Simon Ramchandani - simonra@student.ethz.ch

1 Motion and Mapping

1.1 Motion

1.1.1 Velocity Motion Model (control-based prediction)

Estimates pose using commanded velocities (v, ω) and Δt (dead reckoning). Assumes ideal motion + noise (no slip).
Pros: Simple, requires only control input.
Cons: Drift accumulates over time.

1.1.2 Odometry Motion Model (measurement-based update)

Estimates pose from measured movement using wheel encoders (v_l, v_r).
Pros: More accurate with good sensors.
Cons: Still affected by slip and terrain.

1.1.3 Simplified Velocity Motion Model

Assumes motion as **straight-line move + rotation**.

State update: $\begin{bmatrix} x_t \\ y_t \\ \phi_t \end{bmatrix} = \begin{bmatrix} x_{t-1} \\ y_{t-1} \\ \phi_{t-1} \end{bmatrix} + \begin{bmatrix} v_t \Delta t \cdot \cos(\phi_{t-1}) \\ v_t \Delta t \cdot \sin(\phi_{t-1}) \\ \omega_t \Delta t \end{bmatrix}$

1.1.4 Two-Wheel Encoder Model

Linear velocity (v): $v_t = \frac{W}{4}(\dot{\vartheta}_{t,R} - \dot{\vartheta}_{t,L})$

Angular velocity (ω): $\omega_t = \frac{W}{2L}(\dot{\vartheta}_{t,R} - \dot{\vartheta}_{t,L})$

Turn radius (r): $r_t = \frac{v_t}{\omega_t} = \frac{L(v_{t,R}+v_{t,L})}{2(v_{t,R}-v_{t,L})} = \frac{L(\dot{\vartheta}_{t,R}+\dot{\vartheta}_{t,L})}{2(\dot{\vartheta}_{t,R}-\dot{\vartheta}_{t,L})}$

L: wheel distance, W: wheel diameter, $\dot{\vartheta}_{t,R}$ and $\dot{\vartheta}_{t,L}$: angular velocity
Note: no slipping and ideal motion.

1.2 Occupancy Grids

Represents the environment as a 2D grid where each cell's probability (often in log-odds) indicates occupancy.

- **Free:** No obstacle.
- **Occupied:** Contains an obstacle.
- **Inflated:** Buffer area around obstacles.
- **Log-odds Threshold:** Determines when a cell switches state.

1.3 Line Algorithms

Bresenham: Step along longest axis; round to nearest grid cell.
Diagonal Addition: Step in direction of last cell; allows diagonal moves.
Collision Detection: Extended Bresenham; includes all intersected cells using geometry.

1.4 Log Odds Binary Bayes Filter

Occupancy: Represent cells as *occupied* or *free* using log-odds.
Posterior Update: $l_{t,i,j} = l_{t-1,i,j} + l_{\text{cell}} - l_0$
l_{cell}: log-odds from measurement, l_0: prior (typically 0).
Thresholds: Occupied if $l >$, free if $l < -1.9956$ ($\approx 99\%$ confidence).
Why? Avoids instability near 0/1; simplifies Bayes updates to addition.

2 Planning

2.1 Common Terms

Criteria

- **Completeness**: Finds solution if one exists.
- **Optimality**: Finds shortest path.
- **Terminable**: Always finishes in finite time.

Characteristics

- **Admissible**: Heuristic $\leq$ true cost.

2.2 Distance Metrics

- **Manhattan (L1)**: $d = |x_1 - x_2| + |y_1 - y_2|$
  Grid moves only (no diagonal).
- **Euclidean (L2)**: $d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$
  Straight-line distance.
- **Chebyshev (L∞)**: $d = \max(|x_1 - x_2|, |y_1 - y_2|)$
  Allows 8 directions with equal cost.
- **Octile**: $d = \max(dx, dy) + (\sqrt{2} - 1) \cdot \min(dx, dy)$
  Diagonals cost $\sqrt{2}$, orthogonal cost 1.

2.3 Cost Terms in Path Planning

G-cost (Motion Cost): Cost from start to current node; considers explored area + obstacles; updated if a shorter path is found; obstacles $\to \infty$.
H-cost (Heuristic): Estimated cost to goal (ignores obstacles); static during search; must be admissible (never overestimates).
F-cost: $f = g + h$ — total cost used by A*; drives optimal path through lowest-f nodes.

2.4 Path Planners

DFS (Depth-First Search)

- Uses a stack (LIFO); step-based.
- Fast, complete, terminable, but **not optimal**.
- Explores deep and backtracks; poor quality, unsuitable for motion pl.

