Appunti automative

ANTONIO SESSA E AMICI

April 2025

Contents

1 Introduction

Veicolo a guida autonoma (AV): è un veicolo che utilizza una combinazione di sensori (telecamere, LIDAR, radar, GPS) e intelligenza artificiale (AI) per muoversi autonomamente, secondo determinati livelli di automazione definiti dallo Standard SAE J3 016, su strade che non siano state pre-adattate allo scopo.

Veicolo autonomo connesso (CAV): è un veicolo che assiste o sostituisce l'umano alla guida, avvalendosi di tutti i dati forniti nell'ambito di una smart road dagli altri veicoli e da tutti gli elementi che fanno parte della strada (segnali,semafori, linee, marciapiedi,spartitraffico etc.).

1.1 Livelli di guida autonoma

CRITERI: Attenzione del guidatore Azioni del guidatore Contesto nel quale il veicolo può svolgere il task di guida autonoma

Contesto di guida:in quali contesti si può avere l'automazione della guida. In inglese il contesto di guida è Operational Design Domain ODD

Livello 0, no automazione Livello 1, automazione solo su sterzata o solo su accelerazione e frenata (ma non entrambi insieme), sarebbe il Cruise Control o Lane Keeping Assistance Livello 2, automazione sia su sterzata e acceleraizione/frenata; ma guidatore attento e in solo in certi contesti Livello 3, Livello 2 + il guidatore deve rimanere attento, ma di meno, dato che il veicolo stesso avvertirà il guidatore quando ci sono eventi anomali. Livello 4, autonomia elevata (o selettiva), totalmente autonomo in determinati contesti. Livello 5, autonomia totale in qualsiasi contesto (non esistono esempi)

1.2 Cosa fa un veicolo a guida autonoma

Motion planning, stabilire il percorso da seguire da partenza a destinazione. **Controllo del veicolo**, impartire camandi di accelerazione, sterzata e frenata. **Percezione**, percepire segnali stradali, pedoni e altri veicoli. Usando sensori e interagendo con la smart road e veicoli connessi.

2 Trasformazioni tra Sistemi di Riferimento

Quando più veicoli o agenti rilevano lo stesso oggetto, ciascuno misura la sua posizione in base al proprio sistema di riferimento locale. Ad esempio, se un ostacolo ha coordinate globali (x_m, y_m), i veicoli 1 e 2 lo percepiranno rispettivamente come (x_{v1}, y_{v1}) e (x_{v2}, y_{v2}), che rappresentano la posizione dell'ostacolo nel proprio frame locale. Tuttavia, i sensori come LiDAR, radar o sonar forniscono soltanto misurazioni relative alla posizione del veicolo stesso, non le coordinate globali reali.

2.1 Necessità della Trasformazione di Riferimento

Senza trasformazioni corrette tra sistemi di riferimento, la combinazione delle misurazioni da più agenti può portare a conclusioni errate. Per rendere confrontabili le misure tra diversi veicoli, è necessario trasformare le coordinate da un sistema all'altro. Ciò richiede la gestione di due operazioni fondamentali: **rotazione** e **traslazione**.

2.2 Coordinate Omogenee

Per facilitare le trasformazioni, si utilizzano le **coordinate omogenee**. Invece di rappresentare un punto come (x, y, z) (coordinate cartesiane), esso viene rappresentato come (x, y, z, w), dove tipicamente $w = 1$. Le coordinate cartesiane possono essere recuperate dividendo ogni componente per w:

$$(x, y, z, w) \Rightarrow \left(\frac{x}{w}, \frac{y}{w}, \frac{z}{w}\right)$$

Quando $w = 0$, il punto rappresenta una direzione all'infinito, che non può essere rappresentata nello spazio cartesiano, ma è utile per operazioni geometriche avanzate.

2.3 Matrice di Trasformazione dell'Agente

Per convertire una misurazione dal sistema di riferimento dell'agente B a quello dell'agente A, è necessario calcolare la **matrice di trasformazione relativa** $T_{B\to A}$, che si ottiene in due passaggi:

1. Si trasforma il punto misurato dall'agente B nel sistema di coordinate globali tramite la matrice T_B.
2. Si trasforma questo punto dal sistema globale al sistema di riferimento dell'agente A, usando l'inversa della matrice T_A: T_A^{-1}.

Quindi, la trasformazione completa si scrive come:

$$T_{B\to A} = T_A^{-1} T_B$$

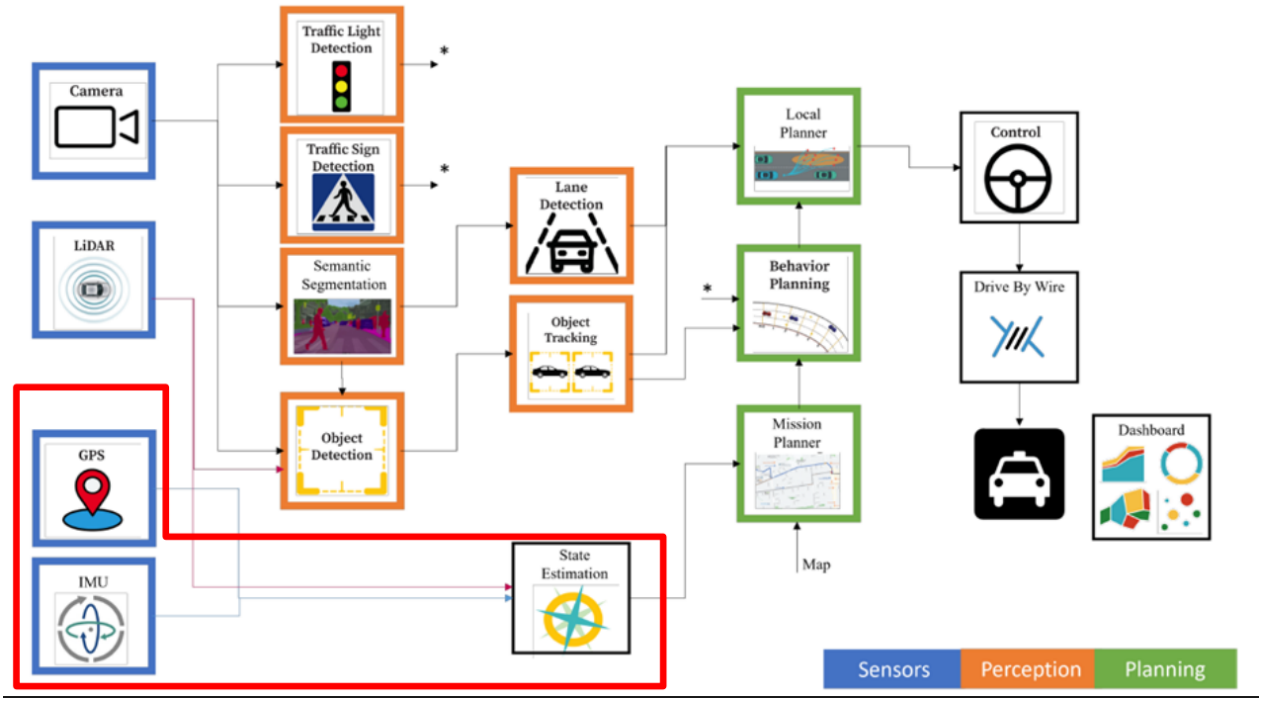


Figure 1: Sensors, perception, planning

2.4 Rotazioni: Roll, Pitch e Yaw

Le rotazioni nello spazio tridimensionale possono essere descritte tramite tre angoli: **Roll** (rollio), **Pitch** (beccheggio) e **Yaw** (imbardata), associati rispettivamente alle rotazioni attorno agli assi x, y e z. Le matrici di rotazione elementari sono:

- $R_x(\alpha)$: rotazione attorno all'asse x
- $R_y(\beta)$: rotazione attorno all'asse y
- $R_z(\gamma)$: rotazione attorno all'asse z

Componendo queste rotazioni si ottiene la matrice di rotazione R che entra nella definizione di T, la matrice di trasformazione completa (rotazione + traslazione) in coordinate omogenee:

$$T = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix}$$

dove t è il vettore di traslazione.

