1. State-Space (SS) Modelling

A linear time-invariant (LTI) system is represented by two time-domain equations. The **State Equation**, $\dot{\mathbf{x}}(t) = A\mathbf{x}(t) + B\mathbf{u}(t)$, describes the system's internal dynamics. The **Output Equation**, $\mathbf{y}(t) = C\mathbf{x}(t) + D\mathbf{u}(t)$, relates these internal states to the measurable output. In this formulation, $\mathbf{x}(t)$ is the **State Vector** (the minimum set of variables describing the system's internal state), $\mathbf{u}(t)$ is the **Input Vector**, and $\mathbf{y}(t)$ is the **Output Vector**. The matrices are A (System), B (Input), C (Output), and D (Feedforward). The C matrix notably acts as a "selector" to determine which states are passed to the output. **Stability** is determined by the eigenvalues of the A matrix; the system is stable if and only if all eigenvalues have a negative real part.

Example 1: ODE to State-Space (SISO, Phase Variable)
Given a 2nd-order ODE: $\ddot{y}(t) + 5\dot{y}(t) + 6y(t) = u(t)$

1. **Identify Order:** This is a 2nd-order ODE, so we need 2 state variables.
2. **Choose States:** We use the "phase variable" form:
   - $x_1(t) = y(t)$ (the output)
   - $x_2(t) = \dot{y}(t)$ (the output's derivative)
3. **Form State Equations ($\dot{\mathbf{x}} = A\mathbf{x} + B\mathbf{u}$):**
   - $\dot{x}_1 = \frac{d}{dt}(y) = \dot{y}$. By our definition, this is $\dot{x}_1 = x_2$.
   - $\dot{x}_2 = \frac{d}{dt}(\dot{y}) = \ddot{y}$. We find $\ddot{y}$ by rearranging the original ODE:
     $\ddot{y}(t) = -6y(t) - 5\dot{y}(t) + u(t)$.
   - Substituting our state variables gives: $\dot{x}_2 = -6x_1 - 5x_2 + u(t)$.
4. **Form Output Equation ($\mathbf{y} = C\mathbf{x} + D\mathbf{u}$):**
   - The output is $y(t)$, which we defined as x_1.
   - $y(t) = (1 \cdot x_1 + 0 \cdot x_2)$.
5. Assemble Matrices:
   $$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -6 & -5 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix}u(t)$$
   $$y(t) = \begin{bmatrix} 1 & 0 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + [0]u(t)$$
   - This gives: $A = \begin{bmatrix} 0 & 1 \\ -6 & -5 \end{bmatrix}, B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, C = \begin{bmatrix} 1 & 0 \end{bmatrix}, D = 0$.

Example 2: State-Space to Transfer Function (SISO)
Formula: $G(s) = C(sI - A)^{-1}B + D$
Given: $A = \begin{bmatrix} -4 & -1 \\ 3 & -1 \end{bmatrix}, B = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, C = \begin{bmatrix} 1 & 0 \end{bmatrix}, D = 0$

1. Calculate $(sI - A)$:
   $$sI - A = s\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \begin{bmatrix} -4 & -1 \\ 3 & -1 \end{bmatrix} = \begin{bmatrix} s+4 & 1 \\ -3 & s+1 \end{bmatrix}$$
2. **Find the inverse, $(sI - A)^{-1}$:**
   - Determinant: $\det(sI - A) = (s+4)(s+1) - (1)(-3) = s^2 + 5s + 7$
   - Inverse: $(sI - A)^{-1} = \frac{1}{s^2+5s+7}\begin{bmatrix} s+1 & -1 \\ 3 & s+4 \end{bmatrix}$
3. **Calculate $G(s) = C \cdot (sI - A)^{-1} \cdot B$:**
   - First, $(sI - A)^{-1}B$:
     $$\frac{1}{s^2+5s+7}\begin{bmatrix} s+1 & -1 \\ 3 & s+4 \end{bmatrix}\begin{bmatrix} 1 \\ 1 \end{bmatrix} = \frac{1}{s^2+5s+7}\begin{bmatrix} (s+1)(1) + (-1)(1) \\ (3)(1) + (s+4)(1) \end{bmatrix} = \frac{1}{s^2+5s+7}\begin{bmatrix} s \\ s+7 \end{bmatrix}$$
   - Then, $C \cdot$ [result]:
     $$G(s) = \begin{bmatrix} 1 & 0 \end{bmatrix}\left(\frac{1}{s^2+5s+7}\begin{bmatrix} s \\ s+7 \end{bmatrix}\right) = \frac{1}{s^2+5s+7}((1)(s) + (0)(s+7))$$
4. **Final Answer:** $G(s) = \frac{s}{s^2+5s+7}$

