

TRIMESTER 1

EXAM COMPANION

Dec 5, 6, 7 — 2026 | 73 days from Sep 23 | IIT Guwahati MTech RAI

This guide explains what to learn in each subject, how deeply, and what kind of exam questions to expect. Each section gives you specific topics with explanation of WHY the exam tests them, worked examples showing the reasoning process, and practice questions with the approach you should use. Read this alongside your lecture notes — it fills in the conceptual understanding that notes alone miss.

Exam strategy: 3.5 to 4.5 hours per day on exam prep. Keep research preparation (PDF 1) running in parallel — they reinforce each other. After Dec 7, redirect exam hours to T2 coursework and T1 projects.

HOW TO PREPARE: THE METHOD

Before listing topics, this is how to actually learn them. Most students read their notes, feel familiar with the material, and then fail to solve unseen problems in the exam. Familiarity is not mastery. Mastery means you can reconstruct the solution to any problem in the topic from first principles, without having seen that exact problem before.

The specific technique: for every topic, find the deepest question you do not yet understand. Do not move on until you can answer it. For transforms: the deepest question is not "what is a rotation matrix" but "if I have T_{AB} , what does the translation column mean physically, and how do I compute T_{BA} from it without matrix inversion." For the Kalman filter: not "what are the Kalman equations" but "why does the covariance decrease during the update step, and by how much should it decrease if my sensor is perfect."

Worked examples: after you understand a topic, solve the worked examples in this guide with the solution covered. Write your full solution. Then compare. If your reasoning is different from the shown solution, understand why — do not assume your answer is wrong. There can be multiple valid approaches. But if your final numbers differ, you have an error somewhere.

Weekly review: Every Sunday, pick three topics from the previous week. Without looking at your notes, write the key equation, state the key assumption, give one example where the method fails, and describe one application. If you cannot do this for a topic, it is not yet learned.

The Exam Tests Understanding, Not Memorisation

The exams for SSO5201, SSO5202, SSO5203 are not fact-recall tests. They test whether you can apply concepts to novel scenarios. "Derive the Jacobian for a 3R planar arm" requires the same procedure as "derive the Jacobian for a 2R arm" but on a new instance. "Given this new probabilistic model, write the forward algorithm" requires the same reasoning as the HMM example but on different numbers. If you understand why the method works, you can handle any variation. If you memorised the steps, a slight variation will block you.

SSO5201: FUNDAMENTALS OF ROBOTICS

What to Learn, How Deeply, and Why

lead

This course is about describing and controlling the geometry of mechanical systems. Every topic connects to the same thread: where is the end-effector, how did it get there, how do we control where it goes next.

Topic 1: Coordinate Frames and Spatial Transformations

This is the single most important topic in the course. Every subsequent topic — forward kinematics, inverse kinematics, Jacobians, dynamics — relies on getting transforms right. A coordinate frame has an origin and three orthogonal axes. A point P expressed in frame A has coordinates $[x, y, z]_A$. The same point in frame B has coordinates $[x, y, z]_B$. The transformation from B to A is a 4×4 homogeneous matrix T_{AB} .

The rotation part R (upper 3×3 of T) satisfies $R^T = R^{-1}$ and $\det(R) = +1$. The translation part t (upper 3×1) is the position of frame B 's origin, expressed in frame A . Composing transforms: $T_{AC} = T_{AB} * T_{BC}$ (read right to left: first express in C , then in B , then in A). Inverting: $T_{BA} = [[R^T, -R^T t], [0, 0, 0, 1]]$.

What the exam tests: Given T_{AB} and T_{BC} , compute T_{AC} . Given T_{AB} , compute T_{BA} . Transform a point expressed in C to frame A . The mistakes to avoid: multiplying in the wrong order, and thinking that T_{BA} is just T_{AB} with the translation negated (it is $-R^T * t$, not just $-t$).

WORKED EXAMPLE: Frame Composition

Problem: Frame A is the world. Frame B has its origin at $[1, 0, 0]_A$ and is rotated 90 degrees around the z -axis relative to A . Frame C has its origin at $[0, 1, 0]_B$ and is not rotated relative to B . Where is the origin of C , expressed in frame A ?

