Grundlagen der künstlichen Intelligenz – Robotics

Matthias Althoff

TU München

January 24, 2020

1/23
1/30

Organization

1. Robot Hardware
2. Robotic Perception
3. Path Planning
4. Planning Uncertain Movements
5. Planning in Uncertain Environments (Own Research)
6. Control of Movements
7. Robotic Software Architectures
8. Application Domains

The content is covered in the AI book by the section “Robotics”.

Learning Outcomes

- You can categorize robots into *manipulators*, *mobile robots*, and *mobile manipulators*.
- You understand the difference between *passive sensors*, *active sensors*, *proprioceptive sensors* and name typical examples.
- You can determine the *degrees of freedom* of a robot.
- You can determine whether a robot is *holonomic* or *non-holonomic*.
- You can explain the difference between *sensing* and *perception*.
- You can explain the difference between deterministic and stochastic *transition models* for robot localization.
- You can explain the terms *workspace*, *configuration space*, *forward kinematics*, and *inverse kinematics*.
- You can explain important planning concepts: *cell decomposition*, *skeletonization*, *re-planning*, and *robustification*.
- You can explain the basic concept of *feedback control*.

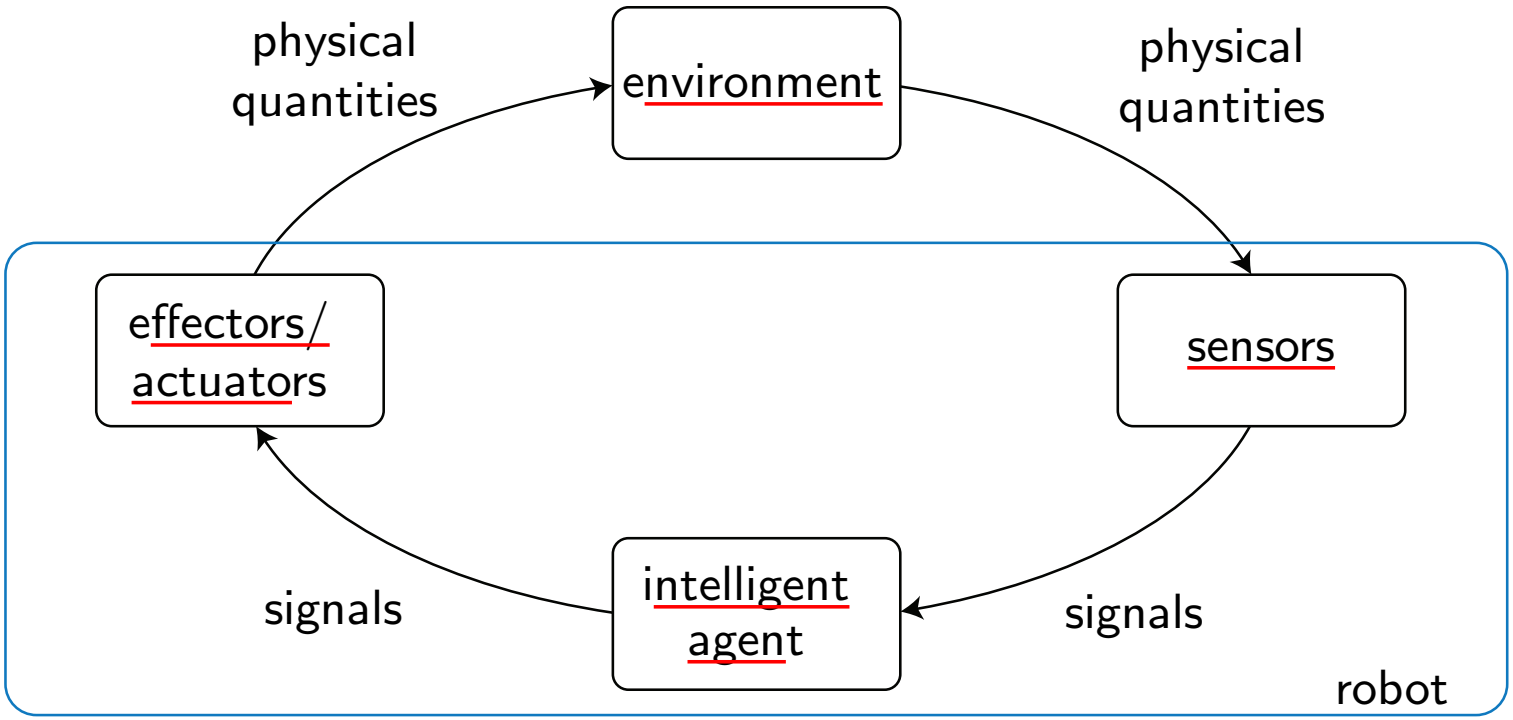
Introduction

Definition

Robots are physical agents that perform tasks by manipulating the physical world.

Robots are typically an electro-mechanical machine guided by a computer program or electronic circuitry.

The environment is sensed by **sensors** and manipulated by **effectors**:



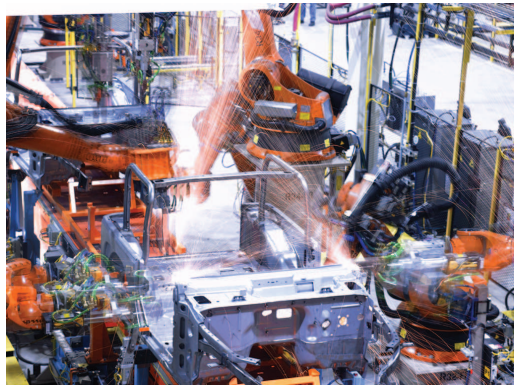
The Three Laws of Robotics

The Three Laws of Robotics (also known as Asimov's Laws) are a set of rules devised by the science fiction author Isaac Asimov in 1942.

1. A robot may not injure a human being or, through inaction, allow a human being to come to harm.
2. A robot must obey the orders given to it by human beings, except where such orders would conflict with the First Law.
3. A robot must protect its own existence as long as such protection does not conflict with the First or Second Law.

Primary Categories

- **Manipulators** or robot arms
  Physically anchored to their workspace, e.g., in a factory assembly line.
- **Mobile robots**
  Mobile robots move about their environment using wheels, legs, or similar mechanisms. Examples are unmanned ground vehicles and unmanned air vehicles.
- **Mobile manipulators**
  Combination of the previous categories, e.g., humanoid robots.



source: www.kuka.de

source: www.hhla.de

source: www.asimo.honda.com

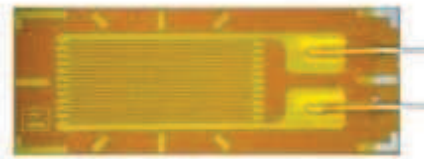
Why is Robotics Challenging?

- **Environment**
  Real robots must cope with environments that are partially observable, stochastic, dynamic, and continuous. Many robot environments are sequential and multiagent as well.
- **Time-consuming tests**
  The real world does not move faster than real time. It is e.g., not possible to only learn based on simulations, which can be performed much faster than real time.
- **Real time constraints**
  Tasks in AI can be very time consuming, see e.g., the n-queens problem. In robotics, the robot cannot think for hours before performing an action.
- **Interdisciplinarity**
  Robotics brings together researchers from mechanical and electrical engineering, computer science, ergonomics, cognitive sciences, and psychology.

Typical Sensors

acceleration sensor

light sensor



force/strain sensor

SONAR (SOund Navigation And Ranging)

video camera

LIDAR (light+radar)

pressure sensor

angle sensor

Sensor Categories

- **Passive sensors**
  Detection of effects that are generated by other sources in the environment.
  Examples: Video cameras, infrared cameras, global positioning system (GPS), light sensor, etc.

- **Active sensors**
  Energy is used to send signals into the environment, that are reflected and detected. Pro: typically more information provided. Con: increased power consumption and danger of interference.
  Examples: SONAR, RADAR, LIDAR, etc.

- **Proprioceptive sensors**
  Proprioceptive sensors inform the robot of its own motion.
  Examples: Angle sensor, acceleration sensor, force/strain sensor, pressure sensor, etc.

Effectors: Degrees of Freedom

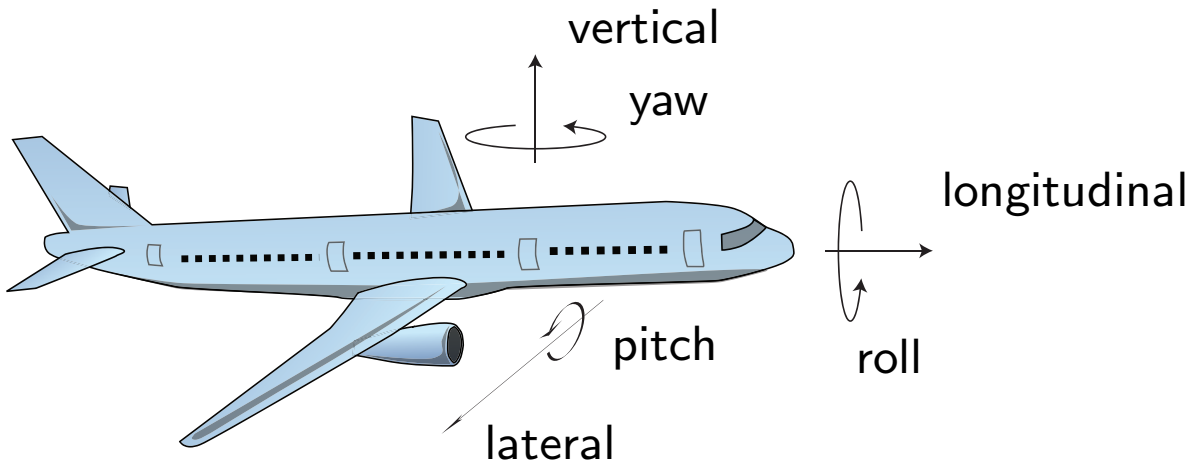
Degree of freedom

We count one degree of freedom for each independent direction in which the robot can move.

More scientifically: Number of independent parameters that define the configuration of the robot.

A single, solid three-dimensional object has 6 degrees of freedom:

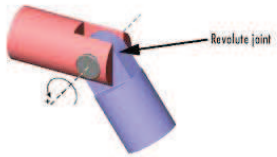
- 3 translational directions (x, y, z; or: longitudinal, lateral, vertical)
- 3 angular directions (yaw, roll, pitch)



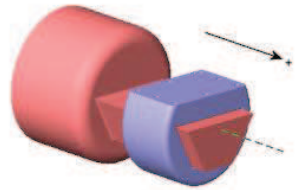
Effectors: Joints

There are different kinds of robotic joints with different degrees of freedom:

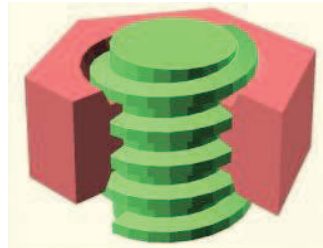
- **Revolute joint**
  One rotational degree of freedom



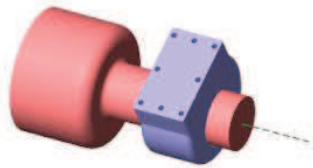
- **Prismatic joint**
  One translational degree of freedom



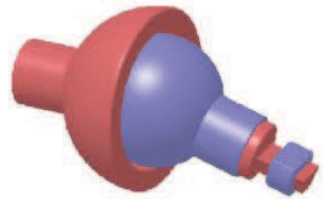
- **Helical joint**
  Displacement d and relative angle θ are linearly related: $d = h\theta$.



- **Cylindrical joint**
  One rotational and one translational degree of freedom.

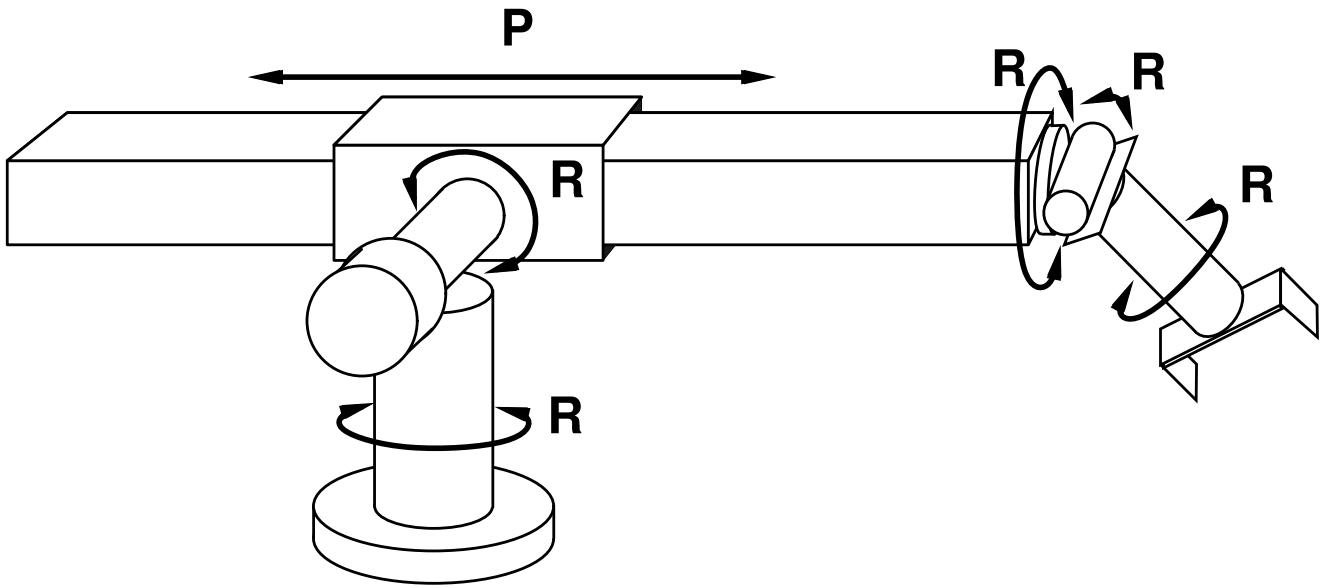


- **Spherical joint**
  Three rotational degrees of freedom.



- **Planar joint**
  Three translational degrees of freedom (no figure shown).

Effectors: Exemplary Manipulator



- Configuration of robot specified by 6 numbers
  $\Rightarrow$ 6 degrees of freedom (DOF).
- 6 is the minimum number required to position end-effector arbitrarily.

Tweedback Question

Does the human arm have more than 6 degrees of freedom?

Experiment: Put your hand on the table. Can you still move your elbow? What does the result imply?

A We have less than 6 degrees of freedom.

B We have exactly 6 degrees of freedom.

C We have more than 6 degrees of freedom.

For object manipulation, we just need 6 DoF

i.e upper hand has 6, lower has 6, one can consider additional revolute at elbow

Nonholonomic Robots

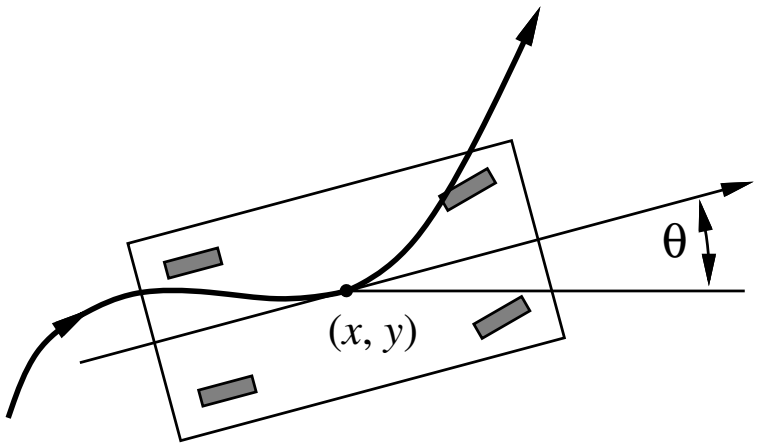
Definition

We say a robot is **nonholonomic** if it has more effective DOFs than controllable DOFs and **holonomic** if the two numbers are the same.

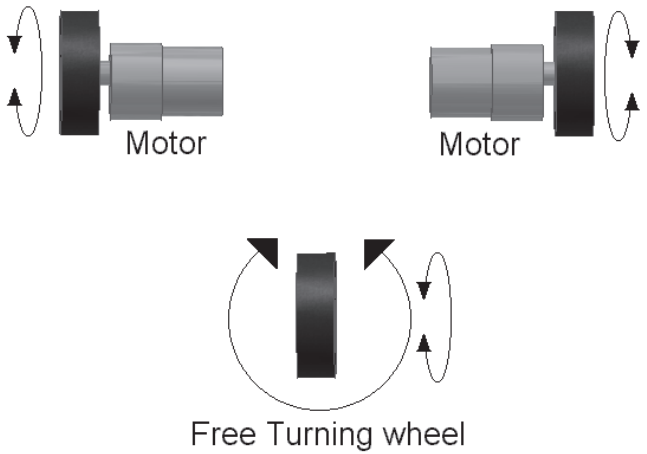
Example: A car has 3 DOFs (two translational x, y, one rotational θ) and 2 controllable DOFs (steering, accelerating/braking)

$\Rightarrow$ cars are nonholonomic.

$\Rightarrow$ cars cannot generally transition between two infinitesimally close configurations.