Example 3: State-Space to Transfer Function Matrix (MIMO)
Formula: $H(s) = C(sI - A)^{-1}B + D$
Given: $A = \begin{bmatrix} -1 & -1 \\ 6.5 & 0 \end{bmatrix}, B = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}, C = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, D = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$

1. **Inverse $(sI - A)^{-1}$:**
   - $(sI - A) = \begin{bmatrix} s+1 & 1 \\ -6.5 & s \end{bmatrix}$
   - $\det = (s+1)(s) - (1)(-6.5) = s^2 + s + 6.5$
   - $(sI - A)^{-1} = \frac{1}{s^2+s+6.5}\begin{bmatrix} s & -1 \\ 6.5 & s+1 \end{bmatrix}$
2. **Calculate $H(s) = C \cdot (sI - A)^{-1} \cdot B$:**
   - First, $(sI - A)^{-1}B$:
     $$\frac{1}{\det}\begin{bmatrix} s & -1 \\ 6.5 & s+1 \end{bmatrix}\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$$
     $$= \frac{1}{\det}\begin{bmatrix} (s)(1) + (-1)(1) & (s)(1) + (-1)(0) \\ (6.5)(1) + (s+1)(1) & (6.5)(1) + (s+1)(0) \end{bmatrix}$$
     $$= \frac{1}{\det}\begin{bmatrix} s-1 & s \\ s+7.5 & 6.5 \end{bmatrix}$$
   - Next, $C \cdot$ [result]. Since C is the Identity matrix, the result is unchanged.
3. **Final Answer:** $H(s) = \begin{bmatrix} \frac{s-1}{s^2+s+6.5} & \frac{s}{s^2+s+6.5} \\ \frac{s+7.5}{s^2+s+6.5} & \frac{6.5}{s^2+s+6.5} \end{bmatrix}$

Example 4: Transfer Function to State-Space (Phase Variable)
Formula: For a general TF $\frac{Y(s)}{U(s)} = \frac{b_{n-1}s^{n-1} + \dots + b_0}{s^n + a_{n-1}s^{n-1} + \dots + a_0}$, the Phase Variable (or Controllable Canonical) form is:
$$A = \begin{bmatrix} 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \\ -a_0 & -a_1 & \dots & -a_{n-1} \end{bmatrix}, B = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}, C = \begin{bmatrix} b_0 & b_1 & \dots & b_{n-1} \end{bmatrix}, D = [0]$$
Given: $\frac{Y(s)}{U(s)} = \frac{s+3}{s^2+10s+24}$

1. **Identify Coefficients:**
   Denominator: $s^2 + a_1 s + a_0 \Rightarrow a_1 = 10, a_0 = 24$.
   Numerator: $b_1 s + b_0 \Rightarrow b_1 = 1, b_0 = 3$.
2. **Assemble Matrices:**
   - $A = \begin{bmatrix} 0 & 1 \\ -a_0 & -a_1 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -24 & -10 \end{bmatrix}$
   - $B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$
   - $C = \begin{bmatrix} b_0 & b_1 \end{bmatrix} = \begin{bmatrix} 3 & 1 \end{bmatrix}$
   - $D = [0]$

---

2. Kalman Filter (KF)

The Kalman Filter is an optimal recursive estimator that fuses sensor data with a system model to handle noise and uncertainty. It operates on a discrete state-space model. The State Equation is $x_{k+1} = A_k x_k + v_k$, where v_k is the Process Noise (model uncertainty) with covariance Q_k, $v_k \sim \mathcal{N}(0, Q_k)$. The Measurement Equation is $y_k = H_k x_k + w_k$, where w_k is the Measurement Noise (sensor inaccuracy) with covariance R_k, $w_k \sim \mathcal{N}(0, R_k)$.
The filter operates in a two-step algorithm. The Time Update (Prediction) step projects the state and covariance forward: $\hat{x}_k^- = A_{k-1}\hat{x}_{k-1}$ and $P_k^- = A_{k-1}P_{k-1}A_{k-1}^\top + Q_{k-1}$. The Measurement Update (Correction) step incorporates the new measurement y_k. First, the Kalman Gain $K_k = P_k^- H_k^\top (H_k P_k^- H_k^\top + R_k)^{-1}$ is computed, which acts as a blending factor. The state is then corrected using the measurement residual: $\hat{x}_k = \hat{x}_k^- + K_k(y_k - H_k\hat{x}_k^-)$. Finally, the error covariance is corrected: $P_k = (I - K_k H_k)P_k^-$.
Example: Simple Recursive Estimator (KF)