Solution: T_{AB} : $R_{AB} = [[0, -1, 0], [1, 0, 0], [0, 0, 1]]$ (90-degree z rotation), $t_{AB} = [1, 0, 0]$. T_{BC} : $R_{BC} = I$, $t_{BC} = [0, 1, 0]$. $T_{AC} = T_{AB} * T_{BC}$. The origin of C in A is: $R_{AB} * t_{BC} + t_{AB} = [[0, -1, 0], [1, 0, 0], [0, 0, 1]] * [0, 1, 0] + [1, 0, 0] = [-1, 0, 0] + [1, 0, 0] = [0, 0, 0]$. So C 's origin coincides with A 's origin. Verify by checking: the rotation $T_{AC} = R_{AB} * I = R_{AB}$, and the translation $= R_{AB} * [0, 1, 0] + [1, 0, 0] = [-1, 0, 0] + [1, 0, 0] = [0, 0, 0]$. Correct.

Topic 2: Denavit-Hartenberg (DH) Convention

DH is a systematic way to assign coordinate frames to robot links. Each joint is described by four parameters: (a, α, d, θ) where θ is the joint variable for revolute joints and d is the joint variable for prismatic joints. The DH matrix for one link is: $T = \text{Rot}_z(\theta) * \text{Trans}_z(d) *$

$\text{Trans}_x(a) * \text{Rot}_x(\alpha)$. Multiplying all link matrices gives T_{0n} — the transform from the base frame to the end-effector frame.

What the exam tests: Given a robot description (link lengths, joint types, joint limits), build the DH table and compute T_{0n} at a specific configuration. The mistakes to avoid: using the wrong convention (your instructor uses either standard or modified DH — know which one), and misidentifying which parameter is the joint variable.

How to learn this: Build the DH table for a 2-link planar arm by hand. Verify by computing T_{02} at three configurations and checking against the geometric FK formula ($x = l_1 \cos(q_1) + l_2 \cos(q_1 + q_2)$). Only after you can do the 2-link case without errors, move to the 3-link case.

Topic 3: Forward and Inverse Kinematics

Forward kinematics (FK): given joint angles q , compute end-effector pose. This is just evaluating $T_{0n}(q)$. For a 2-link planar arm: $x = l_1 \cos(q_1) + l_2 \cos(q_1 + q_2)$, $y = l_1 \sin(q_1) + l_2 \sin(q_1 + q_2)$. Straightforward once you have the DH matrices.

Inverse kinematics (IK): given end-effector pose, find joint angles q . For the 2-link planar arm, the geometric approach gives: $\cos(q_2) = (x^2 + y^2 - l_1^2 - l_2^2) / (2 * l_1 * l_2)$. This has two solutions (elbow up and elbow down) since $\cos(q_2) = c_2$ gives $q_2 = \arccos(c_2)$ and $q_2 = -\arccos(c_2)$. Once q_2 is known, $q_1 = \text{atan2}(y, x) - \text{atan2}(l_2 \sin(q_2), l_1 + l_2 \cos(q_2))$.

What the exam tests: Find both IK solutions for a given target position. Determine which solutions are within the joint limits. Identify positions that are unreachable (outside the workspace). The mistake to avoid: reporting only one IK solution when two exist.

WORKED EXAMPLE: Inverse Kinematics

Problem: A 2-link arm has $l_1 = l_2 = 1\text{m}$. Find all joint angle combinations that reach the point (1.0, 0.0).

Solution: $\cos(q_2) = (1^2 + 0^2 - 1^2 - 1^2) / (2 * 1 * 1) = -1/2$. So $q_2 = \arccos(-1/2) = 120$ degrees OR $q_2 = -120$ degrees. Solution 1 (elbow up, $q_2 = 120$): $\sin(q_2) = \sin(120) = 0.866$. $q_1 = \text{atan2}(0, 1) - \text{atan2}(1 * 0.866, 1 + 1 * (-0.5)) = 0 - \text{atan2}(0.866, 0.5) = 0 - 60 = -60$ degrees. Verify: $x = \cos(-60) + \cos(-60 + 120) = 0.5 + \cos(60) = 0.5 + 0.5 = 1$. $y = \sin(-60) + \sin(60) = -0.866 + 0.866 = 0$. Correct. Solution 2 (elbow down, $q_2 = -120$): $q_1 = 0 - \text{atan2}(-0.866, 0.5) = 0 - (-60) = 60$ degrees. Verify: $x = \cos(60) + \cos(60 - 120) = 0.5 + \cos(-60) = 0.5 + 0.5 = 1$. $y = \sin(60) + \sin(-60) = 0.866 - 0.866 = 0$. Correct. Both solutions valid.

Topic 4: The Jacobian Matrix

The Jacobian J relates infinitesimal joint velocities $\dot{q}$ to end-effector linear velocity v (and angular velocity ω for the full $6 \times n$ Jacobian): $v = J(q) * \dot{q}$. For a 2-link planar arm, the position Jacobian (2×2) is: $J = \begin{bmatrix} -l_1 \sin(q_1) - l_2 \sin(q_1 + q_2) & -l_2 \sin(q_1 + q_2) \\ l_1 \cos(q_1) + l_2 \cos(q_1 + q_2) & l_2 \cos(q_1 + q_2) \end{bmatrix}$. This is obtained by differentiating the FK equations with respect to the joint angles.