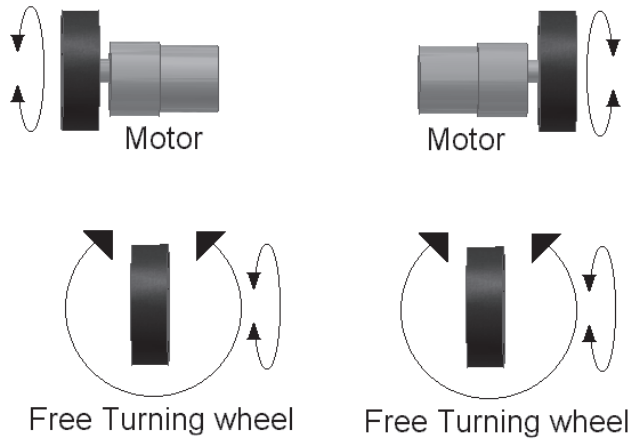
Mobile Platforms (3 wheels)



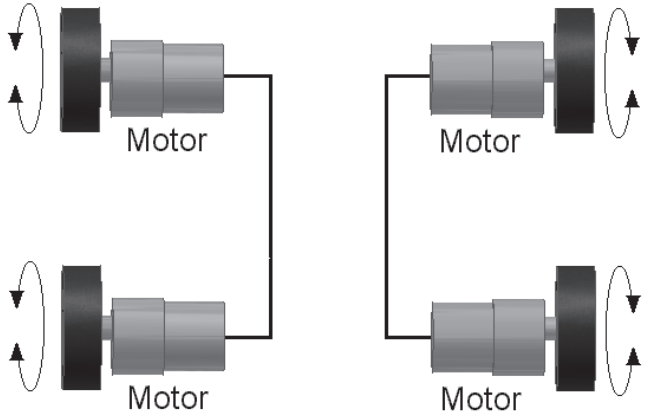
- **Differential drive**
  The robot possesses independently actuated wheels on each side. Steering is performed as for a military tank.
- **Steering**
  Turning is achieved by rotating a wheel around its vertical axis. One motor is sufficient for this design.

Mobile Platforms (4 wheels)

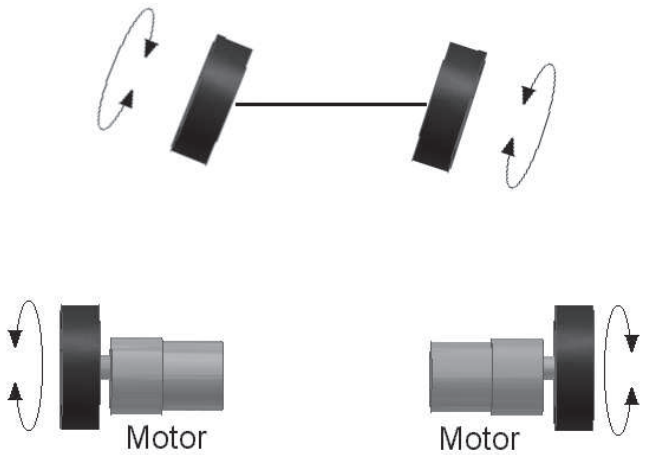
2 powered, 2 free rotating wheels:



2-by-2 powered wheels for tank-like movement



Car-like steering:



Mobile Platforms (special wheels)

Omni wheels and Mecanum wheels are used to build holonomic mobile platforms.

- **Omni wheels** Typically used in a three-wheel setting.

- **Mecanum wheels** Typically used in a four-wheel or more-than-four-wheel setting.

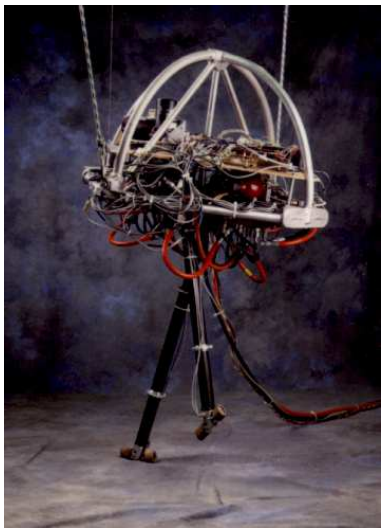
Mobile Platforms with Legs (1)

Why legs?

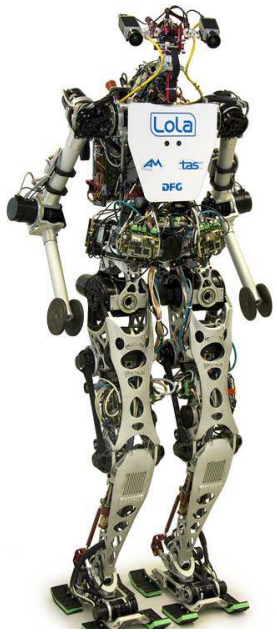
- Legs are notoriously bad on flat surfaces compared to wheeled platforms.
- However, legs are good in rough terrain. There must be a reason why animals have no wheels...

Robotic walking is still an unsolved research problem!

One of the first walking robots from robotic walking pioneer Marc Raibert (3D Biped, 1989-1995):



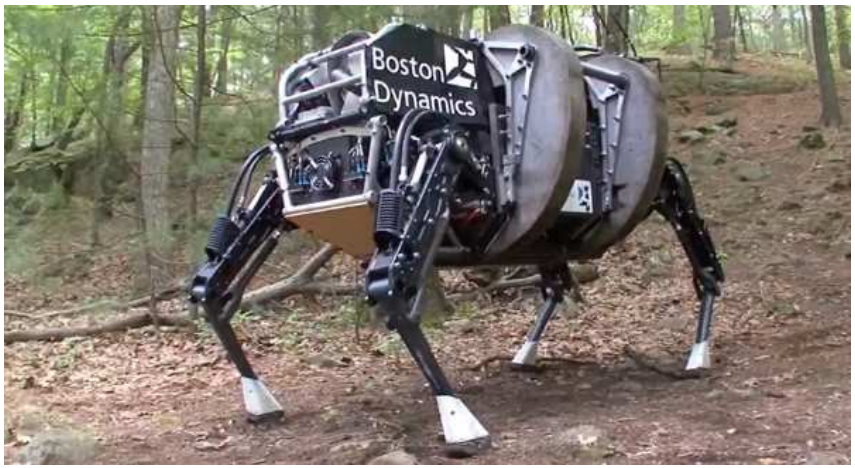
TUM's most recently developed walking robot (Lola, 2010):



Mobile Platforms with Legs (2)

Further legged robots...

Four-legged dynamically-stable robot Big Dog (Boston Dynamics, 2009):



Humanoid H25 NAO from Aldebaran Robotics used in the Standard Platform League of Robo Cup:

Snake Robots

Why snake robots?

- Snake robots are even slower than legged robots.
- But: Snake robots can climb, crawl through tiny openings, swim, are robust (operational if one link fails), and are hard to detect.
- Huge size variations: Some robots are designed to crawl into your body!

Snake robot from Carnegie Mellon University (Uncle Sam, 2010):

Snake-like robots can also be used as manipulators (Festo's Bionic Handling Assistant, 2010):

Aerial Robots: Unmanned Aerial Vehicles (UAVs)

- **Autonomous**: good progress in research, but unsolved legal/liability issues.
- **Remotely piloted**: mostly used in military applications today.

Possible benefits:

- increased payload capabilities (lack of a cabin);
- unique flight envelopes (no g-force restrictions);
- no loss of life in dangerous missions;
- possibility of long surveillance missions;
- personal aerial deliveries and photography.

General Atomics Predator (fixed wing, 1995):

DHL delivery drone (quadrocopter, 2014):

Perception: General Discussion

Sensing

The detection of a physical presence and its conversion into a signal that can be read by an observer or an instrument. i.e no interpretation of the signal

Perception

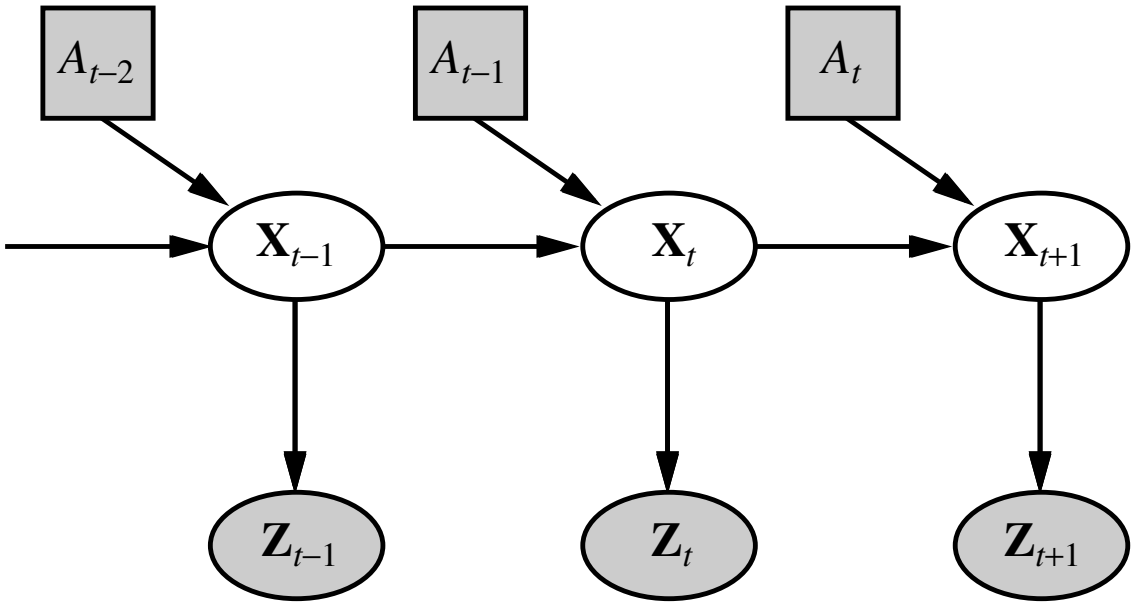
The organization, identification, and interpretation of sensory information in order to represent and understand the environment.

- Perception is difficult because sensors are noisy, and the environment is partially observable, unpredictable, and often dynamic.
- The task of perception has been addressed before by Hidden Markov Models: What are the internal states (representing the environment) given sensor information? We used **filtering** to solve the problem.
- Difference to HMM filtering in robotics:
  - The state also consists of the robot state.
  - The state is continuous.

Filtering in Robotics (1)

we can only use filtering. We can not use smoothing; no info about the future

- In robotics and control theory, the term **state estimation** is typically used instead of filtering.
- We refer to the state of the combined state of the environment and the robot at time t as $\mathbf{X}_t$, $\mathbf{Z}_t$ is the observation, and A_t is the taken action.



Filtering in Robotics (2)

- We compute the belief state identically to the filtering algorithm for HMMs, except that the summation is replaced by an integral (due to continuous states):

$$\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{z}_{1:t+1}, a_{1:t}) =$$
$$\alpha\mathbf{P}(\mathbf{z}_{t+1}|\mathbf{X}_{t+1}) \int \mathbf{P}(\mathbf{X}_{t+1}|\mathbf{x}_t, a_t)P(\mathbf{x}_t|\mathbf{z}_{1:t}, a_{1:t-1})\mathrm{d}\mathbf{x}_t. \quad (1)$$

- As for HMMs, $\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{x}_t, a_t)$ is called the **transition model** and $\mathbf{P}(\mathbf{z}_{t+1}|\mathbf{X}_{t+1})$ is the **sensor model**.

Localization

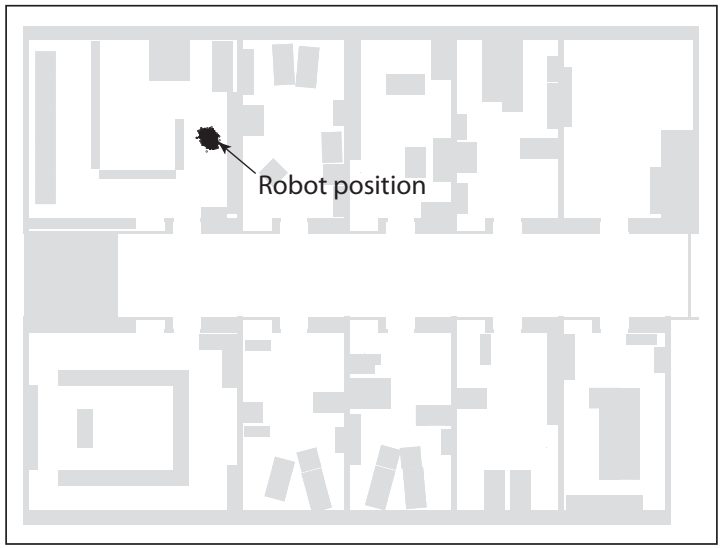
A particular instance of filtering (or state estimation) is **localization**.

Localization

Localization is the problem of finding out where things are – including the robot itself.

To keep things simple, we assume that

- the robot moves slowly on a flat 2D surface,
- the robot knows the exact map of the environment (see figure below).



Deterministic Transition Model

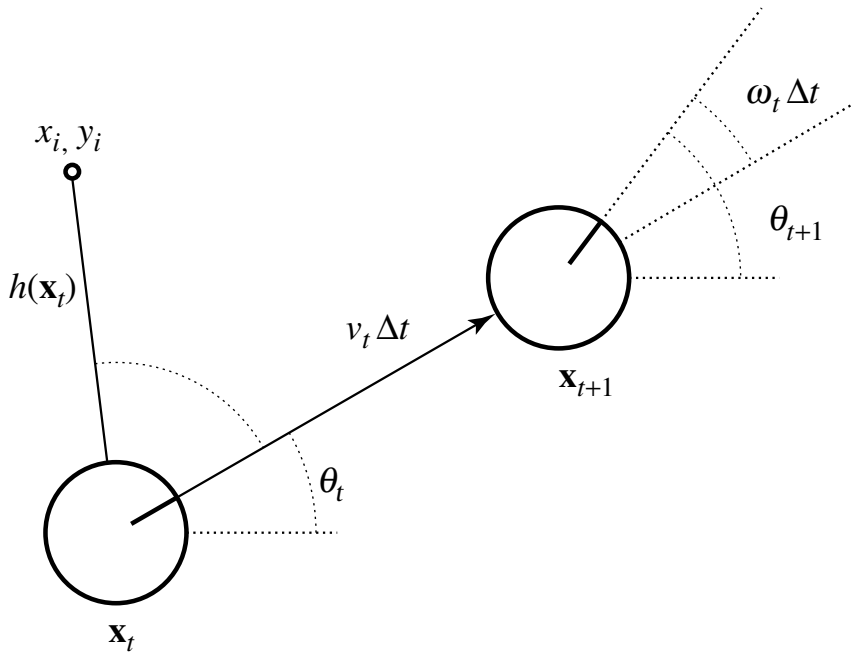
The position on a 2D surface is determined by

- two Cartesian coordinates x and y and
- the heading angle θ.

Given the inputs to the robot, the commanded translational velocity v_t and the rotational velocity ω_t, the transition model for a slow moving robot without uncertainty is

$$\hat{\mathbf{x}}_{t+1} = f(\mathbf{x}_t, \underbrace{v_t, \omega_t}_{a_t})$$

$$= \mathbf{x}_t + \begin{bmatrix} v_t \Delta t \cos\theta_t \\ v_t \Delta t \sin\theta_t \\ \omega_t \Delta_t \end{bmatrix}.$$



((x_i, y_i) is a landmark)

Stochastic Transition Model

- Real robots are to some degree unpredictable, which is often modeled by Gaussian distributions $N(\mathbf{c}, \mathbf{\Sigma})$, where $\mathbf{c}$ is the center and $\mathbf{\Sigma}$ is the covariance matrix.
- We select the state of the deterministic prediction as the center and the covariance as $\mathbf{\Sigma}_x$.
- The stochastic transition model is

$$\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{X}_t, v_t, \omega_t) = N(\hat{\mathbf{X}}_{t+1}, \mathbf{\Sigma}_x).$$

Comment: In case you have never worked with continuous probability distributions – don't worry, it won't be part of the exam.

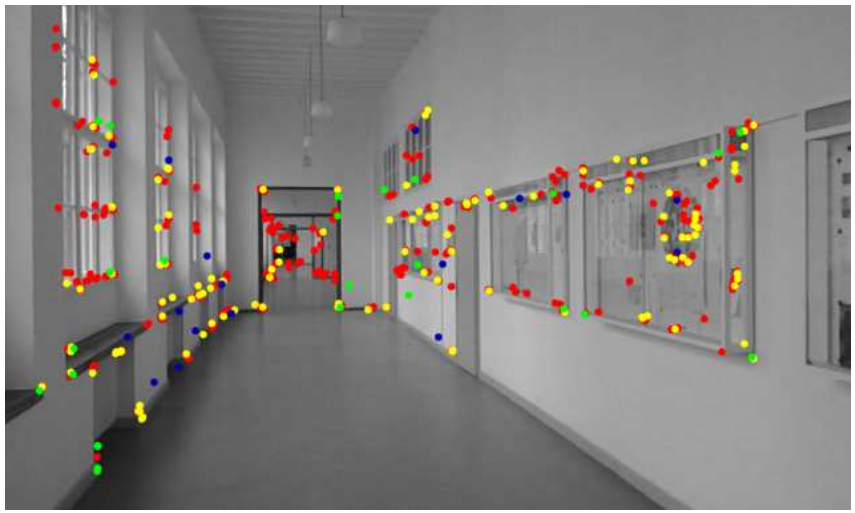
Sensor Model

We consider two example sensing techniques:

- landmark-based sensing and
- an array of range sensors.

