TDDC17
Robotics/Perception II

Mariusz Wzorek
IDA/AIICS

LINKÖPING UNIVERSITY

1

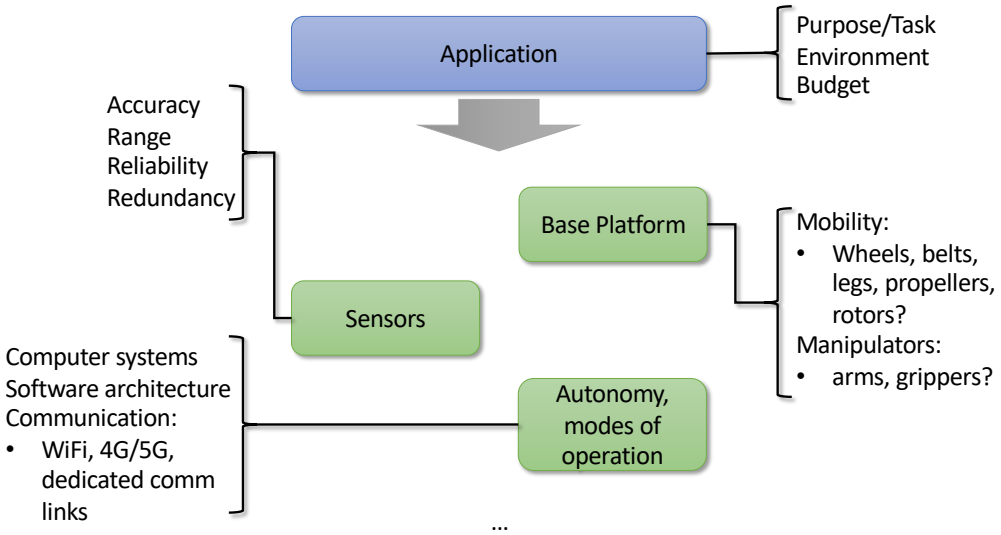
Outline

- Sensors - summary
- Computer systems
- Robotic architectures
- Navigation:
  - Mapping and Localization
  - Motion planning
  - Motion control

LINKÖPING UNIVERSITY

2

Robotics – application perspective



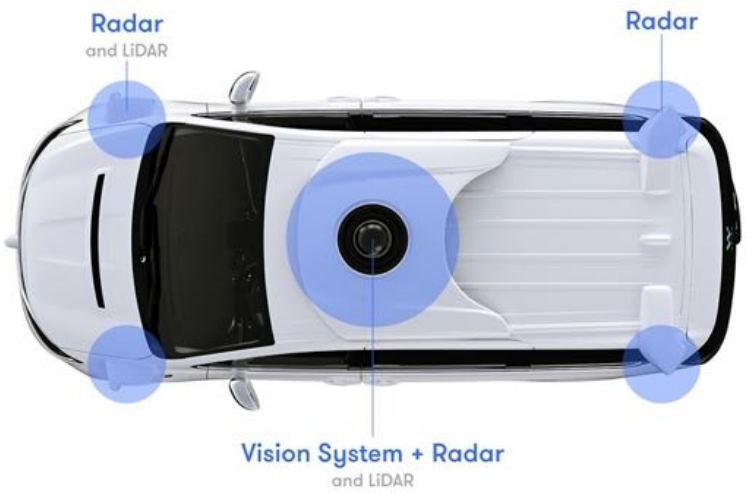
Sensors

Summary of most commonly used sensors for mobile robots.

	Any weather	Any light	Detection in 15m	Fast response	Weight	Affordable
CCD Camera/stereo/ Omnidirectional/o. flow			👍	👍	👍	👍
Ultrasonic		👍		👍	👍	👍
Scanning laser (LiDAR, LRF)		👍	👍	👍		👍
3D Scanning laser		👍	👍			👍*
Millimeter Wave Radar	👍	👍	👍	👍		

* scanning laser on a tilting unit

Sensors cont'd



LINKÖPING UNIVERSITY

waymo.com

5

Sensors cont'd

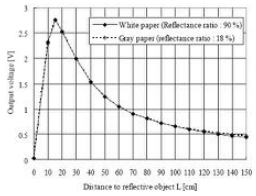
LINKÖPING UNIVERSITY

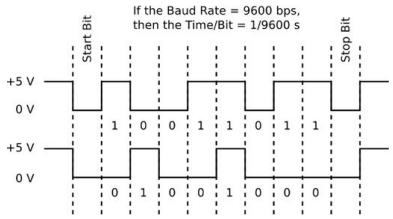
tesla.com

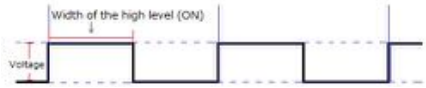
6

Sensors - common interfaces

- analog - voltage level
- digital:
  - serial connections e.g.:
    - RS232
    - I2C
    - SPI
    - USB
  - Ethernet
  - pulse width modulation - PWM
  - ...







Outline

- Sensors - summary
- Computer systems
- Robotic architectures
- Navigation:
  - Mapping and Localization
  - Motion planning
  - Motion control

Computer systems

Many challenges and trade-offs!

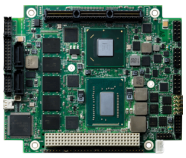
- power consumption
- size & weight
- computational power
- robustness
- different operational conditions moisture, temperature, dirt, vibrations, etc.
- cloud computing?

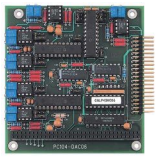
Computer systems cont'd

PC104 - standardised form factor

- industrial grade
- relatively small size
- highly configurable
- >200 vendors world-wide
- variety of components
- variety of processors

~10x10cm

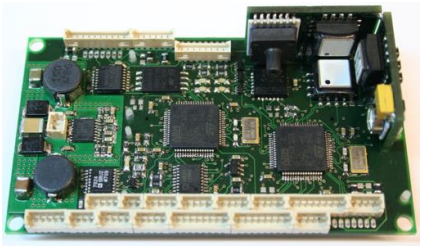




Computer systems cont'd

Custom made embedded systems

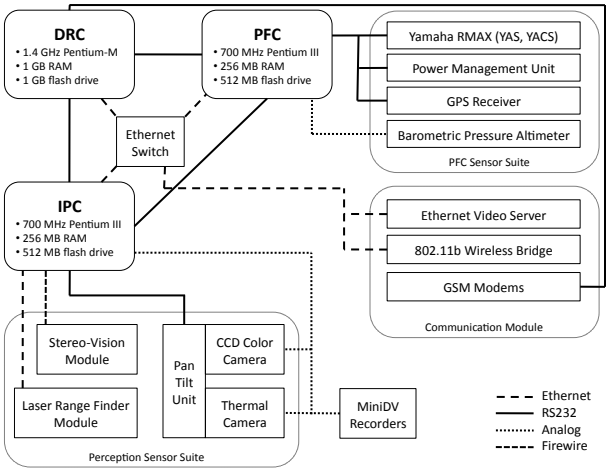
- with integrated sensor suite
- micro scale, light weight
- fitted for platform design



- fitted for platform design

Computer systems cont'd

Example system design:

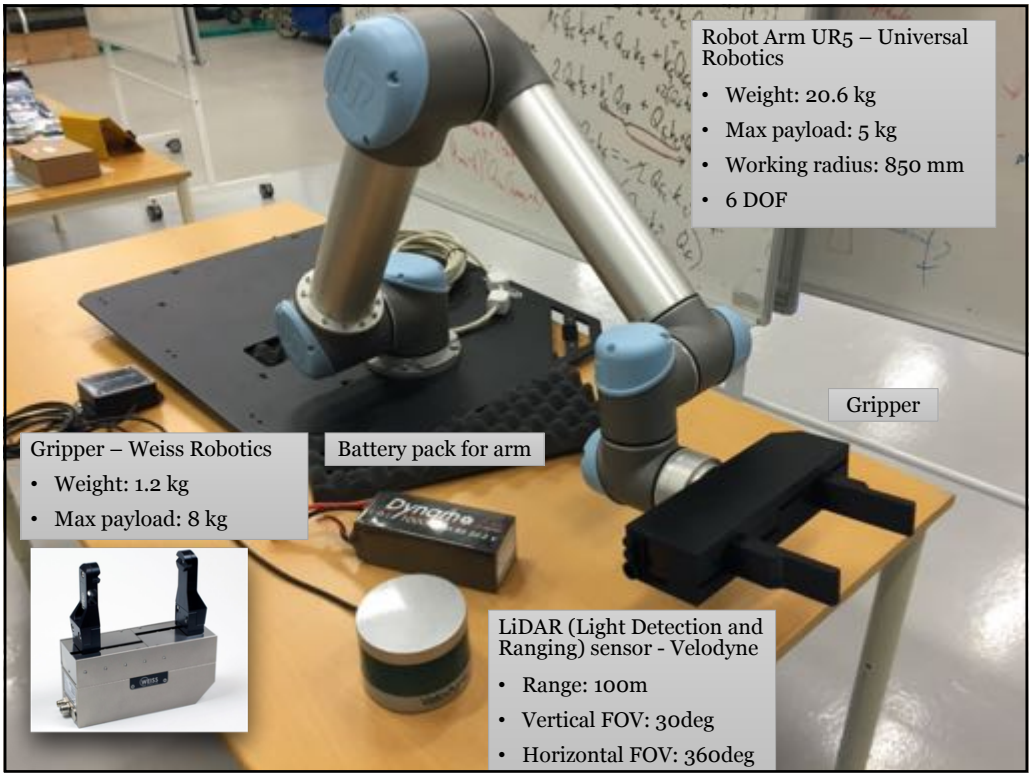


Ground Robot at AIICS LiU

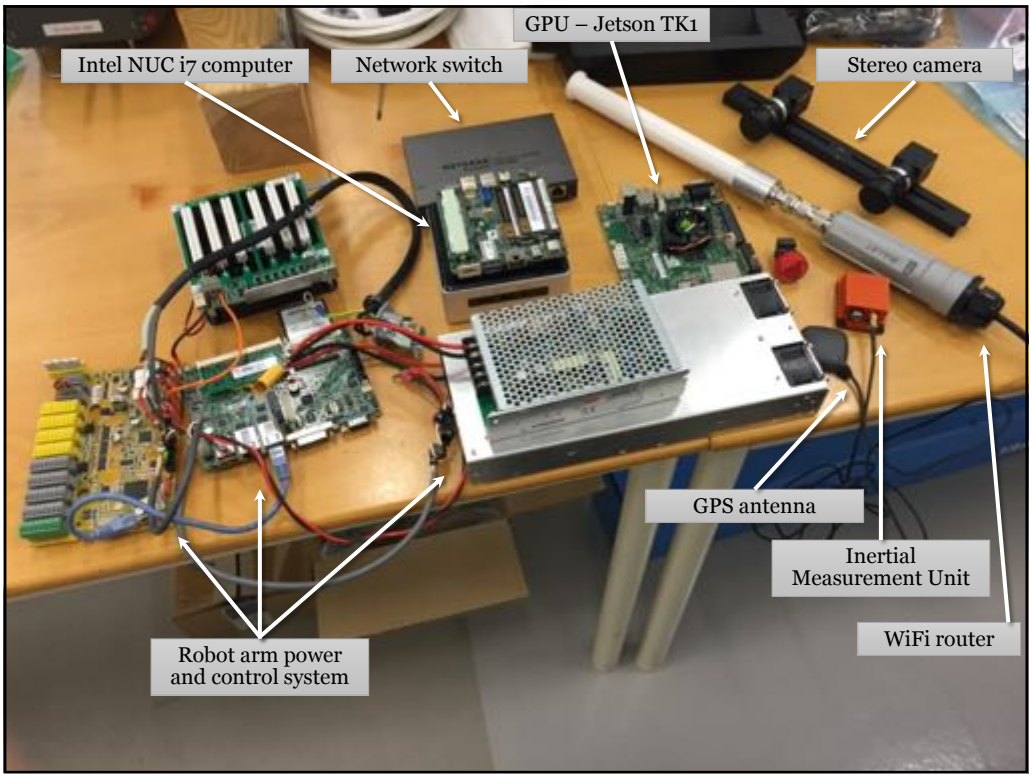
LINKÖPING UNIVERSITY

13

14


Robot Arm UR5 – Universal Robotics
• Weight: 20.6 kg
• Max payload: 5 kg
• Working radius: 850 mm
• 6 DOF
Gripper
Gripper – Weiss Robotics
• Weight: 1.2 kg
• Max payload: 8 kg
Battery pack for arm
LiDAR (Light Detection and Ranging) sensor - Velodyne
• Range: 100m
• Vertical FOV: 30deg
• Horizontal FOV: 360deg

15


GPU – Jetson TK1
Intel NUC i7 computer
Network switch
Stereo camera
GPS antenna
Inertial Measurement Unit
WiFi router
Robot arm power and control system

16

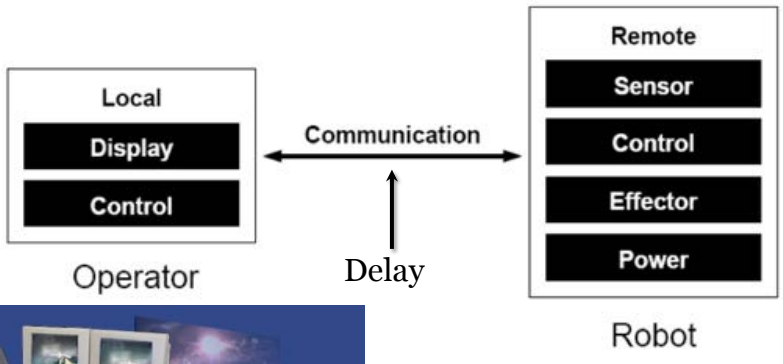
Outline

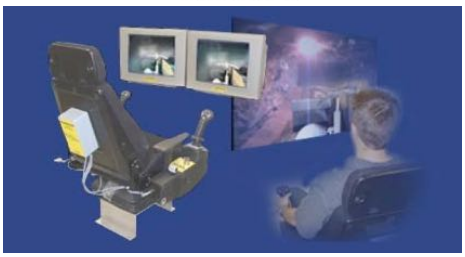
- Sensors - summary
- Computer systems
- Robotic architectures
- Navigation:
  - Mapping and Localization
  - Motion planning
  - Motion control

LINKÖPING UNIVERSITY

17

Telesystems/Telemanipulators





LINKÖPING UNIVERSITY

18

Telepresence

Experience of being fully present at live event without actually being there

- see the environment through robot's cameras
- feel the surrounding through robot sensors

- Realised by for example:
  - stereo vision
  - sound feedback
  - cameras that follow the operator's head movements
  - force feedback and tactile sensing
  - VR system

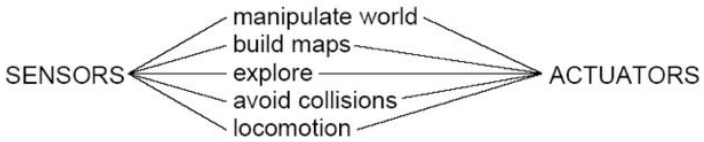
Reactive systems

Robot responses **directly to sensor stimuli**.

Intelligence emerges from combination of simple behaviours.

For example - subsumption architecture (Rodney Brooks, MIT):

- concurrent behaviours
- higher levels subsume behaviours of lower layers
- no internal representation
- finite state machines

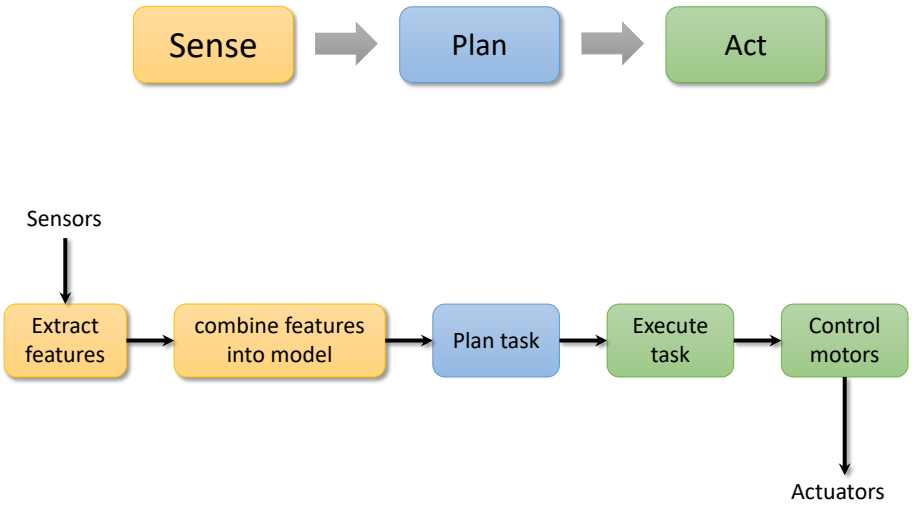


https://www.youtube.com/watch?v=9u0CIQ8P_qk

Limitations of Reactive Systems

- Agents without environment models must have sufficient information available from local environment
- If decisions are based on *local* environment, how does it take into account *non-local* information (i.e., it has a "short-term" view)
- Difficult to make reactive agents that learn
- Since behaviour emerges from component interactions plus environment, it is hard to see how to *engineer* specific agents (no principled methodology exists)
- It is hard to engineer agents with large numbers of behaviours (dynamics of interactions become too complex to understand)

Hierarchical Systems



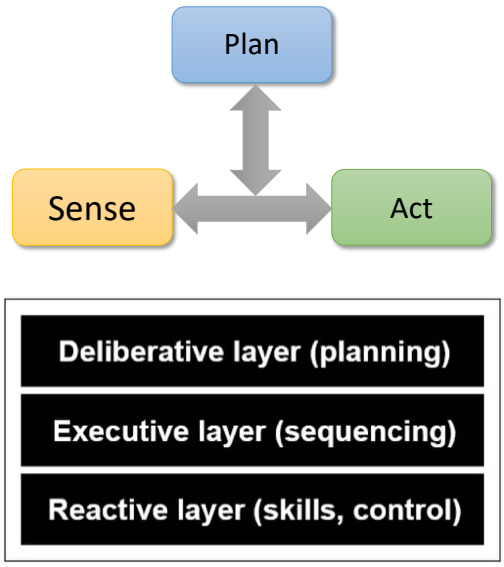
Hierarchical Systems (Shakey Example)

Shakey (1966 - 1972)

- Developed at the Stanford Research Institute
- Used STRIPS planner (operators, pre and post conditions)
- Navigated in an office environment, trying to satisfy a goal given to it on a teletype. It would, depending on the goal and circumstances, navigate around obstacles consisting of large painted blocks and wedges, push them out of the way, or push them to some desired location.
- Primary sensor: black-and-white television camera
- Sting symbolic logic model of the world in the form of first order predicate calculus
- Very careful engineering of the environment

https://www.youtube.com/watch?v=GmU7SimFkpU

Hybrid systems (three-layer architecture)



Hybrid systems (Minerva Example)

