Mission Coordination Cheat Sheet

1 UAV and UGV Control Architectures

Control Architecture Definition: The control architecture of an unmanned aerial or ground vehicle (UAV/UGV) includes all components and layers needed to plan and execute its missions (from high-level planning to low-level control) for single-vehicle or multi-vehicle operations.

Architecture styles:

- *Hierarchical architecture:* Control system is layered (e.g. mission planning, trajectory generation, stabilization). High-level decisions propagate down to low-level controllers, with defined interfaces between layers.
- *Modular architecture:* System is divided into functional modules (e.g. perception, planning, control, communication), each performing a specific task with well-defined inputs/outputs. This improves maintainability and allows parallel development.
- *Neural-enhanced architecture:* Integrates neural network components into control loops or decision-making modules. For example, a neural network might learn and compensate for model uncertainties or optimize control commands online.

Communications Architecture (Multi-robot):

- *Centralized:* All robots communicate with a central node (e.g. a ground station or leader). The central node gathers information and issues commands. Simplifies coordination but introduces a single point of failure.
- *Decentralized (Distributed):* No single leader; robots communicate peer-to-peer or in a network. Decisions are made collaboratively or locally with shared information. More robust to individual failures but requires more complex communication and consensus algorithms.
- *Hybrid:* Combination of central and distributed: e.g. clusters of robots may have internal leaders, or a central station is used for high-level coordination while low-level actions are distributed.

Levels of Autonomy: Teleoperated (human-controlled), Semi-autonomous (robots perform some tasks autonomously under supervision), and Fully autonomous (robots plan and act on their own). Fully autonomous UAV/UGV teams allow inter-robot communication and on-board computation (companion computers) to handle complex coordination, enabling more reactive responses to failures.

Single-Vehicle Architecture: A typical UAV control system is organized in layers and modules:

- *High-level planning:* Mission planning or global path planning, which generates waypoints or strategic goals for the mission.
- *Guidance and trajectory generation:* A local trajectory planner computes feasible reference trajectories or setpoints (respecting kinematic limits and avoiding obstacles).
- *State estimation (Localization):* Sensor fusion of GNSS (GPS), IMU, barometer, camera, LiDAR etc. to estimate the vehicle's pose (position, velocity, attitude).
- *Low-level control:* Autopilot or flight controller (for UAV) / motor and steering controller (for UGV) that tracks the desired trajectory or attitude by issuing commands to motors/actuators.
- *Perception and avoidance:* Obstacle detection from onboard sensors (e.g. cameras, LiDAR) and mapping of free space, feeding into the trajectory planner for collision avoidance.

Typical Sensors: Inertial Measurement Unit (accelerometers + gyros) for motion, GNSS (e.g. GPS) for global position, barometer for altitude, magnetometer for heading, and vision sensors (camera) or LiDAR for environment mapping.

2 Modeling of UAV and UGV Dynamics

2.1 Coordinate Frames and Notation

- *Frames:* Define an Earth-fixed inertial frame and a body frame attached to the vehicle. Position $\mathbf{p} = (x, y, z)$ is given in inertial frame, while orientation can be given by a rotation matrix R (from body to inertial) or equivalently by Euler angles or quaternions.
- *Rotation matrix R:* $R \in SO(3)$ (i.e. $R^T R = I$, $\det R = 1$). The time derivative of R relates to the body angular velocity $\boldsymbol{\omega} = (p, q, r)$ by $\dot{R} = [\boldsymbol{\omega}]_\times R . [\boldsymbol{\omega}]_\times = \begin{pmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{pmatrix}$, which satisfies $[\boldsymbol{\omega}]_\times \mathbf{v} = \boldsymbol{\omega} \times \mathbf{v}$.
- *Notation:* m (mass); I_x, I_y, I_z (moments of inertia about body axes); J_r (rotational inertia of rotor/propeller); g (9.81 m/s^2, gravitational acceleration); $k_{tdx}, k_{tdy}, k_{tdz}$ (drag coefficients along body axes, for aerodynamic drag modeling).

