EE4308 Turtlebot Project

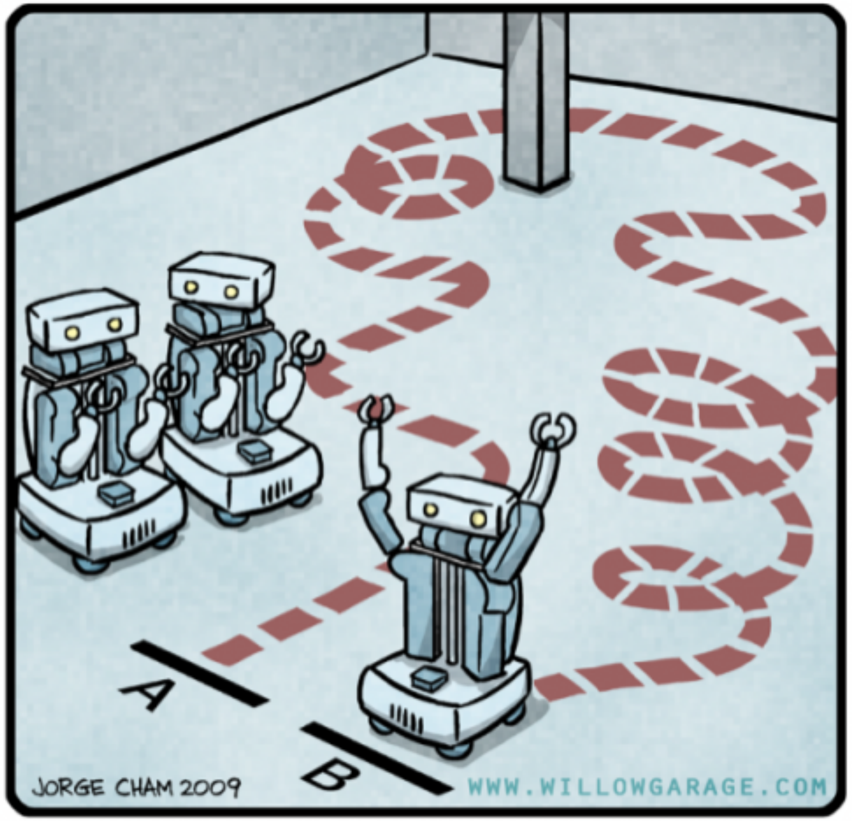
Advances in Intelligent Systems and Robotics
Academic Year 2016-2017

Preben Jensen Hoel
A0158996B

Paul-Edouard Sarlin
A0153124U

What will be discussed

1. Problem statement
2. Overview of the solution
3. Information gathering
4. Global planning
5. Local planning
6. Demonstration

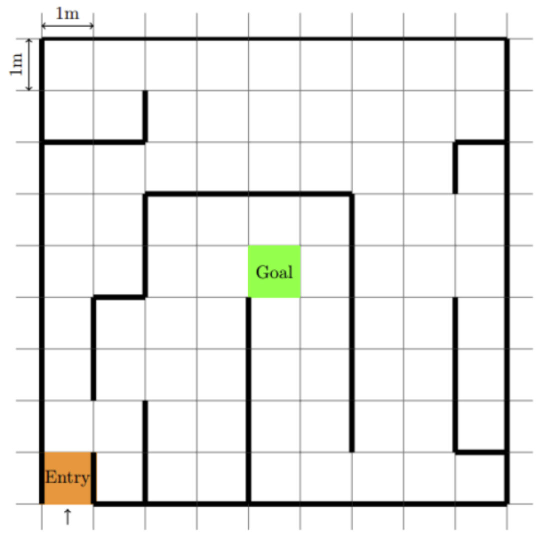


"HIS PATH-PLANNING MAY BE SUB-OPTIMAL, BUT IT'S GOT FLAIR."

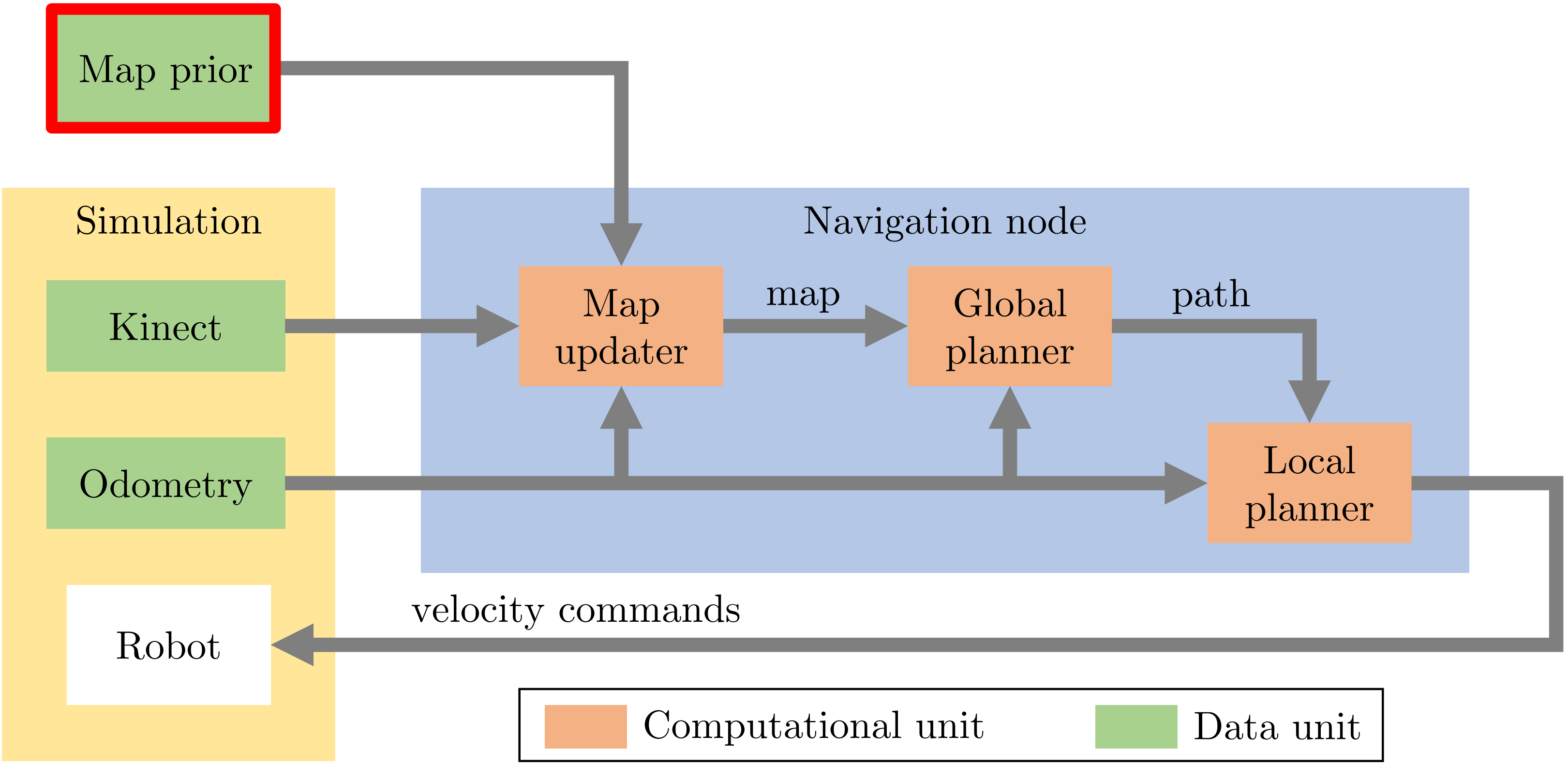
What is to be achieved

- Design a **navigation system** for a mobile robot
- Autonomously navigate from start to goal
- Environment is **not necessarily known**
- Realistic simulation