BFS (Breadth-First / Flood Fill)

- Uses a queue (FIFO); step-based.
- Complete, terminable, **optimal in 4-dir** (uniform expansion).
- Upgraded DFS; may be non-optimal when 8-dir moves are allowed.

GBFS (Greedy Best-First Search) (Upgraded DFS guided by goal)

- Heuristic-only (h), stack-like.
- Fast, complete, terminable, **not optimal**.
- Prioritizes nodes **closest to goal** (ignores actual cost); fast but risky.

Dijkstra

- Uses motion cost only (g), queue-like.
- Complete, terminable, **optimal**.
- Explores from the start, prioritizing lowest cost; robust but slow.

A*

- Combines cost and heuristic: $f = g + h$ (open list, optional closed).
- Complete, terminable, **optimal**; fastest optimal method.
- Balances g and h: if $f = \alpha g + \beta h$, then $\alpha > \beta$ behaves like Dijkstra; $\beta > \alpha$ behaves like GBFS.

2.5 Artificial Potential Fields

Concept: Path planning via artificial forces:
$\to$ *Attractive* to goal, $\to$ *Repulsive* from obstacles
Forces:
Attractive (global, $\propto$ distance to goal),
Repulsive (local, $\propto \frac{1}{\text{dist. to obs}}$)
Equations:
$U(q) = U_{att}(q) + \sum_o U_{rep,o}(q)$
$F(q) = -\nabla U(q) = -\nabla U_{att}(q) - \sum_o \nabla U_{rep,o}(q)$
Motion: Use $F(q)$ as force, accel., or velocity (most common)
Pros: Smooth, fast, easy to integrate with control
Cons: Not complete (local minima), not optimal, improvable with meta-heuristics

3 Local Planners

3.1 Local Planners

Global Planner (Planner):

- Plans optimal path from start to goal / Considers the full map

Smoother:

- Smooths the global path to make it feasible or efficient

Local Planner (Controller):

- Generates short-term commands (velocities/trajectories)
- Reacts to **dynamic obstacles** (adjusts global path locally)

3.2 Smoothers

- **Gradient Descent:** $J = \sum_{i=1}^{N-1} \omega_s \left(\Delta s_{i+1} - \Delta s_i\right)^2 + \omega_d (s_i - x_i)^2$
  s_i: Point i along the smooth path, x_i: Point i along the original path, $\Delta s_i = s_i - s_{i-1}$: Difference between adjacent points on smooth path, ω_s, ω_d: Weighting factors for smoothness and deviation from the original path
  Minimizes quadratic cost to reduce sharp turns while staying close to original path.
- **Simple Moving Average:** $s_i = \frac{1}{2m+1} \sum_{j=-m}^{m} x_{i+j}$
  s_i: Smoothed point at index i, x_i: Original path point at index i, m: Window size
  This smooths the path by averaging over $2m + 1$ neighboring points.
- **Weighted Moving Average:** $s_i = \sum_{j=-m}^{m} a_{i+j} x_{i+j}$
  a_{i+j} are the weights for each neighboring point, with the sum of weights typically equal to 1.
  The weights allow more control over the influence of each point in the window.

3.3 Spline Fitting

C0: $f(x, y)$ continuous.
Discontinuous gradient $\to$ jerky transitions.
C1: $f(x, y), f'(x, y)$ continuous.
Smooth velocity, but jerky acceleration (e.g., elevator jerk).
C2: $f(x, y), f'(x, y), f''(x, y)$ continuous.
Smooth position, velocity, and acceleration at joints.

3.4 Pure Pursuit

Steps:

1. Find closest point on path to robot.
2. Move along path toward goal.
3. Identify lookahead point (first point $\geq$ distance L_t).
4. Drive robot along a circular arc to that point.

Equations:

Curvature: $\kappa = \frac{2y'}{L^2}$
Angular velocity: $\omega_t = \kappa v_t$
Lookahead distance based on speed: $L_t = v_t g_l$

- **Curvature Heuristic:** $v'_t = \begin{cases} v_t \kappa_h / \kappa & \text{if } \kappa > \kappa_h \\ v_t & \text{otherwise} \end{cases}$
- **Proximity Heuristic:** $v'_t = \begin{cases} v_t \cdot \frac{d_O}{d_{prox}} & \text{if } d_O \leq d_{prox} \\ v_t & \text{otherwise} \end{cases}$

4 Introduction to UAV

4.1 Positioning System & Sensors

External Systems (Global): GPS, Beidou, VICON, UWB
Internal Systems (Onboard): Vision, LiDAR, Radar, Sonar

4.2 Control Systems

Core subsystem for stabilizing and guiding UAVs.
Inner Loop (Attitude & Rate Control):
- PID Control (standard), Optimal-, Robust-, Nonlinear Control

Outer Loop (Trajectory & Position Control):
- PID Control (standard), Pole Placement, RPT-, Robust Control

4.3 Onboard Software – Task Scheduling

Real-time multitasking system managed by a task scheduler:
Task sync in sequ: `IMU → DAQ → CAM → CTL → SVO → NET → CMM → DLG`.

