MATH 6335 — Machine Learning and Control Theory

Reinforcement Learning for Robotics and Autonomous Mobility

Foundations, Progress, and Real-World Deployment Challenges

Itay Kadosh and Hyeonduk Sim

Instructor: Dr. Alain Bensoussan

School of Natural Sciences and Mathematics
The University of Texas at Dallas

Date: December 6, 2025



Abstract

Robotics and autonomous mobility increasingly rely on learning-based methods to make decisions in uncertain, dynamic, and high-dimensional environments. Classical control and planning approaches—such as PID, LQR, trajectory optimization, and sampling-based motion planning—remain foundational, but their reliance on accurate models, limited scalability, and brittle performance under model mismatch present challenges for modern systems. Reinforcement learning (RL), especially deep RL, has emerged as a promising alternative capable of handling nonlinear dynamics, partial observability, and unstructured sensory inputs. This reports provides a consolidated survey of robotic and autonomous system history, the classical foundations of the field, and focuses on RL for these systems, synthesizing insights from recent literature with emphasis on real-world successes, manipulation challenges, sample efficiency, safety, and sim-to-real transfer. We contextualize RL within classical robotics pipelines, review major algorithm families, and examine practical considerations for deploying policies on physical robots.. The goal is to offer a rigorous, unified, and comprehensive overview of the field, bridging foundational theory and real-world practice.

Contents

1 Introduction **2**

2 History **3**

2.1 Robotics . . . 3

2.2 Autonomous Driving . . . 6

3 Industry State of the Art **9**

3.1 Robotics: from structured factories to general purpose humanoids . . . 9

3.2 Autonomous driving: commercial deployments and their limits . . . 10

4 Classical Foundations of Robotics and Autonomous Driving **12**

4.1 Perception and State Estimation . . . 13

4.2 Mapping and Planning . . . 17

4.3 Control and Trajectory Tracking . . . 23

5 Reinforcement Learning in Robotics: A Unified Overview **28**

5.1 Foundations . . . 28

5.2 Modern Deep RL . . . 30

5.3 Real-World Successes in Robotic RL . . . 33

5.4 Challenges and Outlook . . . 35

6 Reinforcement Learning for Robotic Manipulation **36**

6.1 Problem setting and task families . . . 36

6.2 Learning setups and policy representations . . . 36

6.3 Large scale systems and datasets . . . 38

6.4 Open challenges and role of classical structure . . . 40

7 Reinforcement Learning for Autonomous Driving **41**

7.1 Driving as a sequential decision problem . . . 41

7.2 Where RL fits in the driving stack . . . 42

7.3 Training signals and data sources . . . 43

7.4 Hybrid approaches and open challenges . . . 46

8 Conclusion **48**

1 Introduction

Robotics and autonomous driving have progressed over the past few decades from speculative research topics to central technologies in manufacturing, logistics, healthcare, and mobility.[1, 2] Industrial manipulators weld and assemble automobiles at scale, mobile robots move shelves and parcels through vast warehouses, and autonomous driving systems are being tested, and in some cases deployed commercially on public roads. These systems sit at the intersection of mechatronics, sensing, control, and algorithms, and increasingly their performance is shaped by methods from machine learning.

Classically, both robotics and autonomous driving have been structured around a model-based pipeline. Sensors provide noisy measurements of the robot and its environment; estimation and mapping algorithms build geometric or probabilistic representations; motion planners compute feasible trajectories that respect dynamics and constraints; and feedback controllers track those trajectories while enforcing safety margins.[3, 4] This paradigm has led to highly reliable systems in structured and semi-structured environments, particularly on factory floors and well-mapped roads. However, it can struggle when confronted with high-dimensional sensory inputs (such as raw images and point clouds), contact-rich manipulation, or long-horizon decision making under uncertainty in open-ended, changing environments.

Machine learning, and deep learning in particular, has substantially reshaped this landscape. Supervised and self-supervised learning have become the dominant approaches for perception tasks such as object detection, semantic segmentation, and depth estimation in both robotics and autonomous driving.[5, 6] Reinforcement learning (RL) and imitation learning extend this data-driven perspective to control and decision making, enabling robots to acquire locomotion and manipulation skills from experience rather than relying solely on hand-designed controllers.[7, 8] In autonomous driving, learning-based policies have been explored for lane keeping, merging, and interaction with other road users, often trained in simulation and evaluated in constrained real-world scenarios.[9, 10] At the same time, most deployed systems remain hybrid: learned components are carefully wrapped inside classical estimation, planning, and control frameworks to satisfy safety, interpretability, and regulatory constraints.

The goal of this report is to provide a comprehensive, narrative survey of robotics and autonomous driving with an emphasis on reinforcement learning and its role in modern systems. We synthesize the state of the art across three layers: (i) the historical development of the fields, (ii) the modern industrial and research landscape, and (iii) the algorithmic foundations and recent advances in RL and deep RL. We highlight where learning methods, and RL in particular, have already had tangible impact in practice, where they remain largely experimental, and how they interact with long-established model-based techniques.

2 History

Before examining the modern marvel of robotics and autonomous driving, it is helpful to briefly review the historical development of these two fields. The basic idea that human-made artifacts might move, perceive, or even "think" on their own is not new: ancient myths describe artificial servants and self-moving machines, and early inventors in the classical and medieval worlds built mechanical automata that mimicked animals or people.[11] These speculative and practical efforts did not yet constitute robotics or autonomous vehicles in the modern sense, but they established a persistent intuition that intelligence and motion could be engineered rather than only born.

As scientific mechanics, electricity, and later computing matured, these early intuitions crystallized into concrete research programs. Robotics emerged at the intersection of automation, control theory, and artificial intelligence, focusing on embodied agents that can sense and act in the physical world. Autonomous driving developed in parallel, bringing similar questions about perception, decision making, and safety into the specific context of road vehicles and traffic systems. Over time, both trajectories converged: methods originally designed for industrial manipulators or mobile robots migrated into self-driving cars, and advances in vehicular autonomy fed back into broader robotic navigation and planning.

2.1 Robotics

The concept of physical artificial agents, human-made machines capable of sensing and acting in the world, long predates the term "robot". The earliest instance of such beings appear in ancient Greek mythology, where figures like *Talos*, a giant bronze guardian forged by *Hephaestus* to patrol the shores of Crete, embody the idea of an engineered, self-moving agent created for a specific task rather than born in the way of nature.[11]

Similar ideas continued to spark up over the course of human history: Judaism had the *golem*, a humanoid servant fashioned from clay and animated through sacred words,[12] Daoist texts such as the *Liezi* describe an artificer who presents a lifelike mechanical man before the king,[13] and other cultures told stories of artificial attendants, clockwork dolls, and self-moving automata—all pointing to a persistent intuition that purposeful, lifelike motion might one day be engineered.[11]



Figure 1: Talos, an early depiction of a robot-like automaton.



Figure 2: Robot rebellion in Čapek's *R.U.R.* (1921), which introduced the word "robot".

The leap from imagined artificial beings to the modern language of "robots" came much later. The word "robot" was first popularized in 1921 by Czech playwright Karel Čapek in his play *R.U.R. (Rossum's Universal Robots)*, where "robots" are mass-produced artificial workers designed to serve humans.[14] Although Čapek's robots are closer to biological androids than mechanical machines, the play crystallized enduring themes around artificial labor, autonomy, and control, and it supplied the term that would soon be adopted by engineers and scientists for real-world automated machines.[14]

A few decades later, Isaac Asimov's fiction reshaped how robots were imagined in science and popular culture. Beginning in the 1940s and collected in the 1950 volume *I, Robot*,[15] Asimov's stories introduced the now-famous *Three Laws of Robotics*, first formulated explicitly in the 1942 short story "Runaround".[16] Rather than portraying robots as mindless monsters, Asimov treated them as rational, law-governed agents constrained by built-in ethical rules: they could not harm humans, had to obey human orders, and were required to protect their own existence as long as this did not conflict with the first two laws. This shift from robots as external threats to robots as carefully engineered tools subject to safety constraints strongly influenced later discussions of human–robot interaction and robot ethics.

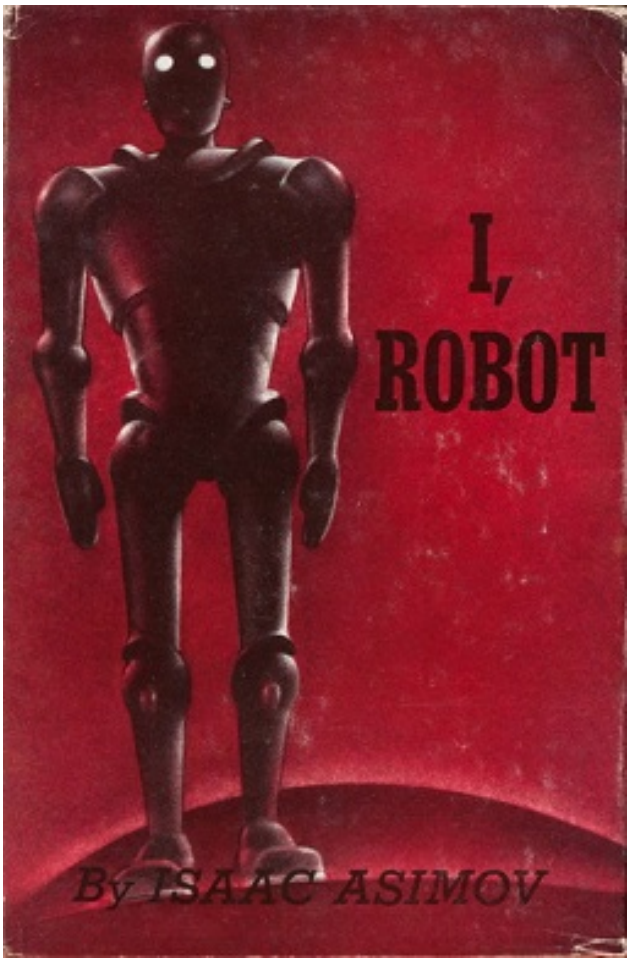


Figure 3: First-edition cover of Isaac Asimov's *I, Robot* (1950), which popularized the Three Laws of Robotics.

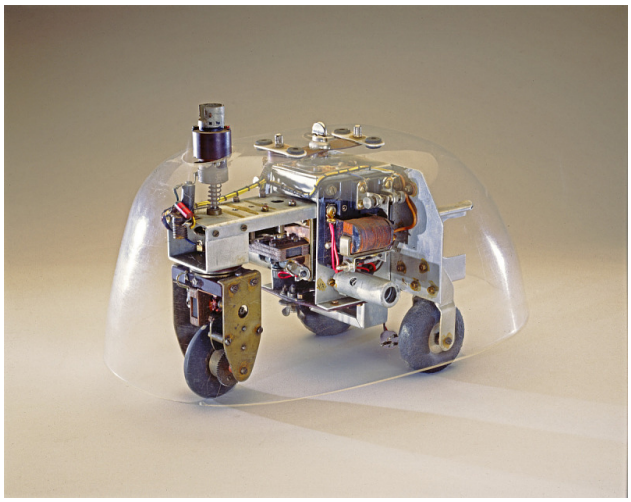


Figure 4: One of W. Grey Walter's "tortoise" robots (late 1940s), an early autonomous mobile robot.

With the conceptual vocabulary in place, physically realized robots began to appear in the mid-twentieth century. In the 1940s, Norbert Wiener's work on *cybernetics* framed control and communication in animals and machines within a unified mathematical theory, emphasizing feedback as the core principle underlying both biological and artificial behavior.[17] Around the same time, neurophysiologist **W. Grey Walter** built a series of small electromechanical "tortoise" robots—nicknamed *Elmer* and *Elsie*—that roamed around using light sensors and simple analog circuits, seeking out illumination and avoiding obstacles. These machines closed the loop between sensing and acting in hardware and demonstrated that relatively simple feedback systems could generate surprisingly lifelike, goal-directed behavior.[18]

Modern industrial robotics emerged shortly thereafter. In the late 1950s and early 1960s, **George Devol** and **Joseph Engelberger** developed *Unimate*, a programmable robotic arm designed for factory automation.[19] The first Unimate was installed on a General Motors assembly line in 1961, where it handled hot die-cast parts that had previously exposed workers to heat and injury.[19] This deployment is widely regarded as the birth of industrial robotics and inaugurated decades of growth in robotic manipulators across automotive and electronics manufacturing. In these domains, robots excelled at repetitive, high-precision tasks in structured environments, emphasizing reliability, speed, and safety over general autonomy.

In research laboratories, robotics took a different trajectory, becoming a testbed for artificial intelligence. From 1966 to 1972, SRI International's **Shakey** project produced one of the first mobile robots capable of perceiving its surroundings, building internal models, and planning multi-step actions.[20] Shakey combined cameras and range sensors with symbolic reasoning and automated planning, and it inspired core AI techniques such as the STRIPS planner and A* search. This "sense–plan–act" paradigm dominated research robotics for several decades, as platforms at SRI, Stanford, MIT, and CMU tackled problems in navigation, mapping, and manipulation in increasingly complex laboratory environments. This paradigm is still popular today, as will be discussed later in the "Classical Foundations" section.



Figure 5: Unimate, the first industrial robot, operating on a General Motors assembly line in the early 1960s.



Figure 6: Shakey the Robot (late 1960s), a landmark AI-driven mobile robot.

By the 1990s and 2000s, robots had begun to leave the factory and the lab. Field robots operated in hazardous or remote settings—on planetary surfaces, in disaster zones, or in agriculture—while *service and consumer robots* appeared in homes and commercial spaces. A notable milestone was iRobot's **Roomba**, introduced in 2002 as one of the first commercially successful domestic robots, which autonomously vacuumed floors using contact sensors and, in later generations, visual and lidar-based navigation.[21] At the same time, fleets of mobile robots in warehouses—exemplified by Kiva Systems (now Amazon Robotics)—reorganized fulfillment centers around autonomous drive units that ferried shelves of goods to human pickers.[22] These developments showed that robots could operate at scale in semi-structured, economically critical environments beyond the traditional assembly line.



Figure 7: First-generation iRobot Roomba (2002), an early mass-market domestic robot.



Figure 8: Kiva/Amazon warehouse robots reorganizing shelves in a fulfillment center.

Over roughly the last two decades, advances in sensing, computation, and data have pushed robotics into a machine-learning–centric era. Probabilistic methods such as Bayesian filtering and simultaneous localization and mapping (SLAM) made it possible to estimate robot state and build maps from noisy sensor data,[3] while deep learning became the dominant approach for perception tasks like object detection, semantic segmentation, and pose estimation. More recently, deep reinforcement learning and imitation learning have been applied to locomotion, dexterous manipulation, and whole-body control, enabling robots to acquire complex skills from data that would be difficult to hand-engineer.[7, 23] Contemporary robots are thus hybrid systems: they combine model-based control and planning with learned components that handle high-dimensional sensory input and adapt to new environments. This convergence of classical robotics with modern machine learning forms the backdrop for the developments examined in the remainder of this report.

2.2 Autonomous Driving

Compared to robotics, whose conceptual roots reach back to ancient myths of artificial servants and mechanical automata, the history of autonomous road vehicles is necessarily shorter. The very idea of a "driverless car" presupposes the existence of cars, paved roads, and coordinated traffic systems—technological and social infrastructures that only emerged over the last century and a half. As a result, the prehistory of

autonomous driving is less about timeless folklore and more about a focused sequence of engineering milestones from the Renaissance onward, and in what follows we highlight only those stages most relevant to modern machine-learning–based self-driving systems.

Ideas about vehicles that move themselves nevertheless appear early. In the late fifteenth century, Leonardo da Vinci sketched a three-wheeled, spring-powered "self-propelled cart" with a programmable steering mechanism and braking system.[24, 25] Although intended for theatrical effects rather than transportation, the design is often cited as a conceptual ancestor of both autonomous vehicles and mobile robots: motion is produced by stored mechanical energy, and the path is determined in advance by an internal mechanism rather than by a human driver. Much later, in 1925, Francis P. Houdina demonstrated a full-sized car in New York City steered by radio signals—a publicity stunt that showed a car apparently driving itself through crowded streets.[25]

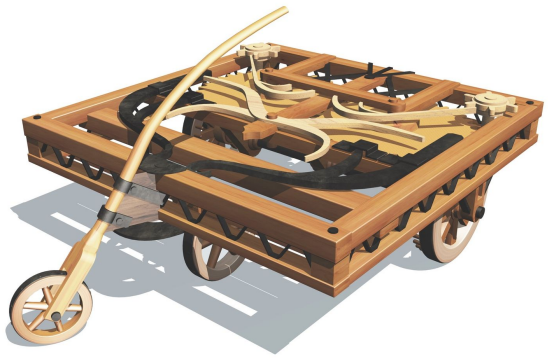


Figure 9: Reconstruction of Leonardo da Vinci's self-propelled cart, often cited as an ancestor of autonomous vehicles.



Figure 10: Francis P. Houdina's radio-controlled "American Wonder" car demonstrated in New York City (1925).

Mid-century visions such as Norman Bel Geddes's "Futurama" exhibit at the 1939 New York World's Fair, and General Motors' automated highway concepts of the 1950s and 1960s, imagined roads instrumented with inductive loops or guide wires that would keep vehicles in lane and at safe headways.[26, 25] These proposals relied on specialized infrastructure rather than onboard intelligence, but they crystallized core concerns about lane keeping, platooning, and traffic safety that continue to shape autonomous driving research and regulation today.

Autonomous driving as a sustained research discipline took shape in the 1980s and 1990s, when digital computing and sensing became fast enough for real-time perception and control. In Europe, Ernst Dickmanns and collaborators led the EUREKA PROMETHEUS project, fielding vision-based Mercedes–Benz test vehicles that could drive hundreds of kilometers on public highways, perform lane changes, and interact with traffic using multi-camera computer vision and model-based control.[27, 25] In the United States, Carnegie Mellon University's *Navlab* program and the ALVINN system used neural networks trained from human driving demonstrations to steer a van along roads, establishing the perception–planning–control pipeline that still underlies many autonomous driving stacks.[28, 26]



Figure 11: Ernst Dickmanns' PROMETHEUS test vehicle, a vision-based autonomous Mercedes–Benz on European highways.



Figure 12: Carnegie Mellon University's Navlab van used with the ALVINN neural-network driving system (late 1980s).

A major inflection point came with the DARPA Grand Challenges of 2004 and 2005 and the Urban Challenge in 2007. In 2005, Stanford's Volkswagen Touareg "Stanley" completed a 212 km off-road course to win the second Grand Challenge, while Carnegie Mellon's "Boss" won the Urban Challenge by navigating an urban-style environment with intersections, traffic rules, and other autonomous and human-driven vehicles.[29, 30, 26] These competitions demonstrated that fully autonomous driving was technically feasible and produced a cohort of researchers who went on to lead industrial programs. Over the following decade, companies such as Google (later Waymo) developed lidar- and radar-equipped self-driving prototypes, while suppliers like Mobileye scaled vision-based advanced driver-assistance systems (ADAS)—lane keeping, automatic emergency braking, adaptive cruise control—to millions of production vehicles.[26, 25]



Figure 13: Stanford's Volkswagen Touareg "Stanley", winner of the 2005 DARPA Grand Challenge.



Figure 14: Carnegie Mellon University's "Boss", winner of the 2007 DARPA Urban Challenge.

Over the last decade, autonomous driving has increasingly adopted deep learning for perception and decision making. Convolutional neural networks became the dominant method for detecting lanes, vehicles, and pedestrians in camera images, and end-to-end approaches showed that a single neural network could map raw sensor input directly to steering commands under constrained conditions.[9, 26] At the same time, probabilistic mapping and localization techniques developed in robotics—such as SLAM and particle-filter–based tracking—were adapted to large-scale road networks. Contemporary autonomous driving systems mirror modern robotics: they combine high-definition maps, model-based planning, and feedback control with learned components for perception, prediction, and sometimes control, forming a hybrid architecture that underlies the machine-learning–centric developments examined later in this report.

3 Industry State of the Art

The history section showed how robots and autonomous vehicles emerged from early industrial arms and research prototypes. In this section we look at where industry is today. On the robotics side there is a clear split between very mature, reliable systems in factories and warehouses, and a new wave of "general purpose" platforms that are still in pilot projects. On the autonomous driving side there is a similar divide: advanced driver assistance is deployed at large scale, but fully driverless services exist only in a few cities and under strict conditions.