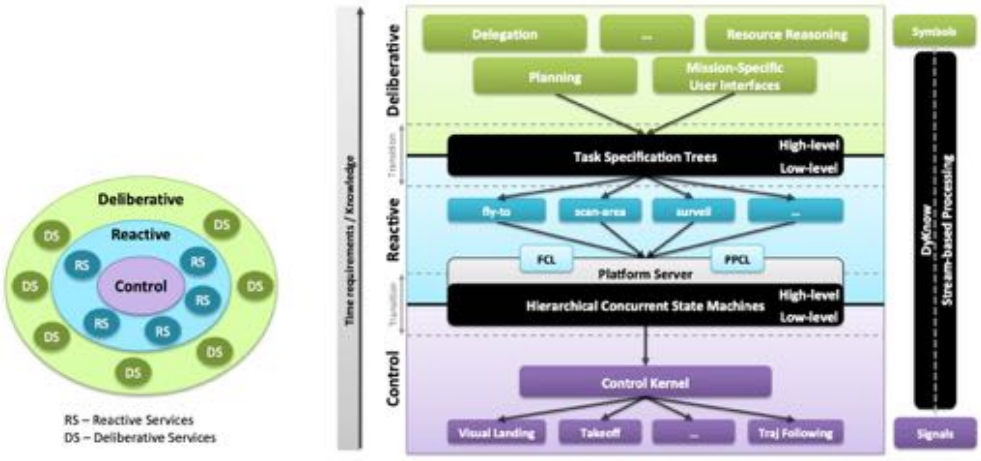
Tour guide at the Smithsonian's National Museum of American History (1998)

high-level control and learning (mission planning, scheduling)
human interaction modules ("emotional" FSA, Web interface)
navigation modules (localization, map learning, path planning)
hardware interface modules (motors, sensors, Internet)

Table 1: Minerva's layered software architecture

http://www.cs.cmu.edu/~thrun/movies/minerva.mpg

Hybrid systems (HDR3 – AIICS/IDA @ LiU Example)



(a) The concentric view (b) The layered view

Outline

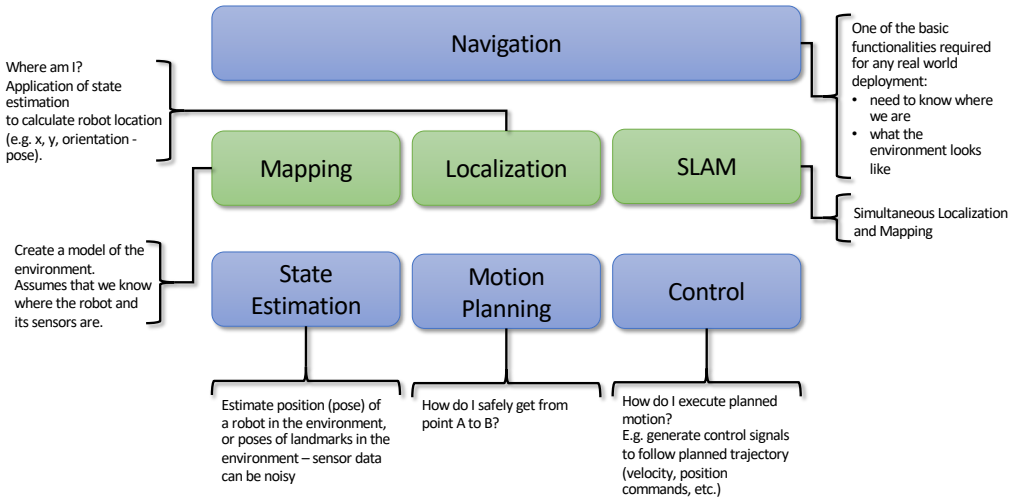
- Sensors - summary
- Computer systems
- Robotic architectures
- Navigation:
  - Mapping and Localization
  - Motion planning
  - Motion control

LINKÖPING UNIVERSITY

27

Navigation

Mapping – Localization – SLAM – State Estimation – Control



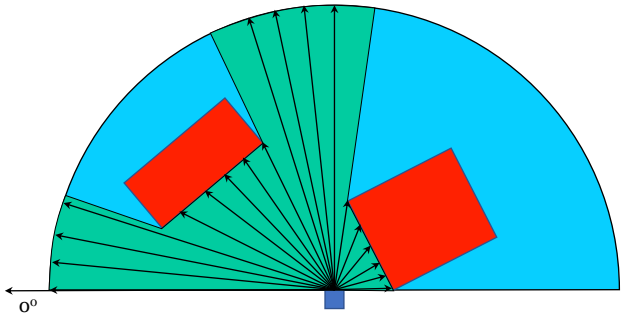
LINKÖPING UNIVERSITY

28

LiDAR – recap

Active sensor based on *time of flight* principle

- emits light waves from a laser
- measures the time it took for the signal to return to calculate the distance

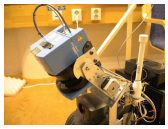


View from top

LiDAR – recap cont'd

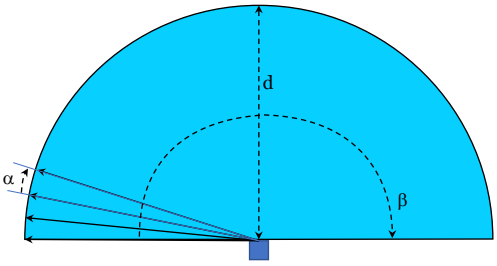
SICK LMS (single line sensor):

- Range (d): 80m
- Field of View - horizontal (α): 0°-180°
- Angular resolution - horizontal (β): 0.25°-1°

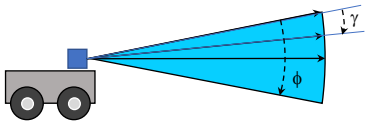


Velodyne Puck (multi-line sensor):

- Range (d): 100m
- Field of View - horizontal (α): 360°
- Angular resolution - horizontal (β): 0.1°-0.4°
- Field of View - vertical (γ): ±15°
- Angular resolution - vertical (ϕ): 2°



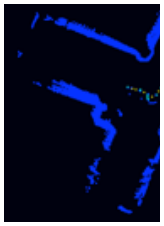
View from top



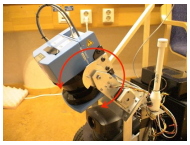
View from side

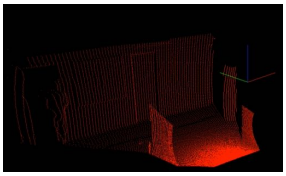
LiDAR – recap cont'd

SICK LMS stationary
range data at certain height

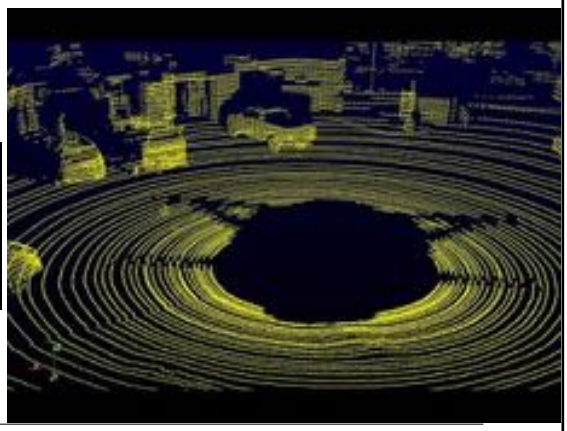


SICK LMS with tilt mechanism





Velodyne stationary



LINKÖPING UNIVERSITY

Velodyne:
https://www.youtube.com/watch?v=WPtHRVdWXSI
https://www.youtube.com/watch?v=KxWrWPpSE8I

31

Mapping

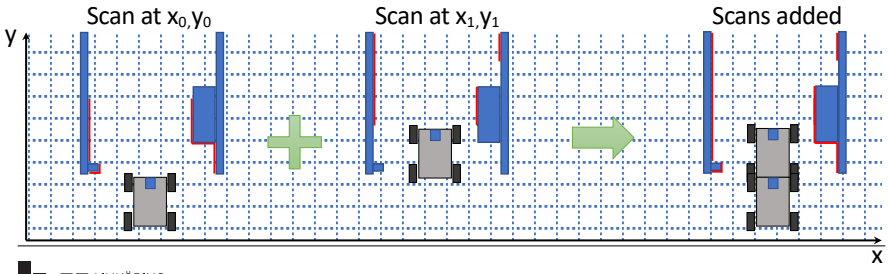
Assume we use a mobile robot platform equipped with a LiDAR sensor

Simple idea: combine measurement of robot motion with LiDAR sensor readings

Measurement of robot motion - state estimation, e.g. odometry, dead reckoning

- Simple odometry - wheel encoders that calculate how many times wheels turned
- Visual odometry etc. – e.g. optical flow sensor, or use sensor fusion to produce more accurate estimation of motion

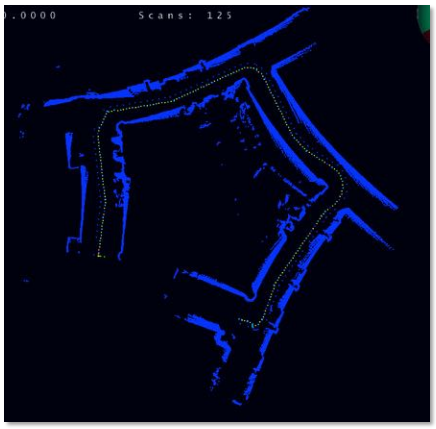
There will always be a measurement error when calculating the state - depending on the technique/sensor used it can be smaller or larger

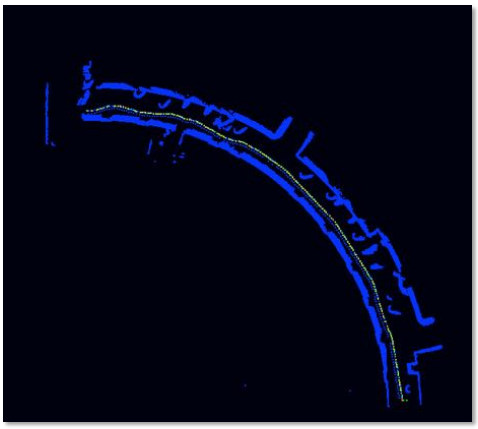


LINKÖPING UNIVERSITY

32

Mapping cont'd





Based only on odometry + LiDAR range data

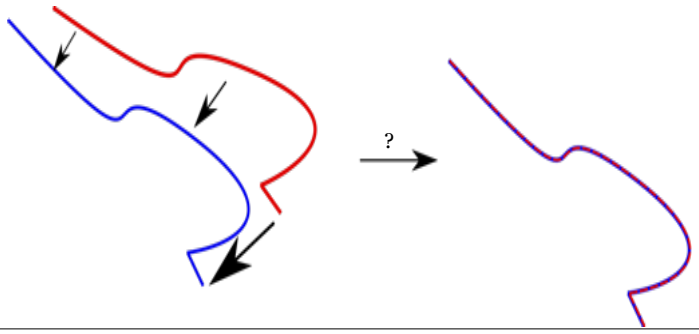
Scan matching

Calculate rotation and translation between two consecutive LiDAR scans, which will correct for odometry error.

