Robot path planning and global navigation

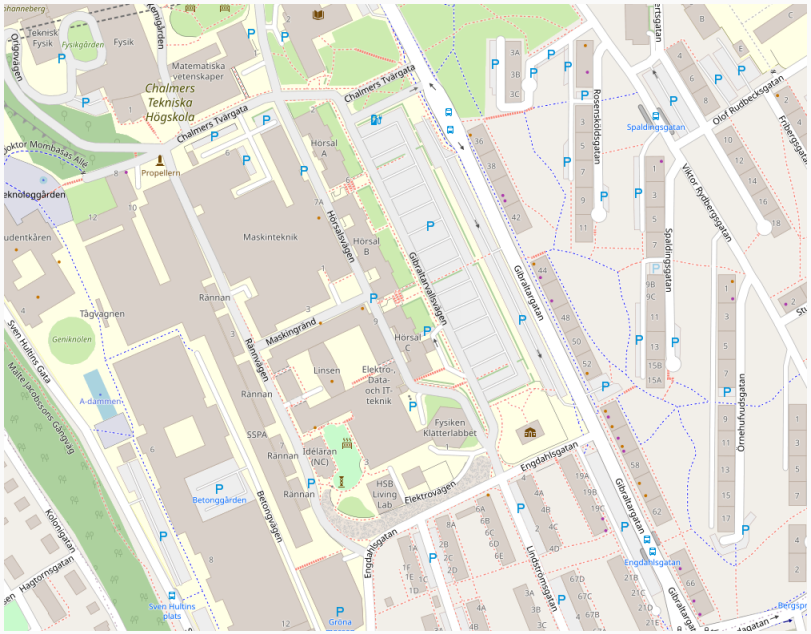
Autonomous robots, TME290

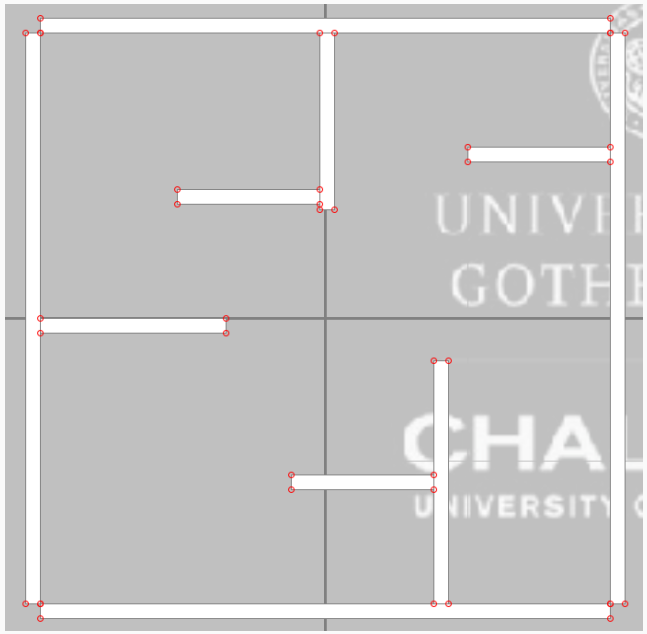
Ola Benderius
ola.benderius@chalmers.se

Applied artificial intelligence
Vehicle engineering and autonomous systems
Mechanics and maritime sciences
Chalmers

Introduction

Maps


Chalmers Tekniska Högskola
Matematiska vetenskaper
Fysik
Chalmers Tvärgata
Propellern
Maskinteknik
Horsal A
Horsal B
Horsal C
Rännan
Tågvagnen
Maskingränd
Rännvägen
Linsen
Elektro- Data- och IT-teknik
Fysiken Klatterlabbet
Idelaran (NC)
HSB Living Lab
Elektrovägen
Betongvägen
Betonggården
Horsalsvägen
Gibraltargatan
Engdahlsgatan
Lindholmsgatan
Rosenholdsgatan
Spaldingsgatan
Olof Rudbecksgatan
Viktor Rydbergsgatan
Örnehufvudsgatan
Sven Hultins Gata
Kolningatan
Hagtornsgatan
Sven Hultins plats
A-dammen



