Task-Driven Co-Design for Resource-Efficient Autonomous Vehicles: Integrating Perception and Motion Planning

Dejan Milojevic[*†], Gioele Zardini[‡§], Miriam Elser[†], Andrea Censi[*] and Emilio Frazzoli[*]
[*]Institute for Dynamic Systems and Control, ETH Zürich, Zürich, Switzerland, {dejanmi, acensi, efrazzoli}@ethz.ch
[†]Chemical Energy Carriers and Vehicle Systems Laboratory, Empa, Dübendorf, Switzerland, miriam.elser@empa.ch
[‡]Department of Aeronautics and Astronautics, Stanford University, Stanford, CA, USA
[§]Laboratory for Information and Decision Systems, MIT, Cambridge, MA, USA, gzardini@mit.edu

***Abstract*—This paper examines the integration challenges and strategies for designing Autonomous Vehicles (AVs), with a particular focus on the task-driven, optimal selection of hardware and software to achieve a balance between safety, efficiency, and the minimal usage of resources, including costs, power consumption, computational requirements, and weight. We emphasize the interplay between perception and motion planning in decision-making by introducing the concept of occupancy queries to quantify the perception requirements for sampling-based motion planners. We propose an approach for efficient sensor and algorithm selection and placement, leveraging the perception requirements and the False Negative Rate (FNR) and False Positive Rate (FPR) to evaluate sensor and algorithm performance under various factors such as geometric relationships, object properties, sensor resolution, and environmental conditions. This forms the basis for a co-design optimization that includes the vehicle body, motion planner, perception pipeline, and computing unit. A case study on developing an AV for urban scenarios provides actionable information for designers, and shows that complex tasks escalate resource demands, with task performance affecting choices of the autonomy stack.**

I. Introduction

The advancement of AVs has enabled the commercial deployments of driverless mobility services in numerous urban areas worldwide. Nonetheless, realizing the full potential of AVs requires safe and efficient operation, which depends on effective design. Recent developments in Autonomous Driving (AD) have made various hardware and software components readily available. Thus, the challenge in developing AVs involves selecting the right combination of these interdependent components. The final design must ensure safety and efficient task performance while minimizing resources required for design and operation, such as cost, power consumption, computation, and weight.

In terms of perception, this includes selecting and placing sensors and choosing algorithms to process the sensor data. Hardware and software selections are not only interdependent but also influence other system components such as computing units, actuators, and decision-making. For instance, a controller relies on references from a motion planner, which are based on state estimates from an estimator, which, in turn, depends on sensor data and power supply. Furthermore, integrating perception software, such as object detection algorithms, brings uncertainties in algorithm outputs that must be accounted for in the design process.

To address these complex issues, a comprehensive framework that employs abstract reasoning across various areas and balances functional requirements with resource constraints and trade-offs is essential. This paper outlines our methodology in [40] for addressing the complex task of AV co-design by tackling such challenges.

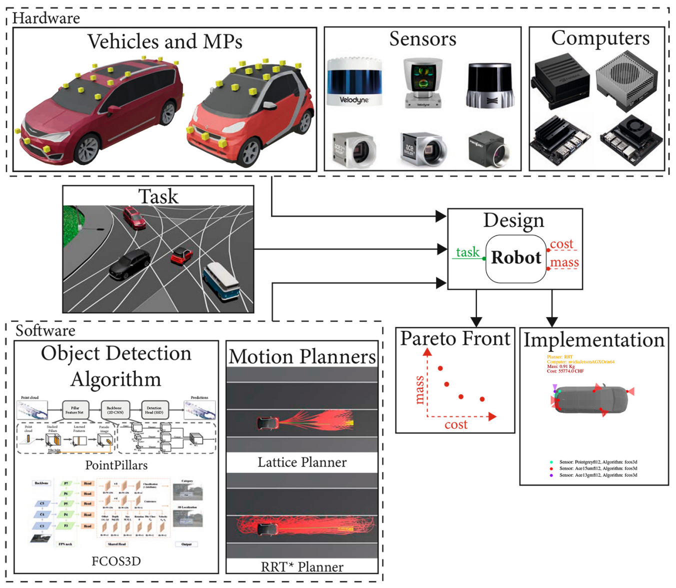


Fig. 1. Graphical illustration of the informal problem definition for designing an AV for urban driving tasks, based on a catalog of hardware and software components with an emphasis on minimizing resources.

Informal Problem Definition: The problem involves defining catalogs that contain both hardware and software components essential for the design of AVs. These include *vehicle bodies* (i.e., vehicle chassis, shape and actuators), *sensor mounting configurations*, *perception pipelines* (i.e., sensors and perception algorithms), *decision-making algorithms*, and *computing units*. Each component is associated with specific resources, including monetary cost (e.g., sensor prices), power consumption (e.g., energy requirements of computers), computational resources (e.g., flops required by perception algorithms), and mass (e.g., weight of sensors). The challenge is to co-design the AV with a particular task by selecting from these components to minimize resource usage while ensuring feasibility. A graphical illustration of the informal problem definition is depicted in Fig. 1.

Assumptions: To address the AV co-design problem, we establish the following assumptions. First, we assume that the robot software architecture is factorized into perception, state estimation, planning, and control. Second, we assume that there is an occupancy query-based interface between estimation and planning. Finally, we assume that object detections of the perception layer are binary: objects are either detected or not, based on the detection received from the perception pipelines. Our methodology remains flexible to variations in software architectures and motion planners, provided that the information needed by the robot to complete the task can be gathered, which must be provided by the perception pipeline. While our focus here is primarily on object detection, the approach could be extended to encompass other perception tasks, such as localization.

Contribution: The contributions are generally applicable to mobile robots including AVs and can be summarized as follows. First, we explore the information requirements for AVs with sampling-based motion planners via the concept of occupancy queries. Second, we show how to formulate and solve the sensor selection and placement problems for an AV, via set cover problem by using the performance requirements and perception performance benchmarks. Third, we develop a co-design framework for an AVs leveraging a monotone theory of co-design optimization, promoting the task as a functionality, and minimizing resource consumption in terms of monetary costs, power and computational needs, and mass. Finally, we illustrate the above contributions through a suite of case studies on AVs design.

Organization of the paper: Sec. II reviews the related work, and contextualizes the efforts proposed in this paper. Sec. III presents in depth our system models, including the robotic platform, tasks, decision-making, perception performance, and requirements. We then present the system co-design optimization problem, and its solution in Sec. IV, and showcase various case studies in Sec. V. Finally, we conclude and provide an outlook for future research in Sec. VI.

II. Related Work

The challenges of design automation for embodied intelligence, such as AVs, are highlighted in several studies [52, 69, 35, 42]. These challenges primarily revolve around developing a framework that can accommodate the complex nature of cyber-physical systems [52, 69], navigating the heterogeneous design landscapes including hardware and software [69], the difficulty of integrating diverse components into robotic systems, and determining the specific information needs for a robot to fulfill a given task [42]. In the following, we review the literature in this field, mainly focusing on sensor selection and its relevance to robotics, design space exploration, comparative analysis of methods and trade-offs, benchmarks, and co-design frameworks.

The challenge of selecting and positioning sensors within a system is complex, and often lacks a closed-form solution. Despite the various contributions in [28, 20, 25, 58, 12, 13, 19, 18], current literature does not fully address the integrated selection and placement of sensor hardware in conjunction with the choice of perception algorithms, especially for object detection tasks, with a particular focus on contemporary deep learning techniques. Furthermore, the critical exploration of sensing requirements for bridging decision-making and perception is underrepresented. The analyzed studies also overlook the necessity for methodologies that unlock seamless integration with other design considerations, such as computer and actuator selection.

Design space exploration has gathered substantial interest in robotics research, with significant contributions in studies such as [24, 49, 68, 41] aiming to delineate the boundaries of sensor and actuator requirements for effective robotic planning. Comparative analyses of robotic components have been advanced through the works of O'Kane, Lavalle, and Censi [44, 34, 8], which explore methodologies for assessing sensor performance and establishing criteria for comparisons. Trade-off analysis in design choices is examined in contributions such as [31, 51, 50].

From the point of view of holistic co-design frameworks, significant advancements have been made in robot design methodologies encompassing both software and hardware elements, facilitated by high-level behavioral specifications. Mehta introduced an approach utilizing linear temporal logic to transform high-level design specifications into tangible selections of robot components from an extensive library [39]. Furthermore, [26] develops a heuristic algorithm specifically targeted at the creation of robotic devices tailored to follow predefined motion trajectories accurately. Similarly, [53] explores the optimization of robotic design by carefully selecting actuation and sensing hardware to minimize design costs while ensuring the robot's ability to execute plans and accomplishing tasks.

The methods previously discussed do not focus on fully automating the design process for an entire robotic system and They overlook several critical co-design challenges, as identified in [63, 52, 69, 35, 42], such as a) formalizing heterogeneous components across varying levels of abstraction, b) composition heterogeneous components to allow co-design across the entire system, c) facilitating collaboration among different systems as well as their domain experts, d) ensuring computational tractability, which allows quantitative design solutions, e) accommodating continuous systems that evolve over time, and f) maintaining intellectual tractability for simple usage and understanding. Our research is based on the monotone theory of co-design [7, 9] and builds on our series of previous works [40, 65, 66, 67], where we studied the co-design of autonomy in the context of AVs and mobility. In the current work, we advance our methodology by modeling each component separately and fostering compositional interconnections, particularly between the perception and the decision-making processes of a robot.

III. System Modeling

A. Modeling the robotic platform

We consider a mobile robot $\mathcal{R}$, defined by its physical body $\mathcal{B}$ with configuration space Γ, and its software, the agent, which we call $\mathcal{A}$.

Agent: We assume that the agent $\mathcal{A}$ consists of a modular software architecture, comprising perception, state estimation, motion planning, and control [47]. In particular, the control function is predicated on a reference trajectory formulated by a motion planner, which itself is based on state estimates from an estimator. The estimator's accuracy relies on the sensor data gathered and processed by the perception system. We want to choose the planner and the perception system for the agent.

Body: The robot body $\mathcal{B}$ encompasses hardware components, including its 3D shape and actuators. We define the robot's body as follows.

Definition 1 (Body)**.** A robot body $\mathcal{B}$ is defined by a tuple

$$\mathcal{B} := \langle \mathrm{SH}, \Gamma, \mathcal{U}, \mathrm{dyn}, \mathrm{HW} \rangle,$$

where $\mathrm{SH} \subset \mathbb{R}^3$ represents the physical 3D shape of the robot, Γ denotes the configuration space, $\mathcal{U}$ refers to the control space, and the dynamics[1] are expressed as $\dot{x}_t := \mathrm{dyn}(x_t, u_t)$, with u_t being the control input and x_t the state at time $t \in \mathbb{R}_{\geq 0}$. The state $x \in \mathcal{X}$, where the state space is defined as $\mathcal{X} := \Gamma \times \mathcal{H}$. All additional hardware components and robot's body appearance, such as actuators, batteries, color, material, etc., are captured in the hardware tuple HW.

The function $\mathrm{sh}_{\mathcal{R}} \colon \mathtt{POW}(\Gamma) \to \mathtt{POW}(\mathbb{R}^2)$ converts a robot configuration into its footprint.

The examination of the robot's structural framework $\mathcal{B}$ involves assessing its mounting positions mp (with $\mathrm{mp} \in \mathrm{MP}$ and $\mathrm{MP} \subset \mathrm{SH}$), as well as the selection of sensors. The sensor hardware with the related perception algorithm is referred to as a "perception pipeline" pp. In particular, our analysis focuses on 3D object detection to demonstrate the perception pipeline's ability to detect objects in the environment. The collection of all perception pipelines is denoted by PP. Furthermore, we evaluate sensor mounting orientations $\mathrm{mo} \in \mathrm{MO}$, characterized by sensor yaw and pitch angles, such that $\mathrm{MO} \subseteq \mathbb{R}^2$. These aspects together form the specification of the robot's body.

Definition 2 (Robot)**.** A robot $\mathcal{R}$ is a tuple consisting of an agent $\mathcal{A}$ and body $\mathcal{B}$: $\mathcal{R} := \langle \mathcal{A}, \mathcal{B} \rangle$.

B. Modeling a task

We define the task as a set of scenario instances. In principle, however, a task could also be defined as a distribution of scenarios, where a set of scenarios can be sampled.

Definition 3 (Task)**.** A *task* $\mathcal{T}$ is a set of scenario instances.

A scenario instance

$$S = \langle \mathcal{W}, \gamma_{\mathrm{start}}, \mathcal{G}, \mathrm{env}, \{C_i\}_{i \in \{1,\ldots,M\}} \rangle$$

represents a concrete realization of a scenario $\mathcal{S}$ (see Def. 16), where the initial configuration $\gamma_{\mathrm{start}} \in \Gamma$, goal $\mathcal{G} \in \mathbb{R}^2$, and environment $\mathrm{env} \in \mathbb{E}$ are drawn from their respective distributions given by the scenario. The operational environment env encompasses various weather and light conditions. Moreover, M number of object class instances C_i are drawn from the corresponding Poisson distributions. An instance of a class $\mathcal{C}_i$ is defined as a tuple $C_i = \langle \mathcal{Q}_i, \mathcal{U}_i, \mathrm{dyn}_i, \mathrm{appear}_i \rangle$, where a particular appearance appear_i is drawn from f_{AP}.

Definition 4 (Object class)**.** An *object class* $\mathcal{C}$ is a tuple

$$\mathcal{C} := \langle \mathcal{Q}, \mathcal{U}, \mathrm{dyn}, f_{\mathrm{AP}} \rangle,$$

where $\mathcal{Q}$ is the configuration space and $\mathcal{U}$ is the control space for the class. The dynamics[1] are defined by $\dot{x}_t = \mathrm{dyn}(x_t, u_t)$ with $u_t \in \mathcal{U}$ being the control input and $x_t \in \mathcal{X}$ the state at time t, where $\mathcal{X} := \mathcal{Q} \times \mathcal{H}$. The appearance of a class is represented by a tuple comprising elements such as shape, color, material, etc., denoted as appear. The set of all possible appearances is represented by AP, with the appearance distribution of a class given by $\mathrm{AP} \sim f_{\mathrm{AP}}(\mathrm{appear})$.

The function $\mathrm{sh}_i \colon \mathtt{POW}(\mathcal{Q}_i) \to \mathtt{POW}(\mathbb{R}^2)$ maps a class configuration into the footprint projecting the 3D shape of the class's appearance onto the ground plane.

Prior: Moreover, for each object class in each scenario we have given the *prior knowledge* $\mathcal{P}$ (see Def. 15), which represents the class configurations, such that $\mathcal{P} \subseteq \pi_3(\mathcal{C}) = \mathcal{Q}$ for a particular class. This prior knowledge essentially outlines the allowed configurations for objects of the specific class in a scenario.

C. Modeling an agent

Motion planning algorithms typically need a notion of the obstacle free configuration space to compute a reference trajectory. Combinatorial motion planning [33, 47, 10, 56, 2] and optimization-based motion planning [47, 11, 22, 21, 30, 48, 62, 36] depend on mathematical models for the free configuration space, represented through geometric shapes or optimization constraints. The task of pinpointing the critical information necessary for calculating a reference trajectory is notably challenging in these frameworks, mainly because they require knowledge of the entire state space including all obstacles. In contrast, sampling-based planners [33, 47, 11] offer a different strategy, sidestepping the need for precise internal representations of obstacles. Such planners generate a state hypothesis by posing a series of questions, such as "*Will there be a collision if I occupy a certain configuration at a certain time?*". These questions are referred to as *occupancy queries* or just *queries* and are represented as elements of the configuration space Γ at a certain time t with a certain environment env. Sampling-based planners thus enable a reverse flow of information within the outlined agent architecture, indicating a progression of data from the motion planning phase back to the perception system. For the sake of simplicity, the term agent throughout the remainder of this paper denotes a sampling-based motion planner.