2.2 UAV Equations of Motion

Translational dynamics: Using Newton's second law, $m\ddot{\mathbf{p}} = \sum \mathbf{F}_{ext}$. For a quadrotor UAV, the main forces are thrust and weight. Let $\mathbf{p} = (x, y, z)$ in an inertial frame (with z upward), and assume the quadrotor produces total thrust T along its body z-axis. The weight is $m\mathbf{g} = [0, 0, -mg]^T$. The rotation matrix R maps the body-axis unit vector $\mathbf{e}_3 = [0, 0, 1]^T$ to inertial. Thus:

$$m \begin{pmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{pmatrix} = m \begin{pmatrix} 0 \\ 0 \\ -g \end{pmatrix} + T\, R \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}.$$

Expanding the right-hand side gives the UAV translational motion:

$$\begin{aligned} \ddot{x} &= \frac{T}{m}(\cos\Phi \sin\Theta \cos\Psi + \sin\Phi \sin\Psi), \\ \ddot{y} &= \frac{T}{m}(\cos\Phi \sin\Theta \sin\Psi - \sin\Phi \cos\Psi), \\ \ddot{z} &= \frac{T}{m} \cos\Phi \cos\Theta - g\ , \end{aligned}$$

using roll Φ, pitch Θ, yaw Ψ angles.

Rotational dynamics: Applying Euler's rotation equations (Newton–Euler) about the center of mass:

$$\mathbf{I}\dot{\boldsymbol{\omega}} + \boldsymbol{\omega} \times (\mathbf{I}\boldsymbol{\omega}) = \boldsymbol{\tau},$$

where $\boldsymbol{\omega} = (p, q, r)$ are body angular rates and $\boldsymbol{\tau} = (\tau_\Phi, \tau_\Theta, \tau_\Psi)$ are control torques (roll, pitch, yaw).

$$\begin{aligned} I_x \dot{p} &= (I_y - I_z)\, qr + \tau_\Phi, \\ I_y \dot{q} &= (I_z - I_x)\, pr + \tau_\Theta, \\ I_z \dot{r} &= \tau_\Psi\ , \end{aligned}$$

where the $(I_y - I_z)qr$ term etc. are gyroscopic coupling terms. For a quadrotor, torques τ_Φ, τ_Θ are generated by thrust differences between motors, and τ_Ψ by counter-torque imbalance of propellers.

Kinematics (Euler angles): The Euler angle rates relate to body rates $\boldsymbol{\omega} = (p, q, r)$ as:

$$\begin{pmatrix} \dot{\Phi} \\ \dot{\Theta} \\ \dot{\Psi} \end{pmatrix} = \begin{pmatrix} 1 & \sin\Phi \tan\Theta & \cos\Phi \tan\Theta \\ 0 & \cos\Phi & -\sin\Phi \\ 0 & \sin\Phi / \cos\Theta & \cos\Phi / \cos\Theta \end{pmatrix} \begin{pmatrix} p \\ q \\ r \end{pmatrix},$$

for a 3-2-1 (roll-pitch-yaw) rotation sequence (assuming $\cos\Theta \neq 0$).

2.3 UGV Models

Kinematic (unicycle) model: A simple nonholonomic ground vehicle model (applicable to differential-drive or car-like robots at low speed without slip):

$$\dot{x} = v\cos\theta, \qquad \dot{y} = v\sin\theta, \qquad \dot{\theta} = \omega,$$

where (x, y) is the position, θ is the heading, v is forward speed and ω is the yaw rate. For a car with steering angle δ (bicycle approximation), $\omega = \frac{v}{L}\tan\delta$ (with L the wheelbase). This kinematic model assumes no sideways slipping (pure rolling).

Dynamic bicycle model (2-DOF): This model accounts for lateral dynamics and tire slip for a car. State variables: longitudinal velocity v_x, lateral velocity v_y, and yaw rate r.

$$\alpha_f = \frac{v_y + l_f r}{v_x} - \delta, \qquad \alpha_r = \frac{v_y - l_r r}{v_x},$$

and lateral tire forces (assuming linear model) $F_{yf} = C_f\alpha_f$, $F_{yr} = C_r\alpha_r$.

$$\begin{aligned} m(\dot{v}_y + v_x r) &= F_{yf} + F_{yr}, \\ I_z\dot{r} &= l_f F_{yf} - l_r F_{yr}\ , \end{aligned}$$

with m mass, I_z yaw inertia. These nonlinear equations show coupling between v_y and r; note v_x may be treated as approximately constant or modeled with an additional equation if needed. In practice v_y is difficult to measure directly (requires special sensors), so observers are used to estimate it.