3.1 Robotics: from structured factories to general purpose humanoids

Industrial robots have been part of manufacturing for decades. Six degree of freedom arms from companies such as FANUC, ABB, and KUKA weld car bodies, place electronic components, and move heavy parts with high speed and repeatability. They usually operate in fenced work cells with fixed fixtures and carefully programmed motion scripts. Once set up, they are extremely reliable, but they do not handle surprises well; if a part is misplaced or a sensor fails, the robot typically stops and waits for a human.

Logistics and e—-commerce drove the next major wave. Companies like Amazon Robotics use fleets of mobile robots to move shelves and totes in warehouses, while human workers handle fine manipulation. Autonomous mobile robots from firms such as Locus Robotics or OTTO Motors follow maps through aisles, delivering goods between stations. These systems operate in semi structured environments: the layout is known, the tasks are repetitive, and human workers are trained to move in predictable ways around the robots.[4]

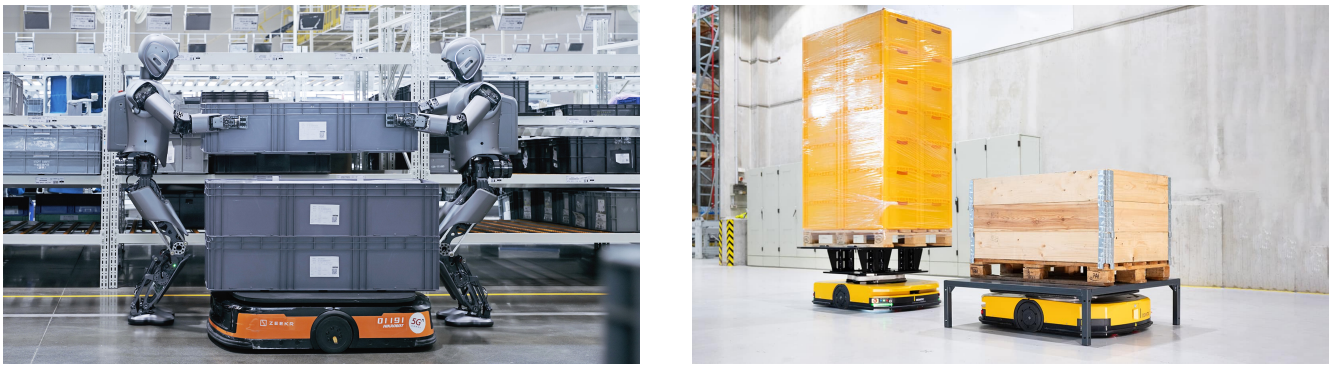


Figure 15: Examples of industrial and logistics robots in practice. Left, a humanoid style robot piloted in a factory setting for pick and place and tending machines. Right, fleets of autonomous mobile robots moving shelves in a warehouse.

Service and medical robots have also made the jump from research to real deployment. Robotic floor scrubbers clean supermarkets and airports, often following pre recorded routes with local obstacle avoidance. Surgical systems such as Intuitive Surgical's da Vinci platform assist human surgeons by providing precise, tremor free control of instruments inside the body. In all of these examples, autonomy is real but narrow: the environment is constrained, the set of tasks is small, and a human operator is always close by.

The "physical intelligence" wave. In the last few years a new group of companies has started to frame robotics as a "physical intelligence" problem. The idea is to give robots a general understanding of the physical world that lets them adapt to many tasks, not just one pre programmed routine, similar in spirit to how large language models capture patterns in text.[31]

Several startups and large firms are betting on humanoid robots as a convenient form factor for this physical intelligence. Figure AI's Figure 01 is a bipedal humanoid designed to work in existing industrial environments. It is being piloted with partners such as BMW on tasks like moving parts and tending machines, and uses a mix of classical control and learned models for perception and high level behavior.[32] 1X's NEO is a humanoid robot explicitly aimed at the home, intended to safely share space with people and eventually help with everyday chores.[33] Tesla's Optimus and Agility Robotics' Digit occupy a similar niche, focusing on warehouse and light industrial work.

In parallel, hardware and software vendors such as NVIDIA are releasing shared toolchains and "foundation models" for robotics, for example within the Isaac stack, that are meant to be reused across many robot bodies and tasks. These platforms combine simulation, perception, planning, and learning tools, and are often used as the experimental playground for the physical intelligence vision.

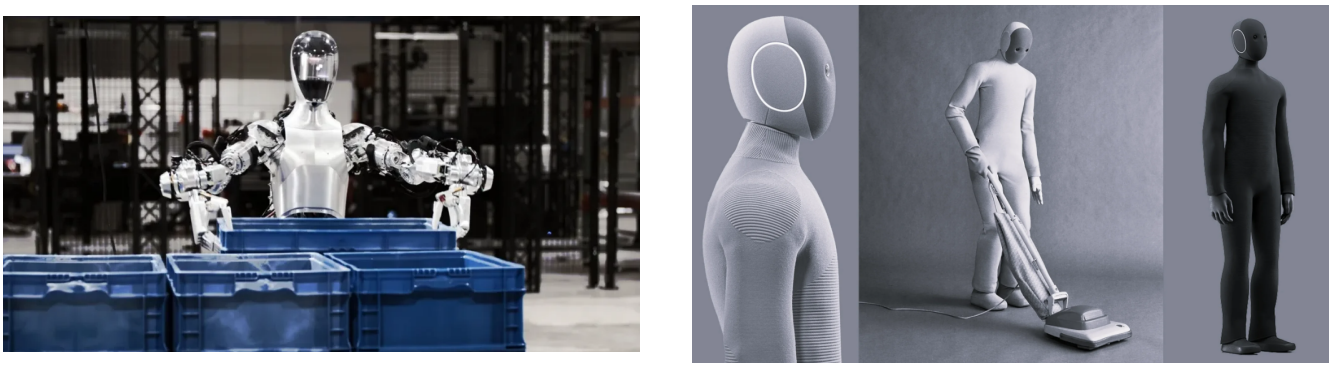


Figure 16: Examples of the physical intelligence wave in robotics. Left, a Figure 01 style humanoid being piloted in an industrial setting. Right, a NEO style humanoid in a home like environment intended for everyday assistance.

Why we are still early. Despite impressive demos, these humanoid and general purpose robots are still far from being household products. The hardware is expensive and delicate, batteries limit how long a robot can work, and many tasks still require dexterity and force control that rival a trained human worker. Most deployments are small pilot programs with a human in the loop for safety or remote assistance.

For this reason, most companies that rely on robots today still focus on structured environments and well defined tasks. The bleeding edge of physical intelligence research is happening in parallel, but commercial systems are largely built on the classical foundations from Section 4, sometimes augmented with deep learning based perception or control modules. A robot that can walk into any home or factory and help with whatever needs doing remains an active research goal rather than a common appliance.

3.2 Autonomous driving: commercial deployments and their limits

On the road, the most widespread form of autonomy today is not full self driving but advanced driver assistance. Many new cars ship with lane keeping, adaptive cruise control, and automatic emergency braking. These systems rely on cameras, radar, and sometimes lidar to keep the car centered in its lane and maintain a safe following distance, but they assume a human driver is always paying attention and ready to take over.

Tesla's Autopilot and Full Self Driving (FSD) packages are among the best known commercial examples. Tesla vehicles use a camera only sensor suite and large neural networks trained on billions of miles of driving data collected from the customer fleet. The system can handle highway driving, lane changes, and, in its most advanced versions, city streets with traffic lights and turns. However, regulators and Tesla

itself still classify FSD as a driver assistance feature: the driver must remain attentive and responsible at all times, and the car can disengage without warning.[34] In practice this means that even when the software handles the low level details, fully unattended operation is not guaranteed.

Waymo pursues a different model based on robotaxis. In a small but growing number of cities, Waymo operates driverless ride hailing services using vehicles equipped with lidar, radar, and cameras, as well as high definition maps. Riders can use a smartphone app to summon a car with no human driver in the front seat, at least within a geofenced service area and under suitable weather and traffic conditions.[35] Technically, these systems build on the same ingredients as our classical foundations: perception, localization against HD maps, prediction of other agents, and a layered planning and control stack, often with machine learning inside the perception and decision modules.

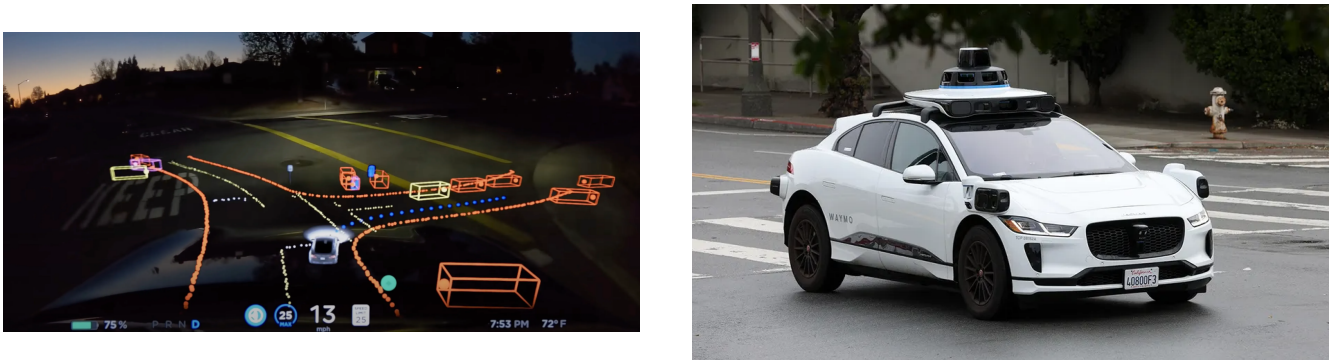


Figure 17: Examples of commercial autonomous driving systems. Left, a Tesla Full Self Driving visualization, where the system controls steering and speed but the human driver must supervise. Right, a Waymo robotaxi operating in a geofenced urban service area without a human driver in the front seat.

The last decade has also seen setbacks. Cruise, General Motors' robotaxi division, suspended its services after a series of safety incidents and regulatory actions, and GM later announced that it would scale back standalone robotaxi efforts in favor of integrating advanced assistance into conventional vehicles.[36] Other startups have quietly reduced ambitions or pivoted away from fully driverless plans. These changes highlight how difficult it is to ensure safety, robustness, and public trust when autonomous vehicles share dense urban streets with human drivers and pedestrians.

Suppliers such as Mobileye, NVIDIA, and others now sell hardware and software stacks that automakers can integrate into their own vehicles. These stacks typically target "Level 2+" or "Level 3" capabilities such as highway pilot or traffic jam assist: the car can handle driving for extended periods under specific conditions, but the human driver may still be required to supervise or to take over when the system reaches its limits.

Commercial reality versus research ambition. Putting these pieces together, the industry picture for autonomy is mixed. On one hand, robots and autonomous functions are quietly everywhere in factories, warehouses, hospitals, and millions of passenger cars. On the other hand, the science fiction vision of fully autonomous robots helping in every home and self driving cars on every street is still a long way off.

Across both robotics and autonomous driving, the companies closest to deployment tend to favor hybrid designs. They combine classical mapping, planning, and control with machine learning components for perception, prediction, and short horizon decision making. Reinforcement learning plays a growing role in simulation and in specialized modules, but rarely runs an entire commercial product end to end on its own. The gap between what we can do in research settings and what is safe, robust, and affordable enough for everyday use is exactly the gap that motivates the techniques surveyed in the rest of this report.

4 Classical Foundations of Robotics and Autonomous Driving

Now that we have discussed both the history, and the modernity of robotics and AV in industry, we can dive deeper under the hood and examine the algorithms and pillars on which these domains stand upon.

Modern robotic and autonomous driving systems are often described as "learning-enabled", but the majority of deployed platforms—and many research systems—still rely heavily on classical model-based techniques. These techniques organize the problem into a perception and state-estimation layer, a mapping and planning layer, and a control layer that executes planned motions under physical and safety constraints.[3, 4, 1] This section briefly reviews these foundations, which form the backbone into which machine learning and reinforcement learning components are now integrated.

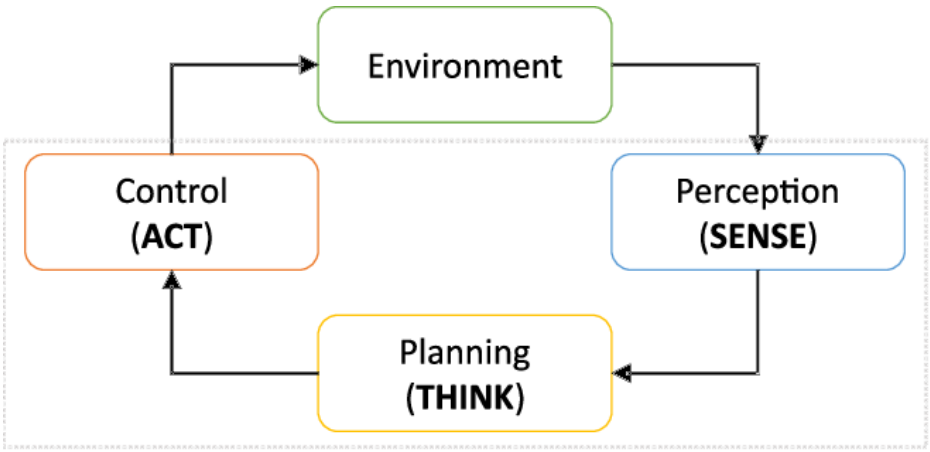


Figure 18: Canonical sense–think–act loop for an embodied autonomous system. Perception converts raw sensor data into state estimates; planning reasons over these estimates and task goals to choose actions; control executes the chosen actions on the robot, which in turn interacts with the environment and produces new observations.

From a systems perspective, almost any embodied autonomous agent can be viewed as repeatedly executing a *sense–think–act* loop. Sensors measure the environment and the robot itself; perception and state estimation convert these raw signals into a compact state representation; planning and decision-making choose actions to move the system toward its goals while satisfying constraints; and low-level control turns these actions into actuator commands that change the environment and generate new observations. Decomposing the problem into these three pillars—perception and state estimation, mapping and planning, and control and trajectory tracking—provides a modular architecture that is analysable, engineerable, and scalable across platforms. Figure 18 illustrates this canonical cycle.

It is important to note that, at this level of abstraction, robotics and autonomous driving are not fundamentally different fields but instances of the same architecture. A mobile manipulator in a warehouse and an autonomous car on a highway both maintain estimates of their state, build and update a representation of their surroundings, plan motions under dynamic and safety constraints, and apply feedback control to track those motions. What differs are the embodiment and operating conditions:

manipulators reason about joint angles, end-effector poses, and nearby objects, whereas vehicles reason about lanes, other traffic participants, and road geometry; manipulators rely primarily on encoders and cameras, while vehicles additionally integrate lidar, radar, GPS, and wheel odometry. For this reason, the classical foundations presented in this section are treated in a unified manner, with examples drawn from both domains.

4.1 Perception and State Estimation

If the sense–plan–act loop is a story, perception and state estimation are the opening scene where the robot wakes up and tries to understand where it is and what surrounds it. Cameras, lidars, radars, encoders, and IMUs all start streaming numbers. On their own these numbers mean very little. The job of perception is to turn them into something that has structure and meaning: where the robot is, what the environment looks like, where obstacles and other agents are, and which parts of the scene are roads, walls, shelves, or people.[3, 1]

Seeing, hearing, and feeling the world. Different autonomous sense the world in different ways. An industrial arm on an assembly line mostly cares about its own joints and a small workspace in front of it, so it relies heavily on joint encoders, force or torque sensors, and perhaps a fixed camera that watches the conveyor belt. A legged robot uses IMUs and joint sensors to feel its own motion, and depth cameras or lidar to understand the terrain. An autonomous car combines stereo or monocular cameras, several lidars, radars, GPS, and wheel odometry to perceive the road, lanes, and other traffic participants.[4]



Figure 19: Examples of sensors used in modern robotic systems, including proximity sensors, RGB and RGB-D cameras, ultrasonic range sensors, 2D and 3D lidars, tactile and force/torque sensors, and specialized depth or structured-light sensors.

At a high level, all of these sensors fit the same pattern. There is a hidden state x_t that describes where the robot is and how it is moving, and a measurement z_t that the sensor provides. Classical perception

models this relationship as a function with noise,

$$z_t = h(x_t) + v_t, \tag{1}$$

where h describes how the world should look to that sensor, and v_t accounts for noise and small errors. For a GPS receiver, h simply returns the position of the car in world coordinates. For a wheel encoder, h returns the wheel rotation. For a camera, h follows the rules of projective geometry.

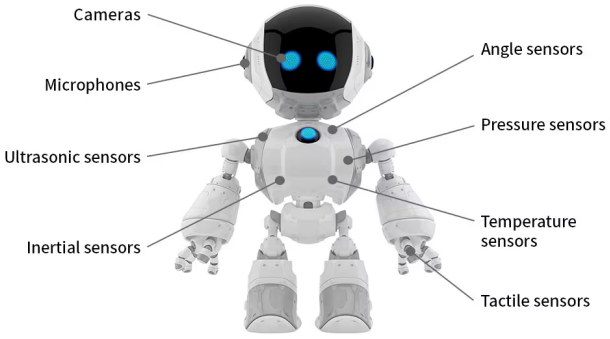


Figure 20: Example sensor suite on a mobile robot with encoders, IMU, depth camera, and 2D lidar.

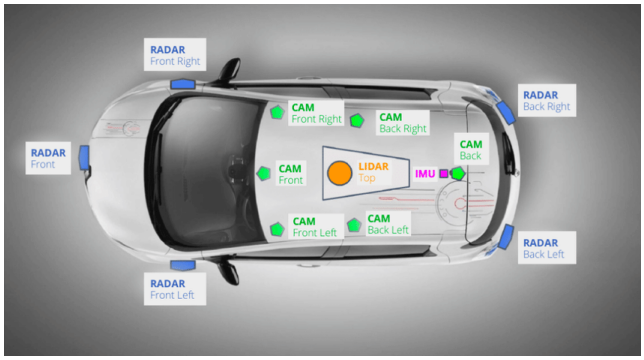


Figure 21: autonomous vehicle sensors, including cameras, lidar, radar, GPS, and wheel odometry.

Classical geometric vision and range perception. Before deep neural networks were widely used, robotic vision was built from relatively simple but carefully engineered geometric pipelines. A classic example is feature based structure from motion. The robot detects interest points in each image (for example, corners on a door frame or texture on a poster), describes their local appearance with a small descriptor, and then matches the same points across multiple frames.[37] From these matches, it can estimate how the camera moved and, by triangulation, reconstruct the three dimensional positions of the points in the world.

The basic mapping from a 3D world point X to an image point u is described by the pinhole camera model,

$$\tilde{\mathbf{u}} \sim K\,[R \mid t]\,\tilde{\mathbf{X}}, \tag{2}$$

where K contains the camera intrinsics, and R and t describe the camera pose.[37] This model is the workhorse behind tasks such as pose estimation of objects for grasping, visual odometry for mobile robots, and lane detection for vehicles that rely on camera images.

Range sensors add another style of perception. A 2D lidar on a mobile robot sweeps a laser beam around and reports distances to the nearest obstacle in each direction. A 3D lidar on an autonomous car produces a dense point cloud of returns on cars, buildings, trees, and road surfaces. Classical algorithms like Iterative Closest Point (ICP) align successive scans, allowing the robot to estimate how it moved between scans and to stitch the scans together into a map.[38] Simple clustering and curve fitting can turn these point clouds into objects, walls, curbs, and lane boundaries.

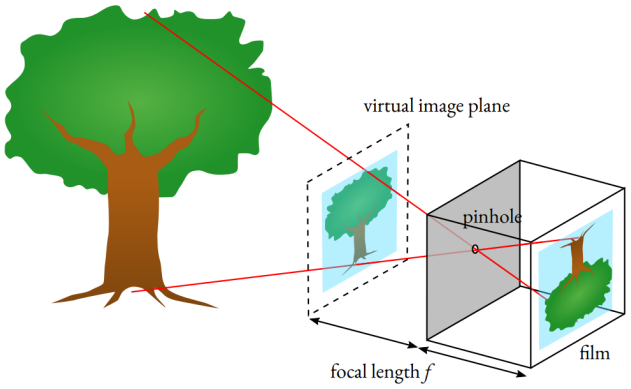


Figure 22: Pinhole camera geometry. A 3D point projects to the image plane using the intrinsic matrix and camera pose.