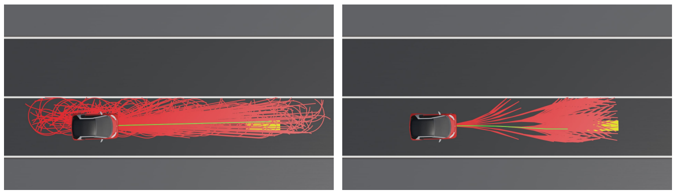
Definition 5 (Query)**.** A *query* is defined as $\psi \in \Psi$, where Ψ is the product space of the configuration space Γ, the time in $\mathbb{R}^+$ and the environment in $\mathbb{E}$: $\Psi := \Gamma \times \mathbb{R}^+ \times \mathbb{E}$.

Different motion planners produce different distributions of queries. Planners such as RRT*, as visualize in Fig. 2a, converge to an optimal solution. However, during the search for the optimal solution, random configurations are sampled, leading to more information requirements for the sensors. Lattice planners, as visualized in Fig. 2b, on the other hand, are not optimal, but by simply discretizing the search space with motion primitives, less information is required from the sensors compared to RRT*.

Given an agent $\mathcal{A}$ and a task $\mathcal{T}$, the goal is to obtain a set of configurations which are generated by the agent's state inference process, motivated by the concept of deterministic sampling-based motion planning in [29].

Definition 6 (Task Queries)**.** Given a task $\mathcal{T}$, the *task queries* generated by an agent $\mathcal{A}$ are the union over all queries of all

[1]Without loss of generality, the dynamics can be stochastic.



(a) Example of an RRT* planner. (b) Example of a lattice planner.

Fig. 2. An illustration of an AV navigating towards the yellow target area. The figure showcases two motion planners: an RRT*-based planner and a lattice planner. The red lines represent the tree of paths generated by each planner, while the green line indicates the solution path identified by the planner.

the scenario instances in the task:

$$\begin{aligned}\mathrm{tq}\colon \mathtt{POW}(\mathbb{T}) \times \mathbb{A} &\to \mathtt{POW}(\Psi),\\ \langle \mathcal{T}, \mathcal{A}\rangle &\mapsto \bigcup_{S\in\mathcal{T}} \mathrm{plan}(\mathcal{A}, S),\end{aligned}$$

such that $\mathrm{tq}(\mathcal{A},\mathcal{T}) \subseteq \Psi$. The function plan maps an agent $\mathcal{A}$ and a scenario instance S to a set of queries: $\mathrm{plan}\colon \mathbb{A}\times\mathbb{T}\to \mathtt{POW}(\Psi)$.

D. *Modeling perception performance and requirements*

The next step is to evaluate the capabilities of a perception pipeline, including sensor hardware and perception software, to measure and provide the information needed by an agent. The perception pipeline detection capabilities are denoted as *perception performance* and are represented in terms of FPR and FNR for a certain object class instance C_i with a certain class configuration in $\mathcal{Q}_i$ and appearance appear_i. For each perception pipeline pp_j and each object class instance C_i, the FNR and FPR functions are defined as:

$$\begin{aligned}\mathrm{fnr}\colon \mathcal{Q}_i \times \mathrm{AP}_i \times \mathrm{PP} \times \mathbb{E} &\to \mathtt{POW}(I),\\ \langle q_i, \mathrm{appear}_i, \mathrm{pp}_j, \mathrm{env}\rangle &\mapsto [a,b].\end{aligned}$$

$$\begin{aligned}\mathrm{fpr}\colon \mathcal{Q}_i \times \mathrm{AP}_i \times \mathrm{PP} \times \mathbb{E} &\to \mathtt{POW}(I),\\ \langle q_i, \mathrm{appear}_i, \mathrm{pp}_j, \mathrm{env}\rangle &\mapsto [a,b].\end{aligned}$$

The set I is the set of all intervals: $[a,b]\subseteq\mathbb{R} : 0\le a\le b\le 1$. The obtained interval $[a,b]$ represents the confidence interval with a lower bound a and upper bound b of the perception pipeline's FNR and FPR. During our selection process, we use the upper bound to conduct a *worst-case analysis*. With the variables appear, pp and env we summarize other relevant parameters for representing the FNR and FPR as for instance the object size, object color, sensor resolution or weather condition. The implementation of the FNR and FPR is not the focus of this work. An illustration of the perception performance is shown in Fig. 3.

Task queries constitute a subset of the robot's configuration space. On the other hand, perception performance relies on the configuration of object classes. In order to establish an interface between task queries and perception performance, the former are converted into *class configurations* which need to be detected by the perception pipelines. Such class configurations are referred to as *perception requirements*.

The transition from queries to class configuration involves determining which class configurations may collide with the robot at a specific query. At a more abstract level, the objective is to identify all class configurations for which the perception

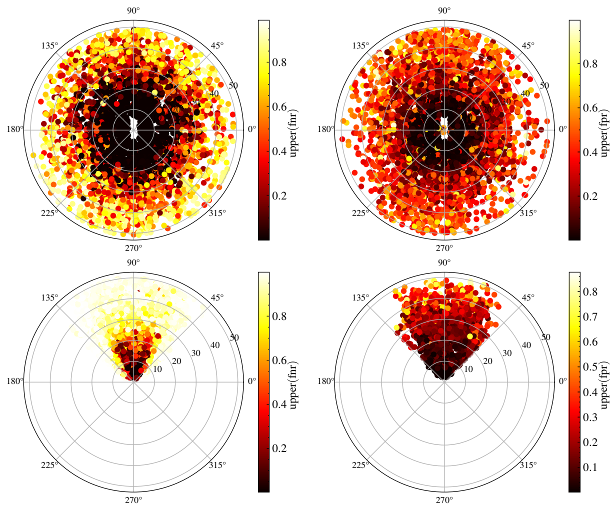


Fig. 3. Comparison of the perception performance of two pipelines: Velodyne HDL32E lidar with PointPillars detection model (top plots) [32] and Basler acA1600-60gc camera with FCOS3D detection model (bottom plots) [61]. Left plots show FNRs and right plots FPRs, highlighting the upper bounds of confidence intervals against radial distance r and relative orientation θ between sensor and object class in polar coordinates. Data is from the nuScenes dataset [4], using models from the MMDetection3D [14] library.

pipelines must indicate a collision, when posed with the query. It is important to emphasize that we are looking at agents which can ask for some occupancy queries $\psi = \langle\gamma, t, \mathrm{env}\rangle$ in the future. It is not just a simple matter of checking which class configurations could collide with the robot at a certain configuration γ. All class configurations at time 0 that would lead to a collision at time t are needed, given the dynamics of the classes and the class prior $\mathcal{P}_i$ provided by the scenario. Therefore, the objective is to derive the set of configurations for all classes within the scenario at time 0, where there exist control inputs that could lead the class to a collision with the robot with configuration γ at time t. Such class configurations are obtained by sampling the dynamics and going backwards in time. In essence, all configurations generated through the sampling are the ones that the perception layer needs to detect. An illustration of this process from queries to perception requirements is shown in Fig. 4.

The following definitions are used to define the perception requirements of a certain task for a given agent.

Definition 7 (Collision)**.** Collision is a mapping that generates all possible class configurations in $\mathcal{Q}_i$ that are in collision with the robot at a certain configuration γ using their footprints $\mathrm{sh}_{\mathcal{R}}(\gamma)$ and $\mathrm{sh}_i(q_i)$.

$$\begin{aligned}\mathrm{collision}\colon \Gamma\times\mathbb{C} &\to \mathtt{POW}(\mathcal{Q}),\\ \langle\gamma, C_i\rangle &\mapsto \Theta_i,\end{aligned}$$

where $\mathbb{C}$ is the set of all class instances and $\Theta_i \subseteq \mathcal{Q}_i \subseteq \mathcal{Q}$.

Definition 8 (Class Trajectory)**.** A class *trajectory* is defined as $\bar{q}_i \in \bar{\mathcal{Q}}_i$, where $\bar{\mathcal{Q}}_i$ is the product space of the class configuration space $\mathcal{Q}_i$ and its power set. Thereby, $\bar{q}_i = \langle q_i, \Theta_i\rangle$, where q_i is the start configuration of the trajectory such that $q_i \in \Theta_i$, where Θ_i contains all the configurations of the trajectory: $\bar{\mathcal{Q}}_i := \mathcal{Q}_i \times \mathtt{POW}(\mathcal{Q}_i)$.

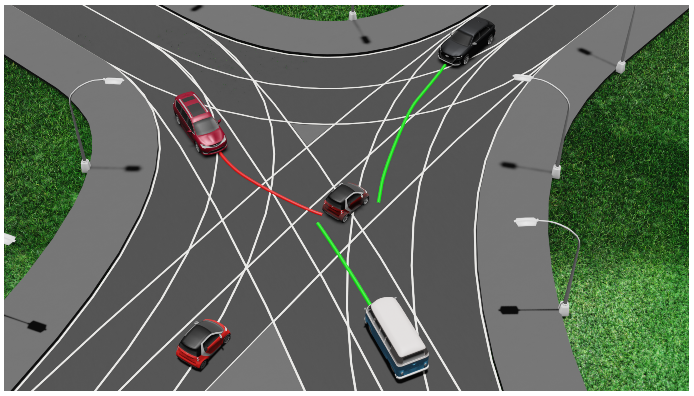


Fig. 4. This figure shows class configurations at time 0 leading to potential collisions with a robot at a specific query $\psi = \langle \gamma, t, \text{env} \rangle$. The robot is depicted as a small red AV on the left and the robot's future configuration γ from the query is the transparent AV in the intersection's center. Surrounding cars represent classes with trajectories that lead to a collision with the AV at time t with configuration γ. Green lines show feasible trajectories based on prior knowledge, and a red line shows an infeasible trajectory that violates the prior. The perception requirements in this example are the depicted car configurations with green trajectories.

Definition 9 (Perceptual Collision Prediction)**.** For each query $\psi = \langle \gamma, \tau, \text{env} \rangle$ of an agent $\mathcal{A}$, there exist class trajectories in $\bar{\mathcal{Q}}_i$ that are the preimage of the class dynamics dyn_i. These trajectories lead to one of the class configurations in $\text{collision}(\gamma, C_i) \subseteq \mathcal{Q}_i$ while starting at time $-\tau$. This mapping is termed *perceptual collision prediction*

$$\begin{aligned} \text{pcp}\colon \texttt{POW}(\Psi) \times \mathbb{C} &\to \texttt{POW}(\bar{\mathcal{Q}}), \\ \langle \Psi_0, C_i \rangle &\mapsto \bar{\Theta}_i, \end{aligned}$$

where $\Psi_0 \subseteq \Psi$ and $\bar{\Theta}_i \subseteq \bar{\mathcal{Q}}_i \subseteq \bar{\mathcal{Q}}$.

Definition 10 (Prior check)**.** The prior check is a map that takes all start configurations q_i of trajectories $\langle q_i, \Theta_i \rangle \in \bar{\mathcal{Q}}_i$, which class configurations Θ_i are a subset of the prior $\mathcal{P}_i$.

$$\begin{aligned} \text{priorcheck}\colon \texttt{POW}(\bar{\mathcal{Q}}_i) \times \texttt{POW}(\mathcal{Q}_i) &\to \texttt{POW}(\mathcal{Q}_i), \\ \langle \bar{\Theta}_i, \mathcal{P}_i \rangle &\mapsto \Theta_i, \end{aligned}$$

where $\bar{\Theta}_i \subseteq \bar{\mathcal{Q}}_i$ and $\Theta_i \subseteq \mathcal{Q}_i$.

Definition 11 (Task Perception Requirements)**.** The perception requirements for an agent $\mathcal{A}$ undertaking a task $\mathcal{T}$ are defined as the mapping from task queries $\text{tq}(\mathcal{A}, \mathcal{T})$ to all possible subsets of class configurations for each environment env within the task. This mapping is established by mapping the queries into class trajectories with perceptual collision prediction pcp and filtering the feasible start configurations from the trajectories with priorcheck:

$$\text{PR}\colon \mathbb{A} \times \texttt{POW}(\mathbb{T}) \to \prod_{\text{env} \in \mathbb{E}} \prod_{k \in \{1, \ldots \text{K}_{\text{class}}\}} \texttt{POW}(\mathcal{Q}_k),$$

where K_{class} is the number of unique object class instances in the task and $\mathbb{E}$ is the set of all environments in the task.

For a given object class instance C_i and environment env within task perception requirement $\text{PR}(\mathcal{A}, \mathcal{T})$, we express this as $\text{PR}(\mathcal{A}, \mathcal{T}, C_i, \text{env})$, such that $\text{PR}(\mathcal{A}, \mathcal{T}, C_i, \text{env}) \subseteq \mathcal{Q}_i$.

IV. Solving the robot co-design problem

In this chapter, we establish an optimization framework for determining the optimal robot design tailored to a specific task, leveraging a monotone theory of co-design [6, 9]. The primary objective is the minimization of resource consumption, which includes power consumption, robot body mass, cost and computing resources. Appendix B introduces the basic principles of co-design. The fundamental unit of the co-design theory is the concept of a Monotone Design Problem with Implementation (MDPI), as detailed in Def. 17. This entity essentially translates implementations (design choices) into specific functionalities and resources, which are represented as Partially Ordered Sets (posets). We will provide a brief introduction to all the MDPIs of our robot co-design problem, which is diagrammatically represented in Fig. 9. The MDPIs are represented as blocks with green wires on the left for functionalities and dashed red ones on the right for resources. More details about the different MDPIs are provided in Appendices C and D. In Sects. IV-A and IV-B we address task-oriented co-design of a complete mobile robot. Sec. IV-B addresses the sensor selection and placement problem, which forms the core of the entire optimization, using the formulations introduced in Sec. III.

A. Modeling from task to perception requirements

First, the **Planner** MDPI represents the choice of motion planner for the robot. It provides a set of scenario instances representing the task $\mathcal{T}$ as a functionality and the average speed in km/h the planner navigates the AV across all scenario instances, indicating the task performance. The Planner MDPI requires occupancy queries Ψ, compute and the robot's dynamics resources. The more scenario instances are required, the more queries are needed by the planner. The compute resource encompasses computational capabilities, including CPU and GPU performance, quantified by operations per second and available memory. An increase in collision checks for occupancy queries leads to a higher demand for compute resources. Robot's dynamics are characterized by parameters such as minimum turning radius, maximum acceleration, and maximum deceleration.

The **Perceptual Collision Prediction** MDPI describes the pcp function to determine all potential feasible class trajectories $\bar{\mathcal{Q}}$ that could result in collisions with the robot at the occupancy queries from the planner. This guides the perception system to focus on critical areas based on the occupancy queries and the class dynamics. Consequently, the occupancy queries Ψ, class dynamics and class footprint$^{\text{op}}$ serve as functionalities of this MDPI. The class dynamics, including minimum turning radius, maximum acceleration, and deceleration, are specified similarly to the robot's dynamics.

The **Prior Check** MDPI describes the priorcheck function as outlined in Def. 10. The function priorcheck takes all start configurations from the class trajectories, which trajectory configurations are all in the prior $\mathcal{P}$ of the class. Thus, the functionalities are the class configurations prior, where classes can be in the scenario instance, and the class trajectories $\bar{\mathcal{Q}}$ generated by pcp. The resources are the final perception requirements PR.