Iterative Closest Point (ICP) – iteratively minimize the sum of square differences between two pairs of points which are selected from reference and a source scan.

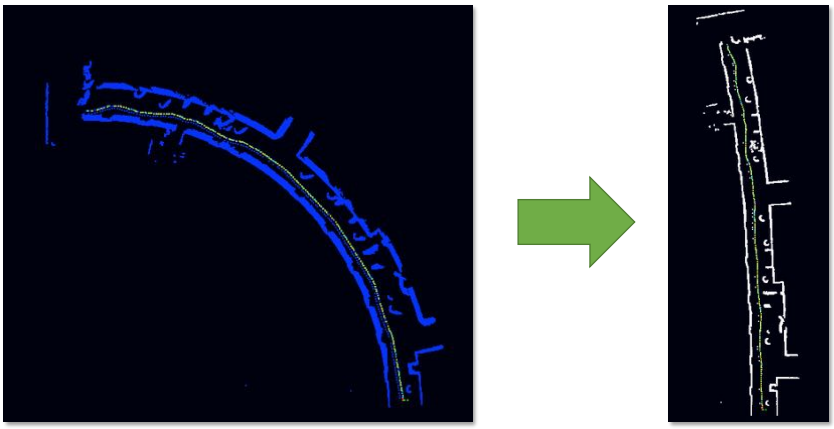
Input: reference scan (blue), source scan (red).

Output: rotation and translation between reference and source scans.



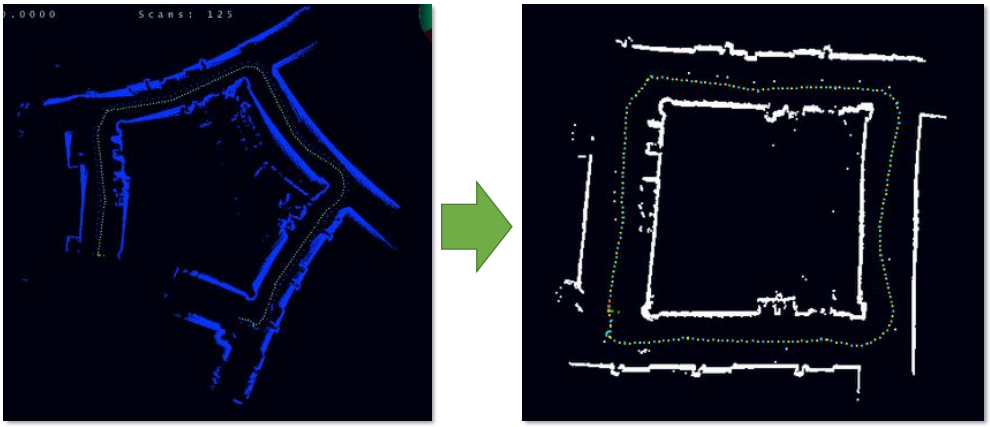
https://en.wikipedia.org/wiki/Iterative_closest_point

Mapping cont'd



Based only on odometry + LiDAR range data + scan matching (ICP)

Mapping cont'd



Based only on odometry + LiDAR range data + scan matching (ICP)

Outline

- Sensors - summary
- Computer systems
- Robotic architectures
- Navigation:
  - Mapping and Localization
  - Motion planning
  - Motion control

LINKÖPING UNIVERSITY

37

Robotic perception

Robot perception viewed as **temporal inference from sequences of actions and measurements** → dynamic Bayes network of first order Markov process

Example: **localization** - holonomic robot with range sensor, estimate pose while moving

$\mathbf{X}_t = (x_t, y_t, \theta)$ – state of the robot at time t -> not observable

$\mathbf{Z}_t = (\text{range}_1, \text{range}_2, \text{range}_3, \ldots)$ – sensor reading at time t -> observable

$\mathbf{A}_t = (v_t, \omega_t)$ – known action executed at time t

$P(\mathbf{X}_t) = P(\mathbf{X}_t \mid \mathbf{z}_{1:t}, \mathbf{a}_{1:t-1})$ – current believe state (captures past)

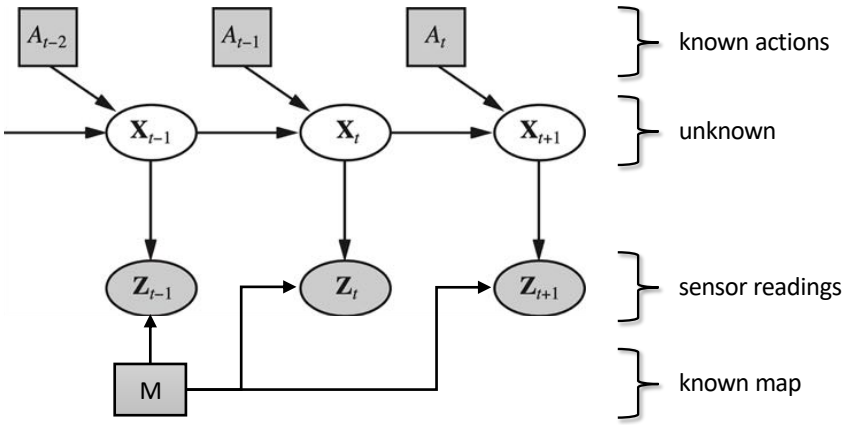
next belief state? → Bayesian inference problem

$P(\mathbf{X}_{t+1} \mid \mathbf{z}_{1:t+1}, \mathbf{a}_{1:t}) = ?$ given $P(\mathbf{X}_t)$ and new observation $\mathbf{z}_{t+1}$

LINKÖPING UNIVERSITY

38

Graphical Model



Recursive filtering equation

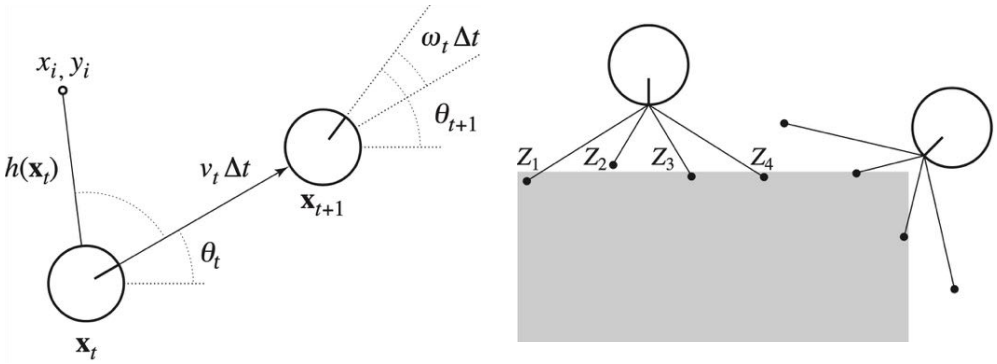
$$P(\mathbf{X}_{t+1} \mid \mathbf{z}_{1:t+1}, \mathbf{a}_{1:t}) = \alpha \underbrace{P(\mathbf{z}_{t+1} \mid \mathbf{X}_{t+1})}_{\text{sensor model}} \int \underbrace{P(\mathbf{X}_{t+1} \mid \mathbf{x}_t, \mathbf{a}_t)}_{\text{motion model}} \underbrace{P(\mathbf{X}_t \mid \mathbf{z}_{1:t}, \mathbf{a}_{1:t-1})}_{\text{previous believe state}} d\mathbf{x}_t$$

using Bayes' rule, Markov assumption, theorem of total probability

Motion model: deterministic state prediction + noise

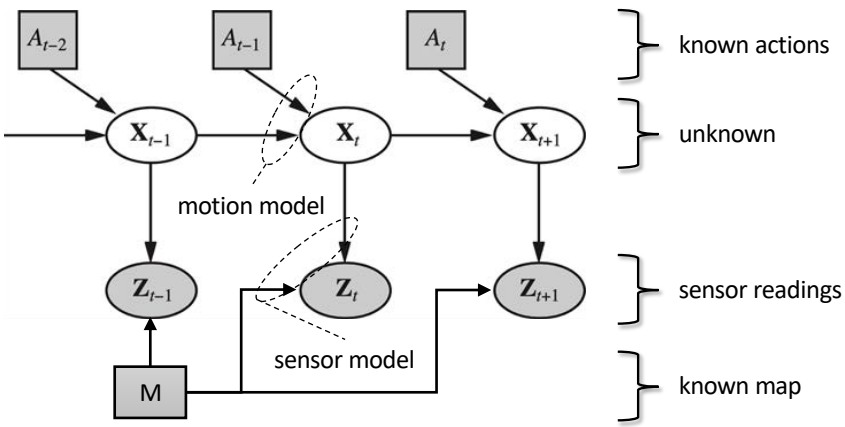
Sensor model: likelihood of making observation $\boldsymbol{z}_{t+1}$ when robot is in state $\boldsymbol{X}_{t+1}$

