ. Map. w/ depth sensor $\mathbb{P}(o_j \mid \boldsymbol{x}_{R,1:k}) = \frac{\mathbb{P}(o_j \mid \boldsymbol{x}_{R,k}, \boldsymbol{z}_k)\mathbb{P}(\boldsymbol{z}_k \mid \boldsymbol{x}_{R,k})\mathbb{P}(o_j \mid \boldsymbol{x}_{R,1:k-1}, \boldsymbol{z}_{1:k-1})}{\mathbb{P}(o_j)\mathbb{P}(\boldsymbol{z}_k \mid \boldsymbol{x}_{R,1:k}, \boldsymbol{z}_{1:k-1})}$

map prior; Prev. occ. est.; Occ. based on curr. range meas.;

Update function: $l(a) := \log(\text{Odds}(a))$ with $\text{Odds}(a) = \frac{\mathbb{P}(a)}{1-\mathbb{P}(a)}$ (inv. sensor model): $l(o_j|\boldsymbol{x}_{R,1:k}, \boldsymbol{z}_{1:k}) = l(o_j|\boldsymbol{x}_{R,k}, \boldsymbol{z}_k) + l(o_j|\boldsymbol{x}_{R,1:k-1}, \boldsymbol{z}_{1:k-1}) - l(o_j)$

In 3D 3D voxel j as signed dist. s and weight w, update: $s_k = \frac{w_{k-1}s_{k-1} + \widetilde{s}_k}{w_{k-1} + 1}$ with $w_k = \min(w_{\max}, w_{k-1} + 1)$

Implementation Using Hash maps or octree (dense grid inefficient)

5.4.1 Iterative Closest Point

Build *correspondences*: associate all live scan points l_i to closest map points m_i. Error term: $\boldsymbol{e} = \boldsymbol{T}_{WL_lL_l}m_i - {}_W\boldsymbol{l}_i$. Minimize this via Gauss-Newton, then re-associate, iterate.

Photometric: $u_{KF} = \pi(\boldsymbol{T}^{-1}_{WC_{KF}}\boldsymbol{T}_{WC_l}\pi^{-1}(\boldsymbol{u}_{KF}, \boldsymbol{D}_{KF}[\boldsymbol{u}_{KF}]))$, error term $\boldsymbol{e} = \boldsymbol{I}_{KF}[\boldsymbol{u}_{KF}] - \boldsymbol{I}_L[\boldsymbol{u}_L]$, where all subscript L are from live image, all subscript KF key frame.

5.5 Dense Tracking and Loop Closure

Idea: Min. sum of sq. errors over all pixels w/ Gauss-Newton.

6 Planning & Control

6.1 SISO & MIMO

Single-Input-Single-Output: r Reference (e.g. speed), u Sys. Input (e.g. gas pedal pos), z Sys. Out. (e.g. speed of car)

Multiple-Input-Multiple-Output: r Reference (typ: trajectory), u Sys. Input (e.g. 4 rotor speeds), x internal states (pos, orient, speed, rot. speed), z Sys. Output (e.g. speed of car)

6.2 Proportional-Integral-Differential (PID)

$u(t) = K_pe(t) + K_i\int_{t_0}^t e(\tau)\ \text{d}\tau + K_d\frac{\text{de}(t)}{\text{dt}}$, where params K_p (curr), K_i (long-term), K_d (trend) reduce corresp. errors, with $e = r - y$

Drone Control ${}_B\boldsymbol{F}$ and ${}_B\boldsymbol{M}$ as in sect. 2.6, use $\boldsymbol{u}' = {}_B\boldsymbol{M}$.

6.3 Linear Quadratic Regulator (LQR)

For lin. cont.-time dyn. $\dot{\boldsymbol{x}}(t) = \boldsymbol{F}_c\boldsymbol{x}(t) + \boldsymbol{G}_c\boldsymbol{u}(t)$.