The **Robot body** MDPI encompasses the characteristics of the robot body $\mathcal{B}$, such as the robot's dynamics, sensor mounting configurations, the robot's footprint, the maximum payload mass capacity, the auxilary power capability, and driving range. Requirements for this MDPI include the robot's shape SH in $\mathbb{R}^3$, associated with fixed costs in CHF and operational costs in CHF/m. The **Computing** MDPI implements the computing units necessary for the robot's software operations, including both motion planning and perception. It provides computational capabilities as a functionality in terms of CPU and GPU performance, measured in memory capacity and operations per second. These computational capabilities are encapsulated as compute. The provision of compute is directly linked to associated cost in CHF, mass in kg and power consumption in W.

B. Sensor selection and placement problem

This section introduces a methodology to obtain the relationship between perception pipelines and perception requirements for a particular task, while accounting for resource consumption. Employing a worst-case approach, this study assumes the absence of filters that account for historical detection data. This premise necessitates that for a perception pipeline to accurately respond to occupancy queries, its FNR and FPR must not exceed a predefined threshold ϵ. Accordingly, this assumption ensures that the identification of class configurations from perception requirements is not influenced by temporal factors. Thus, all class configurations for which the upper bound from the fnr and fpr functions is dominated by the threshold ϵ are considered covered or detectable by the perception pipeline pp_j:

$$\begin{aligned}\{q_i \in \mathcal{Q}_i :& \mathrm{up}(\mathrm{fnr}(q_i, \mathrm{appear}_i, \mathrm{pp}_j, \mathrm{env})) \le \epsilon \\ &\wedge \mathrm{up}(\mathrm{fpr}(q_i, \mathrm{appear}_i, \mathrm{pp}_j, \mathrm{env})) \le \epsilon\},\end{aligned}$$

where up takes the upper bound of an interval. This set of class configurations which can be seen by a perception pipeline depend on the mounting configuration on the robot body as well as the robot body shape itself. The reason is that different mounting configurations will have different relative class configurations to the perception pipeline. Moreover, depending on the mounting configuration on the robot, the shape of the robot could block the sensor Field of View (FoV).

We call a perception pipeline with a mounting position on a robot body and some yaw and pitch mounting orientation as *mounted perception pipeline*.

Definition 12 (Mounted Perception Pipeline)**.** Given a perception pipeline pp, a robot body $\mathcal{B}$, a mounting position of a sensor mp on the body, and the yaw and pitch angle of the sensor mounted on the robot mo, a mounted perception pipeline is a tuple containing the perception pipeline, the robot body, the mounting position and the mounting orientation: $\mathrm{mpp} = \langle \mathrm{pp}, \mathcal{B}, \mathrm{mp}, \mathrm{mo}\rangle$.

The following map is defined, which yields all the class configurations visible to a mounted perception pipeline, considering a specified threshold.

Definition 13 (Mounted Perception Pipeline Class Coverage)**.** Consider a mounted perception pipeline mpp characterized by its perception performance fnr and fpr, a target class instance C, an environment env and a threshold ϵ. The set of class configuration which can be detected by the mounted perception pipeline are defined as

$$\begin{aligned}\mathrm{mppcc}\colon \mathbb{C} \times \mathbb{MPP} \times \mathbb{E} \times \mathbb{R}_{[0,1]} &\to \mathtt{POW}(\mathcal{Q}),\\ \langle C_i, \mathrm{mpp}_j, \mathrm{env}, \epsilon\rangle &\mapsto \Theta_i,\end{aligned}$$

where $\mathbb{MPP}$ is the set of all mounted perception pipelines and Θ_i is a subset of the class configuration set $\mathcal{Q}_i$.

Collections of mounted perception pipelines class coverage for some given $\mathrm{K}_{\text{class}}$ object class instances, M_{env} environments and L_{mpp} mounted perception pipelines are given as

$$\mathrm{MPPC} = \bigcup_{k=1}^{\mathrm{K}_{\text{class}}} \bigcup_{l=1}^{\mathrm{L}_{\text{mpp}}} \bigcup_{m=1}^{\mathrm{M}_{\text{env}}} \mathrm{mppcc}(\mathcal{C}_k, \mathrm{mpp}_l, \mathrm{env}_m, \epsilon).$$

Definition 14 (Sensor selection and placement problem)**.** Consider a task $\mathcal{T}$, an agent $\mathcal{A}$, a body $\mathcal{B}$ with mounting positions MP, perception pipelines PP, mounting orientations MO and a detection threshold ϵ. The task involves $\mathrm{K}_{\text{class}}$ unique number of object class instances and M_{env} number of environments. From the body, perception pipelines and mounting orientation, L_{mpp} number of mounted perception pipelines mpp can be generated. This leads to the task perception requirement $\mathrm{PR}(\mathcal{A}, \mathcal{T})$ and a set MPPC of collections of mounted perception pipelines class coverage with $\mathrm{mppcc}(C_k, \mathrm{mpp}_l, \mathrm{env}_m, \epsilon) \subseteq \mathcal{Q}_k$, and W cost functions $c_w : \mathrm{mpp}_l \to \mathbb{R}_{>0}$. The problem is to identify $\mathrm{MPP} \subseteq \{\mathrm{mpp}_i\}_{i\in\{1,\ldots,\mathrm{L}_{\text{mpp}}\}}$ with the minimum total cost over all cost functions. The subset MPP must cover each element in $\mathrm{PR}(\mathcal{A}, \mathcal{T})$ with a matching $\mathrm{mppcc}(C_k, \mathrm{mpp}, \mathrm{env}_m, \epsilon)$, specific to the same C_k and env_m within $\mathrm{PR}(\mathcal{A}, \mathcal{T}, C_k, \mathrm{env}_m) \subseteq \mathcal{Q}_k$. Furthermore, each $\mathrm{mpp} \in \mathrm{MPP}$ must occupy a unique mounting position mp. The problem is outlined in Eq. (1), employing a binary vector x composed of elements $x_i \in \{0, 1\}$, each denoting a decision variable. Here, $x_i = 1$ signifies the selection of the mounted perception pipeline mpp_i. An indicator function emp is introduced to map a class configuration set to an empty set whenever the associated binary variable $x_i = 0$: Matrix F indicates which mounted perception pipelines share the same mounting positions. In a given row of F, all entries set to 1 signify mounted perception pipelines with identical mounting positions.

$$\begin{aligned}\min \quad & \sum_{i=1}^{\mathrm{L}_{\text{mpp}}} \sum_{j=1}^{W} w_j c_j(\mathrm{mpp}_i) \cdot x_i \\ \text{s.t.} \quad & \mathrm{PR}(\mathcal{A}, \mathcal{T}, C_k, \mathrm{env}_m) \subseteq \\ & \bigcup_{l=1}^{\mathrm{L}_{\text{mpp}}} \mathrm{emp}(\mathrm{mppcc}(C_k, \mathrm{mpp}_l, \mathrm{env}_m, \epsilon), x_l) \\ & \forall k \in \{1, \ldots, \mathrm{K}_{\text{class}}\}, m \in \{1, \ldots, \mathrm{M}_{\text{env}}\}, \\ & F \cdot x \le [1 \quad \ldots \quad 1]^{\mathrm{T}} \quad , \\ & x_i \le 1 \quad \forall i \in \{1, \ldots, \mathrm{L}_{\text{mpp}}\}, \\ & x_i \in \mathbb{N}_0 \quad \forall i \in \{1, \ldots, \mathrm{L}_{\text{mpp}}\}, \\ & \sum_{j=1}^{W} w_j = 1, \; w_j \ge 0, \; j = 1, \ldots, W.\end{aligned} \tag{1}$$

The union of all class configurations detectable by the selected mounted perception pipelines, represented as MPP, across all classes and environmental conditions is denoted as *perception coverage*:

$$\mathrm{PR}(\mathcal{A},\mathcal{T}) \subseteq \bigcup_{k=1}^{\mathrm{K_{class}}} \bigcup_{\mathrm{mpp}\in\mathrm{MPP}} \bigcup_{m=1}^{\mathrm{M_{env}}} \mathrm{mppcc}(C_k, \mathrm{mpp}, \mathrm{env}_m, \epsilon).$$

The nature of Def. 14 closely resembles the weighted set cover problem [59], since it also tries to cover a given set by a collection of subsets while minimizing a cost function. In Appendix E we show how to formulate Def. 14 as a weighted set cover problem and solve it using Integer Linear Programming (ILP). An overview of the whole sensor selection and placement process is shown in Fig. 19. Finally, we can formulate the **Sensor Selection and Placement** MDPI. It implements the sensor selection and placement problem from Def. 14 and it is the composition of the following MDPIs. The **Coverage** MDPI focuses on meeting the robot's perception requirements as a functionality by ensuring sufficient perception coverage as a resource, which includes the ability to detect necessary class configurations to accomplish the task. An increase in perception requirements directly necessitates an enhancement in perception coverage.

The **Mounted Perception Pipelines** MDPI implements the selection and positioning of perception pipelines on the robot to cover all perception requirements, thus ensuring perception coverage, considering all class appearances appear within the task, and accommodating the robot's shape SH. This MDPI requires a set of mounting configurations in $\mathrm{SE}(3)$, a set of perception performance quantified by the upper limits of fnr and fpr functions, and the robot's footprint $\mathrm{sh}_{\mathcal{R}}$.

The **Perception Pipelines** MDPI outlines the implementation of available perception pipelines, encompassing both sensors and perception algorithms. It delivers perception performance as its functionality, demanding cost in CHF, mass in kg, power in W, and compute as resources. The monotonic relationship indicates that enhancing perception performance, aiming for lower FNR and FPR, requires the employment of pricier, high-resolution sensors which generally consume more power and are heavier. Alternatively, it might involve leveraging more complex perception algorithms that demand substantial computational power.

V. Design of Experiments and Results

A. Design of experiments

The components available for design are reported in Tab. II in Appendix F. The urban driving task contains 205 driving scenarios from the CommonRoad library [37], featuring five different vehicle classes. The objective, $\mathcal{G}$, is for the autonomous vehicle to reach a designated area. We analyze scenarios with two nominal speeds: $30\,\mathrm{km/h}$ and $50\,\mathrm{km/h}$. We ran experiments with fewer scenarios to examine how task complexity affects the AV design. Additionally, the experiments varied the task prior assuming no cars can approach the AV from left/rear.

B. Results

We solve the presented co-design problem by fixing selected scenarios, and showing the corresponding Pareto fronts of minimal resources, as illustrated in Fig. 5. The figure shows that more resources are required for more complex tasks. Each task's complexity is represented by the number of scenarios, with simpler tasks as subsets of more complex ones. The upper figure compares price (CHF) on the x-axis against mass (kg) on the y-axis. Red dots indicate optimal solutions within each task, with the surrounding red area highlighting the feasible resource range (i.e., the upper sets of resources). Annotations with capital letters point to the implementations, detailed in the lower sub-figures. We show the top view of the selected vehicle, with cameras marked with dots and lidars with squares to illustrate their mounting positions. Camera orientations are further highlighted by small triangles indicating the initial FoV and yaw direction, providing an indication of their potential coverage area. Each perception pipeline is color-coded. In addition, the graphics show the selected motion planner and computing unit.

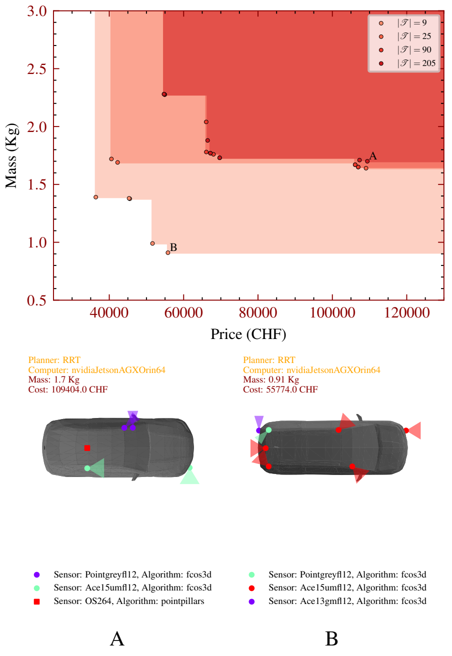


Fig. 5. Pareto front of price and mass across tasks, where tasks with more scenarios demand more resources and encompass those with fewer scenarios. Implementations for points A and B are visualized vertically. A and B indicate the lowest mass for the most and least complex tasks, respectively.

The impact of more resource requirements for the AV design by increasing the nominal speed from $30\,\mathrm{km/h}$ to $50\,\mathrm{km/h}$ within identical task scenarios is visualized in Fig. 6, where we show the Pareto fronts for price (CHF) against computation (Gflops) as well as the corresponding implementations for the different resources.

Fig. 7 demonstrates how restricting car configurations prior within identical task scenarios leads to lower resource requirements, where the Pareto fronts for price (CHF) against power consumption (W), along with implementations are illustrated.

For the aforementioned solutions for various tasks, we queried for the least resources by setting the average speed functionality requirement to just above zero. Thereby, the RRT* motion planner was consistently not selected. Conversely, when examining tasks by requiring higher average speeds (e.g., $24\,\mathrm{km/h}$), as illustrated in Fig. 8 for computation impacts, it becomes evident that the resource demands increase

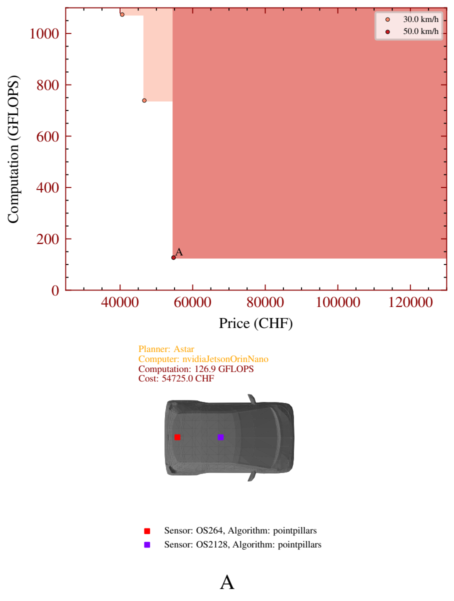


Fig. 6. Pareto front of price and computation across task velocities, where higher velocities for the same set of scenarios require more resources. Implementations for marked point A are visualized vertically. A indicate lowest computation for $50\,\mathrm{km/h}$ and 30 kmh.

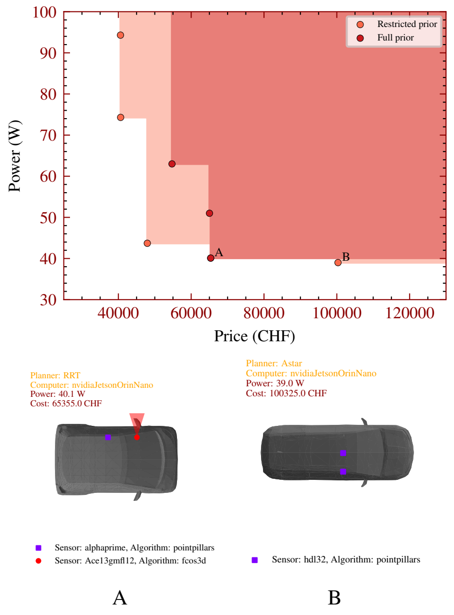


Fig. 7. Pareto front of price and power usage across priors, where priors with more class configurations require more resources. Implementations for points A and B are visualized vertically. A and B indicate the lowest power usage for the least and most restricted prior, respectively.

for higher average speeds, such as $24\,\mathrm{km/h}$ (with nominal speed of $30\,\mathrm{km/h}$). In every solution where minimal power, mass, computation, and cost were evaluated, the RRT* planner, coupled with the sedan vehicle, emerged as the selected choice. This pattern underscores the RRT* planner's superior efficiency within this case study, further highlighted by the sedan vehicle's highest acceleration capabilities and highest price.