Motion and sensor model



Assume Gaussian noise in motion prediction, sensor range measurements

Graphical Model



Localization algorithms

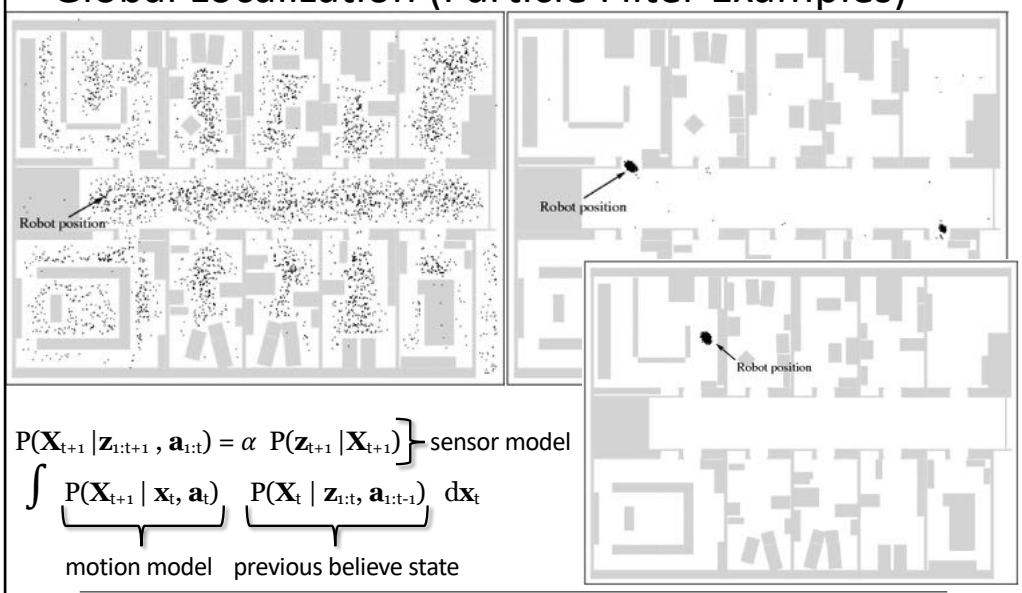
Particle filter (Monte Carlo localization):

- belief state - a collection of particles that correspond to states
- belief state sampled, each sample weighted by likelihood it assigns to new evidence, population resampled using weights

Kalman filter:

- belief state - a single multivariate Gaussian
- each step maps a Gaussian into a new Gaussian, i.e. it computes a new mean and covariance matrix from the previous mean and covariance matrix
- assumes linear motion and measurement models (linearisation →extended KF)

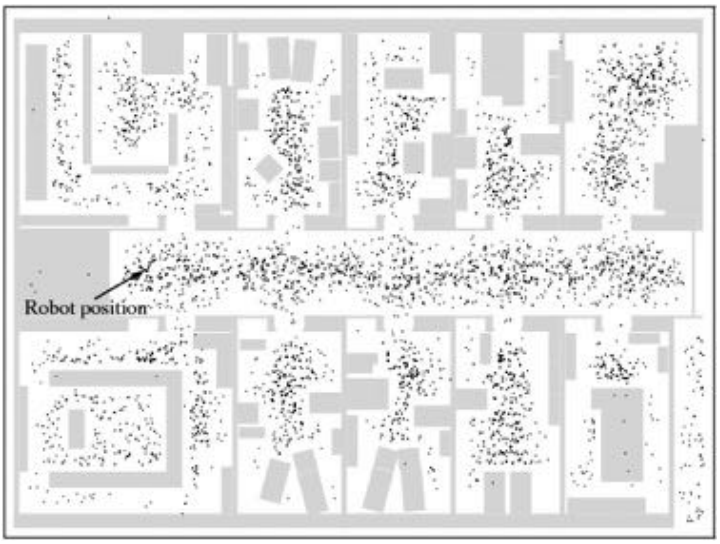
Global Localization (Particle Filter Examples)



$$P(\mathbf{X}_{t+1} \mid \mathbf{z}_{1:t+1}, \mathbf{a}_{1:t}) = \alpha \; P(\mathbf{z}_{t+1} \mid \mathbf{X}_{t+1}) \int \underbrace{P(\mathbf{X}_{t+1} \mid \mathbf{x}_t, \mathbf{a}_t)}_{\text{motion model}} \; \underbrace{P(\mathbf{X}_t \mid \mathbf{z}_{1:t}, \mathbf{a}_{1:t-1})}_{\text{previous believe state}} \, d\mathbf{x}_t$$

sensor model

Global Localization (Particle Filter Examples)

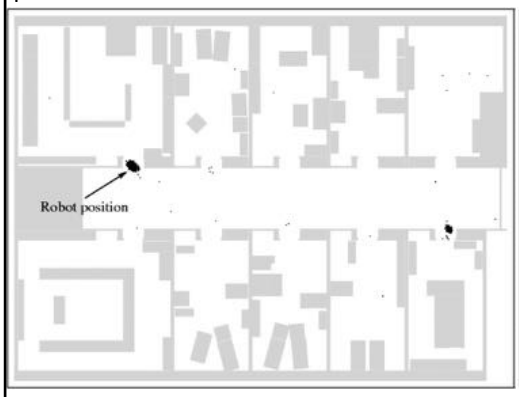


Intuitive explanation:

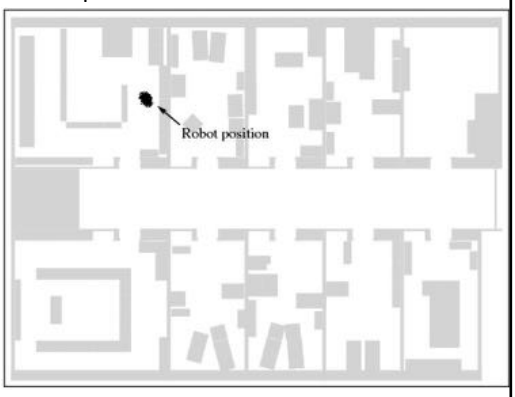
- initialize particles
- execute known action
- apply motion model
- update particle weights based on sensor model
- resample based on weights
- repeat

Global Localization (Particle Filter Examples)

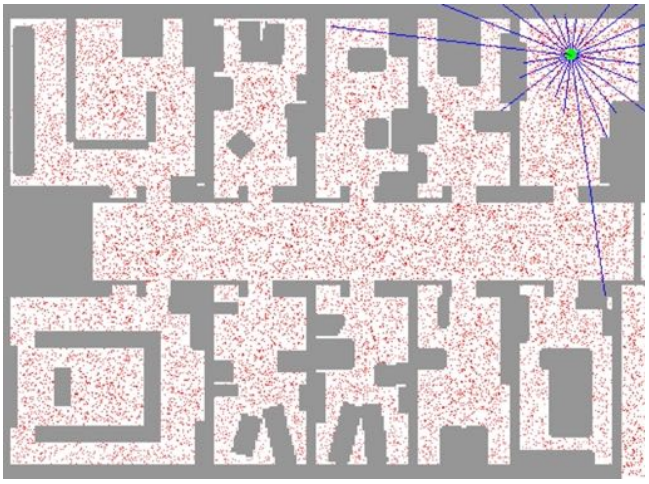
After several iterations – two likely robot position estimates:



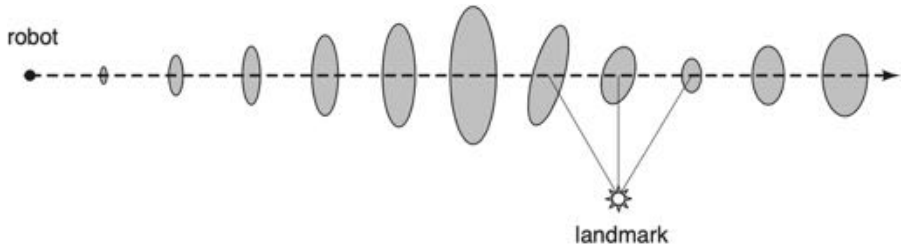
After more iterations – converged to a single robot position estimation:



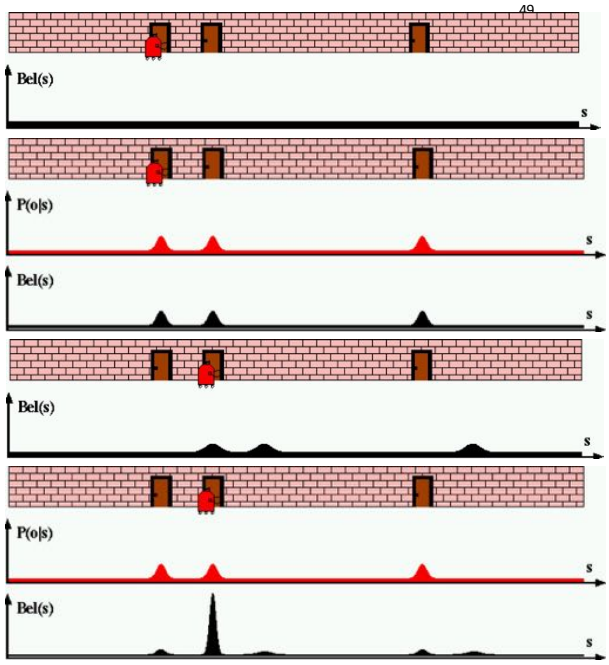
Global Localization
(Particle Filter Examples; sonar vs laser)