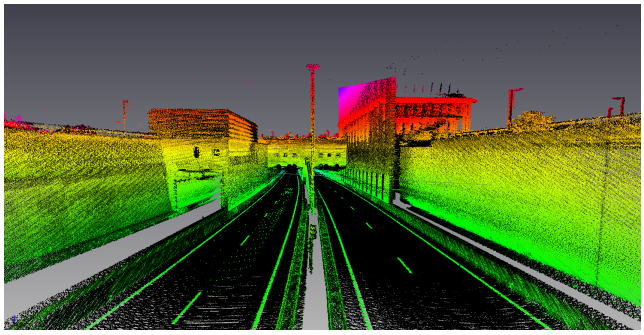


Figure 23: Lidar point cloud from a street scene. Points on the road, sidewalk, and nearby vehicles are visible.

From raw measurements to a belief over state. So far, we have described how to extract geometric information from individual sensor readings, but a robot also needs a stable sense of where it is over time. Wheel odometry alone drifts, GPS can be noisy or unavailable, cameras can be blinded by glare, and lidar can be affected by rain or dust. Classical autonomous systems deals with this by treating the state as a random variable and maintaining a belief over it that is updated whenever new data arrives.

A common viewpoint is the Bayes filter. The state x_t changes over time according to a motion model, and each observation z_t provides information about x_t. In words, the filter alternates between two steps:

- Predict: use the motion model and the previous belief to predict what the state distribution should look like at the next time step.
- Update: use the new measurement to correct that prediction and obtain a refined belief.

In the special case where the models are linear and the noise is Gaussian, this recursion becomes the Kalman filter, which is widely used in aircraft, mobile robots, and vehicles.[3] Variants such as the Extended Kalman Filter adapt it to nonlinear systems. More general particle filters approximate the belief with many samples and are useful when the belief is not well modelled by a single Gaussian.

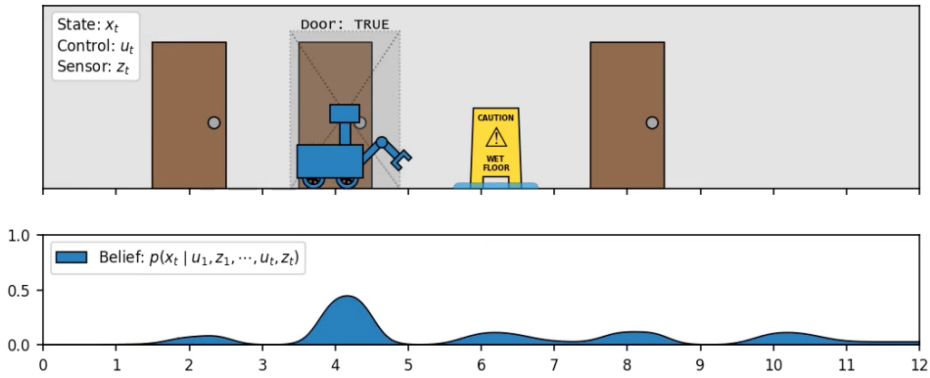


Figure 24: High level view of a Bayes filter. The motion model predicts the state distribution forward in time. The observation model and new measurements are then used to correct this prediction.

In a warehouse robot, such a filter might combine wheel odometry, an IMU, and scan matching from a 2D lidar to maintain a smooth estimate of the robot pose on the floor. In an autonomous car, a similar filter fuses GPS, IMU, wheel odometry, multiple lidars, and radar to estimate the vehicle pose and velocity

relative to a map and to track its lane position.[4] In a manipulator, joint encoders and torque sensors are fused to estimate joint angles and velocities, which are then used for precise motion control.

SLAM, visual inertial odometry, and tracking other agents. If the robot is placed into an unknown environment, it has to build a map at the same time as it localizes itself. This is the simultaneous localization and mapping (SLAM) problem.[39] In classical SLAM, the robot incrementally builds a map representation, such as an occupancy grid or a set of point landmarks, while estimating its own trajectory. When it revisits a previously seen place, loop closure constraints are added, and the full history of poses and landmarks is adjusted to be globally consistent. Many modern SLAM systems formulate this as an optimization problem on a graph, where nodes represent robot poses and landmarks, and edges represent constraints from odometry and sensor measurements.[40]

For robots and vehicles with cameras and IMUs, visual inertial odometry (VIO) provides a lightweight alternative to full SLAM. VIO tracks visual features across frames and uses the IMU to constrain short term motions, producing an estimate of motion with low drift.[41] This is a common choice for drones, mobile robots, and cars that need accurate relative motion even when GPS is unreliable.

Perception is not only about ego motion. An autonomous car must also keep track of other cars, cyclists, and pedestrians. A common classical solution is to assign a simple kinematic state to each tracked object and run a small Kalman filter per object to predict where it will move in the next few seconds. Data association algorithms, such as nearest neighbour or joint probabilistic data association (JPDA), match new sensor measurements, for example radar returns or bounding boxes from detectors, to existing tracks.[42] Service robots in malls or hospitals use similar ideas to track people and avoid collisions.

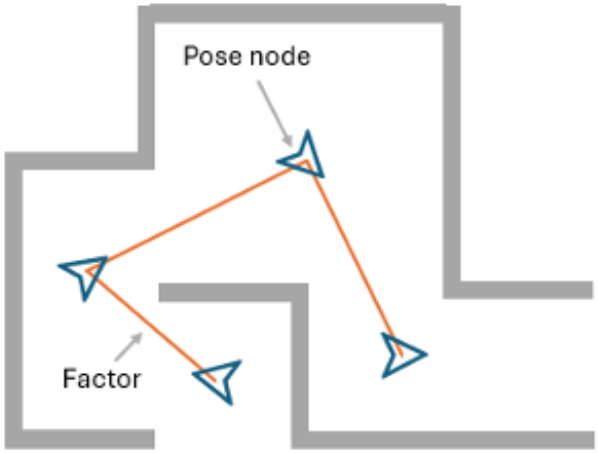


Figure 25: A simple SLAM factor graph. Pose nodes are connected by odometry constraints, and landmark nodes are linked by observation constraints.

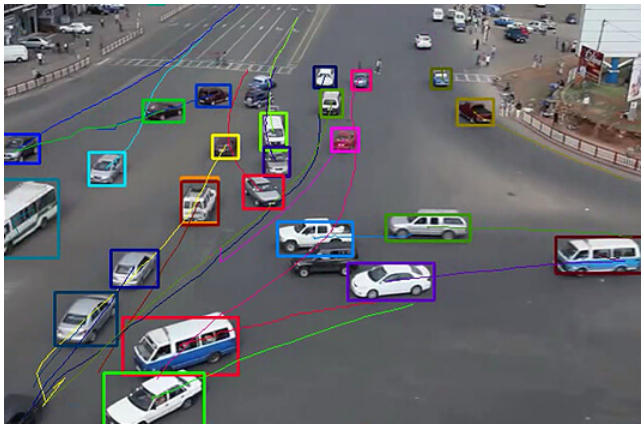


Figure 26: Multi object tracking in an urban scene. Each tracked vehicle is represented by a state estimate and an uncertainty ellipse.

Classical perception in a learning based era. Modern systems increasingly use deep networks to detect objects, segment images, estimate depth from a single camera, or predict optical flow. Instead of hand designed feature detectors and geometric heuristics, a large network is trained on millions of images so that, given a new image, it can mark which pixels belong to cars, people, roads, or shelves, and draw bounding boxes around objects of interest.

A recent trend is the rise of general purpose vision models that can be reused across many robots and

tasks. For example, the Segment Anything family of models (such as SAM and SAMv2) was developed by Meta AI. These models can take almost any natural image and, given a simple prompt like a point, box, or rough outline, return a high quality segmentation mask for "that thing" in the scene. In a warehouse, a robot might click near a box and immediately get the pixels that belong to that box. In a kitchen scene, the same model can be used to highlight the mug on the table or the handle on a cabinet.

Similarly, GroundingDINO and related models, originally released by IDEA Research and collaborators, combine object detection with language. Instead of asking for an internal class ID like "class 7," the robot can ask in plain language: "find all chairs," "find the red car," or "find the traffic lights." The model returns bounding boxes and confidence scores for the objects that match the text description. This makes it much easier to hook vision into higher level planning and instruction following: if a human says "pick up the blue bottle," a grounding model can help the robot find which pixels and which 3D points belong to "the blue bottle" in its camera image.

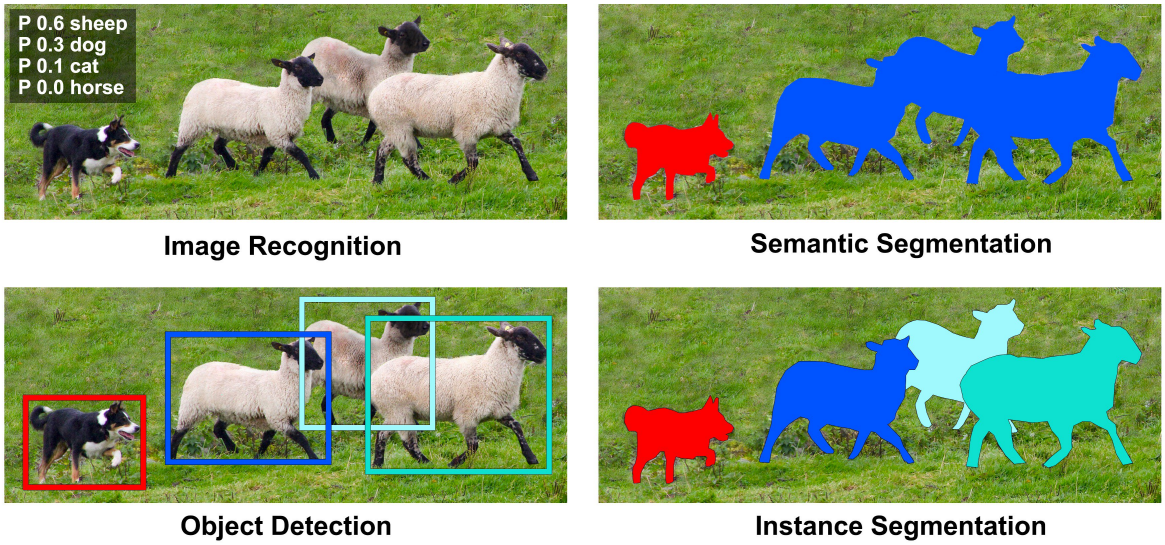


Figure 27: Example output of an object detection and segmentation model (coco). Given an image, the model predicts the pixels that are associated with each animal class, as well as boxing each instance of an animal.

Even with these powerful networks, the overall structure of the perception stack has not disappeared. Network outputs still flow into filters and SLAM back ends to estimate the robot's motion and build maps. Ego motion still flows into mapping and planning as a pose estimate with uncertainty. Tracks of other agents still come from prediction and data association modules, even if the predictions or association scores are learned.

In this sense, classical perception and state estimation remain the backbone of autonomous robots and vehicles. They define the geometry, the state, and the uncertainties that planning, control, and later reinforcement learning modules must work with. In the next subsection we turn to mapping and planning, where these estimates are turned into decisions about where to go next.

4.2 Mapping and Planning

Perception and state estimation tell the robot what the world looks like right now. Mapping and planning answer the next questions: what does the world around me look like as a whole, and how do I get from where I am to where I want to be without crashing into anything? In other words, mapping gives the

robot an internal picture of its surroundings, and planning uses that picture to choose safe, reasonable motions.[3, 4]

Why robots need maps. If a robot only reacted to the most recent sensor readings, it would behave like a person walking in the dark with a very small flashlight. It might avoid the next obstacle, but it would struggle to plan a route through a building or decide which street to take to reach a destination.

A map gives the robot memory. It stores where walls, shelves, doors, and other objects are located, at least approximately. For a warehouse robot, the map might show aisles and racks. For an autonomous car, it might show lanes, intersections, crosswalks, and speed limits.[3, 4] The robot still uses fresh sensor data, but it does so on top of this stored picture of the world.

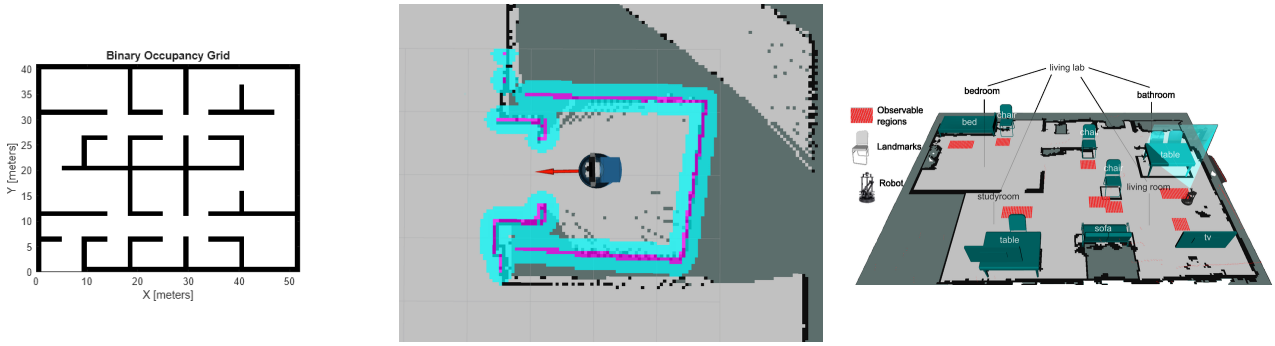


Figure 28: Examples of map representations used in practice: an occupancy grid, a local costmap, and a semantic map of a house. The robot reasons over these internal maps rather than the raw sensor data alone.

Occupancy grids, costmaps, and HD maps. The most common map for mobile robots is the occupancy grid.[3] Imagine laying a checkerboard over the floor plan. Each square (cell) has a number

$$p_i = P(\text{cell } i \text{ is occupied}),$$

which we interpret as "how likely is it that this cell contains an obstacle." When a laser scan or depth image shows a hit in that direction, the robot increases p_i for the corresponding cells. When a beam passes through an area with no hits, the cells along that ray have their p_i decreased. Over time, a two dimensional black and white map of walls and free space emerges.

Implementations often work with *log odds*

$$\ell_i = \log \frac{p_i}{1 - p_i},$$

because new measurements can then be incorporated by simple addition rather than more expensive probability formulas. Intuitively, each sensor reading nudges the belief for each cell up or down.

On top of this binary view, many systems build a *costmap*. Instead of just "free" or "blocked," each cell is assigned a traversal cost c_i. Cells very close to obstacles get high cost, encouraging the planner to stay away from walls or table legs. The robot's footprint is effectively "inflated" on the map, so that gaps narrower than the robot itself are treated as blocked even if there is technically a sliver of free space in the raw grid.

Autonomous vehicles often go further and use high definition maps.[4] These maps are not simple grids. They are structured databases that include lanes, lane boundaries, traffic lights, stop signs, crosswalks,

and sometimes even the three dimensional shape of buildings and curbs. The car localizes itself in this map and uses it to interpret what its sensors see. For example, if perception detects a red light, the HD map helps determine whether that light controls the current lane or some other direction.

There is also growing interest in *semantic maps*, where cells or regions carry labels such as "floor," "table," "shelf," "door," "crosswalk," or "sidewalk." These labels help the robot decide not only where it can move, but also which actions make sense in each place (for example, "place an object on a table," "do not drive on the sidewalk").

From world space to robot space. Maps are usually stored in world coordinates, for example meters in a floor plan or a global coordinate system for a car. Planning, however, is easier to describe if we think in terms of the robot's own possible poses.

We use the word *configuration* for a complete description of the robot's pose. For a car, a configuration might be

$$q = (x, y, \theta),$$

the position (x, y) on the road plane and the heading angle θ. For a robot arm with n joints, a configuration is the vector of joint angles

$$q = (q_1, q_2, \ldots, q_n).$$

The set of all possible configurations is called the configuration space and is usually denoted C.

Some configurations in C are safe, and some lead to collisions. It is useful to separate

$$C_{\text{free}} = \{\text{collision free configurations}\}, \qquad C_{\text{obs}} = \{\text{configurations that collide with obstacles}\}.$$

A path for the robot is then a continuous curve $q(t)$ in configuration space that starts at the initial pose and ends at the goal pose, while staying inside C_{free}.

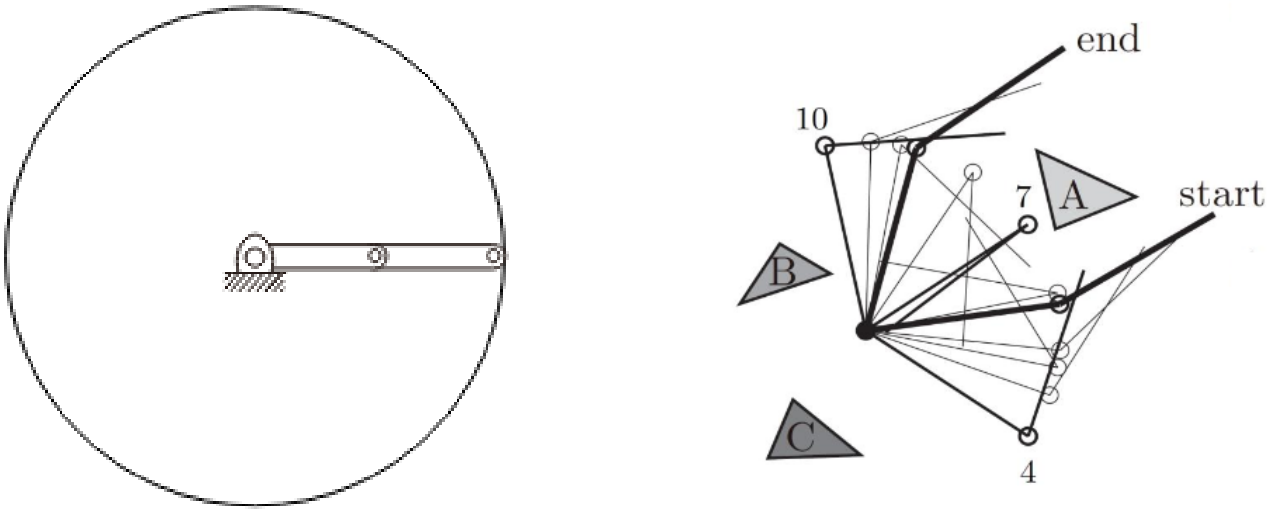


Figure 29: Configuration space idea. A simple obstacle in the workspace of a two link arm becomes a more complex forbidden region in the space of joint angles. Planners search for paths that stay in the safe region.

Figure 29 shows a basic example. An obstacle that looks like a rectangle in the room turns into a curved "no go" region in the space of joint angles (θ_1, θ_2). A valid motion is a path $q(t)$ that stays in the safe region C_{free}.

Search based planning: grids and road graphs. Once the robot has a map, it needs to find a path through it. A simple idea is to overlay a grid or graph on the map and search over that structure. Each grid cell center or graph node represents a possible location (or pose), and edges represent small moves between them.

We can think of this as a graph $G = (V, E)$, where V is the set of nodes and E is the set of edges. Each edge $e \in E$ has a cost $c(e)$, for example the distance to move along that edge. The planning problem becomes: find a path made of edges whose total cost

$$\text{cost}(\text{path}) = \sum_{e \in \text{path}} c(e)$$

is as small as possible.

A classic algorithm in this setting is A*. It is similar to Dijkstra's shortest path algorithm, but it also uses a guess (a heuristic) of how far each node is from the goal. This "look ahead" guess lets A* focus its search toward promising directions while still guaranteeing that the final path is the shortest one on the graph, as long as the guess never overestimates the true remaining cost.

For indoor robots, A* on an occupancy grid is often enough to find a good global route through rooms and corridors. For vehicles, planners usually run A* or related algorithms on a road graph built from the HD map.[4] This gives a high level route: which highway to take, where to exit, and which city streets to follow.

Nonholonomic robots (robots that cannot move sideways, like cars or forklifts) use a small twist on the idea. Instead of allowing arbitrary grid moves, they use a set of short, feasible motion primitives such as "drive forward and turn slightly left." Planners connect these allowed motions to form a lattice. Any path found on the lattice is automatically something the robot can physically follow.

