ShanghaiTech University 上海科技大学 School of Information Science and Technology 信息科学与技术学院

Introduction to Information Science and Technology (EE 100)
Part II: Intelligent Machines and Robotics
Control

Sören Schwertfeger / **师泽仁**

ShanghaiTech University

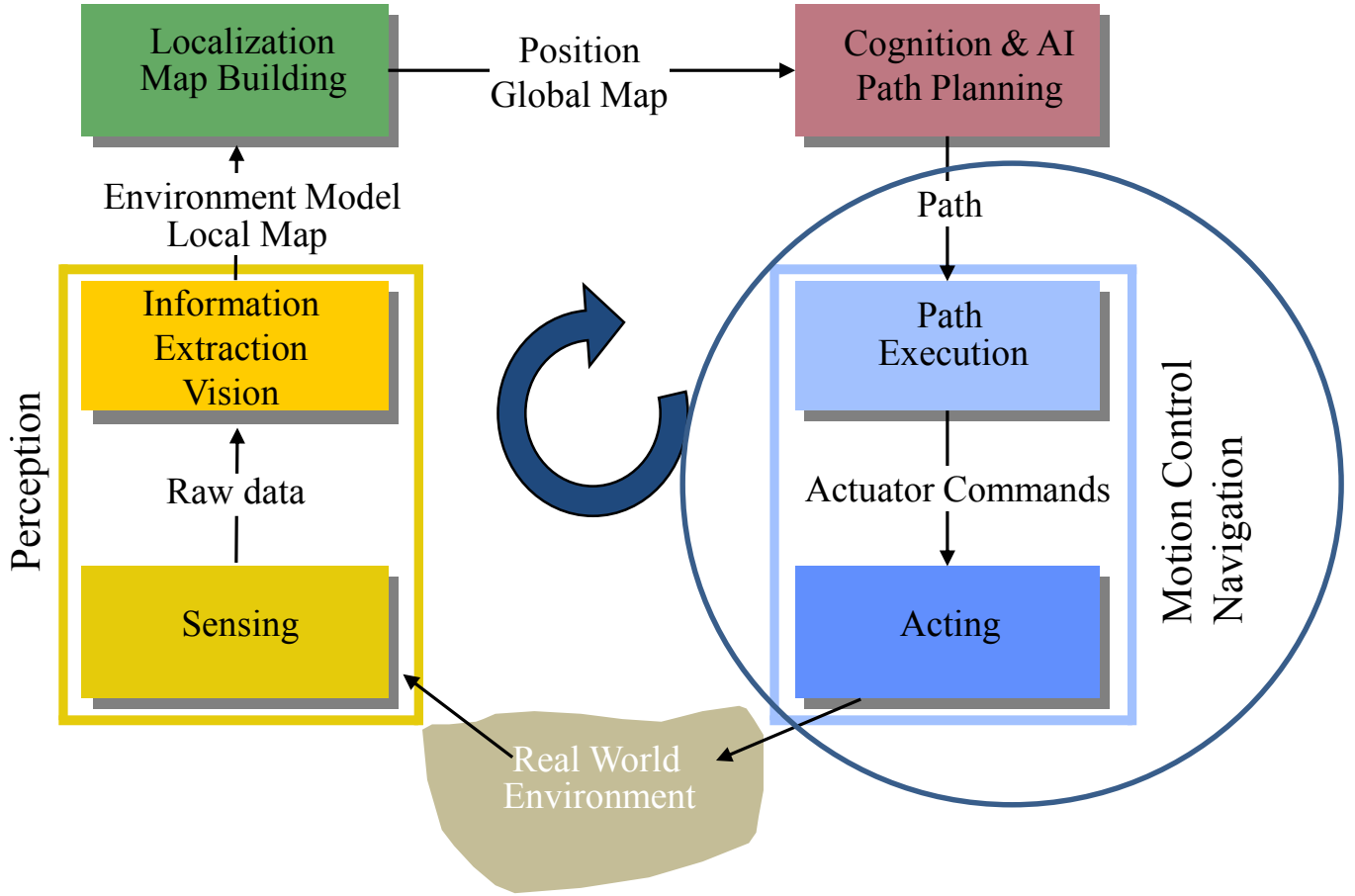
Most important capability

(for autonomous mobile robots)

How to get from place A to place B?

(safely and efficiently)

General Control Scheme for Mobile Robot Systems



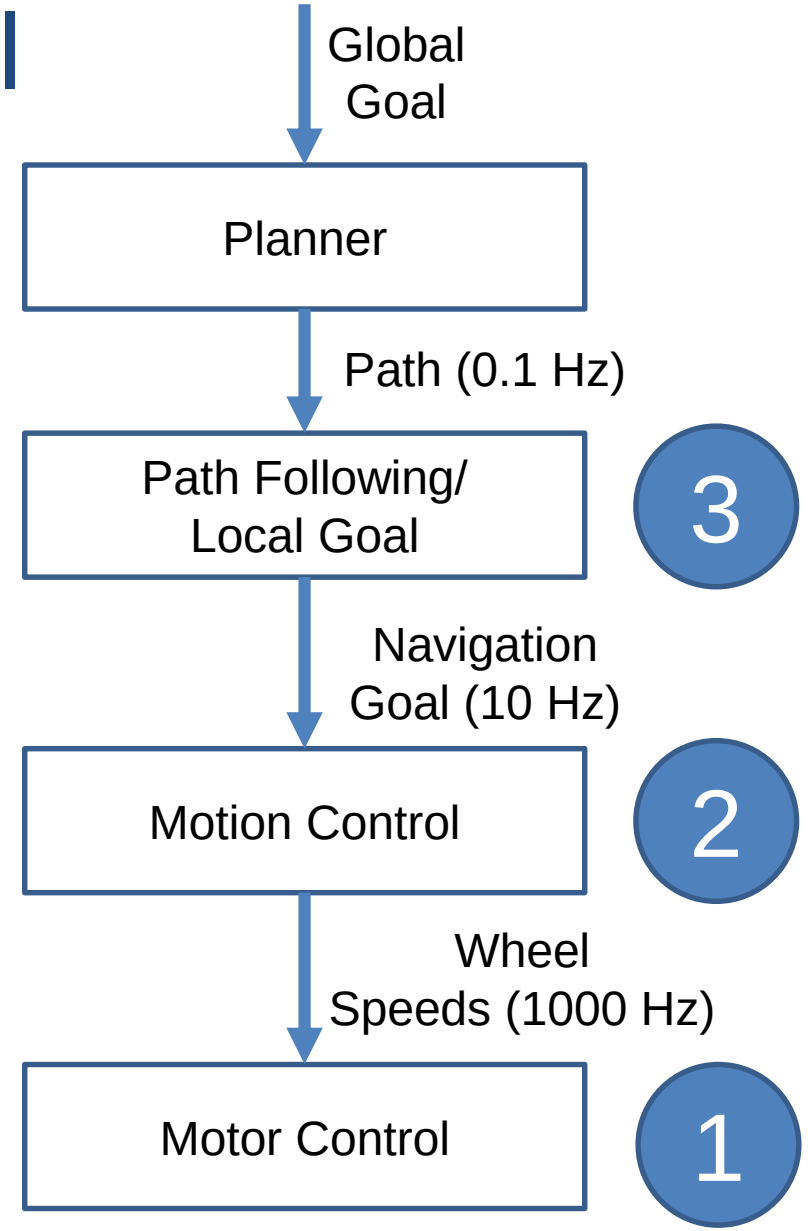
This lecture:
Control and Navigation

With material from Roland Siegwart and Davide Scaramuzza, ETH Zurich

- Autonomous mobile robots move around in the environment. Therefore **ALL** of them:
  - They need to know **where** they **are**.
  - They need to know **where** their **goal** is.
  - They need to know **how** to get there.
- Different levels:
  - Control:
    - How much power to the motors to move in that direction, reach desired speed
  - Navigation:
    - Avoid obstacles
    - Classify the terrain in front of you
    - Predict the behavior (motion) of other agents (humans, robots, animals, machines)
  - Planning:
    - Long distance path planning
    - What is the way, optimize for certain parameters