Easy task for a human,
but complex for a robot...
$\rightarrow$ need to **divide in sub-problems**

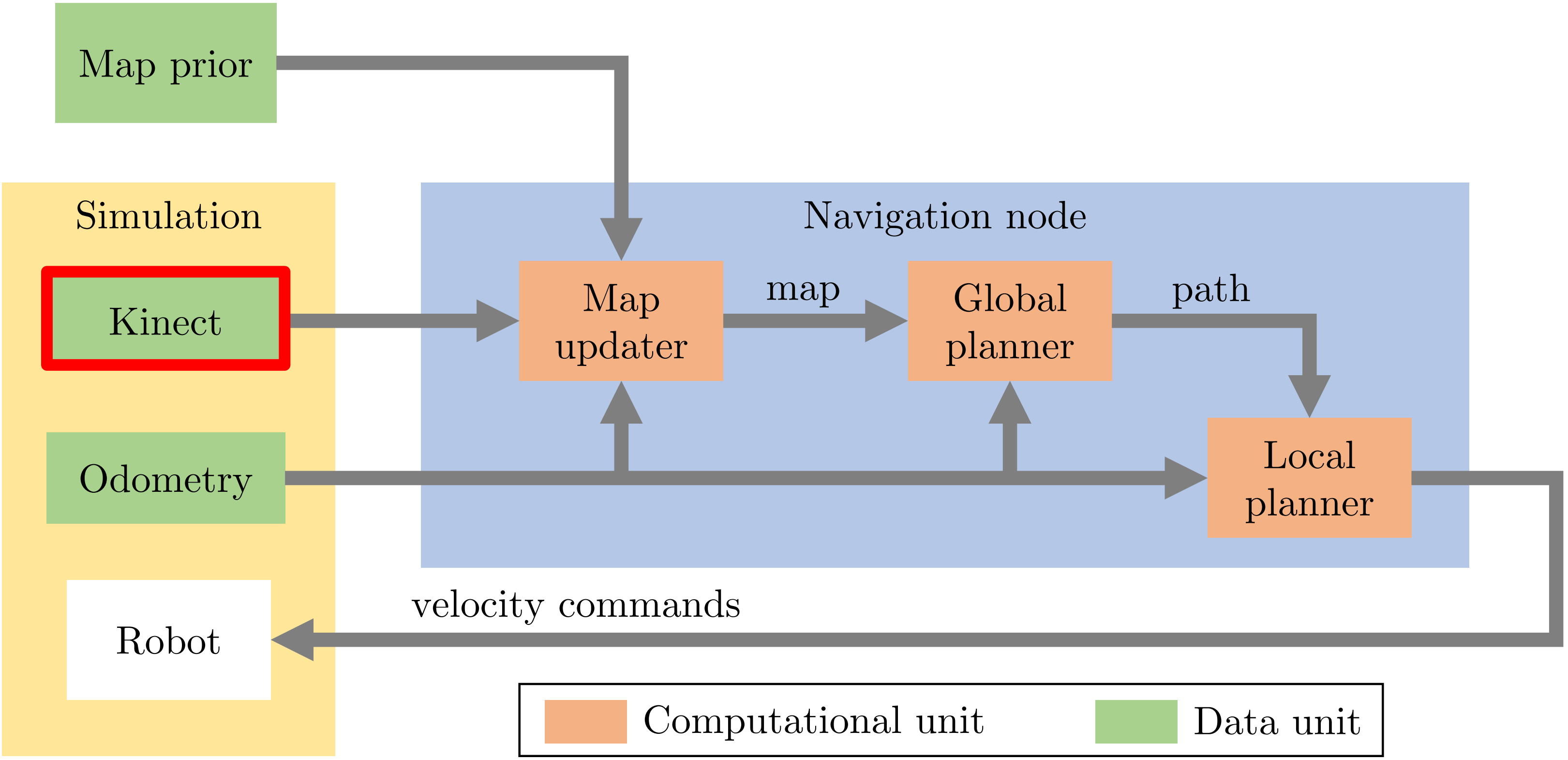


Sub-systems



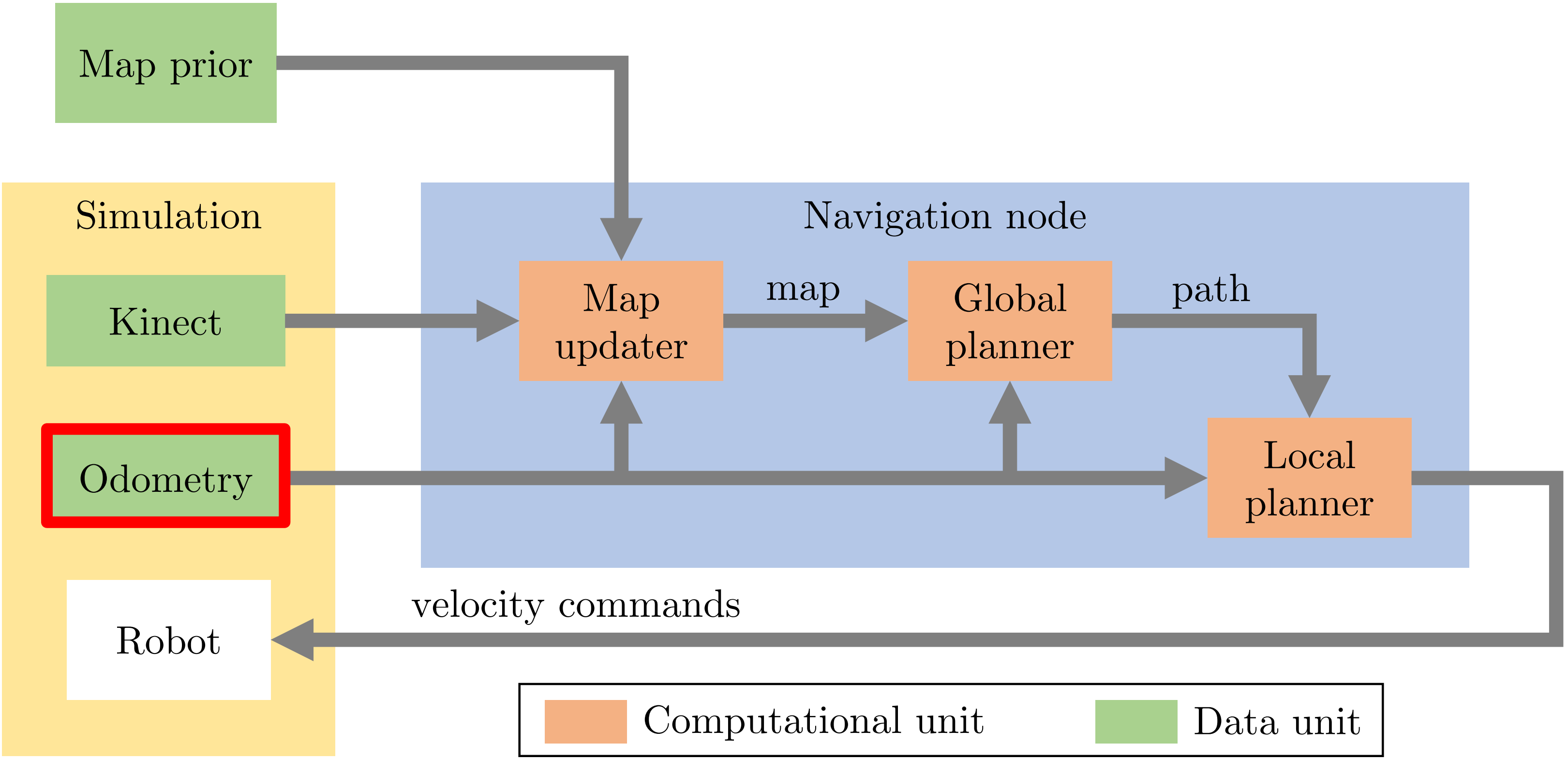