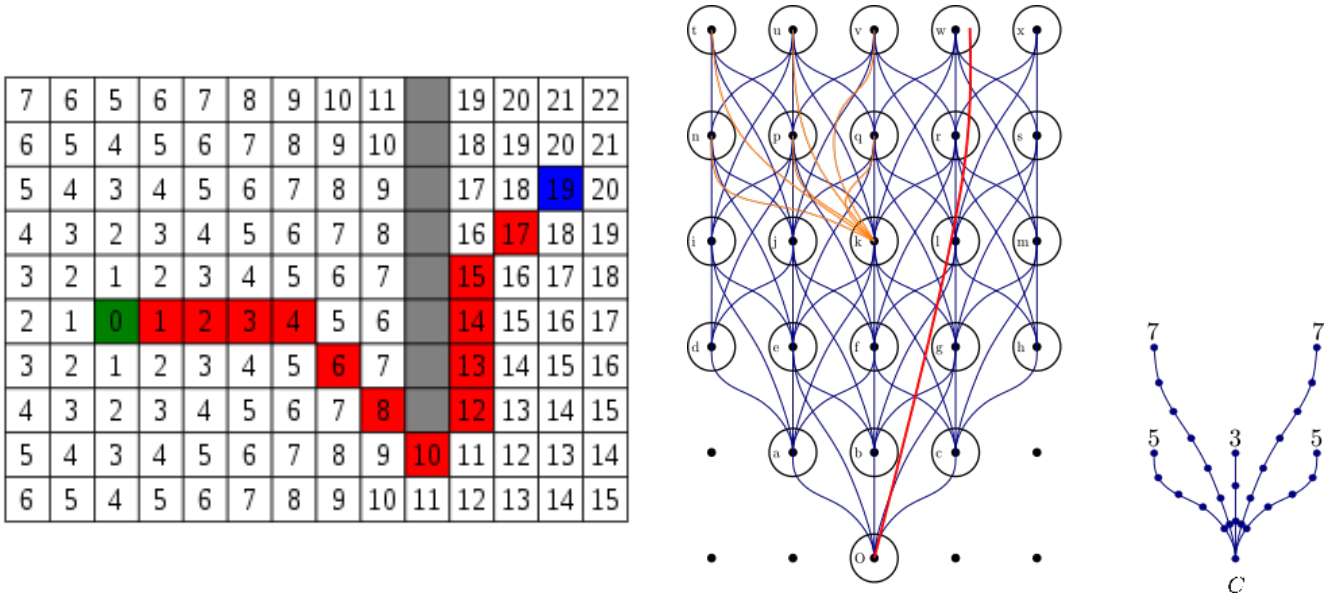


Figure 30: Search based planning. On the left, A* finds a shortest path on a 2D grid. On the right, a lattice planner for a car uses short, car-like motion primitives so that the planned path is directly trackable.

Sampling based planning: exploring by random trial. Grids work well in two or three dimensions, but they quickly become huge when the robot has many joints. Planning a path for a seven degree of

freedom arm in a cluttered shelf with a fine grid would be extremely expensive.

Sampling based planners handle this by exploring the safe region with random samples instead of a fixed grid.[1] Formally, they draw random configurations q in configuration space and keep only those that are collision free, so $q \in C_{\text{free}}$.

Two important examples are:

- **Probabilistic Roadmaps (PRM).** The planner draws many random configurations $q_1, q_2, \ldots$ in C_{free}, then tries to connect nearby ones with simple motions. These connections form a roadmap of safe moves. Once the roadmap is built, the robot can answer many queries by searching this roadmap graph.
- **Rapidly Exploring Random Trees (RRT).** The planner starts from the initial configuration and grows a tree. At each step, it picks a random configuration q_{rand}, finds the closest point q_{near} already in the tree, and extends the tree a little bit in that direction to a new node q_{new}. Over time, the tree spreads into unexplored parts of C_{free} and eventually reaches the goal.

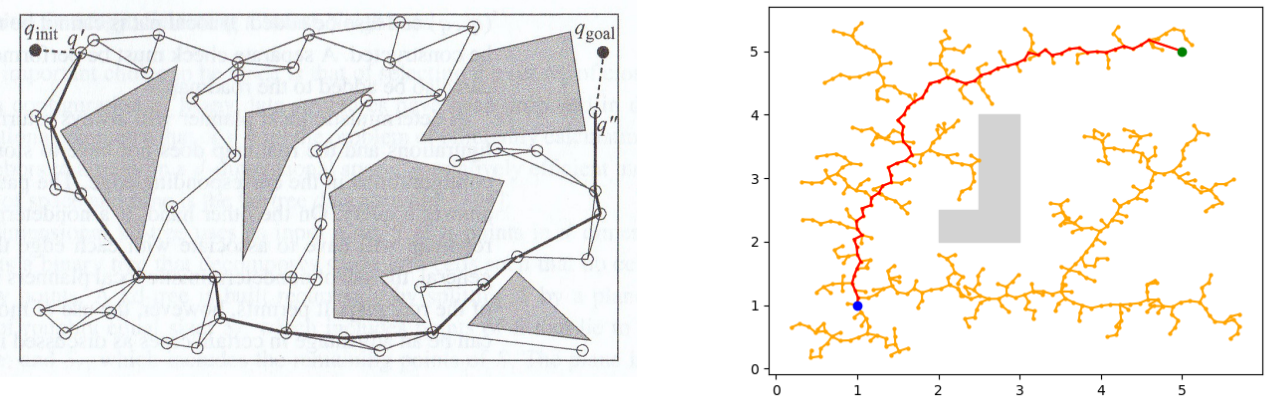


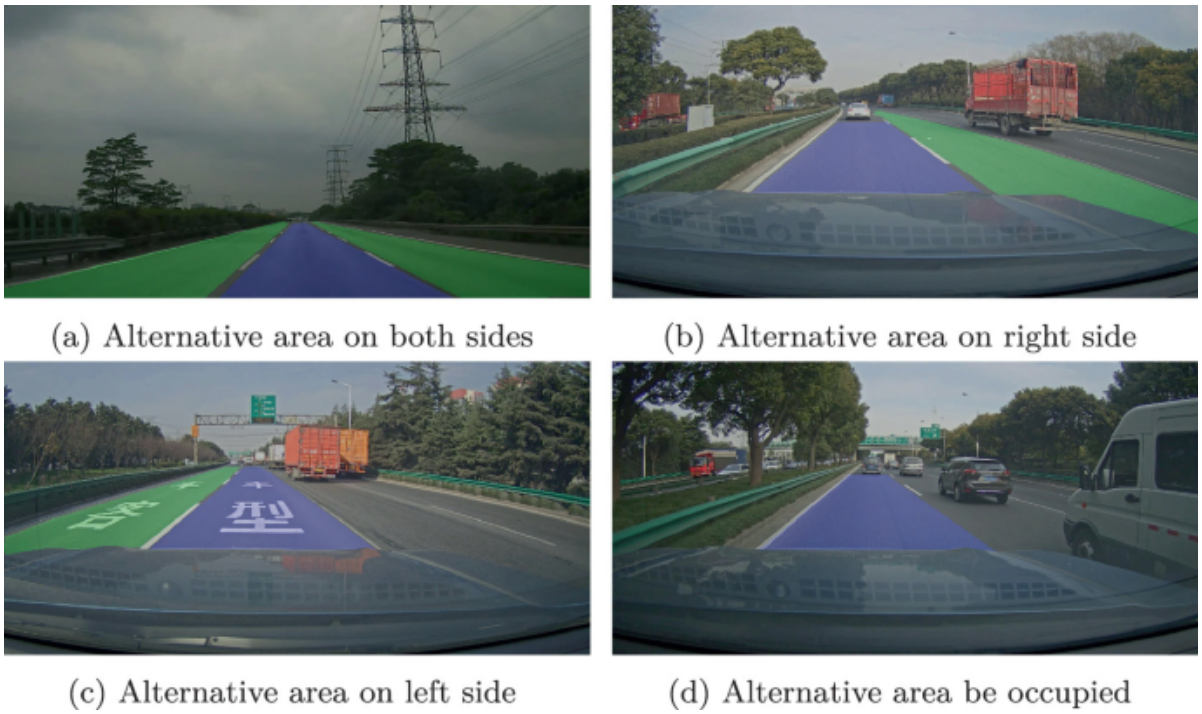
Figure 31: Sampling based motion planning. On the right, PRM builds a roadmap of random collision free configurations and connections. On the left, RRT grows a tree of feasible motions from the start toward the goal.

Sampling based planners are now standard tools in many robot arm libraries. They are especially useful when the robot has to work around complex obstacles, like reaching into a dishwasher or a storage bin, where narrow passages make uniform grids inefficient.

Learning based mapping and planning. Modern systems increasingly use learning not only to see the world, but also to decide where to go next. Instead of hand designed costmaps and manually tuned heuristics for lane changes or obstacle avoidance, a neural network can be trained on large amounts of driving or robot data to suggest which regions are safe to traverse and which maneuvers are likely to work well.

A common idea is to predict a *drivability* or *risk* map directly from sensor inputs. Given camera images or lidar scans, a network outputs a bird's eye view heatmap where bright regions are easy to drive and dark regions are dangerous or blocked. In effect, this is a learned costmap. The planner still searches over this map using classical tools like A* or sampling based planning, but the underlying costs now come from data rather than from hand written rules. Companies working on autonomous driving and mobile robotics use this pattern to capture human like preferences, such as staying centered in a lane, leaving comfortable gaps, or slowing down near crosswalks.

A more ambitious direction is end to end motion planning. Here, a single model takes in raw or lightly processed sensor streams and outputs a short horizon plan, for example a sequence of waypoints or a desired trajectory in front of the vehicle. Early work from NVIDIA showed that a convolutional network could map front camera images directly to steering commands for lane following. More recent systems at companies like Wayve and Waymo use large networks that output future paths for the car, which a lower level controller then tracks. Similar ideas appear in mobile manipulation, where a learned planner proposes arm motions to reach and grasp objects in cluttered scenes.



(a) Alternative area on both sides

(b) Alternative area on right side

(c) Alternative area on left side

(d) Alternative area be occupied

Figure 32: Example of a learned costmap. A neural network turns sensor data into a heatmap of drivability, and a classical planner searches on this map to find a safe and efficient trajectory.

Even with these powerful models, the overall structure of the mapping and planning stack has not disappeared. Learned components typically provide costs, proposals, or short horizon plans, while the surrounding system still maintains maps, enforces basic safety rules, and uses classical planners and controllers to ensure that the resulting motions are physically feasible and respect constraints.

In this subsection we have moved from local sensor snapshots to global maps, from maps to configuration spaces, and from configuration spaces to planned paths and short-horizon trajectories. Whether it is a mobile robot in a warehouse or an autonomous car in city traffic, the pattern is the same: perception builds a picture of the world, mapping turns that picture into a representation the planner can reason over, and planning proposes where the robot should go next while respecting obstacles and basic motion limits.

What is still missing is the final step: turning these planned motions into actual motor commands on real hardware, in the presence of delays, disturbances, and modelling errors. That is the role of the control and trajectory tracking layer. In the next subsection we will look at how classical feedback control laws take a desired motion and continuously adjust the robot's actuators so that its real trajectory stays close to the planned one.

4.3 Control and Trajectory Tracking

Mapping and planning answer the question "where should the robot go next?" Control and trajectory tracking answer the question "how do we actually make the real robot follow that plan?" Even with a perfect map and a beautiful planned trajectory, motors have limits, the floor can be slippery, wheels may slip, and sensors are noisy. The job of the control layer is to take a desired motion and continuously adjust the robot's actuators so that the real motion stays close to the plan.[1]

Why feedback is essential. A simple way to think about control is to imagine a robot arm joint that should hold a specific angle, like 30°. If we just send a fixed motor command and hope for the best, small disturbances (a person bumping the arm, a change in load) will push it away from the target. Instead, we measure the actual angle, compare it to the desired one, and correct the motor command based on the difference. This closed loop is called *feedback*.

In basic control language, we distinguish between:

- a **reference** or **setpoint** $r(t)$, which is what we want (for example, a desired joint angle or vehicle speed),
- the **output** $y(t)$, which is what we measure from the robot,
- the **error** $e(t) = r(t) - y(t)$,
- the **control input** $u(t)$, the signal we send to the actuator (for example, a torque command or throttle value).

The controller is a function that takes the error and possibly its history and produces $u(t)$. The robot and its mechanics are often called the *plant*. Together, they form a feedback loop.

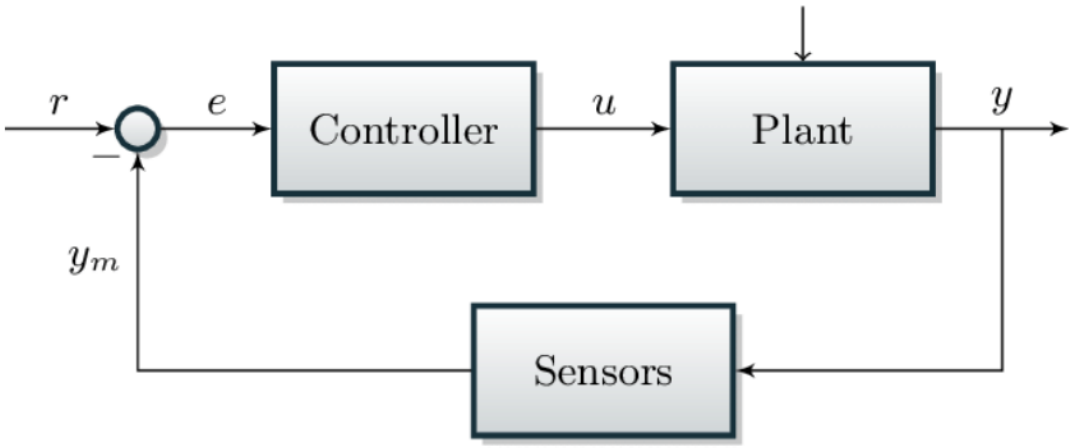


Figure 33: Basic feedback control loop. The controller compares the desired reference $r(t)$ with the measured output $y(t)$, computes the error, and generates a control input $u(t)$ that drives the plant (the robot). Disturbances and modelling errors are corrected by the loop.

This simple idea is deeply practical. A mobile robot uses feedback to keep its heading aligned with a corridor. An autonomous car uses feedback to keep its lateral position centered in a lane. A manipulator uses feedback to ensure its end effector reaches and holds a desired position in space.

Simple models for robots and vehicles. To design controllers, we often write down simple mathematical models of how the robot moves when we apply control inputs. These models are not perfect, but

they capture enough of the mechanics to be useful.

For a robot arm with joint angles $q = (q_1, \ldots, q_n)$, the forward kinematics map joint space to the position of the end effector in Cartesian space,

$$x = k(q),$$

where x might contain the three dimensional position and orientation of the hand. Differentiating with respect to time gives

$$\dot{x} = J(q)\,\dot{q},$$

where $J(q)$ is the Jacobian matrix. This linear relationship between joint velocities $\dot{q}$ and end effector velocities $\dot{x}$ is often used in velocity and force control of manipulators.[1]

For a car like robot, a common kinematic model is the bicycle model. We represent the vehicle's state by its position (X, Y) in the plane and its heading angle θ, and treat the control inputs as the forward speed v and steering angle δ:

$$\dot{X} = v\cos\theta, \qquad \dot{Y} = v\sin\theta, \qquad \dot{\theta} = \frac{v}{L}\tan\delta,$$

where L is the wheelbase. Although real cars have four wheels and complex dynamics, this simple model captures the basic fact that they cannot move sideways and turn by steering the front wheels.[4]

In general, we write the robot's dynamics in a compact form as

$$\dot{x}(t) = f(x(t), u(t)),$$

where $x(t)$ is the state (for example, positions and velocities) and $u(t)$ is the control input (for example, motor torques or steering and acceleration). The goal of control is to choose $u(t)$ so that $x(t)$ follows a desired reference trajectory $x_{\text{ref}}(t)$ produced by the planning layer.

PID control: a workhorse of industry. One of the most widely used controllers in practice is the proportional–integral–derivative (PID) controller. In its simplest form, for a single output $y(t)$ tracking a reference $r(t)$, it is defined by

$$u(t) = K_P e(t) + K_I \int_0^t e(\tau)\,d\tau + K_D \frac{de(t)}{dt},$$

where $e(t) = r(t) - y(t)$ is the error, and K_P, K_I, and K_D are gains chosen by the engineer.[1]

- The proportional term $K_P e(t)$ reacts to the current error and pulls the output toward the reference.
- The integral term $K_I \int e$ accumulates past error and helps eliminate steady state offsets (for example, a joint that always stops a bit short).
- The derivative term $K_D \frac{de}{dt}$ reacts to how quickly the error is changing and can damp oscillations.

PID controllers are used everywhere: to regulate joint positions in industrial arms, to maintain a fixed speed in cruise control, to keep quadrotors level in the air, and to control the longitudinal motion (speed) of autonomous vehicles. Modern hardware often runs many PID loops in parallel at high frequency, sitting underneath more advanced planning and control modules.

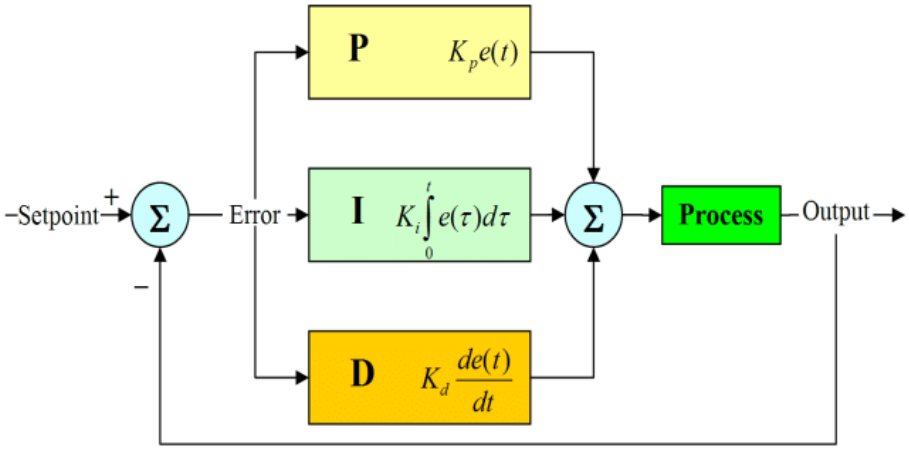


Figure 34: PID controller structure. The proportional, integral, and derivative paths all act on the error $e(t)$ and are combined to produce the control input $u(t)$. Tuning the gains K_P, K_I, and K_D changes the response of the closed loop system.

State space feedback and lane keeping. For more complex robots and vehicles, it is often useful to work with a state space model that is linear (or locally linear) around an operating point,

$$\dot{x}(t) = Ax(t) + Bu(t),$$

where A and B are matrices that capture how the state and inputs interact. In this setting, a common control law is *state feedback*,

$$u(t) = u_{\text{ref}}(t) - K\big(x(t) - x_{\text{ref}}(t)\big),$$

where $x_{\text{ref}}(t)$ and $u_{\text{ref}}(t)$ are the desired state and input from the planner or trajectory generator, and K is a gain matrix chosen to give good tracking performance.

Linear quadratic regulator (LQR) design provides a systematic way to compute K by minimizing a quadratic cost function that penalizes deviations from the reference and large control efforts.[1] In practice, LQR style controllers are used in many balancing and stabilization problems, such as keeping a humanoid robot upright or stabilizing the lateral motion of a car around a lane keeping trajectory.

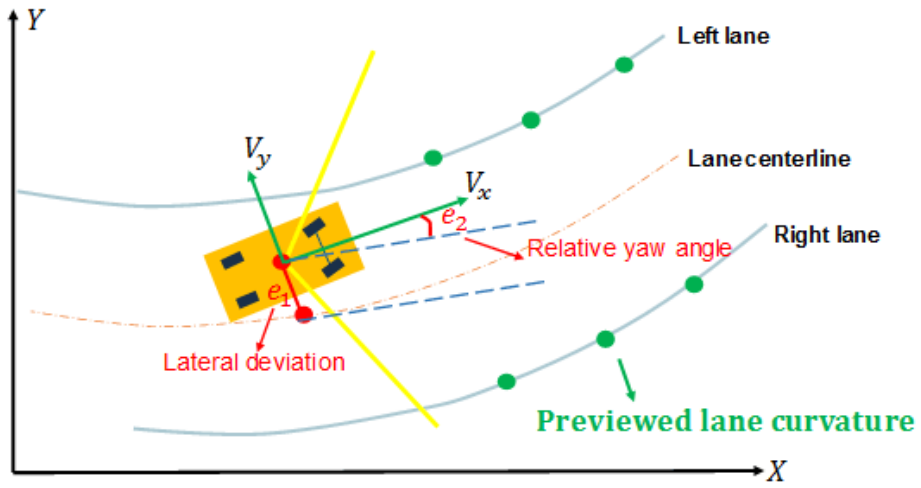


Figure 35: Lateral control for lane keeping. The planned trajectory (centerline) is provided by the planner. A state feedback controller uses the lateral error and heading error to keep the actual vehicle path close to the reference.

