Perception Sensors

- **Radar (Radio Detection and Ranging):**

  Working Principle: Radar systems transmit radio waves (microwaves) and then receive the signals that are reflected back from objects in the environment.

  Data Captured: The time delay between transmission and reception gives range. The Doppler shift (frequency change) of the returned signal gives radial velocity (how fast an object is moving towards or away from the sensor).

  Processing Requirements: Signal processing is used to filter noise and cluster reflected points (called detections) into objects. It's relatively computationally inexpensive.

- **Lidar (Light Detection and Ranging):**

  Working Principle: Lidar systems emit rapid pulses of laser light and measure the time it takes for each pulse to bounce back.

  Data Captured: The time-of-flight directly gives a highly accurate range measurement. By scanning the lasers across a field of view (e.g., using rotating mirrors), a Lidar generates a high-resolution 3D point cloud of the environment, detailing the shape and size of objects.

  Processing Requirements: Processing point clouds is computationally intensive. It involves grouping points into objects, clustering, and classification (e.g., is it a car, pedestrian, or tree?).

- **Camera:**

  Working Principle: Cameras are passive sensors that capture reflected light from the environment through a lens, focusing it onto a 2D digital sensor.

  Data Captured: They capture 2D RGB images (color and intensity) or grayscale images. They provide rich textural and semantic information (e.g., lane markings, traffic signs, color of traffic lights) that other sensors cannot.

  Processing Requirements: Image processing is highly complex. It involves algorithms for object detection, classification, and segmentation, often powered by deep learning.

Stereo Vision

Stereo vision uses two cameras (a stereo pair), horizontally separated by a known distance (the baseline), to perceive depth, much like human eyes.

1. Two Pictures: Two cameras, side-by-side, take a picture of the same thing at the same time. Because they're in slightly different spots, the pictures are a little different.
2. Line Them Up: The pictures are adjusted so that everything lines up perfectly horizontally. This makes the next step much easier.
3. Find the Difference: For every dot (pixel) in the left image, the system looks for the same dot on the same line in the right image. The closer an object is, the more its position will differ between the two images. This difference is called disparity.
4. Calculate Distance: Using the known distance between the cameras and the camera's lens strength, a math formula turns that "disparity" (the difference) into a "depth" measurement (the distance).
    - Big difference (disparity) = Object is close.
    - Small difference (disparity) = Object is far away.

Machine Learning

What are neural networks? How do they learn?

Neural Networks (NNs) are computing systems inspired by biological brains. They consist of interconnected layers of nodes ("neurons"). Each connection has a weight, and each neuron has an activation function.

How they learn: Learning occurs through a process called backpropagation.

1. Forward Pass: Input data is fed through the network, and it produces an output (a prediction).
2. Loss Calculation: The network's output is compared to the correct answer using a loss function, which calculates the error.
3. Backward Pass: The error is propagated backward through the network.
4. Weight Update: An optimization algorithm (like Gradient Descent) uses this error signal to adjust the weights of the connections to minimize the loss, making the network's prediction more accurate over many iterations.

How can NNs aid with processing sensor data?

NNs, particularly Convolutional Neural Networks (CNNs), are exceptionally good at processing perceptual sensor data:

- Camera Images: They are the state-of-the-art for object detection (YOLO, SSD), semantic segmentation (classifying every pixel in an image), and traffic sign recognition.

- Lidar Point Clouds: Specialized NNs (e.g., PointNet, VoxelNet) can directly process 3D point clouds to detect and classify objects like vehicles and pedestrians.
- Sensor Fusion: NNs can be designed to fuse data from multiple sensors (e.g., camera images + lidar points) to create a more robust and complete environmental model than any single sensor could provide.

Kalman Filter

What is it? The Kalman Filter (KF) is an optimal recursive algorithm that estimates the state of a dynamic system from a series of noisy measurements. It operates in a two-step predict-update loop.