Navigation, Motion & Motor Control

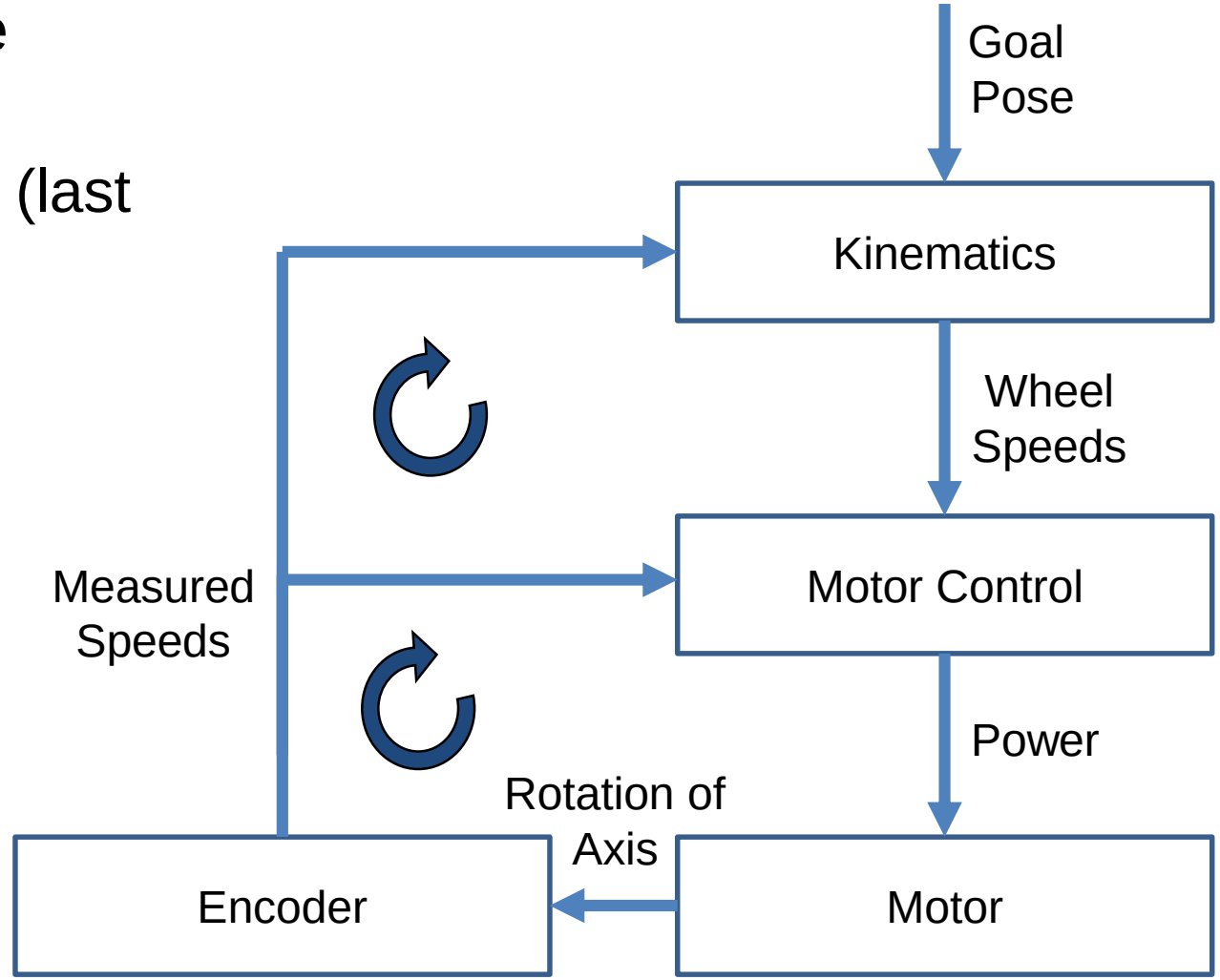
- Navigation/ Motion Control:
  - Where to drive to **next** in order to reach goal
  - Output: motion vector (direction) and speed
  - For example:
    - follow path (Big Model)
    - go to unexplored area (Big Model)
    - drive forward (Small Model)
    - be attracted to goal area (Small Model)
- Motion Control:
  - How use propulsion to achieve motion vector
- Motor Control:
  - How much power to achieve propulsion (wheel speed)



MOTOR & MOTION CONTROL

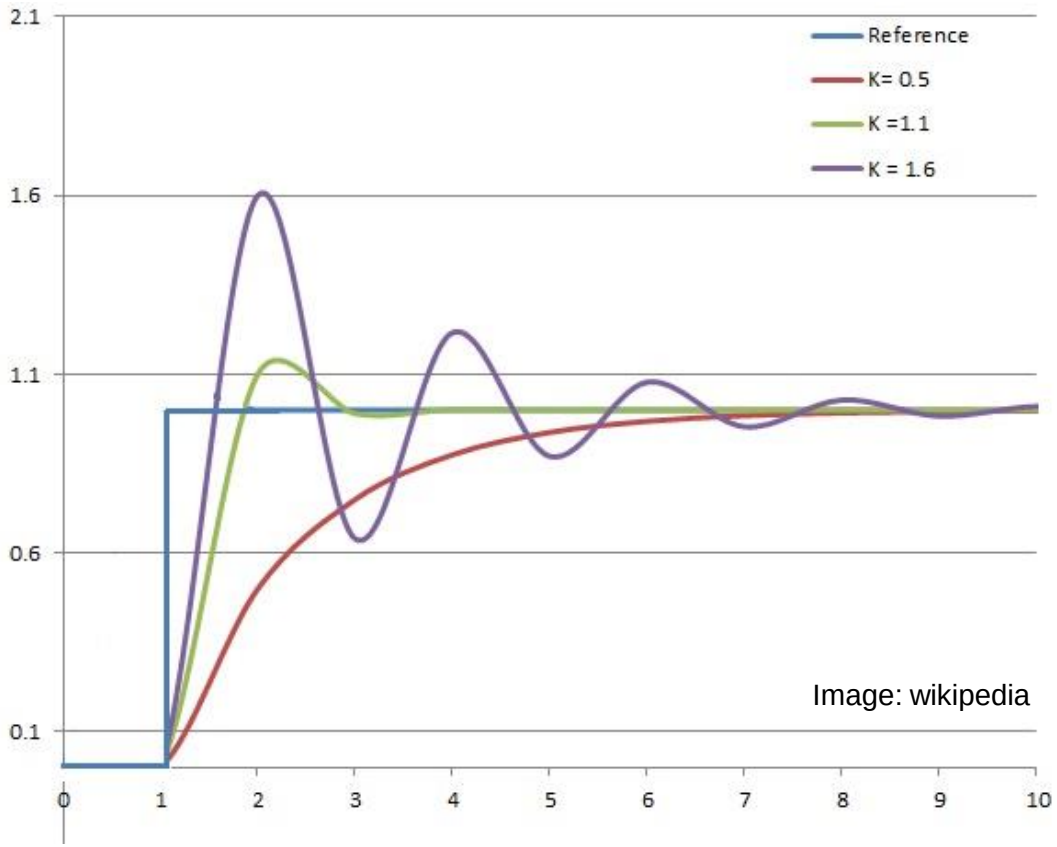
Overview

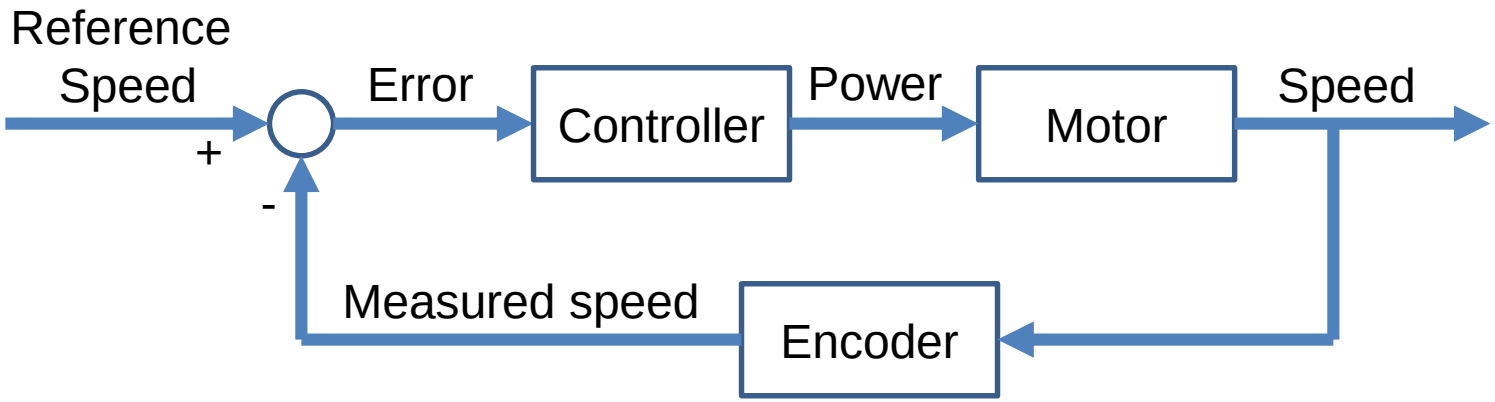
- Assume we have a goal pose (close by)
- Calculate Inverse Kinematics (last lecture, LaValle) =>
- Desired wheel speeds
  - Typically not just one wheel =>
  - Many motor controllers, motors, encoders
- Motor control loop
- Pose control loop