Model predictive control for constrained systems. While PID and linear state feedback work well in many cases, they do not directly handle hard constraints such as actuator limits, obstacle avoidance, or strict speed bounds. Model predictive control (MPC) addresses this by explicitly planning a short horizon control sequence at each time step.[4]

At a high level, MPC works as follows:

- Predict how the system will evolve over the next few seconds using a model of the dynamics.
- Optimize a sequence of future inputs to keep the predicted trajectory close to the desired one while satisfying constraints (for example, bounds on steering angle, acceleration, or lateral deviation from a lane).
- Apply only the first input in this optimal sequence to the real robot.
- At the next time step, repeat the process with updated state information.

Because it explicitly reasons about the future and constraints, MPC has become a popular choice for trajectory tracking in autonomous driving, where the vehicle must follow a planned trajectory, stay within lane boundaries, and obey comfort and safety limits. Variants of MPC are also used in legged robots to coordinate foot placement and body motion over uneven terrain.

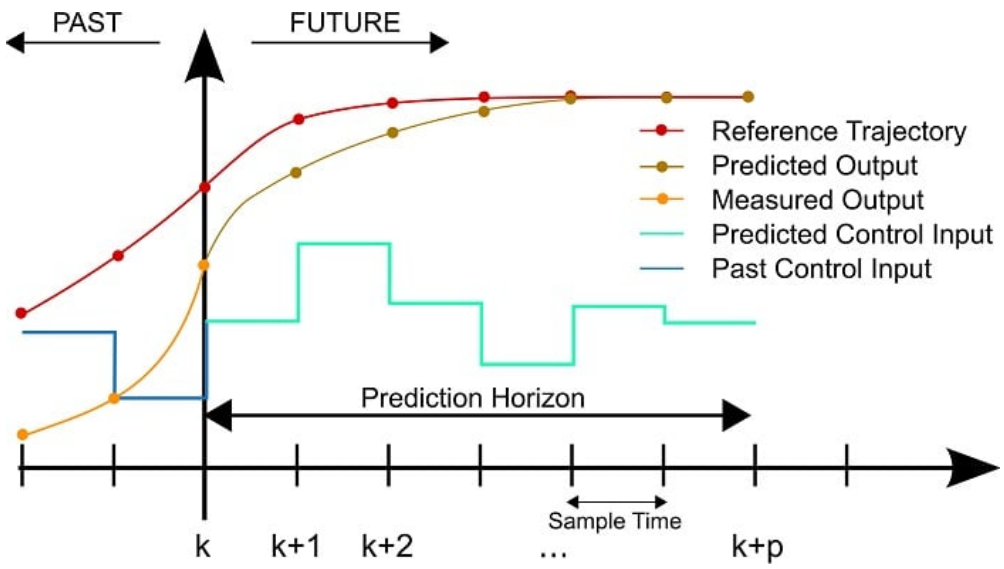


Figure 36: Model Predictive Control (MPC) trajectory tracking.

Safety layers and overrides. Real robots and vehicles must also deal with unexpected events: a pedestrian stepping into the road, a human entering the manipulator workspace, or a sensor glitch. Many systems therefore include additional safety layers that sit alongside or above the main controller.

These layers may:

- monitor distances to obstacles and override commanded accelerations with emergency braking when thresholds are violated,
- enforce speed limits in certain zones (for example, near humans or in narrow aisles),
- restrict joint motions to safe regions when the robot is in a teaching or demonstration mode.

Bringing perception, planning, and control together. We can now see how the three classical pillars interact in a complete system. Perception and state estimation produce a stream of state estimates and maps, with uncertainties. Mapping and planning reason over those estimates to propose paths and short horizon trajectories that move the robot toward its goals while avoiding obstacles. Control and trajectory tracking take those trajectories and turn them into low level motor commands, correcting for disturbances and model errors through feedback.

In modern learning enabled robots and autonomous vehicles, many components of this pipeline are influenced by machine learning, from perception networks to learned cost functions and policies. However, the underlying structure remains close to what we have described here: a loop that senses, plans, and acts, with classical control at the bottom ensuring that the physical robot does what the higher layers intend. In the next sections we build on this foundation and introduce reinforcement learning, which offers a complementary way to synthesize decision making and control policies directly from data.

5 Reinforcement Learning in Robotics: A Unified Overview

The previous section showed how modern robots and autonomous vehicles are built from classical components: perception and state estimation to understand the world, mapping and planning to decide where to go, and control and trajectory tracking to turn plans into motor commands. In many deployed systems, each of these layers is still largely hand designed. Engineers choose state representations, cost functions, planners, and controllers, and then spend a great deal of time tuning parameters by trial and error.

Reinforcement learning (RL) offers a complementary approach. Instead of specifying every rule by hand, we let an agent learn a *policy*, a way of choosing actions by interacting with an environment and receiving feedback in the form of rewards. In a robotic context, the hope is that a robot can learn useful behaviors (such as walking, grasping, or manipulating) from experience, either in simulation or in the real world, rather than having every detail engineered ahead of time.

This section introduces the core ideas of reinforcement learning from the perspective of robotics and autonomous driving. The goal is to provide enough intuition and basic notation to support the more focused sections that follow on manipulation, driving.

5.1 Foundations

The agent–environment loop. In reinforcement learning, we imagine a loop between an *agent* and an *environment*. At each time step t,

- the agent observes some information s_t about the current situation,
- chooses an action a_t,
- the environment responds with a new observation or state s_{t+1} and a scalar *reward* r_{t+1}.

The agent's goal is to choose actions over time so as to maximize the total reward it receives. For a robot arm, the state s_t might include joint angles, joint velocities, and the pose of objects on the table; the action a_t might be desired joint torques or velocities. For an autonomous car, s_t might contain the vehicle pose, speed, and a compact representation of nearby lanes and traffic participants; a_t might be steering and acceleration commands. The reward could encourage progress toward a goal, penalize collisions, and discourage uncomfortable motions.

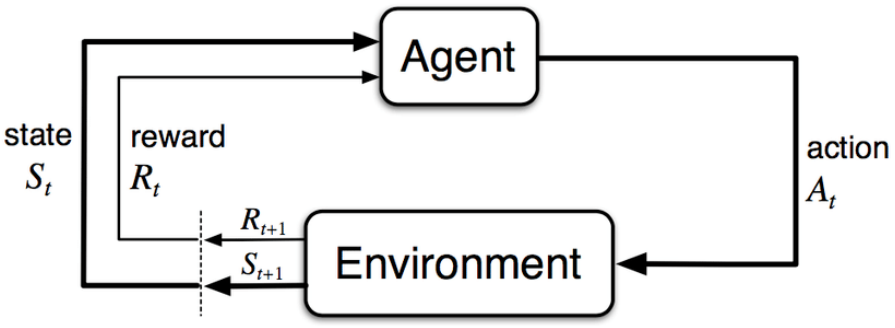


Figure 37: Reinforcement learning as an agent–environment loop. A robotic agent observes a state, chooses an action, and receives the next state and a scalar reward. Over many episodes, it adjusts its policy to choose actions that lead to higher cumulative reward.

From a control perspective, this looks similar to the feedback loop from Section 4.3. The important difference is that in RL we typically do not assume an exact model of the dynamics or a fixed cost function.

Instead, the agent treats both the dynamics and the reward as something it can probe by acting and listening to the feedback it receives.

Markov decision processes. The standard mathematical model for reinforcement learning problems is the *Markov decision process* (MDP). An MDP is defined by a set of states $\mathcal{S}$, a set of actions $\mathcal{A}$, transition probabilities $P(s' \mid s, a)$, a reward function $R(s, a, s')$, and a discount factor $\gamma \in [0, 1)$ that balances present and future rewards.[43]

In practice, roboticists rarely write down all of these components explicitly. Instead, the MDP serves as a mental model. If the agent visits states and actions

$$s_0, a_0, s_1, a_1, s_2, a_2, \ldots$$

and receives rewards $r_1, r_2, r_3, \ldots$, then the *return* from time t is the discounted sum

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots.$$

The objective in most RL problems is to choose actions so that the expected return is as large as possible.

For the rest of this report, it will be enough to think in informal terms: the state is whatever information the agent uses to decide what to do; the action is its choice; the reward is the feedback signal; and the return is the total future feedback it cares about.

Policies and value functions. A *policy* tells the agent how to act. In the simplest case, a policy is a function

$$a_t = \pi(s_t),$$

that maps the current state to an action. In more general settings, the policy can be stochastic, choosing actions according to a probability distribution $\pi(a \mid s)$, or depend on the whole observation history if the environment is only partially observable.

Given a policy, we can ask how good it is. One way to measure this is with *value functions*. The state value function $V^\pi(s)$ is the expected return when we start in state s and then follow policy π forever,

$$V^\pi(s) = \mathbb{E}\big[G_t \mid s_t = s, \pi\big].$$

The action value function $Q^\pi(s, a)$ is similar but assumes we start by taking action a in state s and then follow π afterwards,

$$Q^\pi(s, a) = \mathbb{E}\big[G_t \mid s_t = s, a_t = a, \pi\big].$$

Intuitively, $V^\pi(s)$ answers the question "how good is it to be in this state if I behave like this policy," and $Q^\pi(s, a)$ answers "how good is it to take this particular action here if I then continue with this policy." Many reinforcement learning algorithms are built around estimating these value functions from data and then improving the policy to favor actions with higher estimated value.[43]

Tabular RL and its limitations. In small, abstract problems, such as a robot moving on a grid world with a few dozen cells, we can represent $V^\pi(s)$ or $Q^\pi(s, a)$ as tables and update them directly from experience. Algorithms like temporal difference (TD) learning and Q learning adjust these values

whenever a new transition $(s_t, a_t, r_{t+1}, s_{t+1})$ is observed, nudging the estimates toward consistency with the observed rewards and the values at successor states.[43]

This tabular setting is extremely useful for building intuition, and many textbooks use it to introduce RL. However, it is far from what we need for real robots and autonomous cars. In manipulation and driving, the state includes continuous quantities like positions, velocities, and distances, and the action space is often continuous as well (torques, forces, steering angles). The state space is effectively infinite, and the action space can be high dimensional. In these settings, tables are not enough; we need function approximators that can generalize across states.

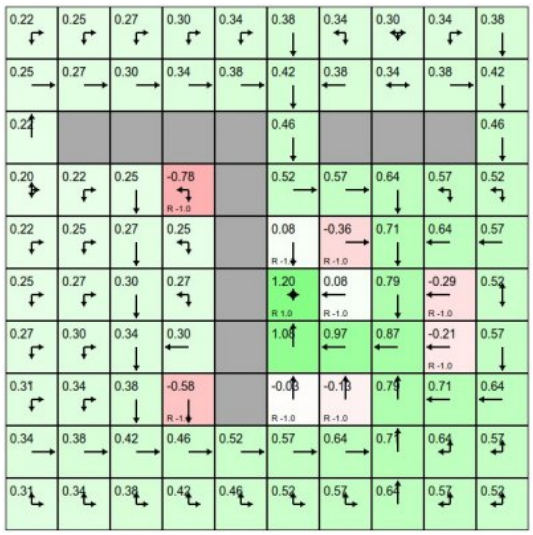


Figure 38: Conceptual picture of tabular reinforcement learning. In a small gridworld environment, the value of each state or state action pair can be stored and updated in a table. Real robotic systems require continuous state and action spaces and therefore use function approximators instead of tables.

5.2 Modern Deep RL

Function approximation with neural networks. Deep reinforcement learning combines classical RL ideas with neural networks as function approximators. Instead of storing values or policies in tables, we represent them with a network that takes in a state (and sometimes an action) and outputs, for example, a value estimate or the parameters of an action distribution.[44, 45] In robotics this is essential, because even a simple arm or mobile robot already has a continuous, high dimensional state space that cannot be covered by a table.

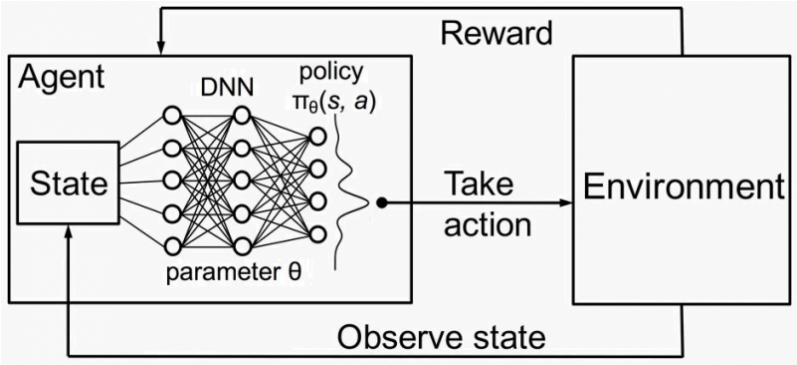


Figure 39: Deep reinforcement learning for continuous control. A policy network takes in the state of a simulated robot and outputs continuous action commands. Instead of storing state-action pairs we approximate using a neural network. After training in simulation, the learned policy can sometimes be transferred to real hardware with additional adaptation.

Key ingredients in practice. Modern deep RL algorithms share a few building blocks that are worth knowing at a high level:

- **Experience replay.** Instead of learning only from the most recent transition, the agent stores past experience $(s_t, a_t, r_{t+1}, s_{t+1})$ in a replay buffer and trains by sampling mini batches from this memory. This breaks up correlations between successive steps and stabilizes gradient based learning.[44]
- **Target networks and bootstrapping.** When learning value functions, the same network is used both to make predictions and to construct training targets. To avoid chasing a moving target, many algorithms keep a slowly updated copy of the network (a *target network*) and use it to compute the next state value in the learning rule.
- **Stochastic policies.** For continuous control, policies usually output the parameters of a probability distribution (for example, a mean and standard deviation of a Gaussian) rather than a single deterministic action. This makes it easy to explore by sampling actions and to compute gradients that tell the policy how to shift probability mass toward better actions.[46, 47]

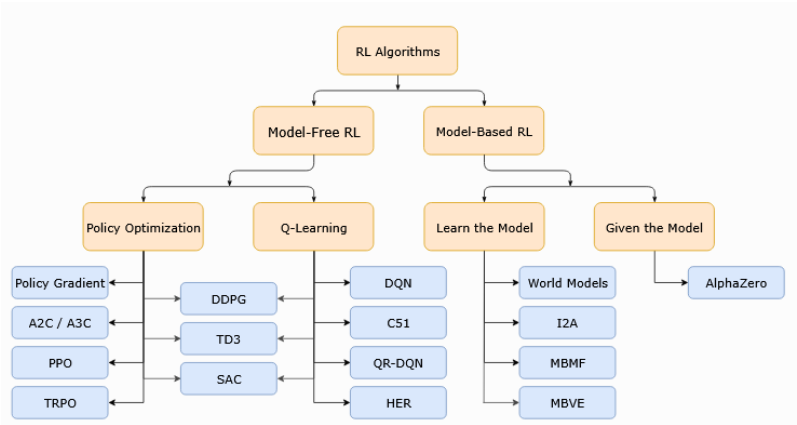


Figure 40: High level hierarchy of reinforcement learning algorithms. At the top level we distinguish between model free and model based methods. Model free methods can be further divided into value based (Q learning) and policy optimization approaches, while model based methods either learn a model of the dynamics or assume a known model and plan with it. In this report we focus mainly on the model free side, which is currently most common in robotics.

These categories show up under slightly different names in different papers, but they give a useful mental map. For robotics, most practical deep RL systems today lie in the model free branch, because building accurate world models for contact rich manipulation or complex driving scenes is still difficult. Within model free methods, three families appear over and over again, and it is helpful to recognize them.

Value based, policy optimization, and actor–critic methods. Conceptually, most model free deep RL algorithms fall into three broad families:

- **Value based (Q learning) methods** such as DQN approximate the action value function $Q(s, a)$ and choose the action with the highest estimated value. These methods are a natural extension of tabular Q learning to deep networks and work well when the action space is small and discrete.[44]
- **Policy optimization methods** directly optimize a parameterized policy $\pi_\theta(a \mid s)$, adjusting the parameters θ in the direction that increases expected return. These methods handle continuous actions more naturally, which is important for robots, but they can suffer from high variance in their gradient estimates.
- **Actor–critic methods** combine both ideas. An *actor* network represents the policy, while a *critic* network estimates value functions and provides a smoother learning signal to guide the actor.[45, 46, 47] Many of the most popular continuous control algorithms for robotics, such as DDPG, PPO, and SAC, fall into this category.

In robotics, actor–critic methods are especially popular because they support continuous torques and velocities and can reuse experience more efficiently than purely policy based methods.

Atari, Go, and continuous control benchmarks. In discrete action domains such as Atari games, the Deep Q Network (DQN) pioneered by Mnih et al. uses a convolutional neural network to map stacks of pixels to estimated Q values for each possible joystick action, and then chooses the action with the highest value.[44] In robotics, we often have continuous actions, so policy gradient and actor–critic methods are more common. Algorithms such as DDPG, TD3, PPO, and SAC learn stochastic policies $\pi_\theta(a \mid s)$ whose parameters θ are updated to increase expected return, while a critic network learns value estimates that guide the updates.[45, 46, 47]

A famous example outside of robotics is the game of Go. DeepMind's AlphaGo and AlphaGo Zero combined deep neural networks with tree search: a *policy network* suggested promising moves, a *value network* estimated the strength of a position, and Monte Carlo tree search (MCTS) used these networks to explore possible future move sequences more efficiently.[48, 49] The networks themselves were trained with a mix of supervised learning from expert games and reinforcement learning from self play, showing how the same basic RL ideas can reach superhuman performance in a very different domain.

Continuous control benchmarks like MuJoCo's HalfCheetah, Hopper, and Humanoid popularized this style of learning, where a simulated robot learns to run, hop, or balance purely from a reward signal that encourages forward motion and penalizes large control efforts.[50] The same techniques can be applied to more realistic robot models in physics engines such as MuJoCo, Isaac Gym, or Bullet, and later transferred to hardware.

These three examples have been extensively covered in previous surveys, and include highly complex architecture. For brevity, they will remain as illustrative examples and will not be covered in detail in this report.

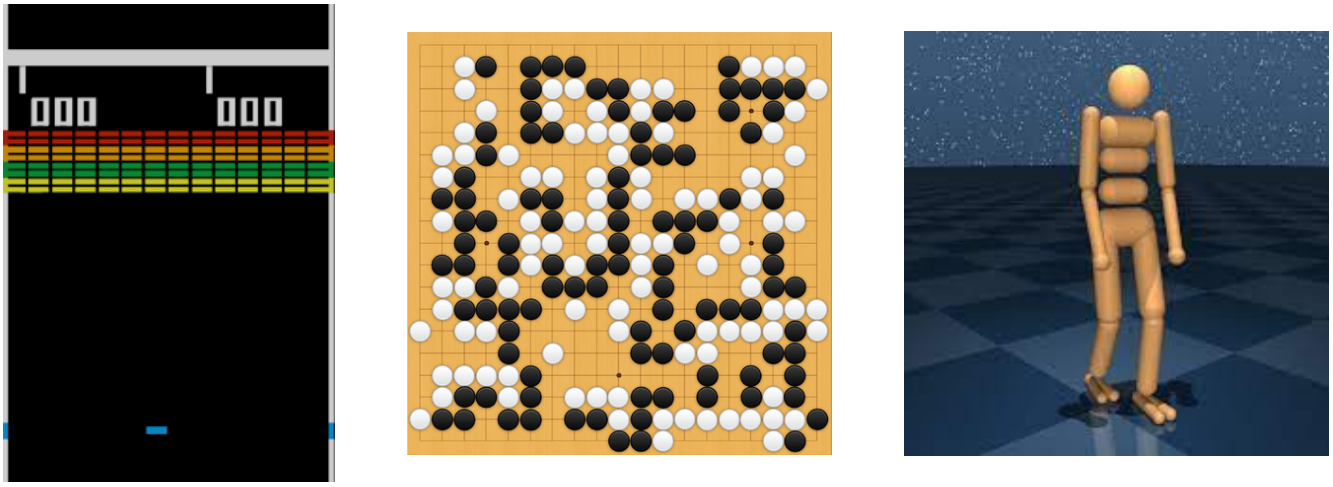


Figure 41: Representative reinforcement learning domains. Left, an Atari game such as Breakout where Deep Q Networks learn from pixels and a small discrete action set. Middle, the game of Go where AlphaGo style systems combine deep networks with tree search. Right, a continuous control benchmark such as a simulated humanoid, where policy gradient and actor–critic methods learn to control robot like bodies.