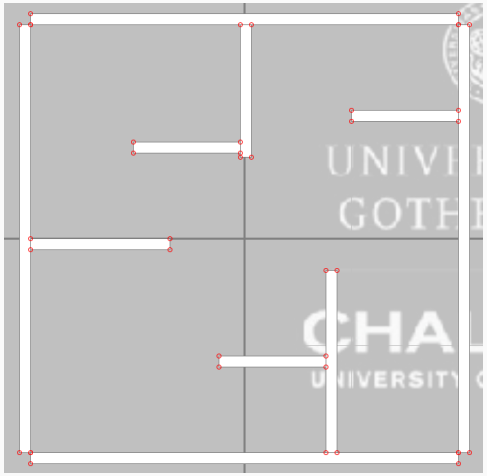
A global map

- A global map can be given as static data
- Or, it can be created online using for example SLAM (next lecture)

Map objectives

The map will be used for navigation, and there are three involved concepts

- Localisation: Find the own position in the map
- Path-planning: Find an efficient path between that position to a target
- Path following: Follow the path by applying actuation (local navigation)



Localisation

Localisation

There are some approaches to localisation

- External signal (triangulation):
  - Global navigation satellite systems (GNSS)
  - Cameras
  - etc.
- Motion measurement: IMU, wheel encoders (odometry)
- Motion estimation: Kinematic or dynamic model (process model)
- Landmarks (features) and a known map (e.g. SLAM)

GNSS coordinates

The Earth is not a sphere or an ellipsoid. It's not trivial how to form a coordinate system.

- World Geodetic System from 1984 (WGS84) is the most common Earth model
- Latitude and longitude
- WGS84 to Cartesian is always a local estimation

GNSS coordinates

The Earth is not a sphere or an ellipsoid. It's not trivial how to form a coordinate system.

- World Geodetic System from 1984 (WGS84) is the most common Earth model
- Latitude and longitude
- WGS84 to Cartesian is always a local estimation

Useful tool

`https://github.com/chrberger/WGS84toCartesian`

GNSS shortcomings

The GNSS will give an absolute global position. Are there any problems?

GNSS shortcomings

The GNSS will give an absolute global position. Are there any problems?

- Relatively inaccurate, noise
- Missing data, shadows etc.
- Low frequency (e.g. 20 Hz)

Can you think of any ways to improve this? There are two main concepts.

Can you think of any ways to improve this? There are two main concepts.

These two concepts are formalised in the *Kalman filter*.

GNSS with sensors and internal model

The GNSS signals can be improved using (1) onboard sensors, and (2) an internal (process) model. This is an example of sensor fusion.

GNSS with sensors and internal model

The GNSS signals can be improved using (1) onboard sensors, and (2) an internal (process) model. This is an example of sensor fusion.

- The GNSS gives a global reference occasionally

GNSS with sensors and internal model

The GNSS signals can be improved using (1) onboard sensors, and (2) an internal (process) model. This is an example of sensor fusion.

- The GNSS gives a global reference occasionally
- The onboard sensors can keep track of the movement between GNSS updates

GNSS with sensors and internal model

The GNSS signals can be improved using (1) onboard sensors, and (2) an internal (process) model. This is an example of sensor fusion.

- The GNSS gives a global reference occasionally
- The onboard sensors can keep track of the movement between GNSS updates
- The internal model removes noise since it contains information about possible movement

GNSS with sensors and internal model

The GNSS signals can be improved using (1) onboard sensors, and (2) an internal (process) model. This is an example of sensor fusion.

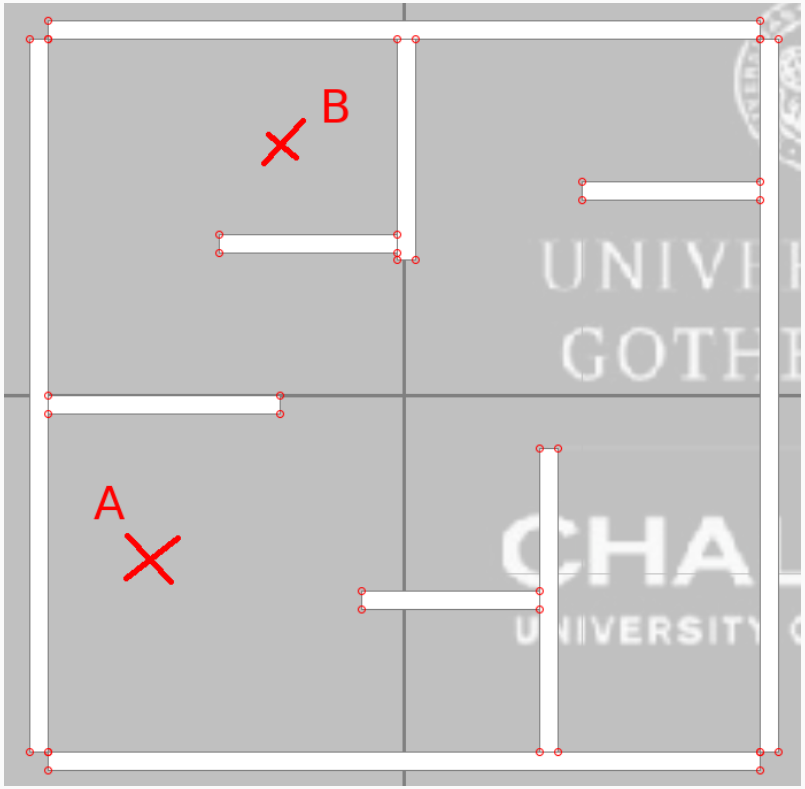
- The GNSS gives a global reference occasionally
- The onboard sensors can keep track of the movement between GNSS updates
- The internal model removes noise since it contains information about possible movement

