1 Mathematical Foundations

Skew-Symmetric Operator:
$\boldsymbol{r}^\times = \begin{bmatrix} 0 & -r_3 & r_2 \\ r_3 & 0 & -r_1 \\ -r_2 & r_1 & 0 \end{bmatrix}$ $\boldsymbol{x}^T\boldsymbol{w}^\times\boldsymbol{x} = 0$ $\boldsymbol{C}\boldsymbol{w}^\times\boldsymbol{C}^T = [\boldsymbol{C}\boldsymbol{w}]^\times, \boldsymbol{C} \in SO(3)$

Singularity/Constraint Dilemma: There is no representation of rotations that has exactly 3 parameters (i.e. no constraints) and also no singularities.

Rotation Matrices: $\boldsymbol{C}_{21} = \boldsymbol{\mathcal{F}}_2 \cdot \boldsymbol{\mathcal{F}}_1^T \implies \boldsymbol{r}_2 = \boldsymbol{C}_{21}\boldsymbol{r}_1$. $\boldsymbol{C}^T\boldsymbol{C} = \boldsymbol{1}, \det \boldsymbol{C} = 1$.

Principal Rotations: Transforms between this frame and a frame obtained by rotating about an axis by an angle. *Note the notation here is the opposite of what's used in most other courses. The matrices used to rotate a vector about a coordinate axis are the inverse of the matrices here.*

$\boldsymbol{C}_1(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix}$ $\boldsymbol{C}_2(\theta) = \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix}$ $\boldsymbol{C}_3(\theta) = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Euler Angles: One of 12 compositions of principal rotations. Has singularities.

Axis-Angle: $\boldsymbol{C} = \cos(\phi)\boldsymbol{1} + (1 - \cos\phi)\boldsymbol{a}\boldsymbol{a}^T - \sin\phi\boldsymbol{a}^\times$ constraint $\boldsymbol{a}^T\boldsymbol{a} = 1$.

Quaternion: $\boldsymbol{q} = q_0 + \mathbf{q} = q_0 + q_1 i + q_2 j + q_3 k \in \mathbb{H}$ are normalized to be unit length on $\mathbb{S}^3$, which forms a **double cover** of $SO(3)$. $\boldsymbol{q}$ is the same rotation as $-\boldsymbol{q}$. $\bar{\boldsymbol{q}}$ (conjugate) is the opposite rotation. $\boldsymbol{q} = \begin{bmatrix} \eta \\ \boldsymbol{\varepsilon} \end{bmatrix} = \begin{bmatrix} \cos(\phi/2) \\ \boldsymbol{a}\sin(\phi/2) \end{bmatrix}$.

$\boldsymbol{C} = (\eta^2 - \boldsymbol{\varepsilon}^T\boldsymbol{\varepsilon})\boldsymbol{1} + 2\boldsymbol{\varepsilon}\boldsymbol{\varepsilon}^T - 2\eta\boldsymbol{\varepsilon}^\times$.

Gibbs Vector: $\boldsymbol{C} = (1 + \boldsymbol{g}^\times)^{-1}(1 - \boldsymbol{g}^\times), \boldsymbol{g} = \boldsymbol{a}\tan\frac{\phi}{2}$

Infinitesimal Rotations: $\boldsymbol{C} = 1 - \boldsymbol{\theta}^\times$ where $\boldsymbol{\theta} = \phi\boldsymbol{a}$.

Velocity in Another Frame: $\underset{\circ}{\boldsymbol{r}} = \underset{\circ}{\boldsymbol{r}}^\circ + \underset{\circ}{\boldsymbol{\omega}}_{21} \times \underset{\circ}{\boldsymbol{r}} \iff \dot{\boldsymbol{r}}_1 = \boldsymbol{C}_{12}(\dot{\boldsymbol{r}}_2 + \boldsymbol{\omega}_2^{21\times}\boldsymbol{r}_2)$

Poisson's Equation: $\dot{\boldsymbol{C}}_{21} = -\boldsymbol{\omega}_2^{21\times}\boldsymbol{C}_{21}$

Transforms: $\boldsymbol{r}_i^{pi} = \boldsymbol{C}_{iv}\boldsymbol{r}_v^{pv} + \boldsymbol{r}_i^{vi} \Leftrightarrow \begin{bmatrix} \boldsymbol{r}_i^{pi} \\ 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{C}_{iv} & \boldsymbol{r}_i^{vi} \\ \boldsymbol{0}^T & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{r}_v^{pv} \\ 1 \end{bmatrix} = \boldsymbol{T}_{iv} \begin{bmatrix} \boldsymbol{r}_v^{pv} \\ 1 \end{bmatrix}$

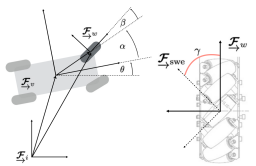
Note vehicle pose is $\boldsymbol{T}_{iv}$ (vehicle-to-inertial), not $\boldsymbol{T}_{vi}$!

Inversion: $\boldsymbol{T}_{iv}^{-1} = \begin{bmatrix} \boldsymbol{C}_{iv}^T & -\boldsymbol{C}_{iv}^T\boldsymbol{r}_i^{vi} \\ \boldsymbol{0}^T & 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{C}_{vi} & -\boldsymbol{r}_v^{vi} \\ \boldsymbol{0}^T & 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{C}_{vi} & \boldsymbol{r}_v^{iv} \\ \boldsymbol{0}^T & 1 \end{bmatrix} = \boldsymbol{T}_{vi}$

Combined Velocity: $\boldsymbol{\varpi}_v^{vi} = \begin{bmatrix} \boldsymbol{\nu}^{vi} \\ \boldsymbol{\omega}_v^{vi} \end{bmatrix}, \dot{\boldsymbol{T}}_{vi} = \begin{bmatrix} \boldsymbol{\omega}_v^{vi\times} & -\boldsymbol{\nu}_v^{vi} \\ \boldsymbol{0}^T & 0 \end{bmatrix} \boldsymbol{T}_{vi}$

2 Vehicle Modelling

2.1 Wheel Kinematics



For a standard wheel with radius r, rotation φ:

$$\begin{bmatrix} \cos(\alpha+\beta) & \sin(\alpha+\beta) & d\sin\beta \end{bmatrix} \dot{\boldsymbol{\xi}} = \dot{\varphi} r$$

$$\begin{bmatrix} -\sin(\alpha+\beta) & \cos(\alpha+\beta) & d\cos\beta \end{bmatrix} \dot{\boldsymbol{\xi}} = 0$$

where $\dot{\boldsymbol{\xi}} = \begin{bmatrix} v & u & \omega \end{bmatrix}^T$ is the vehicle-frame pose rate. Solving for $\dot{\varphi}$ yields **inverse differential kinematics**. Solving for $\dot{\boldsymbol{\xi}}$ (by pseudoinverse $(\boldsymbol{A}^T\boldsymbol{A})^{-1}\boldsymbol{A}^T$) yields **forward differential kinematics**.

For Swedish wheels with rollers at angle γ ($\dot{\varphi}_s$ usually unconstrained):

$$\begin{bmatrix} \cos(\alpha+\beta+\gamma) & \sin(\alpha+\beta+\gamma) & d\sin(\beta+\gamma) \end{bmatrix} \dot{\boldsymbol{\xi}} = \dot{\varphi} r\cos\gamma + \dot{\varphi}_s r_s$$

$$\begin{bmatrix} -\sin(\alpha+\beta+\gamma) & \cos(\alpha+\beta+\gamma) & d\cos(\beta+\gamma) \end{bmatrix} \dot{\boldsymbol{\xi}} = -\dot{\varphi} r\sin\gamma$$

Differential Drive Kinematics: Origin halfway between wheels (b from wheels).

$$\begin{bmatrix} \dot{\varphi}_r \\ \dot{\varphi}_l \end{bmatrix} = \frac{1}{r}\begin{bmatrix} 1 & b \\ 1 & -b \end{bmatrix}\begin{bmatrix} v \\ \omega \end{bmatrix} \qquad \begin{bmatrix} v \\ \omega \end{bmatrix} = \frac{1}{2}\begin{bmatrix} r & r \\ r/b & -r/b \end{bmatrix}\begin{bmatrix} \dot{\varphi}_r \\ \dot{\varphi}_l \end{bmatrix}$$

2.2 Euler-Lagrange

Nonholonomic constraints written as $\boldsymbol{H}(\boldsymbol{q})^T\dot{\boldsymbol{q}} = \boldsymbol{0}$ where $\boldsymbol{q} = \begin{bmatrix} x & y & \theta \end{bmatrix}^T$. Null space of $\boldsymbol{H}(\boldsymbol{q})^T$ are admissible velocities $\dot{\boldsymbol{q}} = \boldsymbol{G}(\boldsymbol{q})\boldsymbol{p}$ where $\boldsymbol{p}$ is generalized velocity and $\boldsymbol{H}(\boldsymbol{q})^T\boldsymbol{G}(\boldsymbol{q}) = \boldsymbol{0}$.

Euler-Lagrange Equation: $\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial^T L}{\partial \dot{\boldsymbol{q}}}\right) - \frac{\partial^T L}{\partial \boldsymbol{q}} = \boldsymbol{\tau} + \boldsymbol{H}(\boldsymbol{q})\boldsymbol{\lambda}$ where $L = T - V$, $\boldsymbol{\tau} = \boldsymbol{G}(\boldsymbol{q})\boldsymbol{\nu}$ (generalized forces), $\boldsymbol{\lambda}$ are Lagrange multipliers. To eliminate the Lagrange multiplier term, premultiply by $\boldsymbol{G}(\boldsymbol{q})^T$.

2.3 Vehicle Dynamics Models

Generalized Vehicle Model: $\dot{\boldsymbol{x}} = \boldsymbol{A}(\boldsymbol{x})\boldsymbol{x} + \boldsymbol{B}(\boldsymbol{x})\boldsymbol{u}$ where $\boldsymbol{x} = \begin{bmatrix} \boldsymbol{q} & \boldsymbol{p} \end{bmatrix}^T$; with kinematics only, $\boldsymbol{A} = \boldsymbol{0}$.

Unicycle Model: $\begin{bmatrix} \dot{\boldsymbol{q}} \\ \dot{\boldsymbol{p}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{0} & \boldsymbol{G}(\boldsymbol{q}) \\ \boldsymbol{0} & -\boldsymbol{M}^{-1}\boldsymbol{D} \end{bmatrix}\begin{bmatrix} \boldsymbol{q} \\ \boldsymbol{p} \end{bmatrix} + \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{M}^{-1} \end{bmatrix}\boldsymbol{u}$ where

$\boldsymbol{p} = \begin{bmatrix} v \\ \omega \end{bmatrix}$ $\boldsymbol{G}(\boldsymbol{q}) = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix}$ $\boldsymbol{M} = \begin{bmatrix} m & 0 \\ 0 & I \end{bmatrix}$ $\boldsymbol{u} = \begin{bmatrix} f \\ g \end{bmatrix}$ $\boldsymbol{\nu} = -\boldsymbol{D}\boldsymbol{p} + \boldsymbol{u}$

for some damping $\boldsymbol{D}$, thrust f, steering torque g.