5.3 Real-World Successes in Robotic RL

Training in simulation and sim–to–real transfer. Most modern RL for robotics is first trained in simulation. Simulators provide fast, safe, and repeatable environments in which a robot can experience millions of time steps without wearing out hardware or risking damage. Physics engines approximate rigid body dynamics, contacts, and friction, and can model sensors such as cameras and lidar with varying degrees of realism.[50]

The challenge is that simulated physics and sensors never match reality perfectly. Policies that perform well in simulation can fail on the real robot if, for example, friction, motor delays, or object masses are even slightly different. This gap between simulated and real dynamics is known as the *sim to real* problem.

Common strategies to reduce this gap include:

- **Domain randomization**, where simulation parameters (masses, friction, textures, lighting) are randomized during training so that the policy learns to be robust to variations.[51]
- **System identification**, where the simulator is tuned to match measured responses from the real robot as closely as possible before training.
- **Fine tuning on hardware**, where a policy pre trained in simulation is further adapted using a small amount of real world data.

These techniques have enabled several impressive demonstrations, such as quadruped robots learning agile locomotion in simulation and then transferring those skills to hardware, and manipulators trained in simulation to perform grasping or insertion tasks that work with minimal additional training in the real world.

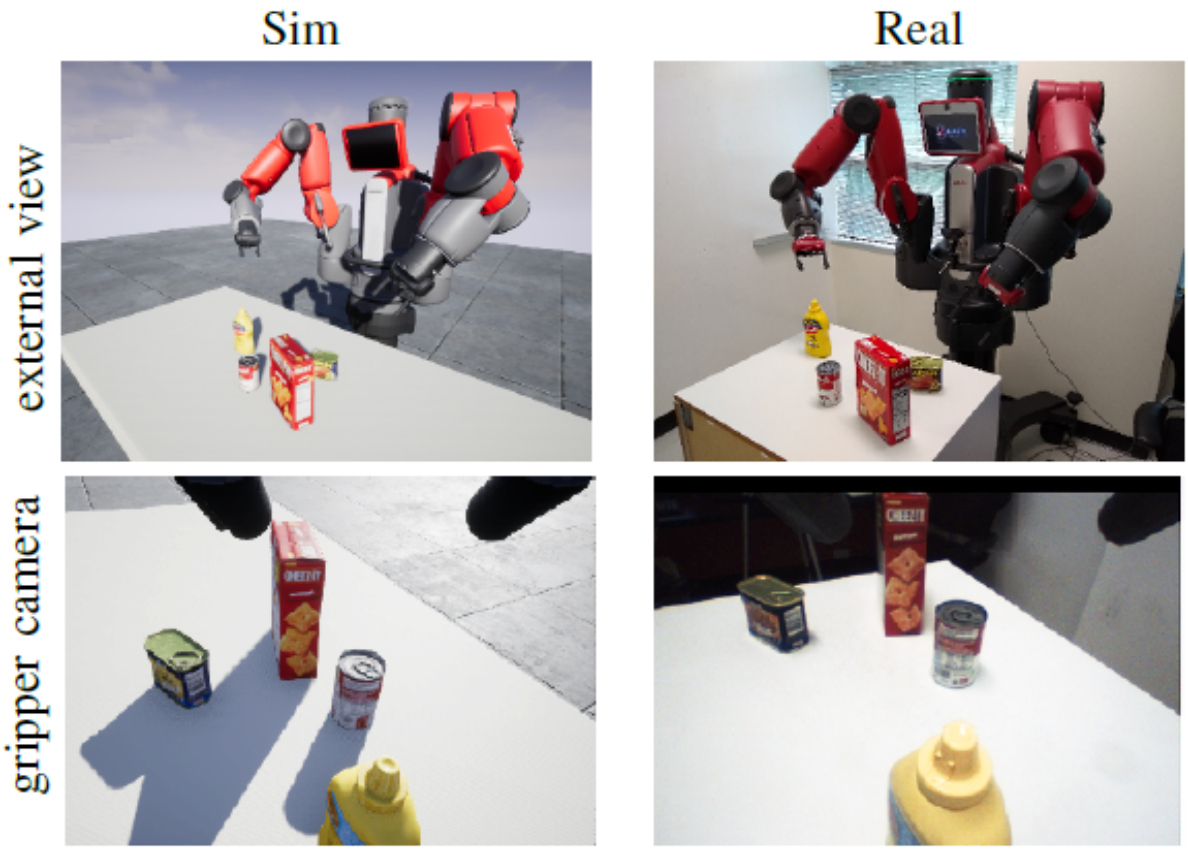


Figure 42: Typical sim–to–real workflow. A policy is trained in a physics simulator with randomized dynamics and visual conditions, then deployed on the real robot. Additional fine tuning or adaptation closes the remaining gap between simulated and real performance.

Learning from demonstrations and large datasets. Pure trial and error RL can be very sample inefficient, especially on real hardware. One way to accelerate learning is to start from demonstrations of good behavior. In *behavior cloning*, the agent learns a policy that imitates expert trajectories, treating them as supervised training data. This can be followed by reinforcement learning that further improves the policy.[52]

Recent large scale efforts in robot manipulation follow this pattern. Google Robotics, for example, collected hundreds of thousands of grasp attempts and used them to train QT Opt, a deep RL system for vision based grasping.[53] More recent projects such as RT 1, RT 2, and the Open X Embodiment effort combine data from many robots and tasks to train models that can generalize across diverse manipulation skills.[54, 55, 56] Similar ideas appear in autonomous driving, where fleets of vehicles collect logs of human driving that can be used for imitation learning, offline RL, or a mix of both.

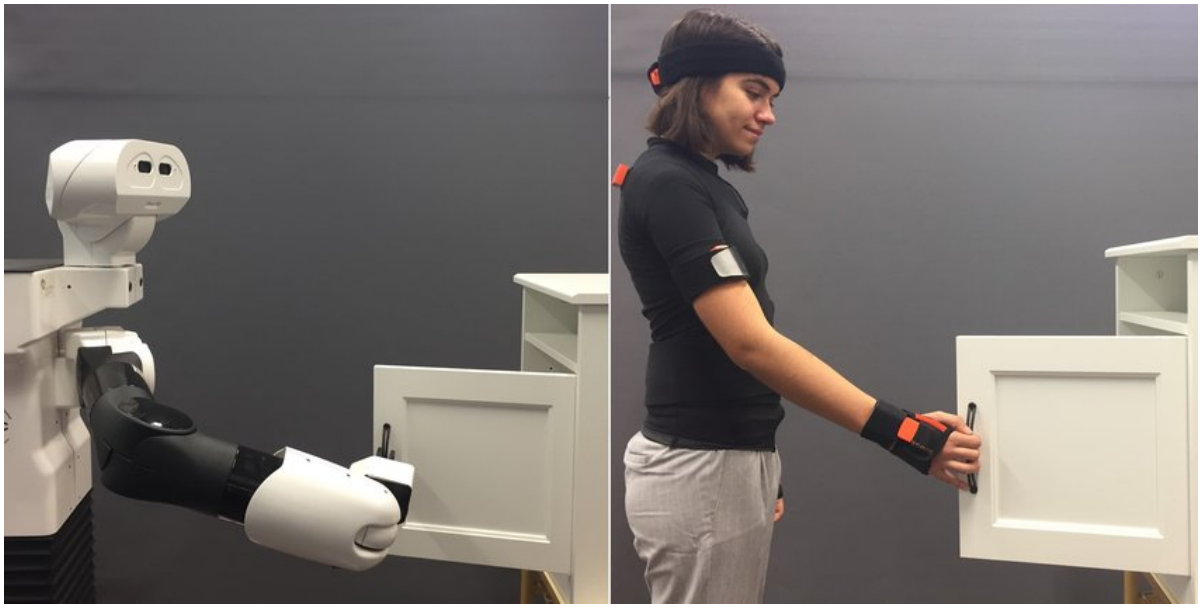


Figure 43: Learning from demonstrations. Human teleoperation or scripted policies generate example trajectories. These are used to train an initial policy by imitation, which is then refined with reinforcement learning or offline RL methods.

5.4 Challenges and Outlook

Despite substantial progress, deploying reinforcement learning on real robots and autonomous vehicles remains challenging. Some of the main difficulties are:

- **Sample efficiency.** Many RL algorithms require millions of environment steps to learn good policies. In simulation this is acceptable; on real hardware it can be prohibitively slow or damaging.
- **Exploration and safety.** Naive exploration strategies that try random actions can easily lead to collisions or unstable behavior. Safe exploration is still an active research area, particularly in driving.
- **Reward design.** It is often hard to translate the true task objective (for example, "drive naturally and safely to the destination") into a simple scalar reward. Poorly designed rewards can encourage undesirable behavior or "reward hacking," where the agent exploits loopholes in the reward instead of solving the intended task.
- **Partial observability and non stationarity.** Real environments are noisy and complex. Other agents (cars, pedestrians, humans) have their own goals and can change their behavior, making the dynamics non stationary from the learning agent's point of view.
- **Generalization.** A policy that works in one building, one home, or one city might fail in another. Achieving robust generalization across environments and tasks is a central goal for modern robot learning.

These challenges have led many researchers and companies to adopt hybrid approaches, where reinforcement learning is used for parts of the problem (such as locomotion skills, grasping strategies, or short horizon driving decisions), while classical mapping, planning, and control provide structure and safety guarantees.

6 Reinforcement Learning for Robotic Manipulation

Robotic manipulation is where reinforcement learning starts to look very concrete. Instead of an abstract agent in a gridworld or a simulated cheetah running in an empty plane, we now have a real arm that needs to reach, grasp, open doors, load dishwashers, and rearrange objects on cluttered tables. These tasks are central to the long term goal of useful household and service robots, but they are also surprisingly hard to hand engineer.

Classical pipelines separate perception, planning, and control. Engineers design grasp planners, collision free trajectories, and controllers by hand, then tune dozens of parameters so that the arm behaves well on a fixed set of objects. Reinforcement learning offers a complementary view: instead of scripting every behavior, we ask the robot to learn its own strategies for reaching, grasping, pushing, and inserting by interacting with the environment and receiving feedback.

This section focuses on how reinforcement learning is used for manipulation, how it differs from the toy examples in the previous section, and what current large scale systems look like in both research and industry.

6.1 Problem setting and task families

In a typical manipulation task, the robot observes the world through one or more cameras and proprioceptive sensors. The state s_t might include joint angles and velocities, end effector pose, and one or more RGB or depth images. The action a_t is often a low level command such as joint torques or velocities, or a slightly higher level command such as desired end effector motion.

Common task families include:

- **Reaching and positioning.** Move the gripper to a desired pose near an object or target point.
- **Grasping and picking.** Close the gripper on an object so that it is stably grasped, often from cluttered piles of diverse shapes.
- **Placing and rearranging.** Move grasped objects to specified locations, sort them into bins, or arrange them according to patterns.
- **Tool use and articulated objects.** Open drawers and doors, pour from containers, or operate switches and faucets.

The reward is designed to encourage success on these tasks. For grasping, it might be $+1$ if the robot lifts an object above the table and 0 otherwise. For rearrangement, it might depend on how close each object ends up to its goal position, with penalties for collisions or dropping.

Even in these simple examples, the state is continuous and high dimensional, the action space is continuous, and contact dynamics are complex and often discontinuous. This is why the deep RL methods from the previous section, especially policy optimization and actor–critic methods, are more common than tabular algorithms in manipulation.

6.2 Learning setups and policy representations

Low level versus structured actions. One design choice is the *action representation*. Some works let the policy output low level joint commands at high frequency. This gives the agent full control over the arm but makes the learning problem harder, because the policy must implicitly handle stability, smoothness, and safety.

Other works use slightly higher level actions, such as short end effector motions in Cartesian space, or discrete "skills" like *move above object*, *close gripper*, and *lift*. In that case, reinforcement learning

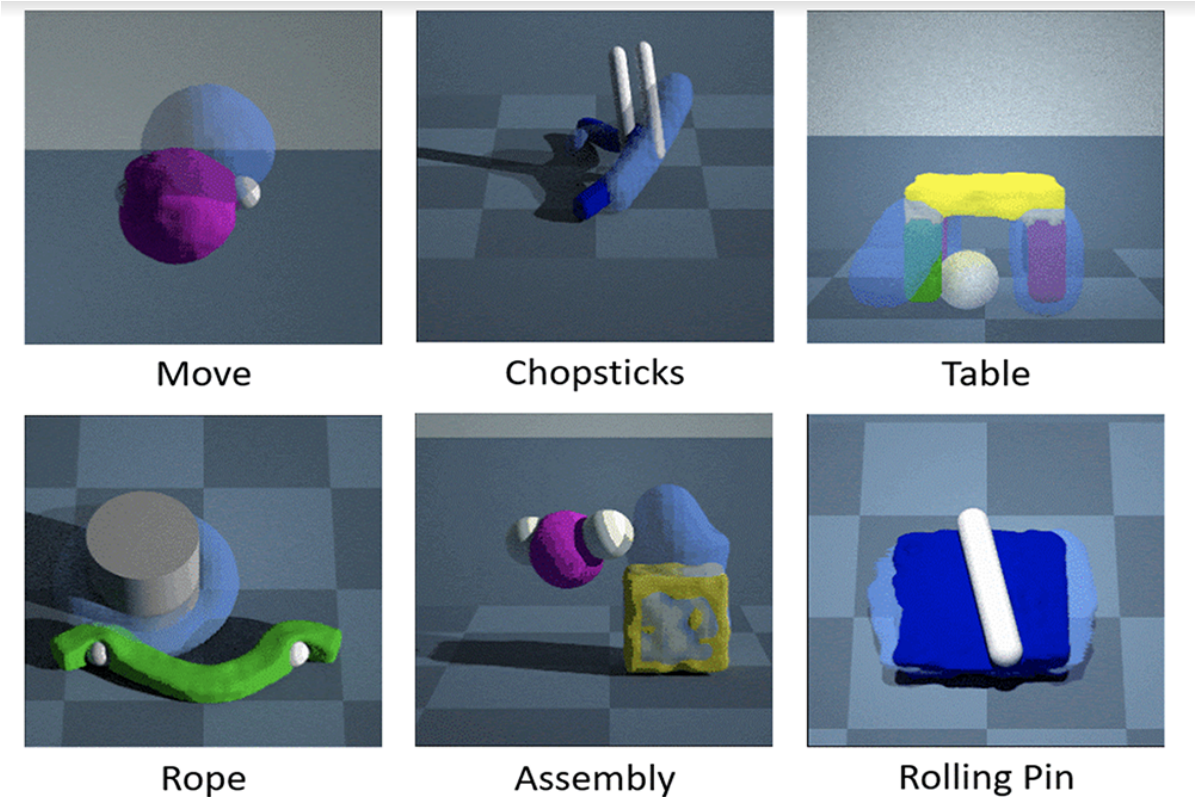


Figure 44: Examples of robotic manipulation tasks: moving an object, using chopsticks, tabletop manipulation, rope manipulation, assembly tasks, and using a rolling pin.

operates on top of a small library of motion primitives, combining them into task level solutions.

Simulation, real robots, and sim–to–real. Most manipulation RL systems are trained at least partly in simulation. Physics engines such as MuJoCo, Isaac Gym, and Bullet simulate rigid body dynamics, contacts, and friction, and can provide synthetic images for vision based policies.[50] Training in simulation allows millions of interaction steps without wearing out hardware.

For real deployment, the learned policy must transfer to a physical robot. As discussed in Section 5, this sim–to–real transfer is usually handled by a mix of domain randomization (varying object masses, friction, and visual conditions during training), careful system identification, and fine tuning on hardware.

Learning from demonstrations. Pure trial and error is often too slow or unsafe for real manipulation tasks. An important trick is to start from demonstrations. In *behavior cloning*, the robot is given trajectories from an expert (for example, a human teleoperating the arm, or a scripted controller), and it trains a policy to mimic the demonstrated actions given the observed states.[52] This is standard supervised learning.

Once this imitation policy is reasonably good, reinforcement learning can continue from there, using the reward signal to further improve performance and adapt to new situations. This combination of demonstration data and RL is common in modern robot learning and is sometimes called learning from demonstration, imitation learning, or apprenticeship learning.[52, 57]

6.3 Large scale systems and datasets

Recent years have seen several striking large scale manipulation systems that combine deep RL, large datasets, and strong industrial hardware.

Vision based grasping with QT Opt. One influential example is QT Opt, a scalable deep reinforcement learning framework for vision based grasping developed at Google Brain and X.[53] Using data from over 580,000 real world grasp attempts collected by seven robot arms, QT Opt learns a deep Q function that maps camera images and gripper states to a value for each candidate motion. The resulting policy achieves around 96% grasp success on previously unseen objects.[53, 58]

Beyond the raw numbers, QT Opt showed that deep RL could learn rich closed loop behaviors: probing piles of objects, regrasping when an initial attempt was unstable, and reacting to disturbances during the lift.[53] This is difficult to achieve with open loop grasp planners that commit to a single grasp pose.

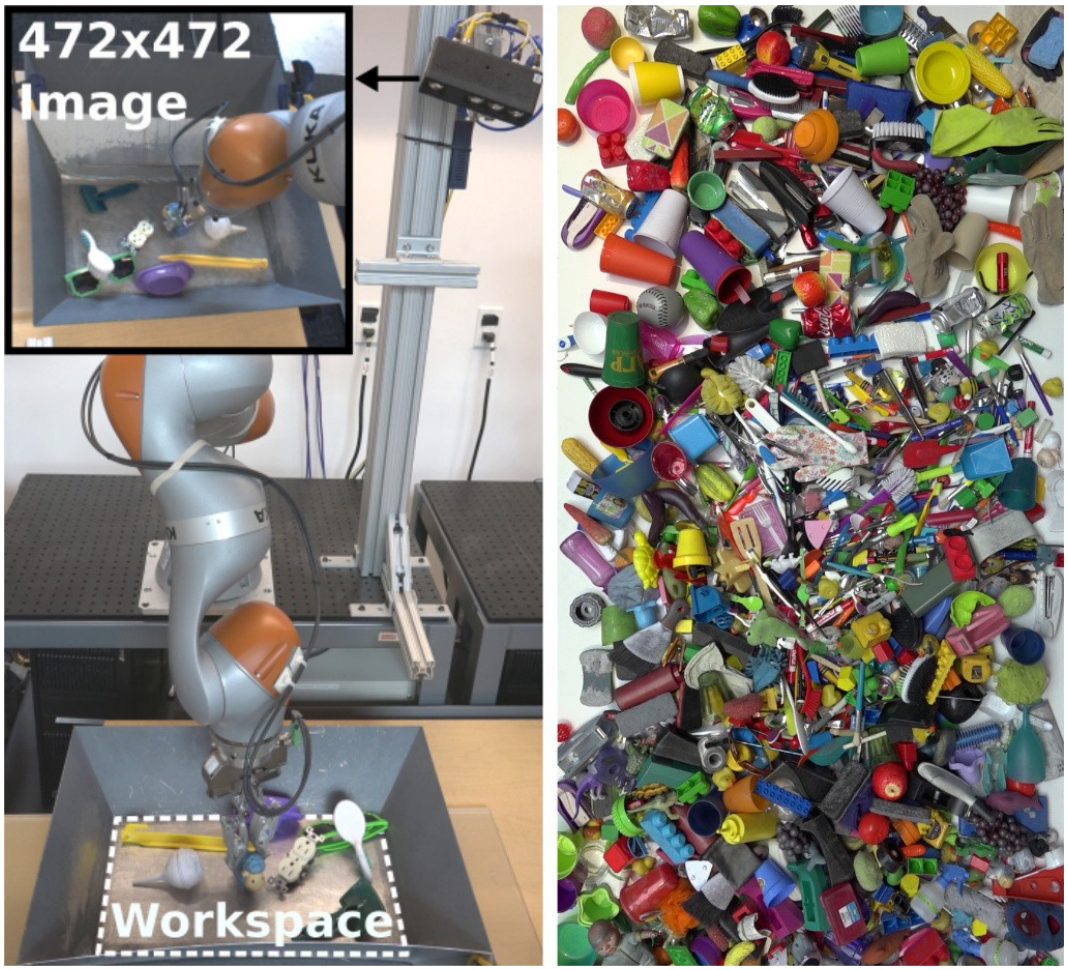


Figure 45: QT Opt style grasping system. Multiple camera views of a bin are used as input to a learned Q function. The robot executes continuous, closed loop grasping motions that are adapted based on visual feedback.

