Resource-Efficient Task-Driven Co-Design of Perception and Decision Making in Autonomous Robots

Author(s):
Milojevic, Dejan; Zardini, Gioele; Elser, Miriam; Censi, Andrea; Frazzoli, Emilio

Publication date:
2024

Permanent link:
https://doi.org/10.3929/ethz-b-000672201

Originally published in:
IEEE Transactions on Robotics

Resource-Efficient Task-Driven Co-Design of Perception and Decision Making in Autonomous Robots

Dejan Milojevic[1,2], Gioele Zardini[3], Miriam Elser[2], Andrea Censi[1], Emilio Frazzoli[1]

***Abstract*—This paper discusses the integration challenges and strategies for designing mobile robots, by focusing on the task-driven, optimal selection of hardware and software to balance safety, efficiency, and minimal usage of resources such as costs, energy, computational requirements, and weight. We emphasize the interplay between perception and motion planning in decision-making, leveraging 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. We introduce the concept of occupancy queries to quantify the perception requirements for sampling-based motion planners, and propose an Integer Linear Programming (ILP) approach for efficient sensor and algorithm selection and placement. This forms the basis for a co-design optimization that includes the robot body, motion planner, perception pipeline, and computing unit. A case study on developing an Autonomous Vehicle (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. The study demonstrates that resource prioritization influences sensor choice: cameras are preferred for cost-effective and lightweight designs, while lidar sensors are chosen for better energy and computational efficiency.**

***Index Terms*—Co-design, mobile robots, sensor selection.**

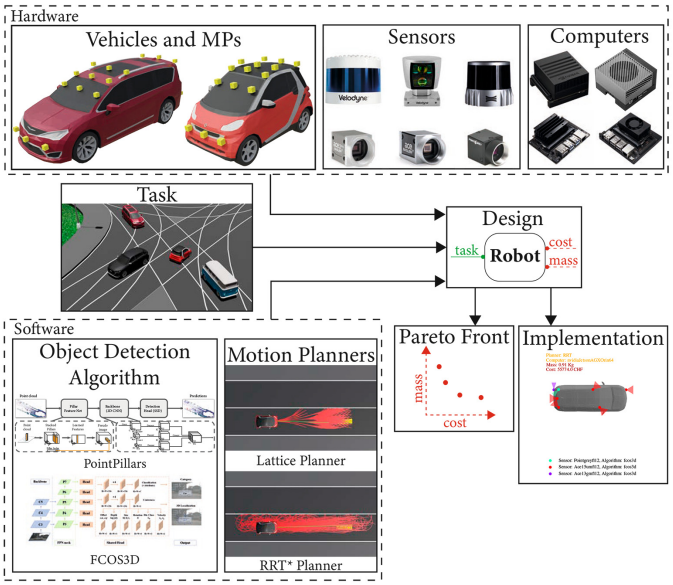


Fig. 1. Graphical illustration of the informal problem definition for designing an AV for urban driving tasks, based on a catalog of hardware and software components with an emphasis on minimizing resources.

I. Introduction

Embodied intelligent systems hold great promise for addressing critical societal challenges and enhancing our daily lives. Whether revolutionizing mobility via autonomous driving, or supply-chain via automated logistics, this technology will impact the world we live in. However, realizing the full potential of these advances depends on the efficient design and safe operation of such systems. The complexity of developing embodied intelligence lies in selecting the optimal mix of interdependent hardware and software components. The final design must ensure safety and efficient task performance while minimizing the resources required for design and operation, such as cost, power consumption, computation, and weight.

In the context of robot perception this involves the choice and placement of sensors and the selection of algorithms which process the sensor measurements. Clearly, hardware and software choices are interdependent and influence each other. Moreover, they are interconnected with other systems such as the computing units, actuators, or decision making. Indeed, a controller relies on the reference created by a motion planner, which is based on state estimates from an estimator, which in turn depends on sensor data and power supply. In addition, the integration of perception software such as object detection algorithms introduces uncertainties in the algorithm output that must be carefully considered in the design.

To tackle these intricate issues, a comprehensive framework which applies abstract reasoning across different areas, and balances functional requirements with resource constraints and trade-offs is needed. This work outlines our method for addressing the complex task of robot co-design by tackling such challenges.

Informal Problem Definition: The problem features the definition of catalogs with both hardware and software components necessary for the robot design. These include:

- *Robot Bodies*: a selection of mobile robot chassis, each with its shape and actuators.
- *Sensor Mounting Configurations*: options for mounting sensors on each robot body.
- *Perception Pipeline*: combinations of sensors and their corresponding perception algorithms for processing data.
- *Decision-Making Algorithms*: software for determining the robot's motion and actions to complete the task.
- *Computing Units*: catalog of computing resources to support the software's operational needs.

Each component is linked to a certain resource, including monetary cost (e.g., sensor prices), power consumption (e.g., computer energy requirements), computational resources (e.g., flops required by perception algorithms), and mass (e.g., sensor weight). The problem is to co-design the robot with a particular task by selecting from these components to minimize resource

[1]Institute for Dynamic Systems and Control, ETH Zürich, Zürich, Switzerland {dejanmi, acensi, efrazzoli}@ethz.ch
[2]Chemical Energy Carriers and Vehicle Systems Laboratory, Empa - Swiss Federal Laboratories for Materials Science and Technology, Dübendorf, Switzerland miriam.elser@empa.ch
[3]Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA, USA gzardini@mit.edu

usage while ensuring feasibility. A graphical illustration of the informal problem definition is shown in Fig. 1.

Assumptions: To address the robot co-design problem, we make 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. This is a fundamental assumption, given that to choose the minimal sensors, we need to know exactly the information the planner needs. Finally, we assume that object detections of the perception layer are binary in nature: objects are either detected or not, based on the detection received from the perception pipelines.

Note that our methodology is adaptable to different software architectures and motion planners, as long as one can acquire the information needed by the robot to complete the task, which must be provided by the perception pipeline. We illustrate how such information can be obtained using sampling-based motion planners through the concept of *occupancy queries*. We focus on object detection as the perception task. The approach could be extended to additional perception tasks such as localization.

Contribution: The contributions can be summarized as follows. First, we explore the interconnections between perception pipelines and 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 a robot, via set cover problems. Third, we develop a robot co-design framework leveraging a monotone theory of co-design optimization, promoting the robot 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 are highlighted in several studies [1]–[4]. Such challenges primarily revolve around developing a framework which can accommodate the complex nature of cyber-physical systems, encompassing both software and hardware within dynamic environments [1], [2]. Additionally, there is a significant need for algorithms that can efficiently navigate the heterogeneous design landscapes available [2]. The work presented in [4] highlights the difficulty of integrating diverse components into robotic systems, and determining the specific information needs for a robot to fulfill a given task. 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. For instance, [5] leveraged convex optimization techniques with the objective of reducing the estimation error of certain parameters. In [6], the authors introduce a stochastic algorithm designed to optimize sensor scheduling and improve coverage. A greedy method for sensor selection aimed at state estimation in linear dynamical systems, utilizing Kalman filtering, is described in [7]. Furthermore, [8] proposed a novel distributed online greedy algorithm selecting sensors based on the real-time feedback of their utility, targeting the maximization of information richness and energy efficiency. One of the earliest approaches to sensor selection in robotics was presented in [9], which introduced a real-time method using stochastic dynamic programming tailored to robotic systems. Erdmann proposed a sensor selection strategy deeply integrated with a robot's task and planning requirements, based on the hypothetical premise of an ideal sensor fulfilling all informational needs for plan formulation [10]. Geometric considerations in sensor selection are explored in [11], which employs Gaussian models to approximate uncertainties in the sensor-environment interaction. Work proposed in [12] addresses an LQG control co-design problem, simultaneously developing control and sensing strategies with resource limitations. Furthermore, [13], [14] present a sensor selection framework specifically designed for localization and mapping, and [15], [16] focus on placement, orientation, and architecture designs in the context of AVs. Additionally, [16] presents a machine learning based framework for generating perception architecture designs for AVs, simultaneously optimizing sensor positions and orientations, detection algorithms, and fusion algorithms for a given target vehicle. Finally, a learning algorithm for sensor placement in the context of soft robotics is presented in [17]. Despite the presented advancements, current literature does not fully address the integrated selection and placement of sensor hardware in conjunction with the choice of perception algorithms. There is a notable gap in discussions on optimizing sensor selection 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, and tend to neglect the impact of sensor uncertainties.

Design space exploration has gathered substantial interest in robotics research, with significant contributions aiming to delineate the boundaries of sensor and actuator requirements for effective robotic planning. In this context, [18], [19] examined the minimal necessary sensors or actuators by assessing the consequences of their degradation on robotic planning capabilities, and [20] proposed an innovative method to identify sensors sufficient for resolving planning problems, employing an upper cover concept to condense sensor data and expedite the exploration process. Nardi introduced a practical approach for navigating design spaces within multi-objective optimization frameworks, specifically applied to hardware design challenges [21]. Furthermore, comparative analyses of robotic components have been advanced through the works of O'Kane, Lavalle, and Censi [22]–[24], which explore methodologies for assessing sensor performance and establishing criteria for comparisons. The notion of sensor dominance and the subsequent development of a sensor lattice [22],

[23] provide a structure means to rank sensors according to their task efficacy. Additionally, [24] conducts a power-performance analysis, comparing different sensor families for specific tasks. In a similar context, [25] introduces a benchmark for evaluating SLAM algorithms in robotics, utilizing metrics such as execution time and energy consumption. Trade-off analysis in design choices is examined in contributions such as [26]–[28]. In [26], a methodology is introduced for exploring trade-offs between performance and resource utilization in the design of mobile robots. On the other hand, [27] outlines design principles aimed at enhancing energy efficiency in legged robots. The trade-off between design complexity of a robot and plan execution are explored in [28].

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 a novel approach utilizing linear temporal logic to transform high-level design specifications into tangible selections of robot components from an extensive library, bridging the gap between abstract design requirements and practical component choices, streamlining the design process [29]. Furthermore, [30] develops a heuristic algorithm specifically targeted at the creation of robotic devices tailored to follow predefined motion trajectories accurately. The algorithm navigates through the vast array of possible configurations of modular components to pinpoint the ones which best match the desired trajectories. In a similar vein, [31] 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. They overlook several critical co-design challenges that must be addressed to achieve a comprehensive and automated design process, as identified in [1]–[4], [32], such as a) formalizing heterogeneous components across varxing 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 [33], [34] and builds on our series of previous works [35]–[38], 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 $\mathcal{B}$ (which includes considerations of shape, actuators, and hardware configurations) with configuration space Γ, and its software, the agent, which we call $\mathcal{A}$.

TABLE I
Table of symbols

Symbol	Meaning
$\mathcal{A}$	decision-making agent
$\mathrm{appear} \in \mathrm{AP}$	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
$\mathrm{env} \in \mathbb{E}$	environment
$\mathrm{mo} \in \mathrm{MO}$	mounting orientations (yaw and pitch) in $\mathbb{R}^2$
$\mathrm{mp} \in \mathrm{MP}$	mounting position in $\mathbb{R}^3$
$\mathrm{mpp} \in \mathbb{MPP}$	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
$\mathrm{pp} \in \mathrm{PP}$	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

Agent: We assume that the agent $\mathcal{A}$ consists of a modular software architecture, comprising perception, state estimation, motion planning, and control [39]. 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. In particular, 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.