Bicycle Model: $\begin{bmatrix} \dot{\boldsymbol{q}} \\ \dot{\boldsymbol{p}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{0} & \boldsymbol{G}(\boldsymbol{q}) \\ \boldsymbol{0} & -\boldsymbol{M}(\boldsymbol{q})^{-1}\boldsymbol{D}(\boldsymbol{q},\boldsymbol{p}) \end{bmatrix}\begin{bmatrix} \boldsymbol{q} \\ \boldsymbol{p} \end{bmatrix} + \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{M}(\boldsymbol{q})^{-1} \end{bmatrix}\boldsymbol{u}$ where

$\boldsymbol{q} = \begin{bmatrix} x \\ y \\ \theta \\ \gamma \end{bmatrix}$ $\boldsymbol{p} = \begin{bmatrix} v \\ \varsigma \end{bmatrix}$ $\boldsymbol{G}(\boldsymbol{q}) = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ \frac{\tan\gamma}{d} & 0 \\ 0 & 1 \end{bmatrix}$ $\boldsymbol{M} = \begin{bmatrix} m & 0 & 0 & 0 \\ 0 & m & 0 & 0 \\ 0 & 0 & I + I_f & I_f \\ 0 & 0 & I_f & I_f \end{bmatrix}$

and $\boldsymbol{M}(\boldsymbol{q}) = \boldsymbol{G}(\boldsymbol{q})^T\boldsymbol{M}\boldsymbol{G}(\boldsymbol{q}), \boldsymbol{D}(\boldsymbol{q},\boldsymbol{p}) = \boldsymbol{G}(\boldsymbol{q})^T\boldsymbol{M}\dot{\boldsymbol{G}}(\boldsymbol{q}) + \boldsymbol{D}$ for front wheel steering angle γ, rear wheel velocity v, steering speed ς, distance d between wheels.

3 Path-Tracking Control

Errors: Crosstrack error $\varepsilon_L = y_d - y$ (distance between vehicle and closest point on path) and heading error $\varepsilon_H = \theta_d - \theta$ (based on path tangent at closest point).

Feedback Linearization: Given a nonlinear system $\dot{x} = f(x) + g(x)u$, define new input η and find input mapping $u = a(x) + b(x)\eta$ such that $\dot{x} = f(x) + g(x)(a(x) + b(x)\eta) = Ax + B\eta$, i.e. find A, B such that $f(x) + g(x)a(x) = Ax, g(x)b(x)\eta = B\eta$.

Applied to unicycle model: $\begin{bmatrix} \dot{\varepsilon}_L \\ \dot{\varepsilon}_H \end{bmatrix} = \begin{bmatrix} v\sin\varepsilon_H \\ \omega \end{bmatrix}$ (tangent approximation), change variables $\begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} \varepsilon_L \\ v\sin\varepsilon_H \end{bmatrix}$ to get $\begin{bmatrix} \dot{z}_1 \\ \dot{z}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} z_1 \\ z_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix}\eta$ where $\eta = v\omega\cos\varepsilon_H$. Let output $y = \begin{bmatrix} \alpha & 1 \end{bmatrix}\begin{bmatrix} z_1 \\ z_2 \end{bmatrix}$ and proportional controller $\eta = K(y_d - y)$, take $K = 4\alpha$ to get $\omega = -\frac{4\alpha^2}{v}\frac{\varepsilon_L}{\cos\varepsilon_H} - 4\alpha\tan\varepsilon_H$.

This has singularities at $\varepsilon_H = \pm\pi/2$ or $v = 0$, and overshoots turns.

Pure Pursuit: Steer the robot towards a target lookahead point. Lookahead is hard to select for varying speeds; too short results in oscillation; too long results in slow convergence.

Stanley Controller: For bicycle models, using front axle centre as origin, $\delta(t) = \varepsilon_H(t) + \tan^{-1}\left(\frac{k\varepsilon_L(t)}{k_s + v_f(t)}\right)$ for some steering $\delta(t)$ and forward velocity $v_f(t)$, gain k and softening k_s.

Error dynamics: using $\dot{\varepsilon}_H(t) = \frac{-v_f(t)\sin(\delta(t))}{l}, \dot{\varepsilon}_L(t) = v_f(t)\sin(\varepsilon_H(t) - \delta(t))$, crosstrack error has $\dot{\varepsilon}_L(t) = \frac{-k\varepsilon_L(t)}{\sqrt{1 + \left(\frac{k\varepsilon_L(t)}{v_f}\right)^2}}$, exponential decay for small ε_L.

Model Predictive Control: At each step, predict all future errors for some horizon, find optimal sequence of control inputs, apply the first step (or for a short time) and iterate again.

Parametrize control over horizon: $\begin{bmatrix} v(t) \\ \omega(t) \end{bmatrix} = \begin{bmatrix} v_0 \\ \omega_0 \end{bmatrix} + \begin{bmatrix} v_1 \\ \omega_1 \end{bmatrix}t + \begin{bmatrix} v_2 \\ \omega_2 \end{bmatrix}t^2 + \begin{bmatrix} v_3 \\ \omega_3 \end{bmatrix}t^3$ with $v_{0..3}, \omega_{0..3}$ as parameters. Simulate with full model to minimize cost function $J(\boldsymbol{u}) = \frac{1}{2}\sum_i w_i\varepsilon_{L,i}^2 + \frac{1}{2}\sum_k \left(w_{v,k}v_k^2 + w_{\omega,k}\omega_k^2\right)$.

4 Planning

General Approach: A strategic/high-level (global) planner, planning an optimal route based on a priori knowledge, and a tactical/low-level (reactive) planner that handles real-time obstacle avoidance.

Standard Planning Sequence: Compute a simple path, smooth the path to account for kinematics, design a trajectory to account for dynamics (i.e. adding a velocity profile), and design a controller to track the trajectory.

4.1 Reactive Planners

Potential Fields: Define a potential $V(x_t) = V_{\text{att}}(x_t) + V_{\text{rep}}(x_t)$ where $V_{\text{att}}(q) = K_{\text{att}}\rho(q, q^g)^2$ and $V_{\text{rep}}(q) = K_{\text{rep}}\sum_{i=1}^{n}\frac{1}{\rho(q, O^i)^2}$ and use gradient descent (move in the direction of $-\nabla V$). Can get stuck in local minima; non-optimal; does not account for vehicle constraints.

Extended Potential Fields: Incorporate vehicle heading by adding a rotation potential, which increases if heading towards obstacles.

Trajectory Rollout: A basic form of MPC; simulates a discrete number of trajectories in a short window, checking each for collisions and scoring based on progress to goal, distance to obstacles, similarity to previous choice, etc. Accounts for vehicle constraints but can still get stuck.

4.2 Global Planners

Ideal graph-based planners should be complete, optimal, and efficient.

Resolution Completeness: For grid-based planners, if a path exists it will always be found given a fine enough grid resolution.

Probabilistic Completeness: For probabilistic planners, if a path exists, the failure probability asymptotically approaches zero as more work is performed.

Anytime Optimality: As more work is performed, better paths are found, i.e. planning can be terminated early for a feasible but less optimal path.

4.2.1 Graph Search Methods

Backward Value Iteration: Working backwards from the final state using dynamic programming to find optimal cost-to-go. At each step for each node iterate as $G^*(x) = \min_u\{l(x,u) + G^*(f(x,u))\}$.

Generic Forward Search Algorithm: At each step, take the next state from the queue, mark it as visited, discover all unvisited nodes reachable from this node, and add each reachable unvisited node to the queue. Queue prioritization leads to different algorithms: LIFO (stack) — DFS; FIFO (queue) — BFS; lowest cost-to-come — Dijkstra's algorithm.

A* Algorithm: Prioritize with cost-to-come plus lower bound heuristic (necessary for optimality). After reaching the goal, prune all nodes in queue with a priority worse than the goal and run until queue is empty.

Lifelong Planning A* (LPA*): Reuse results from previous A* runs when only a few edge costs have changed. All cells that have an incoming edge cost change are checked for inconsistency, by recomputing the cost-to-come based on predecessors. Inconsistent cells are set to infinite cost and successors are checked, until all cells are consistent, when the search can be restarted.

Other Improvements: Focused Dynamic A* (D*), D*-Lite are faster than LPA*; Field D* makes smoother paths.

4.3 Probabilistic Roadmaps

PRM Algorithm: Initialize with start and goal nodes. Repeatedly sample configurations randomly and check if they are feasible, and add as milestones if so. For each milestone try to connect to neighbours; if path is collision-free add it as an edge.

Visibility Set: $V(q) = \{q' \mid (q, q') \in F\}$, i.e. the set of all configurations that can be connected to q by a straight line edge. Size of $V(q)$ is called **expansiveness**. Summing over all $q \in C$ produces expansiveness of the entire configuration space. High expansiveness indicates open spaces, wider, and straighter passages.

Visibility Graph: Each vertex on each obstacle is a milestone and connected to each other possible vertex. Produces theoretically optimal paths in 2D but not higher dimensions.

Workspace-Guided Sampling: Find narrow passages in workspace and map into the configuration space to sample those areas more.

Gaussian Sampling: First sample uniformly, then another configuration with a Gaussian distribution centered at the first; if one sample is in free space and the other is in collision, retain the one in free space, otherwise skip. Samples tend to be near the boundaries of free space.

Bridge Sampling: Sample 2 configurations using the same procedure as Gaussian sampling; if both are in collision, find the midpoint between the samples, and retain that sample if it's in free space. Samples tend to be in narrow passages, which may scale better in higher dimensions.

Bounding Volume Hierarchy: Enclose objects into successively tighter bounding volumes, and checking collisions starting from the largest. If in collision, check again with all sub-volumes enclosed within until the desired level of accuracy is reached. Bounding volumes can be computed in advance in static environments.

Adaptive Collision Checks: Each point checked along the path eliminates a section of the path based on distance to closest obstacle. Check end points first, then bisect any remaining sections.

Lazy Collision Checks: Delay collision checking until path has been found; if in collision, recompute the path.

4.4 RRT and RRT*

RRT Algorithm: Sample a point in space, identify the closest node and the input that drives from the node towards the sampled point. Forward simulate for a short time to determine the new vehicle configuration. If no collision, add the new node to the tree. Probabilistically complete but not optimal.

Probability of expanding each vertex is proportional to its Voronoi region, which heavily favours expansion over refinement. Sample within an expanding box of the origin to encourage more refinement.

RRT* Algorithm: Each node carries a cost-to-come. After connecting the new node, consider nodes within a ball of the new node. If the new node can be connected to another node within the neighbourhood to get a lower cost, connect to this node instead. Then, check all other nodes within the ball to see if going from the new node to the other node results in a lower cost. If so, rewire the node so its parent becomes the new node. Results in asymptotically optimal paths.

Informed RRT*: After finding an initial solution, use a heuristic to form an ellipse where possible path improvements can be, and sample only within this space.

4.5 Planning for Autonomous Driving

Structure: Roadmap level mission planner, action-level behavioural planner (FSM, rule-based, reinforcement learning), path and velocity-level local planner (lattice in structured scenes, sampling-based in unstructured scenes, variational $\min_{\delta x} J(x + \delta x)$ for optimality, init with global planner solution).

Conformal Lattice Planner: Fits control actions/grid to the road. Solve with boundary conditions (x, y, θ, κ) with max curvature, min velocity constraints.

Cubic Spirals: Numerically integrate $\kappa(s) = a_3 s^3 + a_2 s^2 + a_1 s + a_0 \implies \theta(s) = \theta_0 + \int_0^s \kappa(s')\mathrm{d}s', x(s) = x_0 + \int_0^s \cos(\theta(s'))\,\mathrm{d}s', y(s) = y_0 + \int_0^s \sin(\theta(s'))\,\mathrm{d}s'$. Specify boundary constraints, check curvature at 1/3, 2/3 points. For optim remap params to 4 control points and arc length and use soft constraints.