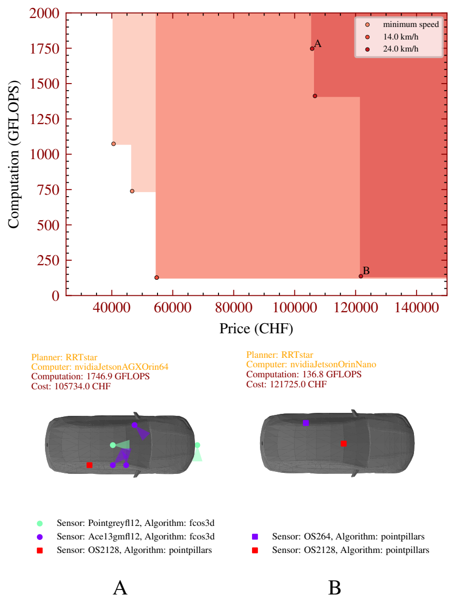


Fig. 8. Pareto front of price and computation across different average speeds, where planners providing higher average speed across all scenarios ($30\,\mathrm{km/h}$ nominal speed) demand more resources. Implementations for points A and B are visualized vertically. A and B indicate the lowest price and lowest computation for the highest average speed, respectively.

VI. Conclusion

This paper introduced a framework for designing mobile robots tailored to specific tasks by selecting hardware and software components. The choice comprises various elements including robot bodies, sensors, perception algorithms, sensor mounting configurations, motion planning algorithms, and computing units. We delved into the decision-making aspect of mobile robots by exploring what information a motion planner requires from the perception system. We introduced occupancy queries for sampling-based motion planners, allowing one to identify the necessary perception requirements based on prior knowledge of object classes, their dynamics, and shapes within the environment. With the obtained perception requirements and the perception performance of a sensor combined with a detection algorithm, abstracted into FNRs and FPRs metrics, we formulated the sensor selection and placement problem and solved it as a weighted set cover problem using an ILP approximation. Our case study on designing an AV for urban driving scenarios revealed that enhanced task complexity, in terms of scenario variety or nominal speeds, necessitates more resources for the robot's design. We demonstrated how restricting prior knowledge of object configurations within scenarios can simplify designs and reduce resource requirements.

In future work, we aim to integrate additional agent architectures and motion planners beyond sampling-based. Additionally, rather than using upper bounds of FNRs and FPRs to determine object detection, we plan to implement filtering and sensor fusion techniques that incorporate considerations of time and uncertainty into the detection and sensor selection process. Moreover, we plan to conduct expanded case studies that include a variety of tasks and robots, not limited to AVs, and utilize state-of-the-art perception and decision-making software.

References

[1] Matthias Althoff, Markus Koschi, and Stefanie Manzinger. Commonroad: Composable benchmarks for motion planning on roads. In *Proc. of the IEEE Intelligent Vehicles Symposium*, 2017. ISBN 9781509048045. doi: 10.1109/ivs.2017.7995802.

[2] Jonathan Backer and David Kirkpatrick. Finding curvature-constrained paths that avoid polygonal obstacles. In *Proceedings of the Twenty-Third Annual Symposium on Computational Geometry*, SCG '07, page 66–73, New York, NY, USA, 2007. Association for Computing Machinery. ISBN 9781595937056. doi: 10.1145/1247069.1247080. URL https://doi.org/10.1145/1247069.1247080.

[3] Basler. Basler cameras, 2024. available online: https://www.baslerweb.com.

[4] Holger Caesar, Varun Bankiti, Alex H. Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[5] Cars. Cars, 2024. available online: https://www.cars.com.

[6] Andrea Censi. Efficient neuromorphic optomotor heading regulation. *Proceedings of the American Control Conference*, 2015-July:3854–3861, 2015. ISSN 07431619. doi: 10.1109/ACC.2015.7171931.

[7] Andrea Censi. A mathematical theory of co-design. *CoRR*, abs/1512.08055, 2015. URL http://arxiv.org/abs/1512.08055.

[8] Andrea Censi, Erich Mueller, Emilio Frazzoli, and Stefano Soatto. A power-performance approach to comparing sensor families, with application to comparing neuromorphic to traditional vision sensors. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3319–3326, 2015. doi: 10.1109/ICRA.2015.7139657.

[9] Andrea Censi, Jonathan Lorand, and Gioele Zardini. *Applied Compositional Thinking for Engineering*. 2024. URL https://bit.ly/3H6pwMo. Work-in-progress book (currently discussing with publishers).

[10] Bernard Chazelle. Approximation and decomposition of shapes. *Algorithmic and Geometric Aspects of Robotics/ed. JT Schwartz, CK Yap*, pages 145–185, 1985.

[11] Laurène Claussmann, Marc Revilloud, Dominique Gruyer, and Sébastien Glaser. A review of motion planning for highway autonomous driving. *IEEE Transactions on Intelligent Transportation Systems*, 21(5):1826–1848, 2020. doi: 10.1109/TITS.2019.2913998.

[12] Anne Collin, Afreen Siddiqi, Yuto Imanishi, Yukti Matta, Taisetsu Tanimichi, and Olivier de Weck. A multiobjective systems architecture model for sensor selection in autonomous vehicle navigation. In Guy André Boy, Alan Guegan, Daniel Krob, and Vincent Vion, editors, *Complex Systems Design & Management*, pages 141–152, Cham, 2020. Springer International Publishing. ISBN 978-3-030-34843-4. doi: 10.1007/978-3-030-34843-4_12.

[13] Anne Collin, Afreen Siddiqi, Yuto Imanishi, Eric Rebentisch, Taisetsu Tanimichi, and Olivier L. de Weck. Autonomous driving systems hardware and software architecture exploration: optimizing latency and cost under safety constraints. *Systems Engineering*, 23 (3):327–337, 2020. doi: https://doi.org/10.1002/sys.21528. URL https://incose.onlinelibrary.wiley.com/doi/abs/10.1002/sys.21528.

[14] MMDetection3D Contributors. MMDetection3D: OpenMMLab next-generation platform for general 3D object detection. https://github.com/open-mmlab/mmdetection3d, 2020.

[15] J. C. Culberson and R. A. Reckhow. Covering polygons is hard. In *Proceedings of the 29th Annual Symposium on Foundations of Computer Science*, SFCS '88, page 601–611, USA, 1988. IEEE Computer Society. ISBN 0818608773. doi: 10.1109/SFCS.1988.21976. URL https://doi.org/10.1109/SFCS.1988.21976.

[16] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, 2 edition, 4 2002. ISBN 9780521784511. doi: 10.1017/CBO9780511809088. URL https://www.cambridge.org/core/product/identifier/9780511809088/type/book.

[17] Van der Coput Johannes. Verteilungsfunktionen i & ii. *Nederl. Akad. Wetensch. Proc.*, 38:1058–1066, 1935. URL https://cir.nii.ac.jp/crid/1571135650584248576.

[18] Joydeep Dey and Sudeep Pasricha. Machine learning based perception architecture design for semi-autonomous vehicles. In Vipin Kumar Kukkala and Sudeep Pasricha, editors, *Machine Learning and Optimization Techniques for Automotive Cyber-Physical Systems*, pages 625–646. Springer International Publishing, Cham, 2023. ISBN 978-3-031-28016-0. doi: 10.1007/978-3-031-28016-0_22. URL https://doi.org/10.1007/978-3-031-28016-0_22.

[19] Joydeep Dey, Wes Taylor, and Sudeep Pasricha. VESPA: A framework for optimizing heterogeneous sensor placement and orientation for autonomous vehicles. *IEEE Consumer Electronics Magazine*, 10(2):16–26, 2021. doi: 10.1109/MCE.2020.3002489.

[20] Michael Erdmann. Understanding action and sensing by designing action-based sensors. *The International Journal of Robotics Research*, 14(5):483–509, 1995. doi: 10.1177/027836499501400506. URL https://doi.org/10.1177/027836499501400506.

[21] P. Falcone, M. Tufo, F. Borrelli, J. Asgari, and H. E. Tseng. A linear time varying model predictive control approach to the integrated vehicle dynamics control problem in autonomous systems. In *2007 46th IEEE Conference on Decision and Control*, pages 2980–2985, 2007. doi: 10.1109/CDC.2007.4434137.

[22] Paolo Falcone, Francesco Borrelli, Jahan Asgari, Hongtei Eric Tseng, and Davor Hrovat. Predictive active steering control for autonomous vehicle systems. *IEEE Transactions on Control Systems Technology*, 15(3):566–580, 2007. doi: 10.1109/TCST.2007.894653.

[23] Flir. Flir cameras, 2024. available online: https://www.flir.com.

[24] Shervin Ghasemlou, Jason M. O'Kane, and Dylan A. Shell. Delineating boundaries of feasibility between robot designs. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages

422–429, 2018. doi: 10.1109/IROS.2018.8593811.

[25] C. Giraud and B. Jouvencel. Sensor selection: a geometrical approach. In *Proceedings 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human Robot Interaction and Cooperative Robots*, volume 2, pages 555–560 vol.2, 1995. doi: 10.1109/IROS.1995.526271.

[26] Sehoon Ha, Stelian Coros, Alexander Alspach, James M. Bern, Joohyung Kim, and Katsu Yamane. Computational design of robotic devices from high-level motion specifications. *IEEE Transactions on Robotics*, 34(5):1240–1251, 2018. doi: 10.1109/TRO.2018.2830419.

[27] John H Halton. On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals. *Numerische Mathematik*, 2:84–90, 1960. doi: https://doi.org/10.1007/BF01386213.

[28] G.E. Hovland and B.J. McCarragher. Dynamic sensor selection for robotic systems. In *Proceedings of International Conference on Robotics and Automation*, volume 1, pages 272–277 vol.1, 1997. doi: 10.1109/ROBOT.1997.620050.

[29] Lucas Janson, Brian Ichter, and Marco Pavone. Deterministic sampling-based motion planning: Optimality, complexity, and performance. *The International Journal of Robotics Research*, 37(1):46–61, 2018. doi: 10.1177/0278364917714338. URL https://doi.org/10.1177/0278364917714338.

[30] Eungcheol Kim, J Kim, and Myoungho Sunwoo. Model predictive control strategy for smooth path tracking of autonomous vehicles with steering actuator dynamics. *International Journal of Automotive Technology*, 15:1155–1164, 2014. doi: https://doi.org/10.1007/s12239-014-0120-9.

[31] Morteza Lahijanian, Maria Svorenova, Akshay A. Morye, Brian Yeomans, Dushyant Rao, Ingmar Posner, Paul Newman, Hadas Kress-Gazit, and Marta Kwiatkowska. Resource-performance tradeoff analysis for mobile robots. *IEEE Robotics and Automation Letters*, 3(3):1840–1847, 2018. doi: 10.1109/LRA.2018.2803814.

[32] Alex H. Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. Pointpillars: Fast encoders for object detection from point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[33] Steven M. LaValle. *Planning Algorithms*. Cambridge university press, 11 2006. URL http://planning.cs.uiuc.edu/.

[34] Steven M. LaValle. Sensing and filtering: A fresh perspective based on preimages and information spaces. *Foundations and Trends® in Robotics*, 1(4):253–372, 2012. ISSN 1935-8253. doi: 10.1561/2300000004. URL http://dx.doi.org/10.1561/2300000004.

[35] Edward A. Lee. Cyber physical systems: Design challenges. In *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*, pages 363–369, 2008. doi: 10.1109/ISORC.2008.25.

[36] Alexander Liniger, Alexander Domahidi, and Manfred Morari. Optimization-based autonomous racing of 1: 43 scale rc cars. *Optimal Control Applications and Methods*, 36(5):628–647, 2015. doi: https://doi.org/10.1002/oca.2123.

[37] Moritz Maierhofer, Moritz Klischat, and Matthias Althoff. Commonroad scenario designer: An open-source toolbox for map conversion and scenario creation for autonomous vehicles. In *Proc. of the IEEE Int. Conf. on Intelligent Transportation Systems*, pages 3176–3182, 2021.

[38] R Timothy Marler and Jasbir S Arora. The weighted sum method for multi-objective optimization: new insights. *Structural and multidisciplinary optimization*, 41:853–862, 2010. doi: https://doi.org/10.1007/s00158-009-0460-7.

[39] Ankur M. Mehta, Joseph DelPreto, Kai Weng Wong, Scott Hamill, Hadas Kress-Gazit, and Daniela Rus. *Robot Creation from Functional Specifications*, pages 631–648. Springer International Publishing, Cham, 2018. ISBN 978-3-319-60916-4. doi: 10.1007/978-3-319-60916-4_36. URL https://doi.org/10.1007/978-3-319-60916-4_36.

[40] Dejan Milojevic, Gioele Zardini, Miriam Elser, Andrea Censi, and Emilio Frazzoli. Resource-efficient task-driven co-design of perception and decision making in autonomous robots. *Submitted to IEEE Transactions on Robotics*, 2024. ISSN 1552-3098. doi: 10.3929/ethz-b-000672201.

[41] Luigi Nardi, David Koeplinger, and Kunle Olukotun. Practical design space exploration. In *2019 IEEE 27th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 347–358, 2019. doi: 10.1109/MASCOTS.2019.00045.

[42] Alexandra Q. Nilles, Dylan A. Shell, and Jason M. O'Kane. Robot design: Formalisms, representations, and the role of the designer. *CoRR*, abs/1806.05157, 2018. URL http://arxiv.org/abs/1806.05157.

[43] NVIDIA. Nvidia products, 2024. URL https://www.nvidia.com. available online: https://www.nvidia.com.

[44] Jason M. O'Kane and Steven M. LaValle. Comparing the power of robots. *The International Journal of Robotics Research*, 27(1):5–23, 2008. doi: 10.1177/0278364907082096. URL https://doi.org/10.1177/0278364907082096.

[45] Ouster. Ouster lidars, 2024. available online: https://ouster.com.

[46] Art B. Owen. A randomized halton algorithm in R, 2017.

[47] Brian Paden, Michal Čáp, Sze Zheng Yong, Dmitry Yershov, and Emilio Frazzoli. A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on Intelligent Vehicles*, 1(1):33–55, 2016. doi: 10.1109/TIV.2016.2578706.

[48] Guilherme V. Raffo, Guilherme K. Gomes, Julio E. Normey-Rico, Christian R. Kelber, and Leandro B. Becker. A predictive controller for autonomous vehicle path tracking. *IEEE Transactions on Intelligent Transportation Systems*, 10(1):92–102, 2009. doi: 10.1109/TITS.2008.2011697.

[49] Fatemeh Zahra Saberifar, Shervin Ghasemlou, Dylan A Shell, and Jason M O'Kane. Toward a language-theoretic foundation for planning and filtering. *The International*

Journal of Robotics Research, 38(2-3):236–259, 2019. doi: 10.1177/0278364918801503. URL https://doi.org/10.1177/0278364918801503.

[50] Fatemeh Zahra Saberifar, Dylan A. Shell, and Jason M. O'Kane. Charting the trade-off between design complexity and plan execution under probabilistic actions. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 135–141, 2022. doi: 10.1109/ICRA46639.2022.9811751.