Remark 2. It is crucial to differentiate between the robot's 3D shape, $\mathrm{SH} \in \mathbb{R}^3$, which includes its elevation, and the robot's footprint, $\mathrm{sh}_{\mathcal{R}}(\gamma) \in \mathbb{R}^2$ for a given configuration $\gamma \in \Gamma$. The footprint is essentially a projection of the robot's shape onto the ground plane. This distinction becomes particularly relevant in later discussions, as outlined in Sec. IV-B.

The examination of the robot's structural framework $\mathcal{B}$ involves assessing its mounting positions mp (with $\mathrm{mp} \in \mathrm{MP}$

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

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

B. 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 $\gamma_{\mathrm{start}} \in \Gamma$ and seeks to reach a goal area denoted as $\mathcal{G}$.[2]

The environment may include both dynamic and static objects. Dynamic objects encompass moving entities such as robots, vehicles, and humans. On the other hand, static objects consist of stationary elements such as trees or buildings.

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 $\mathrm{AP} \sim f_{\mathrm{AP}}(\mathrm{appear})$.

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}.

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

In addition, the operational environment encompasses various weather and light conditions. Such conditions are collectively referred to as *environmental conditions*, denoted as env. For simplicity, we use the term environmental conditions to encapsulate a range of possibilities, which include discrete values such as day and night time or rain and sunny conditions. Without loss of generality, this can also refer to continuous values such as rain density or time of day. The entire set of possible environmental conditions is denoted as $\mathbb{E}$.

Definition 5 (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 of objects follows a Poisson distribution, where λ_i represents the expected number of objects for a given class.

Definition 6 (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 $\Gamma \sim f_\Gamma(\gamma_{\mathrm{start}})$, $\mathbb{R}^2 \sim f_{\mathbb{R}^2}(\mathcal{G})$ and $\mathbb{E} \sim f_{\mathbb{E}}(\mathrm{env})$ 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 $S = \langle \mathcal{W}, \gamma_{\mathrm{start}}, \mathcal{G}, \mathrm{env}, \{C_i\}_{i \in \{1,\dots,M\}} \rangle$ 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. In this work, 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 7 (Task)**.** A *task* $\mathcal{T}$ is a set of scenario instances.

In the upcoming section, we detail the software architecture previously introduced. Here, the primary focus is to elaborate on the information required by the agent from the perception system to successfully accomplish the robot's task.

C. Modeling an agent

In a common agent's architecture, including perception, state estimation, motion planning, and control, the dependency of motion planning on perception data underscores the importance of defining the precise "information" necessary for trajectory planning. Identifying the "minimum" required sensors and perception algorithms for a robot, given a particular motion planner, necessitates this specificity. Motion planning algorithms typically need a notion of the obstacle free configuration space to compute a reference trajectory. Combinatorial motion planning [39]–[43] and optimization-based motion planning [39], [44]–[50] 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 [39], [40], [44] 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.

[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.

Definition 8 (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 $\mathbb{E}$: $\Psi \coloneqq \Gamma \times \mathbb{R}^+ \times \mathbb{E}$.

Different motion planners produce different distributions of queries. Planners such as RRT* 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, 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*.

Example 9. Consider an AV paired with the employment of an RRT* planner. Starting from the initial configuration, the RRT* planner incrementally constructs a tree by randomly sampling configurations from the search space. At each step, a configuration is drawn and an attempt is made to establish a connection between the drawn configuration and the nearest configuration in the existing tree. If this connection proves to be feasible, a new branch is added to the tree. The feasibility assessment of a connection includes collision checking, which considers the projection of the robot as well as of the obstacles onto the workspace. Whenever a collision check is executed, its configuration is saved as a query. The planning procedure is illustrated in Fig. 2a.

Example 10. The same approach works in different contexts. Considering a lattice planner, paired with motion primitives and A* search, every state driven by the motion primitives for which a collision check is done to build the lattice, will be saved as a query, as shown in Fig. 2b.

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 [51]. Technically, for an agent $\mathcal{A}$ and a scenario instance S, the set of queries which are generated by the agent's state inference process in the scenario is denoted by $\mathrm{plan}(\mathcal{A}, S) \subseteq \Psi$.

Definition 11 (Plan)**.** The function plan maps an agent $\mathcal{A}$ and a scenario instance S to a set of queries.

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

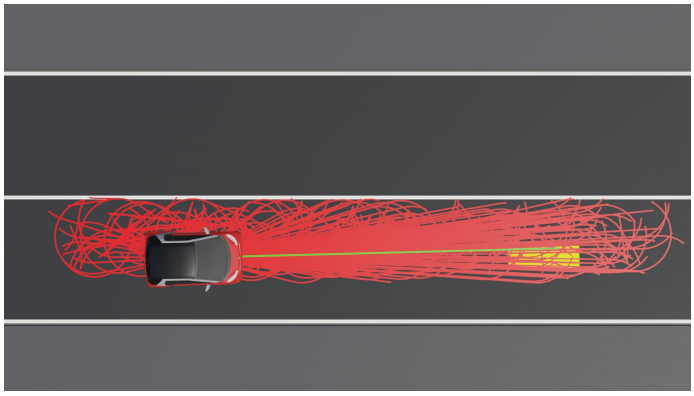
where $\mathbb{A}$ is the set of all possible agents, $\mathbb{T}$ is the set of all possible scenario instances and $\Psi_S \subseteq \Psi$.

Definition 12 (Task Queries)**.** Given a task $\mathcal{T}$, the *task queries* generated by an agent $\mathcal{A}$ are the union over all queries of all the scenario instances in the task:

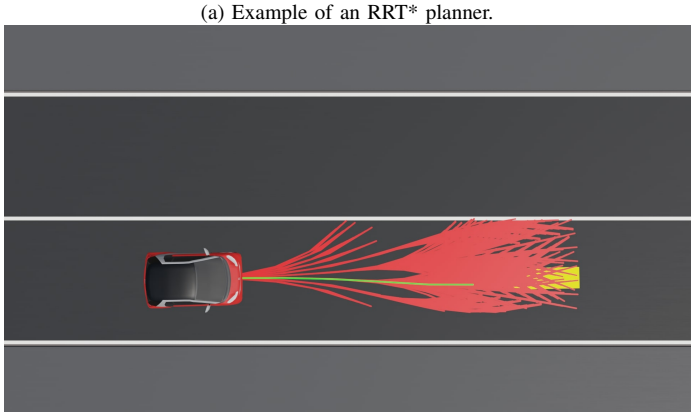
$$\begin{aligned} \mathrm{tq}\colon \texttt{POW}(\mathbb{T}) \times \mathbb{A} &\to \texttt{POW}(\Psi), \\ \langle \mathcal{T}, \mathcal{A} \rangle &\mapsto \bigcup_{S \in \mathcal{T}} \mathrm{plan}(\mathcal{A}, S), \end{aligned} \tag{1}$$

such that $\mathrm{tq}(\mathcal{A}, \mathcal{T}) \subseteq \Psi$.

Given the information required by the agent in the form of queries, the next section presents a representation of the perception pipeline aimed at providing such information. Furthermore, we define the requirements for perception that must be fulfilled to enable the agent to accomplish the task.



(a) Example of an RRT* planner.



(b) Example of a lattice planner, paired with motion primitives and A* search.

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.

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:

$$\begin{aligned} \mathrm{fnr}\colon \mathcal{Q}_i \times \mathrm{AP}_i \times \mathrm{PP} \times \mathbb{E} &\to \texttt{POW}(I), \\ \langle q_i, \mathrm{appear}_i, \mathrm{pp}_j, \mathrm{env} \rangle &\mapsto [a, b]. \end{aligned} \tag{2}$$

$$\begin{aligned} \mathrm{fpr}\colon \mathcal{Q}_i \times \mathrm{AP}_i \times \mathrm{PP} \times \mathbb{E} &\to \texttt{POW}(I), \\ \langle q_i, \mathrm{appear}_i, \mathrm{pp}_j, \mathrm{env} \rangle &\mapsto [a, b]. \end{aligned} \tag{3}$$

The set I is the set of all intervals: $[a, b] \subseteq \mathbb{R} : 0 \leq a \leq b \leq 1$. 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 with two distinct perception pipelines is shown in Fig. 3.

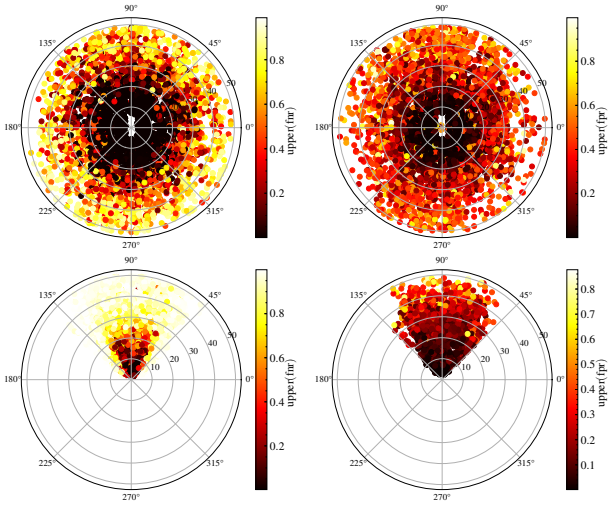


Fig. 3. Comparison of the perception performance of two pipelines: Velodyne HDL32E lidar with PointPillars detection model (top plots) [52] and Basler acA1600-60gc camera with FCOS3D detection model (bottom plots) [53]. Left plots show FNRs and right plots FPRs, highlighting the upper bounds of confidence intervals against radial distance r and relative orientation θ between sensor and object class in polar coordinates. Data is from the nuScenes dataset [54], using models from the MMDetection3D [55] library.

Example 13. Consider a perception pipeline consisting of a Velodyne Alpha Prime lidar paired with a PointPillars 3D object detection model [52]. The FNR and FPR for detecting a pedestrian at the configuration $q_{\text{pedestrian}} = \langle x, y, \theta \rangle$ relative to the sensor lie within $[a_{\text{fnr}}, b_{\text{fnr}}]$ and $[a_{\text{fpr}}, b_{\text{fpr}}]$, respectively.

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 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 $\psi = \langle \gamma, t, \text{env} \rangle$ 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.

Example 14. Consider an AV approaching a four-way intersection. The motion planner has generated a query for the intersection with a specific configuration γ at time t_0. The scenario prior specifies that cars drive only in the direction of the lane and can turn left, right, or drive straight. To identify all car configurations resulting in a collision with the autonomous vehicle at time t_0 and configuration γ, we must compute all car configurations at time 0 where feasible trajectories exist, which would lead to such a collision. These trajectories encompass various possibilities, including cars approaching from the left or right. An illustration of the example is shown in Fig. 4.