We try to minimise cost functional (for $\boldsymbol{u}(t) = -\boldsymbol{K}\boldsymbol{x}(t)$)

$$J = \int_t^\infty \boldsymbol{x}(\tau)^\top\boldsymbol{Q}\boldsymbol{x}(\tau) + \boldsymbol{u}(\tau)^\top\boldsymbol{R}\boldsymbol{u}(\tau)\ \text{d}\tau$$

Solution: $\boldsymbol{K} = \boldsymbol{R}^{-1}(\boldsymbol{G}_c^\top\boldsymbol{P})$, with $\boldsymbol{P}$ found from

$$\boldsymbol{F}_c^\top\boldsymbol{P} + \boldsymbol{P}\boldsymbol{F}_c - (\boldsymbol{P}\boldsymbol{G}_c)\boldsymbol{R}^{-1}(\boldsymbol{G}_c^\top\boldsymbol{P}) + \boldsymbol{Q} = \boldsymbol{0}$$

Finding $\boldsymbol{K}$ is expensive, but *offline*, at runtime only $\boldsymbol{u}(t)$.

Non-Lin: Approx, $\delta\dot{\boldsymbol{x}}(t) = \boldsymbol{F}_c\delta\boldsymbol{x}(t) + \boldsymbol{G}_c\delta\boldsymbol{u}(t)$

6.4 MPC

Cost function ($p(\boldsymbol{x}_N)$ *terminal cost*, sum the *stage cost*)

$$J_{0\to N}(\boldsymbol{x}_0, \boldsymbol{u}_0, \dots, \boldsymbol{u}_{N-1}) = p(\boldsymbol{x}_N) + \sum_{k=0}^{N-1} q(\boldsymbol{x}_k, \boldsymbol{u}_k)$$

We minimize the above s.t. for $k \in \{0, \dots, N-1\}$ we have $\boldsymbol{x}_{k+1} = \boldsymbol{f}(\boldsymbol{x}_k, \boldsymbol{u}_k)$, $\boldsymbol{g}(\boldsymbol{x}_k, \boldsymbol{u}_k) \leq \boldsymbol{0}$ and $\boldsymbol{x}_N \in \mathcal{X}_f$ and $\boldsymbol{x}_0 = \boldsymbol{x}(0)$

Finite-Horizon Lin-Quad Control Quad. Cost:

$$J_{0\to N}(\boldsymbol{x}_0, \boldsymbol{u}_0, \dots) = \boldsymbol{x}_N^\top \boldsymbol{P} \boldsymbol{x}_N + \sum_{k=0}^{N-1} \boldsymbol{x}_k^\top \boldsymbol{Q} \boldsymbol{x}_k + \boldsymbol{u}_k^\top \boldsymbol{R} \boldsymbol{u}_k$$

without constraints, ***State Feedback Law*** $\boldsymbol{u}_0^* = -\boldsymbol{K}\boldsymbol{x}(0)$. With constraints, minimize as above, $\boldsymbol{x}_{k+1} = \boldsymbol{F}\boldsymbol{x}_k + \boldsymbol{G}\boldsymbol{u}_k$

6.5 Motion Planning

Config. Space: $\mathcal{C}$. Subset of robot states for planning.
Free C-Space: $\mathcal{C}_{\text{free}}$, ***C-Space Obstacles*** $\mathcal{C}_{\text{obst}}$ (occupied)
Collision checker: $c(\boldsymbol{x}) : \mathcal{C} \to \{0, 1\}$
Visibility graph: Connect corners, goal outside obstacles
Voronoi Diagram: Edges at max. dist. from obst. (benefit: safer paths). Also tends to be faster than Dijkstra.
Discrete via graph/grid: complete solution, Curse of dimensions
Continuous probabilistic: High-D okay, differential constraints ok, Probabilistic, depending on setup may not find correct sol. or run forever

6.5.1 A* Algorithm

Function `h(...)` is a lower bound of optimal cost.

```
1  procedure AStar(Graph, start, goal)
2      for each v in Graph.Verticies do
3          dist[v] ← ∞
4          totDistEst[v] ← ∞
5          prev[v] ← undef
6      insert start into openSet
7      dist[start] ← 0
8      totDistEst[start] ← h(start, goal)
9      while openSet not empty do
10         u ← vert in openSet w/ min totDistEst
11         remove u from openSet
12         if u == goal then
13             return dist[u], prev
14         for each neighbour v of u do
15             alt ← dist[u] + G.Edges(u, v).dist
16             if alt < dist[v] then
17                 dist[v] ← alt
18                 totDistEst[v] ← alt + h(v, goal)
19                 insert v into openSet
20                 prev[v] ← u
21     return ∞, prev
```