Grazie all'uso delle coordinate omogenee e delle matrici di trasformazione, è possibile convertire correttamente le misure tra diversi sistemi di riferimento. Ciò consente a più agenti di condividere e interpretare in modo coerente le informazioni percettive provenienti dai rispettivi sensori.

3 State estimation

Goal: stima della posa, posa = orientamento + posizione nel mondo (e sulla strada) ad ogni momento
Queste informazioni possono essere misurate da GNSS (Global Navigation Satellite System) e IMU (inertial measurement unit)

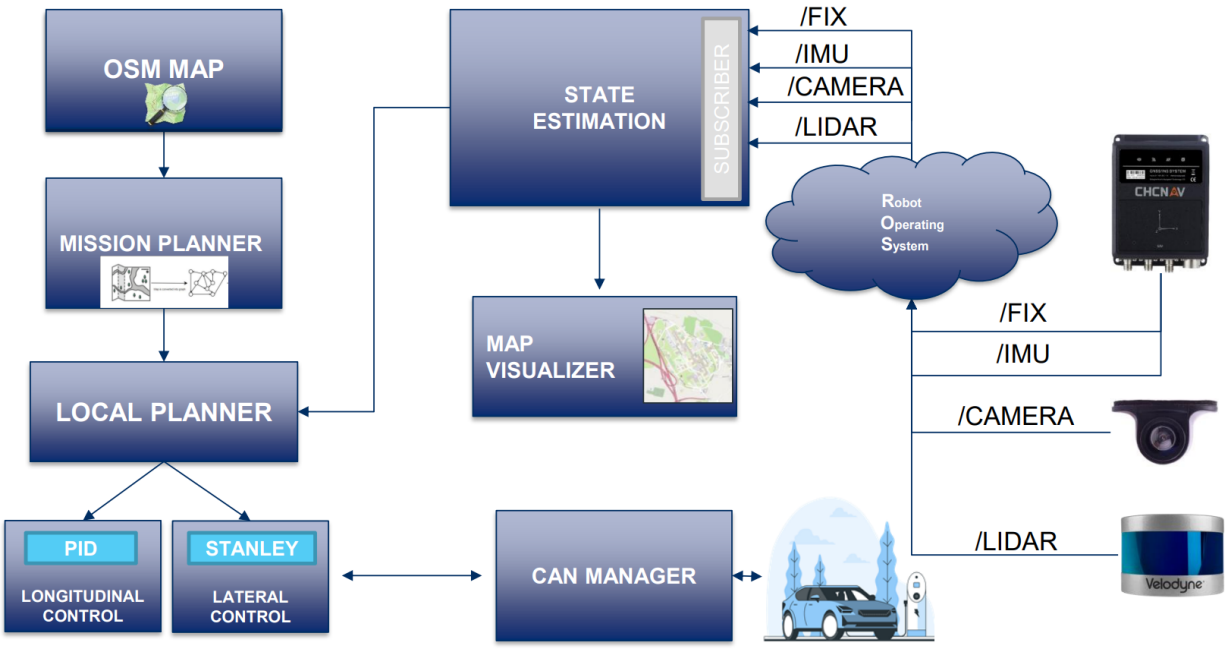


Figure 2: MIVIA CAR software architecture

3.1 IMU, inertial measurement unit

Un IMU è solitamente composto da tre giroscopi e tre accelerometri. I giroscopi misurano le velocità di rotazione angolare attorno a tre assi separati, mentre gli accelerometri misurano le accelerazioni lungo tre assi ortogonali. Alcune IMU includono anche magnetometri, o una bussola, per aiutare a tracciare l'orientamento. I dati in uscita di una IMU sono tipicamente accelerazioni nel sistema di riferimento del corpo, velocità angolari e (opzionalmente) misurazioni del campo magnetico. Imu calcola l'accelarazione inerziale (?) che varia in base a dove l'IMU è localizzato. **Siamo interessati a posizionarlo al centro di massa del veicolo, su cui poi calcoleremo l'accelerazione.**

3.2 Accelerometro MEMS

Un accelerometro è un sensore fondamentale per la misura dell'accelerazione inerziale, ovvero la variazione di velocità nel tempo. Esistono diversi tipi di accelerometri, tra cui quelli meccanici, al quarzo e i MEMS (Micro Electro-Mechanical Systems).

Un accelerometro MEMS è composto da una massa (detta *proof mass*) sospesa da una molla. La direzione in cui la massa può muoversi è chiamata asse di sensibilità. Quando l'accelerometro è soggetto a un'accelerazione lungo questo asse, la massa si sposta proporzionalmente all'accelerazione.

Il movimento della massa provoca una deflessione rilevabile tramite un circuito capacitivo. La variazione di capacità (C), definita come $C = \frac{\varepsilon A}{d}$ (dove ε è la permittività, A l'area delle piastre e d la distanza tra esse), permette di calcolare l'accelerazione. La capacità è inversamente proporzionale alla distanza tra le piastre: se la massa si sposta, cambia la distanza, modificando la capacità.

Per rilevare il movimento in più direzioni, i sensori sono montati lungo assi diversi. I segnali capacitivi vengono amplificati, condizionati, filtrati (low-pass) e infine convertiti in digitale tramite un convertitore analogico-digitale (ADC).

3.3 Giroscopio

Un giroscopio è un sensore inerziale che misura la velocità angolare di un oggetto rispetto a un sistema di riferimento inerziale. Quando una massa si muove con una certa velocità in una direzione e viene applicata

una velocità angolare esterna, si genera una forza (effetto Coriolis) che provoca uno spostamento della massa. Questo spostamento, come per l'accelerometro, causa una variazione di capacità che può essere misurata per determinare la velocità angolare.

3.4 Magnetometro

Il magnetometro è un sensore che misura l'intensità e la direzione di un campo magnetico. Viene spesso utilizzato per determinare l'orientamento (attitude) del dispositivo rispetto al campo magnetico terrestre, fungendo da bussola elettronica.

3.5 Il funzionamento dell'IMU

Un'Unità di Misura Inerziale (IMU) a 6 gradi di libertà è composta da tre accelerometri e tre giroscopi, a cui può essere aggiunto un magnetometro. I giroscopi misurano le velocità angolari, mentre gli accelerometri rilevano la variazione di velocità (l'accelerazione), entrambi nel sistema di riferimento del sensore. Il magnetometro contribuisce alla stima dell'orientamento assoluto.

Tuttavia, le IMU tendono a derivare nel tempo e sono difficili da calibrare con precisione. Per correggere queste derive e migliorare la stima della posizione e dell'orientamento, si utilizza un sistema ausiliario come il GNSS (sistema globale di navigazione satellitare), che fornisce aggiornamenti periodici della posizione assoluta.

3.6 GNSS, Global Navigation Satellite System

Il GNSS usa una constellazione di satelliti per calcolare la posizione. La constellazione è designed tale che almeno 4 satelliti sono visibili da qualsiasi punto delle Terra ad ogni instante. GNSS Receiver, convertono i segnali GNSS visibili in una posizione sulla Terra. Il receiver è composto da una antenna, che riceve il segnale, e dal receiver che è una board con un modulo GNSS che riesce a decodificare le informazioni dal satellite. COME: ogni segnale contiene un codice pseudo-random che identifica il satellite e le posizione del satellite e l'instante di trasmissione del segnale. Con queste informazioni, sapendo che il segnale viaggia alla velocità della luce, si riesce a calcolare la distanza dal satellite comparando il tempo di partenza e di arrivo (tempo di arrivo calcolato col proprio clock). Serve il segnale di 4 segnali per calcolare x,y,z e il clock bias. Servono 3 segnali se non si è interessati all'altezza. Questo processo è detto trilaterazione. L'accuracy è dai 2 a 10 metri. Errori causati dal fatto che il segnale viaggia tra diverse atmosfere che lo rallentano e perturbano. RTK real-time kinematic migliora l'accuracy. Si piazza un GNSS detto base-station ad una posizione di cui è nota nell'ordine dei centimetri la poszione reale. I ricevitori GNSS mobili (detti rover) ricevono dalla base station correzioni per arrivare a precisione centimetriche. Precisione dei 1-2 centimetri se la distanza dalla base-station è tra i 40km-50km (più la base station è lontana più i segnali ricevuti dal rover e base station saranno diversi). Il segnale di correzione è ricevuto attraverso internet NTRIP protocol, ed è quindi necessario avere un receiver equipaggiato con una SIM perché deve essere connesso su internet. MULTISENSOR FUSION FOR STATE ESTIMATION, l'imu solitamente fornice informazioni rumorose sulla posizione ad alta frequenza. Con frequenza più bassa, riceviamo informazioni sul GNSS che integriamo usando il filtro di Kelman esteso.