1. Predict: Uses the system's model to predict the next state and its uncertainty.
2. Update: Fuses this prediction with a new, noisy measurement, giving more weight to the source (model or measurement) with higher certainty.

States and Sensors for a Self-Driving Car:

States to Estimate: [x, y, vx, vy, heading (θ)] - 2D position, velocity components, and orientation.

Sensors:

- GPS: Provides absolute (x, y) position, but it's noisy and has low update rates.
- IMU (Inertial Measurement Unit): Provides high-frequency accelerations (to update velocity) and angular rates (to update heading). It drifts over time.
- Wheel Odometry: Provides velocity (vx) based on wheel rotation. It can slip on wet or icy roads.

The KF beautifully fuses the high-frequency, drift-prone IMU/odometry data with the low-frequency, absolute-but-noisy GPS data to produce a smooth, accurate, and high-frequency state estimate.

SLAM (Simultaneous Localization and Mapping)

- Concept: SLAM is the computational problem of building a map of an unknown environment while simultaneously keeping track of an agent's location within it. It's a key challenge for autonomy in GPS-denied environments.
- Types: Visual SLAM (V-SLAM) uses cameras. Lidar SLAM uses lidar scanners. They can also be categorized by their core algorithm: filter-based (e.g., EKF SLAM) and graph-based (modern approach, e.g., pose-graph optimization).
- How to Build a Map:

1. As the vehicle moves, sensors (Lidar, cameras) perceive landmarks (e.g., corners, trees, unique objects) in the environment.
2. The vehicle's pose (position & orientation) and the positions of these observed landmarks are estimated together.
3. When the vehicle re-observes a landmark from a new location, it creates a "loop closure," which corrects the accumulated drift in the map and the vehicle's trajectory.

- Useful Sensors: Lidar (for precise geometric maps), Cameras (monocular, stereo, or RGB-D for visual and color maps), IMU (to provide motion priors between camera/lidar frames).

● Localization in an Unknown Environment (Two Types):

- Filter-based SLAM (e.g., EKF-SLAM): The state vector contains both the robot's pose and the coordinates of all landmarks. The Kalman Filter is used to estimate this entire state. The main drawback is that the state vector grows quadratically as more landmarks are added, making it computationally expensive for large environments.
- Graph-based SLAM (Modern Approach): This is the current standard. The robot's poses at different times and the landmark positions are modeled as nodes in a graph. Measurements between poses and landmarks are the edges (constraints) connecting these nodes. The map is built by finding the configuration of nodes that minimizes the error across all constraints. Loop closures are simply new constraints added to the graph, triggering an optimization that corrects the entire trajectory and map.

Graph Search Algorithms

● BFS (Breadth-First Search):

- How it works: Explores a graph level-by-level. It starts at the root and visits all neighbors at the present depth before moving on to nodes at the next depth level. Uses a queue (FIFO).
- Pros: Guaranteed to find the shortest path (in an unweighted graph).
- Cons: Inefficient; explores all possible directions equally.

● DFS (Depth-First Search):

- How it works: Explores a graph by going as deep as possible along a branch before backtracking. Uses a stack (LIFO).
- Pros: Memory efficient for graphs with deep branches; can be faster if the goal is deep.
- Cons: Not optimal; it can find a very long path before finding the shortest one.

Dijkstra’s Algorithm:

- How it works: Finds the shortest path from a start node to all other nodes in a weighted graph (with non-negative weights). It expands the node with the smallest current cost from the start. Uses a priority queue.
- Pros: Optimal; guaranteed to find the shortest path.
- Cons: Explores all nodes closer to the start than the goal, which can be inefficient.

A* Algorithm:

- How it works: An extension of Dijkstra's that uses a heuristic function h(n) to guide the search towards the goal. The priority queue is ordered by f(n) = g(n) + h(n), where g(n) is the cost from the start, and h(n) is an estimate of the cost to the goal (e.g., Euclidean distance).
- Pros: Much more efficient than Dijkstra's while still guaranteeing an optimal path, provided the heuristic is admissible (never overestimates the true cost).
- Cons: Requires a good heuristic; performance depends on heuristic quality.