6.5.2 Exploration

Find action sequence by finding goals, planning collision-free paths there, choose goal/path with maximum utility (typically max map information)

6.5.3 Rapidly-Exploring Random Tree (RRT)

```
1  procedure RRT(start, goal)
2      InsertVertex(start, Graph)
3      for k < MAX_ITER do
4          s ← scal. unif. rand. sample on [0, 1]
5          if s < GOAL_CONNECT_PROB then
6              x ← goal
7          else
8              x ← random coll.-free config
9          x_n ← NearestConfiguration(Graph, x)
10         x_f ← StoppingConfiguration(x_n, x)
11         if x_f ≠ x_n then
12             insertVertexGraph, x_f
13                                  ▷ Extension of RRT* goes here
14             insertEdgeGraph, x_n, x_f
15         if x_f==x_n then
16             return SUCCESS, Graph
17     return FAILURE, Graph
```

Returns a collision-free path as graph. Need nearest neighbour search. Extension to RRT* to make path better:

```
1  X_near ← Neighbours(Graph, x_f, R)
2  x_min ← Neighbours(Graph, x_f, R)
3  c_min ← Cost(x_n) + EdgeCost(x_n, x_f)
4  for each x_near in X_near do
5      if EdgeCollisionFree(x_near, x_f) and
       Cost(x_near) + EdgeCost(x_near, x_f) < c_min then
6          InsertEdge(Graph, x_f, x_near)
7          x_parent ← Parent(Graph, x_near)
8          RemoveEdge(Graph, x_parent, x_near)
```

Informed RRT* extension: Once conn. betw. start and goal found, restrict sampling to (hyper)ellipsoid. $b = 0.5d^2 - ||x_s - x_g||^2$.

6.5.4 Collision Avoidance

Dynamic Window Approach Assume: Robot moves inst. on circ. arcs (v, ω). Compute arcs with coll. Accounts for Kino-Dyn, Cost func prone to loc. min, assumes static obj
Velocity Obstacles Assume: Robot in str. line (v_x, v_y). Comp pos with coll. Vel. of obj, prone to local optima, no kino-dyn.
Potential Field Methods Define *repulsive* and attractive potential $c = c_{\text{att}} + c_{\text{rep}}$. With e.g. $c_{\text{att}} = \frac{1}{2} k_{\text{att}} ||\boldsymbol{x} - \boldsymbol{x}_{\text{goal}}||^2$ and $c_{\text{rep}} = \begin{cases} \frac{1}{2} k_{\text{rep}} \left(\frac{1}{\rho(\boldsymbol{x})} + \frac{1}{\rho_{\text{lim}}} \right) & \rho \leq \rho_{\text{lim}} \\ 0 & \text{else} \end{cases}$ Simple control laws, may trap in loc. min., no diff. const, no guar. to avoid coll

6.6 Learning To Act

Used if there is no state-transition model or cost/reward func.

6.6.1 Markov Decision Process

The goal is to maximize the reward. Along the route, get small reward, at the end large reward (good or bad).
Def. by states $\boldsymbol{x} \in \mathcal{X}$ (RL: s), actions $\boldsymbol{u} \in \mathcal{U}$ (RL: a), prob. state trans. $\mathcal{T}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{x}_+) = \mathbb{P}(\boldsymbol{x}_+ \mid \boldsymbol{x}, \boldsymbol{u})$, reward func $\mathcal{R}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{x}_+)$, start state $\boldsymbol{x}_0$, optional terminal state $\boldsymbol{x}_N$.
Utility Func Expected reward: $V = \sum_{k=0}^{N} r_k$ or *discounted* reward $V = \sum_{k=0}^{\infty} \gamma^k r_k$ with $\gamma < 1$ (if inf. runtime)