Objectives: Path length $s_f = \int_{x_i}^{x_f}\sqrt{1 + \left(\frac{\mathrm{d}y}{\mathrm{d}x}\right)^2}\,\mathrm{d}x$, time $T_f = \int_0^{s_f}\frac{1}{v(s)}\,\mathrm{d}s$, bending energy $\int_0^{s_f}\|\kappa(s)\|^2\,\mathrm{d}s$ (distributes curvature).

Velocity Generation: Track reference and minimize $\int_0^{s_f}\|v(s) - v_{ref}(s)\|\,\mathrm{d}s$ or Hinge loss (penalize too fast, not too slow). Penalize jerk $\int_0^{s_f}\|\dddot{x}(s)\|^2\,\mathrm{d}s$ for smoothness. Get desired v_f based on minimum of v_{ref}, lead vehicle speed, and acceleration within friction ellipse/comfort rectangle ($a_{lat} = v^2\kappa$). Connect to current v_0 with linear ($v_f = \sqrt{2as + v_0^2}$) or trapezoidal ramp.

5 Probability

Total Probability: $\int_{a_N}^{b_N}\cdots\int_{a_1}^{b_1} p(x_1, \ldots, x_N)\,\mathrm{d}x_1 \cdots \mathrm{d}x_N = 1$

Bayes' Rule: $p(\boldsymbol{x}|\boldsymbol{y}) = \frac{p(\boldsymbol{y}|\boldsymbol{x})p(\boldsymbol{x})}{p(\boldsymbol{y})}$

Moments:

0. Always 1 by total probability.
1. Mean: $\boldsymbol{\mu} = \mathbb{E}[\boldsymbol{x}] = \int \boldsymbol{x}p(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}$.
2. (Co)variance: $\boldsymbol{\Sigma} = \mathbb{E}[(\boldsymbol{x} - \boldsymbol{\mu})(\boldsymbol{x} - \boldsymbol{\mu})^T]$.
3. Skewness
4. Kurtosis

Independence: $p(\boldsymbol{x}, \boldsymbol{y}) = p(\boldsymbol{x})p(\boldsymbol{y})$; implies uncorrelated.

Uncorrelated: $\mathbb{E}[\boldsymbol{x}\boldsymbol{y}^T] = \mathbb{E}[\boldsymbol{x}]\mathbb{E}[\boldsymbol{y}]^T \iff \boldsymbol{\Sigma}_{xy} = \boldsymbol{0}$

Shannon Information: $H(\boldsymbol{X}) = -\mathbb{E}[\ln p(\boldsymbol{x})] = -\int_a^b p(\boldsymbol{x})\ln p(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}$

5.1 Gaussians

PDF: $p(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^N \det\boldsymbol{\Sigma}}}\exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x} - \boldsymbol{\mu})\right)$

Factoring: $p(\boldsymbol{x}, \boldsymbol{y}) = \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu}_x \\ \boldsymbol{\mu}_y \end{bmatrix}, \begin{bmatrix} \boldsymbol{\Sigma}_{xx} & \boldsymbol{\Sigma}_{xy} \\ \boldsymbol{\Sigma}_{yx} & \boldsymbol{\Sigma}_{yy} \end{bmatrix}\right) = p(\boldsymbol{x}|\boldsymbol{y})p(\boldsymbol{y})$

$$p(\boldsymbol{x}|\boldsymbol{y}) = \mathcal{N}(\boldsymbol{\mu}_x + \boldsymbol{\Sigma}_{xy}\boldsymbol{\Sigma}_{yy}^{-1}(\boldsymbol{y} - \boldsymbol{\mu}_y), \boldsymbol{\Sigma}_{xx} - \boldsymbol{\Sigma}_{xy}\boldsymbol{\Sigma}_{yy}^{-1}\boldsymbol{\Sigma}_{yx})$$
$$p(\boldsymbol{y}) = \mathcal{N}(\boldsymbol{\mu}_y, \boldsymbol{\Sigma}_{yy})$$

Direct Product: $\boldsymbol{\Sigma}^{-1} = \sum_{k=1}^{K}\boldsymbol{\Sigma}_k^{-1}, \boldsymbol{\mu} = \boldsymbol{\Sigma}\sum_{k=1}^{K}\boldsymbol{\Sigma}_k^{-1}\boldsymbol{\mu}_k$

Linearization: $p(\boldsymbol{y}) = \int p(\boldsymbol{y}|\boldsymbol{x})p(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} \approx \mathcal{N}(\boldsymbol{g}(\boldsymbol{\mu}_x), \boldsymbol{R} + \boldsymbol{G}\boldsymbol{\Sigma}_{xx}\boldsymbol{G}^T)$ where $p(\boldsymbol{x}) = \mathcal{N}(\boldsymbol{\mu}_x, \boldsymbol{\Sigma}_{xx})$, $p(\boldsymbol{y}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{g}(\boldsymbol{x}), \boldsymbol{R})$, and $\boldsymbol{G} = \left.\frac{\partial \boldsymbol{g}(\boldsymbol{x})}{\partial \boldsymbol{x}}\right|_{\boldsymbol{x}=\boldsymbol{\mu}_x}$.

Shannon Information: $H(\boldsymbol{X}) = \frac{1}{2}\ln((2\pi e)^N \det\boldsymbol{\Sigma})$; $\sqrt{\det\boldsymbol{\Sigma}}$ is the uncertainty ellipsoid volume.

6 Sensors

Taxonomy: Interoceptive (measures internal states) or exteroceptive (measures external states); active (emits energy) or passive (measures ambient energy).

6.1 Interoceptive Sensors

Inertial Measurement Unit (IMU): Measures proper acceleration (relative to freefall) and angular velocity in 3 axes. MEMS sensors use capacitive spring-mass systems and measures stretching or Coriolis forces of an oscillating mass. Sensitive to vibration. Calibration needed to estimate time-varying biases.

Compass/Magnetometer: Measures heading using magnetic field. Must account for geographical variation. Easily disrupted by motors etc.

Sun Sensor: Measures angle to a celestial body. Usually used outside Earth.

Inclinometer: Measures pitch and roll using gravity. Affected by acceleration.

Encoder: Measures axial rotation. Absolute: unique binary pattern for each sector; relative: pulse for each increment (quadrature uses 2 sensors a half slit apart).

6.2 Exteroceptive Sensors

LiDAR: Measures distance using time-of-flight of laser pulses, determined by measuring phase shift of an intensity modulated signal. Beams steered using spinning heads, mirrors, or MEMS. Measurement model: $\rho_{ij} = \sqrt{x_{ij}^2 + y_{ij}^2 + z_{ij}^2} + \Delta\rho$,

$$\theta_{ij} = \tan^{-1}\left(\frac{y_{ij}}{x_{ij}}\right) + \Delta\theta,\ \alpha_{ij} = \tan^{-1}\left(\frac{z_{ij}}{\sqrt{x_{ij}^2 + y_{ij}^2}}\right) + \Delta\alpha.$$

ToF Camera: Like LiDAR but captures the entire point cloud at once. Much shorter range; suitable for indoor use since sunlight causes problems.

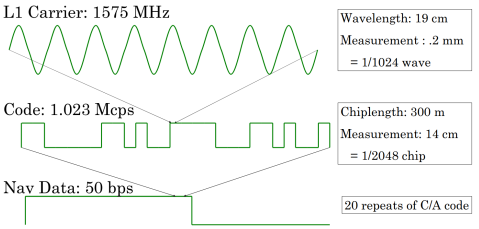
Sonic Rangefinder/SODAR: Measures distance using ToF of ultrasonic pulse. Limited range, resolution, and rate. Popular underwater.

Radar: Measures distance and velocity using radio waves. Good for distance, unreliable for angle.

GNSS/GPS: Measures position using distance to satellites. At least 4 for solution (3 position + 1 clock offset). Satellites use atomic clocks and correct for relativistic effects. Uses fixed WGS84 globe model.

Base carrier frequency at 1575 MHz, modulated to carry a code (chips) at 1.023 Mcps; 1023 chips form a unique orthogonal gold code identifying the satellite (repeated at 1 kHz). Correlation with code determines ToF. 20 code repeats encodes 1 bit (50 bps), carrying orbital ephemeris corrections, time, etc. Raw distances (pseudoranges) are used to solve for position and clock bias using NLLS.

Errors from satellite geometry (dilution of precision), ionospheric/tropospheric effects, multi-path, giving 10 m accuracy. DGPS uses known base station to eliminate atmospherics and satellite clock error (10 cm relative to base). RTK GPS also uses carrier phase (2 cm).



6.3 Cameras

6.3.1 Pinhole Camera Model

Basic Projective Map: Normalized by focal length,

$$\bar{\boldsymbol{x}} = \mathcal{P}_z(\boldsymbol{p}) = \boldsymbol{p}/p_z = \begin{bmatrix} x/z & y/z & 1 \end{bmatrix}^T$$

Intrinsic Matrix: For focal lengths f_x, f_y, center c_x, c_y (pixels), skew s:

$$\begin{bmatrix} x_s \\ y_s \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x/z \\ y/z \\ 1 \end{bmatrix} = \boldsymbol{K}\frac{\boldsymbol{p}}{z}$$

This projects 3D to pixel coords. Note one can also normalize after projection. One can obtain a ray of points that project to the same pixel by multiplying the augmented pixel vector by $\boldsymbol{K}^{-1}$.

FoV, Sensor Width, Focal Length: $\tan\frac{\theta}{2} = \frac{W}{2f}$

Extrinsic Matrix: $\boldsymbol{E}_{CW} \in SE(3)$ contains the world frame to camera frame transform. $\boldsymbol{E}_{CW} = \boldsymbol{E}_{WC}^{-1}$ where $\boldsymbol{E}_{WC}$ is the camera pose in world frame.

Projective Matrix: $\tilde{\boldsymbol{P}} = \begin{bmatrix} \boldsymbol{K} & \boldsymbol{0} \\ \boldsymbol{0}^T & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{C} & \boldsymbol{t} \\ \boldsymbol{0}^T & 1 \end{bmatrix} = \tilde{\boldsymbol{K}}\boldsymbol{E}$. Project a world point to image plane as $\tilde{\boldsymbol{x}}_s = \tilde{\boldsymbol{P}}\tilde{\boldsymbol{p}}_w$, then normalize by the third element.

6.3.2 Distortion & Optical Effects

Plumb Bob Distortion Model: From normalized image coordinates to distorted coordinates, where $r = \sqrt{x_n^2 + y_n^2}$ and x_d, y_d normalized by f_x, f_y:

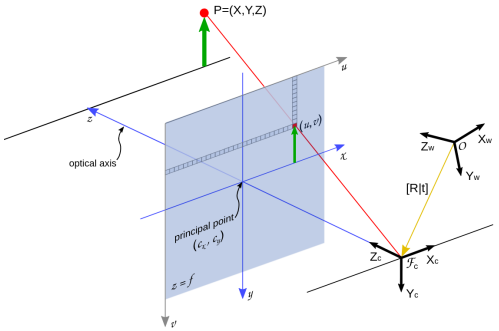
$$\begin{bmatrix} x_d \\ y_d \end{bmatrix} = (1 + \kappa_1 r^2 + \kappa_2 r^4 + \kappa_3 r^6)\begin{bmatrix} x_n \\ y_n \end{bmatrix} + \begin{bmatrix} 2\tau_1 x_n y_n + \tau_2(r^2 + 2x_n^2) \\ 2\tau_2 x_n y_n + \tau_1(r^2 + 2y_n^2) \end{bmatrix}$$

Note distortion is applied after extrinsics & projective map, but before doing intrinsics.

Radial Distortion: Barrel (corner points pushed in), pincushion (corner points pulled out), or mustache (combination).

Tangential Distortion: Pixels shifted in one direction (further pixels shifted more), due to the lens and imaging plane not being parallel.