Landmark

A **landmark** is a stable, recognizable feature of the environment used for navigation.



(Features in corridor at TUM Arcisstr)

Landmark Model

We require the distance and bearing of each landmark.

Given the robot state as $\mathbf{x}_t = \left[x_t, y_t, \theta_t\right]^T$ and the landmark location as $\left[x_i, y_i\right]^T$, we obtain the distance d and bearing ϕ as

$$d = \sqrt{(x_t - x_i)^2 + (y_t - y_i)^2},$$

$$\phi = \arctan\left(\frac{y_i - y_t}{x_i - x_t}\right) - \theta_t,$$

which are combined in the observation

$$\hat{\mathbf{z}}_t = h(\mathbf{x}_t) = \begin{bmatrix} d \\ \phi \end{bmatrix}.$$

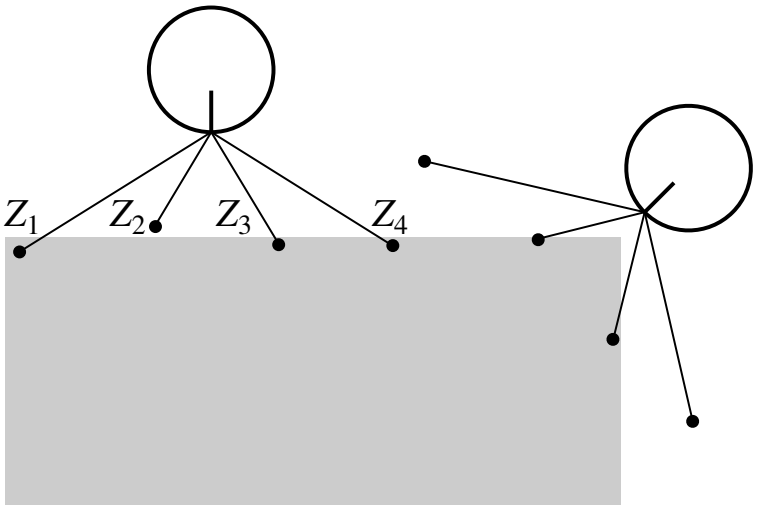
Again, we add Gaussian noise to model measurement errors:

$$P(\mathbf{z}_t|\mathbf{x}_t) = N(\hat{\mathbf{z}}_t, \mathbf{\Sigma}_z).$$

Model for Array of Range Sensors

An array of range sensors produces a vector of range values

$$\mathbf{z}_t = \left[z_1, \ldots, z_M\right]^T.$$



(beams of an array of range sensors)

(image of a Velodyne LIDAR mounted on a car)

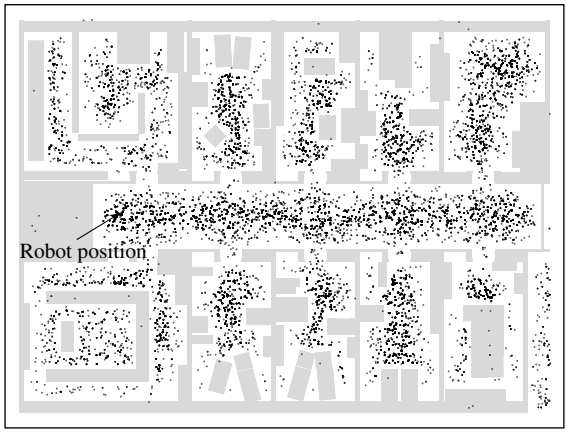
Assuming that the measurements are corrupted by Gaussian noise, which acts independently on each beam, we have

$$P(\mathbf{z}_t|\mathbf{x}_t) = \alpha \prod_{i=1}^{M} e^{-(z_j - \hat{z}_j)/2\sigma^2}.$$

Solving the Recursive Filtering Equation

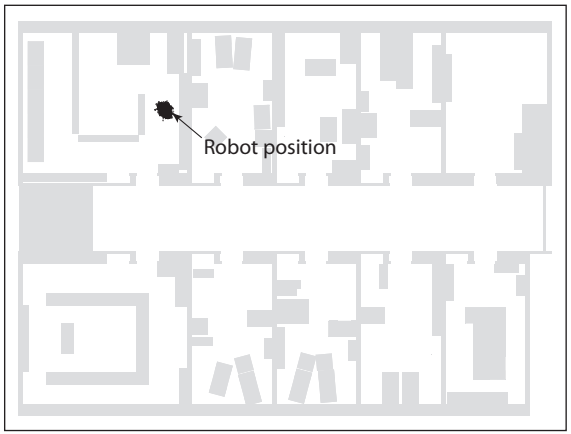
We mention two practical approaches for the solution of the recursive filtering equation, see (1) on slide 24.

- Monte Carlo localization (see hidden Markov models)



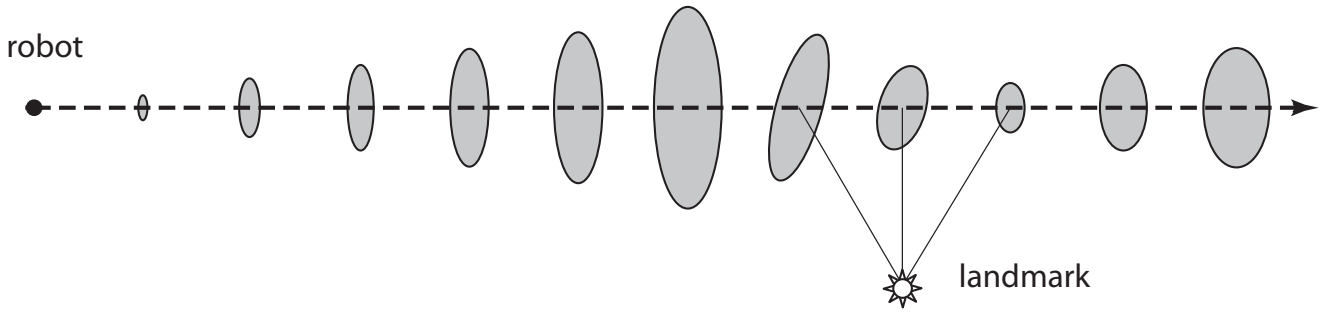
(first snapshot)

(second snapshot)



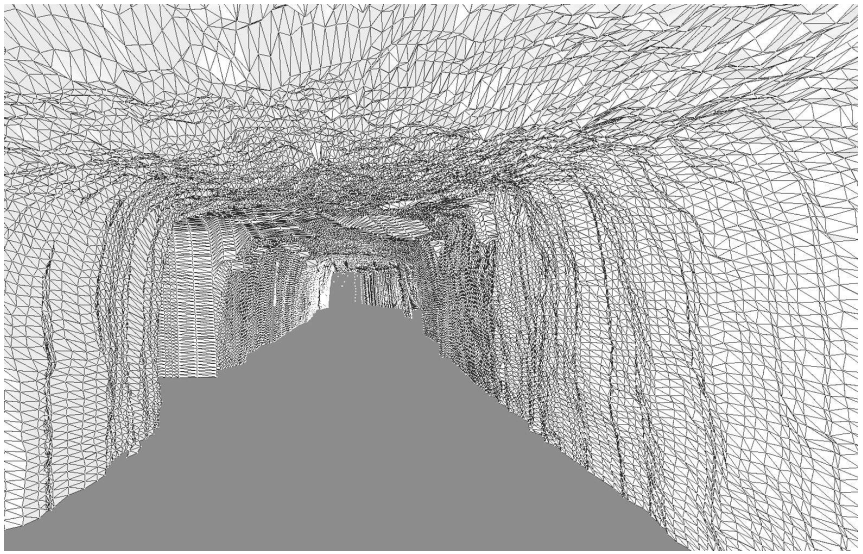
(third snapshot)

- Kalman filter (see lecture "Cyber-Physical Systems")



Simultaneous Localization and Mapping

In many applications, no map of the environment is available, which requires **simultaneous localization and mapping (SLAM)**.



Types of Movement

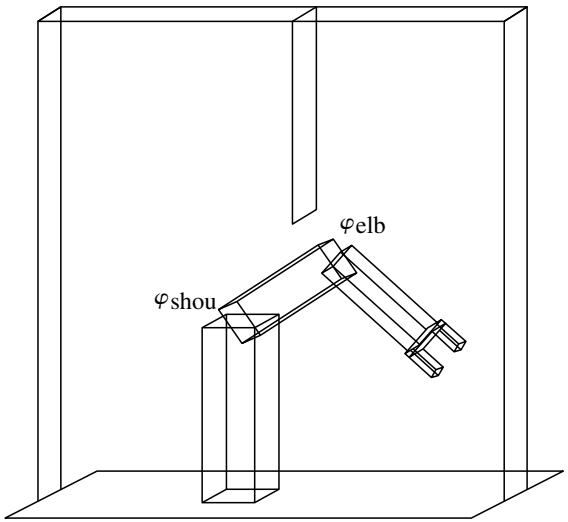
Major types of movement

- **Point-to-point motion**: Position the robot or its end effector to a designated target location.
- **Compliant motion**: the robot moves while being in physical contact with an object.
  Examples: A robot screws in a light bulb, or pushes a box across a table top.

- The more challenging task is compliant motion, since the movement is constrained such that appropriate forces need to be applied.
- We will introduce the **configuration space** to standardize the planning problem.
- We introduce two techniques to transform the continuous problem into a discrete search problem: **cell decomposition** and **skeletonization**.

Workspace

We introduce the workspace by example. Consider a robot arm with two independent joints:



The configuration of the 2D robot (displayed in 3D) can be described by a 4-dimensional coordinate:

- $[x_e, y_e]$ for the location of the elbow relative to the environment and
- $[x_g, y_g]$ for the location of the gripper.

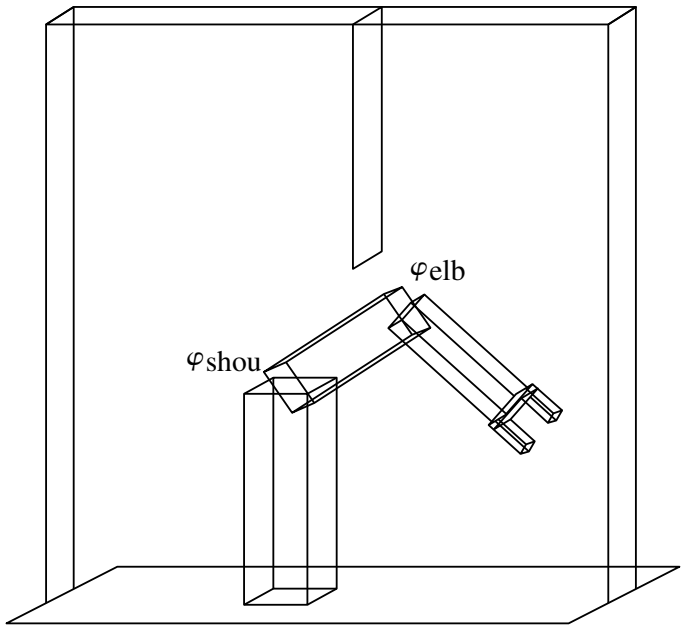
These coordinates constitute what is known as the **workspace representation** (coordinates are in the coordinates of the objects to be manipulated).

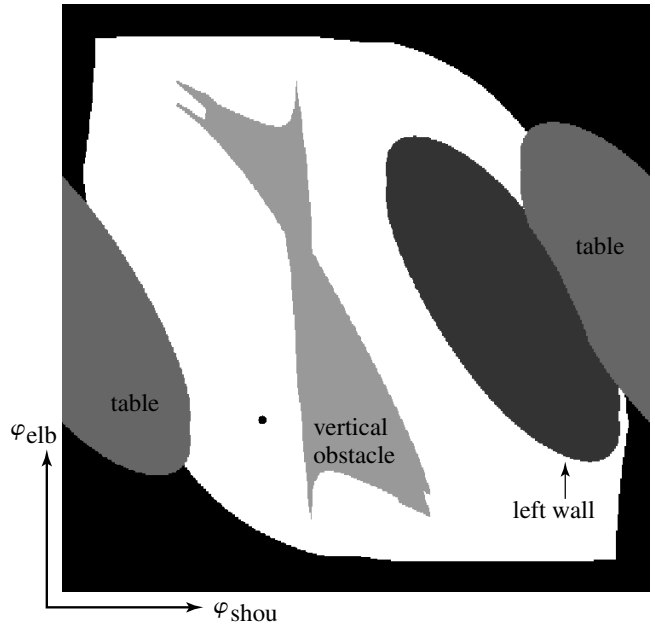
Disadvantages of the Workspace Representation

- Not all workspace coordinates are possible, even in the absence of obstacles.
- E.g., due to the **linkage constraints**, the position of the elbow and the gripper are always a fixed distance apart.
- It is difficult for a planner to meet all these constraints, which are continuous and nonlinear.
- The configuration space circumvents these issues.

Configuration Space

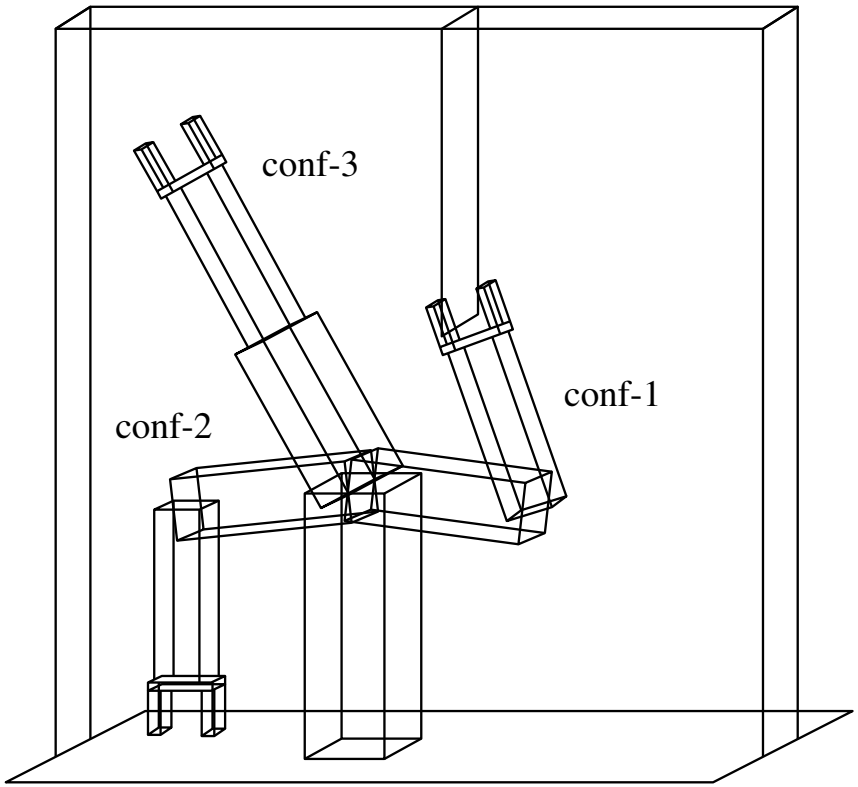
- In **configuration space** we represent the robot by a configuration of its joints.
- Our example robot possesses two joints ϕ_s and ϕ_e (shoulder and elbow joint).
- Due to constraints of joint movement and obstacles, not all joint values are possible:

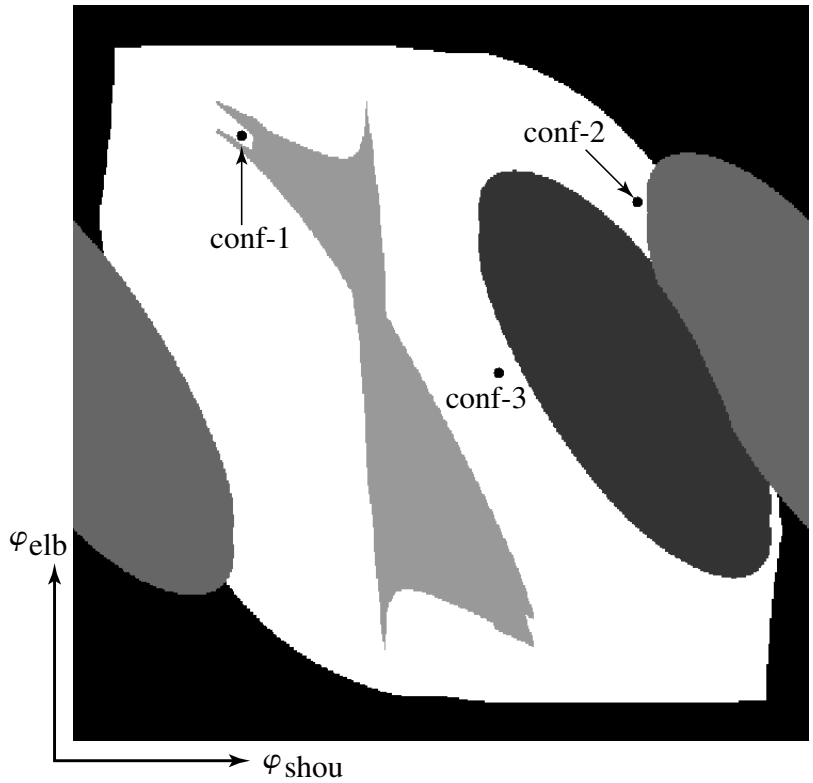




Configuration Space: Examples

Here are some example configurations:





The attainable configurations are often called **free space**, and the unattainable configurations are often called **occupied space**.