3.7 DGPS e RTK

Il **DGPS** (Differential GPS) è un sistema di posizionamento che migliora la precisione del GPS standard tramite correzioni differenziali. Una stazione di riferimento fissa, la cui posizione è nota con precisione, riceve i segnali dai satelliti GPS e confronta la posizione calcolata con quella reale. La differenza tra le due è usata per generare un segnale di correzione, che viene trasmesso a ricevitori mobili (rover). Il DGPS classico si basa sulla correzione degli errori nei codici pseudocasuali (C/A code), ottenendo precisioni dell'ordine del metro o del decimetro.

Il **RTK** (Real-Time Kinematic) è un'estensione avanzata del DGPS che sfrutta, oltre ai codici, anche la *fase della portante* del segnale GNSS. La fase, rappresentata da un'onda sinusoidale ad alta frequenza, permette di ottenere precisioni centimetrica o sub-centimetrica. Il ricevitore calcola la propria distanza dal satellite con l'equazione:

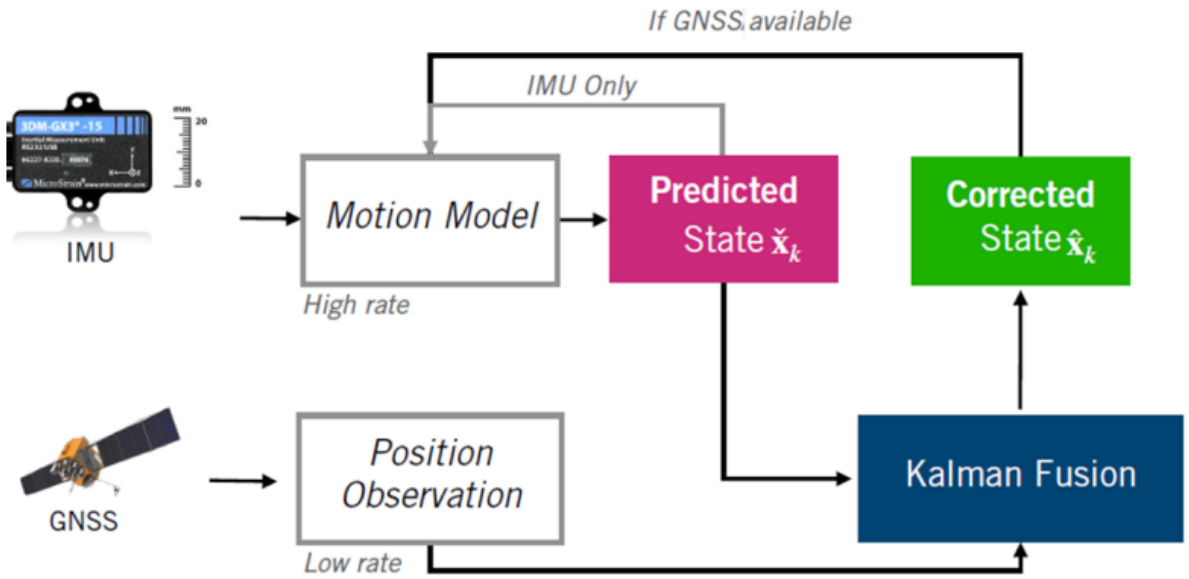


Figure 3: Multi sensor fusion

$$\rho = N \cdot \lambda + \phi \cdot \lambda$$

dove ρ è la distanza stimata, N è il numero intero di cicli (ambiguità intera), λ è la lunghezza d'onda, e ϕ è la parte frazionaria del ciclo.

Il cuore del metodo RTK è la **risoluzione delle ambiguità intere** (N), che si effettua confrontando la fase del segnale tra la base e il rover. Una volta risolta, il sistema è in grado di calcolare la posizione del rover con precisione centimetrica, a patto che la comunicazione con la base station avvenga in tempo reale.

- DGPS classico: correzione ai codici, precisione $\approx$ 1–3 m.
- RTK: correzione a codici + fasi portanti, precisione $\approx$ 1–2 cm.

4 Controllo longitudinale e laterale

Il controllo può essere visto come un algoritmo che minimizza il segnale di errore, portanto il processo allo stato desiderato. Il termine proporzionale dipende dall'errore (differenza tra il riferimento e stato attuale), da solo non potrà mai portare a 0 l'errore, perché per generare un output, il controllo proporzionale ha sempre bisogno dell'errore. Causa overshoot, diminuisce il tempo di salita. (Per produrre un segnale, il controllo proporzionale deve avere errore) Il termine integrale dipende dalla comulazione degli errori passati, quindi è capace di portare l'errore a 0, ma causa overshoot, diminuisce il tempo di salita ma aumento il tempo di assestamento. Il termine derivativo cerca di diminuire l'errore futuro, basandosi sul suo tasso di variazione attuale (derivata)

4.1 Controllo laterale, controller di Stanley

Usa il centro dell'asse anteriore come punto di riferimento per calcolare gli errori. Gli errori presi in considerazioni sono quelli di heading (angolazione), dove il veicolo sta puntando rispetto a dove deve puntare) e di crosstrack (distanza laterale) tra il veicolo (rappresentato dal punto centrale dell'asse anteriore) e la traiettoria desiderata. L'equazione del **Stanley Controller** è la seguente:

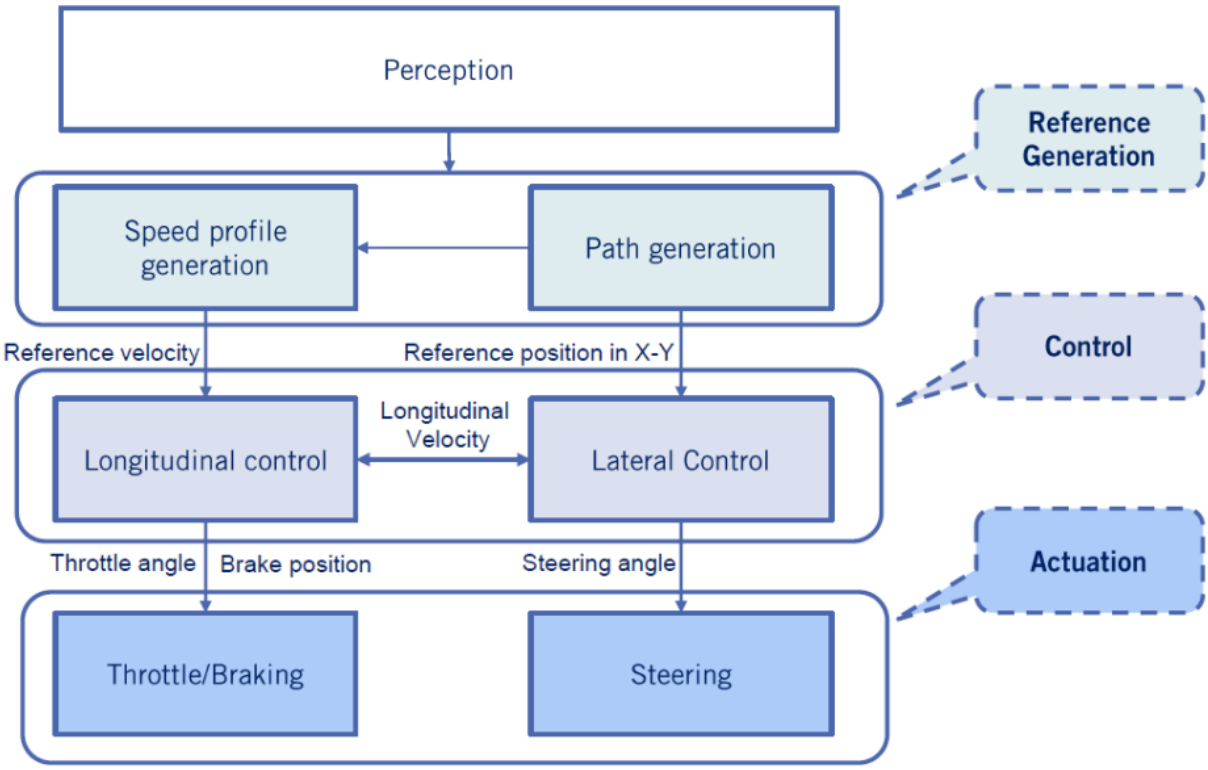


Figure 4: Architettura della strategia del controllo nei veicoli

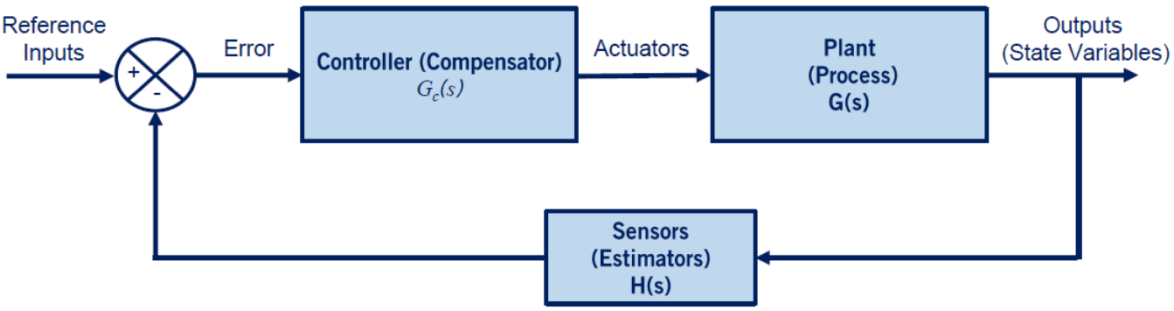


Figure 5: Architettura del controllo in retroazione

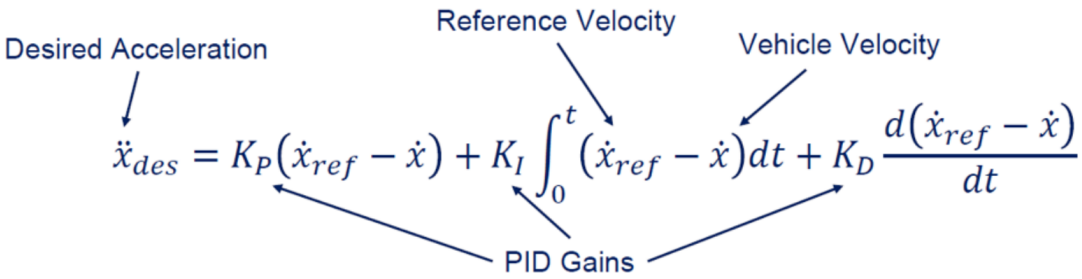


Figure 6: Espressione del controllo PID

$$\delta = \theta_e + \arctan\left(\frac{k \cdot \text{CTE}}{v}\right)$$

$$\delta_{\text{cmd}} = \text{clip}\left(\delta, -\delta_{\max}, +\delta_{\max}\right)$$

oppure con il termine di CTE con un termine di ammortizzamento k_s per limitare gli effetti dovuti a bassa velocità che porterebbero valori elevati.

$$\delta = \theta_e + \arctan\left(\frac{k \cdot \text{CTE}}{v + k_s}\right)$$

In più si può aggiungere un termine di ammortizzamento anche sull'heading, per evitare sterzate troppo forte in casi di alta velocità. I simboli rappresentano:

- δ è l'angolo di sterzata (steering angle)
- θ_e è l'errore di orientamento (heading error), che sarebbe la differenza tra l'orientamento attuale e l'orienamento desiderato
- CTE è la Cross Track Error, la distanza minima (ortogonale) tra il punto centrale dell'asse anteriore e la traiettoria desiderata
- k è il guadagno del controller (gain)
- v è la velocità del veicolo

Per un grande errore di CTE (distanza), arctan(k*CTE/v) = pi/2 (la funzione arctan tende a pi/2 se argomento è positivo infinito, e a -pi/2 per argomento negativo infinito)

La velocità non influenza il tempo di convergenza, un veicolo veloce farà più strada, ma arriverà ad essere sul percorso allo stesso momento di un veicolo più lento che avrà fatto meno strada.

5 Sensors

I sensori sono i device che misurono/rilevano proprietà dell'ambiente o cambiamenti. I sensori si dividono in esteroricettivi, che rilevano l'ambiente, e propriorecettivi, che rilevano proprietà del veicolo.

5.1 Esterorecettivi

5.1.1 Camera (e stereocamera)

Camera, che si distinguono per Field of view, risoluzione e dynamic range (differenza tra il punto più scuro e più chiaro dell'immagine), importante per l'auto sopratutto quando è notte. Stereo camera, combinazione di due camere con overlapping fox, che permetta la stima di profondità.

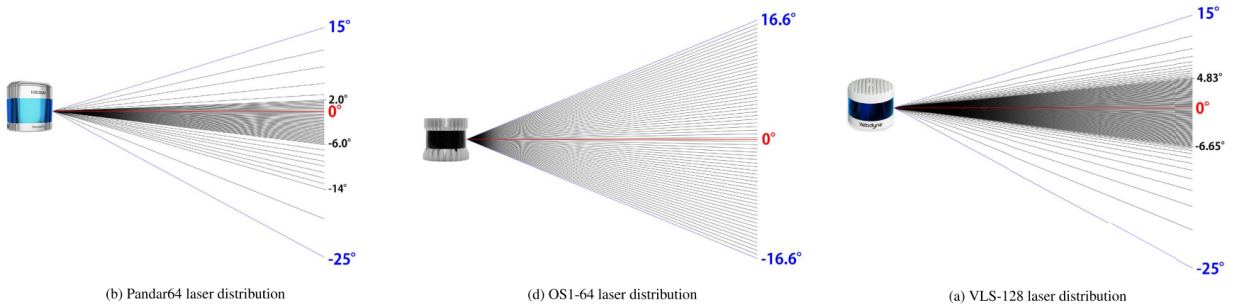
5.1.2 Lidar

Lidar, light detection and ranging sensor, basato sul time of flight dei raggi di luce, che permettono il calcolo della distanza e dimensione di oggetti su cui riflettono. I lidar include un elemento rotante em multipli emittitori luminosi montati su di esso. L'output del lidar è una cloud map tridimensionale. Lidar non è affetto da disturbi ambientali come scarse condizioni luminose. Le componenti principali dei LIDAR sono un laser, un fotorivelatore e un preciso cronometro. Per calcolare la distanza si usa il Time-of-Flight ranging.

$$r = 1/2 * c * t$$

dove: r è la distanza, c è la velocità della luce, t è il round-trip time (tempo che ci mette il raggio di luce per arrivare al fotorilevatore)

Data la velocità della luce, si possono considerare tutti gli oggetti nell'ambiente come stazionari. Inoltre i LIDAR misurano anche l'intensità del raggio di ritorno, che può essere utile per ricreare immagini 2D su cui usare tipici algoritmi di visione.



