Project Concept Review: `kinematics.py`

This document explains every major robotics concept used in your codebase. Use this to answer "Why did you use this?" and "How does it work?" questions during your presentation.

1. Forward Kinematics (FK)

What it is: Calculates the End-Effector (Hand) position (x, y, z) from Joint Angles $(\theta_1, \theta_2, d_3)$.

Why are we using it?

- **Visualization:** To draw the robot on the screen (Simulation).
- **Collision Checking:** We need to know where every part of the arm is in 3D space to see if it hits anything.
- **Feedback:** To verify the robot reached where we wanted it to go by checking if the calculated position matches the target.

How are we using it?

- **Code:** `SCARAKinematics.forward_kinematics` (Line 36)
- **Math:** We use **Trigonometry**.
  - $x = l_1 \cos(\theta_1) + l_2 \cos(\theta_1 + \theta_2)$
  - $y = l_1 \sin(\theta_1) + l_2 \sin(\theta_1 + \theta_2)$
  - $z = H_{base} - d_3$ (Decoupled Z-axis for SCARA robots).

2. Inverse Kinematics (IK)

What it is: Calculates required Joint Angles $(\theta_1, \theta_2, d_3)$ to reach a Target Position (x, y, z).

Why are we using it?

- **Control:** We tell the robot "Go to the bolts location (x,y,z)". The motors don't understand "x,y,z", they only understand angles. IK translates the goal into motor commands.

How are we using it?

- **Code:** `SCARAKinematics.inverse_kinematics` (Line 51)
- **Math:** We use **Geometry (Law of Cosines)** because it is faster and more stable than numerical methods for simple arms.
  - Calculate distance D to target.
  - Solve triangle formed by l_1, l_2, D to find elbow angle θ_2.
  - Use atan2 to find shoulder angle θ_1.

3. Lagrangian Dynamics

What it is: Calculates forces/torques required to create motion $\tau = M(q)\ddot{q} + C(q, \dot{q})\dot{q} + G(q)$.

Why are we using it?

- **Realism:** To know if our motors are strong enough.
- **Gravity Compensation:** The vertical motor needs constant power just to hold the arm up against gravity.
- **Inertia:** It's harder to spin the arm when it is fully stretched out than when it is folded. Dynamics accounts for this changing "rotational weight".

How are we using it?

- **Code:** `SCARAKinematics.calculate_dynamics` (Line 83)
- **Math:**
  - **Inertia Matrix (M):** Calculates mass distribution based on arm pose.
  - **Gravity Vector (G):** Adds force $m \cdot g$ to the Z-axis motor.
  - **Result:** We sum these to get the total Torque needed.

4. Jacobian Matrix

What it is: A matrix of partial derivatives relating Joint Velocity to Cartesian Velocity $(\dot{x} = J\dot{q})$.

Why are we using it?

- **Straight Line Motion:** If we just moved joints linearly, the hand would move in a curve (try swinging your arm). To move in a straight line, we need to adjust joint speeds constantly. The Jacobian calculates these exact speeds.
- **Singularity Detection:** It tells us if the robot is "stuck" (full reach).

How are we using it?

- **Code:** `SCARAKinematics.jacobian` (Line 108)
- **Math:**

$$J = \begin{bmatrix} \frac{\partial x}{\partial \theta_1} & \frac{\partial x}{\partial \theta_2} \\ \frac{\partial y}{\partial \theta_1} & \frac{\partial y}{\partial \theta_2} \end{bmatrix}$$

  We invert this matrix (J^{-1}) in `generate_straight_line_path` to convert desired straight-line velocity into motor velocity.

5. Singularity Detection

What it is: Detecting when the robot arm is fully stretched or fully folded.

Why are we using it?

- **Safety:** At a singularity, the robot loses a degree of freedom. If you try to move in the "locked" direction, the required motor speed becomes **infinite** (Divide by Zero error), which can break the robot or crash the code.

How are we using it?

- **Code:** `SCARAKinematics.is_singular` (Line 163)
- **Math:** We check the **Determinant** of the Jacobian ($\det(J)$). If $\det(J) \approx 0$, we are in a singularity.

6. Collision Detection

What it is: Checking if two robots occupy the same space.

Why are we using it?

- **Multi-Robot Safety:** Since we have two SCARA robots working next to each other, they might crash. We need to stop them *before* they hit.

How are we using it?

- **Code:** `CollisionDetector.check_collision` (Line 217)
- **Math: Segment-to-Segment Distance**.
  - We model the robot arms as line segments.
  - We calculate the shortest distance between every segment of Robot 1 and Robot 2.
  - If distance < $(Radius_1 + Radius_2)$, it's a collision.