How to do this is the topic of the next lecture!

Path-planning

Path-planning

If knowing where the robot is located and any desired target, how can a path be generated?



Common algorithms

There are some very common algorithms

- Dijkstra's algorithm: Basic, always shortest path
- A*: Commonly used, quick
- D* Lite: Suitable for real-world applications

Dijkstra's algorithm

- Normally, algorithms finds the best path in a network or in a grid

Dijkstra's algorithm

- Normally, algorithms finds the best path in a network or in a grid
- Here, a grid can be dynamically created if assuming a fixed step distance

Dijkstra's algorithm

- Normally, algorithms finds the best path in a network or in a grid
- Here, a grid can be dynamically created if assuming a fixed step distance
- During creation, each new step can be matched towards obstacles, for example by using intersection check between line segments

Dijkstra's algorithm

- Normally, algorithms finds the best path in a network or in a grid
- Here, a grid can be dynamically created if assuming a fixed step distance
- During creation, each new step can be matched towards obstacles, for example by using intersection check between line segments
- Important: Make sure to get an appropriate offset to any obstacle

Dijkstra's algorithm

- Normally, algorithms finds the best path in a network or in a grid
- Here, a grid can be dynamically created if assuming a fixed step distance
- During creation, each new step can be matched towards obstacles, for example by using intersection check between line segments
- Important: Make sure to get an appropriate offset to any obstacle
- Each node is known by its coordinate, each new step needs to be checked for revisit

Dijkstra's algorithm

1. The robots start position is denoted the start node, which becomes the current node. Assign the distance value 0 to the start node.

Dijkstra's algorithm

1. The robots start position is denoted the start node, which becomes the current node. Assign the distance value 0 to the start node.
2. Go through all the non-visited accessible neighbours a_i of the current node, and compute their distance d from the *start* node. For each node i, if d is smaller than the previously stored distance d_i (initially infinite), then
   2.1 Update the previously stored distance, i.e. set $d_i = d$.
   2.2 Assign the current node as the predecessor node of a_i.

Dijkstra's algorithm

1. The robots start position is denoted the start node, which becomes the current node. Assign the distance value 0 to the start node.
2. Go through all the non-visited accessible neighbours a_i of the current node, and compute their distance d from the *start* node. For each node i, if d is smaller than the previously stored distance d_i (initially infinite), then
   2.1 Update the previously stored distance, i.e. set $d_i = d$.
   2.2 Assign the current node as the predecessor node of a_i.
3. After checking all the neighbours of the current node, set its status to visited.

Dijkstra's algorithm

1. The robots start position is denoted the start node, which becomes the current node. Assign the distance value 0 to the start node.
2. Go through all the non-visited accessible neighbours a_i of the current node, and compute their distance d from the *start* node. For each node i, if d is smaller than the previously stored distance d_i (initially infinite), then
   2.1 Update the previously stored distance, i.e. set $d_i = d$.
   2.2 Assign the current node as the predecessor node of a_i.
3. After checking all the neighbours of the current node, set its status to visited.
4. Select the node (among *all* the non-visited nodes) with the smallest distance from the start node, and set it as the new current node.

Dijkstra's algorithm

1. The robots start position is denoted the start node, which becomes the current node. Assign the distance value 0 to the start node.
2. Go through all the non-visited accessible neighbours a_i of the current node, and compute their distance d from the *start* node. For each node i, if d is smaller than the previously stored distance d_i (initially infinite), then
   2.1 Update the previously stored distance, i.e. set $d_i = d$.
   2.2 Assign the current node as the predecessor node of a_i.
3. After checking all the neighbours of the current node, set its status to visited.
4. Select the node (among *all* the non-visited nodes) with the smallest distance from the start node, and set it as the new current node.
5. Return to step 2, unless the target has been reached.