- **Map prior**: what is known beforehand (*a priori*) of the environment

Sub-systems



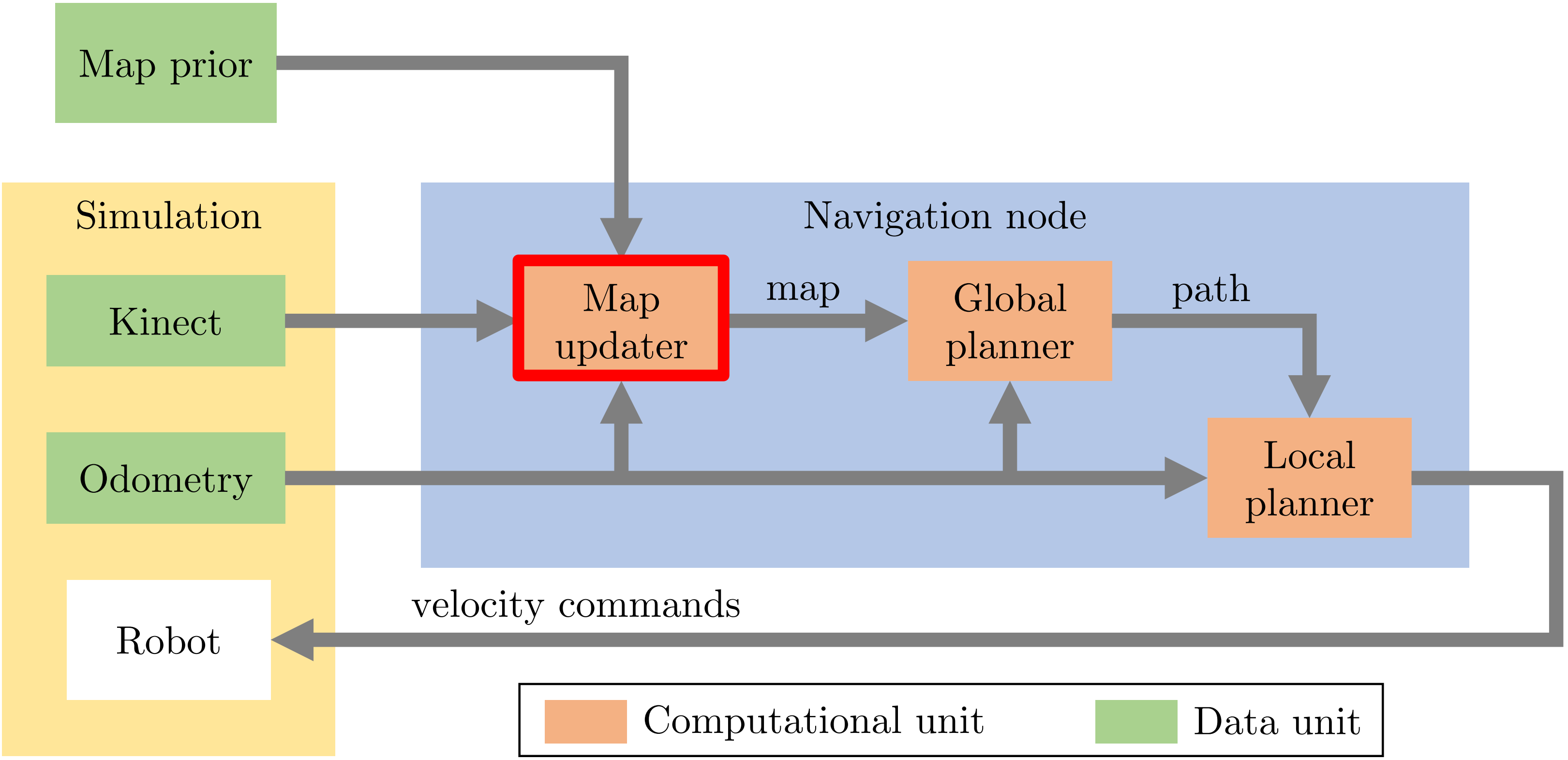
- **Kinect**: a depth camera, returns a point cloud (PC)