Motor Control

- How much power is needed for desired reference speed?
  - Inertia of the motor + robot:
  - Need more power during acceleration of robot vs. constant speed
  - Up hill/ down hill different power needs
  - Motors are even used to break the robot!
- Closed loop control (negative feedback)
- Proportional-Integral-Derivative controller (PID)
- Motor speed reacts slowly to power changes

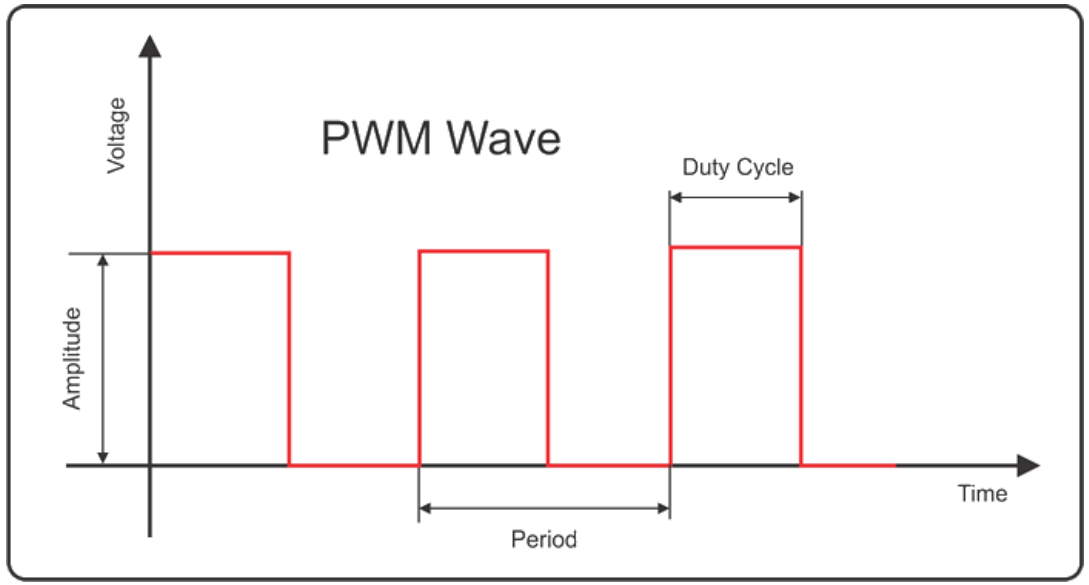


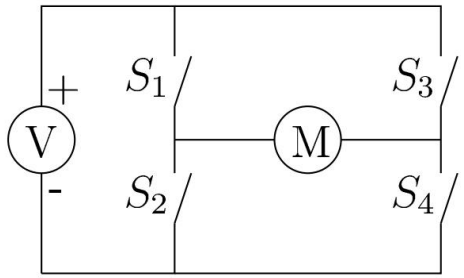


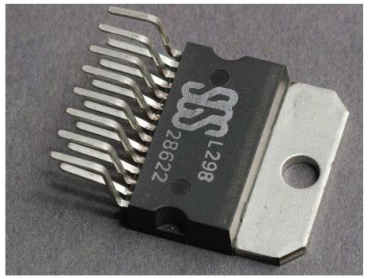
Motor Driver

- How can Controller control power?
- Pulse Width Modulation (PWM)
  - Frequency in kHz (= period less than 1ms)
- Signal (e.g. 3V) to Motor Driver
- Motor Driver switches according to signal and direction => H-bridge
- Output: Power (e.g. 24V, 4A)
  - PWM modulated
  - Sound of motors from PWM!

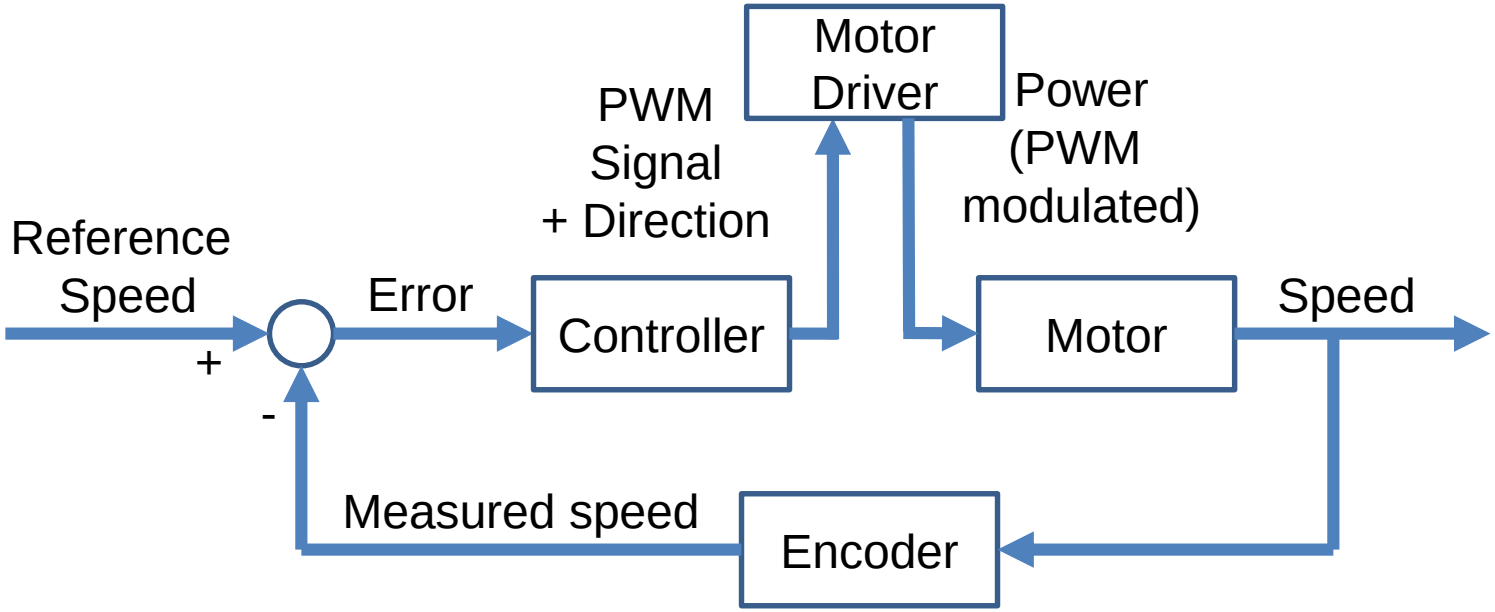
Image: zembedded.com







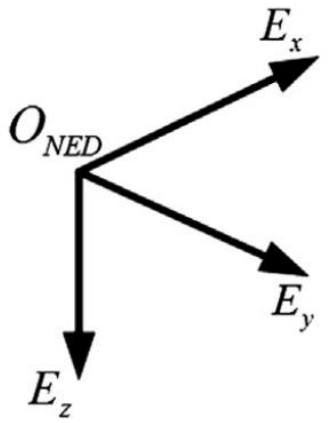
Steven M. LaValle, “Mobile Robotics: An Information Space Approach”



Pose Control

- Ground robots:
- Check difference current pose ⇔ desired pose
  - Using wheel encoders => Forward kinematics => Odometry
  - Using external localization (e.g. in map, via markers)
  - => slow update rate (10 – 0.1 Hz)

- Quadcopters:
  - Translation: roll or pitch
  - Very sensitive to roll and pitch
  - => keep robot at desired roll/ pitch angles
  - Very high update rate > 1000 Hz