State-space representation: The above models can be expressed in state-space form. Generally, $\dot{x} = f(x, u)$ (continuous) or $x_{k+1} = f_d(x_k, u_k)$ (discrete). For linear systems (or linearized models) we write $\dot{x} = Ax + Bu$, $y = Cx + Du$. For example, a discretized linear model for a vehicle:

$$x_{k+1} = F_k\, x_k + B_k\, u_k + w_k,$$

where w_k is process noise.

3 Control Methods for UAV/UGV

3.1 PID and PD Control

- **PID Controller:** A classical feedback controller with Proportional-Integral-Derivative terms. For example, for a position coordinate with error $e(t) = x_{\text{des}}(t) - x(t)$, a PID law is $u(t) = K_P e(t) + K_I\int_0^t e(\tau)\,d\tau + K_D\dot{e}(t)$. The K_P term reduces immediate error, K_I term removes steady-state error, and K_D term adds damping and anticipates changes. PID control is widely used in both UAVs (e.g. attitude stabilization, altitude hold) and UGVs (speed or steering control) due to its simplicity. The gains (K_P, K_I, K_D) are tuned empirically or via rules.

- **PD Controller:** A PID without the integral part, i.e. $u = K_P e + K_D\dot{e}$. PD control reacts to error and its rate, providing faster response and less overshoot for inner-loop control (like UAV attitude) but may have steady-state error if used alone (often an outer loop handles that).

- *Cascade control:* Control loops are often nested. For example, a UAV's position loop (outer loop) computes desired accelerations or velocities, which feed into an inner attitude loop. The attitude loop (inner, often PD) commands the motors to achieve desired angles quickly. Thus, slow dynamics (position) are handled by the outer loop and fast dynamics (attitude) by the inner loop.

For a quadrotor, the horizontal acceleration commands (a_x^{ctrl}, a_y^{ctrl}) from the outer-loop controller can be translated into desired roll and pitch. Assuming a desired yaw Ψ^{des}, one set of formulas is:

$$\Phi^{des} = \frac{1}{g}\big(\cos\Psi^{des}\, a_y^{ctrl} - \sin\Psi^{des}\, a_x^{ctrl}\big),$$

$$\Theta^{des} = -\frac{1}{g}\big(\cos\Psi^{des}\, a_x^{ctrl} + \sin\Psi^{des}\, a_y^{ctrl}\big),$$

so that the UAV banks/tilts to accelerate in the horizontal plane. The collective thrust command would be $T = m(g - a_z^{ctrl})$. (These expressions assume small angles and z roughly vertical).

3.2 Sliding Mode Control (SMC)

- **Sliding surface:** Choose a surface $s(x,t)=0$ in the state-error space that represents desired motion. For instance, for position tracking error $e(t)$, one can set $s=\dot{e}+\lambda e$ (with $\lambda>0$) so that $s=0$ implies an exponentially decaying $e(t)$.
- **Control law:** Apply a control input that drives $s\to 0$ and keeps it there (on the "sliding surface"). A typical law: $u=u_{eq}(x)-K\,\mathrm{sgn}(s)$, where u_{eq} cancels nominal dynamics and the $-K\,\mathrm{sgn}(s)$ term forces s toward zero. K must be large enough to overcome uncertainties.
- **Robustness:** Once on the sliding surface ($s=0$), the system is insensitive to certain model errors and disturbances (robustness by design). However, the discontinuous sgn control can cause chattering (high-frequency oscillation). In practice, one uses a saturation or boundary layer around $s=0$ to smooth the control, or employs higher-order sliding mode techniques to reduce chattering.
- **Example:** Given a simplified UAV horizontal motion model $\ddot{x}=F(x)+G\,u+\delta(x,t)$ (with δ representing uncertainty),