Sub-systems



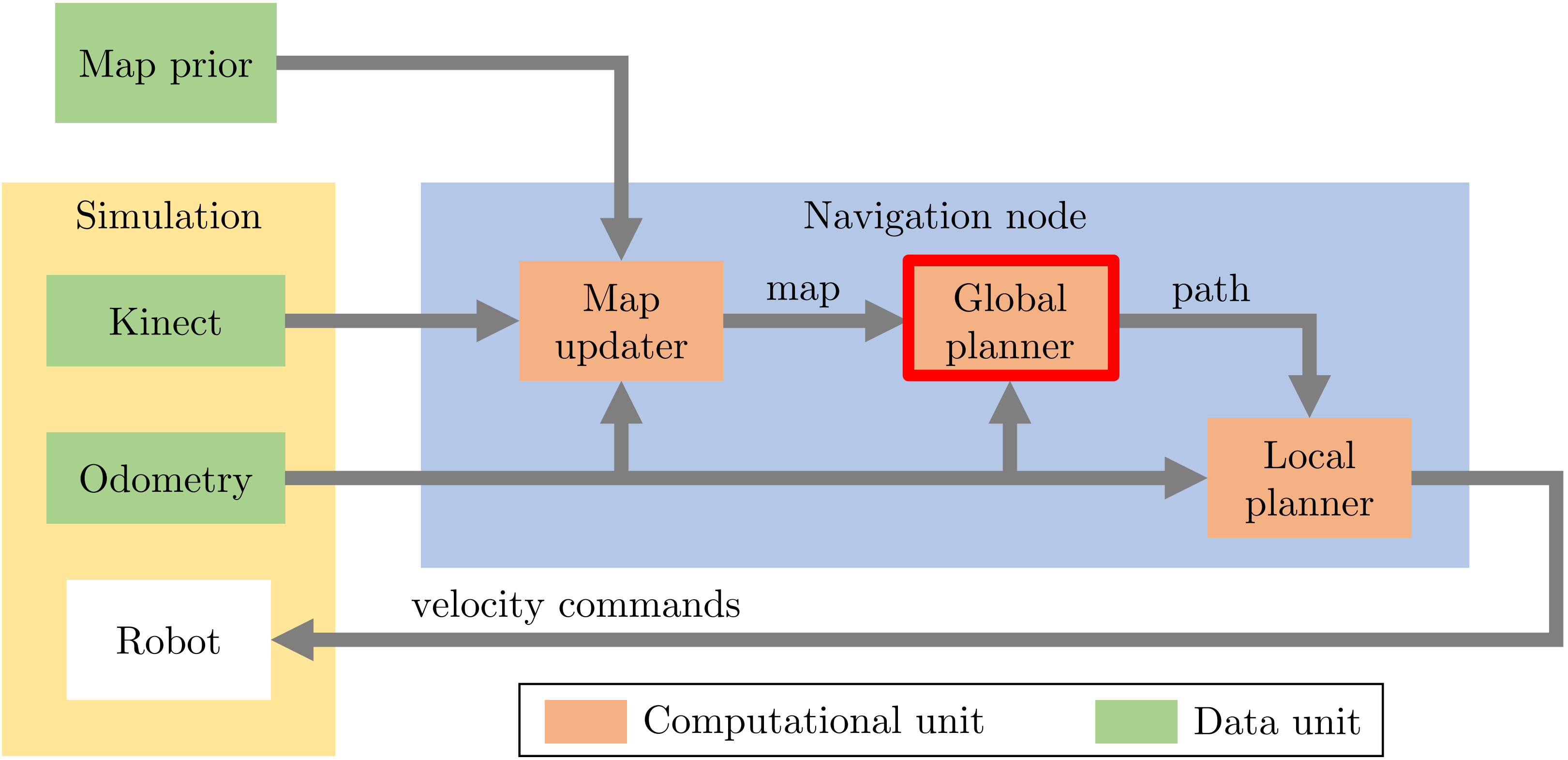
- **Odometry**: the position of the robot in the maze

Sub-systems



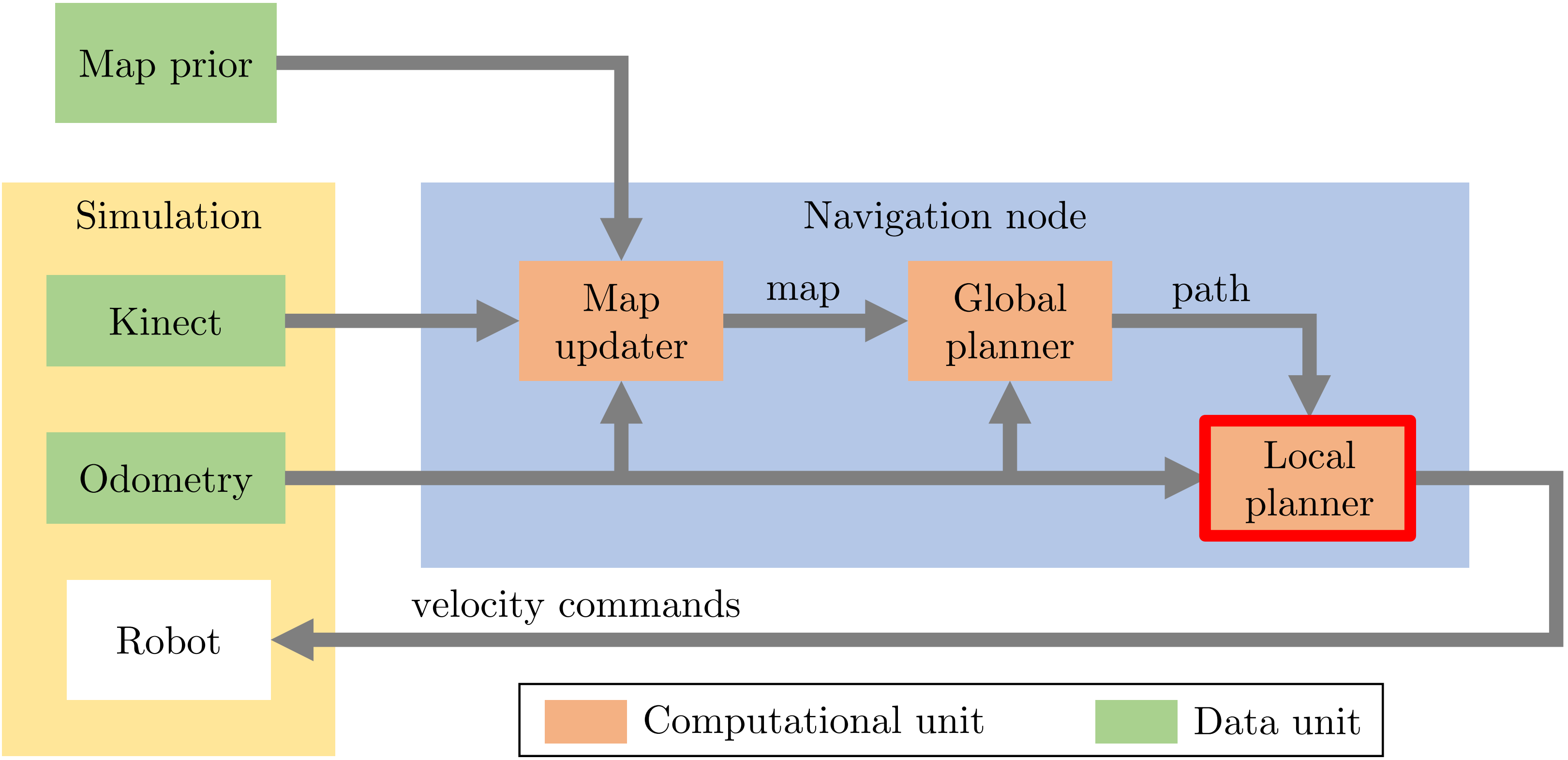
- **Map updater**: process the data to build and update an internal map of the environment