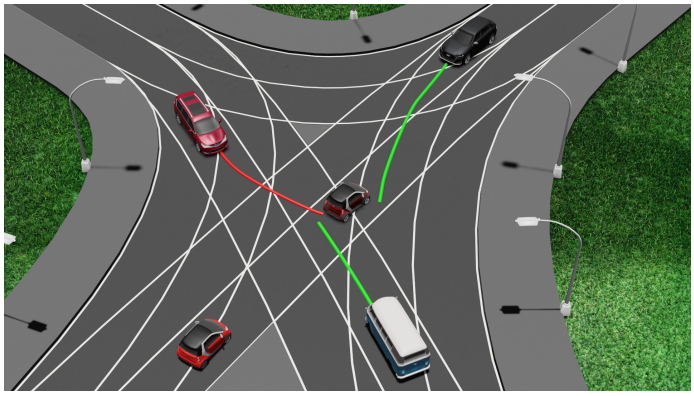


Fig. 4. This figure shows class configurations at time 0 leading to potential collisions with a robot at a specific query $\psi = \langle \gamma, t, \text{env} \rangle$. The robot is depicted as a small red AV on the left and the robot's future configuration γ from the query is the transparent AV in 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.

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

Definition 15 (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 $\text{sh}_{\mathcal{R}}(\gamma)$ and $\text{sh}_i(q_i)$.

$$\begin{aligned} \text{collision}\colon \Gamma \times \mathbb{C} &\to \texttt{POW}(\mathcal{Q}), \\ \langle \gamma, C_i \rangle &\mapsto \Theta_i, \end{aligned} \tag{4}$$

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

Definition 16 (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: $\bar{\mathcal{Q}}_i \coloneqq \mathcal{Q}_i \times \texttt{POW}(\mathcal{Q}_i)$.

Definition 17 (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*

$$\begin{aligned} \text{pcp}\colon \texttt{POW}(\Psi) \times \mathbb{C} &\to \texttt{POW}(\bar{\mathcal{Q}}), \\ \langle \Psi_0, C_i \rangle &\mapsto \bar{\Theta}_i, \end{aligned} \tag{5}$$

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

Definition 18 (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 $\mathcal{P}_i$.

$$\begin{aligned}\text{priorcheck}\colon \mathtt{POW}(\bar{\mathcal{Q}}_i) \times \mathtt{POW}(\mathcal{Q}_i) &\to \mathtt{POW}(\mathcal{Q}_i),\\ \langle \bar{\Theta}_i, \mathcal{P}_i\rangle &\mapsto \Theta_i,\end{aligned} \tag{6}$$

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

Definition 19 (Task Perception Requirements)**.** The perception requirements for an agent $\mathcal{A}$ undertaking a task $\mathcal{T}$ are defined as the mapping from task queries $\mathrm{tq}(\mathcal{A}, \mathcal{T})$ 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 $\mathrm{priorcheck}$:

$$\mathrm{PR}\colon \mathbb{A} \times \mathtt{POW}(\mathbb{T}) \to \prod_{\mathrm{env}\in\mathbb{E}} \prod_{k\in\{1,\ldots \mathrm{K}_{\mathrm{class}}\}} \mathtt{POW}(\mathcal{Q}_k), \tag{7}$$

where $\mathrm{K}_{\mathrm{class}}$ 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 $\mathrm{PR}(\mathcal{A}, \mathcal{T})$, we express this as $\mathrm{PR}(\mathcal{A}, \mathcal{T}, C_i, \mathrm{env})$, indicating that $\mathrm{PR}(\mathcal{A}, \mathcal{T}, C_i, \mathrm{env}) \subseteq \mathcal{Q}_i$.

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 [34], [56]. The primary objective is the minimization of resource consumption, which includes power consumption, robot body mass, cost and computing resources. Sec. IV-A introduces the basic principles of co-design. Subsequently, in Sects. IV-B and IV-C we address task-oriented co-design of a complete mobile robot. Sects. IV-C and IV-D addresses the sensor selection and placement problem, which forms the core of the entire optimization, using the formulations introduced in Sec. III.

A. 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 [57]).

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

Definition 20. Given partially ordered sets (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:

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

where $(\cdot)^{\mathrm{op}}$ 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. 5.

Remark 21 (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 22 (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 [32], [34]–[38].

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 23. 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 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 [32], [34].

[3]A detailed list of compositions is provided in [32], [34]. Formally, their specification makes the category of design problems a traced monoidal category, with locally posetal structure.

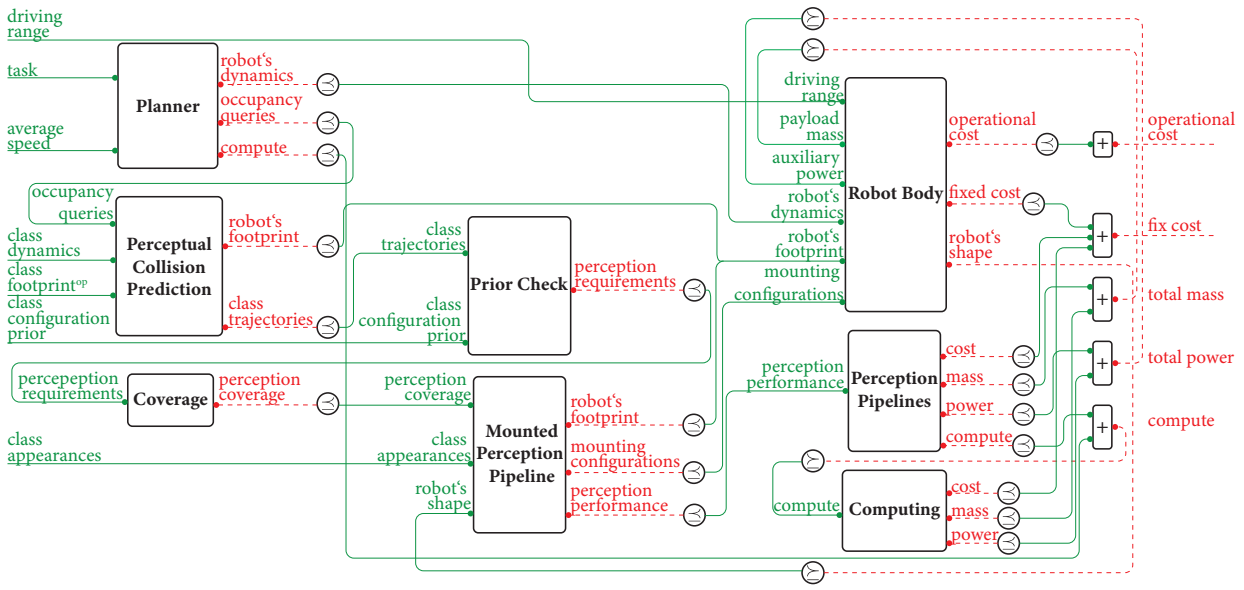


Fig. 5. The co-design diagram for the design of a mobile robot tailored to accomplish a task, including a collection of scenario instances and class instances, aiming to achieve specified average speed and driving range. The class instances include class dynamics, class footprint$^{\text{op}}$, class appearances, and class configurations prior. The objective is to minimize the fix cost and operational cost.

B. Modeling from task to perception requirements

First, we describe the **Planner** MDPI in Fig. 6, representing 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, as detailed in Lemma 24. 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. 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 24. The task occupancy queries tq is monotone in the task, as shown in Fig. 6.

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

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

■

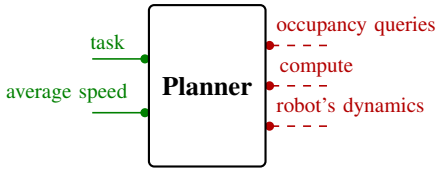


Fig. 6. The planner MDPI which implements a motion planner for the robot to accomplish scenario instances of a task and thereby providing average speed, while requiring occupancy queries Ψ, compute and robot's dynamics

The **Perceptual Collision Prediction** MDPI, visualized in Fig. 7, 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. Again, the minimum turning radius is treated the opposite of a poset. The class footprint$^{\text{op}}$ 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 25 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 26, 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$^{\text{op}}$

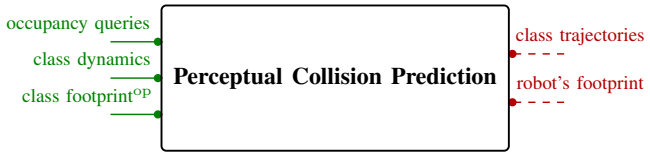


Fig. 7. The Perceptual Collision Prediction MDPI which implements the function pcp. The functionalities are the occupancy queries for the planner, the class dynamics and the class footprint$^{\text{OP}}$. The required resources are the class trajectories and robot's footprint.

indicates earlier collisions with the robot, thus generating fewer class trajectories. This inverse relationship uses the opposite of a poset for the class footprint$^{\text{OP}}$.

Lemma 25. The class trajectories from pcp are monotone with respect to the occupancy queries as shown in Fig. 7.

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

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

■

Lemma 26. The robot's footprint is monotone with respect to the occupancy queries as shown in Fig. 7.

Proof. A larger robot's footprint can exclude more class trajectories, according to Def. 15 and Def. 17. ■

The **Prior Check** MDPI, illustrated in Fig. 8, describes the priorcheck function as outlined in Def. 18. 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. According to Lemma 27, priors that encompass more class configurations tend to filter out fewer configurations during priorcheck, resulting in more perception requirements. Given the relations established in Lemma 24 and Lemma 25, where more complex tasks generate more class trajectories, it follows, as demonstrated in Lemma 28, that increased task complexity (more class trajectories) also amplifies the perception requirements.

Lemma 27. The class configurations in the class trajectories are monotone with respect to the class configurations prior as shown in Fig. 8.

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 28. The perception requirements PR are monotone in the task, respectively in the class trajectories (Fig. 8).

Proof. Consider two tasks $\mathcal{T}_1 \subseteq \mathcal{T}_2$. From Lemma 24 we know that occupancy queries are monotone in the task and from Lemma 25 we know that class trajectories are monotone with the queries. We have

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

■

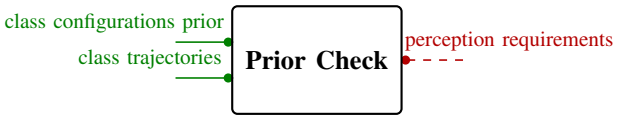


Fig. 8. The Prior Check MDPI, which implements priorcheck, provides class configurations prior and class trajectories functionalities and requires perception requirements.

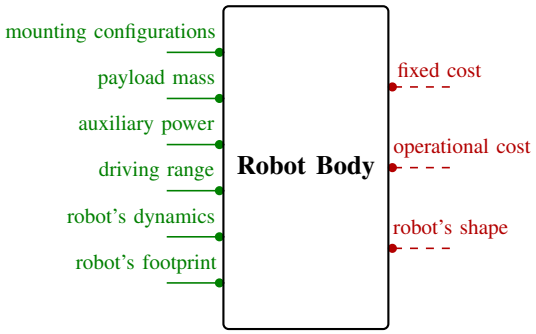


Fig. 9. The Robot body MDPI which provides the dynamics dyn, the mounting configurations for sensors each in SE(3), the body footprint $\text{sh}_{\mathcal{R}}$, the payload mass in kg, the auxilary power in W and the driving range in m, while requiring robot's shape SH, fixed cost in CHF and operational costs in CHF/m.