(b) Pandar64 laser distribution (d) OS1-64 laser distribution (a) VLS-128 laser distribution

Figure 7: Lidar laser diagram

Metriche per cui si distinguono i lidar sono: Numero di raggi, 16,32,64,128 Punti per secondo, indica quanti singoli impulsi laser vengono emessi, riflessi da un oggetto e rilevati dal sensore ogni secondo. Rotation rate, è la velocità di rotazione del sensore attorno al suo asse verticale, più veloce è più veloce viene aggiornata la cloud map. Range, quanto lontano può "vedere" il lidar, determinato dall'intensita luminoso dell'output luminoso. Field of view, l'estensione angolare del LIDAR.

5.1.3 Radar

Radar, sta per radio detection e ranging, usa le onde radio per localizzare gli oggetti. COME: il principio operativo è basato sulla trasmissione e ricezione di un segnale radio ad una determinata frequenza. Il segnale radio viene trasmessa da un antenna che manda segnali elettromagnetici in un intervallo regolare e riceve l'eco. Basandosi sul round trip time del segnale e sull'effetto Dopler, permette di determinare la distanza e la velocità relativi degli oggetti. (The Doppler effect is the change in frequency in the radar echo due to the relative motion of the target. Based on the Doppler effect, the radar emits electromagnetic waves that the detected object reflects. The device immediately determines whether these waves are compressed or stretched, i.e. whether the object is moving closer or further away and calculate the speed at which the object is moving. This allows for instant detection of any target approaching or receding from the radar) Metriche per cui si distringuono i radar sono: range, quanto lontano può rilevare field of view, position and speed accuracy. Configurazioni possibili sono: FOV largo, short range e FOV stretto, long range.

5.1.4 Sonar

Sonar, sound navigation and ranging (detto anche ultrasonic), metrice di comparazione sono range, fov e costo. Sono economici e usati per il parking assistance.

5.2 Proprioricettivi

GNSS/IMU, Odometria delle Ruote, misura la velocità delle ruote e l'orientamento, usando per calcolare la velocità e orientamento del veicolo.

5.3 Configurazione dei sensori

Come dovremmo piazzare i sensori così che abbiamo sufficiente tempo per il guidatore di reagire? Le manovre che il guidatore solitamente intrapende sono: stop di emergenza, considerando l'equazione:

$$Stoppingdistance : d = v^2/2a$$

E decellerazione aggressiva 5 m/s$\hat{2}$, i sensori longitudinali devono arrivare almeno a 110 metri se ci si trovo ad una velocità di 120 km/h (la maggior parte dei veicoli arrivono a 150-200 metri) Per i sensori laterali servono almeno 3.75 metri (dimensione di una corsia in Italia).

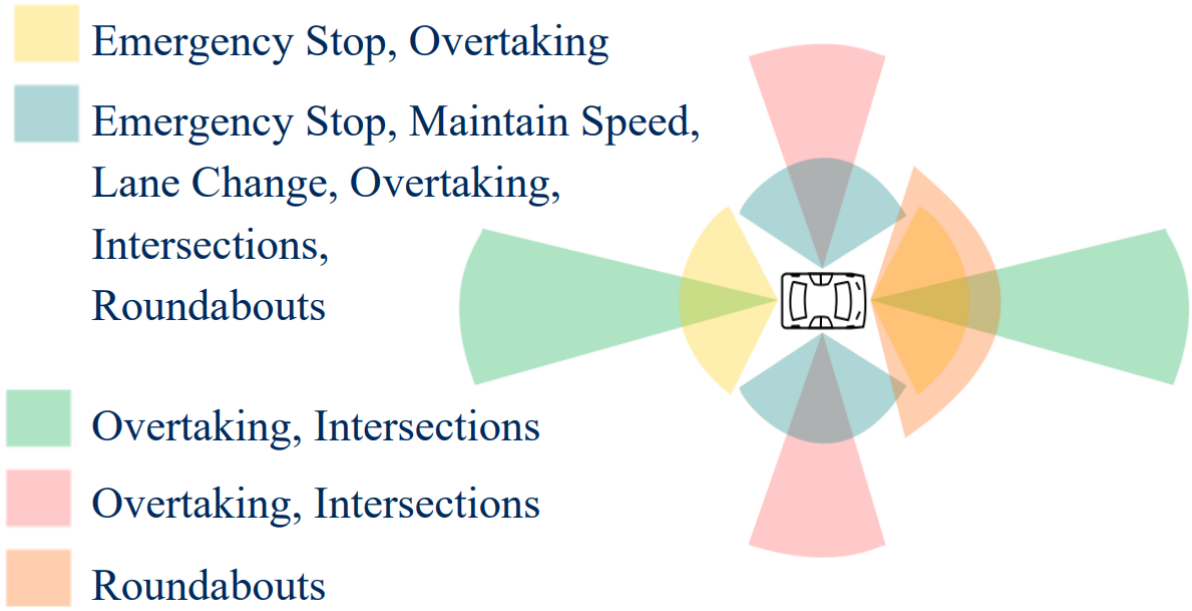


Figure 8: Coverage necessaria per la guida autonoma

Figure 9: Coverage di una Tesla

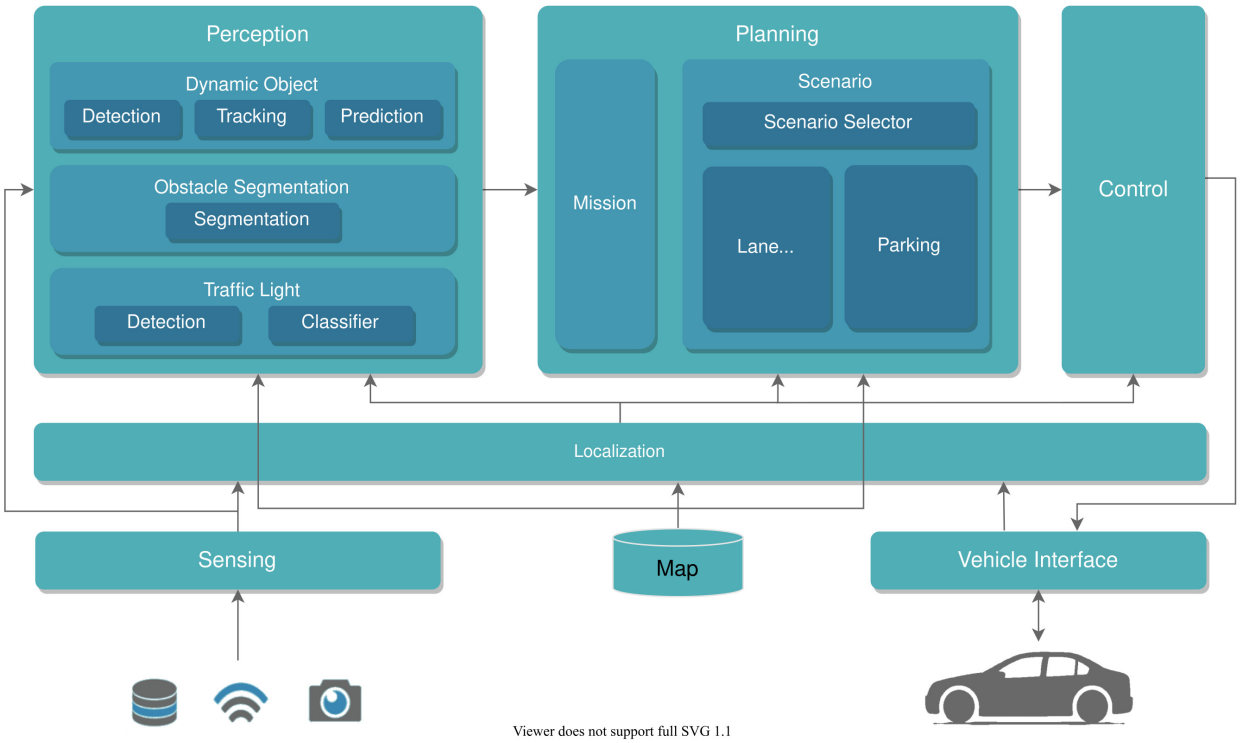


Figure 10: Architettura Autoware

6 Autoware, componenti dell'architettura

Autoware è costruito su ROS (Robot Operating System). Autoware usa una **pipeline architecture**, basata su due moduli principali, il **CORE**, che contiene moduli per il sensing, computazione e attuazione, e l' **UNIVERSE**, che permette l'estensione dei moduli forniti dal core e il loro enhancing.