Unwarping: Undoing distortion by precomputing distorted location of each pixel, then using bilinear interpolation.



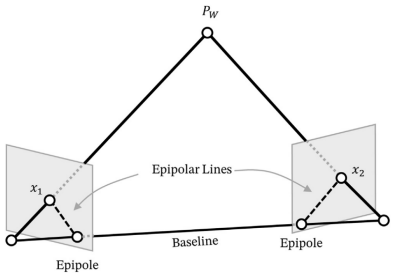
6.3.3 Image Operations & Transformations

Histogram Equalization: $T(r_k) = \sum_{j=0}^{k} p_r(r_j) = \sum_{j=0}^{k} \frac{n_j}{n}$ where $T(r_k)$ is the CDF and $p_r(r_j)$ is the PDF (normalized histogram) as a function of pixel intensity. Each pixel value is mapped to the proportion of pixels that it's brighter than, resulting in a flat-ish histogram.

Linear Filtering: $g(i,j) = f * h = \sum_{k,l} f(i-k, j-l)h(k,l)$ where $h(k,l)$ is the **kernel**. Note how the kernel is flipped. Separable kernels can be expressed as $\boldsymbol{K} = \boldsymbol{v}\boldsymbol{h}^T$ and is equivalent to convolution by $\boldsymbol{v}$ then $\boldsymbol{h}$.

Geometric Transform: Applies to domain, $g(\boldsymbol{x}) = f(\boldsymbol{h}(\boldsymbol{x}))$. Better to compute the inverse transformation and map each pixel in the output to a location in the input, and bilinearly interpolate.

6.3.4 Stereo Vision



Epipolar Geometry: Form the epipolar plane from a point and the cameras' optical centres; the intersection of the epipolar plane with each imaging plane forms an epipolar line, which is where correspondences lie.

Stereo Rectification: Transforming and unwarping the images so the imaging planes become parallel, at the same depth, and rotationally aligned (fronto-parallel or standard rectified geometry). After rectification all epipolar lines are horizontal.

Disparity: Difference in x values of a point between cameras. $Z = \frac{fb}{d}$ where b is baseline and d is disparity. Larger disparity indicates a point is closer.

Stereo Camera Model: $\begin{bmatrix} x_0 \\ y_0 \\ d \end{bmatrix} = \boldsymbol{f}(\boldsymbol{p}_c) = \frac{f}{Z}\begin{bmatrix} X \\ Y \\ b \end{bmatrix} + \begin{bmatrix} c_x \\ c_y \\ 0 \end{bmatrix}$

Baseline: Distance between cameras. Larger baseline improves accuracy, but makes matching more difficult as overlapping FOV decreases and foreshortening is increased (oblique objects appear as different sizes). Wider baseline also makes errors more Gaussian-like (shorter tail); for short baseline, using Gaussian errors biases the points to be closer.

Combined Stereo Model: Forward model produces pixel locations in both cameras. Frame origin is halfway between two cameras.

Forward: $\boldsymbol{f}(\boldsymbol{p}) = \boldsymbol{M}\frac{1}{z}\boldsymbol{p} = \begin{bmatrix} f_u & 0 & c_u & f_u\frac{b}{2} \\ 0 & f_v & c_v & 0 \\ f_u & 0 & c_u & -f_u\frac{b}{2} \\ 0 & f_v & c_v & 0 \end{bmatrix}\frac{1}{z}\boldsymbol{p} = \begin{bmatrix} u_l \\ v_l \\ u_r \\ v_r \end{bmatrix}$.

Inverse model: $\boldsymbol{g}(\boldsymbol{y}) = \frac{b}{u_l - u_r}\begin{bmatrix} \frac{1}{2}(u_l + u_r) - c_u \\ \frac{f_u}{f_v}\left(\frac{1}{2}(v_l + v_r) - c_v\right) \\ f_u \end{bmatrix}$.

Jacobians: forward $\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{p}} = \boldsymbol{M}\frac{1}{p_3}\begin{bmatrix} 1 & 0 & -p_1/p_3 & 0 \\ 0 & 1 & -p_2/p_3 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -p_4/p_3 & 1 \end{bmatrix}$, inverse $\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{y}} = \frac{b}{(u_l - u_r)^2}\begin{bmatrix} -u_r + c_u & 0 & u_l - c_u & 0 \\ -\frac{f_u}{f_v}\left(\frac{1}{2}(v_l + v_r) - c_v\right) & \frac{f_u}{2f_v} & \frac{f_u}{f_v}\left(\frac{1}{2}(v_l + v_r) - c_v\right) & \frac{f_u}{2f_v} \\ -f_u & 0 & f_u & 0 \end{bmatrix}$

Pixel-Disparity Stereo Model: Forward model produces a pixel location and disparity. Frame origin is at the left camera.

Forward: $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{p}) = \boldsymbol{M}\frac{1}{p_3}\boldsymbol{p} = \begin{bmatrix} f_u & 0 & c_u & 0 \\ 0 & f_v & c_v & 0 \\ 0 & 0 & 0 & f_u b \end{bmatrix}\frac{1}{p_3}\boldsymbol{p} = \begin{bmatrix} u_l \\ v_l \\ d \end{bmatrix}$.

Inverse model: $\boldsymbol{g}(\boldsymbol{y}) = \frac{b}{d}\begin{bmatrix} u_l - c_u \\ \frac{f_u}{f_v}(v_l - c_v) \\ f_u \end{bmatrix}$.

Forward Jacobian: $\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{p}} = \frac{1}{p_3}\boldsymbol{M}\begin{bmatrix} 1 & 0 & -p_1/p_3 & 0 \\ 0 & 1 & -p_2/p_3 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -p_4/p_3 & 1 \end{bmatrix}$

Inverse Jacobian: $\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{y}} = \frac{b}{d^2}\begin{bmatrix} d & 0 & c_u - u_l \\ 0 & \frac{f_u}{f_v}d & \frac{f_u}{f_v}(c_v - v_l) \\ 0 & 0 & -f_u \end{bmatrix}$

6.3.5 Image Features

Features should be salient (distinctive), local, repeatable (can be found in other images) and compact.

Weighted Summed Squared Difference: Images I_0, I_1, displacement $\boldsymbol{u}$

$$E_{WSSD}(\boldsymbol{u}) = \sum_i w(\boldsymbol{x}_i)(I_1(\boldsymbol{x}_i + \boldsymbol{u}) - I_0(\boldsymbol{x}_i))^2$$

For $I_0 = I_1$ this is **autocorrelation** for a small $\Delta\boldsymbol{u}$; high autocorrelation means the area is distinctive. $E_{AC}(\Delta\boldsymbol{u}) \approx \Delta\boldsymbol{u}^T\boldsymbol{A}\Delta\boldsymbol{u}$. As convolution, $\boldsymbol{A} = w * \begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix}$.

Harris Corner Detector: $\det\boldsymbol{A} - \alpha\,\mathrm{tr}\,\boldsymbol{A}^2 = \lambda_0\lambda_1 - \alpha(\lambda_0 + \lambda_1)^2$. Largest when both eigenvalues are large, indicating distinctiveness in both directions, i.e. a corner.

SIFT: Scale and rotation invariant, capable but slow. Construct scale space representation by repeated blurring and downsampling. Find keypoints with Difference of Gaussians (similar to LoG). Reject keypoints using Hessians similar to Harris. Assign each keypoint an orientation using gradients of points around the keypoint in a histogram. Assemble final descriptor using 4x4 grid of subpatches, each subpatch having a gradient orientation histogram.

SURF: Like SIFT but faster. Uses box filters instead of Gaussians (can use integral images). Smaller descriptor size (half as SIFT, 64-dimensional vector).

FAST: Very fast but not scale or rotation invariant. For each pixel look at the 16 pixels in a ring around it, if there are 12 contiguous pixels that are brighter or darker than the centre it is a feature. Use non-maximum suppression (suppress non-maximum features in an area around each maximum).

BRISK: Sample pairs of points in a circular pattern around the centre (Gaussian applied around each sampled point); compare the pairs and generate one bit in the feature for each pair.

BRIEF: Draw points randomly (one of several distributions) around the centre, compare them similarly to BRISK to form descriptor. Sampling pattern is decided once and used consistently across images. 512-bit features. Fast but not scale/rotation invariant and sensitive to noise.

ORB: Combines FAST and BRIEF to get a fast and reliable feature. Downsample similarly to SIFT and apply FAST at all scales to find keypoints. Compute orientation using intensity centroid. Use rotated BRIEF (rotate sampling pattern or image subpatch) to form descriptor.

Learning-Based Detectors: Often use classic features to generate training data for supervised methods, or image + pose sequences for unsupervised methods. LIFT is one such method using 3 CNNs to detect, estimate orientation, and generate descriptor, trained using contrastive learning.

7 Odometry

7.1 Dead Reckoning/Wheel Odometry

Key Characteristics: Mean is unbiased, variance is unbounded.

Derivation: Get $\dot{\boldsymbol{q}}(\dot{\boldsymbol{\varphi}})$ from kinematics (transform (v, u, ω) to inertial frame first), discretize to get $\boldsymbol{q}(t+h)$ and split $\boldsymbol{q}(t) = \bar{\boldsymbol{q}}(t) + \delta\boldsymbol{q}(t)$.

Linearize about $\bar{\boldsymbol{q}}(t)$ and drop product of Gaussians to get $\bar{\boldsymbol{q}}(t+h) + \delta\boldsymbol{q}(t+h)$. Take $\bar{\boldsymbol{q}}(t+h) = \mathbb{E}[\bar{\boldsymbol{q}}(t+h) + \delta\boldsymbol{q}(t+h)]$.

Subtract $\bar{\boldsymbol{q}}(t+h)$ to get linear update equation $\delta\boldsymbol{q}(t+h) = \boldsymbol{A}(t)\delta\boldsymbol{q}(t) + \boldsymbol{B}(t)\boldsymbol{w}(t)$. Take $\boldsymbol{\Sigma}(t+h) = \mathbb{E}[\delta\boldsymbol{q}(t)\delta\boldsymbol{q}(t)^T] = \boldsymbol{A}(t)\boldsymbol{\Sigma}(t)\boldsymbol{A}(t)^T + \boldsymbol{B}(t)\boldsymbol{Q}\boldsymbol{B}(t)^T$ where $\boldsymbol{Q}$ is covariance of $\boldsymbol{w}(t)$.

Differential Drive: Let $\boldsymbol{R} = \begin{bmatrix} r/2 & r/2 \\ r/(2b) & -r/(2b) \end{bmatrix}$

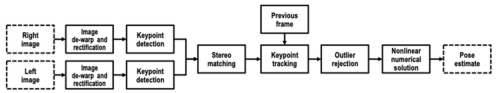
$$\bar{\boldsymbol{q}}(t+h) = \bar{\boldsymbol{q}}(t) + \begin{bmatrix} \cos\bar{\theta}(t) & 0 \\ \sin\bar{\theta}(t) & 0 \\ 0 & 1 \end{bmatrix}\boldsymbol{R}\begin{bmatrix} \Delta\varphi_r(t) \\ \Delta\varphi_l(t) \end{bmatrix}$$

$$\delta\boldsymbol{q}(t+h) = \delta\boldsymbol{q}(t) + \begin{bmatrix} -\sin\bar{\theta}(t) & 0 \\ \cos\bar{\theta}(t) & 0 \\ 0 & 0 \end{bmatrix}\boldsymbol{R}\begin{bmatrix} \Delta\varphi_r(t) \\ \Delta\varphi_l(t) \end{bmatrix}\delta\theta(t) + \begin{bmatrix} \cos\bar{\theta}(t) & 0 \\ \sin\bar{\theta}(t) & 0 \\ 0 & 1 \end{bmatrix}\boldsymbol{R}\begin{bmatrix} w_r(t) \\ w_l(t) \end{bmatrix}$$

Params: $r = 2\frac{x_{\text{true}}}{\sum_k(\Delta\varphi_r(kh) + \Delta\varphi_l(kh))}; b = \frac{r}{2}\frac{\sum_k(\Delta\varphi_r(kh) + \Delta\varphi_l(kh))}{2\pi N}$

7.2 Feature-Based Stereo Visual Odometry

Given images captured in frame a and frame b, find $\boldsymbol{T}_{ba}$. Use where high accuracy short-term localization is needed, e.g. indoors, Mars, or high wheel slip environments.