Forward and Inverse Kinematics

Forward Kinematics

The **forward kinematics** provides the position of the end-effector or other joint positions from the joint values.

Forward kinematics is rather trivial and straightforward to compute using rotation matrices (see lecture *Robotik* by Darius Burschka).

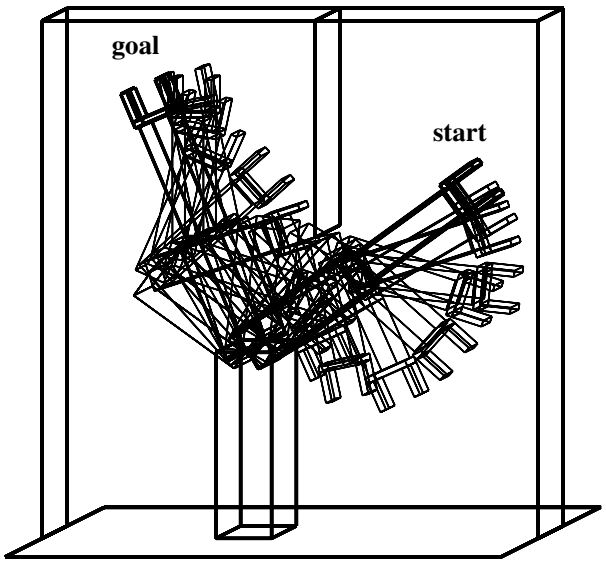
Inverse Kinematics

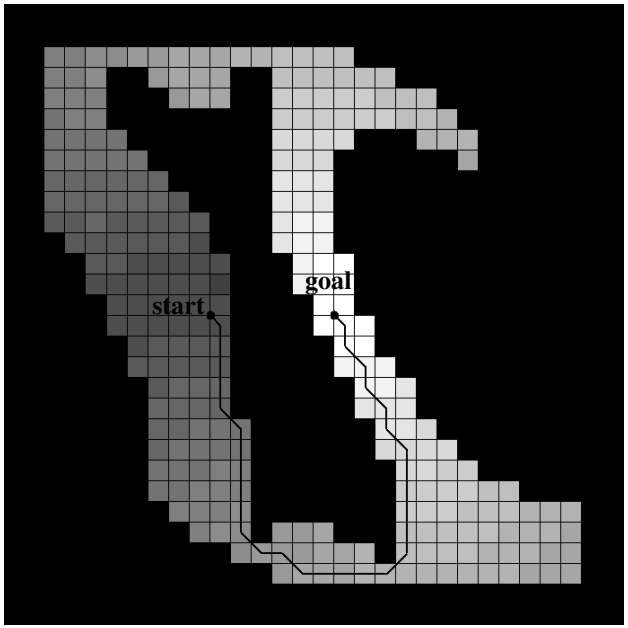
The **inverse kinematics** provides the joint angles for a desired position of the end-effector or other joints.

- Inverse kinematics can be seen as the inverse function of forward kinematics.
- The problem with inverse kinematics is that the solution is often not unique.
- For instance, different joint angles of a 6 DOF robot are possible when only the 3D position of the end-effector is fixed. How many? How many are possible for the previous 2D robot?

Cell Decomposition Methods

- How to plan in continuous space? We need to discretize!
- The easiest technique is to construct a grid with a finite number of vertices.
- One can use, e.g., A* to find the optimal path.
- The optimal path together with a color coding for the cost-to-go (the darker, the more costly) is shown for our example robot:





Cell Decomposition: Advantages and Disadvantages

Advantage

Very simple to implement.

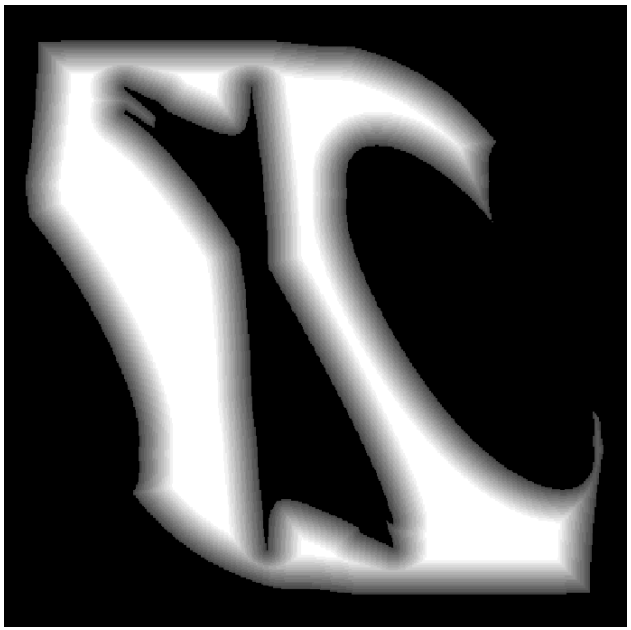
Disadvantages

- Only feasible for low-dimensional configurations since the number of grid cells grows exponentially with the dimension d.
- Cells that are partially feasible: When accepting, the path might be *unsound*. Otherwise the planner is *incomplete*.
- The obtained path might not be smooth (differentiable) and thus not feasible in practice.

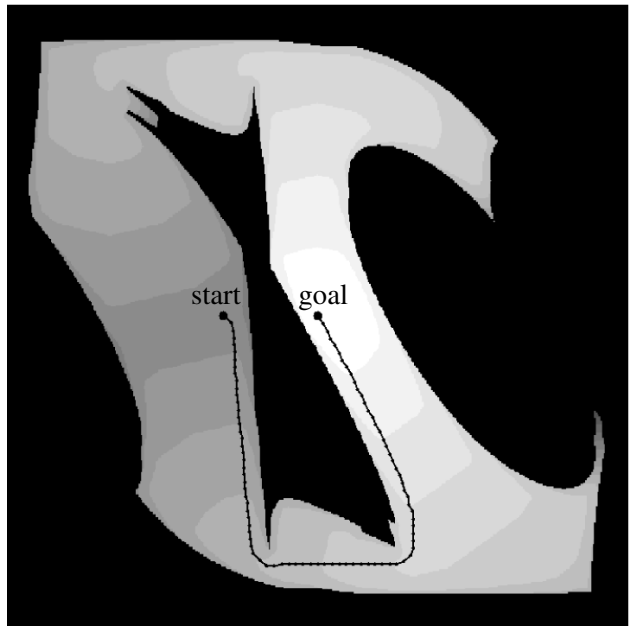
Disadvantages can be overcome by *subdividing* partially feasible cells.

Modified Cost Functions

- One can observe that the optimal path on slide 39 is close to the border.
- **Problem:** The previous approach is not robust against disturbances (e.g., car parking with 1 mm clearance is not advisable).
- **Solution:** We add additional cost for being close to a border. This is often done using **potential fields**.



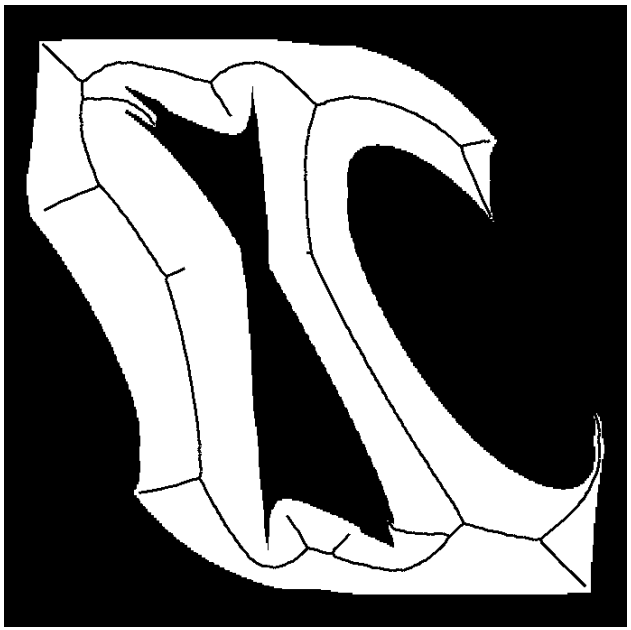
Repelling potential field.



Path found by minimizing path length and potential.

Skeletonization Methods: Voronoi Graph[1]

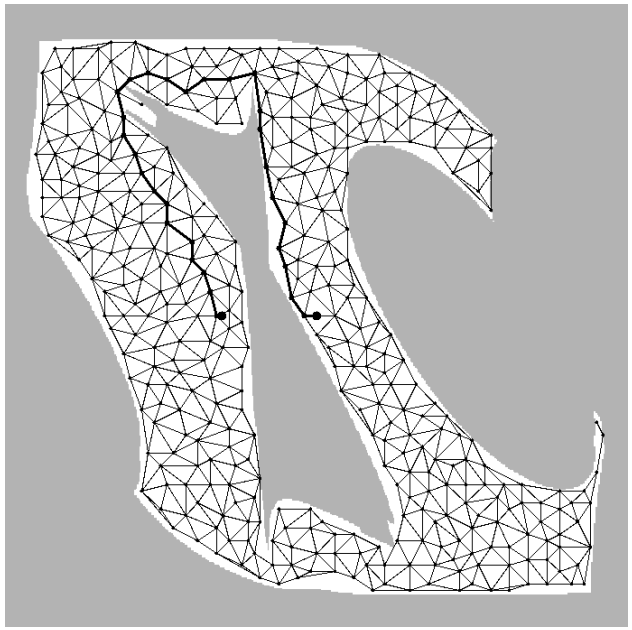
- An alternative to cell decomposition is **skeletonization**: the free space is approximated by a line of which other lines branch-off.
- A possible skeleton is a **generalized Voronoi graph**, which is the set of points equidistant to two or more obstacles (much clearance, longer paths):



- The robot first changes its present configuration to a point on the generalized Voronoi graph, then follows it until the robot is nearest to the target, from which it moves to the target.

Skeletonization Methods: Probabilistic Roadmaps

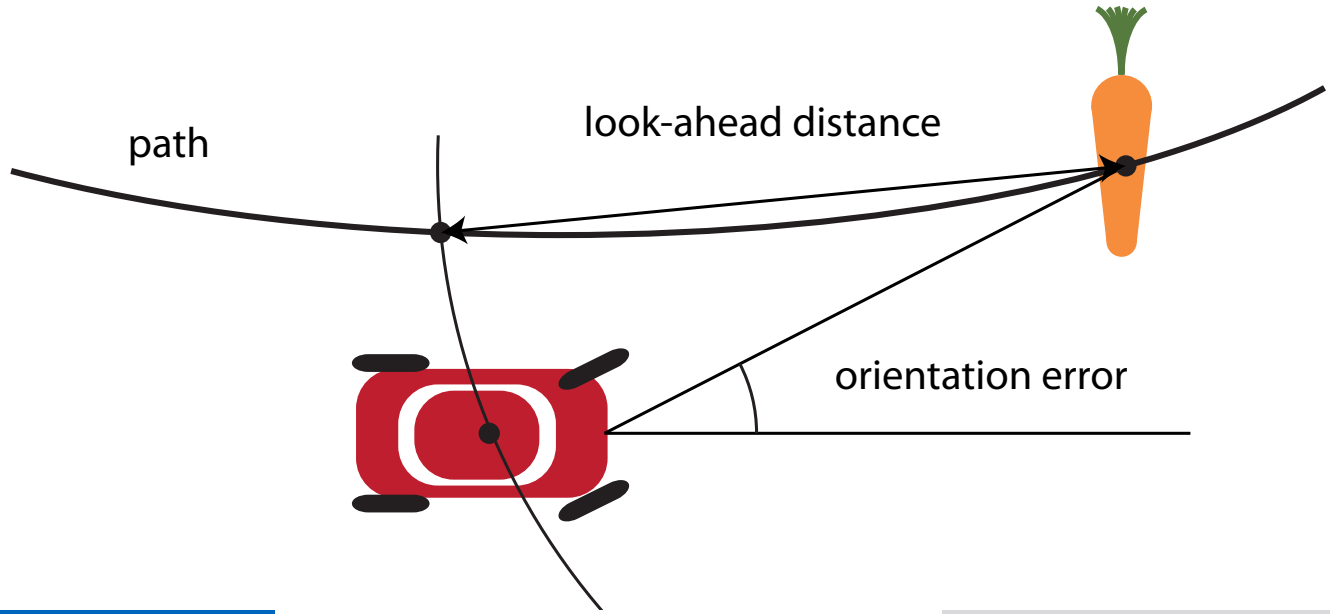
- Another skeletonization method is that of **probabilistic roadmaps**, which offer more possible routes compared to Voronoi graphs.
- Probabilistic roadmaps are generated by randomly sampling possible configurations and connecting them if a feasible path exists:



- Theoretically this approach is incomplete. However, one can bound the probability of failure in terms of the number of points generated and in terms of certain geometric properties of the configuration space.

Online Replanning

- None of the previous motion-planning algorithms address **uncertainty** in state estimation and movements.
- A simple and practical approach is to use the most likely state of the estimation in combination with one of the previous path planners.
- The uncertainty in state estimation and movement is compensated by constantly replanning the movement.
- An example of this approach is the follow-the-carrot method for mobile robots:



MDPs and POMDPs

- Sometimes the uncertainty is too large for *online replanning*.
- If the robot faces uncertainty only in its state transition, but its state is fully observable, the problem is best modeled as a Markov decision process (MDP).
- The solution of an MDP is an optimal **policy**, which tells the robot what to do in every possible state (see lecture *Rational Decisions Over Time*).
- When the state is also uncertain, the problem is best modeled as a partially observable Markov decision process (POMDP).
- In such situations, the robot maintains an internal belief state (see lecture *Rational Decisions Over Time*).
- POMDP naturally result in **information gathering action**.
- POMDP are often infeasible for real applications, such that heuristics are used, e.g., moving close to landmarks to reduce uncertainty (**coastal navigation**).

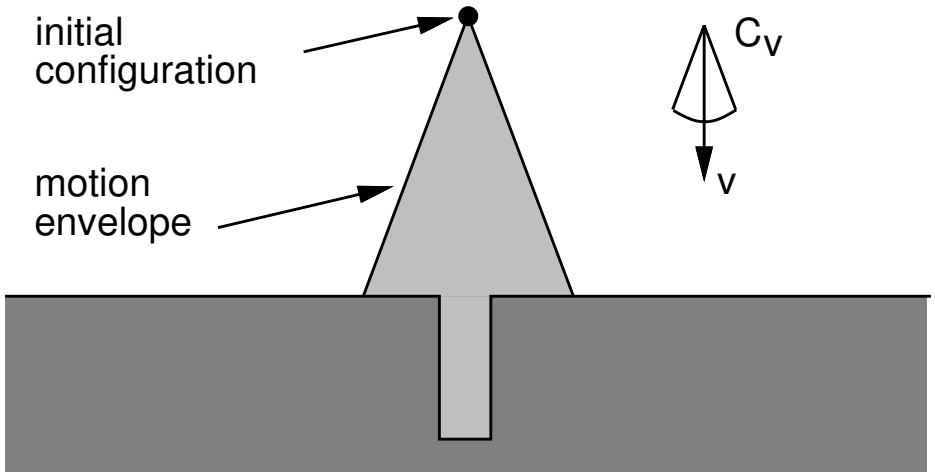
Robust Methods

Robust solution

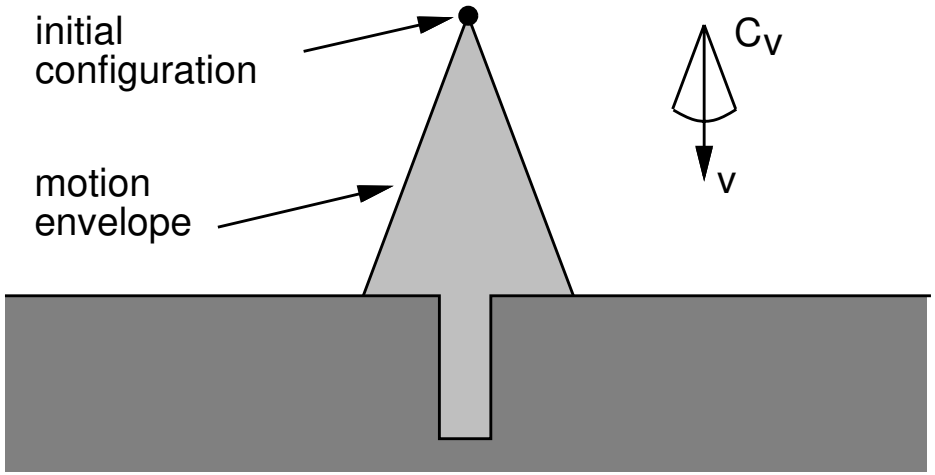
A robust solution works no matter what actual values occur, provided they are within an assumed interval.

In contrast to previous methods, robust solutions do not require probabilities of values within the allowed interval.