Dijkstra's algorithm, example

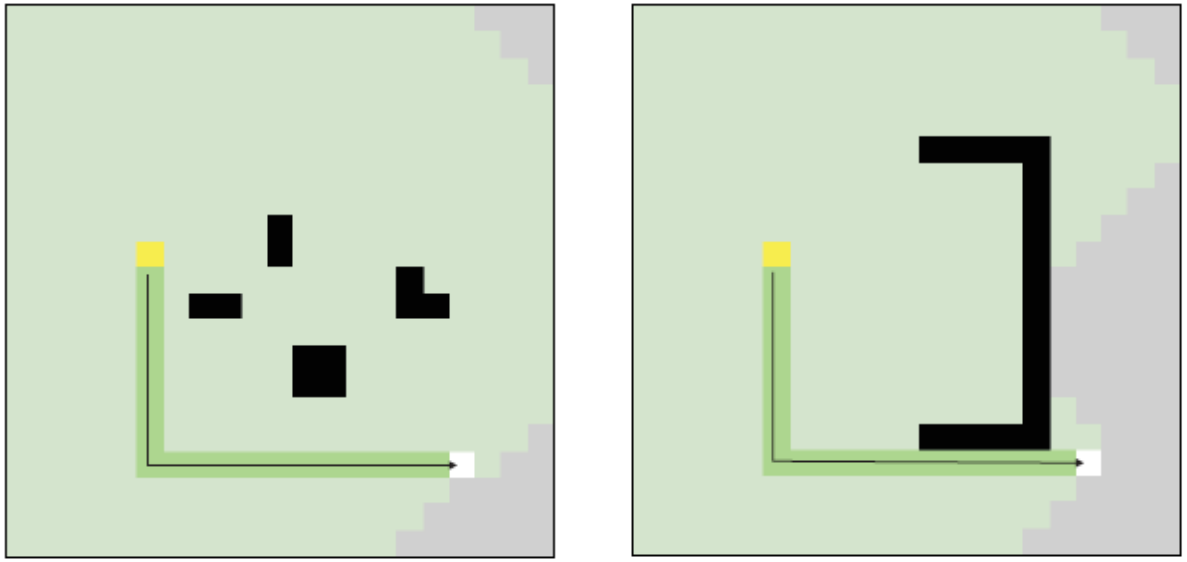
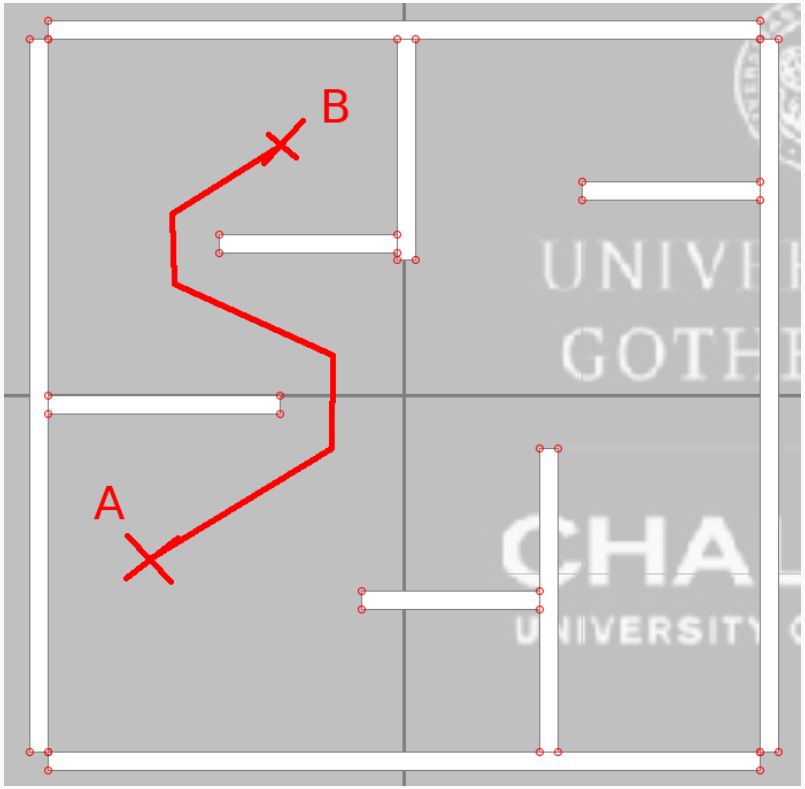