3.3 Neural-Network Enhanced Control

- **Adaptive neural control:** Neural networks can be integrated with controllers to handle complex or uncertain dynamics. A neural network (NN) can learn in real time to approximate an unknown function or disturbance $\delta(x,t)$ in the system, and the controller uses the NN output to cancel or compensate it.
- **AI in guidance/control:** Beyond low-level control, NNs are used for high-level tasks (path planning, target recognition) and in *neuroadaptive controllers* that augment classical control laws. In such controllers, the NN serves as a function approximator improving performance and robustness (e.g., reducing tracking error under model changes). Stability is ensured via adaptive control theory (Lyapunov-based adaptation laws).

4 Localization and State Estimation

4.1 Sensor Fusion Basics

- Mobile robots use multiple sensors to observe their state and environment: IMU (measures acceleration and rotation rates), GNSS (e.g. GPS for global position), barometer (altitude), magnetometer (heading), wheel encoders (odometry), cameras and LiDAR (mapping and obstacle detection). Each sensor has noise and limitations (e.g. IMU drifts, GPS can lose signal, cameras have limited range).
- **Sensor fusion:** The process of combining sensor data to estimate the robot's state more accurately and robustly than any single sensor could. By leveraging complementary information (e.g. fast IMU updates with stable GPS readings), fusion algorithms can correct errors (IMU drift is corrected when GPS is available, etc.).
- A common approach is the Kalman Filter (KF) for linear Gaussian systems, or its nonlinear extensions for UAV/UGV dynamics: Extended KF (EKF) and Unscented KF, which handle nonlinear models, and Particle Filters for highly non-linear/non-Gaussian scenarios. The choice depends on model complexity and noise characteristics.

4.2 Kalman Filtering (Estimation)

In a discrete-time Kalman Filter, the system model is:

$$x_{k+1}=F_k\,x_k+B_k\,u_k+w_k,\qquad y_k=H_k\,x_k+v_k,$$

with w_k process noise (covariance Q) and v_k measurement noise (covariance R).

- *Prediction (Time update):* $\hat{x}_{k|k-1}=F_{k-1}\,\hat{x}_{k-1|k-1}+B_{k-1}u_{k-1};\quad P_{k|k-1}=F_{k-1}P_{k-1|k-1}F_{k-1}^T+Q_{k-1}.$

- *Correction (Measurement update):* $K_k = P_{k|k-1}H_k^T(H_kP_{k|k-1}H_k^T + R_k)^{-1}$ (Kalman gain). Update state $\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k\big(y_k - H_k\hat{x}_{k|k-1}\big)$. Update covariance $P_{k|k} = (I - K_kH_k)\,P_{k|k-1}$.

The Extended Kalman Filter (EKF) handles nonlinear models by linearizing about the current state estimate: use $F_k \approx \frac{\partial f}{\partial x}\big|_{\hat{x}_{k|k-1}}$ and $H_k \approx \frac{\partial h}{\partial x}\big|_{\hat{x}_{k|k-1}}$ for system $\dot{x} = f(x,u)$ and measurement $y = h(x)$. The EKF then applies the same predict-update steps with these Jacobians.

Kalman filtering provides real-time state estimates. For example, a UAV can fuse fast IMU data with slower, drift-free GPS: during GPS outages, the prediction (integrating IMU) carries the estimate, and when GPS signals resume, the correction brings the estimate back on track. Proper tuning of Q and R is critical: higher Q assumes more model uncertainty (trust sensors more), higher R assumes more sensor noise (trust the model more).

4.3 High-Gain Observers and Other Observers

- **Deterministic observer design:** Given a known model, one can design an observer (e.g., Luenberger observer) instead of a stochastic filter. For a system $\dot{x} = f(x,u)$ with output $y = Cx$, an observer might be: $\dot{\hat{x}} = f(\hat{x},u) + L\big(y - C\hat{x}\big)$. L is chosen such that the estimation error $e_x = x - \hat{x}$ converges to 0. A **high-gain observer (HGO)** uses large L gains to achieve very fast error convergence.