Example: **Fine-motion planning task**, in which the relevant features of the environment are very small. Thus, entering the hole is difficult if the motion direction is uncertain within the cone C_v:

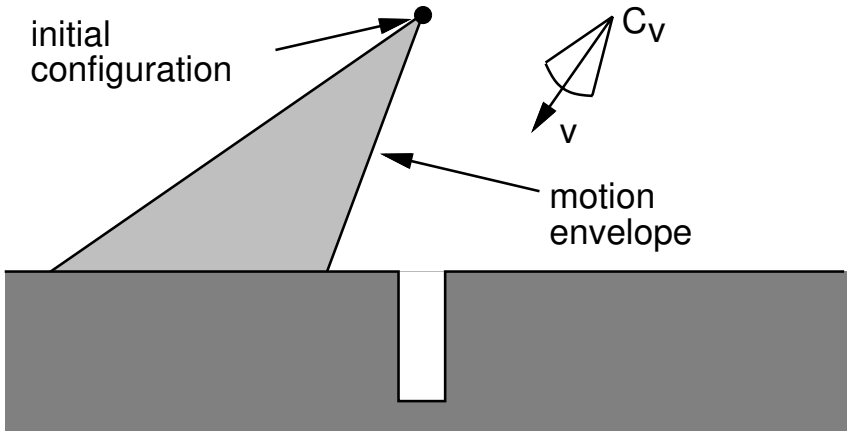


Peg-In-Hole: Unsuccessful Strategy

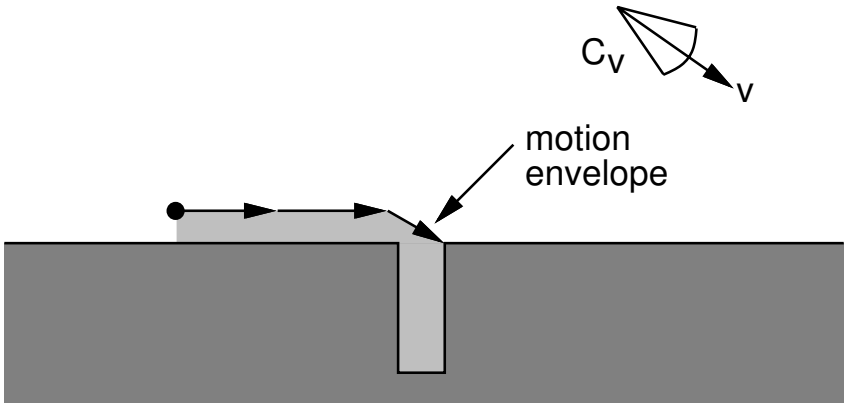


- A fine-motion plan consists of a series of **guarded motions**.
- Each guarded motion consists of
  - a motion command and
  - a termination condition, which is a predicate on the sensor values (returns true at the end of the guarded move).
- The peg possibly lands in the hole or on its side. If the peg is not in the hole, where to move next without sensor information?

Peg-In-Hole: Successful Strategy



Step 1



Step 2

- The new directions of the velocity cones guarantee that the peg slides into the hole.
- This is ensured by switching the direction cone after a contact with the surface has been detected.

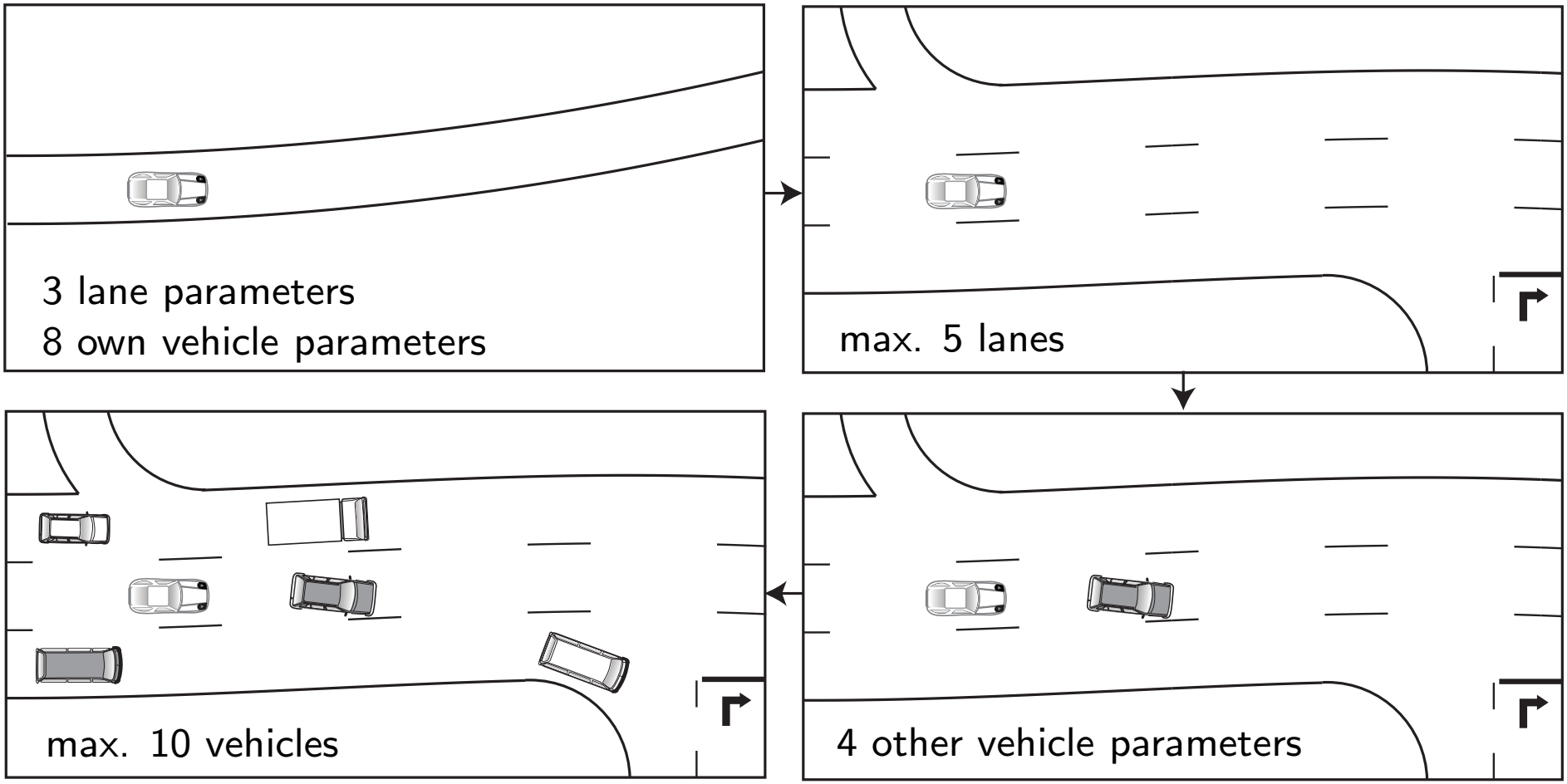
Cyber-Physical Systems (CPS) in Uncertain Environments

How to ensure safety in uncertain environments?
Example: automated driving, robotics, and smart grids.




Possible Traffic Situations: A Rough Estimation

We assume that each variable of the verification problem has 20 values.



$(20^3)^5 \cdot (20^4)^{10} \cdot 20^8 \approx 9.2 \cdot 10^{81}$ scenarios

Are We Considering All Scenarios?

Self-driving cars

Google self-driving car caught on video colliding with bus

The crash – the first caused by a self-driving car – tore off its radar, flattened its tire, and crumpled its side. The bus driver was not at fault, the footage shows

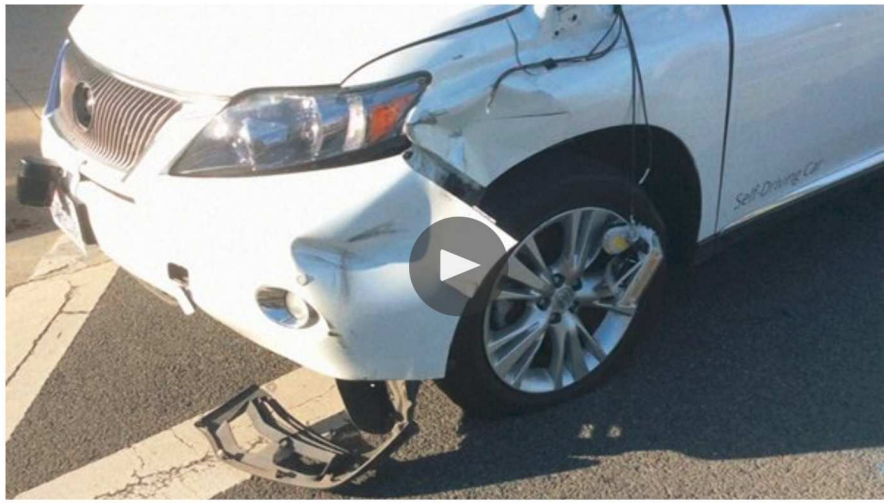
Associated Press

Wednesday 9 March 2016 20.01 GMT

This article is 2 months old

Shares 1638 Comments 283

Save for later



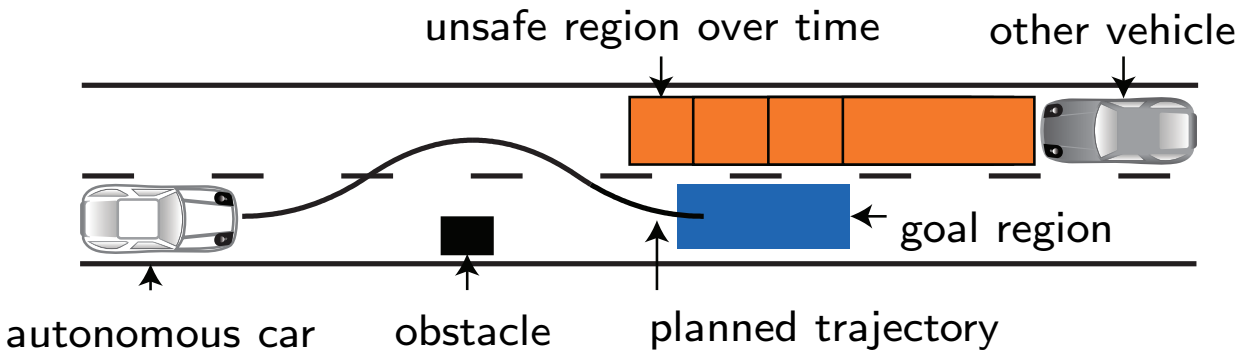
source: theguardian.com

Automated driving: classical testing [N. Kalra and S. M. Paddock (2016)]

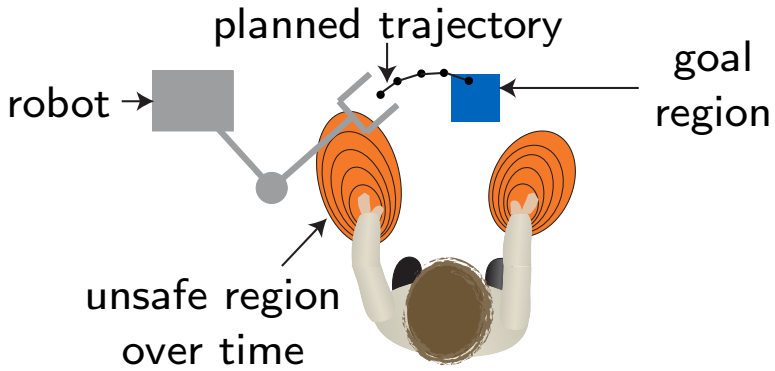
- 440 million km to demonstrate better performance than humans. (95% confidence)
- 12.5 years with 100 test vehicles continuously driving.

Possible Applications

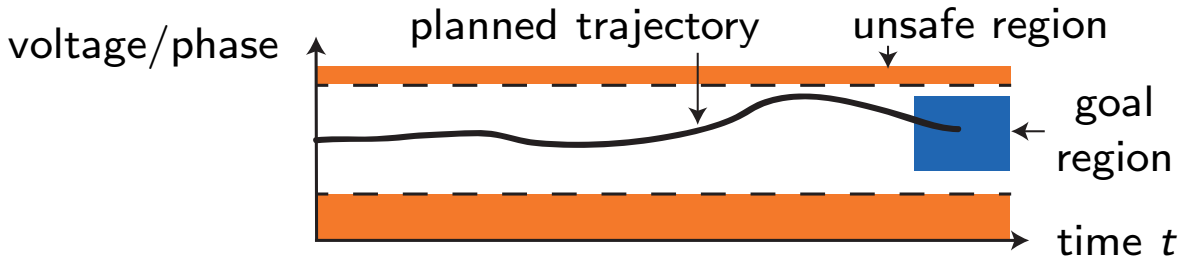
Automated Driving (main application)