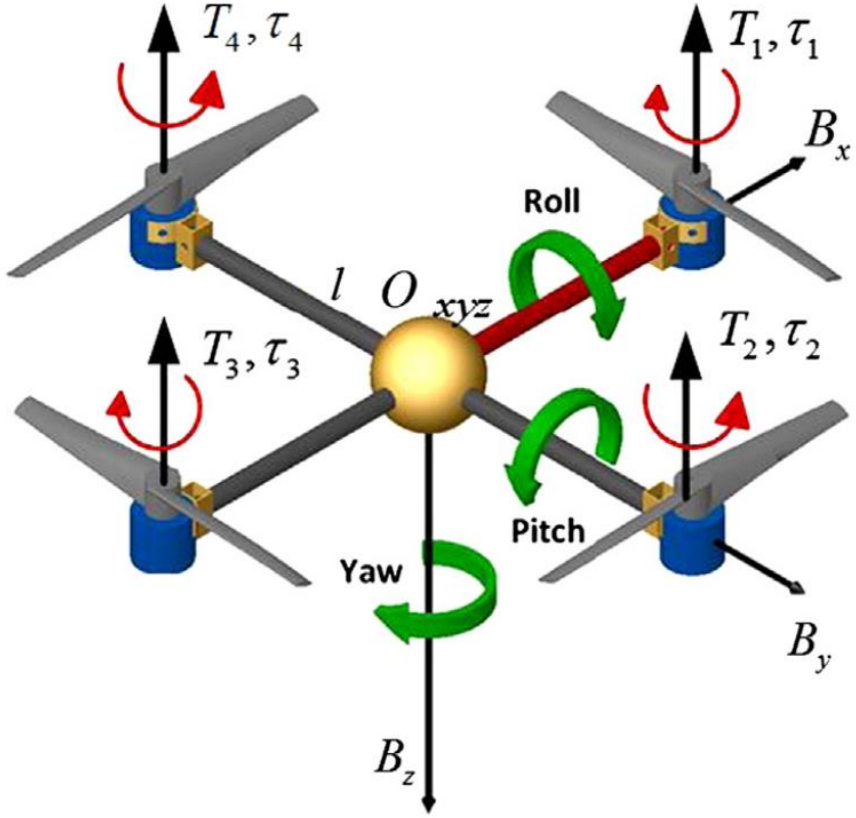


Image from: Weihua Zhao, Tiauw Hiong Go, “Quadcopter formation flight control combining MPC and robust feedback linearization”

The Flying Machine Arena

Cooperative Quadrocopter Ball Throwing and Catching

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Video

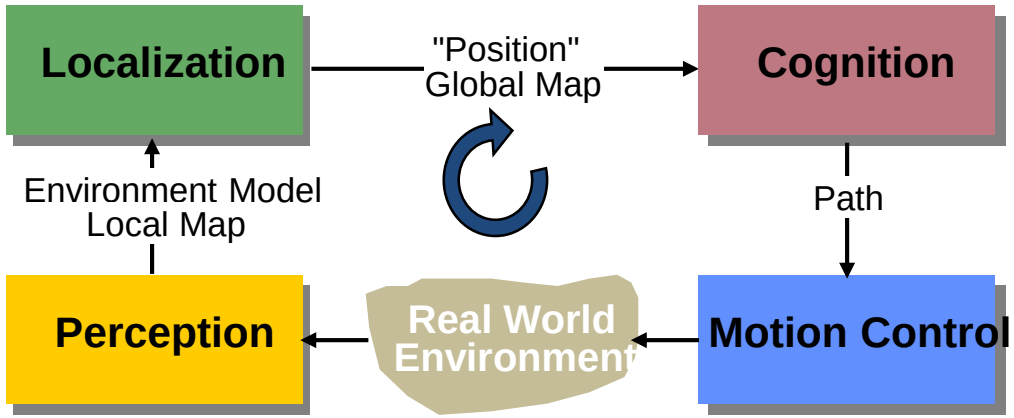
- Localization of robot pose via external cameras
- Control loops:
  - Task control: Position of net => position of robots
    - Take dynamics into account
  - Pose control: Position of each quadcopter
  - Flight control: Roll and pitch to reach pose
  - Motor control

BEHAVIOR BASED ROBOTICS

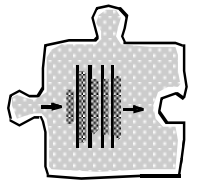
"Small Model" – "Little Brain"

Control Architectures / Strategies

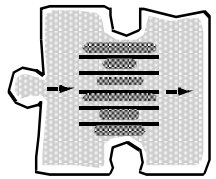
- Control Loop
  - **dynamically changing**
  - **no compact model available**
  - **many sources of uncertainty**



- Two Approaches
  - Classical AI (Big Model)
    - complete modeling
    - function based
    - horizontal decomposition

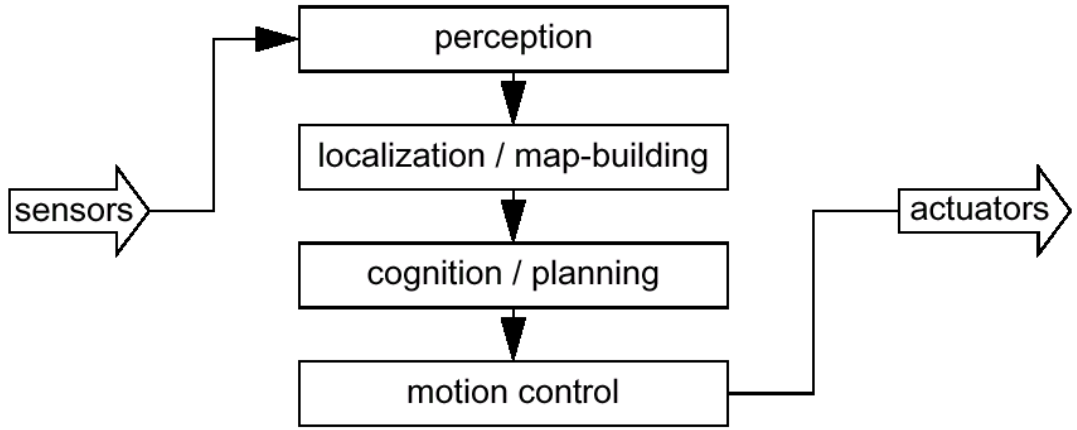


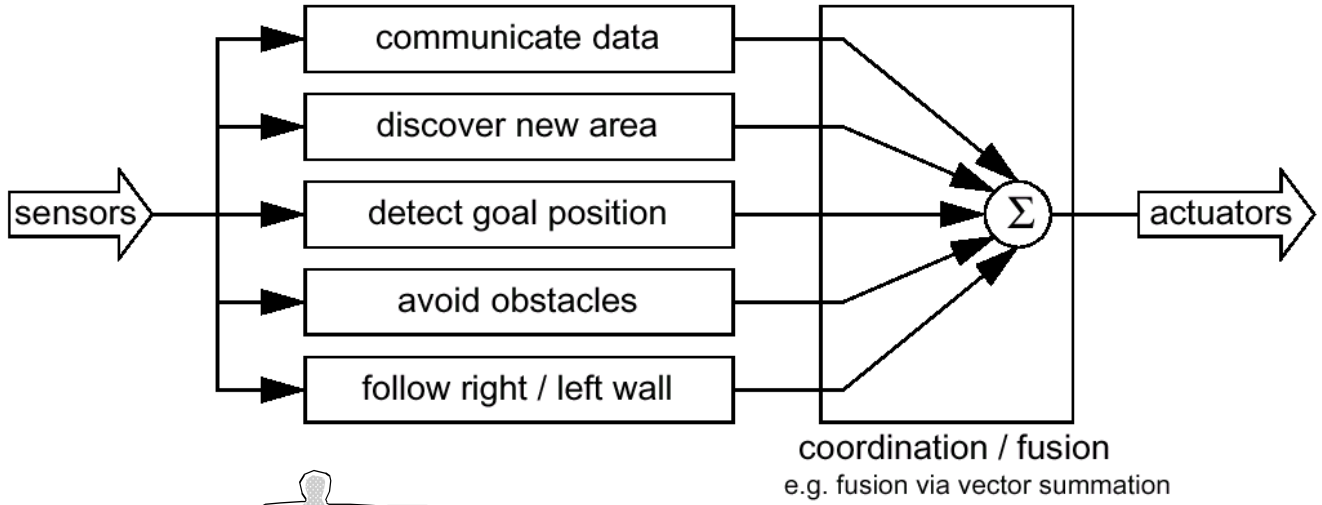
  - New AI (Nouvelle AI; Small Model; Behavior Based Robotics)
    - sparse or no modeling
    - behavior based
    - vertical decomposition
    - bottom up