Sub-systems



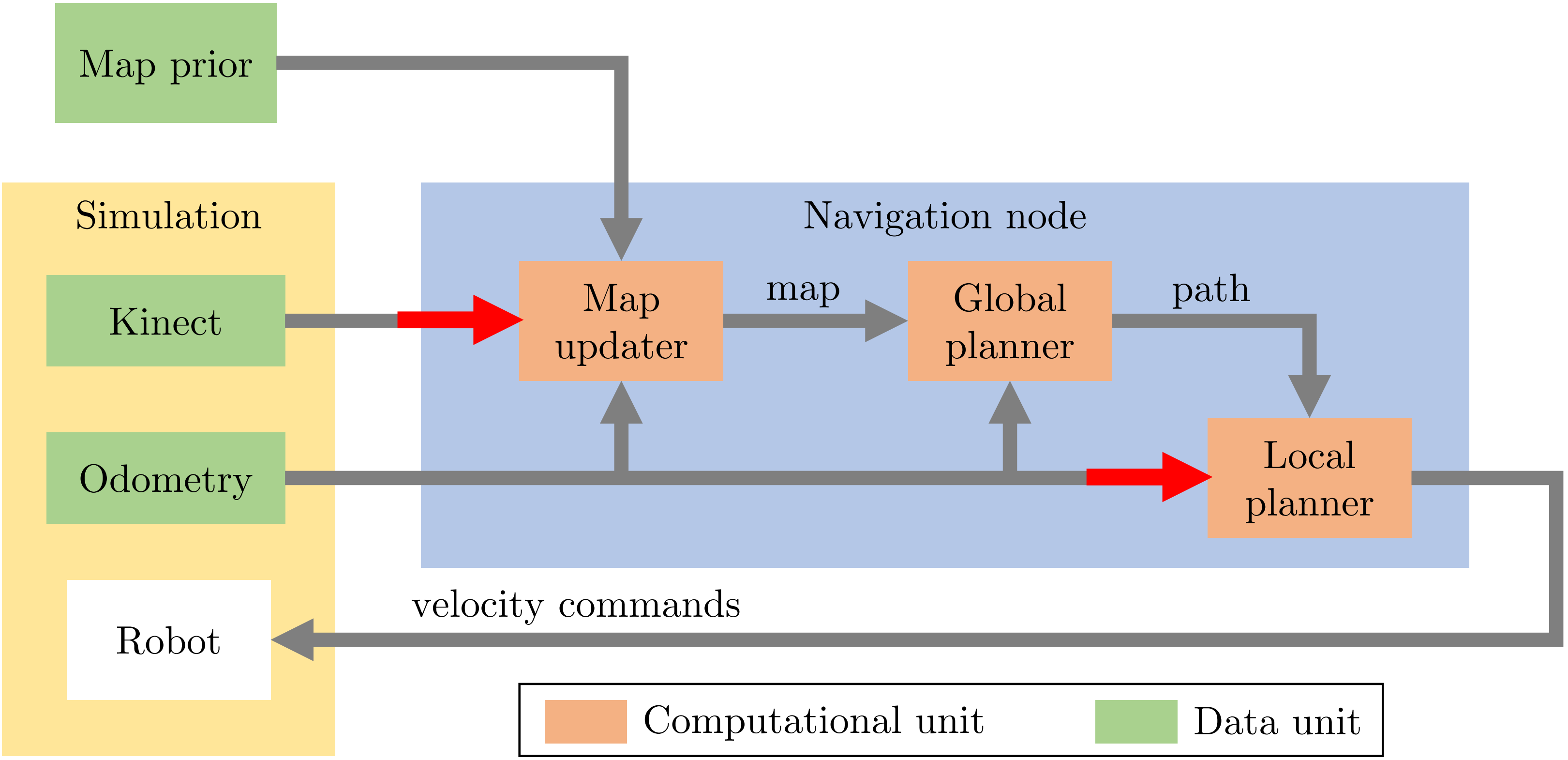
- **Global planner**: compute a path from start to goal using the map

Sub-systems



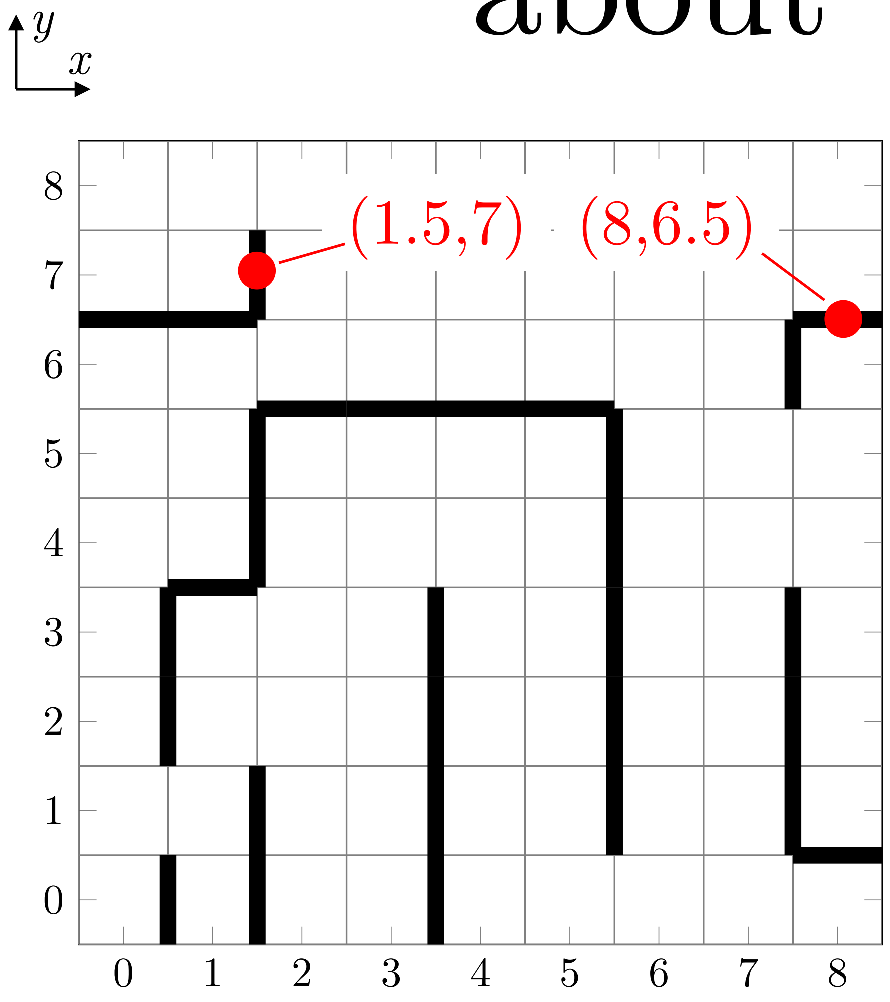
- **Local planner**: ensure that the robot actually follows the path, computes the velocity commands

Implementation



- **Navigation system** = ROS node, in Python
- **Callbacks** trigg. by new Kinect/Odometry data

How to efficiently store information about the environment ?