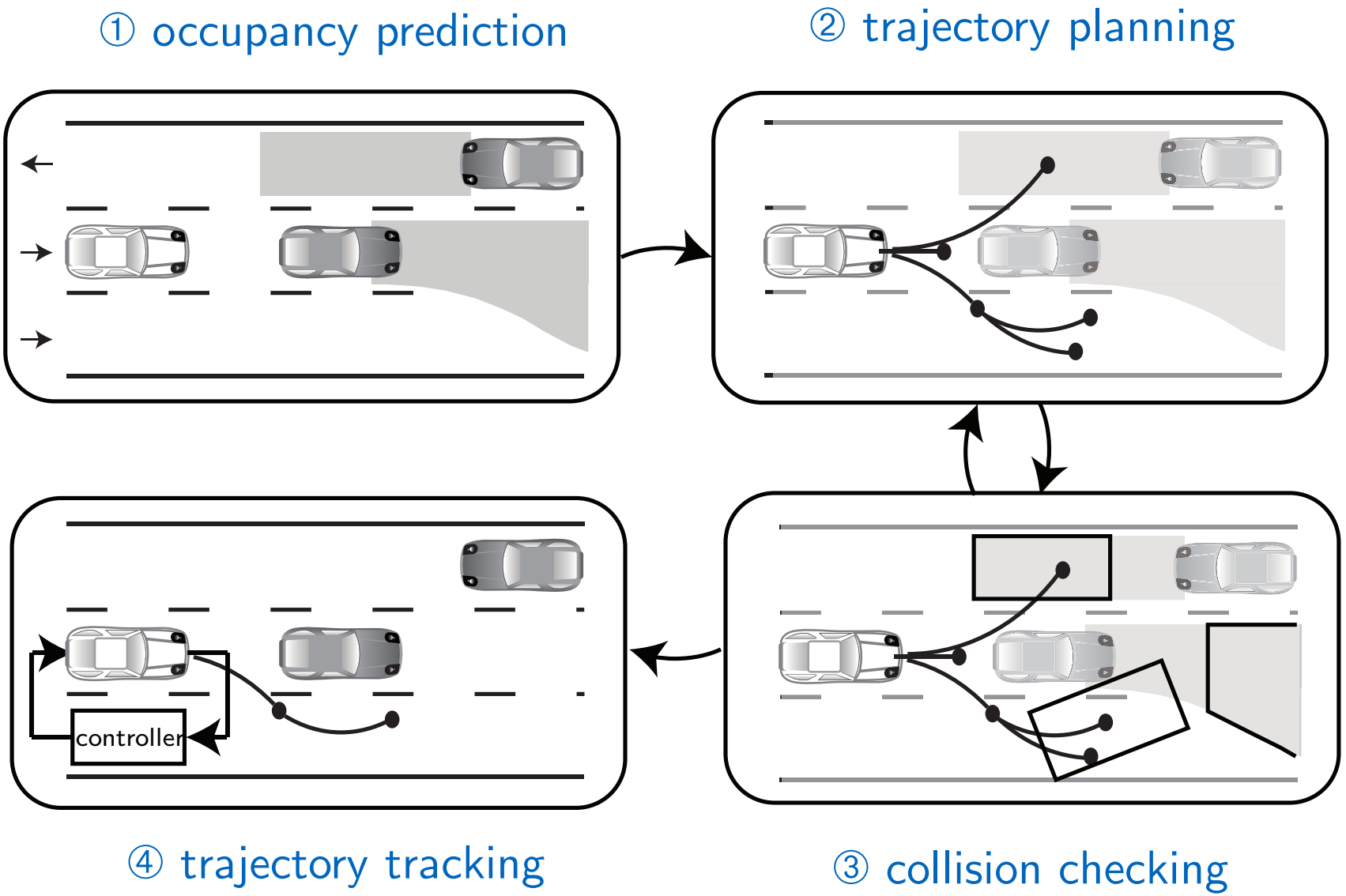
Human-Robot Coexistence



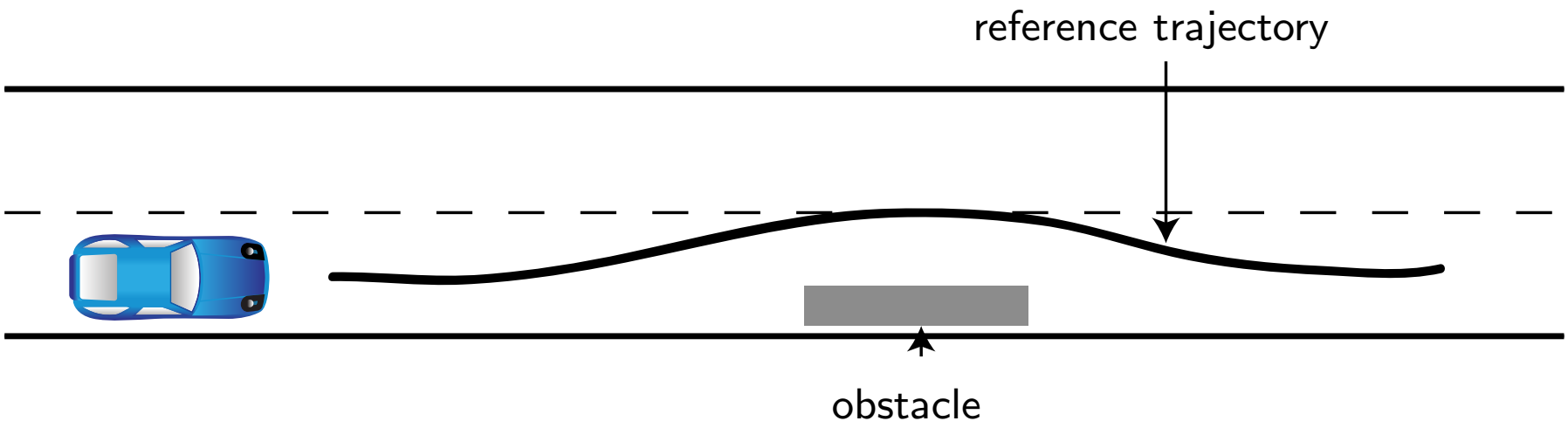
Smart Grid



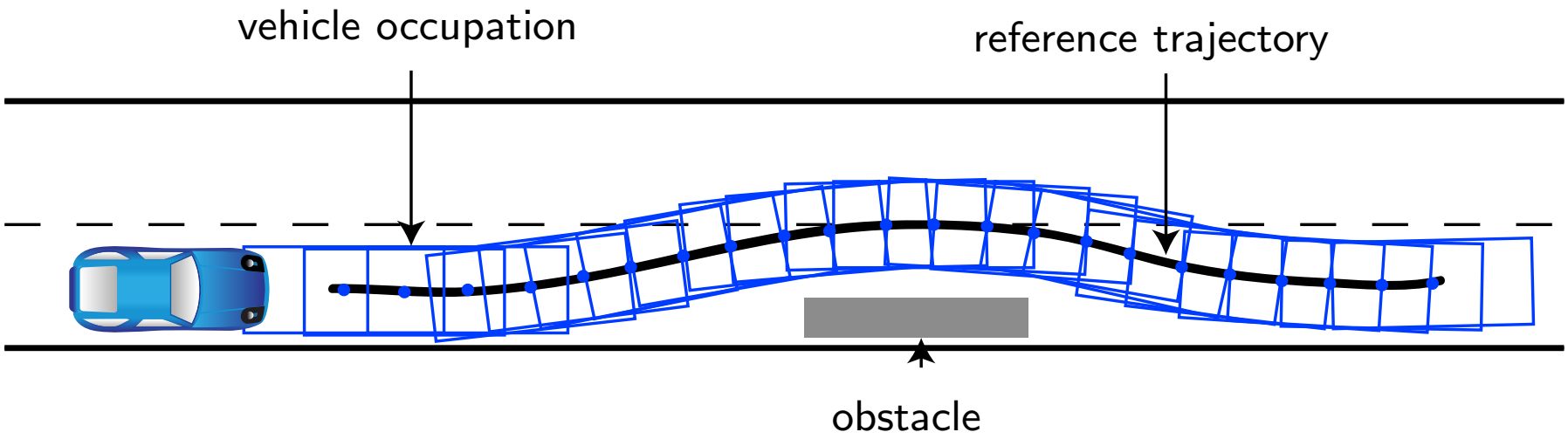
Overview of the Approach



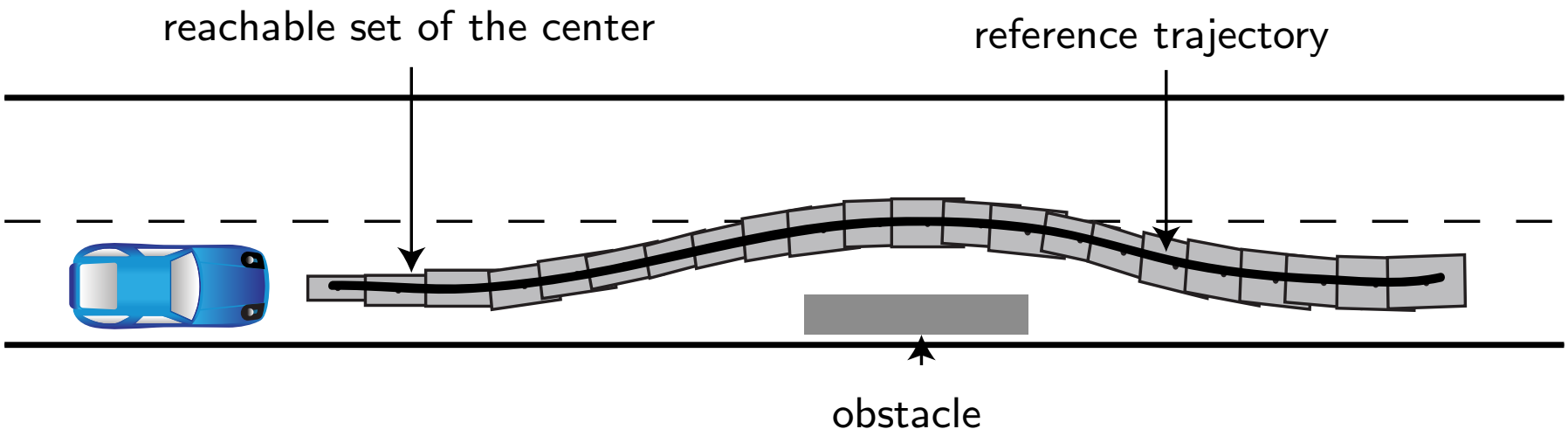
Trajectory Verification: Situation



Trajectory Verification: Standard Approach



Trajectory Verification: Considering Uncertainties



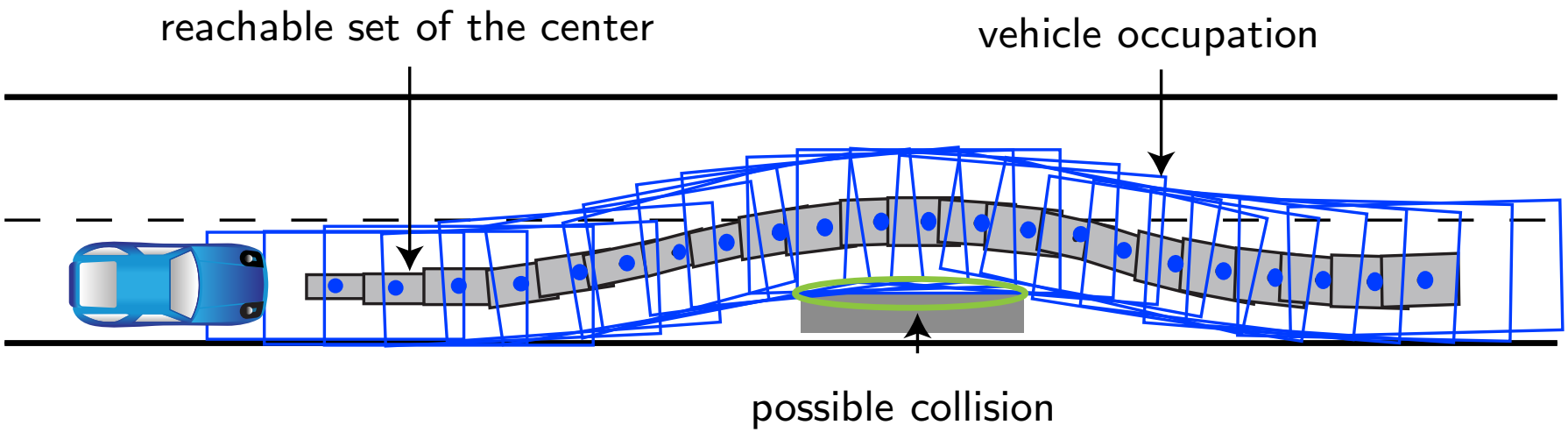
Robust Safety Problem

Is the planned maneuver of the autonomous vehicle still safe under

- uncertain initial states,
- uncertain measurements, and
- disturbances?

Objective: Guarantee safety when bounds on uncertainties are known.

Formal Verification Reveals Problems



Robust Safety Problem

Is the planned maneuver of the autonomous vehicle still safe under

- uncertain initial states,
- uncertain measurements,
- and disturbances?

Objective: Guarantee safety when bounds on uncertainties are known.

Interaction between Planning and Verification

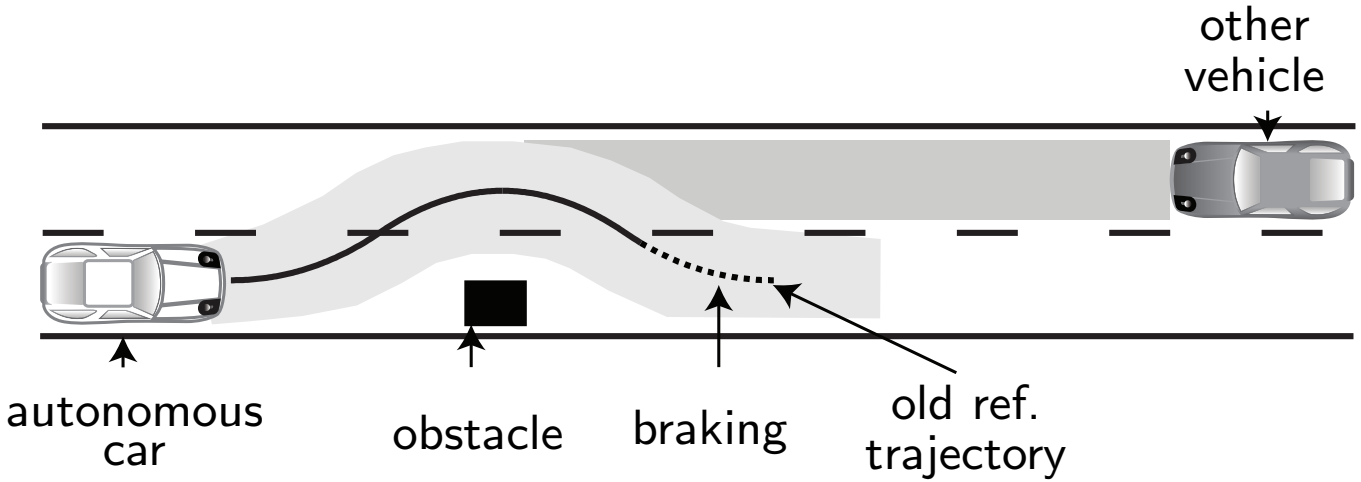
Planning Challenges

- What if the reference trajectory is unsafe?
- Is there enough time to re-plan and verify a new trajectory?
- What if a software bug or hardware failure occurs?

Planning Solution

- Plan maneuvers that are safe for all times (details later).
- Only change the previous plan if the new plan has already been verified.

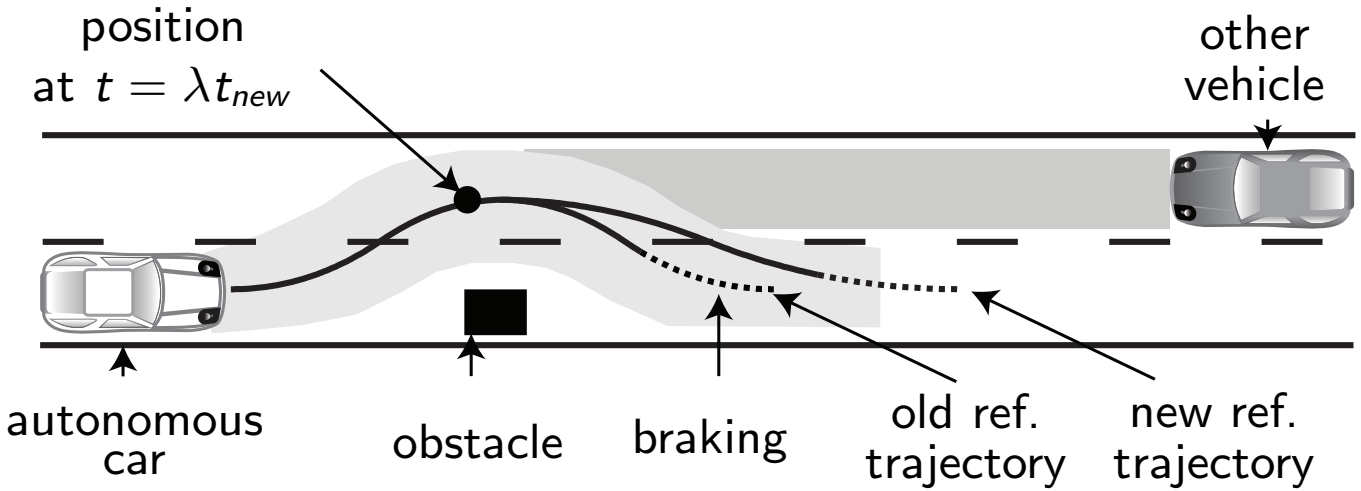
Maneuvers Verified for all Times



General Idea

- Add a braking maneuver to the end of the originally intended maneuver.
- The vehicle has to stop in a safe location (e.g., not on a railway crossing).
- The additional braking trajectory is only executed when no new safe plan is ready for execution.

Deviation from Previous Plan

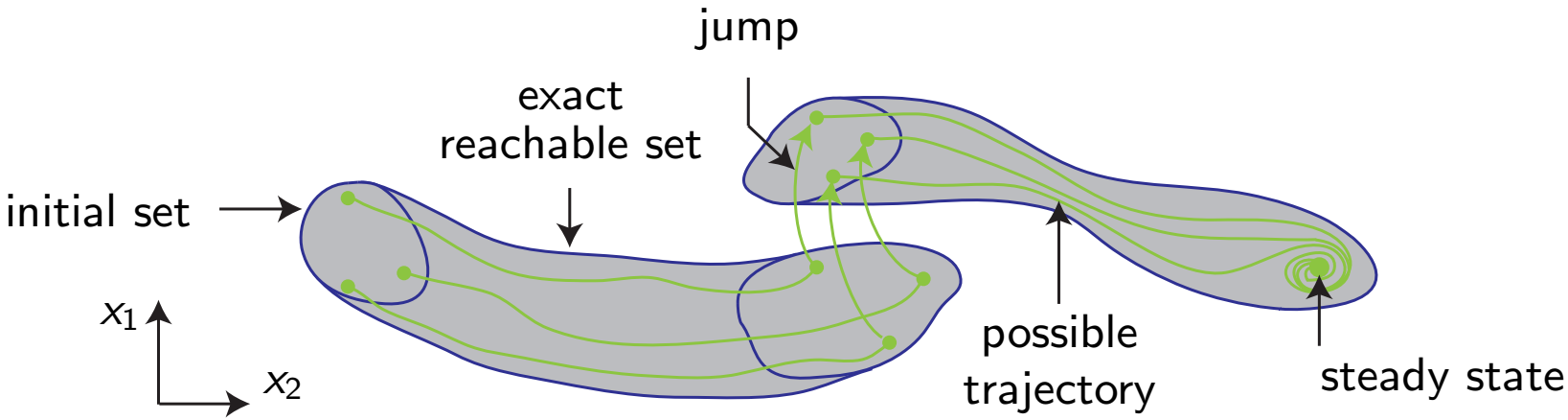


General Idea

- Change maneuver only at a point after which the new reference trajectory has been verified.
- Verification time is linear in the time horizon t_f: $t_{ver} = \lambda t_f$.

$\rightarrow$ Change previous plan at $\lambda t_{f,\text{new}}$.