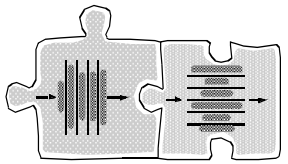


Two Approaches

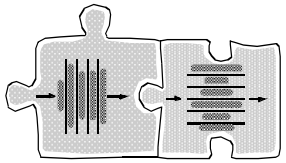
- Classical AI
  (model based navigation)
  - complete modeling
  - function based
  - horizontal decomposition
- New AI
  (behavior based navigation)
  - sparse or no modeling
  - behavior based
  - vertical decomposition
  - bottom up
- Possible Solution
  - Combine Approaches (= Hybrid Approach)

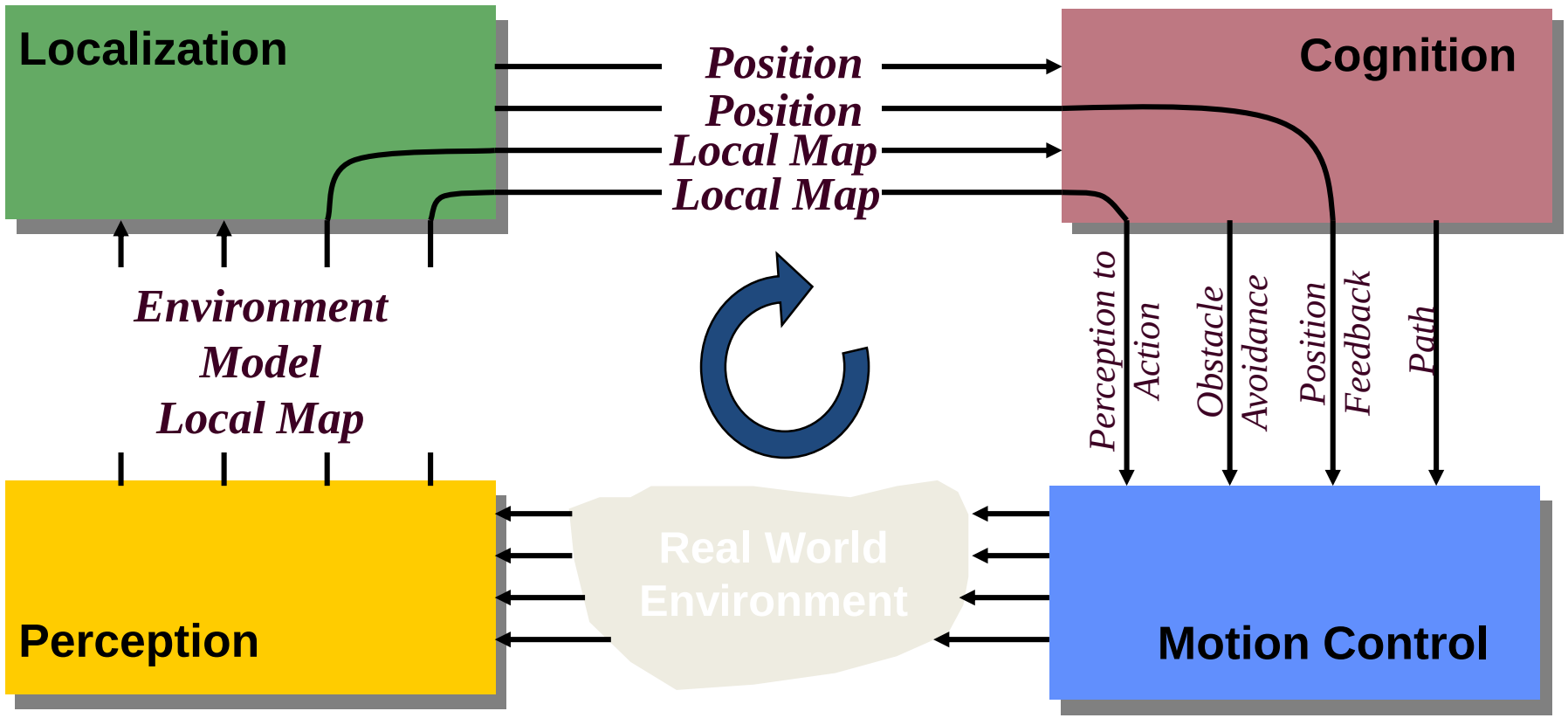






Mixed Approach Depicted into the General Control Scheme



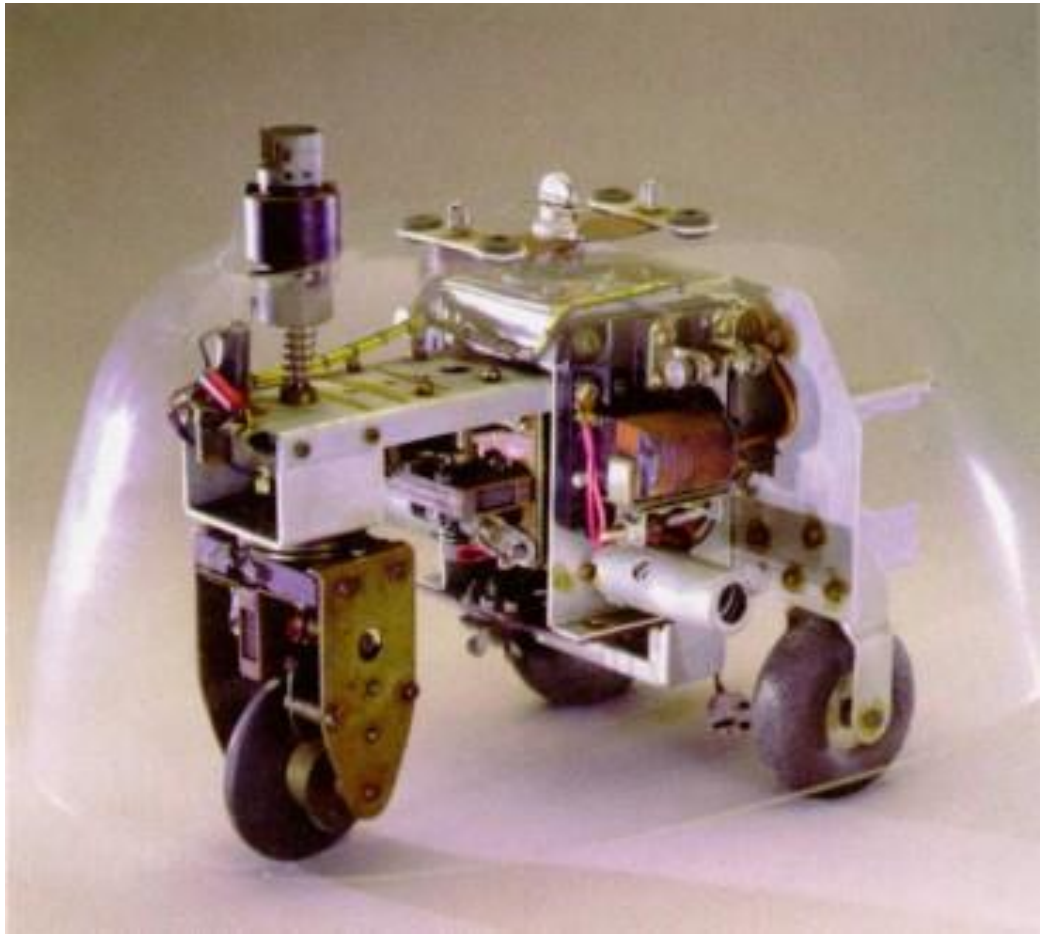


Emergence

- Adaptive behavior
  - emerges from complex interactions between body, world and brain
- Non-centrally controlled (or designed) behavior
  - results from the interactions of multiple simple components
- Meanings:
  - Surprising situations or behaviors
  - Property of system not contained in any of its parts
  - Behavior resulting from agent-environment interaction not explicitly programmed
- Ant colony:
  - self-organized; simple individuals; local interactions =>
  - emergent behavior - No global control