Solving (Val iter) $V_0(\boldsymbol{x}) = 0$ and $V_{i+1}(\boldsymbol{x}) = \max_{\boldsymbol{u}} Q(\boldsymbol{x}, \boldsymbol{u})$ with

$$Q(\boldsymbol{x}, \boldsymbol{u}) = \sum_{\boldsymbol{x}_+} \mathbb{P}(\boldsymbol{x}_+ \mid \boldsymbol{x}, \boldsymbol{u}) [\mathcal{R}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{x}_+) + \gamma V_i(\boldsymbol{x}_+)]$$

Repeat until conv. to V^* ($\mathcal{O}\left(|\mathcal{U}||\mathcal{X}|^2\right)$ per iter). Optimal policy: $\boldsymbol{\pi}^*(\boldsymbol{x}) = \text{argmax}_{\boldsymbol{u}} Q(\boldsymbol{x}, \boldsymbol{u})$

Using ***policy iter***:

```
1  Choose π_0(x)
2  while policy has not converged do
3      repeat V_{i+1}^{π_j}(x) = Q(x, π(x)) ∀x and fixed pol. π_j
4      until values converge
5  One step: π_{j+1}(x) = argmax_u Q(x.u) with V_i = V_{i+1}^{π_j}
```

Model-based learning uses empirical models of $\mathcal{T}$ and $\mathcal{R}$

6.6.2 Reinforcement Learning

Passive *Direct Evaluation* Act according to policy $\boldsymbol{\pi}$, store sum of discounted rewards, average them. (But too simple)
Sample-Based Use $V_{i+1}^{\boldsymbol{\pi}}(\boldsymbol{x}) = Q(\boldsymbol{x}, \boldsymbol{\pi}(\boldsymbol{x}))$ w/ $V_0^{\boldsymbol{\pi}}(\boldsymbol{x}) = 0$. We need state trans. model, instead $\widetilde{R}_j$ (approx. prob. w/ statistics) and thus

$$V_{i+1}^{\boldsymbol{\pi}}(\boldsymbol{x}) = \frac{1}{N} \sum_{j=1}^{N} \widetilde{R}_j(\boldsymbol{x}, \boldsymbol{\pi}(\boldsymbol{x}), \boldsymbol{x}_+) + \gamma V_i^{\boldsymbol{\pi}}(\boldsymbol{x}_+)$$

Active Find optimal policy $\boldsymbol{\pi}$ instead of state values $V(\boldsymbol{x})$.
Q-Learn. Here: restate Val. Iter in Q-Values ($Q_0(\boldsymbol{x}, \boldsymbol{u}) = 0$):

$$Q_{i+1}(\boldsymbol{x}, \boldsymbol{u}) = Q(\boldsymbol{x}, \boldsymbol{u}) \quad \text{with } V_i(\boldsymbol{x}_+) = \max_{\boldsymbol{u}}(\boldsymbol{Q}_i(\boldsymbol{x}_+, \boldsymbol{u}))$$

Compute using sample:

$$\widetilde{Q}_i(\boldsymbol{x}, \boldsymbol{u}) = \widetilde{R}_i(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{x}_+) + \gamma \max_{\boldsymbol{u}}(\boldsymbol{Q}_i(\boldsymbol{x}_+, \boldsymbol{u}))$$

Then update: $Q_{i+1}(\boldsymbol{x}, \boldsymbol{u}) = (1 - \alpha) Q_i(\boldsymbol{x}, \boldsymbol{u}) + \alpha \widetilde{Q}_i(\boldsymbol{x}, \boldsymbol{u})$. Called off-policy learning, needs exploration. Simplest is random actions (ε-greedy): ε is prob. to act randomly, $1 - \varepsilon$ is prob. to act on pol. Space explored, still doing random stuff
DL-Approaches *Model-based* (estimate trans. model, e.g. Dyna), Value-based (estimate val or Q-func and extract pol., e.g. Q-Learn), Actor-Critic (estim. val or Q of curr. pol., improve pol., e.g. A3C, SAC), Policy-Gradient (diff. expect. reward w.r.t. params of policy network, e.g. REINFORCE)