A singularity occurs when J loses rank. For the 2-link arm, singularities occur when the arm is fully extended ($q_2 = 0$) or fully folded ($q_2 = 180$). At these configurations, the end-effector cannot move in

one direction without infinite joint velocity.

Static force mapping: the joint torques required to support an end-effector force F are $\tau = J^T * F$. This is the transpose Jacobian, not the inverse.

WORKED EXAMPLE: Jacobian and Singularity

Problem: For a 2-link arm with $l_1=l_2=1$, at configuration $q_1=0$, $q_2=0$ (arm pointing straight along x-axis), compute J and determine the singular direction.

Solution: $J = [-\sin(0)-\sin(0), -\sin(0)], [\cos(0)+\cos(0), \cos(0)] = [[0, 0], [2, 1]]$. The first column is $[0, 2]$, second is $[0, 1]$. Both columns point in the same direction (y-axis). The matrix has rank 1 — it is singular. The null space vector is $[-1, 2]$ (can be verified: $J*[-1, 2] = [0*(-1)+0*2, 2*(-1)+1*2] = [0, 0]$). This means moving joint 1 backward by 1 unit and joint 2 forward by 2 units produces zero end-effector velocity — these joint motions cancel. The lost direction is x: the end-effector cannot be moved in x without infinite joint velocity at this configuration.

Topic 5: Robot Dynamics

Dynamics describes the relationship between forces/torques and motion. The equation of motion for a robot is: $M(q)*\ddot{q} + C(q, \dot{q})*\dot{q} + G(q) = \tau$, where M is the inertia matrix, C contains Coriolis and centrifugal terms, G is the gravity vector, and τ is the joint torque. For the Lagrangian approach: define kinetic energy T and potential energy V , then $\tau_i = \frac{d}{dt}(\frac{dT}{dq_{\dot{i}}}) - \frac{dT}{dq_i} + \frac{dV}{dq_i}$.

Start with the pendulum. A simple pendulum has one degree of freedom (θ). Kinetic energy: $T = \frac{1}{2}*m*l^2*\dot{\theta}^2$. Potential energy: $V = m*g*l*(1-\cos(\theta))$ (taking the equilibrium as zero). The equation of motion: $m*l^2*\ddot{\theta} + m*g*l*\sin(\theta) = \tau$. This is the foundation. A 2-link arm is two pendulums — more algebra but the same approach.

Topic 6: Path Planning and Control

Path vs trajectory: A path is a geometric curve in joint space or task space (it says where to go, not when). A trajectory adds a time parametrisation to a path (it says when to be where). A cubic polynomial trajectory $q(t) = a_0 + a_1*t + a_2*t^2 + a_3*t^3$ satisfies four constraints: initial position, initial velocity, final position, final velocity. Solving the 4×4 linear system gives the coefficients.

PID control for robots: A joint-space PD controller applies torque $\tau = K_p*(q_{des} - q) + K_d*(\dot{q}_{des} - \dot{q})$. With a gravity compensation term, this becomes $\tau = K_p*e + K_d*\dot{e} + G(q)$, which achieves zero steady-state error for constant references. Without gravity compensation, the steady-state error is non-zero. PID adds integral action to eliminate this error, but integral windup must be handled.

SSO5202: ARTIFICIAL INTELLIGENCE

Search, Uncertainty, and Decision-Making

This course covers two fundamentally different types of AI: search-based methods (where the world is known and the problem is finding a solution) and probabilistic methods (where the world is uncertain and the problem is inferring and deciding under that uncertainty). You need both.

Topic 1: Problem Formulation and Uninformed Search

Before choosing a search algorithm, you must formulate the problem. A search problem has: a state space (the set of all possible situations), an initial state, a goal test (which states are solutions), a set of actions, a transition model (what each action does), and an action cost. Formulating the problem well is as important as choosing the right algorithm.

BFS uses a FIFO queue and explores the shallowest nodes first. It is complete (finds a solution if one exists) and optimal when all edge costs are equal. Time and space complexity: $O(b^d)$ where b is branching factor and d is the depth of the shallowest solution.

DFS uses a LIFO stack and explores the deepest nodes first. It is not optimal. Space complexity is only $O(b \cdot m)$ where m is the maximum depth. This is its only advantage over BFS.