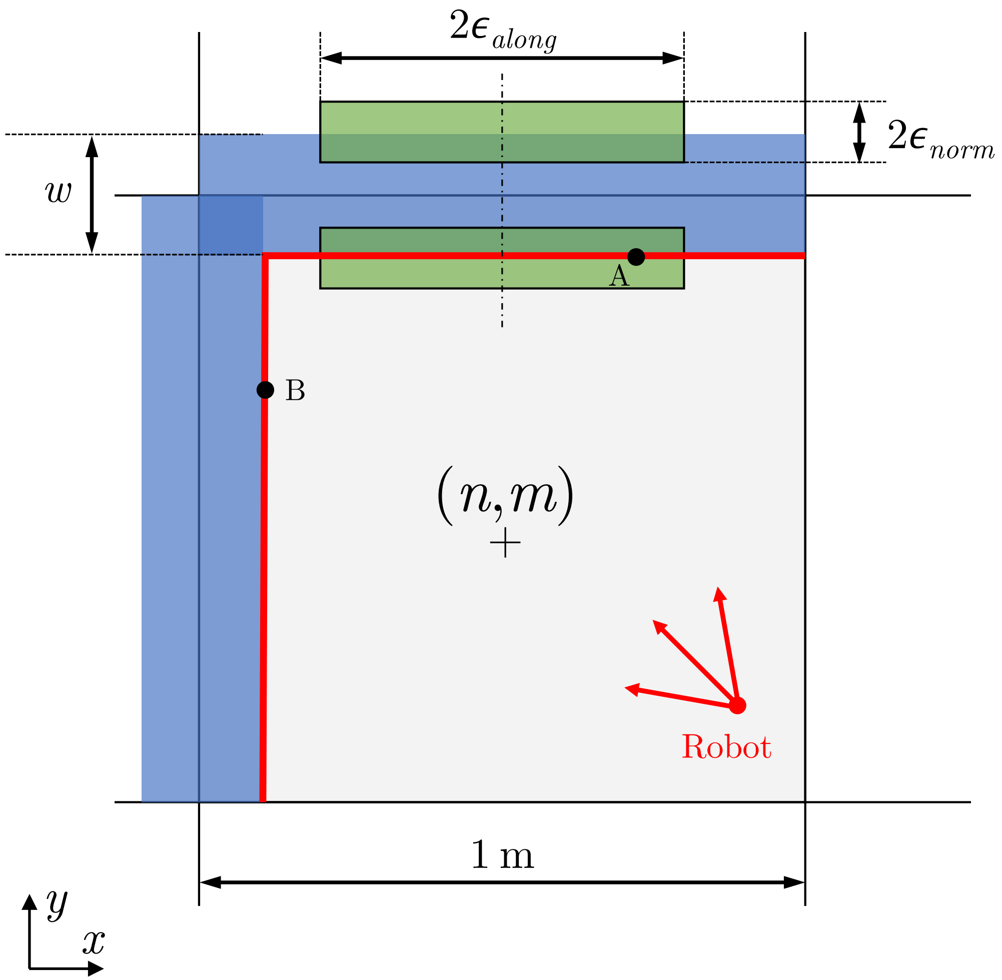


Use the constraints on the maze: walls only at boundaries of cells

→ only store the coordinates of the centres of the walls !

One of the two coordinates is an integer:
x → horizontal wall
y → vertical wall

How to extract walls from the Kinect ?



1. Extract a row of the 2D PC
2. Define a **ROI** with tolerances:

$$(n \pm \epsilon_{along}, m + 0.5 \pm w/2 \pm \epsilon_{norm}) \quad \text{horizontal}$$

$$(n + 0.5 \pm w/2 \pm \epsilon_{norm}, m \pm \epsilon_{along}) \quad \text{vertical}$$

3. For each point: **check** if in any possible ROI
4. If enough points are assumed to belong to a wall
   $\rightarrow$ **add it to the map**

Global planner:
1. Find a path from S to G with A*

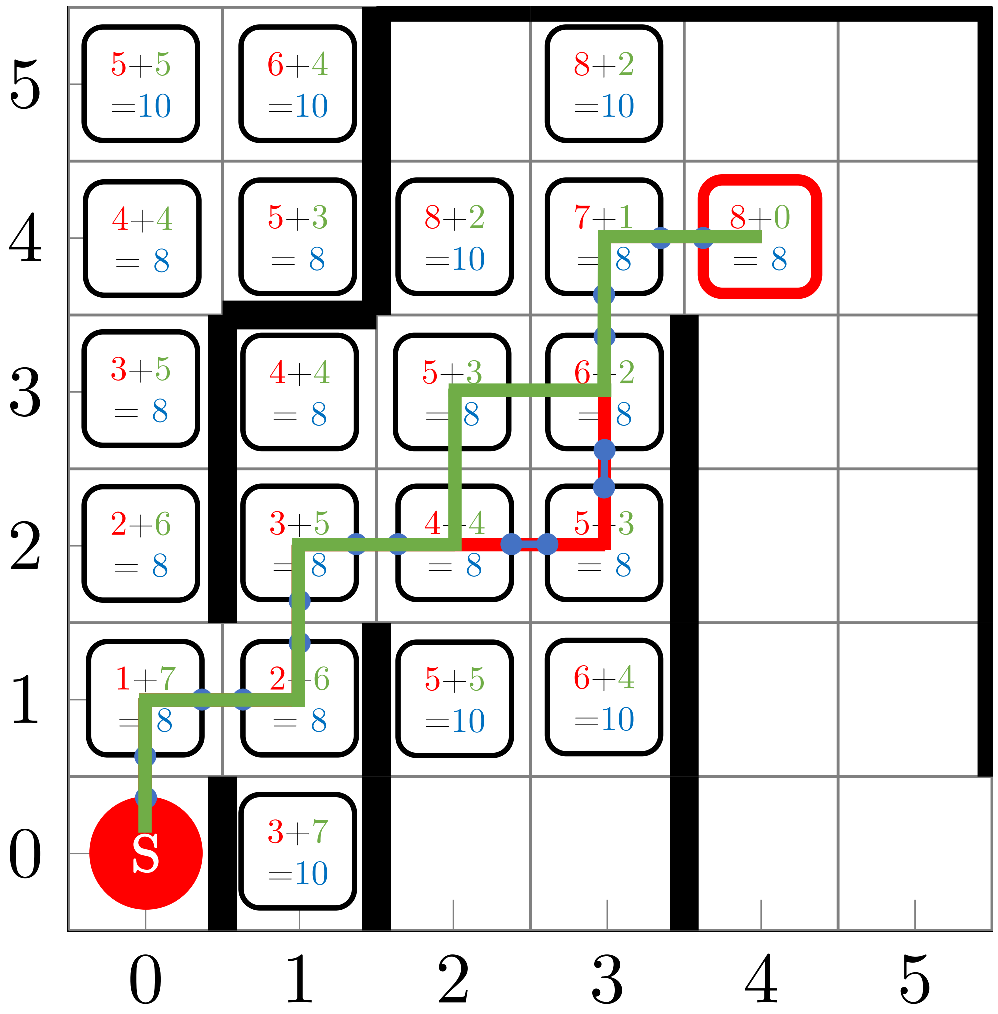
Expand cost from S:

$f = g + h$

g move cost from S

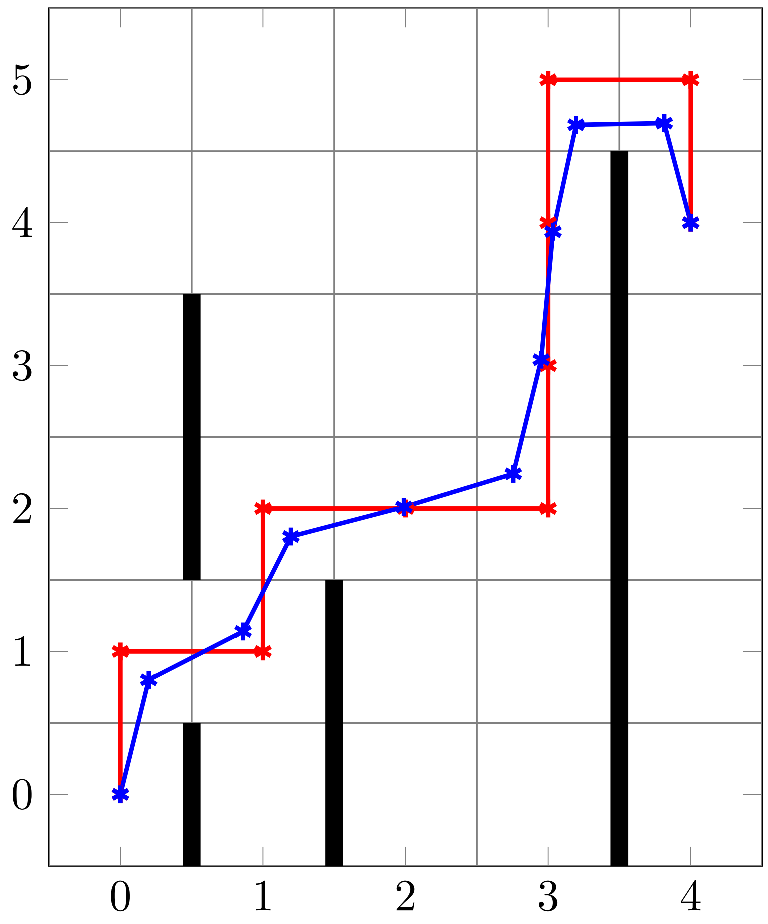
h Manhattan dist. to G

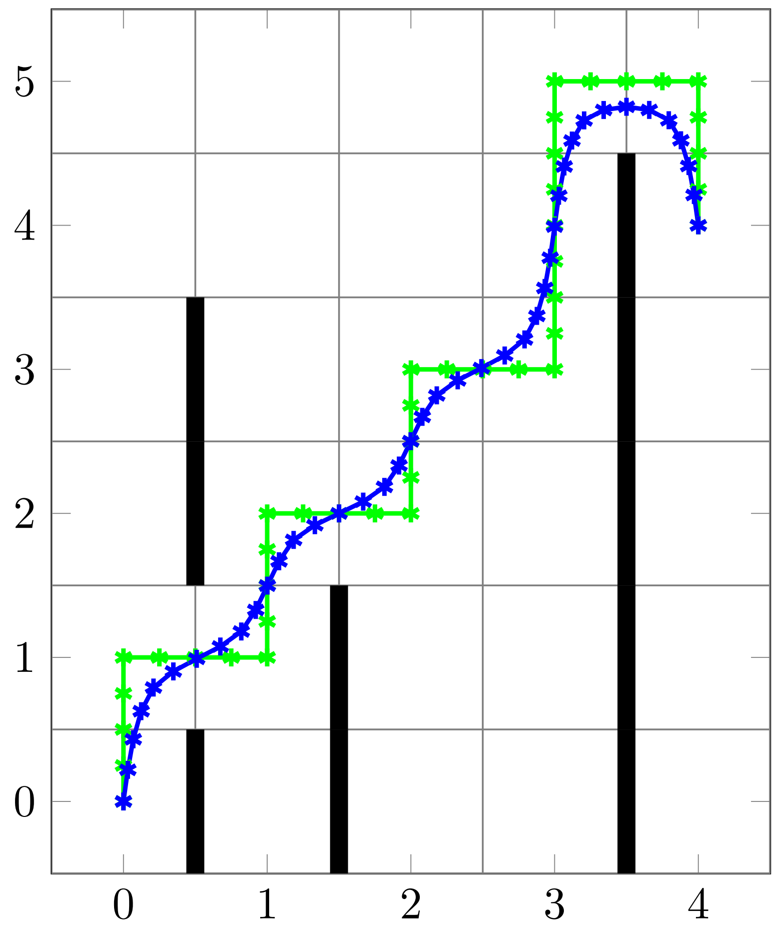
Can dis/encourage turns/lines by tuning g