Point Cloud Alignment: Given $\boldsymbol{p}_a^j, \boldsymbol{p}_b^j$, minimize cost

$$E(\boldsymbol{C}_{ba}, \boldsymbol{r}_a) = \frac{1}{2}\sum_{j=1}^{J}\left(\boldsymbol{p}_b^j - \boldsymbol{C}_{ba}(\boldsymbol{p}_a^j - \boldsymbol{r}_a)\right)^T\boldsymbol{\Sigma}^j\left(\boldsymbol{p}_b^j - \boldsymbol{C}_{ba}(\boldsymbol{p}_a^j - \boldsymbol{r}_a)\right)$$

with matrix weights $\boldsymbol{\Sigma}^j = \left(\boldsymbol{G}_b^j\boldsymbol{R}_b^j\boldsymbol{G}_b^{jT} + \boldsymbol{C}_{ba}\boldsymbol{G}_a^j\boldsymbol{R}_a^j\boldsymbol{G}_a^{jT}\boldsymbol{C}_{ba}^T\right)^{-1}$ where $\boldsymbol{G}_b^j = \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{y}}\Big|_{\boldsymbol{f}(\boldsymbol{p}_b^j)}, \boldsymbol{G}_a^j = \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{y}}\Big|_{\boldsymbol{f}(\boldsymbol{p}_a^j)}$, and $\boldsymbol{R}_b^j, \boldsymbol{R}_a^j$ are pixel measurement covariances.

Closed-form solution for scalar weights:

1. Align centroids: $\boldsymbol{p}_a = \frac{\sum_{j=1}^J w^j\boldsymbol{p}_a^j}{\sum_{j=1}^J w^j}, \boldsymbol{p}_b = \frac{\sum_{j=1}^J w^j\boldsymbol{p}_b^j}{\sum_{j=1}^J w^j}$
2. Outer product: $\boldsymbol{W}_{ba} = \frac{1}{\sum_{j-1}^J w^j}\sum_{j=1}^{J} w^j(\boldsymbol{p}_b^j - \boldsymbol{p}_b)(\boldsymbol{p}_a^j - \boldsymbol{p}_a)^T$
3. Compute SVD $\boldsymbol{V}\boldsymbol{S}\boldsymbol{U}^T = \boldsymbol{W}_{ba}$, then compute rotation $\boldsymbol{C}_{ba} = \boldsymbol{V}\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \det(\boldsymbol{U})\det(\boldsymbol{V}) \end{bmatrix}\boldsymbol{U}^T$, and translation $\boldsymbol{r}_a = -\boldsymbol{C}_{ba}^T\boldsymbol{p}_b + \boldsymbol{p}_a$.

Solution is unique if rank $\boldsymbol{W}_{ba} \geq 2$. If rank $\boldsymbol{W}_{ba} = 2$ points are coplanar (unique if we enforce no reflection). If rank $\boldsymbol{W}_{ba} = 1$ points are colinear.

7.3 LiDAR Scan Matching

Given reference $M = \{\boldsymbol{p}_M^1, \ldots, \boldsymbol{p}_M^{N_M}\}$, data $D = \{\boldsymbol{p}_D^1, \ldots, \boldsymbol{p}_D^{N_D}\}$, where $N_M = N_D$ (i.e. one-to-one correspondence), find $\boldsymbol{T}_{MD} = \{\boldsymbol{C}_{MD}, \boldsymbol{t}_D^{MD}\}$.

ICP Algorithm: Until convergence, do:

1. Transform $\boldsymbol{p}_M^j = \boldsymbol{C}_k(\boldsymbol{p}_D^j - \boldsymbol{t}_k)$ to form $\tilde{D}$
2. Find correspondences $\{I, J\} = \mathrm{NN}(M, \tilde{D})$
3. Error: $\mathcal{J} = \frac{1}{2}\sum_{j=1}^J w^j\left(\boldsymbol{p}_M^i - \boldsymbol{C}_k\left(\boldsymbol{p}_D^j - \boldsymbol{t}_k\right)\right)^T\left(\boldsymbol{p}_M^i - \boldsymbol{C}_k\left(\boldsymbol{p}_D^j - \boldsymbol{t}_k\right)\right)$
4. Gradient descent: $\boldsymbol{T}_{k+1} = \boldsymbol{T}_k + \alpha\nabla\mathcal{J}$

Subsampling: *Uniform* (voxel-based): take one point per voxel, preserves structure; *random*: preserves density, amy lose structure; *feature-based*: identifies structural features to preserve environment traits; *normal-space*: sample a subset with surface normals uniformly distributed, good for smooth areas with sparse features.

Surface Normal Estimation: For a neighbourhood of local points, try to fit a plane $a^Tp = c$ by minimizing $\frac{1}{k}\sum_{i=1}^k(\boldsymbol{a}^T\boldsymbol{p}_i - c)^2$; equivalent to PCA of $M = \frac{1}{k}\sum_{i=1}^k(\boldsymbol{p}_i - \bar{\boldsymbol{p}})(\boldsymbol{p}_i - \bar{\boldsymbol{p}})^T$, where the eigenvector of the smallest eigenvalue (least variance) is the normal.

Data Association: *Closest compatible point*: check normals/colours/etc. within range; *normal shooting*: first point along the surface normal direction; *outlier rejection* (by checking distances of neighbours).

Point-to-Plane Matching: Only penalize errors in the normal direction;

$$\mathcal{J} = \frac{1}{2}\sum_{j=1}^{J} w^j\left(\boldsymbol{p}_M^i - \boldsymbol{C}_k\left(\boldsymbol{p}_D^j - \boldsymbol{t}_k\right)\right)^T\boldsymbol{n}^j\boldsymbol{n}^{jT}\left(\boldsymbol{p}_M^i - \boldsymbol{C}_k\left(\boldsymbol{p}_D^j - \boldsymbol{t}_k\right)\right)$$

Plane-to-Plane Matching: Uses normals in both sets;

$$\mathcal{J} = \frac{1}{2}\sum_{j=1}^{J}\left(\boldsymbol{p}_M^i - \boldsymbol{C}_k(\boldsymbol{p}_D^j - \boldsymbol{t}_k)\right)^T(\boldsymbol{\Sigma}_i^M + \boldsymbol{C}_k\boldsymbol{\Sigma}_j^D\boldsymbol{C}_k^T)^{-1}\left(\boldsymbol{p}_M^i - \boldsymbol{C}_k(\boldsymbol{p}_D^j - \boldsymbol{t}_k)\right)$$

$\boldsymbol{\Sigma}_i^M = \boldsymbol{C}(\boldsymbol{n}_i^M)\boldsymbol{\Sigma}_\varepsilon\boldsymbol{C}(\boldsymbol{n}_i^M)^T$, $\boldsymbol{\Sigma}_j^D = \boldsymbol{C}(\boldsymbol{n}_j^D)\boldsymbol{\Sigma}_\varepsilon\boldsymbol{C}(\boldsymbol{n}_j^D)^T$, $\boldsymbol{\Sigma}_\varepsilon = \mathrm{diag}(\varepsilon, 1, 1)$.

Normal Distribution Transform (NDT): Divide into cells and model each as $\rho_{c_i}(\boldsymbol{p}) = \exp\left(-\frac{1}{2}(\boldsymbol{p} - \boldsymbol{\mu}_i)^T\boldsymbol{\Sigma}_i^{-1}(\boldsymbol{p} - \boldsymbol{\mu}_i)\right)$; score each point in D based on the Gaussian in its cell: $\Lambda(\boldsymbol{T}) = -\sum_{\boldsymbol{p}_D^j \in D}\rho_c(\boldsymbol{T}(\boldsymbol{p}_D^j))$.

Practical Considerations: Motion compensation; anisotropic (e.g. distance-based) filtering to remove low-information points; coarse-to-fine alignment (starting with smaller subsample); quality estimation via final association covariance, overlap between scans, etc.

8 Localization and Mapping

Map Types by Application: *Planning*: tracks occupancy, safe/unsafe places; *application*: tracks spatially distributed properties, e.g. topography; *navigation*: tracks localization features, may be sparse.

Map Types by Information: *Metric*: retains scale, metric consistency; *topometric*: locally metric, globally topographic; *topological*: no distances, just interconnectedness.

8.1 Occupancy Grid Mapping

Assumptions: Cell occupancy independent from other cells (no space connectivity) or robot movement, measurements independent given robot states and map.

Probabilistic Update: $p(c_k|y^t,x^t)=\frac{p(y_t|c_k,x_t)}{p(y_t)}p(c_k|y^{t-1},x^{t-1})$

Log-Odds: $l(x)=\ln\left(\frac{p(x)}{1-p(x)}\right)\iff p(x)=\frac{\exp(l(x))}{1+\exp(l(x))}$

Log-Odds Update: $l(c_k|y^t,x^t)=l(c_k|y^{t-1},x^{t-1})+\ln\left(\frac{p(y_t|c_k,x_t)}{1-p(y_t|c_k,x_t)}\right)$ where the last term is α if c_k near measurement, $-\beta$ if c_k closer than measurement, 0 otherwise.

8.2 Terrain Assessment

Determine which parts of terrain are safe to traverse and which parts are not by fitting planes to local sections. Use RANSAC to reject outliers. Use info such as plane height and slope to determine traversibility.

Regular Grid: Patch j: let $\boldsymbol{A}_j=\begin{bmatrix}1 & x_{1j} & y_{1j}\\ \vdots & \vdots & \vdots\\ 1 & x_{n_jj} & y_{n_jj}\end{bmatrix},\boldsymbol{x}_j=\begin{bmatrix}a_j\\ b_j\\ c_j\end{bmatrix},\boldsymbol{b}_j=\begin{bmatrix}z_{1j}\\ \vdots\\ z_{n_jj}\end{bmatrix}$ minimize error $\boldsymbol{e}_j=\boldsymbol{A}_j\boldsymbol{x}_j-\boldsymbol{b}_j$, with solution $\boldsymbol{x}_j=(\boldsymbol{A}_j^T\boldsymbol{A}_j)^{-1}\boldsymbol{A}_j^T\boldsymbol{b}_j$.