Uniform-cost search (UCS) expands the node with the lowest path cost first, using a priority queue. It is complete and optimal for any positive edge costs. It is equivalent to BFS when all edge costs are equal.

What the exam tests: Trace the execution of BFS and UCS on the same graph. Explain why they find different paths when edge costs are unequal. Compute the optimal path cost.

WORKED EXAMPLE: BFS vs UCS

Problem: Graph: nodes A,B,C,D,E. Edges: A-B(cost 1), A-C(cost 5), B-D(cost 1), C-D(cost 1), D-E(cost 1). Start=A, Goal=E. Find the path found by BFS and by UCS.

Solution: BFS (ignores costs, finds fewest edges): A->B->C->D->E? No, BFS frontier: start with A. Expand A: frontier=[B,C]. Expand B: frontier=[C,D]. Expand C: frontier=[D,D] (mark C visited). Expand D (first occurrence): frontier=[D,E]. Expand second D: skip (visited). Expand E: goal. Path: A->B->D->E (3 edges, cost=1+1+1=3). UCS (minimises cost): expand A (cost 0). Add B(cost 1), C(cost 5). Expand B (lowest cost=1). Add D (cost=1+1=2). Expand D (cost=2). Add E(cost=2+1=3), also can reach C via D(cost=3), but C already in open. Expand E(cost=3): goal. Path: A->B->D->E (cost=3). In this case both agree. Change C-D to cost 0.1: BFS still gives A->B->D->E. UCS now gives A->C->D->E (cost=5+0.1+1=6.1 > 3), so UCS still gives A->B->D->E. For different costs, UCS considers the total path cost, not the number of steps.

Topic 2: Heuristic Search and A*

A* uses the evaluation function $f(n) = g(n) + h(n)$, where $g(n)$ is the actual cost from start to n , and $h(n)$ is the estimated cost from n to the goal. A* is optimal when h is admissible: $h(n)$ never overestimates the true cost to the goal. It is also complete when the search space is finite or costs are bounded below by a positive constant.

A consistent (or monotone) heuristic satisfies: $h(n) \leq c(n, n') + h(n')$ for every edge (n, n') . Consistency implies admissibility. With a consistent heuristic, A* never needs to reopen a node.

What the exam tests: Show that a specific heuristic is admissible (by proving it cannot exceed the true cost). Trace A* on a graph. Explain what happens when h is not admissible.

Topic 3: Adversarial Search

In a two-player zero-sum game, the minimax algorithm assumes both players play optimally: MAX maximises utility, MIN minimises it. The minimax value of a node is: for MAX nodes, the maximum of the children's values; for MIN nodes, the minimum of the children's values; for leaf nodes, the utility.

Alpha-beta pruning eliminates branches that cannot influence the final decision. Alpha is the best value MAX can guarantee so far; beta is the best value MIN can guarantee so far. At a MAX node, prune if the node's value becomes $\geq \beta$. At a MIN node, prune if the node's value becomes $\leq \alpha$. Alpha-beta does not change the result — it just avoids computing parts of the tree that cannot affect it.

WORKED EXAMPLE: Minimax with Alpha-Beta

Problem: A game tree: root is MAX. Two MIN children (L, R). L has leaf values [3, 5]. R has leaf values [2, 9]. What is the minimax value of the root? Which branches can be pruned with alpha-beta?

Solution: Bottom up: $L = \min(3, 5) = 3$. $R = \min(2, 9) = 2$. $\text{Root} = \max(3, 2) = 3$. Alpha-beta: Start $\alpha = -\infty$, $\beta = +\infty$. Expand left MIN node L: first child=3. $\beta = \min(\infty, 3) = 3$. α at root = $\max(-\infty, 3) = 3$. Second child=5. $\beta = \min(3, 5) = 3$. No pruning ($3 \geq \alpha = -\infty$ at L). Left subtree done, $\alpha = 3$. Now expand right MIN node R: first child=2. $\beta = \min(\infty, 2) = 2$. Is $\beta(2) \leq \alpha(3)$? YES. Prune R's second child (value 9 cannot help). $\text{Root} = \max(3, 2) = 3$. The second child of R (value 9) was pruned.

Topic 4: Temporal Probability and Hidden Markov Models

A Hidden Markov Model (HMM) has: a hidden state X_t (not directly observable), an observation E_t (what you can see), a transition model $P(X_t | X_{t-1})$ (how the state changes), and an observation model $P(E_t | X_t)$ (what you observe given the state). The three key inference problems are:

Filtering: compute $P(X_t | e_{1:t})$ — the current state distribution given all observations so far. This is the forward algorithm. $P(X_t | e_{1:t})$ is proportional to $P(e_t | X_t) * \sum_{x_{t-1}} [P(X_t | x_{t-1}) * P(x_{t-1} | e_{1:t-1})]$.