The **Robot body** MDPI in Fig. 9 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. 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 SE(3), the robot's footprint $\text{sh}_{\mathcal{R}}$ 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. 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. 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. The **Computing** MDPI, visualized in Fig. 10, 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. As the demand for compute increases to accommodate more sophisticated software algorithms or larger data volumes, the specifications

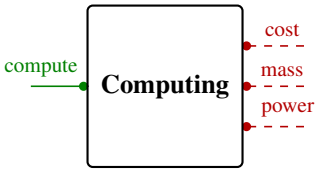


Fig. 10. The Computing MDPI which implements the computing units. It provides compute and requires cost in CHF, the mass in kg and power consumption in W.

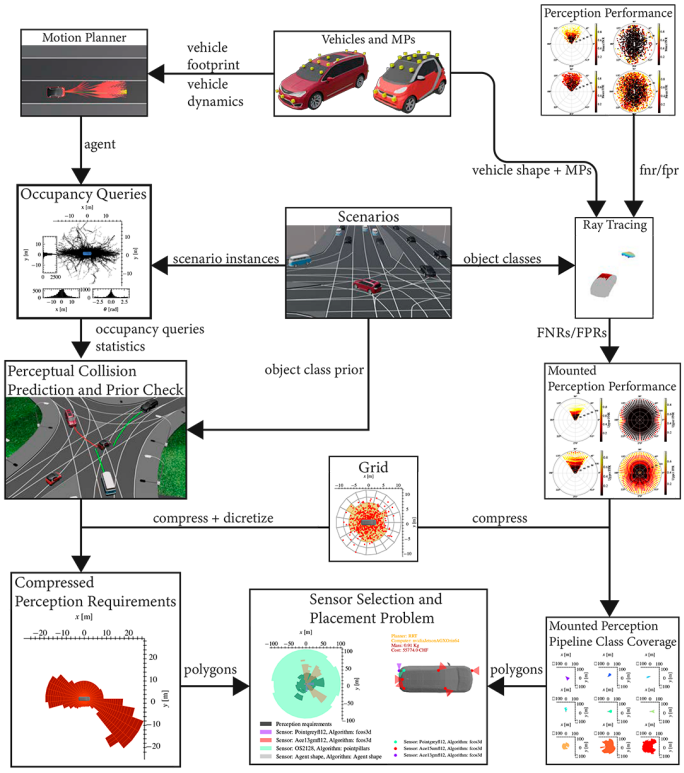


Fig. 11. Overview of the sensor selection and placement process: starting with a catalog of robot bodies, sensor positions, orientations, perception pipelines, and motion planners, alongside with scenarios. The workflow splits into agent activities (left) that transform task queries into perception requirements, 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.

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.

C. 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 (see Fig. 11). 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:

$$\{q_i \in \mathcal{Q}_i : \mathrm{up}(\mathrm{fnr}(q_i, \mathrm{appear}_i, \mathrm{pp}_j, \mathrm{env})) \leq \epsilon \\ \wedge\, \mathrm{up}(\mathrm{fpr}(q_i, \mathrm{appear}_i, \mathrm{pp}_j, \mathrm{env})) \leq \epsilon\},$$

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).

Example 29. Consider a lidar sensor positioned on the roof of a vehicle. Due to its placement, some lidar beams are blocked by the vehicle's roof, preventing the lidar from measuring objects in close proximity to the vehicle.

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 30 (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 31 (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

$$\begin{aligned} \mathrm{mppcc} \colon \mathbb{C} \times \mathbb{MPP} \times \mathbb{E} \times \mathbb{R}_{[0,1]} &\to \mathtt{POW}(\mathcal{Q}), \\ \langle C_i, \mathrm{mpp}_j, \mathrm{env}, \epsilon \rangle &\mapsto \Theta_i, \end{aligned} \tag{8}$$

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 $\mathrm{K_{class}}$ object class instances, $\mathrm{M_{env}}$ environments and $\mathrm{L_{mpp}}$ mounted perception pipelines are given as

$$\mathrm{MPPC} = \bigcup_{k=1}^{\mathrm{K_{class}}} \bigcup_{l=1}^{\mathrm{L_{mpp}}} \bigcup_{m=1}^{\mathrm{M_{env}}} \mathrm{mppcc}(\mathcal{C}_k, \mathrm{mpp}_l, \mathrm{env}_m, \epsilon).$$

Definition 32 (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 $\mathrm{K_{class}}$ unique number of object class instances and $\mathrm{M_{env}}$ number of environments. From the body, perception pipelines and mounting orientation, $\mathrm{L_{mpp}}$ 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 $\mathrm{MPP} \subseteq \{\mathrm{mpp}_i\}_{i\in\{1,\dots,\mathrm{L_{mpp}}\}}$ 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. (9), 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 $x_i = 0$:

$$\mathrm{emp}(\mathcal{Q}, x) = \begin{cases} \mathcal{Q} & \text{if } x = 1 \\ \emptyset & \text{if } x = 0 \end{cases}.$$

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.

$$\begin{aligned} \min \quad & \sum_{i=1}^{\mathrm{L_{mpp}}} \sum_{j=1}^{W} w_j c_j(\mathrm{mpp}_i) \cdot x_i \\ \text{s.t.} \quad & \mathrm{PR}(\mathcal{A}, \mathcal{T}, C_k, \mathrm{env}_m) \subseteq \\ & \bigcup_{l=1}^{\mathrm{L_{mpp}}} \mathrm{emp}(\mathrm{mppcc}(C_k, \mathrm{mpp}_l, \mathrm{env}_m, \epsilon), x_l) \\ & \forall k \in \{1, \dots, \mathrm{K_{class}}\}, m \in \{1, \dots, \mathrm{M_{env}}\}, \\ & F \cdot x \leq \begin{bmatrix} 1 & \dots & 1 \end{bmatrix}^{\mathrm{T}} \quad , \\ & x_i \leq 1 \quad \forall i \in \{1, \dots, \mathrm{L_{mpp}}\}, \\ & x_i \in \mathbb{N}_0 \quad \forall i \in \{1, \dots, \mathrm{L_{mpp}}\}, \\ & \sum_{j=1}^{W} w_j = 1, \ w_j \geq 0, \ j = 1, \dots, W. \end{aligned} \tag{9}$$

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*:

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

Finally, we can formulate the **Sensor Selection and Placement** MDPI, visualized in Fig. 15. It implements the sensor selection and placement problem from Def. 32 and it is the composition of the following MDPIs. The **Coverage** MDPI, illustrated in Fig. 12, 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, which is demonstrated in Lemma 33.

Lemma 33. The perception requirements PR are monotone with perception coverage, as shown in Fig. 12.

Proof. Consider the first constraint in Eq. (9), 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. ■

Fig. 12. The Coverage MDPI which provides perception requirements PR and requires perception coverage as a set of mppcc.

The **Mounted Perception Pipelines** MDPI in Fig. 13 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 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 performance considers the opposite order of fnr and fpr upper limits, where a pipeline 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. 7, 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.

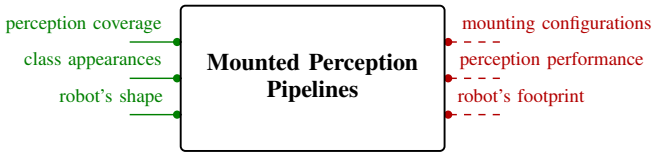


Fig. 13. The Mounted Perception Pipelines MDPI which provides the perception coverage, the set of all class appearances appear in the task and the robot's shape SH as functionalities. The required resources are the set of mounting configurations in SE(3), the perception performance and the robot's footprint $\mathrm{sh}_{\mathcal{R}}$.

In Fig. 14, 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.

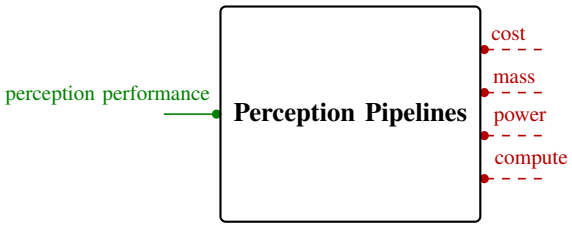


Fig. 14. The Perception Pipelines MDPI which provides the perception performance and requires cost in CHF, mass in kg, power in W and compute.

Fig. 15. The Sensor Selection and Placement MDPI which is the composition of the Coverage, Mounted Perception Pipelines and Perception Pipelines MDPIs.

D. Solving the Sensor Selection and Placement Set Cover Problem

The nature of Def. 32 closely resembles the weighted set cover problem [58], since it also tries to cover a given set by a collection of subsets while minimizing a cost function.

Definition 34 (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. 32, we choose the ILP relaxation of the set cover problem, as outlined in Eq. (10). 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 [58].

$$\begin{aligned} \min \quad & \sum_{i=1}^{K} c(S_i) \cdot x_i \\ \text{s.t.} \quad & \sum_{i:e\in S_i} x_i \geq 1 \quad \forall e \in U, \\ & x_i \leq 1 \quad \forall i \in \{1, \ldots, K\}, \\ & x_i \in \mathbb{N}_0 \quad \forall i \in \{1, \ldots, K\}. \end{aligned} \tag{10}$$

To formulate the Def. 32 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 [59], 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 $\{[-\pi, -\pi + \Delta\theta), [-\pi + \Delta\theta, -\pi + 2 \cdot \Delta\theta) \ldots [\pi - \Delta\theta, \pi)\}$. 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*.

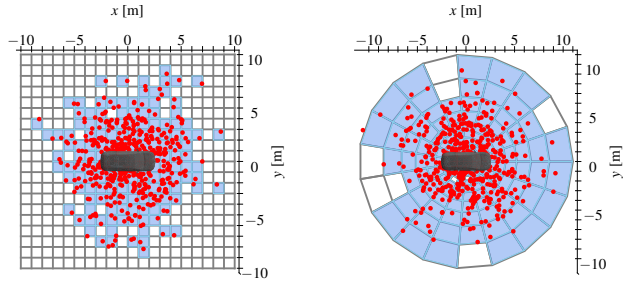
Definition 35 (Compress)**.** $\mathrm{compress}$ is a mapping that generates a set of multi-polygons μ from a set of class configurations $\mathcal{Q}_i$ and T number of class configurations θ-intervals.

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

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

Applying $\mathrm{compress}$ 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 $\mathrm{compress}$ is applied for each environment in PR, nested sets are obtained for each environment, object class, and θ interval.