6.1 Sensing

- Applica pre-elaborazioni primitive ai dati grezzi dei sensori.
- Definisce i formati di input dei sensori e astrae i formati dati per consentire l'uso di sensori di vari fornitori.
- Esempi di input: `sensor_msgs/PointCloud2` (LiDAR), `sensor_msgs/Image` (Camere), `radar_msgs/RadarScan`, `sensor_msgs/NavSatFix` (GNSS).

6.2 Map Data

Autoware si basa su mappe point cloud ad alta definizione e mappe vettoriali.

- **Vector Map:**
  - Formato **Lanelet2** (con modifiche specifiche per Autoware).
  - Contiene informazioni accurate sulla rete stradale, geometria delle corsie, semafori, regole stradali (limiti di velocità, diritto di precedenza), ecc.
  - Essenziale per la pianificazione del percorso, il rilevamento dei semafori e la previsione delle traiettorie.
  - Deve coprire l'intera area operativa del veicolo.

- **Point Cloud Map (opzionale):**
  - Formato **PCD** (Point Cloud Data).
  - Contiene coordinate X, Y, Z; opzionalmente intensità o valori RGB.
  - Risoluzione minima consigliata: 0.2m per una localizzazione affidabile.
  - Deve essere georeferenziata (coordinate globali) per l'uso con dati GNSS.
- **Projection Info:**
  - File YAML che specifica il sistema di coordinate della mappa e il metodo di proiezione per convertire tra coordinate locali e globali.
- **Output della Mappa:** Forniti a Sensing (conversione GNSS), Localizzazione (LiDAR-based, road markings), Percezione (segmentazione ostacoli, predizione traiettorie), Pianificazione (pianificazione del comportamento), API layer (conversione risultati localizzazione).

6.3 Localization

- **Obiettivi:** Stimare posa, velocità e accelerazione del veicolo; diagnosticare la stabilità della stima; funzionare con varie configurazioni di sensori.
- **Input:** Messaggi dei sensori (LiDAR, camera, GNSS, IMU, CAN Bus), dati della mappa, trasformate `tf`.
- **Output:** `PoseWithCovarianceStamped`, `TwistWithCovarianceStamped`, `AccelWithCovarianceStamped`, diagnostica, `tf(map to base_link)`.
- **Implementazione:**
  - **Pose Estimator:** Stima la posa del veicolo sulla mappa confrontando le osservazioni dei sensori esterni con la mappa.
  - **Twist-Accel Estimator:** Produce velocità, velocità angolare, accelerazione e accelerazione angolare del veicolo e le loro covarianze, basandosi su sensori interni.
  - **Kinematics Fusion Filter:** Fonde la posa ottenuta dal Pose Estimator con velocità e accelerazione dal Twist-Accel Estimator.
  - **Localization Diagnostics:** Monitora e garantisce la stabilità e l'affidabilità della stima della posa.

6.4 Perception

- Riceve input da Sensing, Localization e Map; aggiunge informazioni semantiche (riconoscimento oggetti, segmentazione ostacoli, ecc.) e le passa al Planning.
- **Moduli principali:**
  - **Object Recognition:** Riconosce oggetti dinamici (pedoni, veicoli, ecc.) e predice le loro traiettorie future.
  - **Obstacle Segmentation:** Identifica nuvole di punti originate da ostacoli (dinamici e statici come cordoli, barriere, alberi).
  - **Occupancy Grid Map:** Rileva angoli ciechi dove oggetti dinamici potrebbero apparire.
  - **Traffic Light Recognition:** Riconosce i colori dei semafori e le direzioni delle frecce.
- Utilizza librerie da **OpenMMLab** (es. MMDeploy, MMDetection3D) per le funzionalità di computer vision.
- **Funzioni comuni:** Rilevatori 3D basati su LiDAR DNN, rilevatori 2D basati su Camera DNN, clustering LiDAR, fusione di oggetti da più rilevatori, interpolazione, tracciamento, predizione.

6.5 Planning

Componente cruciale per generare traiettorie target (percorso e velocità) per veicoli autonomi. L'architettura è progettata per essere personalizzabile ed estendibile.

- **Modalità di estensione:**
  - Aggiungere o sostituire singoli moduli.
  - Sostituire sotto-componenti (es. Mission Planning).
  - Sostituire l'intero componente Planning.
- **Sotto-Moduli Principali:**
  - **Mission Planning:** Calcola i percorsi dalla posizione corrente alla destinazione, utilizzando i dati della mappa. Include la gestione di Failsafe e Minimal Risk Maneuver (MRM).
  - **Behavior Planning:** Si concentra sul calcolo di percorsi sicuri e conformi alle regole, gestendo decisioni per cambi di corsia, ingressi in incroci e arresti.
    - *Behavior Path Planner:* Migliora la sicurezza, approccio basato su regole. Genera percorsi privi di collisioni (es. con profilo a jerk costante per cambi di corsia fluidi) e aree di manovra sicure (*drivable area*). Utilizza algoritmi di valutazione delle collisioni ispirati a RSS (Responsibility-Sensitive Safety).
    - *Behavior Velocity Planner:* Gestisce la velocità in base a semafori, stop, attraversamenti pedonali, ecc.
  - **Motion Planning:**
    - *Obstacle Avoidance Planner:* Genera traiettorie cinematicamente fattibili e prive di collisioni.
    - *Obstacle Velocity Limiter:* Limita la velocità in prossimità di ostacoli.
    - *Obstacle Stop Planner:* Inserisce un punto di arresto o una sezione di decelerazione nella traiettoria.
  - **Adaptive Cruise Controller:** Integra la velocità massima nella traiettoria in presenza di un veicolo che precede.
  - **Validation:** Assicura la sicurezza e l'appropriatezza delle traiettorie pianificate, con capacità di risposta alle emergenze.

6.6 Control

- Genera il segnale di controllo basato sulle traiettorie di riferimento dal componente Planning.
- **Moduli:**
  - **Trajectory Follower:** Genera comandi di controllo del veicolo (es. angolo di sterzata, velocità target) per seguire la traiettoria di riferimento.
  - **Vehicle Command Gate:** Filtra i comandi di controllo per prevenire valori anomali e permette lo switching tra diverse sorgenti di comando (es. MRM, controllo remoto).
- Utilizza informazioni generali (es. accelerazione/decelerazione target) e non informazioni specifiche del veicolo (es. pressione dei freni), facilitando l'integrazione e la messa a punto.

6.7 Vehicle Interface

- Gestisce le diverse interfacce del powertrain dei veicoli (angolo di sterzata, coppia sterzante, velocità, pedali acceleratore/freno, pressione freni).
- Include un modulo **Adapter** che converte tra i tipi di messaggi proprietari usati dal veicolo e i tipi generici usati da Autoware (es. da pressione freni ad accelerazione desiderata).
- Esempio di integrazione: **MiviaCar** tramite CAN bus.

7 Modern Automotive Architectures

- Modern vehicles integrate over 70 ECUs (Electronic Control Units), with premium cars exceeding 150 ECUs.
- **Critical functions** (engine control, vehicle dynamics) require **hard real-time** constraints (response times of a few milliseconds).
- Most other functionalities (infotainment) demand at least **soft real-time** behavior.

7.1 The Role of Software Architecture

- Refers to high-level structures of a software system, needed to reason about the system.
- Early-stage architecting influences hardware allocation and high-level systemization.
- Serves as a blueprint for development phases (design to testing).

7.2 The Role of a Software Architect

- Provide technology leadership and define system-wide principles.
- Establish architectural styles and guide development choices.
- Ensure consistency and adherence to principles throughout the software lifecycle.

7.3 Architecture vs Design

- **Software Architect**: Defines high-level structure, constraints, system-wide rules. (Scope: System, Hard to change).
- **Software Designer**: Implements architecture using concrete software entities. (Scope: Module, Easy to change).
- Architecture is **abstraction-focused**; Design is **implementation-focused**.