Reachable Sets

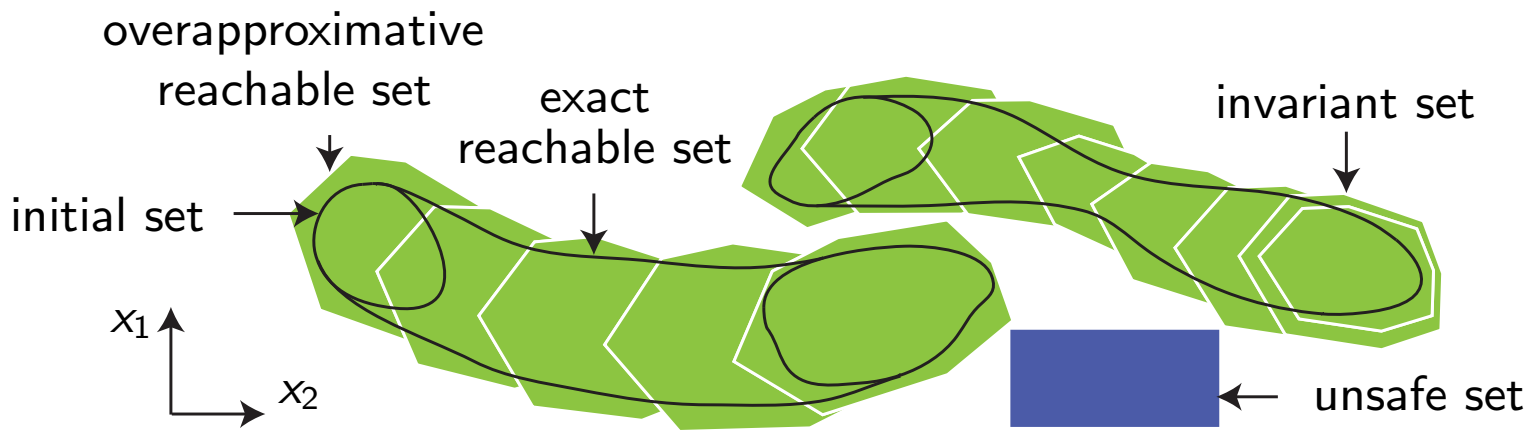


Informal Definition

A reachable set is the set of states that can be reached by a dynamical system in finite or infinite time for

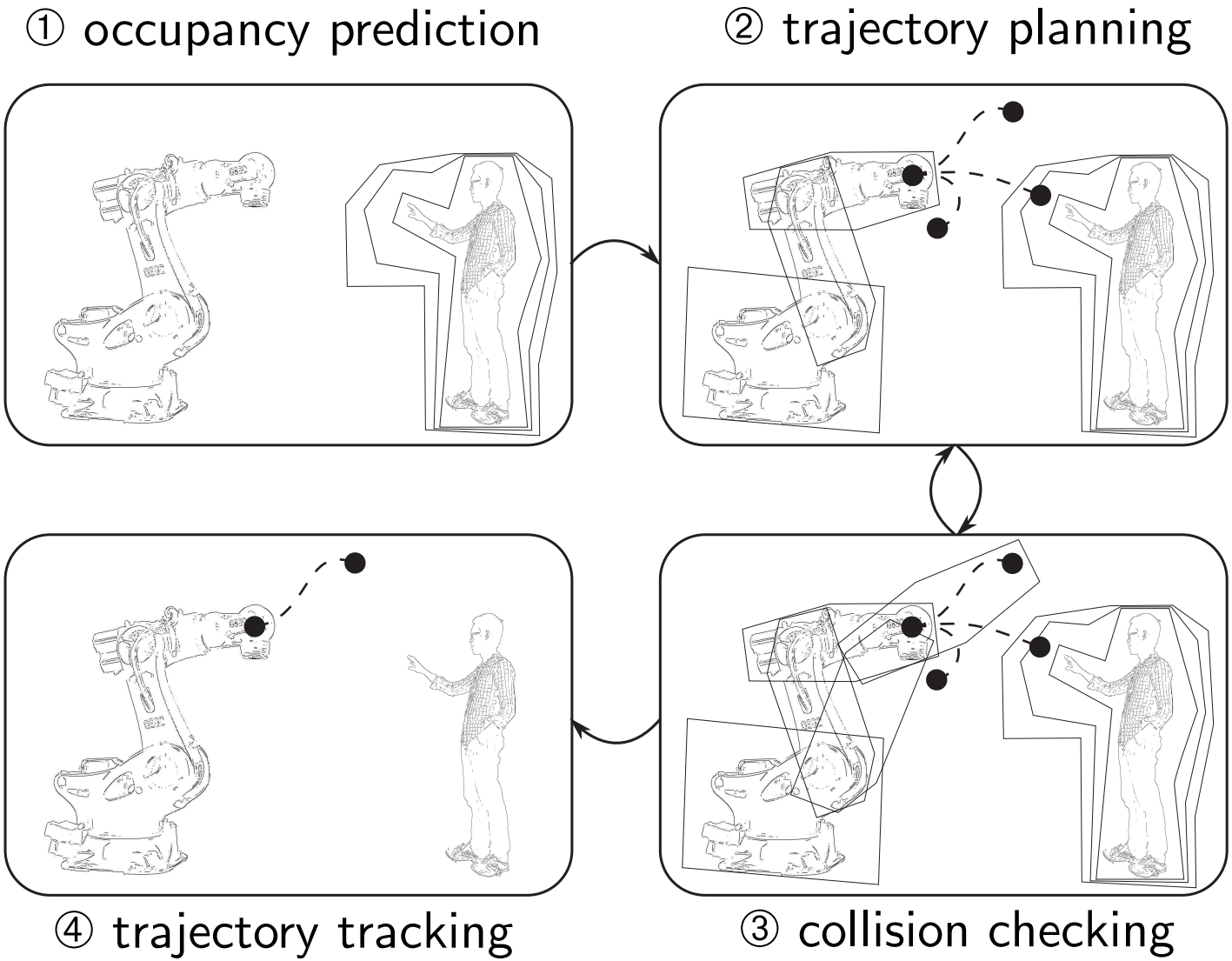
- a set of initial states,
- uncertain inputs, and
- uncertain parameters.

Overapproximative Reachable Sets



- Exact reachable set only for special classes computable
  $\rightarrow$ overapproximation computed for consecutive time intervals (taught in lecture **Cyber-Physical Systems**).
- Overapproximation might lead to spurious counterexamples.
- Simulation cannot prove correctness.

Same Approach Used in Human-Robot Co-Existence

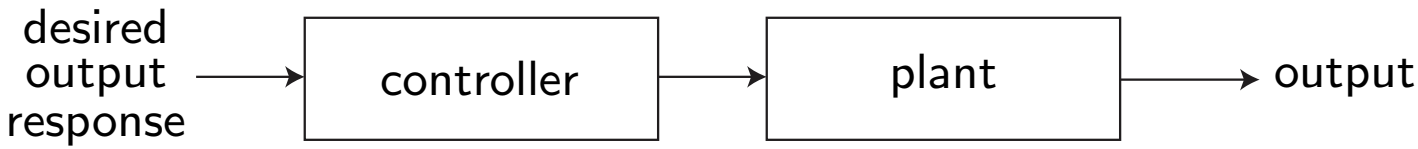


Executing Plans

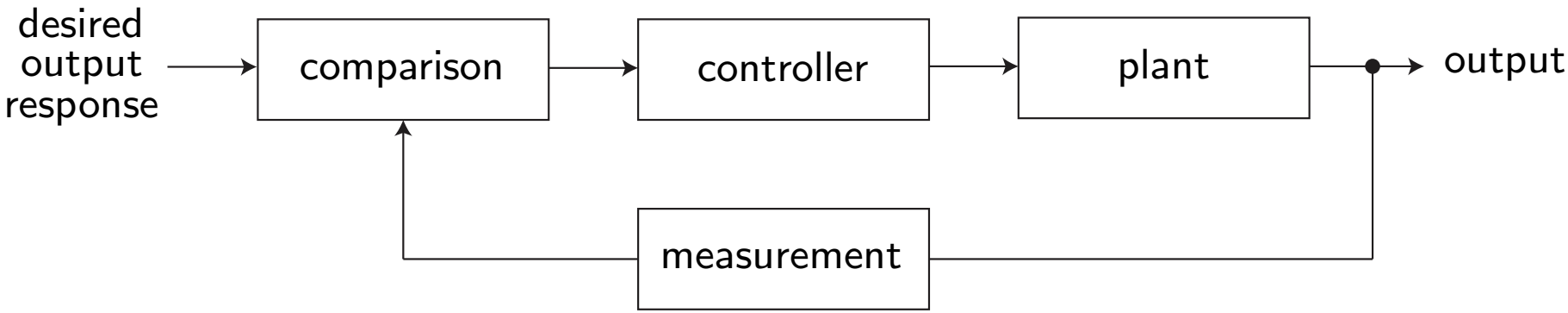
- So far, we have talked about how to *plan* motions, but not about how to *execute* them.
- We require a solution that controls the actuators such that the robot moves as planned.
- This problem is solved using **control theory**.
- A deeper insight into control of discrete, continuous, and hybrid systems can be gained in the lecture *Cyber-Physical Systems* (in summer).
- The following control topics are briefly addressed:
  - Classical PID control
  - Potential-field control
  - Reactive control
  - Learning control strategies

Open-Loop and Closed-Loop Control

Open-loop control:

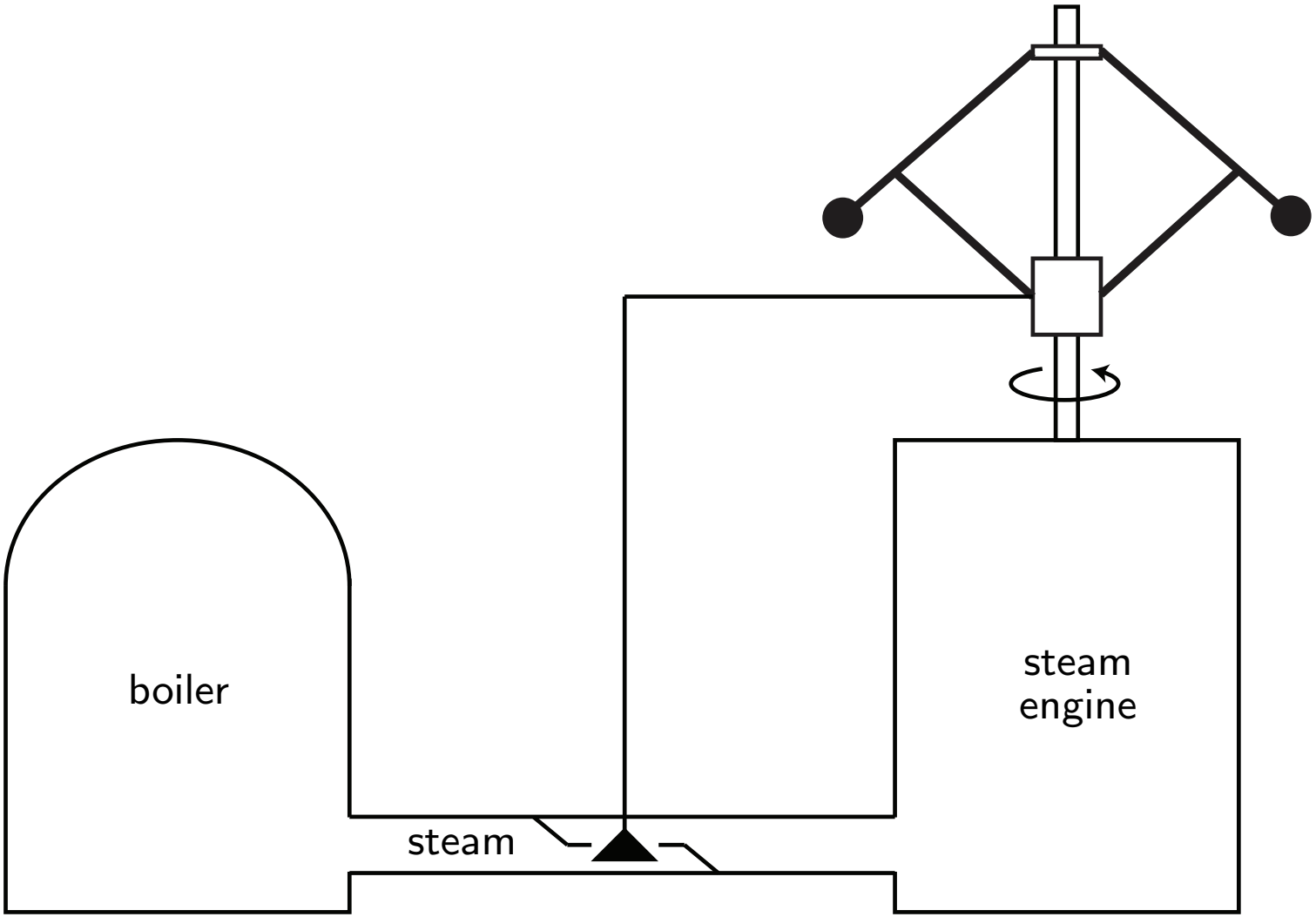


Closed-loop control:



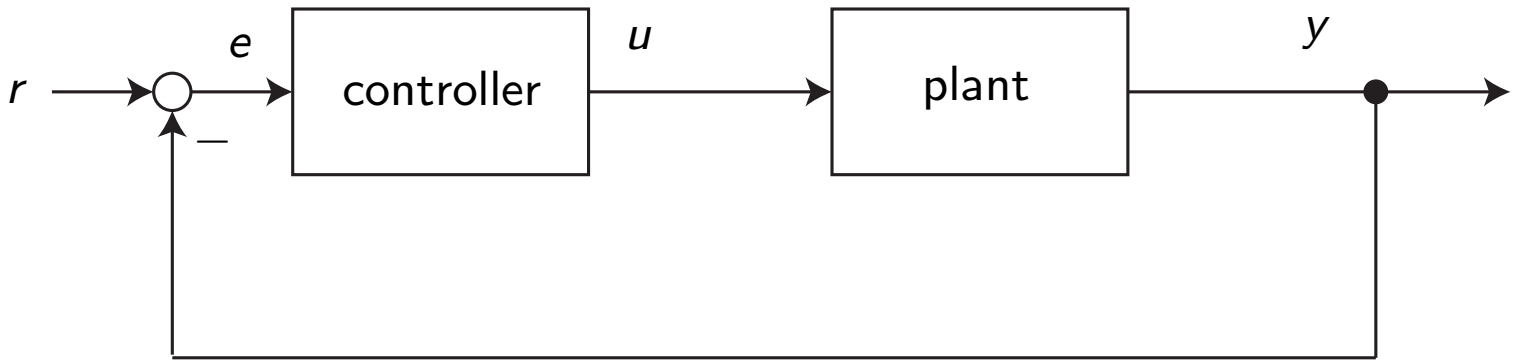
Example of Closed-Loop Control

Watt flyball governor:



Standard Control Loop

The standard control loop can be visualized with a block diagram:



Classical PID Control

Let us assume a simple system with a single continuous input $u(t)$ and a simple continuous output $y(t)$. The error is denoted by $e(t) = y(t) - r(t)$ and the letters *PID* stand for proportional (P), derivative (D), and integral (1).

P controller

$$u(t) = K_P\, e(t)$$

PD controller

$$u(t) = K_P\, e(t) + K_D \frac{\mathrm{d}e(t)}{\mathrm{d}t}$$

PID controller

$$u(t) = K_P\, e(t) + K_I \int e(t)\mathrm{d}t + K_D \frac{\mathrm{d}e(t)}{\mathrm{d}t}$$

PID Controller: Discussion

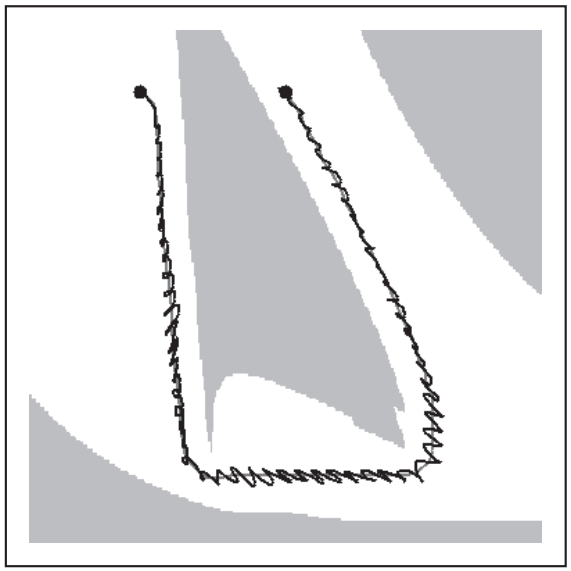
The control gains can be interpreted as follows:

- **Proportional term:** The larger the error $e(t)$, the larger the control action $K_P e(t)$. The proportional gain considers the **present** error.
- **Integral term:** This term integrates the error $e(t)$ until $e(t)$ becomes 0. This is why the integral part ensures stationary behavior when the closed-loop system is stable. This part considers **past** errors.
- **Differential term:** The control action of this term depends on the change of the error $\dot{e}(t)$. The differential part reacts to errors developing in the **future**.

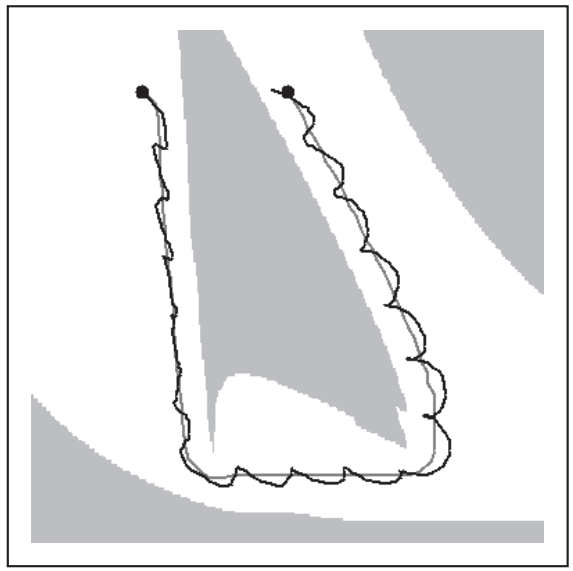
Bad parameterization of a PID controller can lead to **instability**, i.e., the error $e(t)$ cannot be made arbitrarily small for arbitrarily small perturbations.

PID Controller: Performance

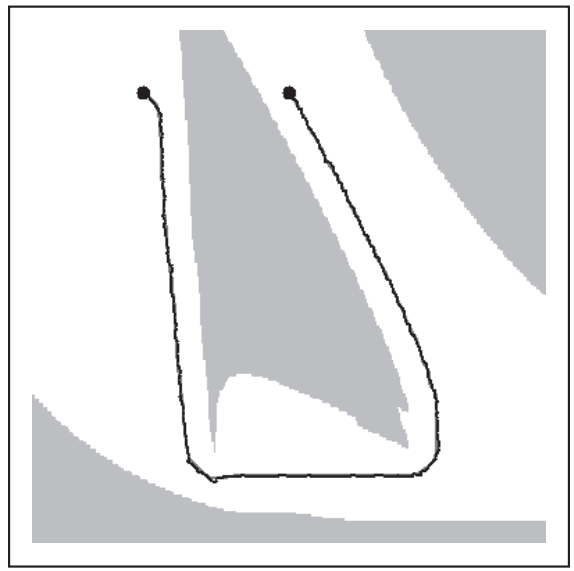
Robot arm example using P, PD, and PID control:



P control with $K_P = 1.0$.