Multi task manipulation with Robotics Transformers. QT Opt focuses on a single family of tasks (grasping) with a relatively small action space. More recent work from Google DeepMind and collaborators takes a broader view. The RT 1 Robotics Transformer is a large model trained on hundreds of thousands

of real robot trajectories from Everyday Robots platforms performing more than 700 different tasks, such as picking, placing, opening doors, and operating appliances.[54, 59] RT 1 tokenizes images, language instructions, and actions, then uses a transformer architecture to predict the next action token.[54, 59]

Follow up work such as RT 2 and the Open X Embodiment project scales this idea further. RT 2 connects vision language models to robotics, allowing zero shot generalization to novel instructions like "throw away the yellow bag" after training mostly on web scale vision language data.[55] Open X Embodiment aggregates more than one million real robot trajectories from 22 different robot embodiments (single arms, bimanual systems, mobile manipulators, quadrupeds, and more) and trains RT X models that can transfer skills across hardware platforms.[56, 60]

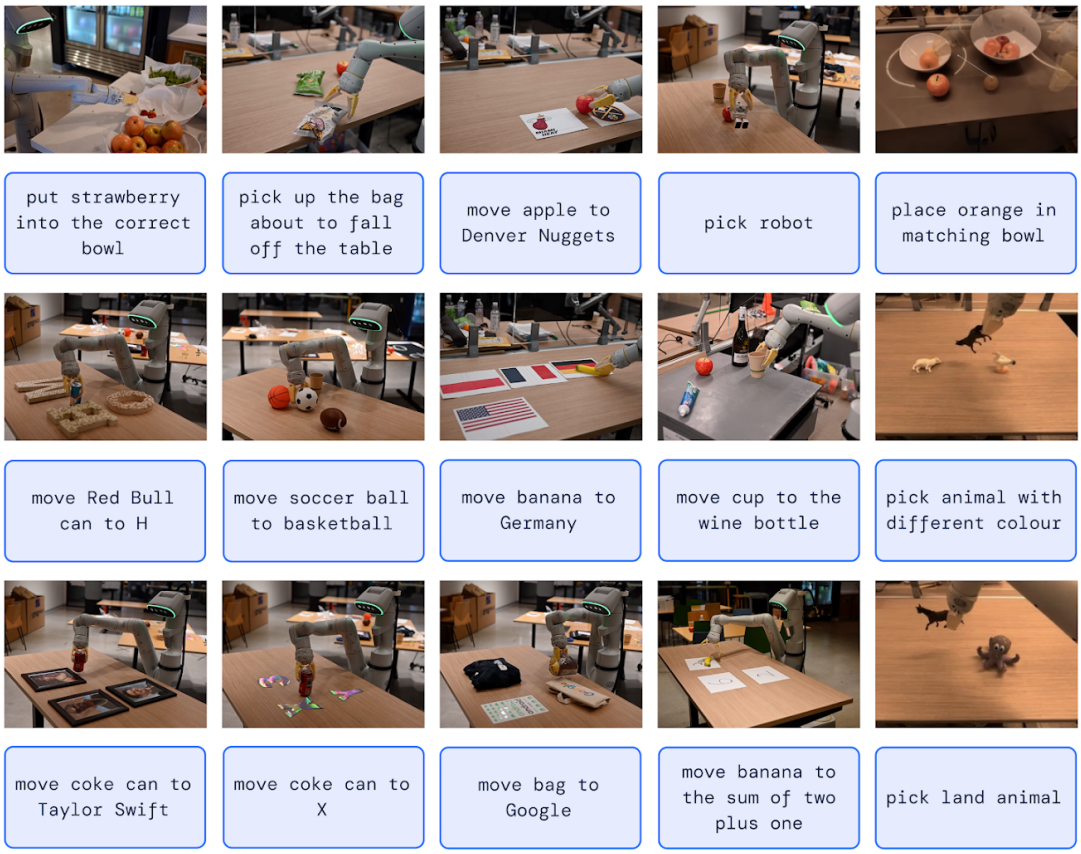


Figure 46: Large scale multi task manipulation. Robotics Transformer models such as RT 1 and RT 2 are trained on diverse real robot datasets to perform many everyday tasks from vision and language instructions.

These systems blur the line between "pure" reinforcement learning and other forms of learning. They typically use a mixture of behavior cloning on logged data, offline RL, and online fine tuning, but the underlying idea remains the same: learn general purpose policies that map rich sensory inputs and instructions to useful manipulation behaviors.

Other notable examples. OpenAI's Dactyl system demonstrated that a dexterous hand could learn to manipulate and solve a Rubik's cube primarily via reinforcement learning in simulation, combined with heavy domain randomization before deployment to real hardware.[61] Industrial companies such

as Covariant and Intrinsic (a Google affiliated robotics company) have explored learned pick and place policies and skill libraries for warehouses and factories, although many details remain proprietary.

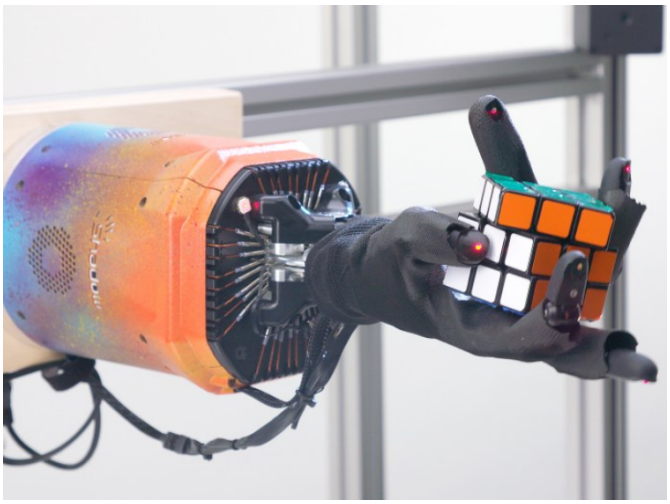


Figure 47: OpenAI's Dactyl model manipulating a Rubik's Cube.

6.4 Open challenges and role of classical structure

Despite these successes, reinforcement learning for manipulation faces many of the challenges outlined in Section 5:

- Collecting large amounts of real world interaction data is expensive. Systems like QT Opt and RT 1 rely on fleets of robots operating for long periods.[53, 54]
- Safety and wear and tear limit naive exploration on physical hardware. Many promising ideas still depend heavily on simulation and carefully curated datasets.
- Generalization to new objects, tools, and layouts remains an open problem, even for large models trained on millions of examples.[54, 56]

For these reasons, most practical manipulation systems today are hybrids. Classical perception, mapping, and control provide structure, safety, and guarantees. Reinforcement learning and related techniques are used to learn pieces that are difficult to program by hand, such as grasp scoring, contact rich skills, or high level action selection given complex visual scenes.

In the next section, we turn from manipulation on tables and workspaces to decision making on roads. We will see how similar reinforcement learning ideas can be applied to autonomous driving, and how they interact with prediction, planning, and rule based modules in that domain.

7 Reinforcement Learning for Autonomous Driving

Autonomous driving is another natural application area for reinforcement learning, but it has a very different flavor from tabletop manipulation. Instead of a single arm working in a small workspace, we now have a vehicle moving at high speed among many other agents, under strict traffic rules and safety constraints. Decisions span longer time horizons, and failures can have serious consequences.

As in the classical foundations section, most deployed self driving systems today are still built around a modular stack with perception, mapping, prediction, planning, and control.[4, 62] Reinforcement learning is usually introduced in specific parts of this stack, especially short horizon behavior and negotiation with other traffic participants, rather than as a single end to end controller that replaces everything.

7.1 Driving as a sequential decision problem

State and actions in road environments. From a reinforcement learning point of view, we can think of an autonomous car as an agent that repeatedly observes a compact description of the traffic scene and outputs driving decisions. A typical driving state might include:

- the ego vehicle pose, velocity, and acceleration relative to the road,
- a local lane graph, including lane centers, boundaries, and speed limits,
- estimates of nearby vehicles, cyclists, and pedestrians, with their positions and velocities,
- traffic light and stop sign states,
- sometimes compressed features from camera or lidar networks, such as bird's eye view (BEV) feature maps.

The action space depends on how we choose to insert RL into the stack:

- **Low level control** actions are direct steering, throttle, and brake commands at high frequency.
- **Trajectory level** actions select one of many candidate trajectories in front of the vehicle, such as a lane keeping trajectory, a lane change, or a braking maneuver.
- **Maneuver level** actions choose among high level options like keep lane, change to left lane, change to right lane, stop at crosswalk, or yield.

Most practical work uses RL at the maneuver or trajectory level. A downstream motion planner and controller then refine the choice into a smooth, dynamically feasible trajectory. Directly learning raw steering and throttle from pixels is an active research direction but remains rare in deployed systems because of safety and interpretability concerns.[62]

Rewards that capture safe and comfortable driving. A driving reward has to encode several objectives at once:

- **Safety:** strong penalties for collisions, near misses, and clearly unsafe maneuvers.
- **Progress:** positive reward for moving toward the goal and maintaining reasonable speed when safe.
- **Comfort:** penalties for harsh acceleration, braking, and jerk, to avoid uncomfortable or unstable driving.
- **Rule obedience:** penalties for running red lights, stopping in crosswalks, or driving on the wrong side of the road.

Balancing these terms is delicate. If progress is weighted too heavily, the agent may drive aggressively. If safety penalties dominate, it may become overcautious and refuse to move. Designing robust reward functions is therefore a central challenge for RL in driving.

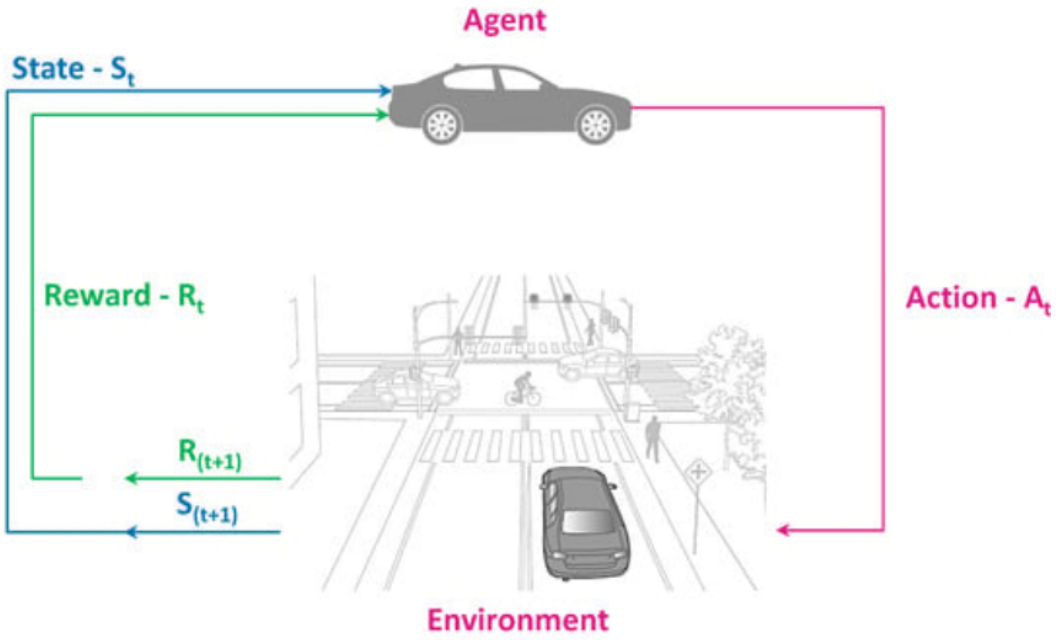


Figure 48: Example driving scenario for reinforcement learning. The ego vehicle must decide whether to keep lane, change lanes, merge, or slow down based on nearby traffic, lane geometry, and traffic rules.

7.2 Where RL fits in the driving stack

Behavior planning and negotiation. A typical autonomous driving stack separates the problem into perception, prediction, behavior planning, motion planning, and control.[4] Reinforcement learning is most often explored in the behavior planning layer, where the system decides what to do at a tactical level:

- when to change lanes on a highway,
- how to merge in front of or behind another car,
- how to handle a four way stop or an unprotected left turn,
- when to yield to pedestrians or cyclists.

In these settings, RL policies usually operate on a structured state that includes lanes and nearby agents and output maneuver choices or short horizon trajectories. A downstream motion planner then refines the chosen maneuver into a smooth path that satisfies vehicle dynamics and hard safety constraints.

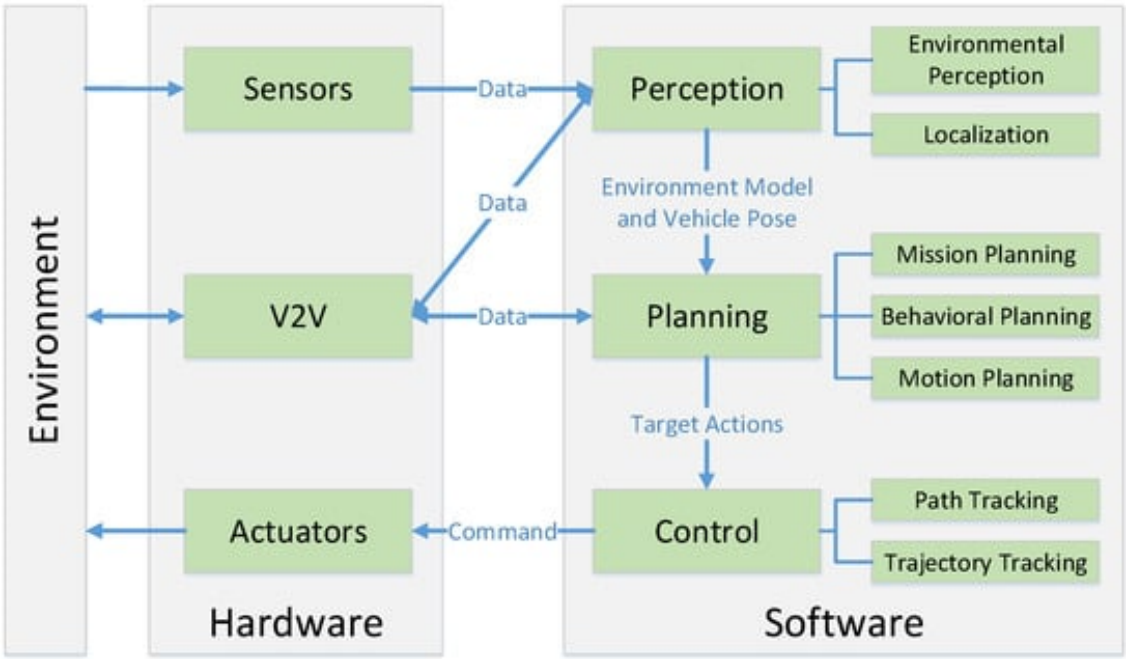


Figure 49: Illustrative autonomous driving stack. Perception and prediction estimate the state of the world and other agents. A behavior layer, which can include reinforcement learning components, chooses maneuvers or short horizon trajectories. A motion planner and controller track the chosen trajectory under strict safety constraints.

Multi agent interactions. Driving is inherently multi agent. The outcome of a lane change, merge, or left turn depends on how other drivers react. To capture this, some research uses multi agent RL formulations, in which several vehicles are modeled as learning agents that interact in a shared environment.[62] Even when deployed systems do not assume that other drivers are using RL, this viewpoint is still helpful. It encourages explicit modeling of interaction patterns such as:

- cooperative behavior, for example another car slowing down to let the ego vehicle merge,
- adversarial or unexpected behavior, such as sudden braking or illegal lane changes,
- social norms, such as leaving gaps, not cutting in aggressively, and respecting informal right of way.

Some approaches train RL policies against a mixture of heuristic and learned models of other drivers so that the learned policy can handle a variety of behaviors and still behave conservatively when uncertain.

7.3 Training signals and data sources

Simulation based training. Direct online RL on real roads is not practical. Unsafe exploration cannot be allowed, and even mild mistakes are unacceptable. Instead, most RL for driving relies on simulators. These range from simplified highway environments with a few vehicles to high fidelity platforms that model detailed city layouts, realistic sensors, and traffic agents with human like behavior.

RL policies can be trained in these simulators to perform tasks such as highway merging, lane changing, intersection handling, or roundabout navigation. The main advantages are:

- the ability to explore unsafe or rare scenarios without risk,
- the possibility to run many parallel simulations for faster training,
- fine grained control over environment variations, such as traffic density and weather.

The main drawback is the sim to real gap. Human drivers behave differently from simple traffic models, pedestrians may act unpredictably, and real world road networks are more varied than most simulators capture. Bridging this gap is an active research area.[62]

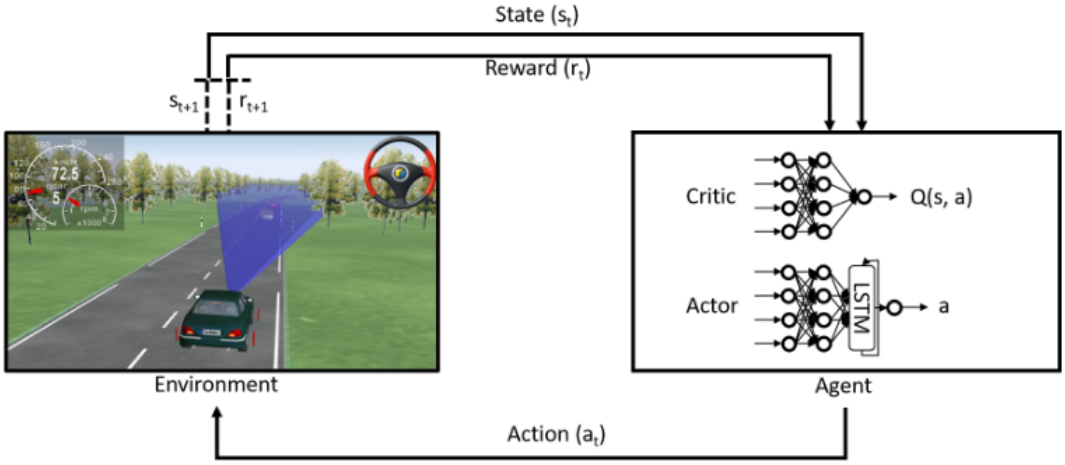


Figure 50: Simulation based RL for driving. A virtual environment provides traffic scenes and sensor inputs. The RL policy learns maneuvers such as merging, lane changing, and intersection handling before any deployment on real roads.

Learning from logged driving data and offline RL. Commercial autonomous driving programs and advanced driver assistance systems collect large amounts of logged human driving data. Each log contains sensor streams, estimated maps and tracks, and the control commands issued by the human driver or the onboard controller. This opens the door to imitation learning and offline RL:

- **Imitation learning** trains a policy to mimic human driving in common situations, for example predicting steering and acceleration from a local scene representation.
- **Offline reinforcement learning** uses reward style objectives on the same logs to improve behavior without online exploration, for example by emphasizing safety, comfort, or rule compliance.[63]

In practice, many state of the art systems use a mixture of both. Imitation learning provides a strong baseline that behaves roughly like a human driver, while RL style objectives fine tune the policy to avoid rare failures, respect stricter safety margins, or improve ride comfort.

End to end visuomotor policies. A more radical line of work trains policies that map directly from raw sensor inputs, such as front facing camera images, to control commands. These end to end driving policies are often trained mainly by imitation, with RL style objectives added as a fine tuning signal on specific scenarios such as lane changes or merges.[62]

Architecturally, these policies resemble the vision based manipulation policies from Section 6. A deep visual encoder processes images into a latent representation, which is combined with vehicle state and map features and fed into a policy head that outputs steering and acceleration commands.