Extended Kalman Filter (EKF) - example



EKF example



SLAM

Localization: given map and observed landmarks, update pose distribution

Mapping: given pose and observed landmarks, update map distribution

<u>SLAM</u>: given observed landmarks, update pose and map distribution

Probabilistic formulation of SLAM:

add landmark locations $L_1, \ldots, L_k$ to the state vector, proceed as for localization

<u>Problems:</u>

- dimensionality of map features has to be adjusted dynamically
- identification of already mapped features

SLAM Example

http://wiki.ros.org/hector_slam
https://youtu.be/F8pdObV_df4

Outline

- Sensors - summary
- Computer systems
- Robotic architectures
- Navigation:
  - Mapping and Localization
  - Motion planning
  - Motion control

Motion Planning

Motion types:

- point-to-point
- compliant motion (screwing, pushing boxes)

Representations: configuration space vs workspace

Kinematic state: robot's configuration (location, orientation, joint angles), no velocities, no forces

Path planning: find path from one configuration to another

Problem: continuous state space, can be high-dimensional

Motion Planning - representations

Workspace - physical 3D space (e.g. joint positions)

Robot has rigid body of finite size

Well-suited for collision checking

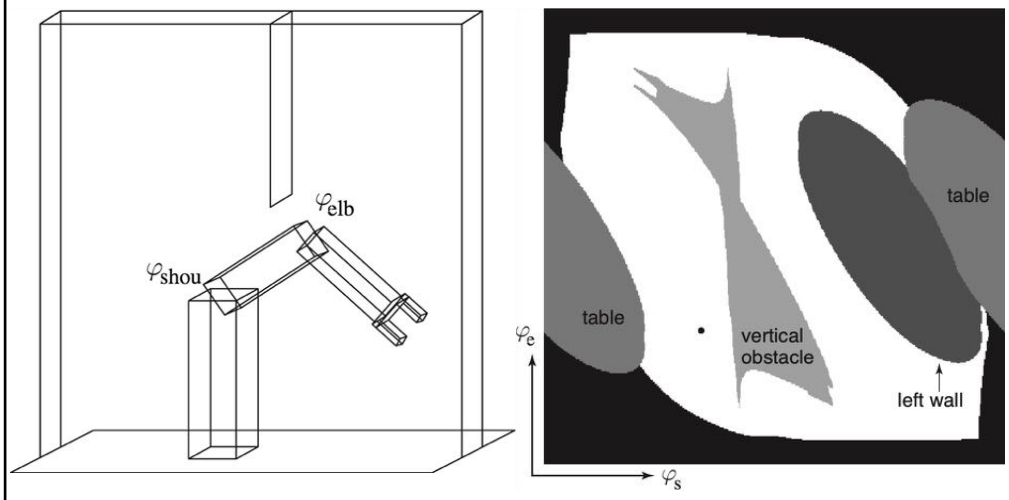
Problem: linkage constraints (not all workspace coordinates attainable) makes path planning difficult in workspace

Configuration Space (C-space) - space of robot states (e.g. joint angles)

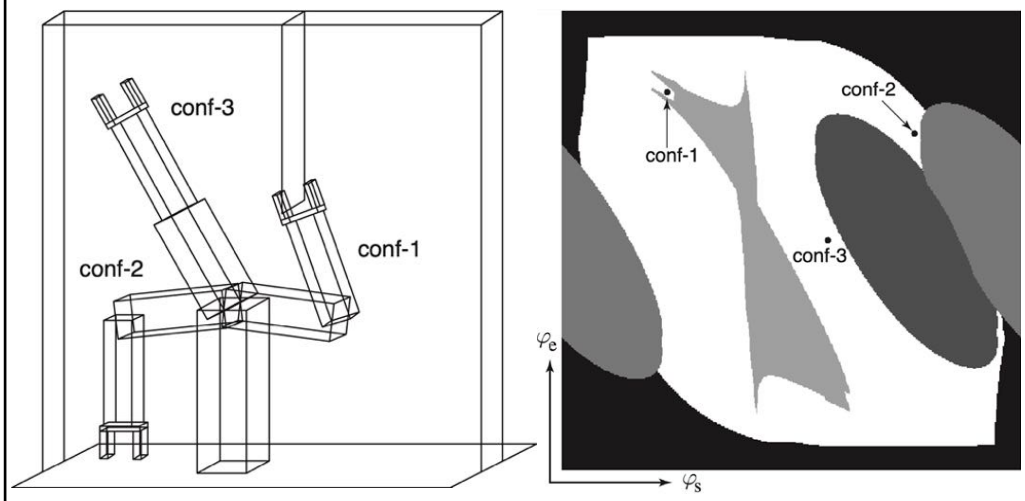
Robot is a point, obstacles have complex shapes

Problem: tasks are expressed in workspace coordinates, obstacle representation problematic

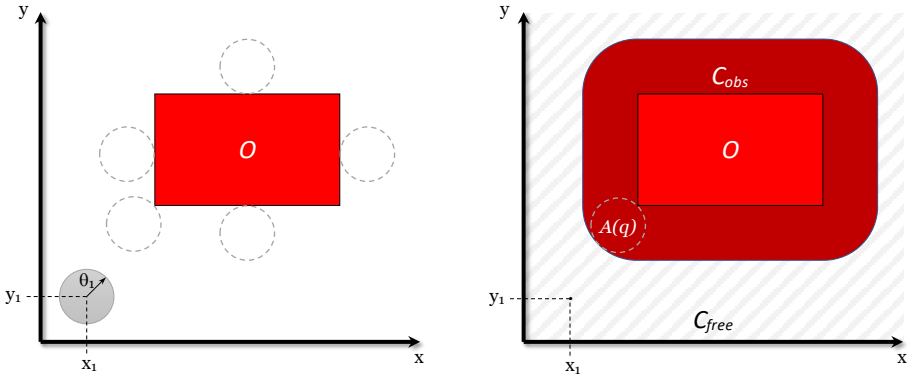
Workspace vs. Configuration Space



Workspace vs. Configuration Space



Workspace vs. Configuration Space



W – workspace world either $\mathbb{R}^2$ or $\mathbb{R}^3$
O – obstacle region, $O \subset W$

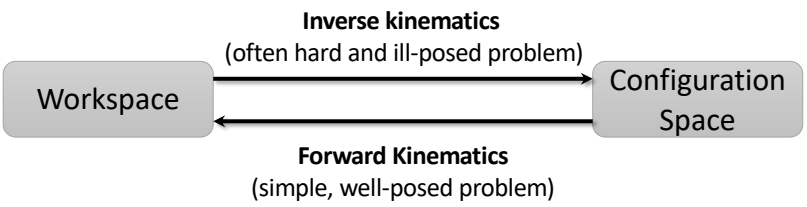
q – a robot configuration e.g. (x_1, y_1, θ_1)
$A(q)$ – set of points on the robot in configuration q

C – all possible robot configurations
$C_{obs} = \{q : q \in C$ ***and*** $A(q) \cap O \neq \{\}\}$
$C_{free} = C - C_{obs}$

Motion Planning - representations

Free space (attainable configurations) ->occupied space (not attainable configurations, obstacles)

Planner may generate configuration in *configuration space*, apply kinematics and check in *workspace for obstacles*



Path Planning

Basic problem: convert infinite number of states into finite state space

Cell decomposition:
- divide up space into simple *cells*,
- each of which can be traversed "easily"

Skeletonization:
- identify a finite number of easily connected points/lines
- form a graph such that any two points are connected by a path

Graph search and colouring algorithms

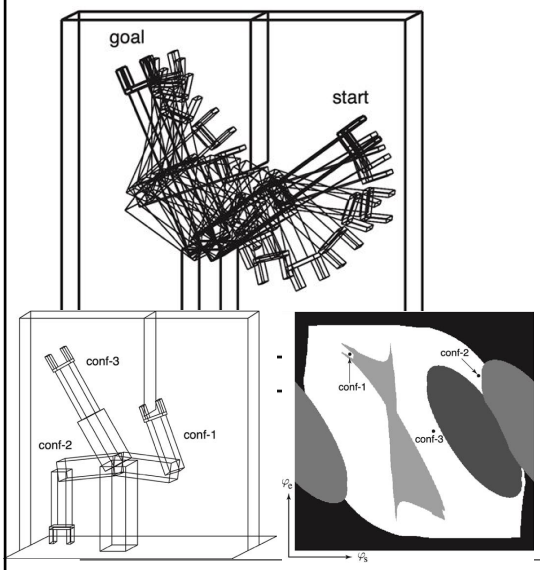
Assumptions: motion deterministic, localization exact, static scenes

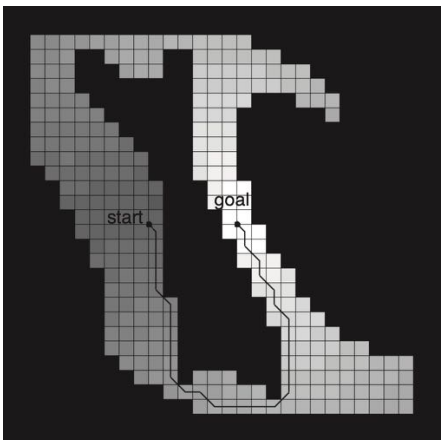
Not robust with respect to small motion errors, does not consider limits due to robot dynamics

LINKÖPING UNIVERSITY

59

Cell Decomposition





Grayscale shading - cost from the grid cell do the goal

LINKÖPING UNIVERSITY

60

Cell Decomposition

Problem: may be no path in pure free space cells

Soundness

(wrong solution if cells are mixed)

vs.