[51] Sangok Seok, Albert Wang, Meng Yee Chuah, Dong Jin Hyun, Jongwoo Lee, David M. Otten, Jeffrey H. Lang, and Sangbae Kim. Design principles for energy-efficient legged locomotion and implementation on the MIT Cheetah robot. *IEEE/ASME Transactions on Mechatronics*, 20(3):1117–1129, 2015. doi: 10.1109/TMECH.2014.2339013.

[52] Sanjit A. Seshia, Shiyan Hu, Wenchao Li, and Qi Zhu. Design automation of cyber-physical systems: Challenges, advances, and opportunities. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 36(9):1421–1434, 2017. doi: 10.1109/TCAD.2016.2633961.

[53] Dylan A. Shell, Jason M. O'Kane, and Fatemeh Zahra Saberifar. On the design of minimal robots that can solve planning problems. *IEEE Transactions on Automation Science and Engineering*, 18(3):876–887, 2021. doi: 10.1109/TASE.2021.3050033.

[54] Ivan P Stanimirovic, Milan Lj Zlatanovic, and Marko D Petkovic. On the linear weighted sum method for multi-objective optimization. *Facta Acta Univ*, 26(4):49–63, 2011.

[55] Ioan A. Şucan, Mark Moll, and Lydia E. Kavraki. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, December 2012. doi: 10.1109/MRA.2012.2205651. https://ompl.kavrakilab.org.

[56] O. Takahashi and R.J. Schilling. Motion planning in a plane using generalized voronoi diagrams. *IEEE Transactions on Robotics and Automation*, 5(2):143–150, 1989. doi: 10.1109/70.88035.

[57] TurboSquid. Turbosquid 3d models, 2024. available online: https://www.turbosquid.com/3d-models/3d-40-cars-1703688.

[58] Vasileios Tzoumas, Luca Carlone, George J. Pappas, and Ali Jadbabaie. LQG control and sensing co-design. *IEEE Transactions on Automatic Control*, 66(4):1468–1483, 2021. doi: 10.1109/TAC.2020.2997661.

[59] Vijay V Vazirani. *Approximation algorithms*, volume 1. Springer Berlin, Heidelberg, 1 edition, 2001. ISBN 978-3-662-04565-7. doi: https://doi.org/10.1007/978-3-662-04565-7.

[60] Velodyne. Velodyne lidars, 2024. available online: https://velodynelidar.com.

[61] Tai Wang, Xinge Zhu, Jiangmiao Pang, and Dahua Lin. FCOS3D: Fully Convolutional One-Stage Monocular 3D Object Detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) Workshops*, pages 913–922, October 2021.

[62] Yongsoon Yoon, Jongho Shin, H. Jin Kim, Yongwoon Park, and Shankar Sastry. Model-predictive active steering and obstacle avoidance for autonomous ground vehicles. *Control Engineering Practice*, 17(7):741–750, 2009. ISSN 0967-0661. doi: https://doi.org/10.1016/j.conengprac.2008.12.001. URL https://www.sciencedirect.com/science/article/pii/S0967066108002025.

[63] Gioele Zardini. *Co-design of complex systems: From autonomy to future mobility systems*. PhD thesis, ETH Zurich, 2023.

[64] Gioele Zardini, Andrea Censi, and Emilio Frazzoli. Co-Design of Autonomous Systems: From Hardware Selection to Control Synthesis. *2021 European Control Conference, ECC 2021*, pages 682–689, 2021. doi: 10.23919/ECC54610.2021.9654960.

[65] Gioele Zardini, Dejan Milojevic, Andrea Censi, and Emilio Frazzoli. Co-design of embodied intelligence: A structured approach. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7536–7543, 2021. doi: 10.1109/IROS51168.2021.9636513.

[66] Gioele Zardini, Zelio Suter, Andrea Censi, and Emilio Frazzoli. Task-driven modular co-design of vehicle control systems. In *2022 IEEE 61st Conference on Decision and Control (CDC)*, pages 2196–2203, 2022. doi: 10.1109/CDC51059.2022.9993107.

[67] Gioele Zardini, Nicolas Lanzetti, Andrea Censi, Emilio Frazzoli, and Marco Pavone. Co-design to enable user-friendly tools to assess the impact of future mobility solutions. *IEEE Transactions on Network Science and Engineering*, 10(2):827–844, 2023. doi: 10.1109/TNSE.2022.3223912.

[68] Yulin Zhang and Dylan A. Shell. Abstractions for computing all robotic sensors that suffice to solve a planning problem. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 8469–8475, 2020. doi: 10.1109/ICRA40945.2020.9196812.

[69] Qi Zhu and Alberto Sangiovanni-Vincentelli. Codesign methodologies and tools for cyber–physical systems. *Proceedings of the IEEE*, 106(9):1484–1500, 2018. doi: 10.1109/JPROC.2018.2864271.

Appendix

A. Appendix: Modeling a task

Consider a robot $\mathcal{R}$, operating within the workspace $\mathcal{W} \subset \mathbb{R}^3$. The robot starts its mission from an initial configuration denoted by $\gamma_{\text{start}} \in \Gamma$ and seeks to reach a goal area denoted as $\mathcal{G}$.[2]

Definition 15 (Object Class Distribution)**.** An *object class distribution* $\mathcal{O}$ is defined as a tuple:

$$\mathcal{O} := \langle \mathcal{C}, \mathcal{P}, \lambda \rangle,$$

where the class of the object is denoted by $\mathcal{C}$, and $\mathcal{P}$ represents the prior configurations, such that $\mathcal{P} \subseteq \pi_3(\mathcal{C}) = \mathcal{Q}$ for a particular class. This prior essentially outlines the allowed configurations for objects of the specific class. The distribution

[2]We consider $\mathcal{G} \subset \mathbb{R}^2$, but in general the goal $\mathcal{G}$ can manifest in various forms, including a terminal configuration q_{end}, a volume in $\mathbb{R}^3$ to be reached, following another object, or the ability to move for a specified duration.

TABLE I
TABLE OF SYMBOLS

Symbol	Meaning
$\mathcal{A}$	decision-making agent
appear $\in$ AP	class appearances
$\mathcal{B}$	robot's body
$\mathcal{C}$, $C \in \mathbb{C}$	object class, instance of an object class
$q \in \mathcal{Q}$	configuration space
$\gamma \in \Gamma$	configuration space of the robot
c	cost function
env $\in \mathbb{E}$	environment
mo $\in$ MO	mounting orientations (yaw and pitch) in $\mathbb{R}^2$
mp $\in$ MP	mounting position in $\mathbb{R}^3$
mpp $\in \mathbb{MPP}$	mounted perception pipeline
mppcc	mounted perception pipeline class coverage
MPPC	set of mounted perception pipeline class coverage
$\mathcal{O}$	object class distribution
pcp	perceptual collision prediction map
pp $\in$ PP	perception pipeline
PR	task perception requirements
$\mathcal{P}$	the object class configurations prior
$\psi \in \Psi$	occupancy query space
$\mathcal{R}$	robot
$\mathcal{S}$, S	scenario, instance of a scenario
SH	robot's 3D shape
sh	map which returns the footprint from a configuration
$\mathcal{T}$	robot's task
tq	map that generates task queries of an agent
$\bar{q} \in \bar{\mathcal{Q}}$	object class trajectories

of objects follows a Poisson distribution, where λ_i represents the expected number of objects for a given class.

Definition 16 (Scenario)**.** A scenario is given by

$$\mathcal{S} := \langle \mathcal{W}, f_\Gamma, f_{\mathbb{R}^2}, f_\mathbb{E}, \{\mathcal{O}_i\}_{i\in\{1,\dots,N\}} \rangle,$$

where $\Gamma \sim f_\Gamma(\gamma_{\text{start}})$, $\mathbb{R}^2 \sim f_{\mathbb{R}^2}(\mathcal{G})$ and $\mathbb{E} \sim f_\mathbb{E}(\text{env})$ are the distributions governing the initial configuration of the robot, the goal area of the robot, and the environmental conditions, respectively. The scenario includes N object classes with a corresponding object class distribution $\mathcal{O}_i$.

A scenario instance

$$S = \langle \mathcal{W}, \gamma_{\text{start}}, \mathcal{G}, \text{env}, \{C_i\}_{i\in\{1,\dots,M\}} \rangle$$

represents a concrete realization of a scenario $\mathcal{S}$, where the initial configuration, goal, and environment are drawn from their respective distributions f_Γ, $f_{\mathbb{R}^2}$, and $f_\mathbb{E}$. Moreover, M number of object class instances are drawn from the corresponding Poisson distributions.

B. Appendix: Background on a monotone theory of co-design

The reader is assumed to be familiar with posets and basic concepts of order theory (a good source is [16]).

a) Formulating co-design problems: The atom of the theory is the notion of a MDPI, through which we will model different components of the autonomy stack.

Definition 17. Given posets $\mathcal{F}, \mathcal{R}$, (mnemonics for functionalities and resources), we define a *MDPI* as a tuple $\langle \mathcal{I}_d, \mathsf{prov}, \mathsf{req} \rangle$, where $\mathcal{I}_d$ is the set of implementations, and prov, req are maps from $\mathcal{I}_d$ to $\mathcal{F}$ and $\mathcal{R}$, respectively:

$$\mathcal{F} \xleftarrow{\mathsf{prov}} \mathcal{I}_d \xrightarrow{\mathsf{req}} \mathcal{R}.$$

We compactly denote the MDPI as $d \colon \mathcal{F} \nrightarrow \mathcal{R}$. Furthermore, to each MDPI we associate a monotone map $\bar{d}$, given by:

$$\begin{aligned} \bar{d} \colon \mathcal{F}^{\text{op}} \times \mathcal{R} &\to \langle \mathcal{P}(\mathcal{I}_d), \subseteq \rangle \\ \langle f^*, r \rangle &\mapsto \{ i \in \mathcal{I}_d \colon (\mathsf{prov}(i) \succeq_\mathcal{F} f) \wedge (\mathsf{req}(i) \preceq_\mathcal{R} r) \}, \end{aligned}$$

where $(\cdot)^{\text{op}}$ reverses the order of a poset. The expression $\bar{d}(f^*, r)$ returns the set of implementations (design choices) $S \subseteq \mathcal{I}_d$ for which functionalities f are feasible with resources r. A MDPI is represented in diagrammatic form as a block with green wires on the left for functionalities, and dashed red ones on the right for resources, as visualized in Fig. 9.

Remark 18 (Monotonicity)**.** What does monotonicity mean in this context? Consider a MDPI for which $\bar{d}(f^*, r) = S$:

- One has: $f' \preceq_\mathcal{F} f \Rightarrow \bar{d}(f'^*, r) = S' \supseteq S$. Intuitively, decreasing the provided functionalities will not increase the required resources;
- One has: $r' \succeq_\mathcal{R} r \Rightarrow \bar{d}(f^*, r') = S'' \supseteq S$. Intuitively, increasing the available resources cannot decrease the provided functionalities.

Remark 19 (Populating the models)**.** The presented framework is very flexible. In practice, one populates the MDPIs via analytic relations (e.g., cost functions), numerical analysis of closed-form relations (e.g., solving optimal control problems), and in a data-driven, on-demand fashion (e.g., via POMDPs, simulations, or by solving instances of optimization problems). For detailed examples related to mobility and autonomy, please refer to [64, 65, 67, 63, 66, 9].

One can compose individual MDPIs in several ways to form a co-design problem (i.e., a multigraph of MDPIs, where nodes are MDPIs, and edges their interconnections), which is again a MDPI (i.e., closure). This makes the presented framework practical to decompose a large problem into smaller ones, and to interconnect them[3] Series composition happens when the functionality of a MDPI is required by another MDPI (e.g., information acquired by a sensor is processed by an estimator). The symbol $\preceq$ is the posetal relation, representing a co-design constraint: the resource a problem requires cannot exceed the functionality another problem provides. Parallel composition, instead, formalizes decoupled processes happening together. Finally, loop composition describes feedback.

b) Solving co-design problems: Given a MDPI, we essentially have two queries. First, given some desired functionalities, find the optimal design solutions which minimize resources (FixFunMinRes). Alternatively, given some available resources, find the optimal design choices which maximize functionalities (FixResMaxFun).

Definition 20. Given a MDPI d, one defines monotone maps

- $h_d \colon \mathcal{F} \to \mathsf{A}\mathcal{R}$, mapping a functionality to the *minimum* antichain of resources providing it;
- $h'_d \colon \mathcal{R} \to \mathsf{A}\mathcal{F}$, mapping a resource to the *maximum* antichain of functionalities provided by it.

Solving MDPIs requires finding such maps. If such maps are Scott continuous, and posets are complete, one can rely on

[3]A detailed list of compositions is provided in [9, 63]. Formally, their specification makes the category of design problems a traced monoidal category, with locally posetal structure.

Kleene's fixed point theorem to design an algorithm solving both queries (and returning the related optimal design choices).

Interestingly, the resulting algorithm is guaranteed to converge to the set of optimal solutions, or to provide a certificate of infeasibility. Furthermore, the complexity of solving such problems is only linear in the number of options available for each component (as opposed to combinatorial). For more details, refer to [9, 63].

C. Appendix: Modeling from task to perception requirements

Planner MDPI: The more scenario instances are required, the more queries are needed by the planner, as detailed in Lemma 21. Higher acceleration and deceleration expand the range of possible queries, enabling faster achievement of goals in scenario instances. A smaller minimum turning radius increases the diversity of occupancy queries and the robot's capability to navigate through complex scenarios, such as tight passages that a large turning radius would not permit. Consequently, we utilize the opposite of a poset for minimum turning radius. Additionally, greater acceleration necessitate more computational resources to quickly process planning strategies. Extending the average speed requires improved dynamics with quicker acceleration, or a more efficient planner, which increases the need for compute resources and occupancy queries.

Lemma 21. The task occupancy queries tq is monotone in the task, as shown in Fig. 10.

Proof: Consider two tasks $\mathcal{T}_1 \subseteq \mathcal{T}_2$. We have

$$\begin{aligned}\mathrm{tq}(\mathcal{A}, \mathcal{T}_1) &\subseteq \big(\mathrm{tq}(\mathcal{A}, \mathcal{T}_1) \cup \mathrm{tq}(\mathcal{A}, \mathcal{T}_2 \setminus \mathcal{T}_1)\big) \\ &= \mathrm{tq}(\mathcal{A}, \mathcal{T}_2).\end{aligned}$$

■

Perceptual Collision Prediction MDPI: The class footprint$^{\mathrm{op}}$ is the planar shape of the class in 2D, generated by the map sh. The resources include class trajectories $\bar{\mathcal{Q}}$ and the robot's footprint from $\mathrm{sh}_{\mathcal{R}}$. Lemma 22 illustrates the monotonic relationship between class trajectories and occupancy queries, indicating that an increase in occupancy queries leads to an equal or greater number of class trajectories. This relationship also applies to class dynamics, altering class dynamics results in new class trajectories. Specifically, higher acceleration and deceleration and a smaller minimum turning radius produce a broader range of class trajectories. In Lemma 23, the monotonicity of the robot's footprint with occupancy queries is shown, implying a larger robot's footprint is required as queries increase, assuming a fixed set of class trajectories. For example, if a robot's footprint encompasses $\mathbb{R}^2$, no class trajectory can collide with it, as the robot already occupies all available space. Similarly, a larger class footprint$^{\mathrm{op}}$ indicates earlier collisions with the robot, thus generating fewer class trajectories. This inverse relationship uses the opposite of a poset for the class footprint$^{\mathrm{op}}$.

Lemma 22. The class trajectories from pcp are monotone with respect to the occupancy queries as shown in Fig. 11.