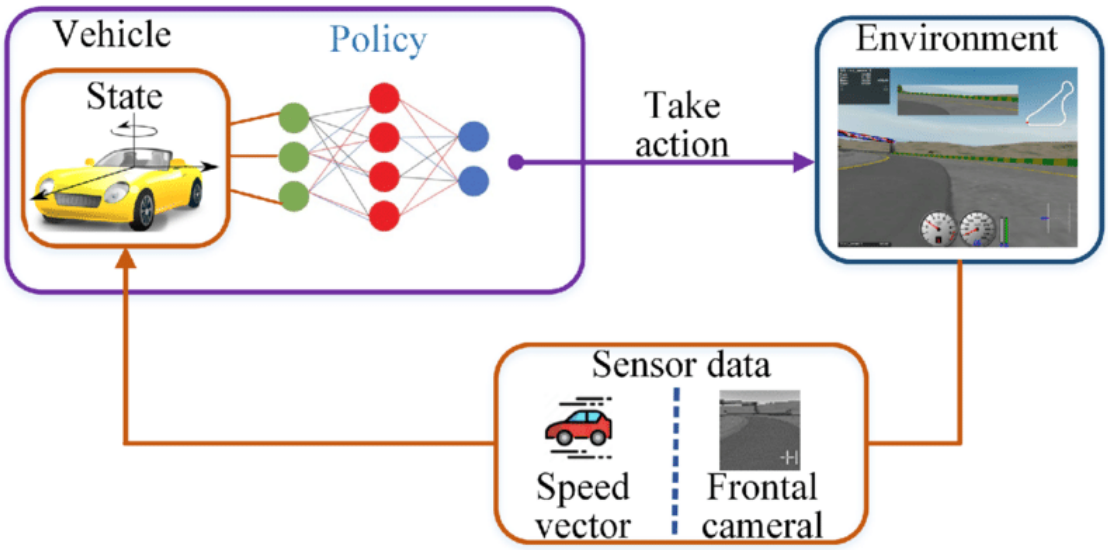


Figure 51: End to end driving policy. A visual encoder processes camera images, map features, and ego state. A policy head outputs steering and acceleration commands. Reinforcement learning style losses can be combined with imitation losses to tune behavior.

Industry example: Tesla's data driven FSD. Tesla's Full Self Driving (FSD) system is marketed as an advanced driver-assistance feature rather than a fully autonomous product, but its training pipeline is a useful illustration of large-scale, data-driven policy learning in industry. Tesla vehicles form a global "data engine": they collect real-world driving data from the fleet, including difficult scenarios and driver interventions, which are then uploaded, filtered, auto-labeled, and used for training. Updated models are sent back to vehicles through over-the-air software updates.

As drivers operate their cars with or without FSD engaged, the system logs how humans actually drove in each situation, along with the behavior proposed by the onboard planner and moments where the human overrode automation. This provides the company with trajectories that reflect "good" driving under many conditions, plus a concentrated set of edge cases where the current policy failed and the driver had to intervene. Tesla then uses this curated dataset to train large neural networks that form the backbone of the FSD system. This feedback loop is powerful: the policy can be improved continuously based on real-world experience from a very large fleet.

Public descriptions of FSD training emphasize two main ingredients:

- **Imitation learning.** Large neural networks are trained to mimic human driving in ordinary situations, learning lane keeping, car following, and basic maneuvers from vast collections of expert trajectories. Recent versions (for example FSD v12) are described as end-to-end imitation models that map short histories of multi-camera video directly to steering, acceleration, and braking commands, using human drivers as the reference behavior.
- **Fleet-scale offline optimization.** On top of simple behavior cloning losses, Tesla mines rare or problematic cases from the fleet (such as disengagements, strong interventions, or near misses) and uses them to refine the policy. Conceptually, these additional objectives act like "reward-like" penalties for unsafe, uncomfortable, or rule-violating behavior and bonuses for smooth, correct behavior, but they are implemented as supervised or contrastive losses on logged data rather than as online reinforcement learning on public roads.

Architecturally, Tesla has described a camera-only perception stack (*Tesla Vision*) based on occupancy networks: neural networks that take synchronized multi-camera video and predict a 3D occupancy

field around the car, indicating which regions of space are free, occupied, or uncertain. On top of this representation, a learned planner network proposes short-horizon trajectories in a bird's-eye-view space, and lower-level control optimizes the execution of the chosen trajectory subject to vehicle dynamics and comfort constraints. Recent FSD versions move toward a more end-to-end style in which perception and planning blocks are trained jointly in a single, unified network, while still maintaining internal structure such as an intermediate vector-space or occupancy representation.

From a reinforcement learning perspective, this pipeline looks like a very large-scale offline learning system built on logged experience. Human driving logs implicitly encode high-reward behavior (for example, collision-free, rule-respecting trajectories), and the fleet provides a continuous stream of new data and edge cases. Rather than running online RL directly in the real world, Tesla combines imitation learning with RL-inspired objectives on logged and simulated data to iteratively refine the driving policy while keeping training offline.

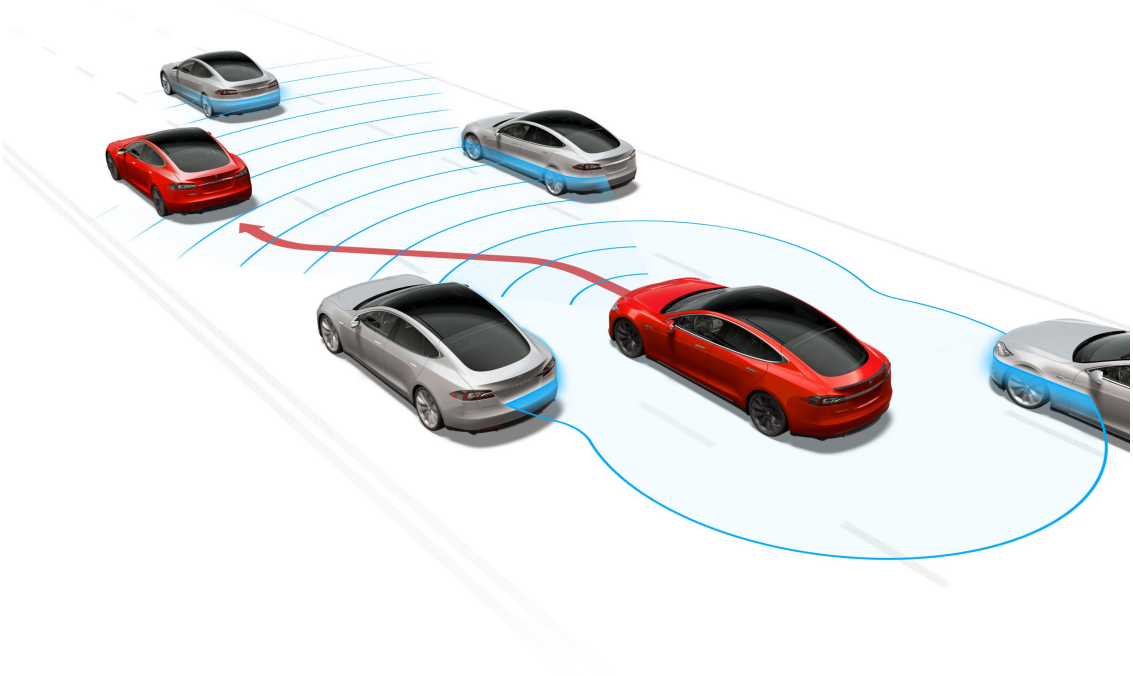


Figure 52: Tesla cars driving on the road, using perception to be aware of one another, and planning and control to avoid unwanted behavior.

7.4 Hybrid approaches and open challenges

Safety shields and rule based guards. Because driving is safety critical, purely learned policies are almost always combined with rule based safeguards. Examples include:

- collision checkers that prevent trajectories with predicted collisions from being executed,
- emergency braking controllers that override the learned policy if a collision is imminent,
- rule checkers that veto maneuvers violating traffic laws, such as entering an intersection on red or driving against traffic.

From an RL perspective, these mechanisms act as a safety shield around the learned policy. The policy suggests actions, but the shield filters them and may replace them with safe alternatives when necessary.

Distribution shift and rare events. A recurrent challenge in driving is distribution shift. Policies are trained on common situations but must cope with rare, unusual configurations, such as:

- a pedestrian suddenly emerging from behind a parked truck,
- an emergency vehicle approaching at high speed,
- unusual road layouts, temporary construction, or missing markings.

These rare events have a disproportionate impact on safety but are hard to cover in training data. Simulation can help by generating synthetic edge cases, and RL style objectives can be used to stress test policies against adversarial scenarios, but reliably handling all rare events remains an open problem.[62]

Role of RL in future driving systems. In summary, reinforcement learning for autonomous driving is currently used in a targeted way:

- to tune behavior and negotiation in specific scenarios such as merging, lane changes, and unprotected turns,
- to learn from large offline datasets of human driving, improving over simple imitation while avoiding unsafe online exploration,
- to explore end to end policies in research, while production systems remain more modular.

The classical mapping, prediction, planning, and control foundations from earlier sections remain central. RL is layered on top as a tool for refining behavior where hand crafted rules are brittle or incomplete. This mirrors the broader theme of this report: modern autonomous driving systems, including those built by industry leaders such as Tesla, remain hybrids in which classical structure provides the backbone and data driven learning, including reinforcement learning, fills in the parts that are hardest to design by hand.

8 Conclusion

This report has tried to tell one coherent story: how robots and autonomous vehicles sense the world, plan and control their actions, and increasingly learn behaviors from data and experience. We began with the historical arc from ancient automata and early industrial manipulators to modern warehouse fleets and self driving cars. We then opened the black box of deployed systems, where perception, mapping, planning, and control form a modular sense–plan–act loop. On top of this classical backbone, machine learning and reinforcement learning offer a way to acquire skills from interaction and data rather than hand crafting every rule.

Across the examples, one pattern stands out: today's systems are almost always *hybrid*. Perception stacks mix geometry and deep networks; planning blends search, optimization, and sometimes short horizon learned policies; control layers still rely heavily on classical feedback designs. Reinforcement learning appears in roles where it has a clear advantage, such as continuous control skills, grasping strategies, or local driving decisions, while classical components provide structure, safety, and interpretability.[3, 1, 4, 43]

Although robot arms and cars look very different, they share the same basic problem structure. Both must estimate their state under noisy sensors, maintain maps or priors over their surroundings, plan collision free motions under dynamics and safety constraints, and execute those motions reliably. Manipulators do this in joint and task spaces over relatively small workspaces; vehicles do it over long horizons with traffic rules and human drivers in the loop.[1, 4] This shared structure explains why ideas transfer between domains: SLAM and tracking support both indoor robots and highway driving; sampling based planners serve both arms and mobile bases; algorithms like PPO or SAC can control quadrupeds, dexterous hands, or lane change maneuvers with appropriate state and action design.[47, 46, 50]

Modern learning methods already help in several concrete ways. Vision models such as SAM style segmenters or language grounded detectors provide rich scene understanding that would be very hard to engineer by hand, but still feed into classical estimation and mapping back ends. RL and imitation learning are particularly effective for low level skills that are hard to specify analytically, such as agile locomotion on varied terrain or contact rich manipulation where friction, forces, and compliance matter.[53, 54, 55] In both manipulation and driving, fleets of robots or vehicles generate large logs of human behavior and controller outputs; offline and imitation learning methods can turn these logs into policies that generalize across many scenarios.[56, 62]

Looking ahead, it seems unlikely that purely classical systems or purely learned systems will dominate. Instead, the trend points toward increasingly integrated stacks in which perception and world models are largely learned but expose geometrically meaningful state estimates and uncertainties; planners exploit structure from dynamics and maps while using learned components for prediction and short horizon decisions; and control layers remain grounded in physics and safety analysis while being tuned and adapted with data. As datasets, simulators, and algorithms improve, we can expect robots and autonomous vehicles that learn faster, generalize more reliably, and behave more naturally around people. The examples in this report are modest steps in that direction, but they show how the classical and learning based pieces can already be combined on real hardware today.

References

[1] B. Siciliano and O. Khatib, Eds., *Springer Handbook of Robotics*, 2nd ed. Springer, 2016.

[2] J. M. Anderson, N. Kalra, J. Stanley *et al.*, "Autonomous vehicle technology: A guide for policymakers," RAND Corporation, Tech. Rep., 2014.

[3] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. Cambridge, MA: MIT Press, 2005.

[4] B. Paden, M. Çakmak, 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.

[5] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[6] A. Geiger, P. Lenz, and R. Urtasun, "Vision meets robotics: The KITTI dataset," *International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013.

[7] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

[8] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *Journal of Machine Learning Research*, vol. 17, no. 39, pp. 1–40, 2016.

[9] M. B. et al., "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016. [Online]. Available: https://arxiv.org/abs/1604.07316

[10] A. El Sallab, M. Abdou, E. Perot, and S. Yogamani, "Deep reinforcement learning framework for autonomous driving," *arXiv preprint arXiv:1704.02532*, 2017.

[11] A. Mayor, *Gods and Robots: Myths, Machines, and Ancient Dreams of Technology*. Princeton, NJ: Princeton University Press, 2018.

[12] G. Scholem, *On the Kabbalah and Its Symbolism*. New York: Schocken Books, 1996, essay "The Idea of the Golem".

[13] A. C. Graham, *The Book of Lieh-tzu: A Classic of the Tao*. New York: Columbia University Press, 1990.

[14] K. Čapek, *R.U.R. (Rossum's Universal Robots)*. London: Penguin Classics, 2004, originally published in Czech in 1920.

[15] I. Asimov, *I, Robot*. New York: Gnome Press, 1950.

[16] ——, "Runaround," *Astounding Science-Fiction*, vol. 29, no. 1, Mar. 1942.

[17] N. Wiener, *Cybernetics: Or Control and Communication in the Animal and the Machine*. Cambridge, MA: MIT Press, 1948.

[18] W. G. Walter, "An imitation of life," *Scientific American*, vol. 182, no. 5, pp. 42–45, 1950.

[19] G. C. Devol, "Programmed article transfer," US Patent US2 988 237A, Jun. 13, 1961, filed December 10, 1954.

[20] N. J. Nilsson, "Shakey the robot," Artificial Intelligence Center, SRI International, Menlo Park, CA, Tech. Rep. Technical Note 323, 1984.

[21] iRobot Corporation, "The hottest gift item this year is for grown ups; irobot's ..." Press release, Nov. 2002.

[22] P. R. Wurman, R. D'Andrea, and M. Mountz, "Coordinating hundreds of cooperative, autonomous vehicles in warehouses," *AI Magazine*, vol. 29, no. 1, pp. 9–19, 2008.

[23] H. A. Pierson and M. A. Gashler, "Deep learning in robotics: A review of recent research," *arXiv preprint*, 2017.

[24] Leonardo da Vinci, "Self-propelled cart," https://www.da-vinci-inventions.com/self-propelled-cart.aspx, c. 1478, accessed: 2025-12-04.

[25] Mobileye, "A brief history of autonomous vehicles – from renaissance to reality," https://www.mobileye.com/blog/history-autonomous-vehicles-renaissance-to-reality/, 2023, accessed: 2025-12-04.

[26] J. M. Anderson and et al., *Autonomous Vehicle Technology: A Guide for Policymakers*. Santa Monica, CA: RAND Corporation, 2016. [Online]. Available: https://www.rand.org/pubs/research_reports/RR443-2.html

[27] E. D. Dickmanns, "Vision for ground vehicles: History and prospects," *International Journal of Vehicle Autonomous Systems*, vol. 1, no. 1, pp. 1–44, 2002.

[28] D. A. Pomerleau, "ALVINN: An autonomous land vehicle in a neural network," in *Advances in Neural Information Processing Systems 1 (NIPS 1988)*, D. S. Touretzky, Ed. Morgan Kaufmann, 1989, pp. 305–313.

[29] S. T. et al., "Stanley: The robot that won the DARPA grand challenge," *Journal of Field Robotics*, vol. 23, no. 9, pp. 661–692, 2006.

[30] C. U. et al., "Autonomous driving in urban environments: Boss and the DARPA urban challenge," *Journal of Field Robotics*, vol. 25, no. 8–9, pp. 425–466, 2008.

[31] Physical Intelligence Team, "Physical intelligence: Large-scale models for embodied robots," https://www.physicalintelligence.company/, 2024.

[32] Figure AI, Inc., "Figure robotics collaboration with bmw," https://www.figure.ai/, 2024, press release / announcement.

[33] 1X Technologies, "NEO: Humanoid robot by 1x," https://1x.tech/neo, 2024, accessed: 2025-12-06.

[34] Tesla, Inc., "Tesla full self-driving," https://www.tesla.com/autopilot, 2025, representative reference for Tesla's FSD system.

[35] Waymo LLC, "Waymo one robotaxi service," https://waymo.com/, 2025, representative reference for commercial robotaxi deployment.

[36] Cruise LLC, "Cruise autonomous vehicle program," https://www.getcruise.com/, 2024, representative reference for GM's Cruise robotaxi program.

[37] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. Cambridge University Press, 2003.

[38] P. J. Besl and N. D. McKay, "A method for registration of 3-d shapes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 2, pp. 239–256, 1992.

[39] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: Part i," *IEEE Robotics & Automation Magazine*, vol. 13, no. 2, pp. 99–110, 2006.

[40] G. Grisetti, R. Kümmerle, C. Stachniss, and W. Burgard, "A tutorial on graph-based SLAM," *IEEE Intelligent Transportation Systems Magazine*, vol. 2, no. 4, pp. 31–43, 2010.

[41] A. I. Mourikis and S. I. Roumeliotis, "An efficient multi-state constraint Kalman filter for vision-aided inertial navigation," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2007, pp. 3565–3572.

[42] Y. Bar-Shalom, X. R. Li, and T. Kirubarajan, *Estimation with Applications to Tracking and Navigation*. John Wiley & Sons, 2004.

[43] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. MIT Press, 2018.

[44] V. Mnih, K. Kavukcuoglu, D. Silver *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[45] T. P. Lillicrap, J. J. Hunt, A. Pritzel *et al.*, "Continuous control with deep reinforcement learning," in *International Conference on Learning Representations (ICLR)*, 2016.

[46] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," in *Proceedings of the 34th International Conference on Machine Learning (ICML)*, 2017, arXiv:1707.06347.

[47] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proceedings of the 35th International Conference on Machine Learning (ICML)*, 2018.

[48] D. Silver, A. Huang, C. J. Maddison *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[49] D. Silver, J. Schrittwieser, K. Simonyan *et al.*, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," *arXiv preprint*, vol. arXiv:1712.01815, 2017.

[50] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A physics engine for model-based control," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2012, pp. 5026–5033.

[51] J. Tobin, R. Fong, A. Ray *et al.*, "Domain randomization for transferring deep neural networks from simulation to the real world," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 23–30.

[52] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and Autonomous Systems*, vol. 57, no. 5, pp. 469–483, 2009.

[53] D. Kalashnikov, A. Irpan, P. Pastor *et al.*, "QT-Opt: Scalable deep reinforcement learning for vision-based robotic manipulation," in *Conference on Robot Learning (CoRL)*, 2018, arXiv:1806.10293.

[54] A. Brohan, N. Brown, J. Carbajal *et al.*, "RT-1: Robotics transformer for real-world control at scale," *arXiv preprint*, vol. arXiv:2212.06817, 2022.

[55] A. Brohan *et al.*, "RT-2: Vision-language-action models transfer web knowledge to robotic control," *arXiv preprint*, vol. arXiv:2307.15818, 2023.

[56] O. X.-E. Collaboration, "Open X-Embodiment: Robotic learning datasets and RT-X models," *arXiv preprint*, vol. arXiv:2310.08864, 2023.

[57] A. Correia and L. A. Alexandre, "A survey of demonstration learning," *Robotics and Autonomous Systems*, vol. 182, p. 104812, 2024, online first 2023.

[58] Google Brain Robotics Team, "Qt-Opt for robotic grasping," https://ai.googleblog.com/2018/10/qt-opt-scalable-deep-reinforcement.html, 2018, accessed: 2025-12-06.

[59] Google Robotics Team, "RT-1: Robotics transformer for real-world control at scale," https://ai.googleblog.com/2022/12/rt-1-robotics-transformer-for-real.html, 2022, accessed: 2025-12-06.

[60] Google DeepMind and collaborators, "Introducing open X-Embodiment," https://openxembodiment.github.io/, 2023, accessed: 2025-12-06.

[61] I. Akkaya, M. Andrychowicz, M. Chociej *et al.*, "Solving rubik's cube with a robot hand," *arXiv preprint*, vol. arXiv:1910.07113, 2019.

[62] B. R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. El Sallab, S. Yogamani, and R. Pérez, "Deep reinforcement learning for autonomous driving: A survey," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 6, pp. 4909–4926, 2022.

[63] S. Levine, A. Kumar, G. Tucker, and J. Fu, "Offline reinforcement learning: Tutorial, review, and perspectives on open problems," *arXiv preprint*, vol. arXiv:2005.01643, 2020.