5 UAV GPS INS Navigation

5.1 Coordinate Frames: Geodetic, ECEF, NED

Geodetic Coordinates $\mathbf{P}_g = (\lambda, \phi, h)$
- λ: Longitude – angle from Prime Meridian.
- ϕ: Latitude – angle from equatorial plane.
- h: Altitude above the reference ellipsoid.

ECEF Coordinates $\mathbf{P}_e = (X_e, Y_e, Z_e)$
- Z_e: through true north.
- X_e: intersection of Equator and Prime Meridian.
- Y_e: 90° east of X_e, lies on Equator.

Local NED Coordinates $\mathbf{P}_n = (X_n, Y_n, Z_n)$
- X_n: geodetic north, Y_n: geodetic east, Z_n: down (normal to ellipsoid).

5.2 Coordinate Transforms

$$e^2 = 1 - \frac{b^2}{a^2}, \quad N_\phi = \frac{a}{\sqrt{1 - e^2 \sin^2 \phi}}$$

Geodetic to ECEF:

$$X_e = (N_\phi + h)\cos\phi\cos\lambda$$
$$Y_e = (N_\phi + h)\cos\phi\sin\lambda$$
$$Z_e = \left(\frac{b^2}{a^2}N_\phi + h\right)\sin\phi$$

a = equatorial radius, b = polar radius, h = altitude

ECEF to NED:

$$\begin{bmatrix} X_n \\ Y_n \\ Z_n \end{bmatrix} = \underbrace{\begin{bmatrix} -\sin\phi\cos\lambda & -\sin\lambda & -\cos\phi\cos\lambda \\ -\sin\phi\sin\lambda & \cos\lambda & -\cos\phi\sin\lambda \\ \cos\phi & 0 & -\sin\phi \end{bmatrix}^T}_{\mathbf{R^T}} \begin{bmatrix} X_e - X_{e0} \\ Y_e - Y_{e0} \\ Z_e - Z_{e0} \end{bmatrix}$$

Where (X_{e0}, Y_{e0}, Z_{e0}) is the ECEF origin of the local NED frame.

NED to Body Frame:

$$\begin{bmatrix} X_b \\ Y_b \\ Z_b \end{bmatrix} = \underbrace{\begin{bmatrix} c_\psi c_\theta & c_\psi s_\theta s_\phi - s_\psi c_\phi & c_\psi s_\theta c_\phi + s_\psi s_\phi \\ s_\psi c_\theta & s_\psi s_\theta s_\phi + c_\psi c_\phi & s_\psi s_\theta c_\phi - c_\psi s_\phi \\ -s_\theta & c_\theta s_\phi & c_\theta c_\phi \end{bmatrix}}_{\mathbf{R_{b\leftarrow n}}} \begin{bmatrix} X_n \\ Y_n \\ Z_n \end{bmatrix}$$

ϕ = roll, θ = pitch, ψ = yaw

5.3 Kalman Filter Summary

Overview: The Kalman Filter is a recursive estimator that blends a model prediction with noisy measurements to produce optimal state estimates.

System Model:

$$x_{k+1} = F\,x_k + G\,u_k + w_k, \quad w_k \sim \mathcal{N}(0, Q)$$
$$y_k = H\,x_k + v_k, \quad v_k \sim \mathcal{N}(0, R)$$

- $\mathbf{x}_k$: State vector at time k.
- F: State transition matrix (how the state evolves).
- $\mathbf{u}_k$: Control vector.
- G: Control input matrix.
- w_k: Process noise (uncertainty in the model) with covariance Q.
- $\mathbf{y}_k$: Measurement vector.
- H: Measurement matrix (maps state to measurement).
- v_k: Measurement noise with covariance R.