Smoothing: compute $P(X_t | e_{1:T})$ — the state distribution at time t given ALL observations including future ones. This is more accurate than filtering because it uses future evidence.

Viterbi (most likely sequence): find the single most likely state sequence $X_{1:T}$ given all observations. This is different from filtering: the sequence of most likely states (from filtering) is not necessarily the most likely sequence.

WORKED EXAMPLE: HMM Filtering

Problem: Umbrella world: two states (Rain=R, NoRain=N). Transition: $P(R|R)=0.7$, $P(N|R)=0.3$, $P(R|N)=0.3$, $P(N|N)=0.7$. Observation (umbrella=U): $P(U|R)=0.9$, $P(U|N)=0.2$. Initial: $P(R_1)=0.5$, $P(N_1)=0.5$. Observation at $t=1$: umbrella seen. Compute $P(R_1|U_1)$.

Solution: $P(X_1|e_1)$ proportional to $P(e_1|X_1) * P(X_1)$. $P(R,U_1)=P(U|R)*P(R)=0.9*0.5=0.45$. $P(N,U_1)=P(U|N)*P(N)=0.2*0.5=0.10$. Normalise: $P(R|U_1)=0.45/(0.45+0.10)=0.45/0.55=0.818$. $P(N|U_1)=0.10/0.55=0.182$. Interpretation: after seeing umbrella on day 1, probability of rain is 81.8% (up from 50% prior). Now compute $P(X_2|U_1,U_2)$ where U_2 is also seen: Predict: $P(R_2|U_1)=0.7*0.818+0.3*0.182=0.573+0.055=0.627$. $P(N_2|U_1)=0.3*0.818+0.7*0.182=0.245+0.127=0.373$. Update: $P(R_2,U_2|U_1)=0.9*0.627=0.564$. $P(N_2,U_2|U_1)=0.2*0.373=0.075$. Normalise: $P(R_2|U_1,U_2)=0.564/0.639=0.883$.

Topic 5: The Kalman Filter (Detailed)

The Kalman filter is the optimal filtering algorithm when: the dynamics are linear, the observations are linear functions of the state, and all noise is Gaussian. Under these conditions, the posterior $P(X_t | e_{1:t})$ is always Gaussian, and the Kalman filter computes its mean and covariance exactly.

Predict step: $\hat{x}_{prior} = F*\hat{x}_{post} + B*u$. $P_{prior} = F*P_{post}*F^T + Q$. The mean propagates through the linear dynamics. The covariance grows because of process noise Q .

Update step: Innovation: $y = z - H*\hat{x}_{prior}$. Innovation covariance: $S = H*P_{prior}*H^T + R$. Kalman gain: $K = P_{prior}*H^T*S^{-1}$. Updated estimate: $\hat{x}_{post} = \hat{x}_{prior} + K*y$. Updated covariance: $P_{post} = (I - K*H)*P_{prior}$.

Intuition for K: If R is small (sensor is precise), K is large, and we trust the measurement. If P_{prior} is small (we are already confident in our estimate), K is small, and we trust our prediction. K balances the two sources of information.

WORKED EXAMPLE: 1D Kalman Full Cycle

Problem: State: position x (scalar). Prior: $\hat{x}_{hat}=5.0$, $P=4.0$. System: no motion ($F=1$, $Q=0.5$). Sensor: direct position, $H=1$, $R=1.0$. Measurement: $z=6.0$. Run one full predict-update cycle.

Solution: Predict: $\hat{x}_{prior}=1*5.0+0=5.0$. $P_{prior}=1*4.0*1+0.5=4.5$. Update: Innovation $y=6.0-1*5.0=1.0$. Innovation cov $S=1*4.5*1+1.0=5.5$. Kalman gain $K=4.5*1/5.5=0.818$. Updated estimate $\hat{x}_{hat}=5.0+0.818*1.0=5.818$. Updated cov $P=(1-0.818*1)*4.5=0.182*4.5=0.818$. Interpretation: we moved 81.8% of the way toward the measurement because our prior uncertainty ($P=4.5$) was much larger than sensor noise ($R=1.0$). The posterior uncertainty $P=0.818$ is less than both the prior (4.5) and the measurement noise (1.0) — combining two sources of information always reduces uncertainty.

Topic 6: MDPs and Decision-Making

A Markov Decision Process has: a set of states S , actions A , transition probabilities $P(s'|s,a)$, rewards $R(s,a)$, and a discount factor γ . The goal is to find a policy $\pi(s)$ that maximises the expected discounted cumulative reward.