- **Goal:** Estimate a constant value $x_k = c$ (e.g., the true voltage of a battery) from a series of noisy measurements y_k.
- **Model:**
  State Eq: $x_k = 1 \cdot x_{k-1} + v_k$ (We assume the state is constant, so $A = 1$. We can set $Q = 0$ if we are certain, it's constant).
  - Measurement Eq: $y_k = 1 \cdot x_k + w_k$ (We can measure the state directly, so $H = 1$. w_k is the sensor noise with variance R).
- Result: This model simplifies to the intuitive recursive averaging formula, where the Kalman Gain K_k becomes $\frac{1}{k}$:

$$\hat{x}_k = \hat{x}_{k-1} + \frac{1}{k}(y_k - \hat{x}_{k-1})$$

This shows the KF is a generalized form of recursive averaging.

---

3. PGNC & Motion Control

PGNC is a hierarchical framework for autonomous systems. Planning is the high-level, offline task generation, such as finding the optimal order of waypoints (a Travelling Salesman Problem or TSP). Guidance converts this plan into real-time desired states, like calculating a heading to the next waypoint or minimizing cross-track error. Navigation is the state estimation task ("Where am I?"), often using a Kalman Filter to fuse sensor data. Control is the low-level, fast-acting loop that applies actuator commands to follow the guidance system's desired state and reject disturbances.

The controller's job is **Motion Control**. A **PID Controller** calculates the control input $u(t)$ based on the error $e(t) = x_{\text{desired}}(t) - x(t)$ using the formula $u(t) = K_P e(t) + K_I \int e(t)\,dt + K_D \frac{de(t)}{dt}$. Here, K_P (Proportional) reacts to **present** error, K_I (Integral) corrects for **past** error to remove steady-state bias, and K_D (Derivative) reacts to **future** error to dampen oscillations. The guidance system's job is to provide x_{desired}, often as a **Line-of-Sight** heading command. This command ψ_{cmd} to a waypoint $(x_{\text{wp}}, y_{\text{wp}})$ from a current position (x, y) is calculated using $\psi_{\text{cmd}} = \text{arctan2}(y_{\text{wp}} - y, x_{\text{wp}} - x)$.

---

4. Unconstrained Optimisation

Optimisation is the process of finding the best solution x by minimizing or maximizing a Cost Function $J(x)$. There are several methods for solving this. A Brute Force search computes $J(x)$ over a large grid, which is simple but slow. A Gradient-Based search starts at a guess $\hat{x}_0$ and iteratively moves "downhill" using the gradient: $\hat{x}_{k+1} = \hat{x}_k - \gamma \frac{dJ}{dx}(\hat{x}_k)$; this can get stuck in local minima. The Calculus-Based Approach is analytical. First, find all Stationary Points by solving $\frac{dJ}{dx} = 0$. Second, classify each point using the 2nd derivative test: $\frac{d^2J}{dx^2} > 0$ is a Local Minimum, $\frac{d^2J}{dx^2} < 0$ is a Local Maximum, and $\frac{d^2J}{dx^2} = 0$ is a Saddle Point. The Global Minimum is the stationary point with the lowest $J(x)$ value.

Example 1: Parameter Estimation (Least Squares)

To find model parameters θ that best fit measured data y_k^d, we can minimize the Sum of Squared Error (SSE) cost function. The error is $e_k = y_k^\theta - y_k^d$, and the cost function is $J(\theta) = \sum_k (e_k)^2 = \sum_k (y_k^\theta - y_k^d)^2$.

Example 2: System Design (Paddock Area)

- **Goal:** Maximize Area $J = L \cdot W$ given fixed fence length $T = 2L + 2W$.
- **Constraint:** $W = (T - 2L)/2$.
- **Cost Function (unconstrained):** $J(L) = L \cdot \left(\frac{T-2L}{2}\right) = \frac{T}{2}L - L^2$.
- **Solve:** $\frac{dJ}{dL} = \frac{T}{2} - 2L = 0 \Longrightarrow L = T/4$. (A square).