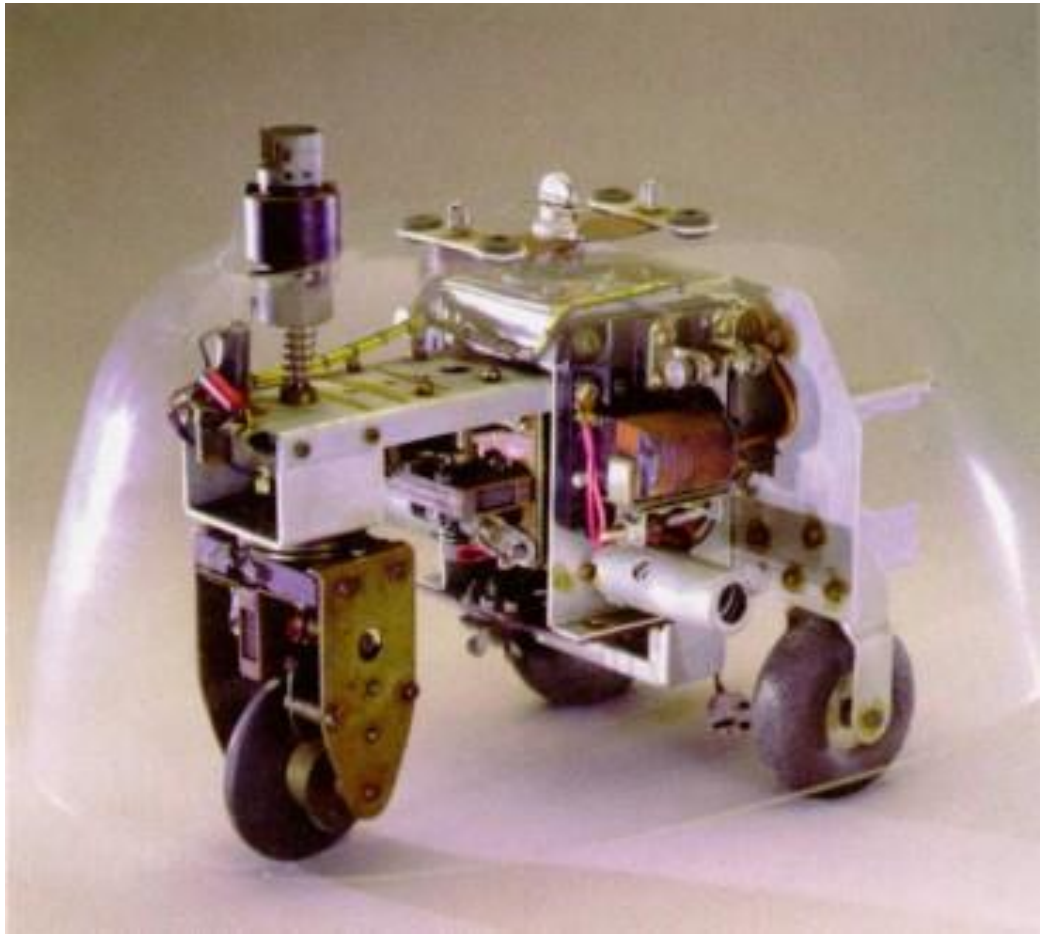
Grey Walter's Tortoise

- Turtle shape robots 1949
- Purely analogue electronics
- Phototaxis: go towards the light
- Sensors:
  - 1 photocell,
  - 1 bump sensor
- 2 motors
- Reactive control



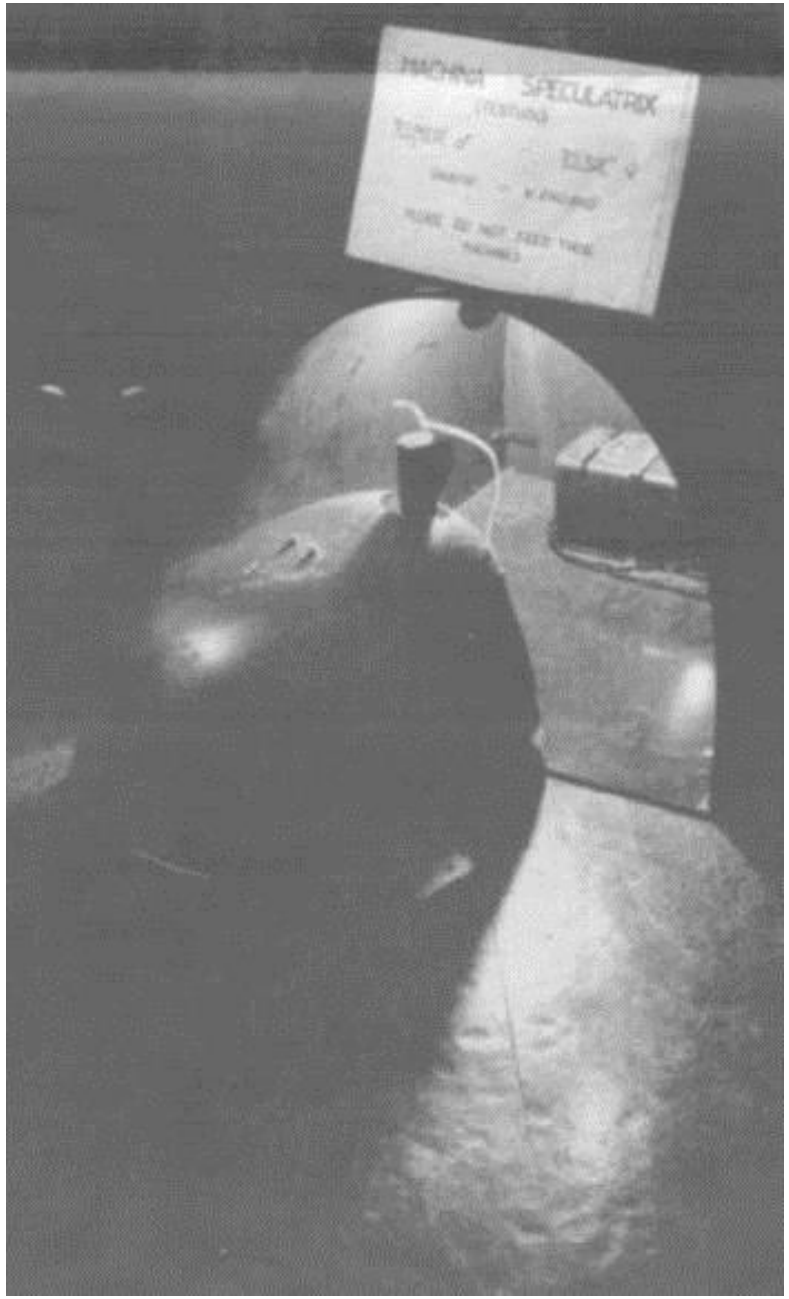
Grey Walter's Tortoise

- Behaviors:
  - Seek light
  - Head toward weak light
  - Back away from bright light
  - Turn and push (obstacle avoidance)
  - Recharge battery