Completeness

(no solution if only pure free cells considered)

Solution: recursive decomposition of mixed (free+obstacle) cells or exact decomposition. Doesn't scale well for higher dimensions.

Skeletonization

Visibility graphs: find lines connecting obstacle vertices through free space, build and search graph; not for higher dimensions

Voronoi graphs: find all points in free space equidistant to two or more obstacles, build and search graph; maximizes clearance, creates unnecessarily large detours, does not scale well for higher dimensions

Sample-based path planners.

Probabilistic roadmaps (PRM):

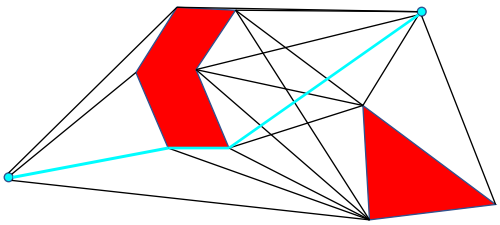
- generate randomly large number of configurations in free space, build graph (construction phase)
- search graph (query phase)

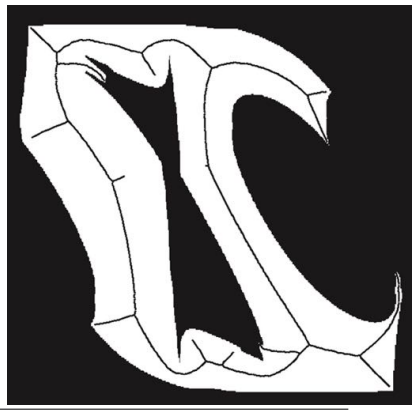
Rapidly exploring Random Trees (RRT):

- generate a tree rooted in start configuration by random sampling of free space until goal configuration is reached (query phase)

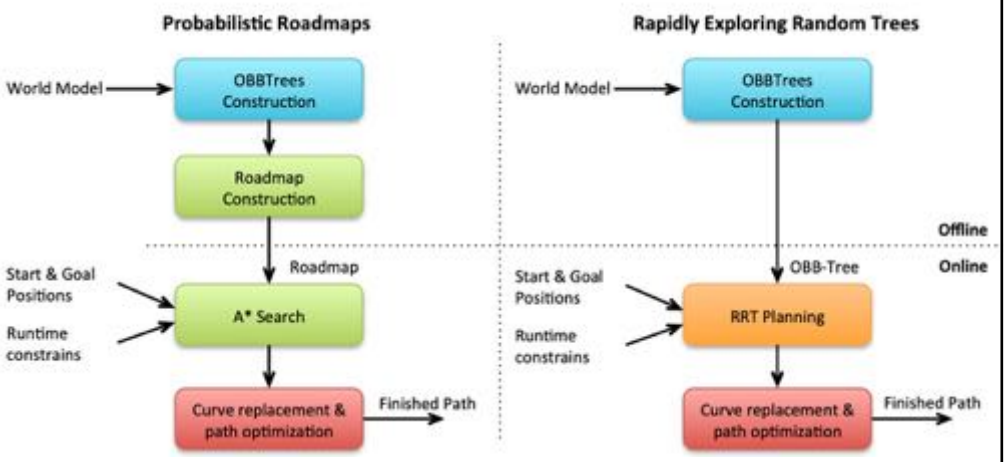
Scales better to higher dimensions but incomplete

Visibility and Voronoi Graph

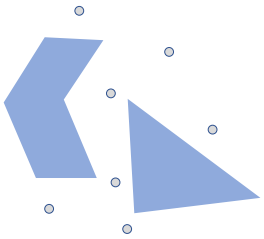




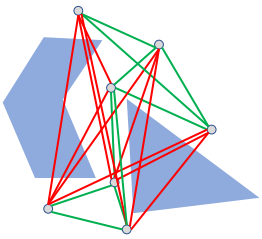
PRM and RRT planning procedure example



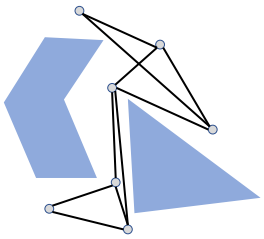
PRM Example (construction phase)



Generate random configurations

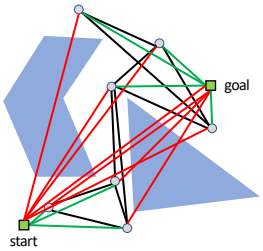


Make connections

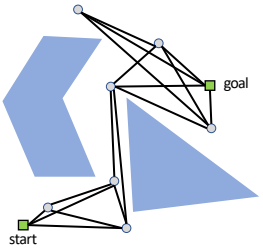


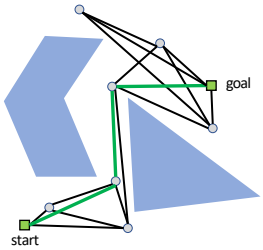
Resulting free space graph representation

PRM Example (query phase)



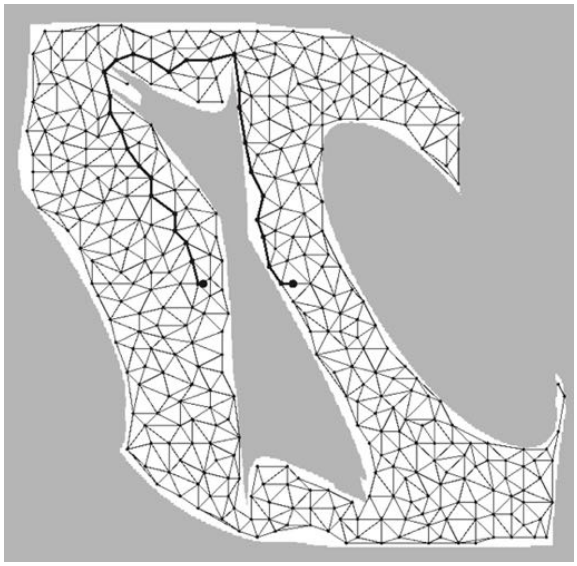
Add start and goal configurations to the roadmap





A* search (+optional postprocessing)

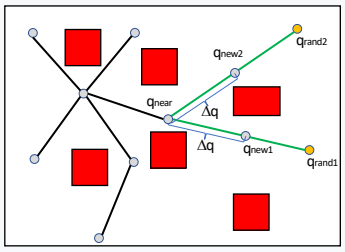
PRM Example



RRT

```
Algorithm BuildRRT
    Input: Initial configuration q_init, number of vertices in RRT K, incremental distance
Δq)
    Output: RRT graph G

    G.init(q_init)
    for k = 1 to K do
        q_rand ← RAND_CONF()
        q_near ← NEAREST_VERTEX(q_rand, G)
        q_new ← NEW_CONF(q_near, q_rand, Δq)
        G.add_vertex(q_new)
        G.add_edge(q_near, q_new)
    return G
```



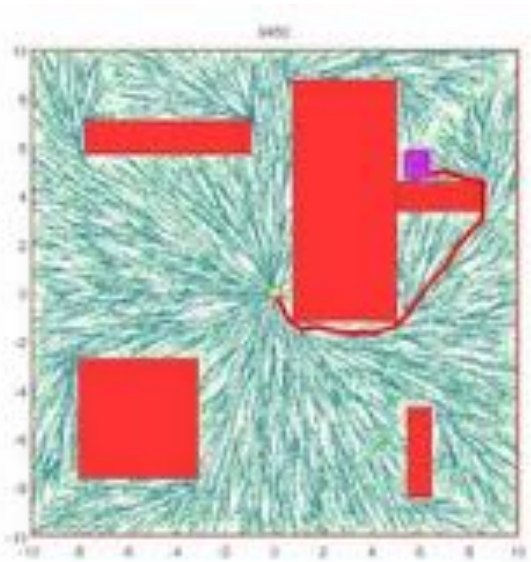
- "←" denotes assignment. For instance, "*largest* ← *item*" means that the value of *largest* changes to the value of *item*.
- "**return**" terminates the algorithm and outputs the following value.

RAND_CONF – samples random configuration from free space q_{rand}.
NEAREST_VERTEX – find q_{near} i.e. the closest vertex in existing graph G from q_{rand}.
NEW_CONF – select new configuration q_{new} by moving at incremental distance Δq from q_{near} in the direction of q_{rand}.

https://en.wikipedia.org/wiki/Rapidly-exploring_random_tree

Demo: https://demonstrations.wolfram.com/RapidlyExploringRandomTreeRRTAndRRT/

RRT*



Asymptotically optimal: Converges to the optimal solution as more and more *milestones* are sampled.