7.4 Responsibilities of a Software Architect

- Defining communication protocols and number of ECUs.
- Selecting relevant standards and ensuring compliance.
- Understanding both software and hardware for system-wide integration.

7.5 System Thinking in Software Architecture

- Architects must think holistically (software and hardware constraints).
- Focus on **why** decisions are made, not just how.
- Influence software design, cost, and project scope.

7.6 Increasing System Complexity

- Automotive systems' complexity is rapidly growing, especially with ACES (Autonomy, Convergence, Ecology, Services).
- Rising liability risks with cybersecurity and safety.
- Domains impose specific requirements: computation speed, reliability, safety, flexibility, scalability.

7.7 Historical Overview

- **1970s-1980s**: Electronics for fuel efficiency (e.g., electronic fuel injection). Software was monolithic, tied to single domains. Safety-critical aspects mostly hardware.
- **1990s**: Central computers for telemetry. Infotainment (GNSS navigation) required integration. Safety features (ABS, early ACC) raised liability questions.
- **Early 2000s**: Growth of ADAS (e.g., City Safety by Volvo). Multiple ECUs and complex interactions demanded robust software architectures. Introduction of AUTOSAR for hardware abstraction and component reuse.
- **2010s**: Vehicle-2-Vehicle (V2V) and Vehicle-2-Infrastructure (V2I) communications. Vehicles became platforms for post-production software additions (OTA updates). Google's self-driving prototypes broadened supplier ecosystem.

7.8 Trends Shaping Automotive Software Development

- **Heterogeneity**: Safety-critical vs. infotainment demands diverse design/validation.
- **Distributed Development**: OEMs and suppliers co-develop software.
- **Variant Management**: Configurable software for global markets.
- **Cost Optimization**: Low unit cost for price-sensitive markets.
- **Connectivity**: Internet-enabled cars, online services.
- **Autonomous Driving**: Complexity shifts to software validation.
- **Sensor Fusion**: LiDARs, RADARs, cameras require processing large data volumes.
- **Increased Processing Power**: Edge computing, enhanced ECUs for perception/control.

7.9 Need for Architectural Redesign

- Resulting complexity requires architectural redesign.
- V2I/V2V demand IT backbones and cloud-based solutions.
- Affects system modeling, testing, Model-in-the-Loop (MiL) simulation.

7.10 Architectural Impacts

- Integration of quality requirements (e.g., functional safety ISO 26262, ASIL).
- Use of secure communication platforms (e.g., Adaptive AUTOSAR).
- Deployment of end-to-end distributed security for remote software updates (OTA).
- Enablement of services: infotainment, online apps, remote diagnostics, emergency call.
- Direct updates at the firmware level.
- Secure OTA updates: authentication, encryption, ISO 21434 compliance.

8 02: Modern Automotive Software Architectures

Architectural Views: 4+1 View Model (Kruchten)

- **Logical view**: Object model of the design.
- **Process view**: Concurrency and synchronization.
- **Physical view**: Hardware architecture, software component mapping.
- **Development view**: Static organization of software in its development environment.
- **Scenario view** (+1): Interactions with external actors and internal component communication.

8.1 Logical View

- Decomposition of vehicle's software system, its logical architecture, and key structural elements.
- Illustrates classes, modules, components, and their interdependencies.
- Best Practices: Identify components as UML classes, define explicit relationships, iterative refinement for consistency.

8.2 Physical View

- Focuses on topology of the car's electrical system: ECUs and their connection via physical buses.
- Includes: Processing power of ECUs, OS and version for each ECU.
- Best Practices: Base on predefined physical architecture, identify key ECUs (EnCU, PSCU, TCU), organize ECUs by design/model.

8.3 Architectural Styles

- Defines high-level design principles shaping system behavior.
- Guide software design, akin to building architecture.
- Specific styles apply due to stringent reliability and robustness requirements.

8.3.1 Monolithic Architectural Style

- System as a single, large component.
- Common in small safety-critical systems (minimal communication overhead).
- Advantages: Reduces latency.
- Disadvantages: High coupling and complexity, limited scalability and maintainability.

8.3.2 Publisher-Subscriber Architectural Style

- Relies on loose coupling between publishers and subscribers.
- Common for sharing vehicle state information (e.g., speed, tire pressure). ROS2 uses this.
- Advantages: Decoupling prevents publisher overload.
- Disadvantages: Publishers lack control over information use and synchronization.

8.3.3 Layered Architectural Style

- Components organized hierarchically, top-down communication only.
- Common in AUTOSAR and microcontroller designs. Used in autonomous driving for functional separation.
- Advantages: Facilitates modular design and expertise distribution.
- Disadvantages: Unidirectional communication, restricted intra-layer interaction.

8.3.4 Component-Based Architectural Style

- Emphasizes flexible, interchangeable components with well-defined public interfaces (APIs).
- Often combined with design-by-contract.
- Common in automotive infotainment (apps developed/downloaded/integrated independently).
- Advantages: Modular reusability.

8.3.5 Microkernel Architectural Style

- Minimizes kernel responsibilities; "kernel" layer (scheduling, memory, IPC) and "application" layer (user processes, drivers).
- Automotive: Used for high-security components (reduced kernel ensures strict privilege separation). Hypervisors embody this.

8.3.6 Pipes and Filters Architectural Style

- For data-driven systems with sequential processing stages.
- Each filter (component) transforms data; pipes transfer data.
- Common in image recognition for active safety.
- Facilitates reconfiguration.
- Example: Serban et al. [SPV18] - continuous data processing, rapid control decisions, redundant channel for safe-stop. Sensor fusion can replace full hardware redundancy.

8.3.7 Example: Drive-By-Wire (DBW) System

- **Origins**: Aviation (Fly-By-Wire, 1950s), automotive (1980s).
- **Evolution**: Replaces mechanical/hydraulic linkages. Fundamental for autonomous/electric vehicles and ADAS.
- **Core Technologies**: Steer-By-Wire (SbW), Throttle-By-Wire, Brake-By-Wire.
- **Challenges**: Safety/reliability, regulatory needs, real-time performance.
- **Adoption**: EVs, Luxury cars (Tesla, BMW, Infiniti, Mercedes).
- **Electric Power Assisted Steering (EPAS)**: Intermediate step to SbW, enables electronic steering control. Key components: Torque Sensor, Assist Motor, Steering Angle Sensor, Motor Demand, Rack & Pinion.
- (Slides 39-44 show a Simulink model of a T818 drive control system as an example of Pipes and Filters for a Drive-By-Wire system, converting wheel/pedal inputs to control signals for steering, braking, throttle, etc., and managing autonomous modes.)

8.3.8 Middleware Architectural Style

- Based on a common request broker mediating resource usage (inspired by CORBA).
- AUTOSAR uses these principles via a shared meta-model.
- Encourages decoupling of services with standardized interfaces.
- Relevant for modern automotive architectures (scalable, maintainable).

8.4 Describing the Architectures

- No single unified formalism; architecture is a communication tool.
- **ADLs (Architecture Description Languages)**: Notations to describe architectures.
- Help architects clarify guiding principles and discuss implications.
- Focus here: **SysML** (Systems Modeling Language) and **EAST-ADL**.

8.4.1 SysML Overview

- General-purpose extension of UML, reusing UML symbols via profiling.
- Adds specific diagrams for system needs.
- "Block" is the core concept (software/hardware entities).
- Extends UML 2.0 but preserves familiar diagram notations (Activity, Sequence, etc.).
- **Diagram Types**: Block Definition & Internal Block, Parametric, Requirement, Behavioral (Activity, Sequence, State Machine, Use Case).