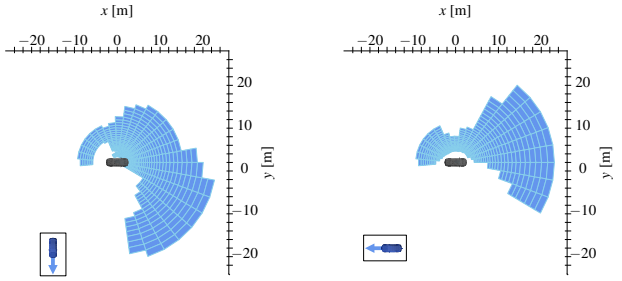
Discretization: To formulate the weighted set cover problem with the obtained polygons, we need to discretize $\overline{\mathrm{PR}}(\mathcal{A}, \mathcal{T})$. A straightforward approach is to create a grid with cells, which can be made uniform as shown in Fig. 16a, e.g., 1 by 1 meters in size. We use a polar grid with logarithmic scaling for radial distance as illustrated in Fig. 16b, 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



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

Fig. 16. 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.

urban environment, for a car class object for two different orientations is shown in Fig. 17.



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

Fig. 17. Example of discretized and compressed perception requirements of a car class (blue) for different orientations relative to the ego vehicle (grey car).

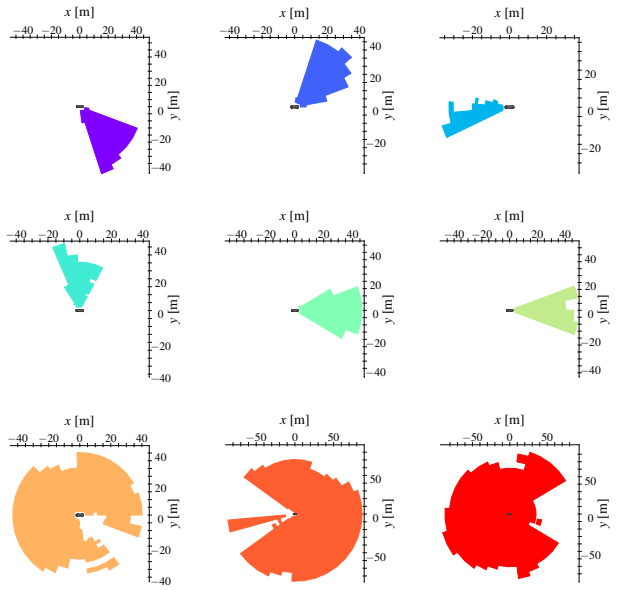


Fig. 18. Examples of compressed mounted perception pipeline class coverage mppcc corresponding to the setting in Fig. 17 with θ-interval $[-100°, -90°)$. Each plot corresponds to a unique mounted perception pipeline.

In Fig. 18, examples of compressed mppcc are depicted for the class and robot specified in Fig. 17 with θ-interval $[-100°, -90°)$. These polygons aim to cover the upper polygon shown in Fig. 17. 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. 32 as an ILP using Eq. (10). 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. (10) 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 [60], [61], a generalized form of the one-dimensional Van der Corput sequence [40], [62], 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 [63], [64].

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 $N \times \mathrm{L}_{\mathrm{mpp}}$, where N represents the number of polygons in $\widehat{\mathrm{PR}}(\mathcal{A}, \mathcal{T})$ and $\mathrm{L}_{\mathrm{mpp}}$ 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 $D \times \mathrm{L}_{\mathrm{mpp}}$, 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. (11).

$$
\begin{aligned}
\min \quad & \sum_{i=1}^{L} \sum_{j=1}^{W} w_j c_j(\mathrm{mpp}_i) \cdot x_i \\
\text{s.t.} \quad & A \cdot x \geq \begin{bmatrix} 1 & \dots & 1 \end{bmatrix}^{\mathrm{T}} , \\
& F \cdot x \leq \begin{bmatrix} 1 & \dots & 1 \end{bmatrix}^{\mathrm{T}} , \\
& x_i \leq 1 \quad \forall i \in \{1, \dots, L\}, \\
& x_i \in \mathbb{N}_0 \quad \forall i \in \{1, \dots, L\}, \\
& \sum_{j=1}^{W} w_j = 1, \; w_j \geq 0, \; j = 1, \dots, W.
\end{aligned}
\tag{11}
$$

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

Variable	Option	Source
Vehicle bodies	Smart Fortwo, Chrysler Pacifica, Mercedes-Benz C63	[65]
Lidars	Velodyne: Alpha Prime, HDL 64, HDL 32; OS2: 128, 64	[66], [67]
Cameras	Basler: acA1600-gm, acA1500-um, acA7-gm; FLIR: Point Grey	[68], [69]
Object Detection Models	FCOS3D, Pointpillars	[52], [53], [55]
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*	[70], [71]
Computer	Jetson Nano, Orin Nano, Xavier NX, Orin NX, AGX Orin 64GB, AGX Orin 32GB, AGX Xavier 32GB	[72]

Fig. 19. Exemplary mounting positions for two different vehicles.

V. DESIGN OF EXPERIMENTS AND RESULTS

In this section, we report a case study on designing an AV for an urban driving task. We outline the experimental design in Sec. V-A, present the results in Sec. V-B, and conclude with a discussion of the findings in Sec. V-C.

A. Design of experiments

Catalogs: The components available for design are reported in the catalog in Tab. II. The 3D meshes of the car bodies are sourced from TurboSquid [73]. Real sensor measurements from the nuScenes open-source dataset [54], along with state-of-the-art 3D object detection algorithms from the MMDetection3D library [55], are used to determine the FNRs and the FPRs for different object classes. The mounting position options are visualized in Fig. 19. We utilize motion planners from the OMPL [70] and CommonRoad [71] libraries, including RRT, RRT*, and a lattice planner enhanced with motion primitives and an A* search algorithm. Specifically, for the RRT* planner from the OMPL library, which is classified as a "geometric" planner, we employ Dubins paths [39], [74] to connect sampled configurations considering the system's geometric and kinematic constraints. This approach enables the computation of paths that can be tracked by low level controllers as depicted in Fig. 2a. In contrast, the RRT planner corresponds to "control-based" implementations in the OMPL library which directly computes trajectories and control inputs, tailored for systems subject to differential constraints and incorporating a steering function. 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.

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.

Task: The urban driving task contains 205 driving scenarios from the CommonRoad library [75], featuring five different vehicle classes. Each vehicle's configuration is defined by $q \in \mathrm{SE}(3)$ and the vehicle dynamics are based on the bicycle model, with the control space for cars specified as $\mathcal{U}_{\mathrm{car}} = [a_{\min}, a_{\max}] \times [\delta_{\min}, \delta_{\max}]$. The car prior configuration, $\mathcal{P}_{\mathrm{car}}$, accounts for all possible car positions on the road, aligning with the driving direction. 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$50\,\mathrm{km/h}$, under dry and rainy weather conditions during daylight and night. We performed 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 and rear. We solved the ILP for the sensor selection and placement problem in Eq. (11) using the Gurobi [76] solver. Our ILP comprised 667 decision variables representing the mounted perception pipelines, with around 250,000 constraints. We optimized with 4 different costs, which are the price, mass and power consumption of the sensor and the computing in flops of the object detection algorithms. Solving the problem took 75 s on a 2.3 GHz Intel Core i7 processor with 16 GB of RAM. Through the sampling sequence, around 3000 weight sets were generated to populate the Pareto front, resulting in 3000 individual optimization problems solved for each robot body, agent, and task combination.

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 Figs. 20, 21 and 22. The figures show 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 figures compare price (CHF) on the x-axis against power consumption (W) in Fig. 20, mass (kg) in Fig. 21, and computation (Gflops) in Fig. 22 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.

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 Figs. 23, 24 and 25, where we again show the Pareto fronts as well as the corresponding implementations for the different resources. Figs. 26, 27 and 28 demonstrate how restricting car configurations prior within identical task scenarios leads to lower resource requirements, where the Pareto fronts, along with implementations are illustrated.

In Fig. 29 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

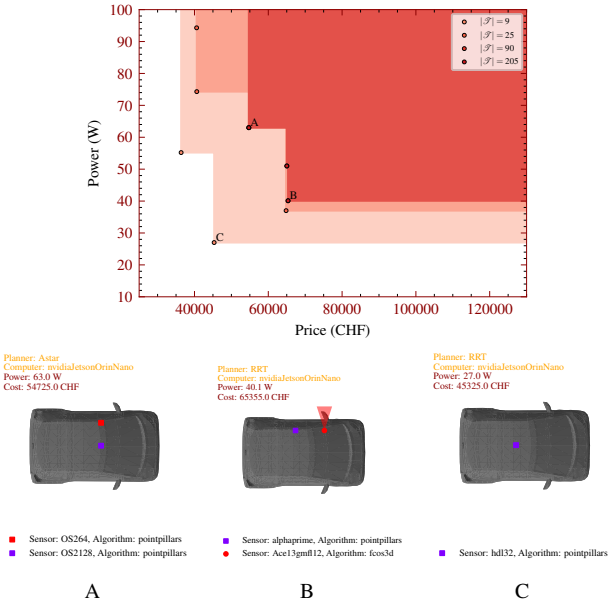


Fig. 20. Pareto front of price and power across tasks, where tasks with more scenarios demand more resources and encompass those with fewer scenarios. Implementations for point A, B, and C are visualized vertically. B and C indicate the least power usage for the most and least complex tasks, respectively, while A shows the minimum price for the most complex task.

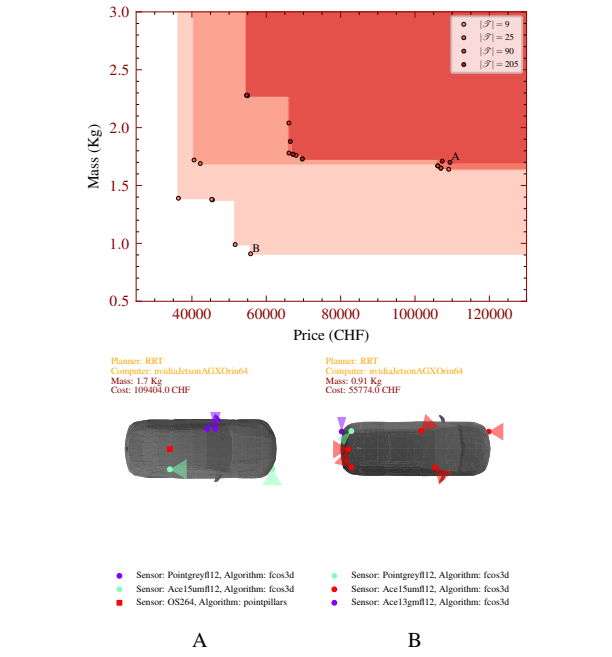


Fig. 21. Pareto front of price and mass across tasks, where tasks with more scenarios demand more resources and encompass those with fewer scenarios. Implementations for points A and B are visualized vertically. A and B indicate the lowest mass for the most and least complex tasks, respectively.

the 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. 30, 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,

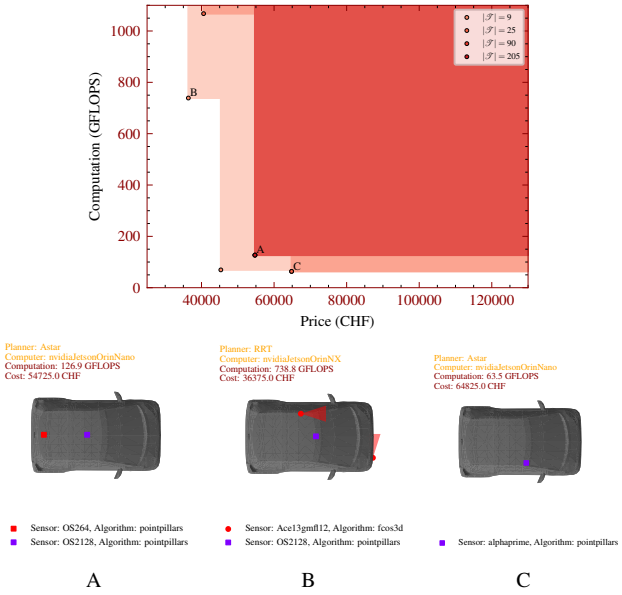


Fig. 22. Pareto front of price and computation across tasks, where more scenarios demand more resources . Implementations plots for point A, B, and C are visualized vertically. A and C indicate the least computation usage for the most and least complex tasks, respectively, while B shows the minimum price for the least complex task.