Proof: Consider two query sets $\Psi_1 \subseteq \Psi_2$. We have

$$\begin{aligned}\mathrm{pcp}(\Psi_1, C_i) &\subseteq \big(\mathrm{pcp}(\Psi_1, C_i) \cup \mathrm{pcp}(\Psi_2 \setminus \Psi_1, C_i)\big) \\ &= \mathrm{pcp}(\Psi_2, C_i).\end{aligned}$$

■

Lemma 23. The robot's footprint is monotone with respect to the occupancy queries as shown in Fig. 11.

Proof: A larger robot's footprint can exclude more class trajectories, according to Def. 7 and Def. 9. ■

Prior Check MDPI: According to Lemma 24, priors that encompass more class configurations tend to filter out fewer configurations during $\mathrm{priorcheck}$, resulting in more perception requirements. Given the relations established in Lemma 21 and Lemma 22, where more complex tasks generate more class trajectories, it follows, as demonstrated in Lemma 25, that increased task complexity (more class trajectories) also amplifies the perception requirements.

Lemma 24. The class configurations in the class trajectories are monotone with respect to the class configurations prior as shown in Fig. 12.

Proof: Consider two priors $\mathcal{P}_{i,1} \subseteq \mathcal{P}_{i,2}$ and a class configuration set Θ_i. If $\Theta_i \subseteq \mathcal{P}_{i,1}$ then it holds also $\Theta_i \subseteq \mathcal{P}_{i,2}$. If $\Theta_i \subseteq \mathcal{P}_{i,2} \setminus \mathcal{P}_{i,1}$, then $\Theta_i \subseteq \mathcal{P}_{i,2}$ but $\Theta_i \cap \mathcal{P}_{i,1} = \emptyset$. ■

Lemma 25. The perception requirements PR are monotone in the task, respectively in the class trajectories (Fig. 12).

Proof: Consider two tasks $\mathcal{T}_1 \subseteq \mathcal{T}_2$. From Lemma 21 we know that occupancy queries are monotone in the task and from Lemma 22 we know that class trajectories are monotone with the queries. We have

$$\begin{aligned}\mathrm{PR}(\mathcal{A}, \mathcal{T}_1) &\subseteq \big(\mathrm{PR}(\mathcal{A}, \mathcal{T}_1) \cup \mathrm{PR}(\mathcal{A}, \mathcal{T}_2 \setminus \mathcal{T}_1)\big) \\ &= \mathrm{PR}(\mathcal{A}, \mathcal{T}_2).\end{aligned}$$

■

Robot body MDPI: As visualized in Fig. 13, this MDPI provides the robot's dynamics functionality, parameterized as minimum turning radius (considered opposite of a poset), maximum acceleration, and deceleration. Additionally, it outlines mounting configurations for sensors within $\mathrm{SE}(3)$, the robot's footprint $\mathrm{sh}_{\mathcal{R}}$ in $\mathbb{R}^2$, the maximum payload mass in kg the robot can carry, its auxiliary power capacity in W for powering hardware such as sensors and computers, and the driving range in m representing the robot's driving range without recharge. Enhanced robot's dynamics, such as greater acceleration/deceleration and a reduced turning radius, typically necessitate higher fixed costs and operational costs. Similarly, increasing the payload mass and auxiliary power capacity implies a need for a more costly or larger robot's shape. Boosting the driving range involves augmenting the battery size, impacting both fixed and operational costs. Additional sensor mounting configurations may necessitate a larger robot's shape to accommodate the setup. As aforementioned, a larger robot's footprint can potentially reduce perception requirements by obstructing more class trajectories. Achieving a larger robot's footprint requires a correspondingly larger robot's shape.

Computing MDPI: As the demand for compute increases to accommodate more sophisticated software algorithms or larger data volumes, the specifications of the computing units must be scaled up accordingly. This, in turn, impacts the overall cost of the computing hardware, its mass, and its power consumption. The Computing MDPI is illustrated in Fig. 14.

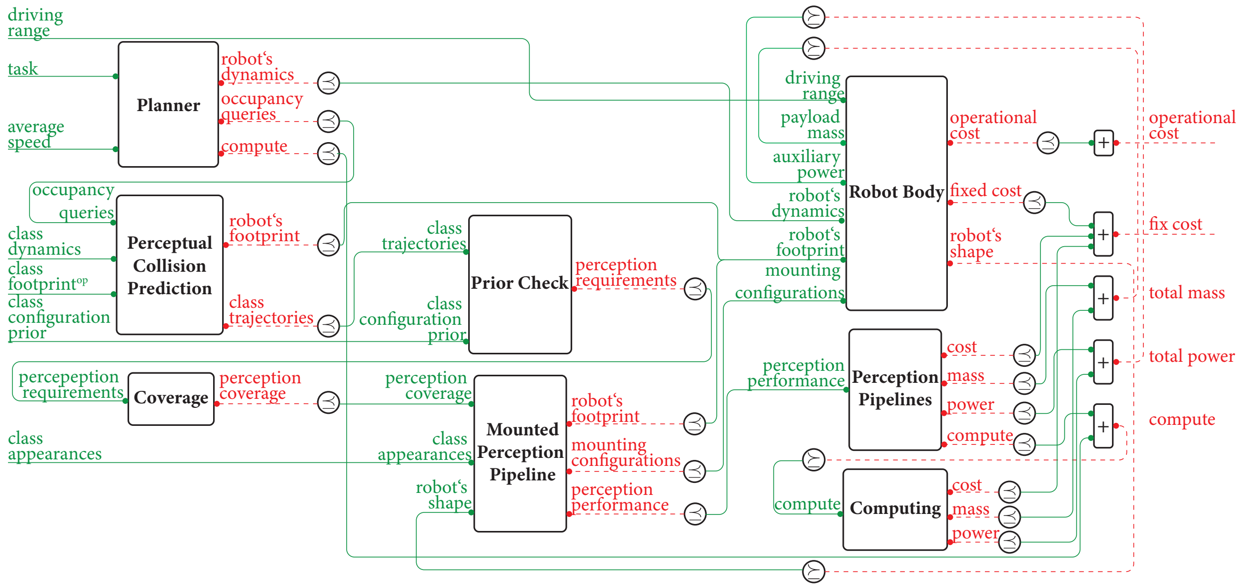


Fig. 9. The co-design diagram for the design of a mobile robot tailored to accomplish a task, including a collection of scenario instances and class instances, aiming to achieve specified average speed and driving range. The class instances include class dynamics, class footprint$^{\mathrm{op}}$, class appearances, and class configurations prior. The objective is to minimize the fix cost and operational cost.

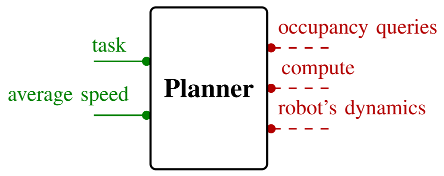


Fig. 10. The planner MDPI which implements a motion planner for the robot to accomplish scenario instances of a task and thereby providing average speed, while requiring occupancy queries Ψ, compute and robot's dynamics

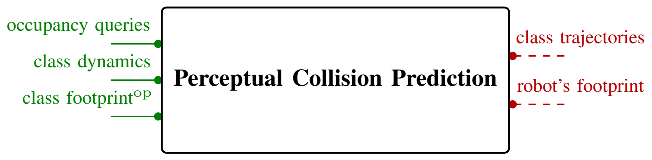


Fig. 11. The Perceptual Collision Prediction MDPI which implements the function pcp. The functionalities are the occupancy queries for the planner, the class dynamics and the class footprint$^{\mathrm{OP}}$. The required resources are the class trajectories and robot's footprint.

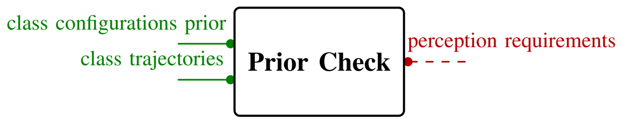


Fig. 12. The Prior Check MDPI, which implements priorcheck, provides class configurations prior and class trajectories functionalities and requires perception requirements.

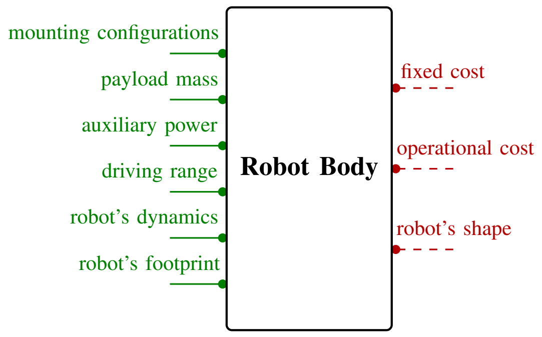


Fig. 13. The Robot body MDPI which provides the dynamics dyn, the mounting configurations for sensors each in SE(3), the body footprint $\mathrm{sh}_{\mathcal{R}}$, the payload mass in kg, the auxilary power in W and the driving range in m, while requiring robot's shape SH, fixed cost in CHF and operational costs in CHF/m.

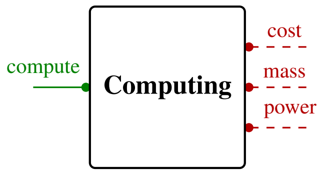


Fig. 14. The Computing MDPI which implements the computing units. It provides compute and requires cost in CHF, the mass in kg and power consumption in W.

D. Appendix: Sensor selection and placement problem

***Coverage** MDPI:* An enhancement in perception requirements necessitates an increase in perception coverage, which is demonstrated in Lemma 26.

Lemma 26. The perception requirements PR are monotone with perception coverage, as shown in Fig. 15.

Proof: Consider the first constraint in Eq. (1), representing the sensor selection and placement optimization problem. Clearly, if one increases the PR set, one needs to increase the union of selected perception pipeline class coverage mppcc, representing the perception coverage. ■

***Mounted Perception Pipelines** MDPI:* In the MDPI visualized in Fig. 16, the perception performance considers the opposite order of fnr and fpr upper limits, where a pipeline

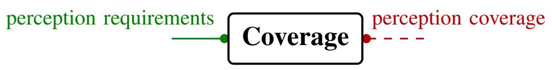


Fig. 15. The Coverage MDPI which provides perception requirements PR and requires perception coverage as a set of mppcc.

pp_a dominates pp_b if it has lower upper bounds for fnr and fpr across all class configurations q_i, class appearances appear_i and environments env.

Adding a class configuration to the perception coverage or new class appearances may necessitate a change to a more capable perception pipeline with improved perception performance to ensure coverage under the defined threshold ϵ. Similarly, enhancing perception coverage with new class configurations or class appearances might necessitate additional mounting configurations.

A larger robot's shape may introduce self-occlusion, impacting the FoV and necessitating additional sensor placements for coverage. While a larger robot's footprint can reduce perception requirements by obstructing potential class trajectories as shown in Fig. 11, balancing between robot's footprint and robot's shape becomes crucial. A theoretically ideal robot's shape would generate a vast robot's footprint but have no elevation, thereby minimizing self-occlusion and perception requirements. Although this poses practical challenges in dynamics and scenario feasibility, where a larger robot's shape usually leads to a larger turning radius.

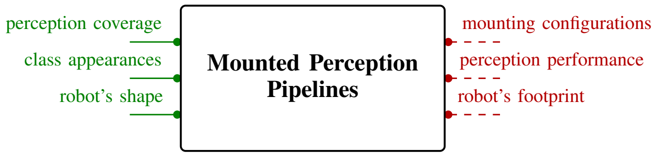


Fig. 16. The Mounted Perception Pipelines MDPI which provides the perception coverage, the set of all class appearances appear in the task and the robot's shape SH as functionalities. The required resources are the set of mounting configurations in SE(3), the perception performance and the robot's footprint $\text{sh}_{\mathcal{R}}$.

***Perception Pipelines** MDPI:* The Perception Pipelines MDPI is illustrated in Fig. 17.

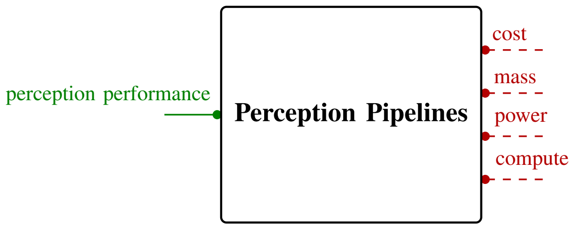


Fig. 17. The Perception Pipelines MDPI which provides the perception performance and requires cost in CHF, mass in kg, power in W and compute.

Finally, the Sensor Selection and Placement MDPI is shown in Fig. 18, which is the composition of the Coverage, Mounted Perception Pipelines and Perception Pipelines MDPIs.

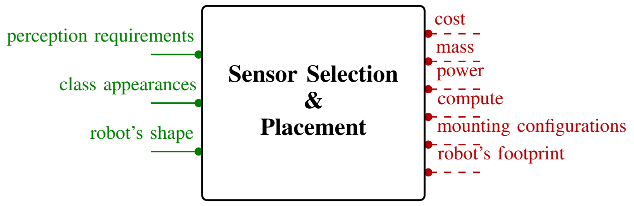


Fig. 18. The Sensor Selection and Placement MDPI which is the composition of the Coverage, Mounted Perception Pipelines and Perception Pipelines MDPIs.

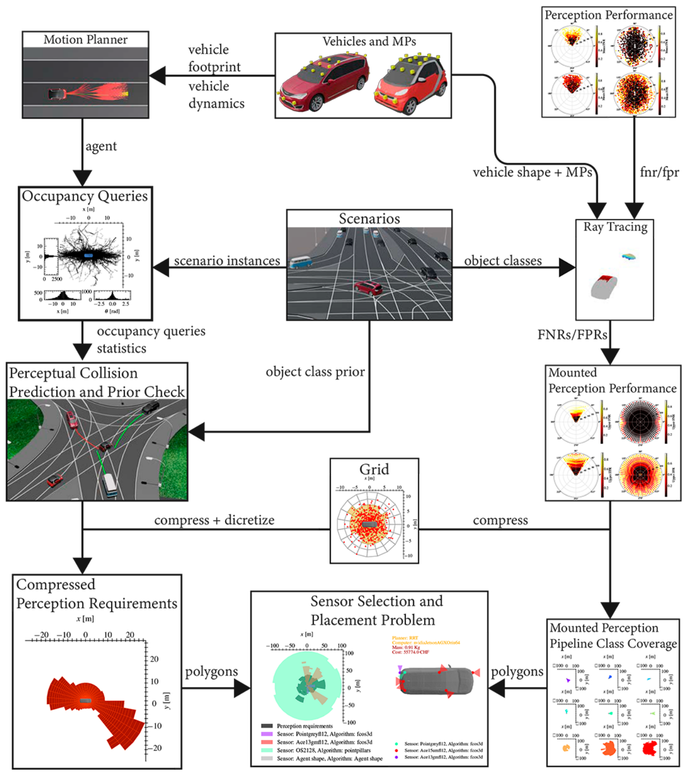


Fig. 19. Overview of the sensor selection and placement process: starting with a catalog of robot bodies, sensor positions, orientations, perception pipelines, and motion planners, alongside with scenarios. The workflow splits into agent activities (left) that transform task queries into perception requirements, and perception activities (right) that determine class configurations detectable by mounted perception pipelines. The process concludes with the selection of optimal pipelines to minimize costs while satisfying perception requirements.

E. Appendix: Solving the Sensor Selection and Placement Set Cover Problem

The nature of Def. 14 closely resembles the weighted set cover problem [59], since it also tries to cover a given set by a collection of subsets while minimizing a cost function.