Recursive Formulation: Let $\boldsymbol{p}_{ij}=\begin{bmatrix}1 & x_{ij} & y_{ij} & z_{ij}\end{bmatrix}^T$ and define $\boldsymbol{P}_j=\sum_i\boldsymbol{p}_{ij}\boldsymbol{p}_{ij}^T=\begin{bmatrix}\boldsymbol{A}_j^T\boldsymbol{A}_j & \boldsymbol{A}_j^T\boldsymbol{b}_j\\ (\boldsymbol{A}_j^T\boldsymbol{b}_j)^T & \boldsymbol{b}_j^T\boldsymbol{b}_j\end{bmatrix}=\begin{bmatrix}p_{11} & p_{12} & p_{13} & p_{14}\\ p_{21} & p_{22} & p_{23} & p_{24}\\ p_{31} & p_{32} & p_{33} & p_{34}\\ p_{41} & p_{42} & p_{43} & p_{44}\end{bmatrix}=\begin{bmatrix}\boldsymbol{Q}^{-1} & \boldsymbol{r}\\ \boldsymbol{r}^T & p_{44}\end{bmatrix}$

- Number of points: $n_j=p_{11}$
- Plane fit: $\begin{bmatrix}a_j & b_j & c_j\end{bmatrix}^T=\boldsymbol{Q}\boldsymbol{r}$
- Residuals: $e_j=p_{44}-\boldsymbol{r}^T\boldsymbol{Q}\boldsymbol{r}$
- Average height: $h_j=\frac{p_{14}}{n_j}$
- Max slope: $\alpha_j=\cos^{-1}\frac{1}{b_j^2+c_j^2+1}$

8.3 Filtering-Based Localization

Markov Assumption: The conditional probabilities of future states are independent of any past states, given the current state; i.e. the state contains enough information to predict the future.

Bayes Filter: $p(\boldsymbol{x}_k|\hat{\boldsymbol{x}}_0,\boldsymbol{v}_{1:k},\boldsymbol{y}_{0:k})=$
$\eta p(\boldsymbol{y}_k|\boldsymbol{x}_k)\int p(\boldsymbol{x}_k|\boldsymbol{x}_{k-1},\boldsymbol{v}_k)p(\boldsymbol{x}_{k-1}|\hat{\boldsymbol{x}}_0,\boldsymbol{v}_{1:k-1},\boldsymbol{y}_{0:k-1})\,\mathrm{d}\boldsymbol{x}_{k-1}$

Particle Filtering: Representing the robot state PDF with particles, where each particle is a hypothesis:

1. Prediction: $\begin{bmatrix}\hat{\boldsymbol{x}}_{k-1,m}\\ \boldsymbol{w}_{k,m}\end{bmatrix}\sim p(\boldsymbol{x}_{k-1}|\hat{\boldsymbol{x}}_0,\boldsymbol{v}_{1:k-1},\boldsymbol{y}_{0:k-1})p(\boldsymbol{w}_k)$ from prior and motion noise; generate predictions $\check{\boldsymbol{x}}_{k,m}=\boldsymbol{f}(\hat{\boldsymbol{x}}_{k-1,m},\boldsymbol{v}_k,\boldsymbol{w}_{k,m})$.
2. Correction: Assign weights $w_{k,m}=\eta p(\boldsymbol{y}_k|\check{\boldsymbol{x}}_{k,m})$ by using expected measurements $\check{\boldsymbol{y}}_{k,m}=\boldsymbol{g}(\check{\boldsymbol{x}}_{k,m},\boldsymbol{0})$ and assuming $p(\boldsymbol{y}_k|\check{\boldsymbol{x}}_{k,m})=p(\boldsymbol{y}_k|\check{\boldsymbol{y}}_{k,m})$; resample based on weights to get new particles $\hat{\boldsymbol{x}}_{k,m}$.

Practical considerations: Maintaining sample variance by adding more noise, adding samples drawn uniformly from sample space, or resampling less often; adapting particle count online using weight sum heuristic.

Madow Systematic Sampling: Create M bins where bin m has size $w_{k,m}$; starting from random, take steps of $1/M$ and sample from each visited bin. Helps preserve particle diversity.

Kalman Filter: Linear model $\boldsymbol{x}_k=\boldsymbol{A}_{k-1}\boldsymbol{x}_{k-1}+\boldsymbol{v}_k+\boldsymbol{w}_k$, $\boldsymbol{y}_k=\boldsymbol{C}_k\boldsymbol{x}_k+\boldsymbol{n}_k$:

1. Prediction: $\begin{cases}\check{\boldsymbol{P}}_k=\boldsymbol{A}_{k-1}\hat{\boldsymbol{P}}_{k-1}\boldsymbol{A}_{k-1}^T+\boldsymbol{Q}_k\\ \check{\boldsymbol{x}}_k=\boldsymbol{A}_{k-1}\hat{\boldsymbol{x}}_{k-1}+\boldsymbol{v}_k\end{cases}$
2. Kalman gain: $\boldsymbol{K}_k=\check{\boldsymbol{P}}_k\boldsymbol{C}_k^T(\boldsymbol{C}_k\check{\boldsymbol{P}}_k\boldsymbol{C}_k^T+\boldsymbol{R}_k)^{-1}$
3. Correction: $\begin{cases}\hat{\boldsymbol{P}}_k=(\boldsymbol{1}-\boldsymbol{K}_k\boldsymbol{C}_k)\check{\boldsymbol{P}}_k\\ \hat{\boldsymbol{x}}_k=\check{\boldsymbol{x}}_k+\boldsymbol{K}_k(\boldsymbol{y}_k-\boldsymbol{C}_k\check{\boldsymbol{x}}_k)\end{cases}$

With correct modeling KF is *consistent* and *unbiased* (correct mean and covariance).

Extended Kalman Filter: Linearize nonlinear models:

$$\boldsymbol{x}_k=\boldsymbol{f}(\boldsymbol{x}_{k-1},\boldsymbol{v}_k,\boldsymbol{w}_k)\approx\check{\boldsymbol{x}}_k+\boldsymbol{F}_{k-1}(\boldsymbol{x}_{k-1}-\hat{\boldsymbol{x}}_{k-1})+\boldsymbol{w}'_k$$
$$\boldsymbol{y}_k=\boldsymbol{g}(\boldsymbol{x}_k,\boldsymbol{n}_k)\approx\check{\boldsymbol{y}}_k+\boldsymbol{G}_k(\boldsymbol{x}_k-\check{\boldsymbol{x}}_k)+\boldsymbol{n}'_k$$

where $\boldsymbol{F}_{k-1}=\frac{\partial\boldsymbol{f}}{\partial\boldsymbol{x}_{k-1}}$, $\boldsymbol{G}_k=\frac{\partial\boldsymbol{g}}{\partial\boldsymbol{x}_k}$, $\boldsymbol{w}'_k=\frac{\partial\boldsymbol{f}}{\partial\boldsymbol{w}_k}\boldsymbol{w}_k$, $\boldsymbol{n}'_k=\frac{\partial\boldsymbol{g}}{\partial\boldsymbol{n}_k}\boldsymbol{n}_k$, $\boldsymbol{f}$ Jacobians evaluated at $(\hat{\boldsymbol{x}}_{k-1},\boldsymbol{v}_k,\boldsymbol{0})$ and $\boldsymbol{g}$ Jacobians at $(\check{\boldsymbol{x}}_k,\boldsymbol{0})$.

1. Prediction: $\begin{cases}\check{\boldsymbol{P}}_k=\boldsymbol{F}_{k-1}\hat{\boldsymbol{P}}_k\boldsymbol{F}_{k-1}^T+\boldsymbol{Q}'_k\\ \check{\boldsymbol{x}}_k=\boldsymbol{f}(\hat{\boldsymbol{x}}_{k-1},\boldsymbol{v}_k,\boldsymbol{0})\end{cases}$
2. Kalman gain: $\boldsymbol{K}_k=\check{\boldsymbol{P}}_k\boldsymbol{G}_k^T(\boldsymbol{G}_k\check{\boldsymbol{P}}_k\boldsymbol{G}_k^T+\boldsymbol{R}'_k)^{-1}$
3. Correction: $\begin{cases}\hat{\boldsymbol{P}}_k=(\boldsymbol{1}-\boldsymbol{K}_k\boldsymbol{G}_k)\check{\boldsymbol{P}}_k\\ \hat{\boldsymbol{x}}_k=\check{\boldsymbol{x}}_k+\boldsymbol{K}_k(\boldsymbol{y}_k-\boldsymbol{g}(\check{\boldsymbol{x}}_k,\boldsymbol{0}))\end{cases}$

where $\boldsymbol{Q}'_k=\frac{\partial\boldsymbol{f}}{\partial\boldsymbol{w}_k}\boldsymbol{Q}_k\left(\frac{\partial\boldsymbol{f}}{\partial\boldsymbol{w}_k}\right)^T$, $\boldsymbol{R}'_k=\frac{\partial\boldsymbol{g}}{\partial\boldsymbol{n}_k}\boldsymbol{R}_k\left(\frac{\partial\boldsymbol{g}}{\partial\boldsymbol{n}_k}\right)^T$. EKF can be biased and inconsistent for severe nonlinearity since it linearizes about the estimate.

Sigma Point Transform: For L-dimensional state and parameter κ:

1. Compute $2L+1$ sigma points, where $\boldsymbol{x}_0=\boldsymbol{\mu}_x$, $\boldsymbol{x}_i=\boldsymbol{\mu}_x+\sqrt{L+\kappa}\,\mathrm{col}_i\,\boldsymbol{L}$, and $\boldsymbol{x}_{i+L}=\boldsymbol{\mu}_x-\sqrt{L+\kappa}\,\mathrm{col}_i\,\boldsymbol{L}$, where $\boldsymbol{L}\boldsymbol{L}^T=\boldsymbol{\Sigma}_x$.
2. Pass each through the nonlinearity: $\boldsymbol{y}_i=\boldsymbol{g}(\boldsymbol{x}_i)$.
3. Output Gaussian: $\boldsymbol{\mu}_y=\sum_{i=0}^{2L}\alpha_i\boldsymbol{y}_i$, $\boldsymbol{\Sigma}_y=\sum_{i=0}^{2L}\alpha_i(\boldsymbol{y}_i-\boldsymbol{\mu}_y)(\boldsymbol{y}_i-\boldsymbol{\mu}_y)^T$, with weights $\alpha_0=\frac{\kappa}{L+\kappa}$, $\alpha_i=\frac{1}{2}\frac{1}{L+\kappa}$.

Unscented Kalman Filter:

1. State Prediction:
   (a) Extract sigma points $\boldsymbol{z}_i$ from $\boldsymbol{\mu}_z=\begin{bmatrix}\hat{\boldsymbol{x}}_{k-1}\\ \boldsymbol{0}\end{bmatrix},\boldsymbol{\Sigma}_{zz}=\begin{bmatrix}\hat{\boldsymbol{P}}_{k-1} & \boldsymbol{0}\\ \boldsymbol{0} & \boldsymbol{Q}_k\end{bmatrix}$.
   (b) Nonlinear motion model: $\boldsymbol{z}_i=\begin{bmatrix}\check{\boldsymbol{x}}_{k-1,i}\\ \boldsymbol{w}_{k,i}\end{bmatrix}$, $\check{\boldsymbol{x}}_{k,i}=\boldsymbol{f}(\hat{\boldsymbol{x}}_{k-1,i},\boldsymbol{v}_k,\boldsymbol{w}_{k,i})$.
   (c) Recombine sigma points:
   $\check{\boldsymbol{x}}_k=\sum_{i=0}^{2L}\alpha_i\check{\boldsymbol{x}}_{k,i}$, $\check{\boldsymbol{P}}_k=\sum_{i=0}^{2L}\alpha_i(\check{\boldsymbol{x}}_{k,i}-\check{\boldsymbol{x}}_k)(\check{\boldsymbol{x}}_{k,i}-\check{\boldsymbol{x}}_k)^T$.
2. Measurement Prediction:
   (a) Extract sigma points $\boldsymbol{z}_i$ from $\boldsymbol{\mu}_z=\begin{bmatrix}\check{\boldsymbol{x}}_k\\ \boldsymbol{0}\end{bmatrix},\boldsymbol{\Sigma}_{zz}=\begin{bmatrix}\check{\boldsymbol{P}}_k & \boldsymbol{0}\\ \boldsymbol{0} & \boldsymbol{R}_k\end{bmatrix}$.
   (b) Nonlinear measurement model: $\boldsymbol{z}_i=\begin{bmatrix}\check{\boldsymbol{x}}_{k,i}\\ \boldsymbol{n}_{k,i}\end{bmatrix}$, $\check{\boldsymbol{y}}_{k,i}=\boldsymbol{g}(\check{\boldsymbol{x}}_{k,i},\boldsymbol{n}_{k,i})$.
   (c) Recombine sigma points: $\boldsymbol{\mu}_{y,k}=\sum_{i=0}^{2L}\alpha_i\check{\boldsymbol{y}}_{k,i}$,
   $\boldsymbol{\Sigma}_{yy,k}=\sum_{i=0}^{2L}\alpha_i(\check{\boldsymbol{y}}_{k,i}-\boldsymbol{\mu}_{y,k})(\check{\boldsymbol{y}}_{k,i}-\boldsymbol{\mu}_{y,k})^T$,
   $\boldsymbol{\Sigma}_{xy,k}=\sum_{i=0}^{2L}\alpha_i(\check{\boldsymbol{x}}_{k,i}-\check{\boldsymbol{x}}_k)(\check{\boldsymbol{y}}_{k,i}-\boldsymbol{\mu}_{y,k})^T$.
3. Update: $\boldsymbol{K}_k=\boldsymbol{\Sigma}_{xy,k}\boldsymbol{\Sigma}_{yy,k}^{-1}$, $\begin{cases}\hat{\boldsymbol{P}}_k=\check{\boldsymbol{P}}_k-\boldsymbol{K}_k\boldsymbol{\Sigma}_{xy,k}^T\\ \hat{\boldsymbol{x}}_k=\check{\boldsymbol{x}}_k+\boldsymbol{K}_k(\boldsymbol{y}_k-\boldsymbol{\mu}_{y,k})\end{cases}$.

UKF deals with nonlinearities better since the approximation is third-order and does not need analytical Jacobians.

9 Simultaneous Localization and Mapping

Given frame-to-frame odometric measurements and measurements of landmarks in robot frames, localize the robot and compute a consistent map of landmarks in the inertial frame.

9.1 Filtering-Based SLAM

Filtering-based approaches marginalize out past states, leading to difficulties with large loop closures.

EKF-SLAM: Form augmented state $\boldsymbol{x}_k$ by stacking robot states and landmark states. Landmarks have identity motion model with no process noise. Use EKF to estimate robot and landmark positions simultaneously. Scales poorly with number of landmarks. Can use sparse extended information filter formulation, which sparsifies $\boldsymbol{P}^{-1}$ and uses sparsity to quickly invert.

Information Filter: Alternative KF formulation using $\boldsymbol{P}^{-1}$ and $\boldsymbol{P}^{-1}\boldsymbol{x}$:

1. Prediction: $\begin{cases}\check{\boldsymbol{x}}_k=\boldsymbol{A}_{k-1}\hat{\boldsymbol{x}}_{k-1}+\boldsymbol{v}_k\\ \check{\boldsymbol{P}}_k=\boldsymbol{Q}_k+\boldsymbol{A}_{k-1}\hat{\boldsymbol{P}}_{k-1}\boldsymbol{A}_{k-1}^T\end{cases}$
2. Correction: $\begin{cases}\hat{\boldsymbol{P}}_k^{-1}=\check{\boldsymbol{P}}_k^{-1}+\boldsymbol{C}_k^T\boldsymbol{R}_k^{-1}\boldsymbol{C}_k\\ \hat{\boldsymbol{P}}_k^{-1}\hat{\boldsymbol{x}}_k=\check{\boldsymbol{P}}_k^{-1}\check{\boldsymbol{x}}_k+\boldsymbol{C}_k^T\boldsymbol{R}_k^{-1}\boldsymbol{y}_k\end{cases}$

Rao-Blackwellized Particle Filter/FastSLAM: Factor the belief as $p(\boldsymbol{x}_{1:K},\boldsymbol{m}|\boldsymbol{y}_{1:K},\boldsymbol{u}_{1:K})=p(\boldsymbol{m}|\boldsymbol{x}_{1:K},\boldsymbol{y}_{1:K})p(\boldsymbol{x}_{1:K}|\boldsymbol{y}_{1:K},\boldsymbol{u}_{1:K})$, represent the second piece (robot states) with particle filter, marginalize out first piece (e.g. Gaussians, occupancy grid). Each particle has its own map; weights are calculated based on measurements and each particle's map. On loop closure the particles with the most consistent map (best explaining measurements) will remain while others drop out.

9.2 Batch SLAM

All poses are kept and globally optimized along with landmark positions, similar to bundle adjustment. Consider $\boldsymbol{x}_k=\boldsymbol{f}(\boldsymbol{x}_{k-1},\boldsymbol{u}_k,\boldsymbol{w}_k)$ and $\boldsymbol{y}_{kl}=\boldsymbol{g}(\boldsymbol{x}_k,\boldsymbol{m}_l,\boldsymbol{n}_k)$, define errors $\boldsymbol{e}_k=\boldsymbol{x}_k-\boldsymbol{f}(\boldsymbol{x}_{k-1},\boldsymbol{u}_k,\boldsymbol{0})\sim\mathcal{N}(\boldsymbol{0},\boldsymbol{Q})$ and $\boldsymbol{e}_{kl}=\boldsymbol{y}_{kl}-\boldsymbol{g}(\boldsymbol{x}_k,\boldsymbol{m}_l,\boldsymbol{0})\sim\mathcal{N}(\boldsymbol{0},\boldsymbol{R})$. Minimize error:

$$\begin{aligned}J&=\frac{1}{2}\sum_k\boldsymbol{e}_k^T\boldsymbol{Q}^{-1}\boldsymbol{e}_k+\frac{1}{2}\sum_{k,l}\boldsymbol{e}_{kl}^T\boldsymbol{R}^{-1}\boldsymbol{e}_{kl}\\&=\frac{1}{2}\sum_k(\bar{\boldsymbol{e}}_k+\boldsymbol{\delta}\boldsymbol{x}_k-\boldsymbol{F}_{k-1}\boldsymbol{\delta}\boldsymbol{x}_{k-1})^T\boldsymbol{Q}^{-1}(\bar{\boldsymbol{e}}_k+\boldsymbol{\delta}\boldsymbol{x}_k-\boldsymbol{F}_{k-1}\boldsymbol{\delta}\boldsymbol{x}_{k-1})\\&\quad+\frac{1}{2}\sum_{k,l}(\bar{\boldsymbol{e}}_{kl}-\boldsymbol{G}_{x,kl}\boldsymbol{\delta}\boldsymbol{x}_k-\boldsymbol{G}_{m,kl}\boldsymbol{\delta}\boldsymbol{m}_l)^T\boldsymbol{R}^{-1}(\bar{\boldsymbol{e}}_{kl}-\boldsymbol{G}_{x,kl}\boldsymbol{\delta}\boldsymbol{x}_k-\boldsymbol{G}_{m,kl}\boldsymbol{\delta}\boldsymbol{m}_l)\\&=\frac{1}{2}(\bar{\boldsymbol{e}}+\boldsymbol{E}\boldsymbol{\delta}\boldsymbol{z})^T\boldsymbol{T}^{-1}(\bar{\boldsymbol{e}}+\boldsymbol{E}\boldsymbol{\delta}\boldsymbol{z})\end{aligned}$$

where $\boldsymbol{F}=\frac{\partial\boldsymbol{f}}{\partial\boldsymbol{x}},\boldsymbol{G}_x=\frac{\partial\boldsymbol{g}}{\partial\boldsymbol{x}},\boldsymbol{G}_m=\frac{\partial\boldsymbol{g}}{\partial\boldsymbol{m}}$, $\boldsymbol{T}$ has $\boldsymbol{Q},\boldsymbol{R}$ repeated along the diagonal, $\boldsymbol{\delta}\boldsymbol{z}$ collects perturbations to robot states and landmark positions, and $\bar{\boldsymbol{e}}$ collects all odometry and measurement errors. Solve for optimal update $\boldsymbol{E}^T\boldsymbol{T}^{-1}\boldsymbol{E}\boldsymbol{\delta}\boldsymbol{z}^*=-\boldsymbol{E}^T\boldsymbol{T}^{-1}\bar{\boldsymbol{e}}\iff\boldsymbol{A}\boldsymbol{\delta}\boldsymbol{z}^*=\boldsymbol{b}$ and iterate until convergence. $\boldsymbol{E}^T\boldsymbol{T}^{-1}\boldsymbol{E}$ has a sparse arrowhead structure for fast inversion.

Levenberg-Marquardt/Damped Least Squares: Solve $(\boldsymbol{E}^T\boldsymbol{T}^{-1}\boldsymbol{E}+\mu\,\mathrm{diag}(\boldsymbol{E}^T\boldsymbol{T}^{-1}\boldsymbol{E}\boldsymbol{\delta}))\boldsymbol{z}^*=-\boldsymbol{E}^T\boldsymbol{T}^{-1}\bar{\boldsymbol{e}}$ where μ is tunable and $\mathrm{diag}(\boldsymbol{A})$ is a matrix with the diagonal elements of $\boldsymbol{A}$. Helps when $\boldsymbol{A}$ is noninvertible and prevents large jumps for highly nonlinear functions.

Line Search: Search in the direction of $\boldsymbol{\delta}\boldsymbol{z}^*$; do $\bar{\boldsymbol{z}}\leftarrow\bar{\boldsymbol{z}}+\alpha\boldsymbol{\delta}\boldsymbol{z}$ where we pick the largest $\alpha\in[0,1]$ that decreases J.

Sliding Window Filter: Marginalize out old poses and invisible landmarks beyond the sliding window, but keeps poses within the window; similar to a hybrid of batch and filtering-based SLAM, but still has issues with large loop closures.

Adaptive Relative Bundle Adjustment: Represent landmark poses relative to observation pose, and robot poses relative to the last pose to avoid a privileged global frame. Optimize only landmarks and poses within the active window. Active window size adapts to incorporate more poses and landmarks on loop closure until error is small enough.

9.3 Visual SLAM

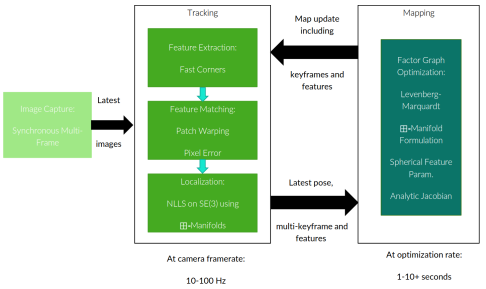
9.3.1 Feature-Based SLAM

Usually consists of a frontend for feature extraction and association (including loop closure), and a backend for map optimization.

MonoSLAM: First realtime monocular SLAM, using Shi-Tomasi features and constant velocity models. Builds probabilistic feature map through per-frame bundle adjustment.

Parallel Tracking and Mapping (PTAM): Separate threads for tracking (against existing point cloud) and mapping (adding to point cloud) into parallel threads. Coarse-to-fine feature matching and local window bundle adjustment for tracking.