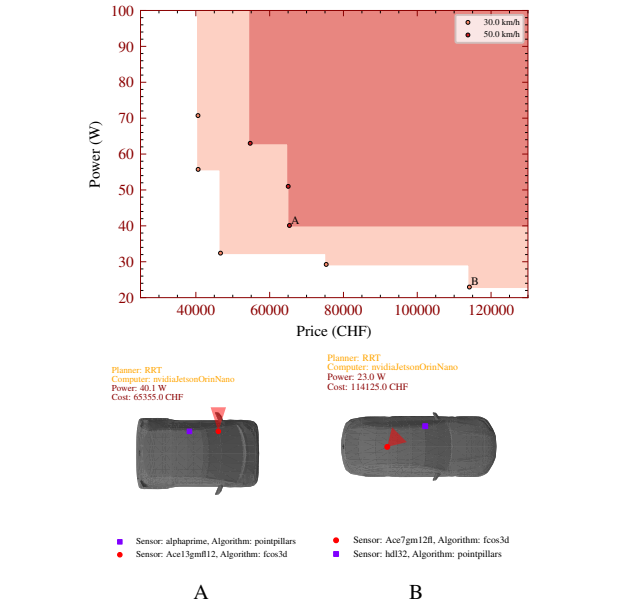


Fig. 23. Pareto front of price and power usage across task velocities, where higher nominal velocities for the same set of scenarios require more resources. Implementations for points A and B are visualized vertically. A and B indicate lowest power usage for $50\,\mathrm{km/h}$ and $30\,\mathrm{km/h}$ nominal velocities, respectively.

as expected from Fig. 2, and to highlight the impact of the planning strategy on the sensor selection and placement process.

In Figs. 22, 25 and 28, 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. 21, 24 and 27, where there is a notable preference for cameras, predominantly coupled with the most powerful computing unit, the NVIDIA Jetson AGX

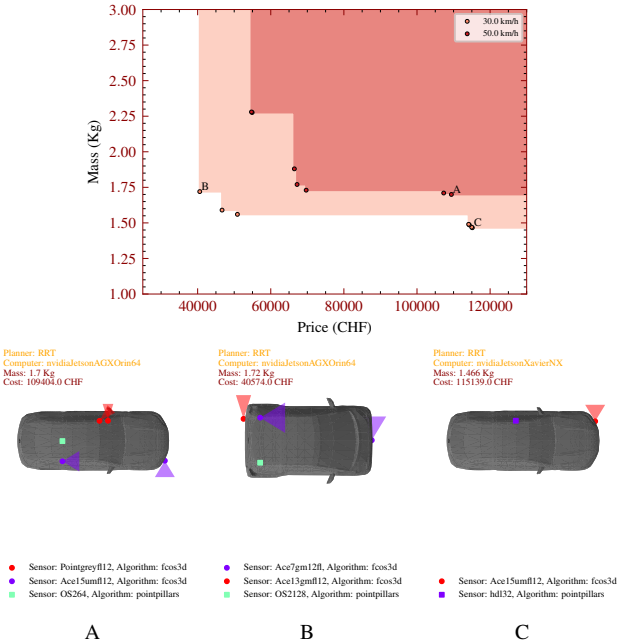


Fig. 24. Pareto front of price and mass across task velocities, where higher nominal velocities for the same set of scenarios require more resourses. Implementations for points A, B and C are visualized vertically. A and C indicate lowest mass for 50 $\mathrm{km/h}$ and 30 kmh nominal velocities, respectively. B indicates lowest price for 30 $\mathrm{km/h}$ nominal speed.

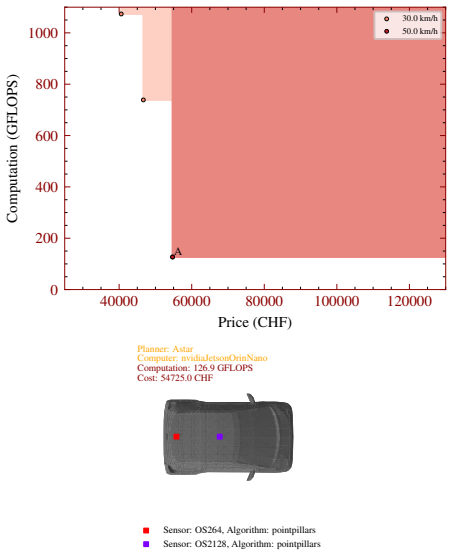


Fig. 25. Pareto front of price and computation across task velocities, where higher velocities for the same set of scenarios require more resources. Implementations for marked point A are visualized vertically. A indicate lowest computation for 50 $\mathrm{km/h}$ and 30 kmh.

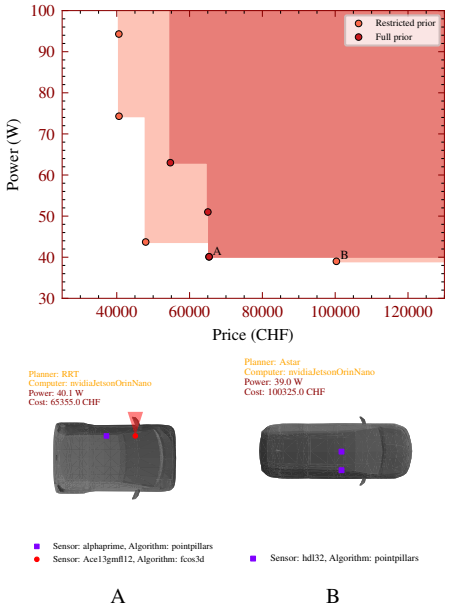


Fig. 26. Pareto front of price and power usage across priors, where priors with more class configurations require more resources. Implementations for points A and B are visualized vertically. A and B indicate the lowest power usage for the least and most restricted prior, respectively.

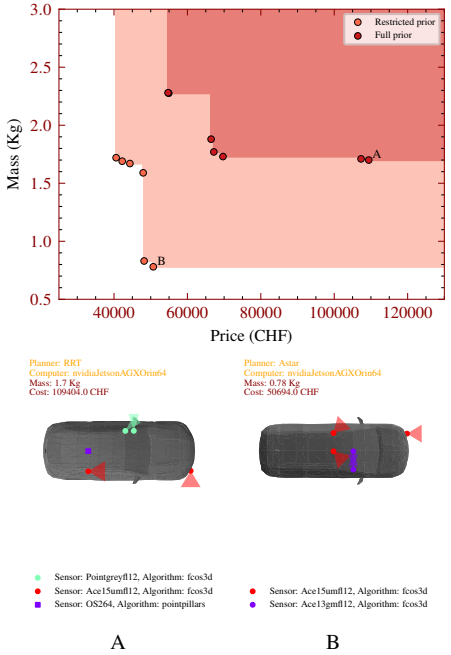


Fig. 27. Pareto front of price and mass across priors, where priors with more class configurations require more resources. Implementations for points A and B are visualized vertically. A and B indicate the lowest mass for the least and most restricted prior, respectively.

Orin 64. In Figs. 20, 23 and 26 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.

Moreover, we present implementations tailored for the most cost-effective AV design in Figs. 20, 22, 24 and 28. Every implementation features at least one lidar sensor. Except for the cases highlighted in Figs. 20 and 28, 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.

Throughout the minimal resource 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 Figs. 31, 32 and 33 for power, mass,

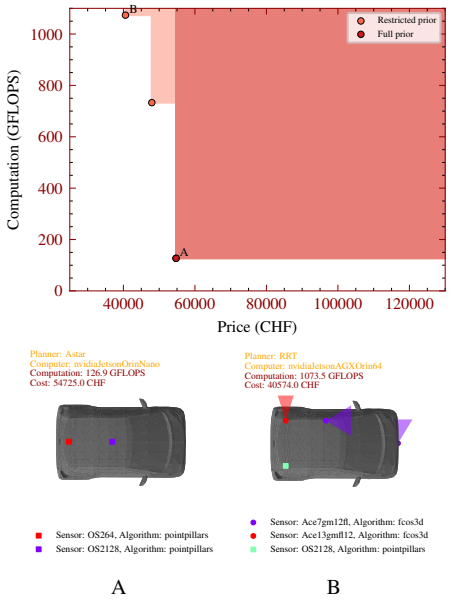


Fig. 28. Pareto front of price and computation across priors, where priors with more class configurations require more resources. Implementations for points A and B are visualized vertically. A indicates the lowest computation for both priors (same implementation) and B indicates lowest price for the most restricted prior.

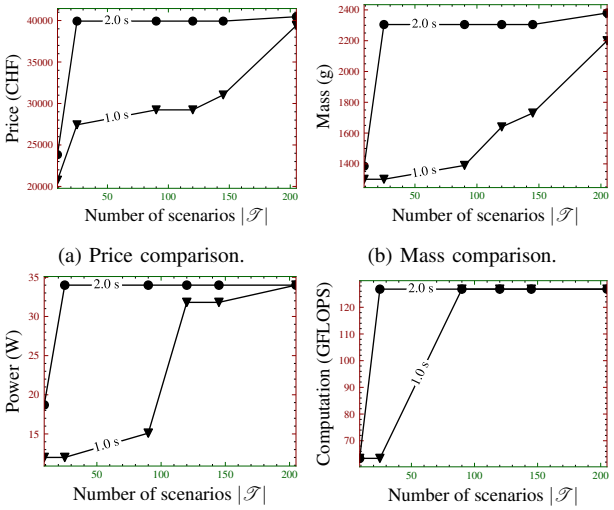


Fig. 29. Higher planning horizons for the same planner and vehicle body require more resources for different tasks. Here we show the lattice planner with A* search and a hatchback vehicle body.