Definition 27 (Weighted set cover problem)**.** Given a set U of N elements (called *universe*), a collection of subsets of U, $S = \{S_1, \ldots, S_K\}$, and a cost function $c : S_i \to \mathbb{R}_{>0}$, find a minimum cost sub-collection of S that covers all elements of U.

The weighted set cover problem is NP-complete. There exist approximations, such as greedy algorithms or ILP. In addressing Def. 14, we choose the ILP relaxation of the set cover problem, as outlined in Eq. (2). In this ILP, each set S_i is associated with a variable $x_i \in \{0,1\}$, where $x_i = 1$ if and only if set S_i is selected. The constraint mandates that for each element $e \in U$, at least one of the sets containing it is chosen [59].

$$\begin{aligned} \min \quad & \sum_{i=1}^{K} c(S_i) \cdot x_i \\ \text{s.t.} \quad & \sum_{i: e \in S_i} x_i \geq 1 \quad \forall e \in U, \\ & x_i \leq 1 \quad \forall i \in \{1, \dots, K\}, \\ & x_i \in \mathbb{N}_0 \quad \forall i \in \{1, \dots, K\}. \end{aligned} \tag{2}$$

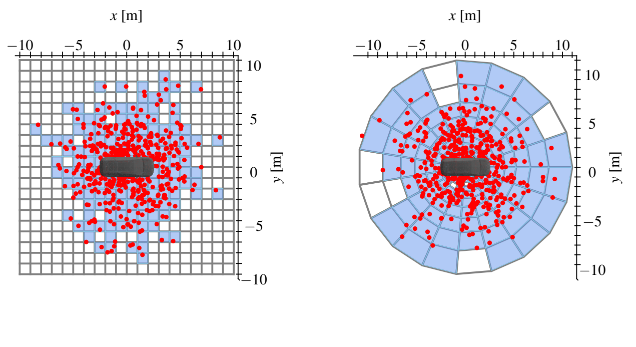
To formulate the Def. 14 as a weighted set cover problem, we need to make certain approximations. This is necessary because both the task perception requirements, denoted as $\mathrm{PR}(\mathcal{A}, \mathcal{T})$, and the coverage of mounted perception pipelines for different classes, denoted as MPPC, are infinite sets. In the next paragraphs we show how we formulate the sensor selection and placement problem as a weighted set cover problem.

Class configurations in $\mathrm{SE}(2)$*:* The first approximation involves constraining all class configurations in both $\mathrm{PR}(\mathcal{A}, \mathcal{T})$ and MPPC to exist within $\mathrm{SE}(2)$. Specifically, each class configuration is now defined as a tuple consisting of position in Cartesian coordinates and the relative orientation θ with respect to the robot frame, denoted as $q_i = \langle x, y, \theta \rangle$. As these class configurations are now geometric in nature and reside in $\mathrm{SE}(2)$, the problem closely resembles the *polygon covering* problem [15], which is a specific case of the set cover problem. In the weighted polygon covering problem, the objective is to cover a target polygon using a set of provided polygons, each associated with a specific cost. This problem permits overlapping among the polygons. However, the class configurations are represented in three-dimensional space ($\mathrm{SE}(2)$) and are essentially volumes rather than polygons. Therefore, we need a method to reduce the dimensionality of these configurations.

From class configurations to polygons: Given the orientation constraint $-\pi \leq \theta \leq \pi$, the class configurations are sorted into θ-intervals, such as $\{[-\pi, -\pi + \Delta\theta), [-\pi + \Delta\theta, -\pi + 2 \cdot \Delta\theta) \dots [\pi - \Delta\theta, \pi)\}$. The subsequent step involves transforming the position coordinates of the class configuration within each θ-interval into a set of polygons. Here, polygons represent surfaces in $\mathbb{R}^2$ with location considerations. This set of polygons is termed a *multi-polygon*, where the polygons in the set are not necessarily contiguous. As a result, a set of multi-polygons is generated, with each element corresponding to a distinct θ-interval. Although various methods can be devised for this transformation, we stick to a worst-case analysis approach for consistency. The detailed description of this process is beyond the scope of this paper. The resulting set of multi-polygons is denoted as *compressed class configurations*.

Definition 28 (Compress)**.** compress is a mapping that generates a set of multi-polygons μ from a set of class configurations $\mathcal{Q}_i$ and T number of class configurations θ-intervals.

$$\text{compress} \colon \mathtt{POW}(\mathcal{Q}_i) \to \prod_{j \in \{1, \dots, T\}} \mathtt{POW}(\mathbb{R}^2),$$



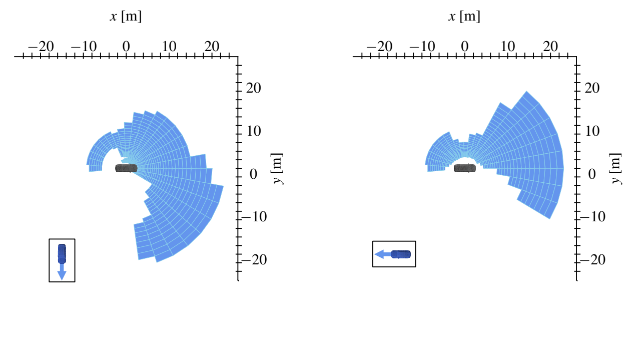
(a) Unit grid cells. (b) Polar grid with logarithmic scale.

Fig. 20. The left image shows a uniform grid, while the right reports a polar grid with logarithmically scaled radial distances. Red dots, representing Gaussian synthetic class configurations, intersect with blue shaded cells.

where $T \in \mathbb{N}^+$.

Applying compress to $\mathrm{PR}(\mathcal{A}, \mathcal{T})$ and MPPC results in $\overline{\mathrm{PR}}(\mathcal{A}, \mathcal{T})$ and $\overline{\mathrm{MPPC}}$, where all sets of class configurations are now expressed as compressed class configurations. Specifically, when compress is applied for each environment in PR, nested sets are obtained for each environment, object class, and θ interval.

Discretization: To formulate the weighted set cover problem with the obtained polygons, we need to discretize $\overline{\mathrm{PR}}(\mathcal{A}, \mathcal{T})$. A straightforward approach is to create a grid with cells, which can be made uniform as shown in Fig. 20a, e.g., 1 by 1 meters in size. We use a polar grid with logarithmic scaling for radial distance as illustrated in Fig. 20b, providing higher granularity for smaller distances and aligning more with sensor perception dynamics which scan the environment radially. This means for each multi-polygon in $\overline{\mathrm{PR}}(\mathcal{A}, \mathcal{T})$, which corresponds to a certain environment, a certain class and a certain θ-interval, we obtain a discretized multi-polygon which is again a multi-polygon. These discretized perception requirements are represented as $\widehat{\mathrm{PR}}(\mathcal{A}, \mathcal{T})$. An example of discretized perception requirements of an AV driving in an urban environment, for a car class object for two different orientations is shown in Fig. 21.



(a) Car class for θ-interval $[-100°, -90°)$. (b) Car class for θ-interval $[-180°, -170°)$.

Fig. 21. Example of discretized and compressed perception requirements of a car class (blue) for different orientations relative to the ego vehicle (grey car).

In Fig. 22, examples of compressed mppcc are depicted

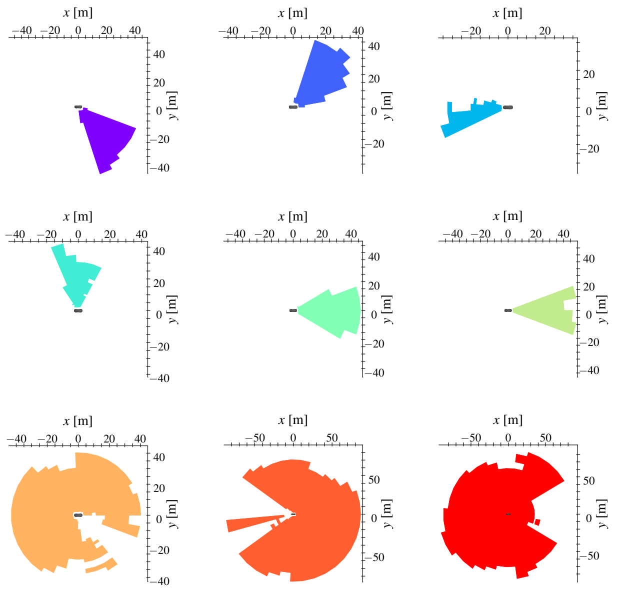


Fig. 22. Examples of compressed mounted perception pipeline class coverage mppcc corresponding to the setting in Fig. 21 with θ-interval $[-100°, -90°)$. Each plot corresponds to a unique mounted perception pipeline.

for the class and robot specified in Fig. 21 with θ-interval $[-100°, -90°)$. These polygons aim to cover the upper polygon shown in Fig. 21. Each polygon is associated with certain costs, and the objective is to minimize the total cost.

With all the components in place, we can formulate the problem in Def. 14 as an ILP using Eq. (2). Once again, we use the binary vector x, where each element $x_i \in \{0, 1\}$ and represents a decision variable. The variable $x_i = 1$ if and only if the mounted perception pipeline mpp_i is chosen.

Cost Functions: We extend the ILP from Eq. (2) to a multi-weighted problem formulation by incorporating W cost functions denoted as c. Each cost function c_j associates a mounted perception pipeline mpp with normalized costs, where $0 \leq c_j(\mathrm{mpp}) \leq 1$. These costs may represent various factors such as the price, mass, or power consumption of the sensor. Additionally, each cost c_j is scaled by a cost weight w_j, ensuring that the sum of all weights equals one, i.e., $\sum_{j=1}^{W} w_j = 1$. The cost function weights are generated by the Halton sequence [27, 46], a generalized form of the one-dimensional Van der Corput sequence [17, 33], where we only take sampled points which sum up to one. This process involves generating a series of weights with low discrepancy and addressing the optimization problem for each weight set. Through this incremental search, we explore the Pareto front of the multi-objective optimization problem with a linear weighted sum [54, 38].

Constraints: The initial constraint within the ILP ensures the coverage of each element in $\widehat{\mathrm{PR}}(\mathcal{A}, \mathcal{T})$. This implies that for every polygon within $\widehat{\mathrm{PR}}(\mathcal{A}, \mathcal{T})$, we must ascertain which mpp is providing coverage. To achieve this, we extract the corresponding multi-polygon from mpp that shares the same object class, environment, and θ-interval. By "cover" we mean that a multi-polygon μ_i covers another polygon μ_j if $\mu_j \subseteq \mu_i$. Consequently, a binary matrix A is populated, possessing dimensions $N \times \mathrm{L_{mpp}}$, where N represents the number of polygons in $\widehat{\mathrm{PR}}(\mathcal{A}, \mathcal{T})$ and $\mathrm{L_{mpp}}$ denotes the number of mounted perception pipelines. The entry in the n-th row and l-th column of matrix A is denoted as a_{nl}, with $a_{nl} = 1$ indicating that polygon n is covered by mpp_l, and $a_{nl} = 0$ otherwise. Subsequently, another binary matrix, denoted as F, is constructed with dimensions $D \times \mathrm{L_{mpp}}$, where D corresponds to the number of mounting positions. Matrix F indicates which mounted perception pipelines share the same mounting positions. In a given row of F, all entries set to 1 signify mounted perception pipelines with identical mounting positions. Finally we can find the mounted perception pipelines which cover $\widehat{\mathrm{PR}}(\mathcal{A}, \mathcal{T})$, while minimizing certain cost c_j by solving the ILP in Eq. (3).

$$\begin{aligned} \min \quad & \sum_{i=1}^{L} \sum_{j=1}^{W} w_j c_j(\mathrm{mpp}_i) \cdot x_i \\ \text{s.t.} \quad & A \cdot x \geq [1 \quad \dots \quad 1]^{\mathrm{T}} \quad , \\ & F \cdot x \leq [1 \quad \dots \quad 1]^{\mathrm{T}} \quad , \\ & x_i \leq 1 \quad \forall i \in \{1, \dots, L\}, \\ & x_i \in \mathbb{N}_0 \quad \forall i \in \{1, \dots, L\}, \\ & \sum_{j=1}^{W} w_j = 1, \; w_j \geq 0, \; j = 1, \dots, W. \end{aligned} \tag{3}$$

F. Appendix: Design of experiments

TABLE II
VARIABLES, OPTIONS AND SOURCES FOR THE AV CO-DESIGN PROBLEM.

Variable	Option	Source
Vehicle bodies	Smart Fortwo, Chrysler Pacifica, Mercedes-Benz C63	[5]
Lidars	Velodyne: Alpha Prime, HDL 64, HDL 32; OS2: 128, 64	[60, 45]
Cameras	Basler: acA1600-gm, acA1500-um, acA7-gm; FLIR: Point Grey	[3, 23]
Object Detection Models	FCOS3D, Pointpillars	[14, 61, 32]
Mounting Orientation Yaw	$-135.0°$, $-90.0°$, $-45.0°$, $0.0°$, $45.0°$, $90.0°$, $135.0°$, $180.0°$,	[-]
Mounting Orientation Pitch	$0°$	[-]
Motion Planner	Lattice panner with A*, RRT, RRT*	[55, 1]
Computer	Jetson Nano, Orin Nano, Xavier NX, Orin NX, AGX Orin 64GB, AGX Orin 32GB, AGX Xavier 32GB	[43]

Catalogs in Tab. II: The 3D meshes of the car bodies are sourced from TurboSquid [57]. Real sensor measurements from the nuScenes open-source dataset [4], along with state-of-the-art 3D object detection algorithms from the MMDetection3D library [14], are used to determine the FNRs and the FPRs for different object classes. The mounting position options are visualized in Fig. 23. We utilize motion planners from the OMPL [55] and CommonRoad [1] libraries, including RRT, RRT*, and a lattice planner enhanced with motion primitives and an A* search algorithm. The three different motion planners operate with 1 s and 2 s planning horizons, which define the time into the future for which a planner calculates its trajectory.

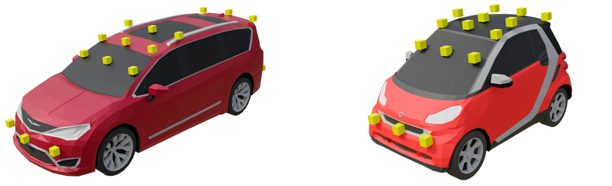
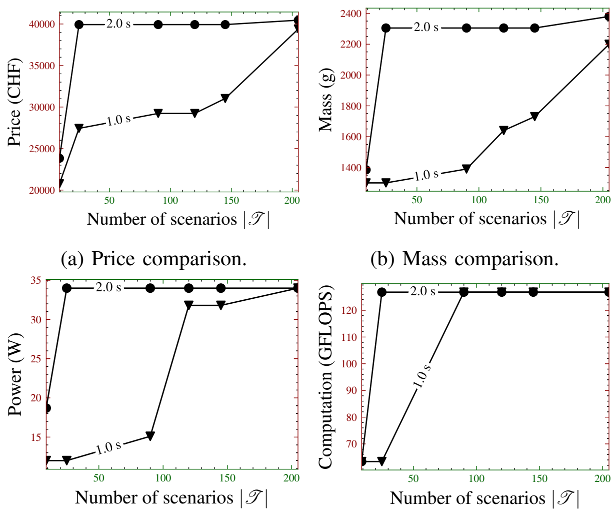


Fig. 23. Exemplary mounting positions for two different vehicles.

Remark. We acknowledge that the catalog may not represent the latest advances in motion planning and perception. The designer is free to create their own catalog.