Figure 6.9: *Two examples of paths generated using Dijkstra's algorithm. The cells (nodes) that were checked during path generation are shown in light green, whereas the actual path is shown in dark green and with a solid line. The yellow cell is the start node and the white cell is the target node.*

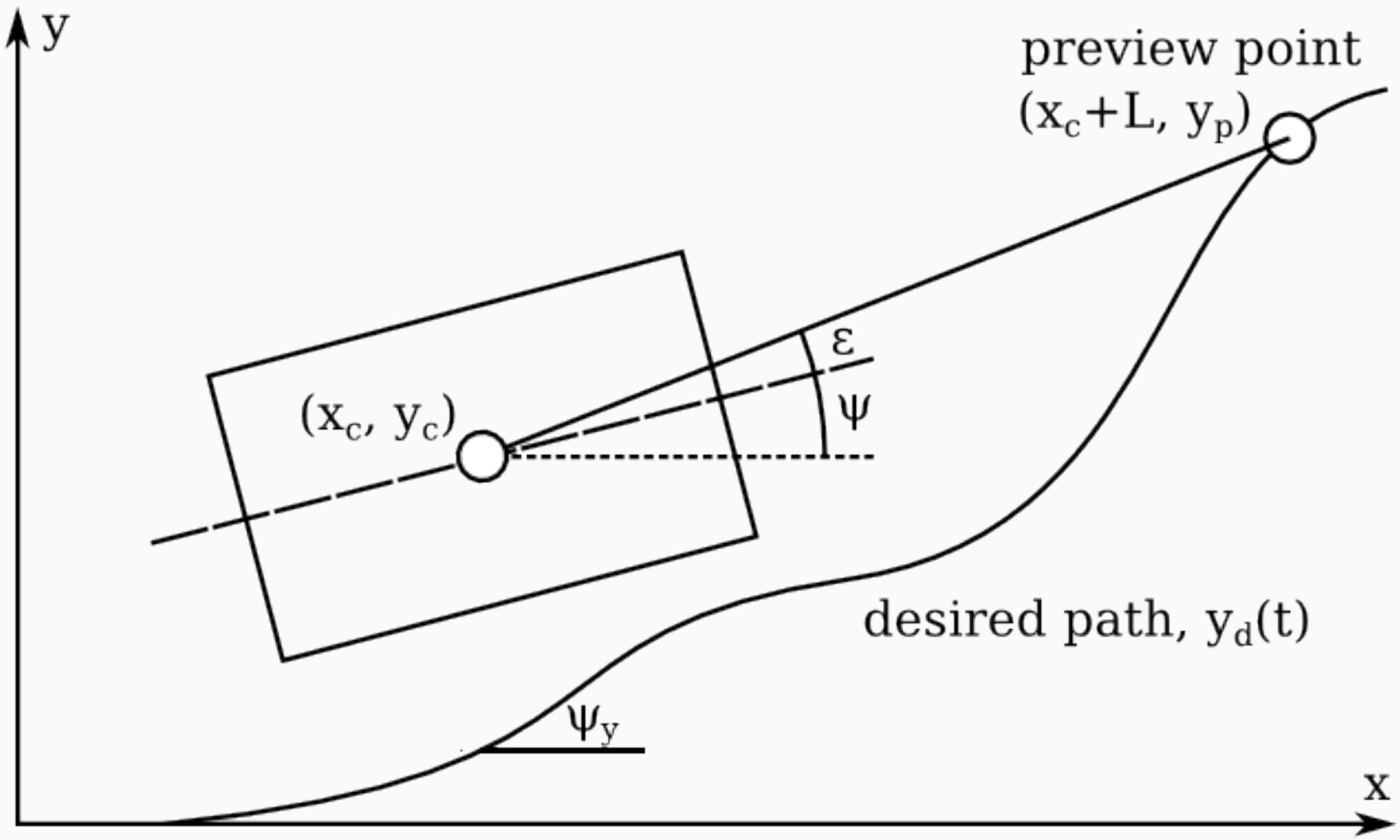
Path following

Path following

If knowing where the robot is located at each time t, and a desired path of movement, how can this path be followed?



Simple path-following model



- To get stable path following, a preview-point or aim point could be used

Collision free navigation

Depending on the preview distance, the navigated path is not guaranteed to be collision free, due to the robot cutting corners.

Collision free navigation

Depending on the preview distance, the navigated path is not guaranteed to be collision free, due to the robot cutting corners.

In general this is good, since it generates a more smooth trajectory. However, collisions must be avoided!

Questions

Please post all questions on the Canvas discussion pages, in that way we can all benefit from the answers, and I can highlight important outcomes.