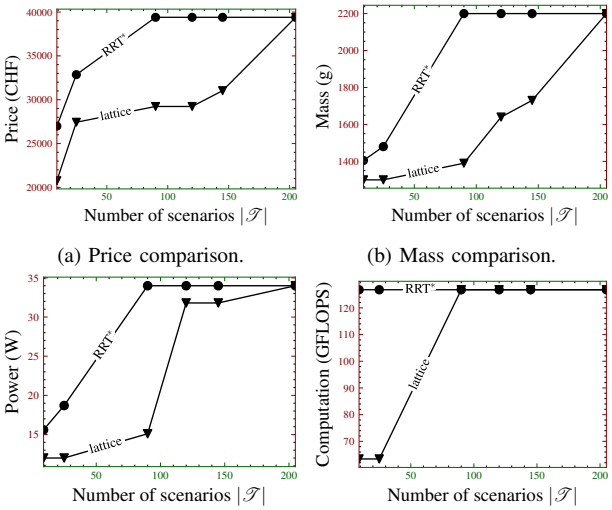


Fig. 30. Resource comparison between RRT* planner and lattice planner with A* search for the same vehicle body (hatchback) and tasks.

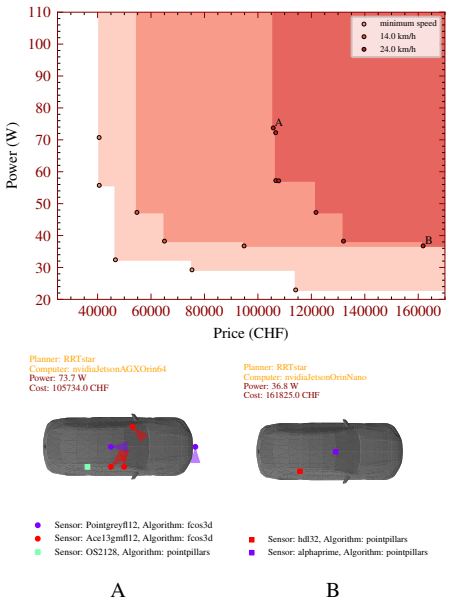


Fig. 31. Pareto front of price and power consumption across different average speeds, where planners providing higher average speed across all scenarios ($30\,\mathrm{km/h}$ nominal speed) demand more resources. Implementations plots for points A and B are visualized vertically. A and B indicate the lowest price and lowest power for the highest average speed, respectively.

and computation impacts, it becomes evident that the resource demands increase 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.

C. 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

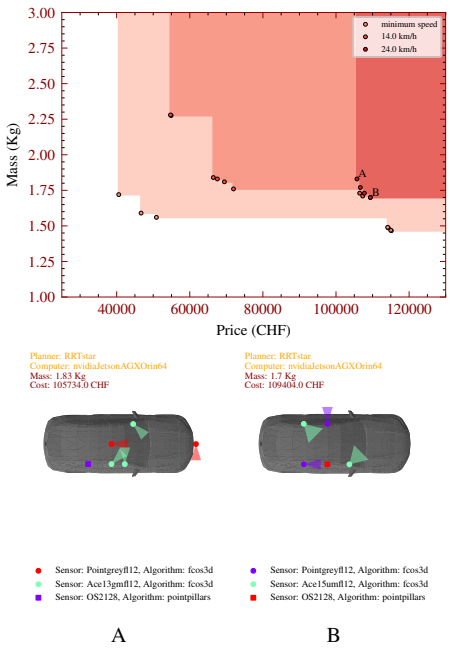


Fig. 32. Pareto front of price and mass across different average speeds, where planners providing higher average speed across all scenarios ($30\,\mathrm{km/h}$ nominal speed) demand more resources. Implementations for points A and B are visualized vertically. A and B indicate the lowest price and mass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

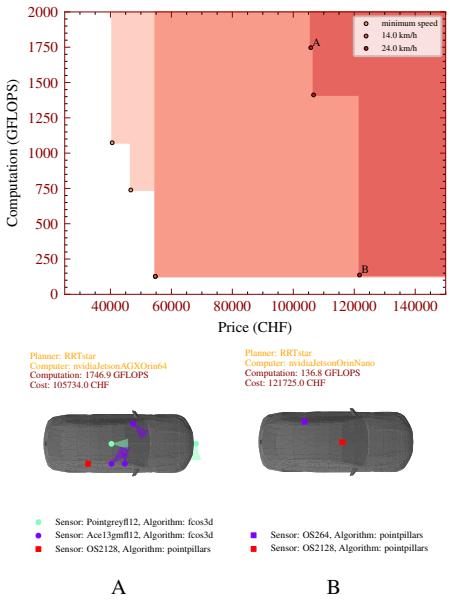


Fig. 33. Pareto front of price and computation across different average speeds, where planners providing higher average speed across all scenarios ($30\,\mathrm{km/h}$ nominal speed) demand more resources. Implementations for points A and B are visualized vertically. A and B indicate the lowest price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

distributions result from either extended planning horizons, as illustrated in Fig. 29, or the inherent strategy of the motion planner, as illustrated in Fig. 2 and Fig. 30. 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 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.

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. Moreover, motion planners that generate broader distributions of occupancy queries or require longer planning horizons lead to increased task performance and perception requirements, necessitating more advanced and costly sensors and perception algorithms for the robot's design. The findings highlight that the preference for specific sensors is influenced by the prioritization of resources. For designs prioritizing lower costs and weight, camera sensors are favored. Conversely, when minimizing power consumption and computing resources, lidar sensors are the preferred choice. Overall, lidar sensors exhibit superior perception performance and coverage, proving to be essential for handling complex tasks. 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] S. A. Seshia, S. Hu, W. Li, and Q. Zhu, "Design automation of cyber-physical systems: Challenges, advances, and opportunities," *IEEE*

Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 36, no. 9, pp. 1421–1434, 2017.

[2] Q. Zhu and A. Sangiovanni-Vincentelli, "Codesign methodologies and tools for cyber–physical systems," *Proceedings of the IEEE*, vol. 106, no. 9, pp. 1484–1500, 2018.

[3] E. A. Lee, "Cyber physical systems: Design challenges," in *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*, 2008, pp. 363–369.

[4] A. Q. Nilles, D. A. Shell, and J. M. O'Kane, "Robot design: Formalisms, representations, and the role of the designer," *CoRR*, vol. abs/1806.05157, 2018.

[5] S. Joshi and S. Boyd, "Sensor selection via convex optimization," *IEEE Transactions on Signal Processing*, vol. 57, no. 2, pp. 451–462, 2009.

[6] V. Gupta, T. H. Chung, B. Hassibi, and R. M. Murray, "On a stochastic sensor selection algorithm with applications in sensor scheduling and sensor coverage," *Automatica*, vol. 42, no. 2, pp. 251–260, 2006.

[7] M. Shamaiah, S. Banerjee, and H. Vikalo, "Greedy sensor selection: Leveraging submodularity," in *49th IEEE Conference on Decision and Control (CDC)*, 2010, pp. 2572–2577.

[8] D. Golovin, M. Faulkner, and A. Krause, "Online distributed sensor selection," in *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks*, ser. IPSN '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 220–231.

[9] G. Hovland and B. McCarragher, "Dynamic sensor selection for robotic systems," in *Proceedings of International Conference on Robotics and Automation*, vol. 1, 1997, pp. 272–277 vol.1.

[10] M. Erdmann, "Understanding action and sensing by designing action-based sensors," *The International Journal of Robotics Research*, vol. 14, no. 5, pp. 483–509, 1995.

[11] 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*, vol. 2, 1995, pp. 555–560 vol.2.

[12] V. Tzoumas, L. Carlone, G. J. Pappas, and A. Jadbabaie, "LQG control and sensing co-design," *IEEE Transactions on Automatic Control*, vol. 66, no. 4, pp. 1468–1483, 2021.

[13] A. Collin, A. Siddiqi, Y. Imanishi, Y. Matta, T. Tanimichi, and O. de Weck, "A multiobjective systems architecture model for sensor selection in autonomous vehicle navigation," in *Complex Systems Design & Management*, G. A. Boy, A. Guegan, D. Krob, and V. Vion, Eds. Cham: Springer International Publishing, 2020, pp. 141–152.

[14] A. Collin, A. Siddiqi, Y. Imanishi, E. Rebentisch, T. Tanimichi, and O. L. de Weck, "Autonomous driving systems hardware and software architecture exploration: optimizing latency and cost under safety constraints," *Systems Engineering*, vol. 23, no. 3, pp. 327–337, 2020.

[15] J. Dey, W. Taylor, and S. Pasricha, "VESPA: A framework for optimizing heterogeneous sensor placement and orientation for autonomous vehicles," *IEEE Consumer Electronics Magazine*, vol. 10, no. 2, pp. 16–26, 2021.

[16] J. Dey and S. Pasricha, "Machine learning based perception architecture design for semi-autonomous vehicles," in *Machine Learning and Optimization Techniques for Automotive Cyber-Physical Systems*, V. K. Kukkala and S. Pasricha, Eds. Cham: Springer International Publishing, 2023, pp. 625–646.

[17] A. Spielberg, A. Amini, L. Chin, W. Matusik, and D. Rus, "Co-learning of task and sensor placement for soft robotics," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 1208–1215, 2021.

[18] S. Ghasemlou, J. M. O'Kane, and D. A. Shell, "Delineating boundaries of feasibility between robot designs," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 422–429.

[19] F. Z. Saberifar, S. Ghasemlou, D. A. Shell, and J. M. O'Kane, "Toward a language-theoretic foundation for planning and filtering," *The International Journal of Robotics Research*, vol. 38, no. 2-3, pp. 236–259, 2019.

[20] Y. Zhang and D. 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)*, 2020, pp. 8469–8475.

[21] L. Nardi, D. Koeplinger, and K. Olukotun, "Practical design space exploration," in *2019 IEEE 27th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2019, pp. 347–358.

[22] J. M. O'Kane and S. M. LaValle, "Comparing the power of robots," *The International Journal of Robotics Research*, vol. 27, no. 1, pp. 5–23, 2008.

[23] S. M. LaValle, "Sensing and filtering: A fresh perspective based on preimages and information spaces," *Foundations and Trends® in Robotics*, vol. 1, no. 4, pp. 253–372, 2012.

[24] A. Censi, E. Mueller, E. Frazzoli, and S. 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)*, 2015, pp. 3319–3326.

[25] L. Nardi, B. Bodin, M. Z. Zia, J. Mawer, A. Nisbet, P. H. J. Kelly, A. J. Davison, M. Luján, M. F. P. O'Boyle, G. Riley, N. Topham, and S. Furber, "Introducing SLAMBench, a performance and accuracy benchmarking methodology for SLAM," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 5783–5790.

[26] M. Lahijanian, M. Svorenova, A. A. Morye, B. Yeomans, D. Rao, I. Posner, P. Newman, H. Kress-Gazit, and M. Kwiatkowska, "Resource-performance tradeoff analysis for mobile robots," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1840–1847, 2018.

[27] S. Seok, A. Wang, M. Y. Chuah, D. J. Hyun, J. Lee, D. M. Otten, J. H. Lang, and S. Kim, "Design principles for energy-efficient legged locomotion and implementation on the MIT Cheetah robot," *IEEE/ASME Transactions on Mechatronics*, vol. 20, no. 3, pp. 1117–1129, 2015.