Global planner: 2. Smooth

Minimise J with Gradient Descent



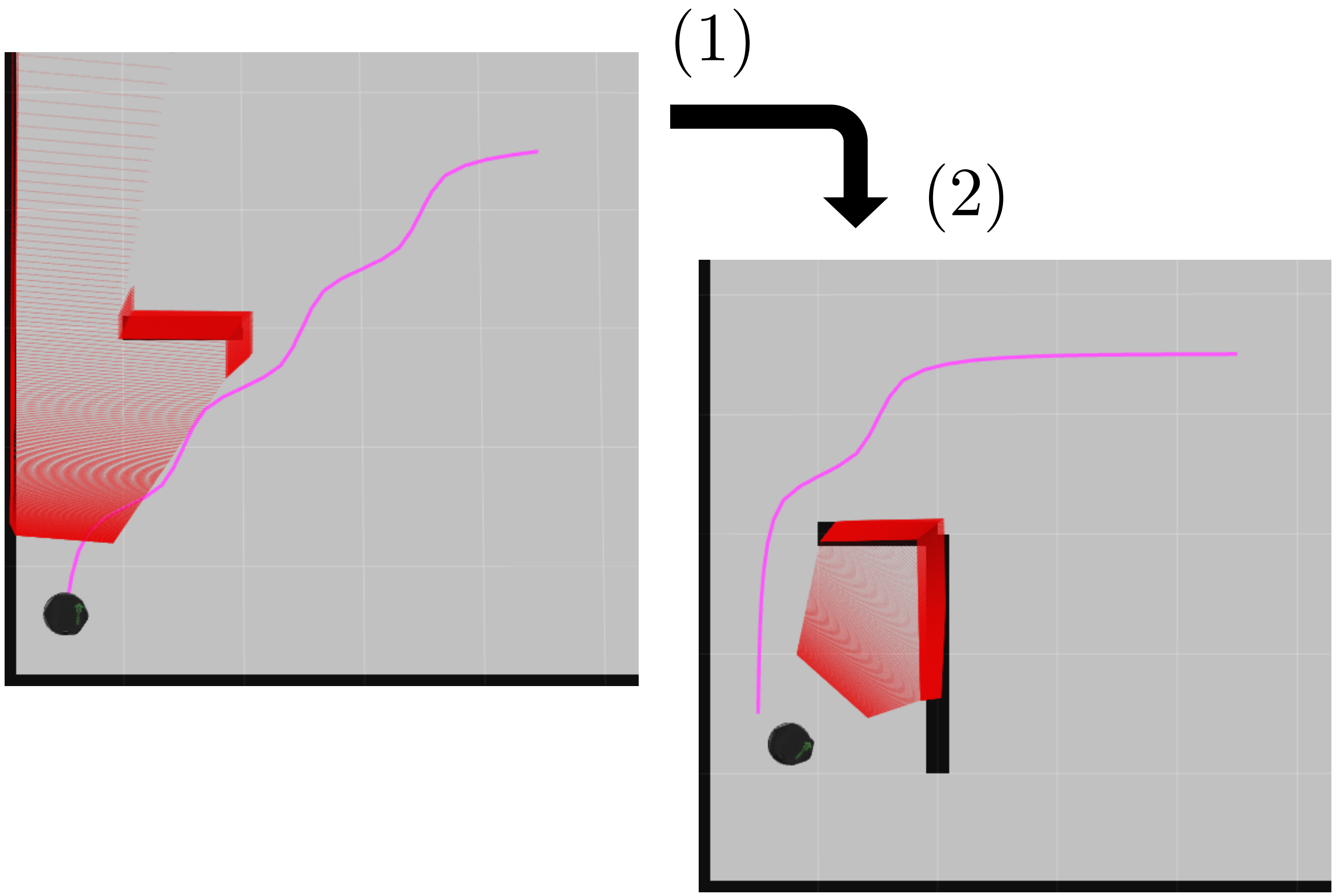


$$J = \frac{1}{2}\sum_{i=1}^{n} \alpha \underbrace{\left(\left(x_i - x_i^{'}\right)^2 + \left(y_i - y_i^{'}\right)^2\right)}_{\text{original path}} + (1-\alpha) \underbrace{\left(\left(x_i^{'} - x_{i+1}^{'}\right)^2 + \left(y_i^{'} - y_{i+1}^{'}\right)^2\right)}_{\text{shortened smooth path}}$$

Known or unknown environment ?

If no map prior:

1. Assume there are no walls (1)
2. Add some as you discover them
3. Update the path if necessary (2)



Local planner: following the path

- PI controller for linear and angular velocities
- Weighted average of the errors on the next k points:

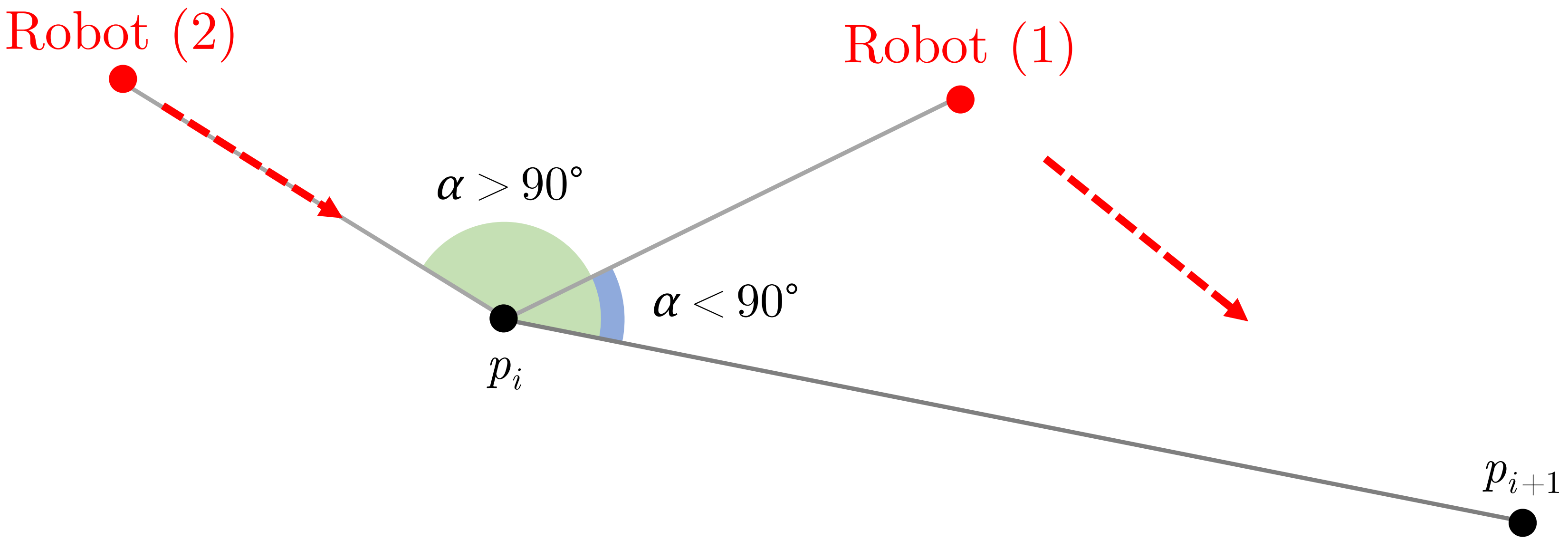
$$\underset{\textit{Orientation error}}{\theta_{PI}} = \frac{\sum_{i=1}^{k} \theta_{j+i}\, d_{j+i}\, w_i}{\sum_{i=1}^{k} d_{j+i}\, w_i} \qquad \underset{\textit{Distance error}}{d_{PI}} = \frac{\sum_{i=1}^{k} d_{j+i}\, w_i}{\sum_{i=1}^{k} w_i}$$

- Act as a **local smoothing**
- Limit speed + anti wind-up

j current point
w_i weights

Local planner: choosing the next point

1. Find the closest point to the current position (here i)
2. Compute the angle α :
   - If smaller than 90° then i+1 is the next
   - Otherwise i



Show time!