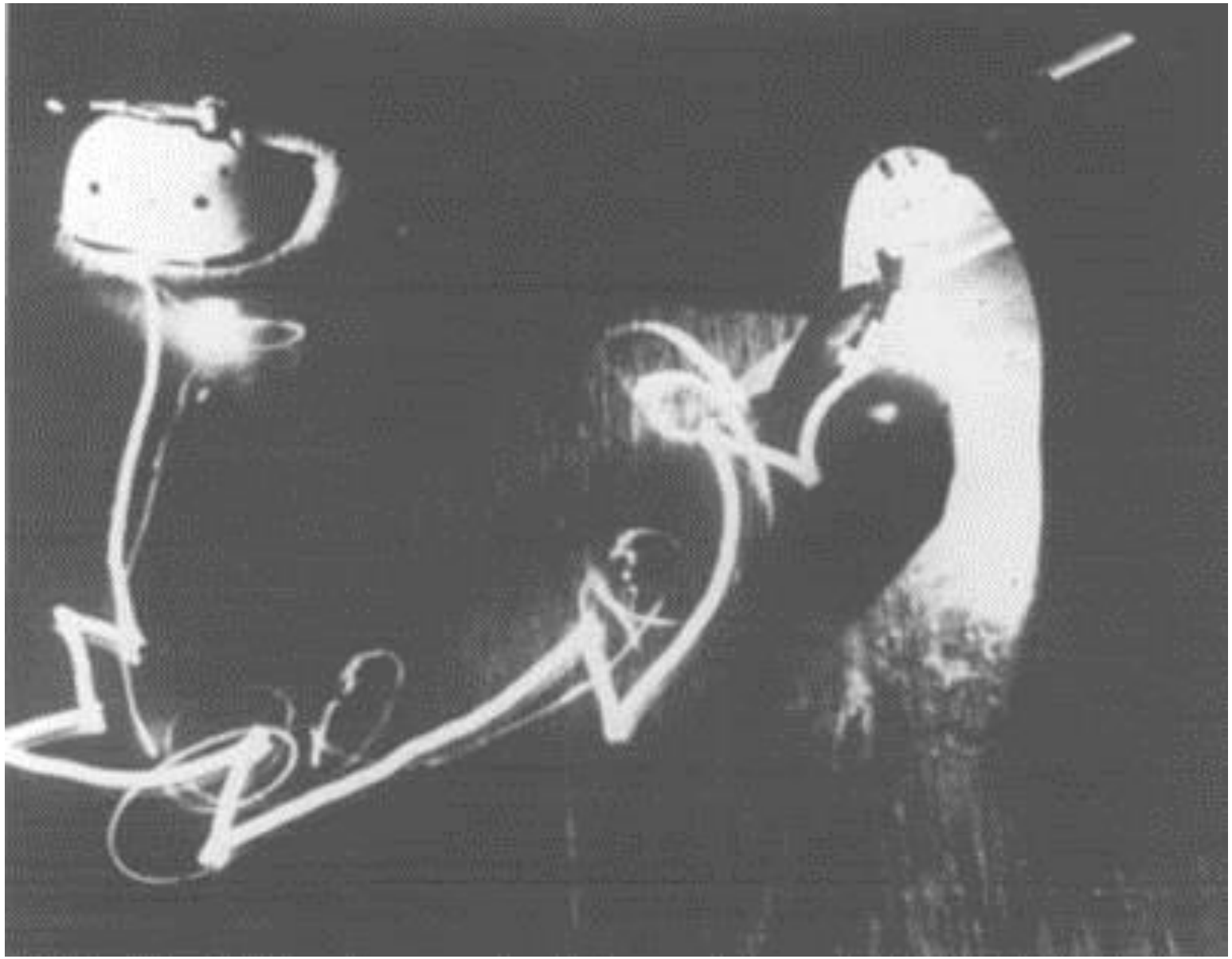
Turtle Principles

- Simple is better
  - e.g., clever recharging strategy
- Exploration/ speculation: keeps moving
  - except when charging
- Attraction:
  - motivation to approach light
- Aversion:
  - motivation to avoid obstacles, slopes



Tortoise behavior

- A path: a candle on top of the shell

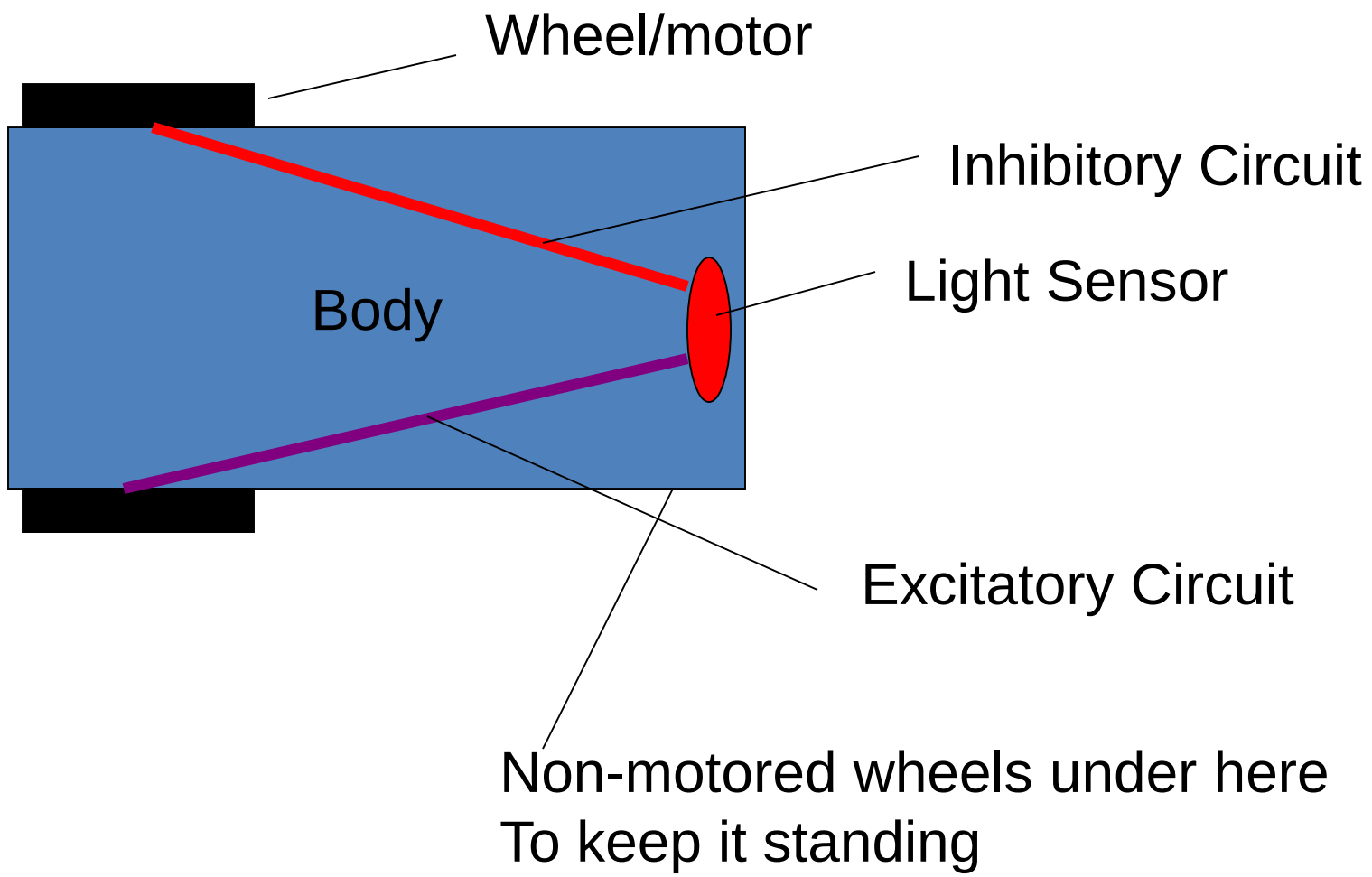


Video …


BRITISH
PATHÉ

Braitenberg's Vehicles

- Valentino Braitenberg (1926)
- 1984: "Vehicles: Experiments in Synthetic Psychology"