[28] F. Z. Saberifar, D. A. Shell, and J. 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)*, 2022, pp. 135–141.

[29] A. M. Mehta, J. DelPreto, K. W. Wong, S. Hamill, H. Kress-Gazit, and D. Rus, *Robot Creation from Functional Specifications*. Cham: Springer International Publishing, 2018, pp. 631–648.

[30] S. Ha, S. Coros, A. Alspach, J. M. Bern, J. Kim, and K. Yamane, "Computational design of robotic devices from high-level motion specifications," *IEEE Transactions on Robotics*, vol. 34, no. 5, pp. 1240–1251, 2018.

[31] D. A. Shell, J. M. O'Kane, and F. Z. Saberifar, "On the design of minimal robots that can solve planning problems," *IEEE Transactions on Automation Science and Engineering*, vol. 18, no. 3, pp. 876–887, 2021.

[32] G. Zardini, "Co-design of complex systems: From autonomy to future mobility systems," Ph.D. dissertation, ETH Zurich, 2023.

[33] A. Censi, "A mathematical theory of co-design," *CoRR*, vol. abs/1512.08055, 2015.

[34] A. Censi, J. Lorand, and G. Zardini, *Applied Compositional Thinking for Engineering*, 2024, work-in-progress book (currently discussing with publishers).

[35] G. Zardini, A. Censi, and E. Frazzoli, "Co-Design of Autonomous Systems: From Hardware Selection to Control Synthesis," *2021 European Control Conference, ECC 2021*, pp. 682–689, 2021.

[36] G. Zardini, D. Milojevic, A. Censi, and E. Frazzoli, "Co-design of embodied intelligence: A structured approach," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021, pp. 7536–7543.

[37] G. Zardini, Z. Suter, A. Censi, and E. Frazzoli, "Task-driven modular co-design of vehicle control systems," in *2022 IEEE 61st Conference on Decision and Control (CDC)*, 2022, pp. 2196–2203.

[38] G. Zardini, N. Lanzetti, A. Censi, E. Frazzoli, and M. Pavone, "Co-design to enable user-friendly tools to assess the impact of future mobility solutions," *IEEE Transactions on Network Science and Engineering*, vol. 10, no. 2, pp. 827–844, 2023.

[39] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," *IEEE Transactions on Intelligent Vehicles*, vol. 1, no. 1, pp. 33–55, 2016.

[40] S. M. LaValle, *Planning Algorithms*. Cambridge university press, 11 2006.

[41] B. Chazelle, "Approximation and decomposition of shapes," *Algorithmic and Geometric Aspects of Robotics/ed. JT Schwartz, CK Yap*, pp. 145–185, 1985.

[42] O. Takahashi and R. Schilling, "Motion planning in a plane using generalized voronoi diagrams," *IEEE Transactions on Robotics and Automation*, vol. 5, no. 2, pp. 143–150, 1989.

[43] J. Backer and D. Kirkpatrick, "Finding curvature-constrained paths that avoid polygonal obstacles," in *Proceedings of the Twenty-Third Annual Symposium on Computational Geometry*, ser. SCG '07. New York, NY, USA: Association for Computing Machinery, 2007, p. 66–73.

[44] L. Claussmann, M. Revilloud, D. Gruyer, and S. Glaser, "A review of motion planning for highway autonomous driving," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 5, pp. 1826–1848, 2020.

[45] P. Falcone, F. Borrelli, J. Asgari, H. E. Tseng, and D. Hrovat, "Predictive active steering control for autonomous vehicle systems," *IEEE Transactions on Control Systems Technology*, vol. 15, no. 3, pp. 566–580, 2007.

[46] 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*, 2007, pp. 2980–2985.

[47] E. Kim, J. Kim, and M. Sunwoo, "Model predictive control strategy for smooth path tracking of autonomous vehicles with steering actuator dynamics," *International Journal of Automotive Technology*, vol. 15, pp. 1155–1164, 2014.

[48] G. V. Raffo, G. K. Gomes, J. E. Normey-Rico, C. R. Kelber, and L. B. Becker, "A predictive controller for autonomous vehicle path tracking,"

IEEE Transactions on Intelligent Transportation Systems, vol. 10, no. 1, pp. 92–102, 2009.
[49] Y. Yoon, J. Shin, H. J. Kim, Y. Park, and S. Sastry, "Model-predictive active steering and obstacle avoidance for autonomous ground vehicles," *Control Engineering Practice*, vol. 17, no. 7, pp. 741–750, 2009.
[50] A. Liniger, A. Domahidi, and M. Morari, "Optimization-based autonomous racing of 1: 43 scale rc cars," *Optimal Control Applications and Methods*, vol. 36, no. 5, pp. 628–647, 2015.
[51] L. Janson, B. Ichter, and M. Pavone, "Deterministic sampling-based motion planning: Optimality, complexity, and performance," *The International Journal of Robotics Research*, vol. 37, no. 1, pp. 46–61, 2018.
[52] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. 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.
[53] T. Wang, X. Zhu, J. Pang, and D. Lin, "FCOS3D: Fully Convolutional One-Stage Monocular 3D Object Detection," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) Workshops*, October 2021, pp. 913–922.
[54] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom, "nuscenes: A multimodal dataset for autonomous driving," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.
[55] M. Contributors, "MMDetection3D: OpenMMLab next-generation platform for general 3D object detection," https://github.com/open-mmlab/mmdetection3d, 2020.
[56] A. Censi, "Efficient neuromorphic optomotor heading regulation," *Proceedings of the American Control Conference*, vol. 2015-July, pp. 3854–3861, 2015.
[57] B. A. Davey and H. A. Priestley, *Introduction to Lattices and Order*, 2nd ed. Cambridge University Press, 4 2002.
[58] V. V. Vazirani, *Approximation algorithms*, 1st ed. Springer Berlin, Heidelberg, 2001, vol. 1.
[59] J. C. Culberson and R. A. Reckhow, "Covering polygons is hard," in *Proceedings of the 29th Annual Symposium on Foundations of Computer Science*, ser. SFCS '88. USA: IEEE Computer Society, 1988, p. 601–611.
[60] J. H. Halton, "On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals," *Numerische Mathematik*, vol. 2, pp. 84–90, 1960.
[61] A. B. Owen, "A randomized halton algorithm in R," 2017.
[62] V. der Coput Johannes, "Verteilungsfunktionen i & ii," *Nederl. Akad. Wetensch. Proc.*, vol. 38, pp. 1058–1066, 1935.
[63] I. P. Stanimirovic, M. L. Zlatanovic, and M. D. Petkovic, "On the linear weighted sum method for multi-objective optimization," *Facta Acta Univ*, vol. 26, no. 4, pp. 49–63, 2011.
[64] R. T. Marler and J. S. Arora, "The weighted sum method for multi-objective optimization: new insights," *Structural and multidisciplinary optimization*, vol. 41, pp. 853–862, 2010.
[65] Cars. (2024) Cars. Available online: https://www.cars.com.
[66] Velodyne. (2024) Velodyne lidars. Available online: https://velodynelidar.com.
[67] Ouster. (2024) Ouster lidars. Available online: https://ouster.com.
[68] Basler. (2024) Basler cameras. Available online: https://www.baslerweb.com.
[69] Flir. (2024) Flir cameras. Available online: https://www.flir.com.
[70] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, December 2012, https://ompl.kavrakilab.org.
[71] M. Althoff, M. Koschi, and S. Manzinger, "Commonroad: Composable benchmarks for motion planning on roads," in *Proc. of the IEEE Intelligent Vehicles Symposium*, 2017.
[72] NVIDIA. (2024) Nvidia products. Available online: https://www.nvidia.com.
[73] TurboSquid. (2024) Turbosquid 3d models. Available online: https://www.turbosquid.com/3d-models/3d-40-cars-1703688.
[74] L. E. Dubins, "On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents," *American Journal of Mathematics*, vol. 79, no. 3, pp. 497–516, 1957.
[75] M. Maierhofer, M. Klischat, and M. 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*, 2021, pp. 3176–3182.
[76] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2023.

Dejan Milojevic is a Ph.D. candidate in Robotics, jointly at the Institute for Dynamic Systems and Control, ETH Zurich, and the Chemical Energy Carriers and Vehicle Systems Laboratory, Empa. He holds a BSc. and MSc. in Mechanical Engineering from ETH Zurich. He has conducted research at Stanford University under Marco Pavone and has worked as a software engineer for Vay in Berlin, Germany. His research interests include co-design, sensor selection, perception, and decision-making in robotics.

Gioele Zardini is an incoming Assistant Professor in the Department of Civil and Environmental Engineering at MIT in Fall 2024, where he is a PI at the Laboratory for Information and Decision Systems (LIDS), and affiliate faculty at the Institute for Data, Systems, and Society (IDSS). Currently, he is a Postdoctoral Scholar in the Department of Aeronautics and Astronautics at Stanford University. He received his BSc., MSc., and Ph.D. in Mechanical Engineering with a focus on Robotics, Systems, and Control from ETH Zurich . He spent time in Singapore as a researcher at nuTonomy (then Aptiv, now Motional), at Stanford University (working with Marco Pavone), and at MIT (in 2020 working with David Spivak, and in 2023 with Munther Dahleh).

Miriam Elser leads the Vehicle Systems Group at the Chemical Energy Carriers and Vehicle Systems Laboratory at Empa. She holds a Master degree in Physics from the University of Milan and completed her Ph.D. in Atmospheric Environmental Science at ETH Zurich and the Paul Scherrer Institute in Switzerland. Miriam's current research focuses on future road mobility, with an emphasis on developing decarbonization strategies for road vehicles, validating and integrating new technologies such as autonomous driving.

Andrea Censi is the deputy director of the Dynamic Systems and Control chair at ETH Zurich, director of the Duckietown Foundation, and founder of Zupermind. He obtained a M.Eng. degree in Control and Robotics from the University of Rome, "Sapienza", and a Ph.D. from California Institute of Technology. He has been a research scientist at the Massachusetts Institute of Technology, and the Director of Research at Aptiv Autonomous Mobility (now Motional). He has been the recipient of NSF and AFRL awards.

Emilio Frazzoli is a Professor of Dynamic Systems and Control at ETH Zurich. Until March 2021, he was Chief Scientist of Motional, the latest embodiment of nuTonomy, the startup he founded with Karl Iagnemma in 2013. He received the Laurea degree in aerospace engineering from the University of Rome, "Sapienza", in 1994, and the Ph.D. degree in Aeronautics and Astronautics from MIT in 2001. Before joining ETH Zurich in 2016, he held faculty positions at UIUC, UCLA, and MIT. His current research interests focus primarily on autonomous vehicles, mobile robotics, and transportation systems. He was the recipient of a NSF CAREER award in 2002, the IEEE George S. Axelby award in 2015, the IEEE Kiyo Tomiyasu award in 2017, the RSS Test of Time award in 2022, and is an IEEE Fellow since 2019.