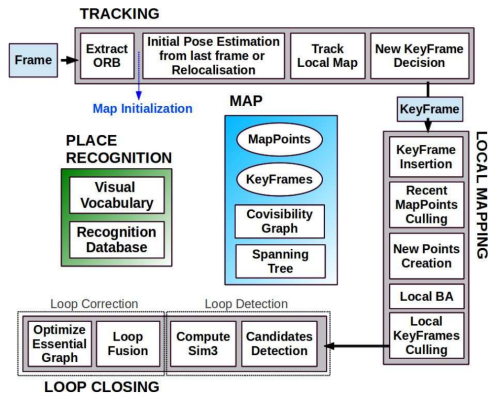
Multi-Camera PTAM (MCPTAM): Extension of PTAM to multiple cameras with non-overlapping FOVs. Spherical coordinate feature parametrization to limit depth uncertainty to a single axis. Use inverse depth to improve numerical optimization. Keyframe selection by estimating the amount of covariance reduction in features.



ORB-SLAM: Improving PTAM with loop closure ORB features. Scale-drift aware loop closure optimizes over and allows variation of the scale parameter when matching features. Covisbility graph uses viewpoints where features were observed to filter out implausible features when attempting matching. Bag of Binary Words extracts BoW descriptor from each frame for relocalization and loop closures. Parallel thread architecture with tracking, local bundle adjustment, and global loop closure/pose graph optimization, and full bundle adjustment after loop closure. Grid-based ORB feature extraction to spread out features. Aggressive adding of keyframes, cull when 90% of points are visible in 3 other frames.

ORB-SLAM2: Extension of ORB-SLAM for RGB-D input, semi-dense and dense mapping.

ORB-SLAM3: Map atlases for large scale mapping.



9.3.2 Direct SLAM

Match images and produce a pose between frames based on image intensities only, without extracting features, and using (almost) every pixel to construct the map, i.e. solving pixel correspondence and camera pose estimation at once, instead of decoupling them like in sparse VO.

Comparison to Feature-Based SLAM:

1. More robust to common failure modes due to inclusion of global information, e.g. blur, self-similar textures, defocus
2. Uses almost all information in the image (as opposed to only pixels around features)
3. Dense maps enable scene understanding and path planning
4. Implicit data association by matching pixel intensities
5. Very high dimensionality (10^5 to 10^6 pixels per image), computationally expensive
6. Contains a lot more local minima (due to implicit data association), sensitive to initialization (works only for small camera movements); IMU data needed for initialization
7. No modelling of geometric noise (e.g. rolling shutter)

Image Warping: Given the location of a pixel in frame $k-1$ and pose transformation $\boldsymbol{T}_{k,k-1}$, $\begin{bmatrix}u_k\\ v_k\\ d_k\end{bmatrix}=\boldsymbol{f}\left(\boldsymbol{T}_{k,k-1}\boldsymbol{g}\left(\begin{bmatrix}u_{k-1}\\ v_{k-1}\\ d_{k-1}\end{bmatrix}\right)\right)$.

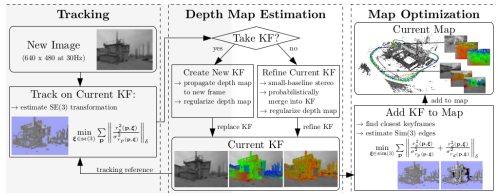
Photometric Error: $e_i=I_{k-1}(\boldsymbol{u}_{k-1}^i)-I'_k(\boldsymbol{u}_k^i)$ where I'_k is interpolated from I_k, indexed with continuous coordinates $\boldsymbol{u}_k^i$, and $\boldsymbol{y}_k^i=\begin{bmatrix}\boldsymbol{u}_k^i\\ d_k^i\end{bmatrix}=\boldsymbol{f}(\boldsymbol{T}_{k,k-1}\boldsymbol{g}(\boldsymbol{y}_{k-1}^i))$ from image warping. Solve for optimal pose transform $\hat{\boldsymbol{T}}_{k,k-1}=\arg\min_{\boldsymbol{T}_{k,k-1}}\sum_{i=1}^N\left(\frac{1}{\sigma}e_i\right)^2$.

Coarse-to-Fine Optimization: Starting with a blurred and subsampled image, then repeatedly optimizing while increasing resolution, using the previous optimization result as the initial guess for the next iteration. First iteration can use IMU/odometry. This smoothes out local minima and increases region of convergence.

Selective Pixel Matching: Performing matching/optimization only over "informative" pixels, based on gradient magnitude. Divide image into a grid, and for each cell take only the pixel with the largest gradient magnitude for optimization.

Photometric Calibration: Dense mapping is affected by gamma, exposure, vignetting, etc., so photometric calibration can be used to build a model correcting these effects.

LSD-SLAM: Seminal work on direct SLAM.



Direct Sparse Odometry (DSO): Integrates photometric calibration and optimizes over a sliding window of 7 keyframes, fixed number of 2000 active points. Region-adaptive gradient threshold for pixel selection. Try to match pixel along epipolar line based on motion estimate.

9.4 LiDAR SLAM

Lidar Odometry and Mapping (LOAM): Uses a keypoint extractor to look for planes and edges, while avoiding oblique planes, and edges caused by occlusion; matches only feature points. High-frequency thread performs odometry and does motion compensation, low frequency thread builds map with fine alignment.

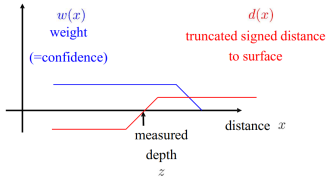
Visual LOAM (VLOAM): Fuses IMU, camera, and LiDAR; high frequency pose propagation using IMU, medium frequency feature-based visual inertial odometry, then low-frequency corrections from LOAM.

LIO-SAM: Tightly coupled LiDAR-inertial odometry and SLAM, fusing LiDAR along with IMU, GPS, magnetometer, with a sliding window factor graph for motion correction.

LiDAR-Camera-Inertial SLAM Approaches: *LiDAR-enhanced visual SLAM*: using LiDAR to initialize depths for bundle adjustment in visual SLAM; *vision-enhanced LiDAR SLAM*: using visual SLAM output poses to initialize scan registration and pose graph optimization; *tightly coupled*: simultaneous optimization of LiDAR and camera measurements (BA or BA+PG).

9.5 Dense RGB-D SLAM

Signed Distance Fields: Represent surfaces implicitly by storing the signed distance to the nearest surface in each cell in a voxel grid Max distance often truncated (capped). Recover surfaces from zero crossings using marching cubes. Noise naturally cancels over time when fusing observations, and surfaces can be represented at sub-voxel accuracy.



Dense Tracking and Mapping (DTAM): TSDF fusion from monocular camera. Builds highly detailed maps but requires many images and suffers with lighting changes and textureless areas. Initialization with feature-based tracking.

KinectFusion: RGB-D dense mapping and odometry using Kinect. Uses TSDF fusion and ICP. Can only map a fixed volume and relies on slow camera movements.

Kintinuous: Extension of KinectFusion to large-scale maps and adds loop closures. KinectFusion builds small submaps attached to poses which are loaded dynamically. When loop closure is detected via place recognition, the submap poses are changed, but their contents are fixed.

ElasticFusion: RGB-D SLAM using surfel maps. Each surfel has a position, normal, radius, colour and confidence. On loop closure a polygonal mesh is created from the surfel map, elastically deformed, and surfel positions are recalculated using interpolation.

Neural Radiance Field (NeRF) Methods: Implicit representation of the scene in a neural network. Sacrifices geometric accuracy but allows rendering realistic views from unseen angles. **NICE-SLAM** does SLAM using NeRFs in real time. **iSDF** uses SDFs stored implicitly in a neural network.

10 Useful Identities

$$\sin\left(x+\tfrac{\pi}{2}\right)=\cos(x) \quad \sin\left(x-\tfrac{\pi}{2}\right)=-\cos(x) \quad \sin\left(\tfrac{3\pi}{2}-x\right)=-\cos(x)$$
$$\cos\left(x+\tfrac{\pi}{2}\right)=-\sin(x) \quad \cos\left(x-\tfrac{\pi}{2}\right)=\sin(x) \quad \cos\left(\tfrac{3\pi}{2}-x\right)=\sin(x)$$
$$\sin(\tilde{\theta}+\delta\theta)\approx\sin\tilde{\theta}+\cos(\tilde{\theta})\delta\theta \quad \cos(\tilde{\theta}+\delta\theta)\approx\cos\tilde{\theta}-\sin(\tilde{\theta})\delta\theta$$
$$\sin(x+y)=\sin x\cos y+\cos x\sin y \quad \sin(x-y)=\sin x\cos y-\cos x\sin y$$
$$\cos(x+y)=\cos x\cos y-\sin x\sin y \quad \cos(x-y)=\cos x\cos y+\sin x\sin y$$
$$\sin(\tan^{-1}x)=\frac{x}{\sqrt{1+x^2}} \qquad \cos(\tan^{-1}x)=\frac{1}{\sqrt{1+x^2}}$$
$$\sin(\cos^{-1}x)=\cos(\sin^{-1}x)=\sqrt{1-x^2}$$
$$\operatorname{atan2}(-y,x)=-\operatorname{atan2}(y,x) \qquad \operatorname{atan2}(y,-x)=\pi-\operatorname{atan2}(y,x)$$
$$\forall a>0,\ \operatorname{atan2}(ay,ax)=\operatorname{atan2}(y,x)$$
$$\operatorname{atan2}\left(x,\sqrt{1-x^2}\right)=\sin^{-1}x \qquad \operatorname{atan2}\left(\pm\sqrt{1-x^2},x\right)=\pm\cos^{-1}x$$
$$\operatorname{atan2}\left(x,-\sqrt{1-x^2}\right)=\begin{cases}\pi-\sin^{-1}x & x>0\\ -\pi-\sin^{-1}(x) & x<0\end{cases}$$
$$v_1\cdot v_2=\|v_1\|\|v_2\|\cos\theta \qquad v_1\times v_2=\|v_1\|\|v_2\|\sin(\theta)\hat{n}$$
$$\boldsymbol{A}^{-1}=\frac{1}{\det\boldsymbol{A}}(\operatorname{cof}\boldsymbol{A})^T \qquad \begin{bmatrix}a&b\\c&d\end{bmatrix}^{-1}=\frac{1}{ad-bc}\begin{bmatrix}d&-b\\-c&a\end{bmatrix}$$
$$\frac{\mathrm{d}}{\mathrm{d}x}\sin^{-1}x=\frac{1}{\sqrt{1-x^2}} \quad \frac{\mathrm{d}}{\mathrm{d}x}\cos^{-1}x=-\frac{1}{\sqrt{1-x^2}} \quad \frac{\mathrm{d}}{\mathrm{d}x}\tan^{-1}x=\frac{1}{1+x^2}$$

Common Derivatives: (Denom. layout/gradients are *column vectors*)

$$\frac{\partial}{\partial\boldsymbol{x}}\boldsymbol{A}\boldsymbol{x}=\boldsymbol{A} \qquad \frac{\partial}{\partial\boldsymbol{x}}\boldsymbol{x}^T\boldsymbol{A}=\boldsymbol{A}^T \qquad \frac{\partial}{\partial\boldsymbol{x}}\boldsymbol{x}^T\boldsymbol{b}=\frac{\partial}{\partial\boldsymbol{x}}\boldsymbol{b}^T\boldsymbol{x}=\boldsymbol{b}$$
$$\frac{\partial}{\partial\boldsymbol{x}}\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x}=(\boldsymbol{A}+\boldsymbol{A}^T)\boldsymbol{x} \qquad \frac{\partial f}{\partial\boldsymbol{x}}=\left(\frac{\partial\boldsymbol{u}}{\partial\boldsymbol{x}}\right)^T\frac{\partial f}{\partial\boldsymbol{u}} \qquad \frac{\partial\|\boldsymbol{x}\|}{\partial\boldsymbol{x}}=\frac{1}{\|\boldsymbol{x}\|}\boldsymbol{x}^T$$