Practical Motion Planning

- Solution to Zig-Zag Paths: The zig-zag behavior is often a result of the graph's discretization (e.g., a grid). A* can only choose from the discrete set of adjacent grid cells. A solution is to apply a path smoothing algorithm after A* has found the optimal grid path.

    - Method: Use a gradient descent algorithm or a spline interpolation technique to smooth the jagged path into a continuous, drivable curve. A simpler method is the String Pulling algorithm, which checks if intermediate waypoints can be removed while still avoiding obstacles.

- Risk of the Solution: The primary risk is that the smoothed path might cut corners and collide with obstacles that were avoided by the original, jagged A* path. The smoothing algorithm must be observe-aware, meaning it must constantly check that the new, smoothed path segment does not intersect with any known obstacles.

Control Theory and Philosophy

- Control Theory: It is a branch of engineering and mathematics that deals with the behavior of dynamical systems. The objective is to develop strategies (controllers) to get a system to behave in a desired way by using feedback and controlling its outputs. Its significance is vast, enabling everything from simple thermostats to autopilots and industrial robotics.
- Open-Loop vs. Closed-Loop:

- Open-Loop: The control action is independent of the system output. Example: A washing machine that runs for a pre-set time regardless of how clean the clothes are. It's simple but inaccurate if there are disturbances.
- Closed-Loop (Feedback Control): The control action is dependent on the system output. The output is measured and fed back to be compared to the desired reference (setpoint). The error is used to correct the system's behavior. Example: A cruise control system that measures the car's speed and adjusts the throttle to maintain it, even on a hill.
- Need for Feedback: You need feedback whenever the system is subject to unknown disturbances (e.g., wind pushing a drone off course) or has model inaccuracies (e.g., an engine's power output varying with temperature). Feedback compensates for these unknowns.

PID Control

- What is a PID Controller? It is the most common control algorithm that calculates an "error" value as the difference between a measured process variable and a desired setpoint. The controller attempts to minimize the error by adjusting the process control inputs through a weighted sum of three terms:

    - Proportional (P): Output $\propto$ Current Error. Provides an immediate control action based on the current error. A high P-gain makes the system react quickly but can cause overshoot and oscillation.
    - Integral (I): Output $\propto$ Accumulation of Past Errors. Eliminates steady-state error (the residual error after the system has settled). It "remembers" all past errors and acts on them. Too much I-gain can cause slow oscillation and windup.
    - Derivative (D): Output $\propto$ Rate of Change of Error. Predicts future error based on its current rate of change. It adds damping to the system, reducing overshoot and oscillation. It is very sensitive to measurement noise.

- Tuning a PID Controller:

1. Manual Tuning: Start with I and D gains set to zero. Increase P until the system oscillates, then reduce it slightly. Then, increase I to eliminate steady-state error. Finally, add D to reduce overshoot.
2. Ziegler-Nichols Method: A heuristic method. Increase P until steady oscillations occur (ultimate gain Ku, ultimate period Pu). Then set P, I, and D gains based on formulas using Ku and Pu.

- Industrial Furnace Temperature Control:

1. Process Variable (PV): Measured Temperature (from a thermocouple).
2. Setpoint (DP): Desired Temperature.
3. Error (e): e = DP - UV(current value).

4. Control Output (u): The signal to the actuator (e.g., a fuel valve or heating element).
5. Actuator: Adjusts the heat input to the furnace based on u.

- **Limitations of PID:**
    - Performance can degrade for non-linear systems.
    - Not ideal for systems with long time delays.
    - Tuning can be difficult for complex, higher-order systems.
    - Scenarios for other controllers: For systems with significant delays, a Smith Predictor might be better. For highly non-linear systems or those where optimal performance is critical (e.g., aerospace), Model Predictive Control (MPC) or state-space controllers are more appropriate.