8.4.2 EAST-ADL Overview

- Architecture Description Language from ITEA EAST-EEA project, aligned with AUTOSAR.
- UML-based modeling language tailored for automotive software architectures.
- Multiple abstraction levels (similar principle to SysML).
- Closer alignment with automotive domain needs (requirements, features, hardware mapping).
- **Abstraction Levels**:
  - **Vehicle Level**: External functionality view (use-case like).
  - **Analysis Level**: Abstract functional model (features, dependencies).
  - **Design Level**: Logical architecture of software and hardware mapping.
  - **Implementation Level**: Detailed design (leveraging AUTOSAR).
  - **Operational Level**: Non-functional properties (timing, failure modes).
- **Key Differences from SysML**:
  - **Requirement Linking**: Tied to "Features" (concept not in SysML).
  - **Automotive-Specific Structuring**: Supports domain concepts (sensors, ECUs).
  - **Visual Representation**: Like UML but domain-specific elements/naming.
  - **Ease of Use**: Mirrors real automotive development processes.

9 03: AUTOSAR: Classic and Adaptive

AUTomotive Open System ARchitecture (AUTOSAR)

- Industry-standard framework initiated in 2003 by OEMs and suppliers.
- Addresses growing system complexity; now de facto standard with over 200 global partners.
- Goals: Improve interoperability, modularity, scalability; reduce costs via reusability; ensure seamless integration.
- Impact: Unified development methodology (OEMs design/verify, suppliers implement); enables OTA updates and Agile methodologies.

9.1 Evolution of AUTOSAR

- **AUTOSAR Classic (2003)**: For traditional functions with real-time constraints, static architecture.
- **AUTOSAR Adaptive (2017)**: For autonomous driving, cloud computing, dynamic updates.
- Coexistence: Adaptive complements Classic, integrates with Android, GENIVI.

9.2 AUTOSAR Classic

- Microcontroller-based (ECU).
- Layers: Application Software, Runtime Environment (RTE), Basic Software (BSW).
- Uses RTOS compliant with OSEK standards.
- Purpose: Simplify development by separating application logic, communication, hardware control.
- Deterministic Scheduling: Ensures hard real-time constraints.

Application Software Layer (Classic)

- Software Components (SWCs) implement specific functions (e.g., engine control).
- Ports and Interfaces: SWCs exchange data via well-defined ports.
- Focus: High-level vehicle behavior.

Runtime Environment (RTE) (Classic)

- Communication Abstraction: Manages data exchange between SWCs.
- ECU Independence: Hides physical distribution.
- Automatic Generation: Based on interfaces and component ports.

AUTOSAR Basic Software (BSW) (Classic)

- Provides essential system functionalities for ECU software.
- **Microcontroller Abstraction Layer (MCAL)**: Interfaces with hardware (drivers for SPI, CAN, I2C, ADC, PWM, GPIO).
- **ECU Abstraction Layer (ECUAL)**: Standardizes hardware access (CanIf, SpiHdlr, EepAbs).
- **Service Layer**: System and communication services (COM, DEM, DCM).

9.3 AUTOSAR Adaptive

- Supports high-performance computing for autonomous driving, advanced connectivity.
- Complements Classic Platform.
- Comparison with Classic:
  - **Classic**: OSEK-based OS, C language, single address space (MPU possible), real-time, fast startup.
  - **Adaptive**: POSIX-based OS, C++ language, applications separately installable with virtual address space (MMU), safety-critical, high computing power, modularity.

9.4 AUTOSAR Use Cases

Classic Use Cases

- Powertrain/Engine Control: ECUs, TCUs, hybrid/EV motor control.
- Chassis/Safety Systems: ABS, ESC, EPS, airbag control.
- Body Electronics/Comfort: Lighting, climate control, power windows, central locking.
- Basic ADAS: Parking sensors, blind-spot monitoring, rearview cameras.
- Low-Cost Domain Controllers/Gateways: CAN/FlexRay routing, LIN communication.

Adaptive Use Cases

- Advanced ADAS/Autonomous Driving: Sensor fusion, object detection, path planning, Level 3/4, V2X.
- Infotainment/Digital Cockpit: Advanced HMI, multimedia, high-res clusters, OTA for infotainment.
- Connected Vehicle Services: TCUs, cloud diagnostics, cybersecurity, smartphone integration.
- High-Performance Vehicle Computing: Ethernet, software-defined vehicles, edge computing.
- Predictive Maintenance/AI: Data collection, driver behavior analysis, smart energy management.

10 04: Automotive Communication Protocols

Intra-Vehicle Communication

- Multiple ECUs exchange data over dedicated buses.
- Key requirements: Reliability, Real-Time Performance, Scalability, Noise Immunity.

Parallel vs. Serial Communication

- **Parallel**: Multiple bits simultaneously, faster for short distances, less common.
- **Serial**: Bits sequentially, reduced wiring/complexity, preferred in embedded.

Serial Protocols: Synchronous vs. Asynchronous

- **Synchronous**: Shared clock, tighter timing, clock sensitive to noise.
- **Asynchronous**: No shared clock (start/stop bits), simpler wiring, fixed transmission rate.

10.1 Common Automotive Serial Protocols

Controller Area Network (CAN)

- Versions: CAN (Classic), CAN FD (Flexible Data-Rate), CAN XL (Extra Long).
- **CAN ISO-OSI Stack**: Defines Data-Link Layer and part of Physical Layer. Application Layer is developer-defined. CAN Controller (Data Link, part of Physical) and CAN Transceiver (rest of Physical) are separated for noise isolation.
- **Data-Link Layer**: Error-free message transfer, bit stuffing, checksums, acknowledgment. Sublayers: LLC (Logical Link Control - filtering, recovery/flow management) and MAC (Medium Access Control - framing, arbitration, acknowledgment, error detection).
- **Physical Layer**: Bit timing, encoding, synchronization. Converts logical signals (TX/RX) to differential signals (CAN_H, CAN_L) for noise immunity.
- **BUS Levels**: Recessive (logic 1, ¡0.5V differential) and Dominant (logic 0, ¿0.5V differential). Dominant bit overdrives recessive.
- **Message Frame**: SOF, Arbitration ID (priority), DLC (data length), Data Field (0-8 bytes for CAN Standard), CRC (error detection), ACK slot.
- **BUS Arbitration**: Non-destructive, bit-wise arbitration (Carrier Sense Multiple Access with Collision Detection and Arbitration on Message Priority - CSMA/CD+AMP). Lower decimal ID means higher priority.
- **Database CAN (DBC)**: Text-based file defining messages, signals, encoding/decoding scheme for ECUs.

11 05: Machine Learning in Automotive Software

Example: Image Classification for Autonomous Emergency Braking (AEB)

- ML system detects vehicle ahead, estimates distance.
- Probabilistic output:
  - False positive (distance too low) -¿ unnecessary braking -¿ rear-end collision risk.
  - False negative (distance too high) -¿ AEB doesn't apply brakes -¿ head-on collision risk.

Mitigation Through Additional Sensors and Safety Cage

- Add RADAR/LiDAR for more data, but they have their own limitations (interference, weather).
- **Safety Cage**: Non-ML deterministic component monitors sensor data, detects out-of-bound parameters, switches to safe mode with predefined values.

11.1 On-Board Training

- Vehicle sensors collect Ground Truth data, send to a powerful training ECU in the vehicle.
- New ML model version is then used. Training ECU is at the edge of architecture.
- **Advantages**: Adapts to individual driving preferences/conditions (e.g., route planning, engine parameters). Improves experience, refines operating parameters (e.g., reduce carbon footprint). No internet needed.
- **Disadvantages**: Developer lacks control over training/evaluation. Risk of local optima or performance worsening. Requires additional, more powerful/expensive ECU.

11.2 Off-Board Training

- Data sent to data center TPUs for training ML models. Updated model downloaded later (OTA).
- Distributed architecture, asynchronous communication.
- Vehicle collects data, sends to data center, which aggregates data from multiple vehicles.
- After evaluation/testing, new version distributed as software update.
- **Advantages**: More control for OEM (can discontinue updates if unsatisfactory). Allows more costly algorithms (higher TPU capacity). Less expensive TPU management. Larger, more varied datasets improve generalization.
- **Disadvantages**: Privacy/security issues for user data transfer (must be anonymized). Model is generalized ("middle ground"), not specialized per vehicle. Differences in acquisition platforms (driving profiles, vehicle types) can alter object appearance.