G. Appendix: Results

In Fig. 24 we show the influence of higher planning horizon leading to higher resource requirements on the selected sensors and perception algorithms by fixing the motion planner and the vehicle body. The figure compares the resources required - power, mass, price, and computation - for different tasks for planning horizons of one and two seconds. Each point represents the minimum resource solution for a given task and time horizon. In Fig. 25, we keep the vehicle body and planning horizon constant, but compare the resource trade-offs of using RRT* versus a lattice planner. This comparison aims to visualize the resource differences between motion planners, as expected from Fig. 2, and to highlight the impact of the planning strategy on the sensor selection and placement process.

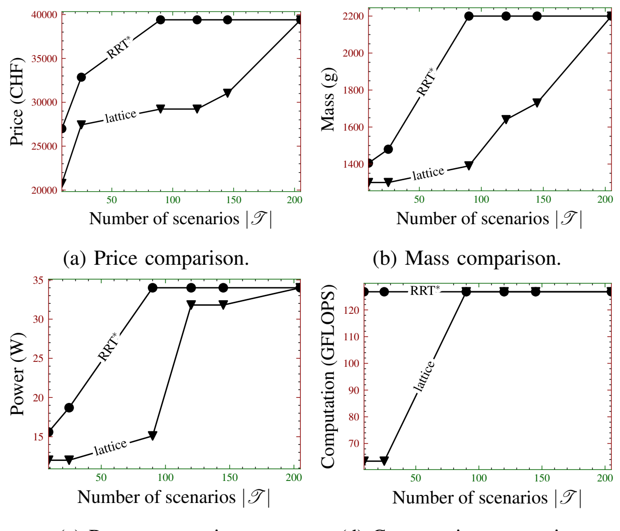


(a) Price comparison. (b) Mass comparison.

(c) Power comparison. (d) Computation comparison.

Fig. 24. Higher planning horizons for the same planner and vehicle body require more resources for different tasks. Here we show the lattice planner with A* search and a hatchback vehicle body.

In Figs. 6, 27 and 31, we display the implementations for the minimal computation solutions. The NVIDIA Jetson Orin Nano was chosen alongside the lattice motion planner using A* search for all cases. Notably, a camera sensor was never chosen for these solutions. The implementations aiming for minimal mass are shown in Figs. 5, 29 and 30, where there is a notable preference for cameras, predominantly coupled with the most powerful computing unit, the NVIDIA Jetson AGX Orin 64. In Figs. 7, 26 and 28 we present the implementations for the AV design with minimal power needs. Similarly as for the minimal computation, only one or two lidars are chosen.



(a) Price comparison. (b) Mass comparison.

(c) Power comparison. (d) Computation comparison.

Fig. 25. Resource comparison between RRT* planner and lattice planner with A* search for the same vehicle body (hatchback) and tasks.

Moreover, we present implementations tailored for the most cost-effective AV design in Figs. 26, 27, 29 and 31. Every implementation features at least one lidar sensor. Except for the cases highlighted in Figs. 26 and 31, corresponding to the most complex task and the task with restricted prior, all configurations additionally incorporate camera sensors. For the most complex task containing the most scenarios, highest nominal speed and no prior restriction, each implementation includes at least one lidar sensor.

H. Appendix: Discussion

Our results show that increased task complexity, manifested by more scenarios, higher speeds, or broader prior knowledge, requires more resources for AV design. Each additional scenario may introduce new occupancy queries and prior knowledge, expanding the perception requirements. Higher speeds require sensor pipelines to detect objects at greater distances to account for the faster movement of the AV and the faster dynamics of the surrounding objects. In addition, a wider range of possible class configurations based on prior knowledge increases the perception requirements, calling for more advanced sensor pipelines that consume additional resources.

Motion planners that generate broader occupancy query distributions require enhanced sensing capabilities, thereby increasing the resource allocation to sensor pipelines to provide the required information. The broader occupancy query distributions result from either extended planning horizons, as illustrated in Fig. 24, or the inherent strategy of the motion planner, as illustrated in Fig. 2 and Fig. 25. In the optimization process for minimal resource solutions at the lowest average speeds, the RRT* planner was consistently not selected. However, when the requirement shifted towards achieving the highest average speeds, the RRT* planner became the exclusive choice, paired with the vehicle body with the highest

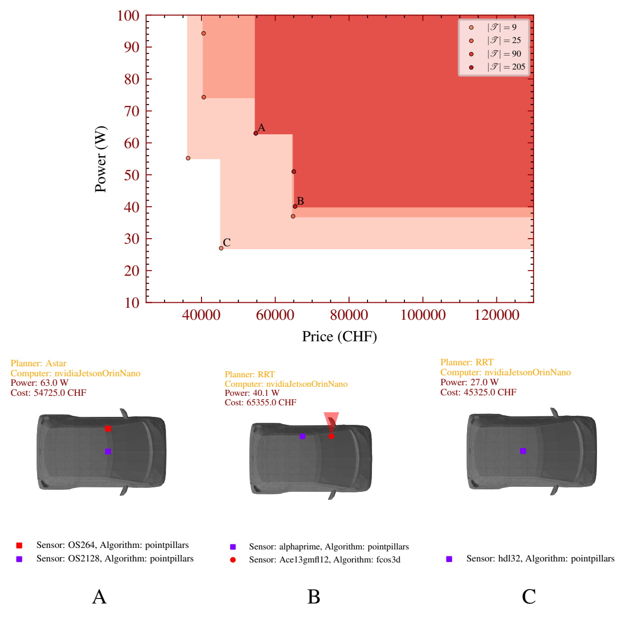


Fig. 26. Pareto front of price and power across tasks, where tasks with more scenarios demand more resources and encompass those with fewer scenarios. Implementations for point A, B, and C are visualized vertically. B and C indicate the least power usage for the most and least complex tasks, respectively, while A shows the minimum price for the most complex task.

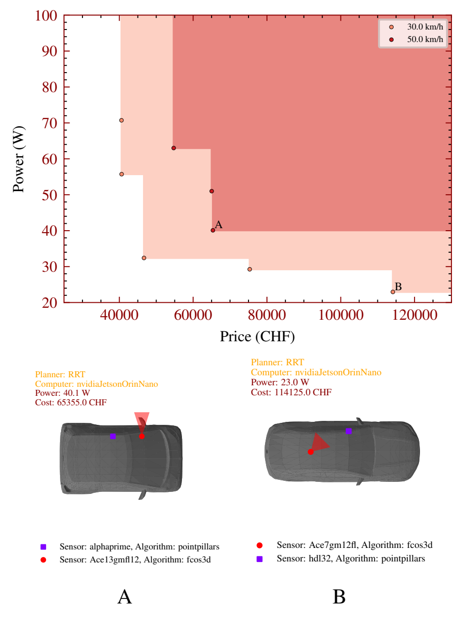


Fig. 28. Pareto front of price and power usage across task velocities, where higher nominal velocities for the same set of scenarios require more resources. Implementations for points A and B are visualized vertically. A and B indicate lowest power usage for 50 $\mathrm{km/h}$ and 30 $\mathrm{km/h}$ nominal velocities, respectively.

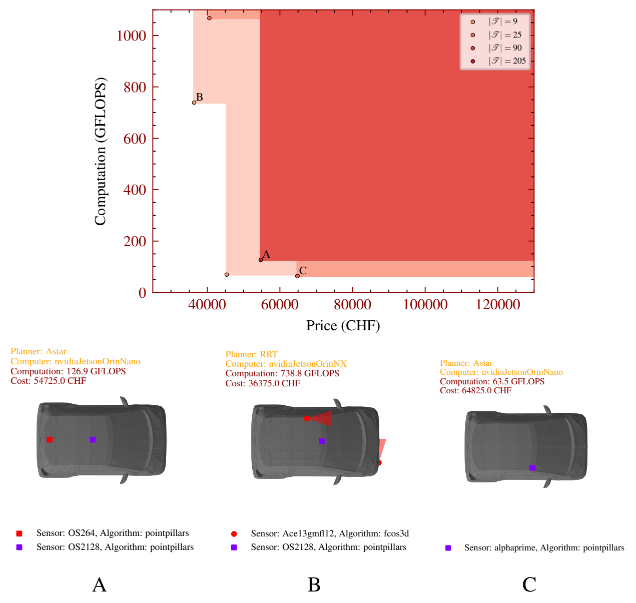


Fig. 27. Pareto front of price and computation across tasks, where more scenarios demand more resources . Implementations plots for point A, B, and C are visualized vertically. A and C indicate the least computation usage for the most and least complex tasks, respectively, while B shows the minimum price for the least complex task.

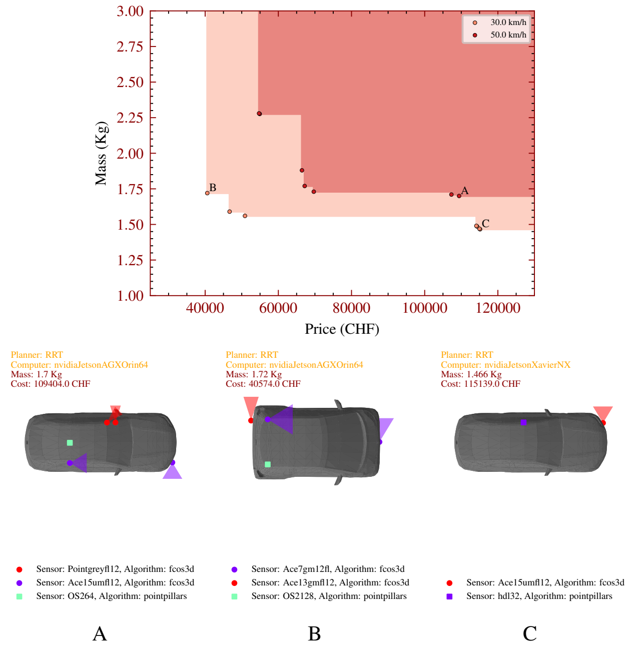


Fig. 29. Pareto front of price and mass across task velocities, where higher nominal velocities for the same set of scenarios require more resourses. Implementations for points A, B and C are visualized vertically. A and C indicate lowest mass for 50 $\mathrm{km/h}$ and 30 kmh nominal velocities, respectively. B indicates lowest price for 30 $\mathrm{km/h}$ nominal speed.

acceleration. This pattern suggests that while the RRT* planner demands more resources, it stands out as the most efficient option for optimizing average speed in the task. Our analysis further confirms that to minimize computational requirements in AV design, lidar sensors emerge as the preferred choice due to their perception algorithms requiring fewer operations per second. Conversely, to reduce mass or cost, camera sensors are preferred due to their lighter weight and lower price compared to lidars. However, designs addressing the most complex task always include lidar sensors. This underscores the superior capability of lidar-equipped sensor pipelines due to their lower FNR and FPR across a wider range of class configurations.

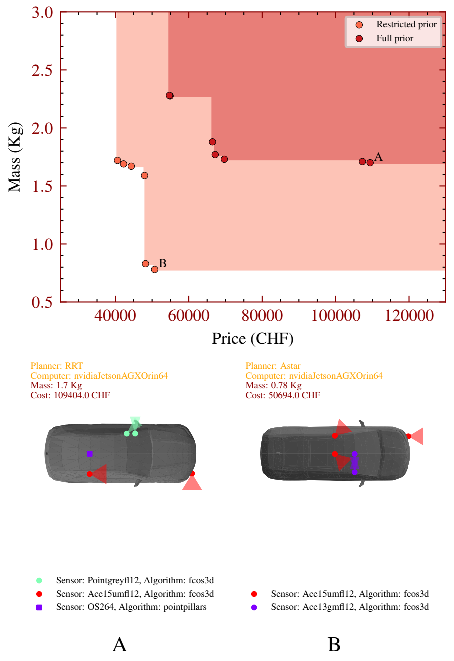


Fig. 30. Pareto front of price and mass across priors, where priors with more class configurations require more resources. Implementations for points A and B are visualized vertically. A and B indicate the lowest mass for the least and most restricted prior, respectively.

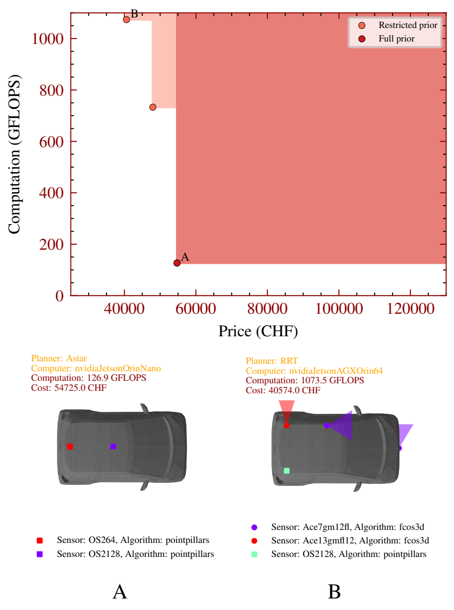


Fig. 31. Pareto front of price and computation across priors, where priors with more class configurations require more resources. Implementations for points A and B are visualized vertically. A indicates the lowest computation for both priors (same implementation) and B indicates lowest price for the most restricted prior.

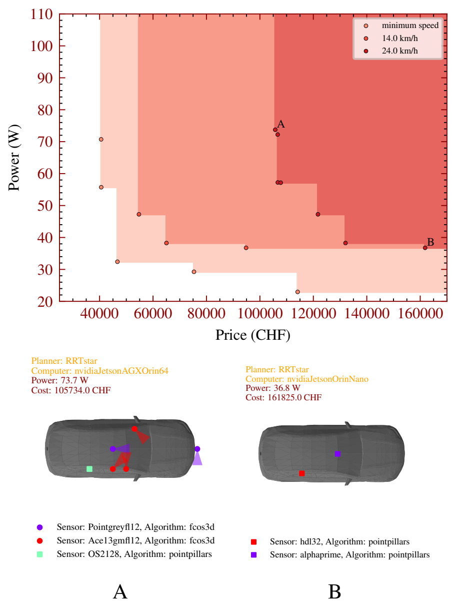


Fig. 32. Pareto front of price and power consumption across different average speeds, where planners providing higher average speed across all scenarios (30 $\mathrm{km/h}$ nominal speed) demand more resources. Implementations plots for points A and B are visualized vertically. A and B indicate the lowest price and lowest power for the highest average speed, respectively.

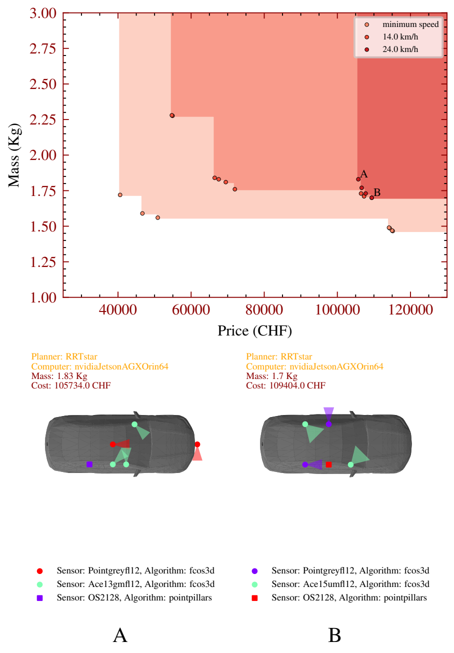


Fig. 33. Pareto front of price and mass across different average speeds, where planners providing higher average speed across all scenarios (30 $\mathrm{km/h}$ nominal speed) demand more resources. Implementations for points A and B are visualized vertically. A and B indicate the lowest price and mass for the highest average speed, respectively.