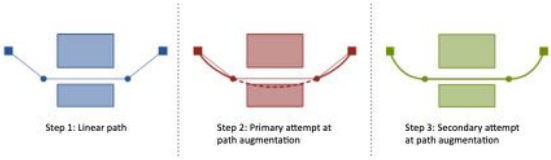
LINKÖPING UNIVERSITY

https://www.youtube.com/watch?v=YKiQTJpPFkA

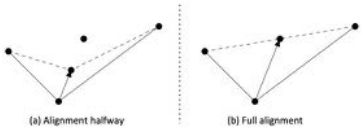
PRM/RRT - Post Processing Example

Example curve replacement and path optimization:

Transformation from linear to cubic (smooth) path segments:



Alignment of nodes for improved path quality:



LINKÖPING UNIVERSITY

Outline

- Sensors - summary
- Computer systems
- Robotic architectures
- Navigation:
  - Mapping and Localization
  - Motion planning
  - Motion control

LINKÖPING UNIVERSITY

71

Motion Control

- Path following involves forces: friction, gravity, inertia,
- Dynamic state: kinematic state + robot's velocities
- Transition models expressed as differential equations
- Path planner assumes robot can follow any path
- Robot's inertia limits manoeuvrability

- Problem: including dynamic state in planners makes motion planning intractable
- Solution: simple kinematic planners + low-level controller for force calculation
- Other solution: motion control without planning: potential field and reactive control

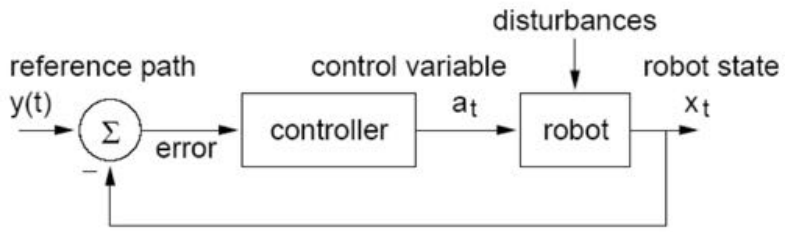
LINKÖPING UNIVERSITY

72

Controllers

- Techniques for **generating robot controls in real time** using feedback from the environment to achieve a control objective
- Reference controller: keep robot on pre-planned path (reference path)
- Optimal controller: controller that optimises a global cost function, e.g. optimal policies for MDPs
- Stable: small perturbations lead to bounded error between robot and reference signal
- Strictly stable: able to return to reference signal

Closed-loop control

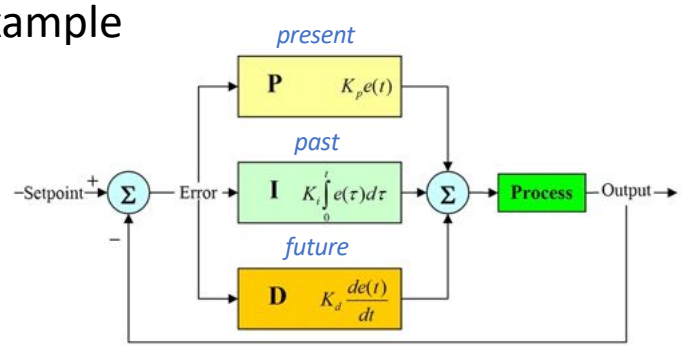


Performance of controller:

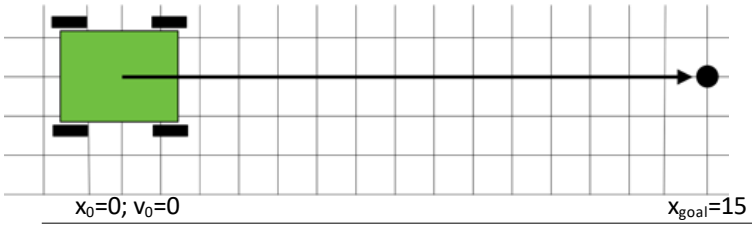
- stability
- overshoot ← inertia
- steady-state error ← friction
- rise time
- settling time

P controller: $a_t = K_P(y(t) - x_t)$ K_P gain parameter

PID Example



$e(t_0)=x_{goal}-x_0=15$



Closed-loop control

PD controller:

- decreases overshoot
- decreases settling time

$$a_t = K_P(y(t) - x_t) + K_D \frac{\partial(y(t) - x_t)}{\partial t}$$

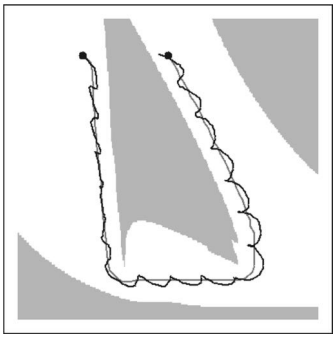
PID controller:

- eliminates steady-state error

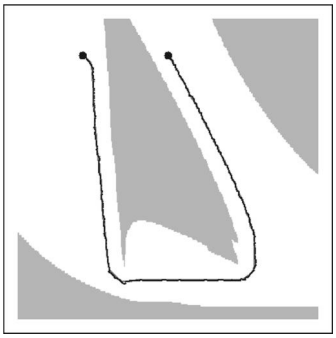
$$a_t = K_P(y(t) - x_t) + K_D \frac{\partial(y(t) - x_t)}{\partial t} + K_I \int (y(t) - x_t)\, dt$$

Closed-loop control

P control:
K_P= 1.0

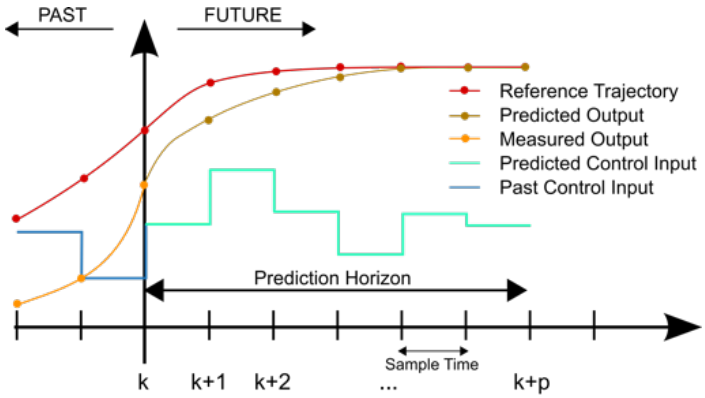


P control:
K_P= 0.1



PD control:
K_P= 0.3 K_D=0.3

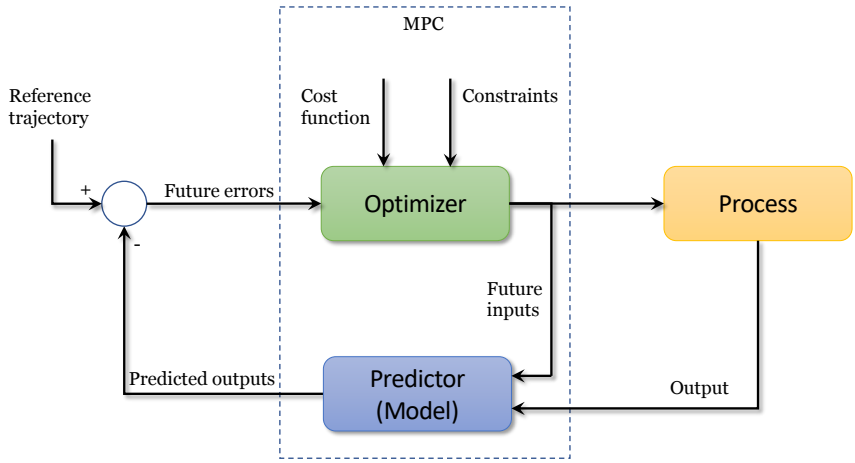
Model Predictive Control



1. At time k, solve open loop optimal control problem over a specified finite time horizon
2. Apply first input
3. At time k+1, repeat from step 1.

https://en.wikipedia.org/wiki/Model_predictive_control

Model Predictive Control



Predictor/Model: fundamental or empirical
Constraints e.g. on inputs, outputs, state are respected

LINKÖPING UNIVERSITY

https://www.youtube.com/watch?v=dL_ZFSvLXlU

79

MPC – simple example

LINKÖPING UNIVERSITY

https://medium.com/@david010/vehicle-mpc-controller-33ae813cf3be
https://github.com/cipher982/MPC-vehicle-controller

80

MPC and Learning

https://doi.org/10.3384/diss.diva-163419

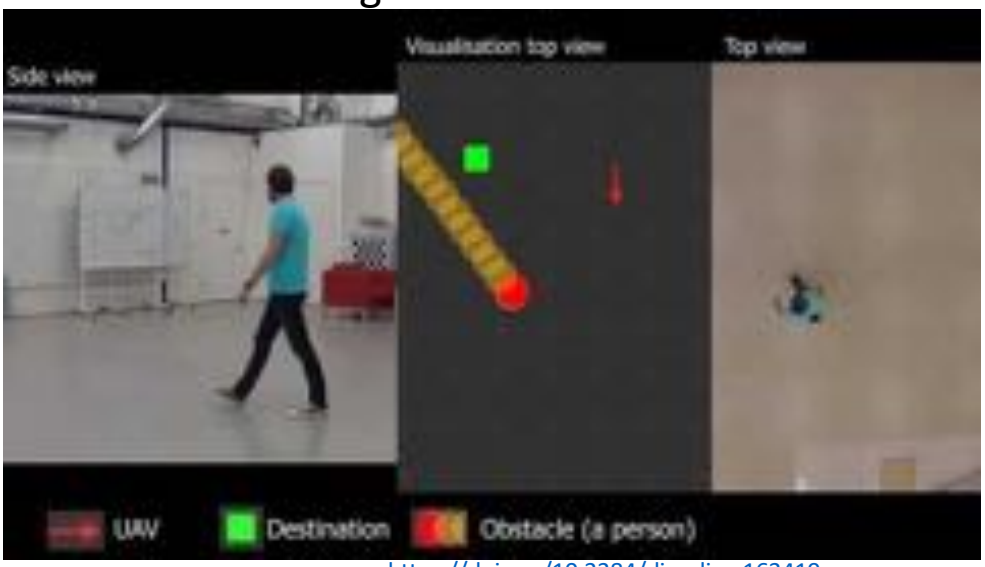
https://www.youtube.com/watch?v=QYYknZ20Zcw

https://www.youtube.com/watch?v=xa53w1tyZl0

LINKÖPING UNIVERSITY

81

MPC and Learning



https://doi.org/10.3384/diss.diva-163419

https://www.youtube.com/watch?v=QYYknZ20Zcw

LINKÖPING UNIVERSITY

82

MPC and Learning

LINKÖPING UNIVERSITY

https://www.youtube.com/watch?v=dL_ZFSvLXlU

83

Questions?

LINKÖPING UNIVERSITY

84