The Bellman equation: $V^*(s) = \max_a [R(s,a) + \gamma \sum_{s'} P(s'|s,a) * V^*(s')]$. Value iteration starts with $V_0(s)=0$ for all s and iteratively applies the Bellman equation until convergence. The optimal policy: $\pi^*(s) = \operatorname{argmax}_a [R(s,a) + \gamma \sum_{s'} P(s'|s,a) * V^*(s')]$.

WORKED EXAMPLE: Value Iteration

Problem: 3 states: A (start), B, G (goal). Actions: go_right (from A to B, from B to G). Deterministic. $R(B, \text{go_right})=0$, $R(A, \text{go_right})=0$, $R(\text{reach } G)=+10$. $\gamma=0.9$. Run value iteration.

Solution: Iteration 0: $V(A)=V(B)=V(G)=0$. Iteration 1: $V(G)=10$ (absorbing). $V(B)=0+0.9*V(G)=0+9=9$. $V(A)=0+0.9*V(B)=0+8.1=8.1$. Iteration 2: $V(G)=10$ (unchanged). $V(B)=0+0.9*10=9$ (unchanged). $V(A)=0+0.9*9=8.1$ (unchanged). Converged. Policy: at A, go right (to B). At B, go right (to G). This is the only rational policy here.

SSO5203: ROBOT SENSING AND VISION

From Pixels to Calibrated Measurements

This course explains how a robot perceives the world through sensors. The thread: a sensor converts a physical quantity to a digital signal. That signal has noise, quantisation error, and limited resolution. Calibration converts raw sensor data to physically meaningful quantities. Higher-level algorithms (optical flow, SLAM) use calibrated data to infer the state of the world.

Topic 1: The Image as a Data Structure

A grayscale image is a 2D array of integers, where each element (pixel) stores an intensity value (typically 0-255 for 8-bit images, 0-65535 for 16-bit). The pixel at position $(\text{row}, \text{col}) = (v, u)$ has intensity $I(v, u)$. Note the convention: v indexes rows (vertical), u indexes columns (horizontal). This is the opposite of (x, y) in standard mathematics, and getting this wrong causes transposition errors.

Image operations: convolution with a kernel computes a weighted average of neighbouring pixels. Gaussian smoothing uses a Gaussian-shaped kernel to reduce noise while preserving edges. Edge detection (Sobel, Canny) computes the gradient of the image intensity, highlighting regions of rapid change. Morphological operations (erosion, dilation) modify the shape of binary regions.

Topic 2: The Camera Model

The pinhole camera model projects a 3D point $P = [X, Y, Z]$ in camera coordinates to a 2D pixel $[u, v]$ as: $u = f_x * X/Z + c_x$, $v = f_y * Y/Z + c_y$. In matrix form: $s * [u, v, 1]^T = K * [X, Y, Z]^T$ where $K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$ is the camera intrinsic matrix. The focal lengths f_x, f_y (in pixels) determine the field of view. The principal point (c_x, c_y) is the pixel where the optical axis intersects the image plane (approximately the image centre).

Lens distortion causes straight lines in the world to appear curved in the image. The dominant distortion is radial: a point at radius r from the principal point is displaced to radius $r * (1 + k_1 * r^2 + k_2 * r^4)$. Distortion must be corrected before using the pinhole model for precise measurements.

Camera calibration estimates K and the distortion coefficients from a set of images of a known calibration target (usually a checkerboard). The calibration procedure: (1) print a checkerboard with known square size, (2) take 10-20 images from different angles, (3) detect corner points in each image (sub-pixel precision), (4) run `cv2.calibrateCamera` which solves a nonlinear optimisation problem to find K and distortion that minimise reprojection error.

WORKED EXAMPLE: Pinhole Projection

Problem: Camera: $f_x=f_y=800$, $c_x=640$, $c_y=480$ (1280x960 image). A point is at world position [0.1m, -0.05m, 2.0m] in the camera frame. What pixel does it project to? What if the point moves 1m farther away to [0.1, -0.05, 3.0]?

Solution: At $Z=2.0\text{m}$: $u = 800 \cdot (0.1/2.0) + 640 = 800 \cdot 0.05 + 640 = 40 + 640 = 680$. $v = 800 \cdot (-0.05/2.0) + 480 = 800 \cdot (-0.025) + 480 = -20 + 480 = 460$. Pixel: (680, 460). At $Z=3.0\text{m}$: $u = 800 \cdot (0.1/3.0) + 640 = 800 \cdot 0.0333 + 640 = 26.7 + 640 = 666.7$. $v = 800 \cdot (-0.05/3.0) + 480 = -13.3 + 480 = 466.7$. The point appears closer to the principal point (680->667, 460->467) but cannot recover the depth from the projection alone. Scale ambiguity: any point on the ray through (680, 460) projects to the same pixel.