---

5. Path Planning & Decision Making

Path Planning Algorithms generate a route from a start to a goal. Potential Field methods model the robot as a particle in a field where obstacles create repulsive forces and the goal creates an attractive force; this is simple but can get stuck in local minima. Dijkstra's Algorithm is a discrete (grid) search that finds the optimal path from the start to all other nodes but can be slow. A* (A-star) Search is also a grid search but is faster than Dijkstra because it uses a heuristic $h(x)$ (an estimate of cost-to-go) to guide its search, minimizing $f(x) = g(x) + h(x)$, where $g(x)$ is the known cost from the start. Sampling-based methods include PRM (Probabilistic Roadmap), which randomly samples many nodes, connects them into a "roadmap," and then searches the map with A*, and RRT (Rapidly Exploring Random Tree), which grows a tree from the start by extending the nearest node toward random points, efficiently exploring space but not guaranteeing optimality.

Decision Making involves choosing an action. A **Markov Decision Process (MDP)** is a framework for modeling decisions where outcomes are partly random, defined by States (S), Actions (A), Transition Probabilities $T(s'|s, a)$, and Rewards (R). For problems with a *finite* set of alternatives, **MADM (Multi-Attribute Decision Making)** is used. A common MADM method is the **Weighted Sum Model (WSM)**, $A_i^{score} = \sum_{j=1}^{n} w_j\, a_{ij}$, where w_j is the weight of criterion j and a_{ij} is the performance value of alternative i for that criterion.

Example: Emergency Landing (MADM)

- **Goal:** Choose the best of three landing sites.
- **Criteria & Weights:** area (w=0.3), length (w=0.3), slope (w=0.1), distance (w=0.3)
- **Alternatives & Values:**
  Site 1: {area: 25, length: 30, slope: 20, distance: 30}
  Site 2: {area: 10, length: 20, slope: 5, distance: 10}
  Site 3: {area: 30, length: 10, slope: 30, distance: 10}
- **Calculation:**
  Site 1 score = (0.3 · 25) + (0.3 · 30) + (0.1 · 20) + (0.3 · 30) = 7.5 + 9 + 2 + 9 = **27.5**
  Site 2 score = (0.3 · 10) + (0.3 · 20) + (0.1 · 5) + (0.3 · 10) = 3 + 6 + 0.5 + 3 = **12.5**
  Site 3 score = (0.3 · 30) + (0.3 · 10) + (0.1 · 30) + (0.3 · 10) = 9 + 3 + 3 + 3 = **18.0**
- **Decision:** Site 1 has the highest score and is the best alternative.

---

6. Sensor Principles

An IMU (Inertial Measurement Unit) contains gyroscopes (measuring angular rate) and accelerometers (measuring specific force). An INS (Inertial Navigation System) is an IMU combined with a "mechanisation" algorithm that integrates the raw data to estimate attitude, velocity, and position. The most common configuration is Strapdown, where sensors are fixed to the vehicle body. All INS systems drift over time due to the integration of sensor errors, primarily Bias (a constant offset) and Angle Random Walk (noise). These systems rely on clear Coordinate Frames, such as the fixed World/Global Frame (GCS) and the moving Body Frame (B).

GPS (Global Positioning System) works on the principle of **trilateration**, measuring the signal travel time from satellites to find its distance (pseudorange). A minimum of **4 satellites** is required to solve for the four unknowns: position (X, Y, Z) and the receiver's **clock error**. Major error sources include **atmospheric delays** (Ionosphere, Troposphere) and **multipath** (signal reflections). The quality of the satellite geometry is measured by **DOP (Dilution of Precision)**; a low DOP value is good (satellites spread out), while a high DOP is bad (satellites clustered). GPS **Vertical Accuracy** is typically ~3 times worse than horizontal accuracy.

Laser Scanners (Lidar) commonly use **Pulsed Time of Flight (ToF)**, which measures the round-trip time (t) for a laser pulse to return. The distance is calculated using the speed of light, c, with the formula range $= \frac{c \times t}{2}$.

Cameras are modeled using the **Pinhole Camera Model**, which idealizes the lens as a single point (the optical centre), historically known as a **Camera Obscura**. Real lenses suffer from **Lens Distortion** that must be calibrated. **Radial Distortion** causes straight lines to curve (**Barrel** distortion curves outward, **Pincushion** distortion curves inward). **Tangential Distortion** is due to a misaligned lens and sensor. Camera data rates can be very high.

- **Example: Bandwidth Calculation**

  Data Rate $=$ (Width $\times$ Height $\times$ Channels $\times$ Bits per Channel) $\times$ FPS
  - $1024 \times 768 \times 3$ (RGB) $\times$ 8 bits $\times$ 20 fps $=$ 377 Mbps (47.2 MB/s)