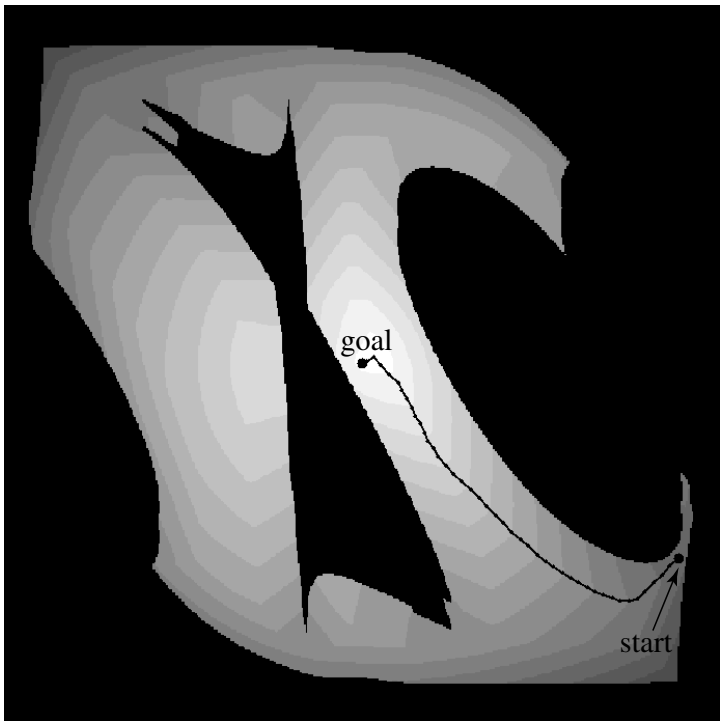
P control with $K_P = 0.3$.



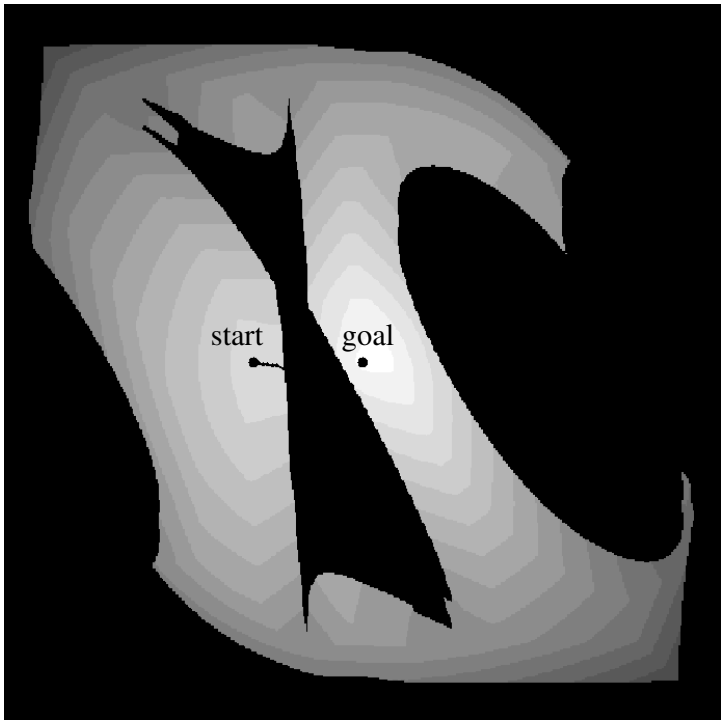
PD control with $K_P = 0.3$ and $K_D = 0.8$.

Potential-Field Control

- We introduced potential fields as an additional cost function for planning.
- Potential fields can directly generate movement (path planning skipped).
- We require attractive forces for the goal and repellent forces for obstacles.
- **Big advantage:** Easy to compute, no real-time issues!
- **Big issue:** Robots can get stuck in local minima!



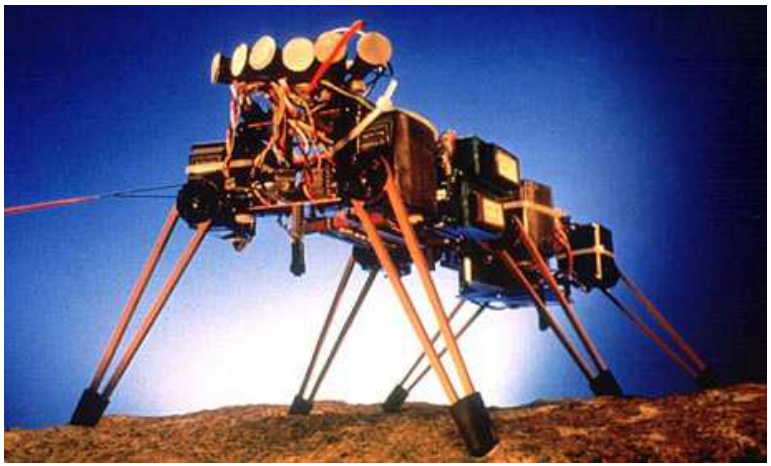
Successful path.



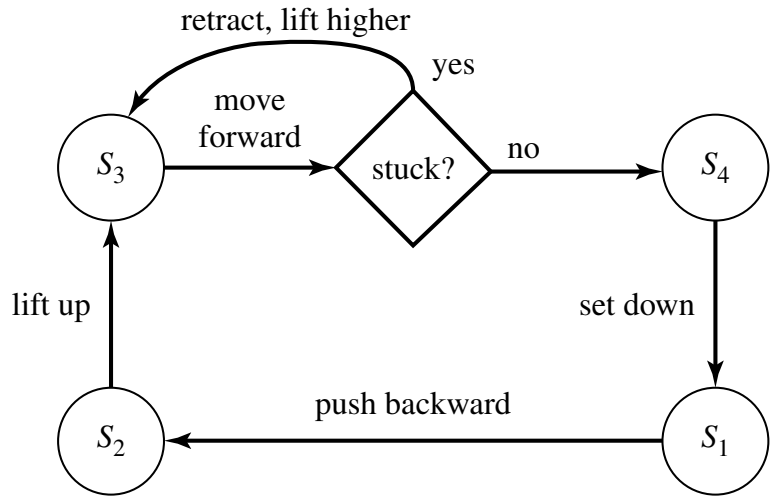
Stuck in local minimum.

Reactive Control

- PID control requires a reference and potential fields require a model of the environment – sometimes none of this is available.
- Reactive control uses no models so that it has a reflex agent architecture.
- A good application of reflex control is movement in unknown environments, such as the leg control of a legged robot:



legged robot Genghis.



Augmented finite state machine as reactive controller.

- PID control is often integrated into reactive control to improve performance.

Learning Control Strategies

- One obstacle for applying efficient controllers is that an accurate model of the system is required.
- This is often hard to achieve, e.g., a helicopter is challenging to model due to its highly nonlinear aerodynamics.
- The solution is to let an expert fly the helicopter and learn the model.
- This made it possible to autonomously perform the challenging flip maneuver of a radio controlled miniature helicopter:

Software Architecture

Definition

Software architecture is the high-level structure of a software system. An architecture often includes languages and tools for writing programs, as well as an overall programming philosophy.

- Robotic software architectures must decide how to combine reactive control and model-based deliberative planning.
- Reactive control is sensor-driven and appropriate for low-level decisions in real time.
- Plausible solutions at the global level are typically achieved by deliberate planning.
- Consequently, most robot architectures use reactive techniques at the lower levels of control and deliberative techniques at the higher levels.
- Previous example: We combined a (reactive) PD controller with a (deliberate) path planner for the robotic arm.

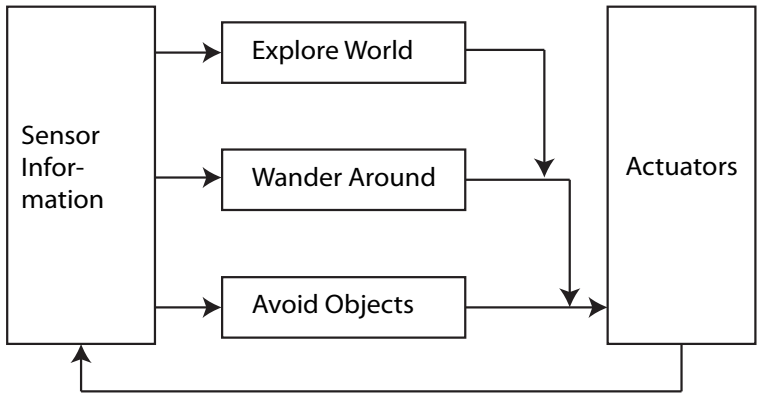
Subsumption Architecture (Brooks, 1986)

Philosophy

- Couples sensory information to action selection in a bottom-up fashion.
- Sub-behaviors are organized into a hierarchy of layers and higher levels are able to subsume lower levels in order to create viable behavior.

Primary Use

- Simple Tasks solvable by reactive behavior.
- Problems which do not change over time.



Three-Layer Architecture

Philosophy

Combination of reaction with deliberation in three layers:

- **Reactive layer:** Low-level control (tight sensor-action loop). Cycle time: on the order of milliseconds.
- **Executive layer:** Accepts plans from the deliberative layer and translates it into reference values for the reactive layer. Typical tasks: Integrating sensor information to estimate the state, optimizing reference paths. Cycle time: on the order of seconds.
- **Deliberative layer**: Creates global solutions using planning. Cycle time: on the order of minutes.

Primary Use

Applicable to most modern-day robot systems. The number of layers might vary.

Pipeline Architecture

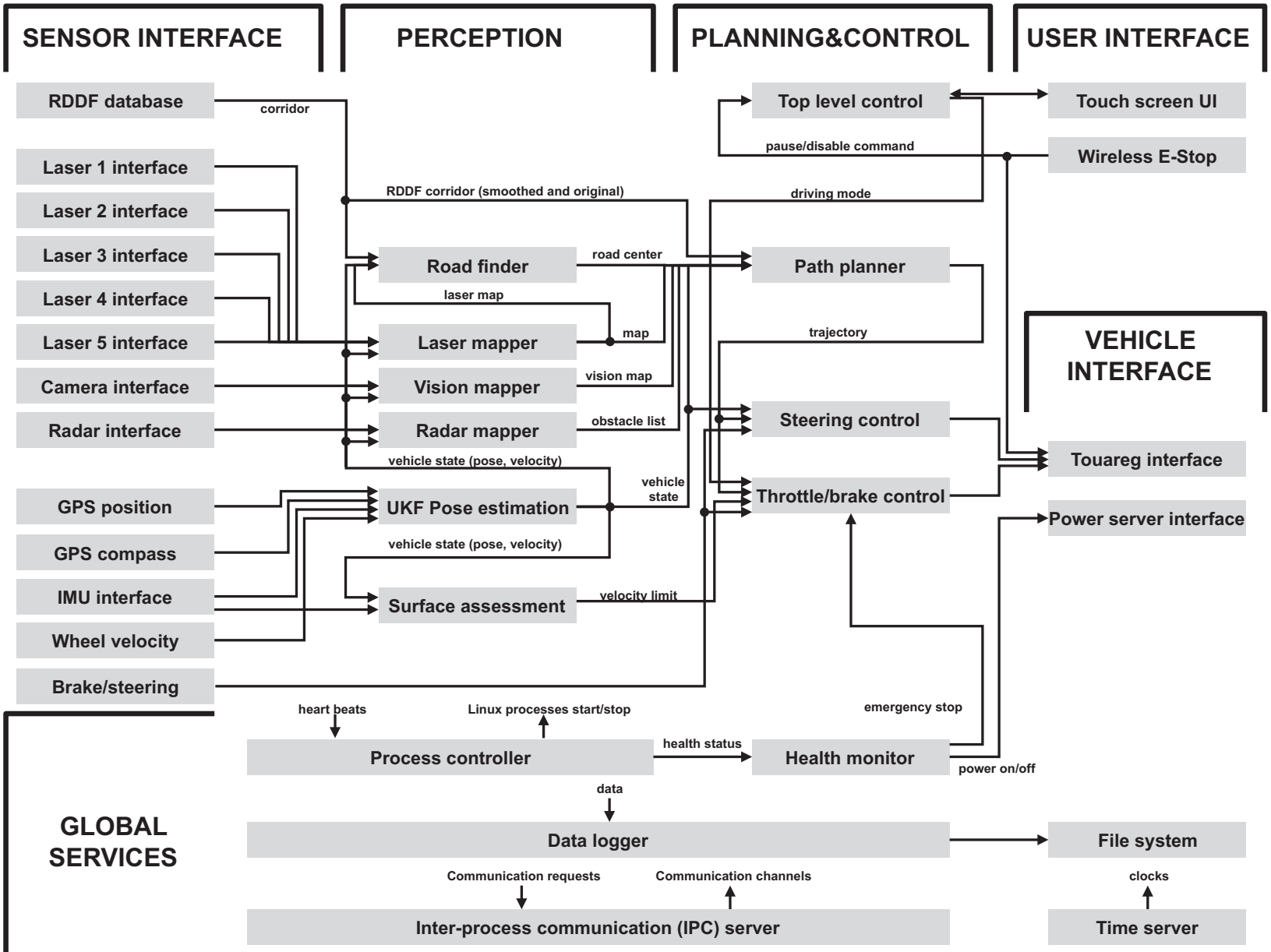
Philosophy

- Just like the subsumption architecture, the pipeline architecture executes multiple processes in parallel.
- However, the specific modules in this architecture resemble those in the three-layer architecture.
- Processes in the pipeline architecture run asynchronously.

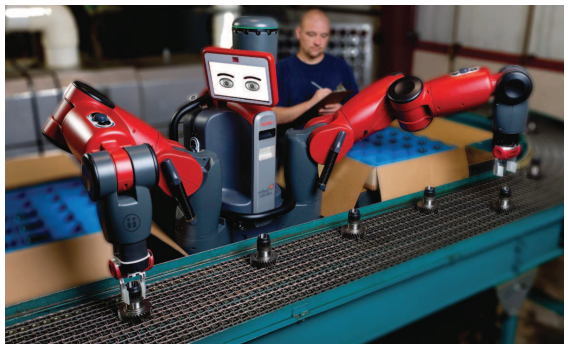
Primary Use

Applicable to large systems with high performance demands, e.g., autonomous cars. An example is shown on the next slide.

Pipeline Architecture: Autonomated Car



Applications I



Industry

source: Rethink Robotics

Agriculture

source: Kesmac

Logistics

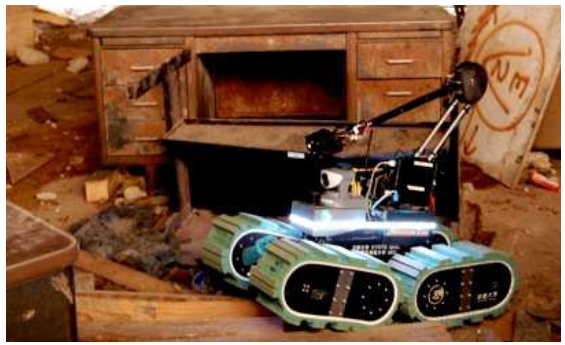
source: Port of Hamburg

Automated vehicles

source: Carnegie Mellon University

Health care

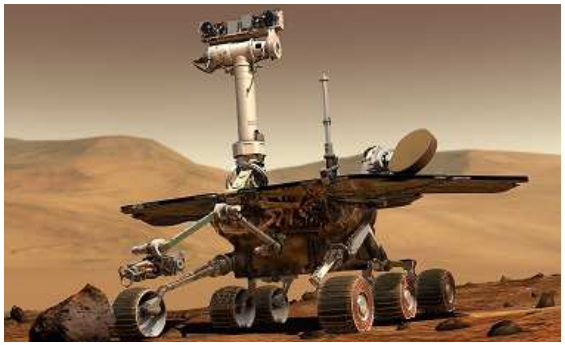
source: daVinci



Hazardous environments

source: Kyoto University

Applications II

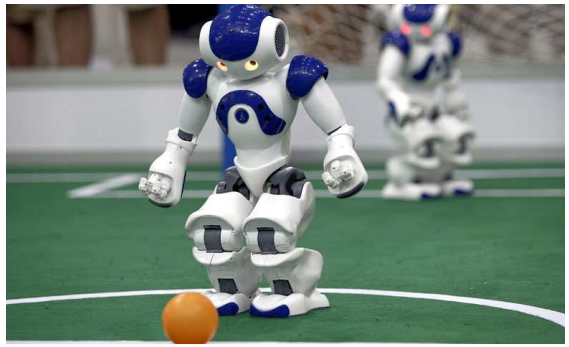


Exploration

source: NASA

Personal service

source: Bosch



Entertainment

source: Aldebaran Robotics

Therapy

source: PARO Robots

Human augmentation

source: ESA

Telepresence

source: Anybot

Summary

- Robots are equipped with **sensors** for perceiving their environment and **effectors** with which they can assert physical forces on their environment.
- Primary robotic categories are **manipulators**, **mobile robots**, and **mobile manipulators**.
- Primary sensor categories are **passive sensors**, **active sensors**, and **proprioceptive sensors**.
- Robotic perception concerns itself with estimating decision-relevant quantities from sensor data, requiring an **internal representation** of the environment (using Kalman filters and particle filters).
- The planning of robot motion is often done in **configuration space**.
- High-level tasks typically require **planning**, while **low-level control** makes sure that sub-tasks are adequately executed.
- Robotic architectures help organizing the software (we discussed the subsumption, three-layer, and the pipeline architecture)