Topic 3: Feature Detection and Matching

A feature is a distinctive point in an image that can be reliably detected across multiple images. The requirements for a good feature: (1) repeatability — it can be found in another image of the same scene, (2) distinctiveness — its descriptor is different from descriptors of other features, (3) locality — it is affected only by a small neighbourhood. SIFT (Scale Invariant Feature Transform) satisfies all three requirements but is slower. ORB (Oriented FAST and Rotated BRIEF) is faster and good enough for most robotics applications.

Matching features between two images: for each feature in image 1, find the feature in image 2 with the smallest descriptor distance (Euclidean distance for SIFT, Hamming distance for ORB). Use the ratio test (Lowe's test): accept a match only if the distance to the best match is less than 0.75 times the distance to the second-best match. This eliminates ambiguous matches.

Topic 4: RANSAC and Robust Estimation

Feature matching produces many correct matches (inliers) and some incorrect matches (outliers). RANSAC (Random Sample Consensus) is an algorithm for fitting a model to data when outliers are present: (1) randomly select the minimum number of points needed to fit the model (e.g., 4 point pairs for homography), (2) fit the model to those points, (3) count how many other points are consistent with the model (inliers), (4) repeat N times and keep the model with the most inliers. The number of iterations N needed to ensure at least one all-inlier sample with probability p is $N = \log(1-p) / \log(1 - e^n)$, where e is the inlier ratio and n is the sample size.

WORKED EXAMPLE: RANSAC Iterations

Problem: You have 200 feature matches. You estimate that 60% are inliers ($e=0.6$). You want to estimate a homography (requires $n=4$ points). How many RANSAC iterations ensure a 99% probability of finding an all-inlier sample?

Solution: Probability that a single random sample of 4 points is all-inliers: $e^n = 0.6^4 = 0.1296$. Probability that at least one of N samples is all-inliers: $1 - (1 - 0.1296)^N = 0.99$. $(1 - 0.1296)^N = 0.01$. $N = \log(0.01) / \log(0.8704) = (-4.605) / (-0.1388) = 33.2$. So 34 iterations are sufficient. If the inlier ratio drops to 30% ($e=0.3$): $e^4 = 0.0081$. $N = \log(0.01)/\log(0.9919) = 566$ iterations. This shows how sensitive RANSAC is to the inlier ratio.

Topic 5: Optical Flow

Optical flow estimates the apparent motion of each pixel between consecutive frames. The brightness constancy assumption: $I(x, y, t) = I(x+dx, y+dy, t+dt)$, meaning the intensity of a pixel stays the same as it moves. Taylor expanding the right side and dividing by dt gives the optical flow constraint equation: $I_x*u + I_y*v + I_t = 0$, where $u=dx/dt$, $v=dy/dt$ is the flow vector and I_x , I_y , I_t are the image derivatives.

This single equation has two unknowns (u , v): the aperture problem. Lucas-Kanade resolves this by assuming flow is constant in a small window W around each pixel. This gives a least-squares system over all pixels in W : $\sum (I_x*u + I_y*v + I_t)^2 = 0$. The solution is: $[u, v]^T = -A^{-1}*b$ where $A = \begin{bmatrix} \sum I_x^2 & \sum I_x*I_y \\ \sum I_x*I_y & \sum I_y^2 \end{bmatrix}$ and $b = [\sum I_x*I_t, \sum I_y*I_t]$. The eigenvalues of A determine tracking quality: both should be large.

Topic 6: SLAM

Simultaneous Localisation and Mapping (SLAM) is the problem of a robot building a map of an unknown environment while simultaneously tracking its location within that map. The core challenge: to localise, you need a map; to build a map, you need to know your location. SLAM methods break this chicken-and-egg problem using probabilistic inference.

Feature-based SLAM (e.g., ORB-SLAM): detect features in each frame, match them to a local map, estimate camera pose, add new features to the map. The key components are tracking (estimate pose from existing map features), local mapping (refine the local map using new observations), and loop closure (detect when the robot has returned to a previously visited location and correct the accumulated drift). Drift is the unavoidable accumulation of small pose estimation errors over time. Loop closure corrects it by recognising a previously seen location and applying a large correction to the entire trajectory.

WEEK-BY-WEEK STUDY PLAN

Sep 23 to Dec 7, 2026

This plan allocates your exam prep time (3.5-4.5 hrs/day). Research preparation (PDF 1) runs in the remaining hours. The plan assumes you attend all lectures. If you miss one, add a catch-up session.