- **Application to UAV:** One can separate fast and slow states and observe them separately. For example, an HGO can estimate translational velocities $[\dot{x}, \dot{y}, \dot{z}]$ quickly from position and accelerometer measurements, while another observer tracks attitude angles from gyroscope and magnetometer data. High gain yields quick convergence but can amplify sensor noise, so in practice gains are made as high as noise allows.

- **Complementary filters:** A simpler alternative for attitude estimation is the complementary filter, which blends gyroscope integration (high-frequency information) with accelerometer/magnetometer data (low-frequency reference). By tuning a filter constant (e.g. α for gyro vs $1-\alpha$ for accelerometer), it achieves a stable orientation estimate without complex EKF equations. Complementary filters are common in drone flight controllers for roll/pitch estimation.

4.4 SLAM (Simultaneous Localization and Mapping)

- SLAM is the problem of building a map of an unknown environment while simultaneously localizing the robot within that map. It is essential for autonomous navigation in GPS-denied or unexplored areas (e.g. indoor UAV flight or UGV in disaster sites).

- **SLAM algorithms:** One approach is EKF-SLAM, where the state vector includes the robot pose and landmark positions; the EKF updates both as the robot moves and observes new landmarks. Another approach is Graph-SLAM: the robot records constraints (relative poses, loop closures) in a graph and then solves an optimization problem to find the most consistent map/trajectory (often via least-squares optimization).

- **Key challenges:** Data association (recognizing when a current sensor observation corresponds to a previously seen landmark), computational complexity (EKF-SLAM scales poorly as landmarks increase, while graph methods scale better but require batch optimization), and loop closure (correcting the map when revisiting a location after a long time to eliminate drift).

- Modern implementations, like visual SLAM (using cameras) or LiDAR SLAM, often run in real-time on onboard computers. In multi-robot SLAM, teams of robots can share local maps or observations to build a global map faster, but this requires communication and merging maps consistently.

5 Multi-Agent Coordination

5.1 Formation and Fleet Control

- **Leader-Follower:** One agent is designated as the leader and others are followers that maintain a specified offset (distance, angle) from the leader.

- **Virtual Structure:** The team is controlled as if all robots are rigidly attached to an imaginary structure. Each robot is assigned a fixed position relative to a virtual reference frame (the "structure"), rather than following another robot directly.
- **Other methods:** *Behavior-based* approaches give each robot simple behaviors (e.g. maintain distance, align velocity) and the formation emerges from these local interactions. *Consensus-based* control uses distributed algorithms for agents to agree on certain values (e.g. a common heading or formation center); if the communication graph remains connected, the team will converge to coordinated motion.

5.2 Distributed Planning and Coordination

- **Task allocation:** In multi-robot missions, tasks (targets to visit, areas to survey, etc.) must be divided among robots. Strategies range from centralized (a single planner assigns tasks to each robot) to distributed (robots hold auctions or consensus to decide who handles each task). Good task allocation optimizes resource use and avoids duplicate work.
- **Decentralized trajectory planning:** Each robot plans its own path but cooperates to avoid collisions or conflicts. They may share intended trajectories or use priority rules. Without a central coordinator, methods like reciprocal avoidance (each robot adjusts for others) or distributed model predictive control (iteratively solving optimization by communication between neighbors) are used to ensure safe, coordinated motion.
- **Consensus and flocking:** Many coordination algorithms rely on consensus to synchronize state. For example, consensus algorithms can make all robots agree on a rendezvous point or move with the same velocity (flocking behavior). Under appropriate conditions (connected communication graph, stable local controllers), consensus ensures the group acts cohesively without central control.
- **Communication network:** Reliable communication is the backbone of distributed coordination. The network topology (which robot can talk to which) may be fixed or dynamic. Maintaining connectivity is crucial; if the network splits, so does coordination. Algorithms need to handle communication delays or dropouts gracefully (e.g., by planning more conservatively or switching to local control if isolated).
- **Fleet management:** Beyond control laws, practical coordination includes managing robot health (monitoring battery, failures), scheduling recharging, and fail-safe behaviors. For instance, if a robot in a team fails, others might need to reassign its tasks among themselves. A robust mission plan accounts for such contingencies to ensure mission success even with partial team losses.