1. Prediction:

$$\hat{x}_{k+1|k} = F\,\hat{x}_{k|k} + G\,u_k$$
$$P_{k+1|k} = F\,P_{k|k}\,F^T + Q$$

- $\hat{x}_{k+1|k}$: Predicted state estimate.
- $P_{k+1|k}$: Predicted uncertainty (covariance matrix).

2. Correction:

$$S_k = H\,P_{k+1|k}\,H^T + R$$
$$K_k = P_{k+1|k}\,H^T\,S_k^{-1}$$
$$\Delta x_{k+1} = K_k\left(y_{k+1} - H\,\hat{x}_{k+1|k}\right)$$

- S_k: Innovation covariance.
- K_k: Kalman Gain.
- Δx_{k+1}: Innovation correction.

3. Update:

$$\hat{x}_{k+1|k+1} = \hat{x}_{k+1|k} + \Delta x_{k+1} \quad \text{(State update)}$$
$$P_{k+1|k+1} = (I - K_k\,H)\,P_{k+1|k} \quad \text{(Covariance update)}$$

- The filter minimizes the mean squared error (optimal estimate) under linearity and Gaussian noise assumptions.

5.4 From Angular Rates to Euler Angle Rates

Given the angular rates measured in the body frame,the time derivatives of the Euler angles (roll ϕ, pitch θ, and yaw ψ) are obtained using the transformation: $\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi/\cos\theta & \cos\phi/\cos\theta \end{bmatrix} \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}$.

Explanation:
- $\dot{\phi}$ is the roll rate, $\dot{\theta}$ is the pitch rate, and $\dot{\psi}$ is the yaw rate.
- The transformation matrix accounts for the coupling between Euler angles due to the non-commutative nature of 3D rotations.
- The terms $\sin\phi$, $\cos\phi$, $\tan\theta$, and $\cos\theta$ appear because the relationship between the body angular velocity and the Euler angle rates depends on the current orientation.

5.5 Control Paradigms

- **PID Control:** Minimizes error using proportional, integral, and derivative terms. *Design:* Tune K_P, K_I, K_D. *Use:* Industrial, robotics, temperature control (simple + effective).
- **LQR:** Minimizes cost balancing state error and control effort. *Design:* Solve Riccati equation with Q, R weights. *Use:* Aerospace, drones, known linear systems.
- **$\mathbf{H}_\infty$:** Ensures robust performance under model uncertainty/disturbances. *Design:* Optimize worst-case disturbance gain ($\mathcal{H}_\infty$-norm). *Use:* Safety-critical/uncertain systems (e.g., autonomous vehicles).
- **RPT Control:** Guarantees perfect tracking + robustness to uncertainty. *Design:* Internal model principle + robust controller. *Use:* UAV/robot path tracking with disturbances.

6 Quiz Questions

6.1 Sensor Types

Interoceptive (Internal State): IMU (accel/gyro), Wheel Odometry, Joint Encoders, Motor Current, Battery Voltage
Exteroceptive (External Environment): LiDAR, Radar, Camera, Sonar, Barometer (pressure), Magnetometer (heading), GPS, UWB

6.2 SLAM (Simultaneous Localization and Mapping)

Goal: Estimate robot pose and build a map simultaneously.
Approaches:
- **Filter-based SLAM:** (e.g., EKF-SLAM, FastSLAM)
  - Uses recursive Bayesian filtering with MMSE estimation.
  - Key Steps: Prediction: Propagate state & covariance via motion model.
    Correction: Update estimates using sensor measurements.
  - Objective: MMSE for robot pose & landmark positions.
- **Graph-based SLAM:**
  - Constructs a pose graph (nodes = poses/landmarks).
  - Global optimization (e.g., nonlinear least squares) refines the trajectory.
  - More scalable for large environments.

Data Association: Match observations to landmarks using the **Mahalanobis Distance** (accounts for prediction and measurement uncertainty).
Drift & Correction:
- **Drift:** Accumulated pose error from odometry.
- **Countermeasures:** Loop closure detection, sensor calibration, multi-sensor fusion. **Dead Reckoning** alone worsens drift.

Key Challenges: Sensor noise/uncertainty, ambiguous landmark matches, real-time computation in large-scale maps, and non-linearities (often requiring linearization in EKF-SLAM).

6.3 Scan Matching & Related Methods

Scan Matching: Aligns sensor scans (e.g., laser data) to estimate robot motion.
- **CSM:** Exhaustively searches translations/rotations using correlation metrics.
- **ICP:** Iteratively matches closest points to minimize Euclidean error.

Split-and-Merge: Segments scans into line segments, then merges similar ones to represent geometric primitives instead of performing direct scan matching.