Weeks 1-2: Sep 23 - Oct 4

Theme: Foundations and first pass

What to study: Spatial transforms (frames, rotation matrices, homogeneous transforms). BFS/DFS/UCS from scratch in Python. Image as array, pixel coordinates, basic OpenCV operations. Run the Week 1 diagnostic: try one problem from each topic without notes.

Watch for

If you cannot invert a homogeneous transform without looking it up, stop and fix that before continuing. Everything in robotics builds on transforms.

Weeks 3-4: Oct 5 - Oct 18

Theme: Core kinematics and probability

What to study: DH convention and forward kinematics for 2R and 3R arms. Geometric inverse kinematics (2R). A* search with admissible heuristic. Bayes theorem with numerical example. Camera intrinsics and pinhole projection. HMM filtering (forward algorithm) by hand.

Watch for

If DH seems abstract, work through the specific 2R example three times: with joint angles (0,0), (90,0), (0,90). Verify each time with the geometric formula.

Weeks 5-6: Oct 19 - Nov 1

Theme: Jacobians and state estimation

What to study: Jacobian matrix: derivation, singularity, force mapping. Kalman filter: predict/update by hand for 1D and 2D examples. EKF: linearisation concept and when it fails. Feature detection, matching, RANSAC. Alpha-beta pruning in minimax.

Watch for

If your Kalman filter numbers do not match, check: (1) is your F matrix correct, (2) is the predict step using P_{prior} not P_{post} , (3) is the update step using P_{prior} not P_{post} .

Weeks 7-8: Nov 2 - Nov 15

Theme: Dynamics, MDPs, and optical flow

What to study: Robot dynamics: Lagrangian approach on pendulum, then 2R arm. Particle filters: concept, weight update, resampling. MDPs and value iteration. Lucas-Kanade optical flow: constraint equation and least-squares solution. Epipolar geometry.

Watch for

Particle filters: if your weights always collapse to one particle after a few steps, your resampling is wrong. Resampling should produce N particles, not fewer.

Weeks 9-10: Nov 16 - Nov 29

Theme: Integration and mock exams

What to study: Full revision of all topics. First timed mock exam for each subject (cold, full duration). Error categorisation: for each wrong answer, identify whether it was a conceptual error, an algebraic error, or a notation error. Address conceptual errors first.

Watch for

A mock exam that you score yourself is only useful if you are honest. A wrong answer you rationalise as "I knew it really" is still a wrong answer.

Weeks 11-12: Nov 30 - Dec 7

Theme: Final consolidation and exams

What to study: Reconstruct key equations from memory (DH matrix, Jacobian formula, Kalman update, Bellman equation, pinhole projection). Shorter targeted practice on specific weak topics. Confirm exam venue, time, permitted materials.

Watch for

In the last week, do not introduce new material. Only strengthen what you already know.

The Readiness Test

One week before the first exam, give yourself this test. Sit at a clean desk with a blank piece of paper. For each topic below, write: the key equation or algorithm, the key assumption, and one example of what the method cannot do. If you cannot write all three without looking anything up,

that topic is not yet learned.

- SSO5201: Homogeneous transform — composition and inversion formula
- SSO5201: DH parameter table — what each parameter represents
- SSO5201: 2-link IK — both solution branches and the workspace boundary
- SSO5201: Jacobian — how to derive it and what singularity means
- SSO5201: Lagrangian dynamics — for a 1-DOF system
- SSO5202: BFS vs UCS — when they give different paths
- SSO5202: A* — why admissibility guarantees optimality
- SSO5202: HMM filtering — the forward algorithm in full
- SSO5202: Kalman filter — predict step and update step with K
- SSO5202: MDP Bellman equation — one iteration of value iteration
- SSO5203: Pinhole projection — the formula and scale ambiguity
- SSO5203: Camera calibration — what reprojection error measures
- SSO5203: RANSAC — number of iterations formula
- SSO5203: Lucas-Kanade flow — the constraint equation and aperture problem
- SSO5203: SLAM — what loop closure corrects and why drift occurs

After Dec 7

The three subjects you studied for these exams are the foundation for everything in the Master Guide (PDF 1). The transforms you learned in SSO5201 are used in T1-A for calibrating the measurement rig. The Kalman filter from SSO5202 is implemented in T1-B. The camera model from SSO5203 is used throughout T1-A. The exams and the research preparation are not separate — they reinforce each other. After Dec 7, the exam hours redirect to T2 coursework and your T1 projects. The research lane (PDF 1) continues without interruption.