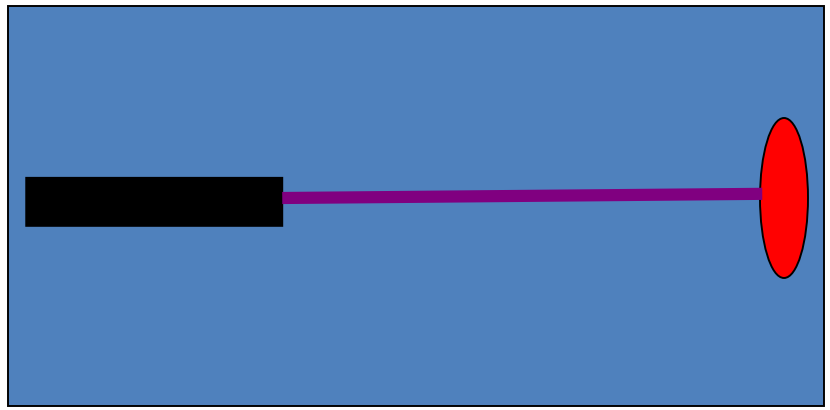


Material: Tommy Walsh http://my.fit.edu/~twalsh/Braitenberg%20Vehicles.ppt

Definitions

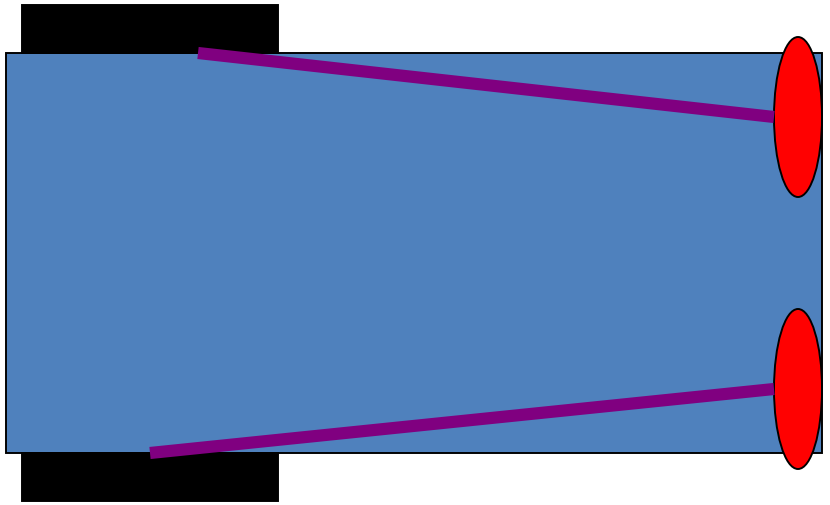
- Inhibitory circuit: when sensor gets activated, motor slows
- Excitatory circuit: when sensor gets activated, motor speeds
- Sensor is a light sensor, unless otherwise noted

Vehicle 1: Alive



Basic Braitenberg vehicle:
Goes towards light source

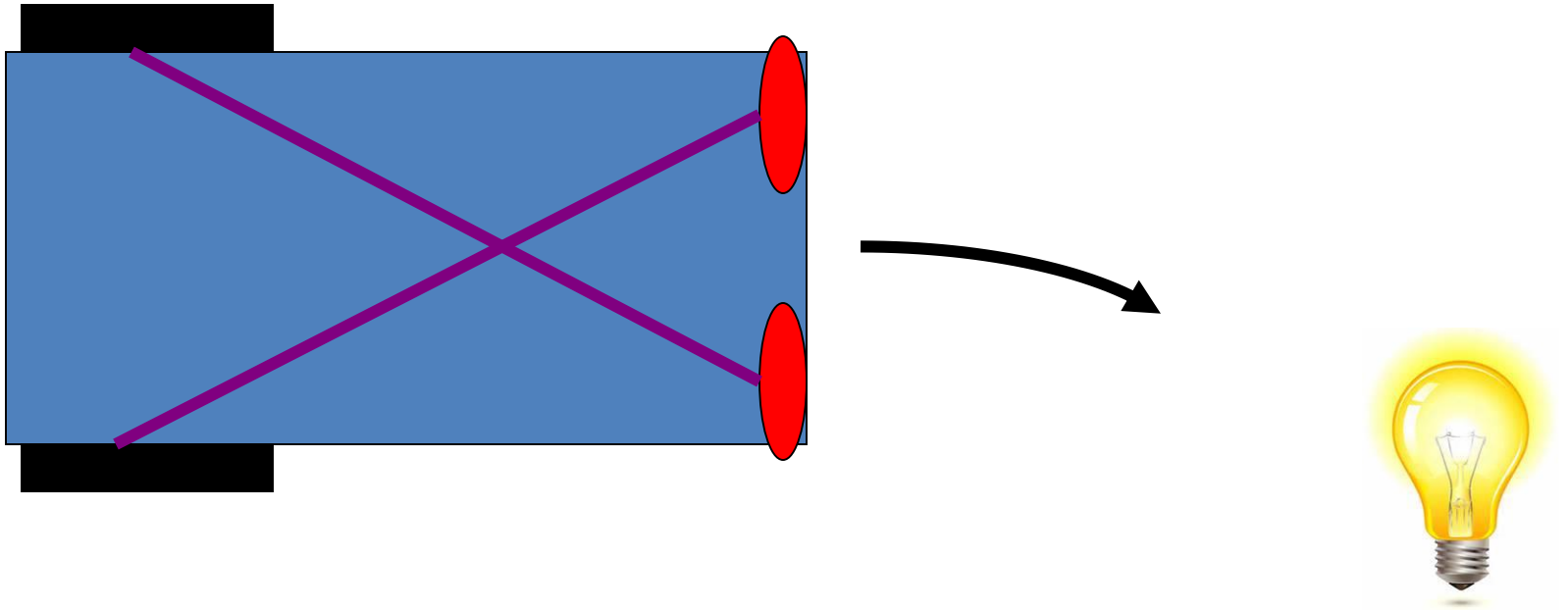
Vehicle 2: Cowardly



More light right →
right wheel turns faster →
turns towards the left, away from the light.

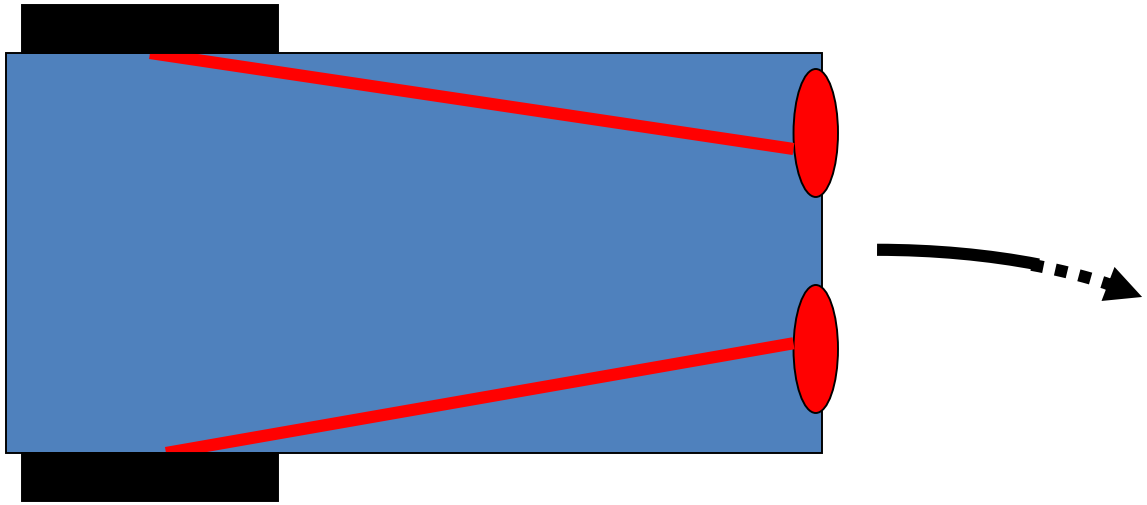
Demonstrates “fight or flight” instinct in animals
Turns away from light if one sensor is activated more than the other
If both are equal, light source is “attacked”

Vehicle 2b: Aggressive



Faces light source and drives toward it

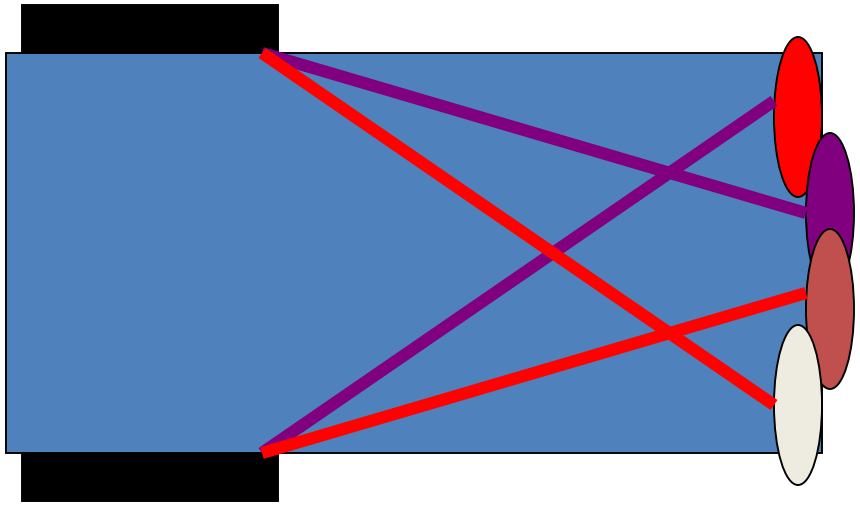
Vehicle 3: Loving



Drives forward
Faces the light source and slows down

Models love/adoration

A little more complicated: Vehicle 3c: Knowing

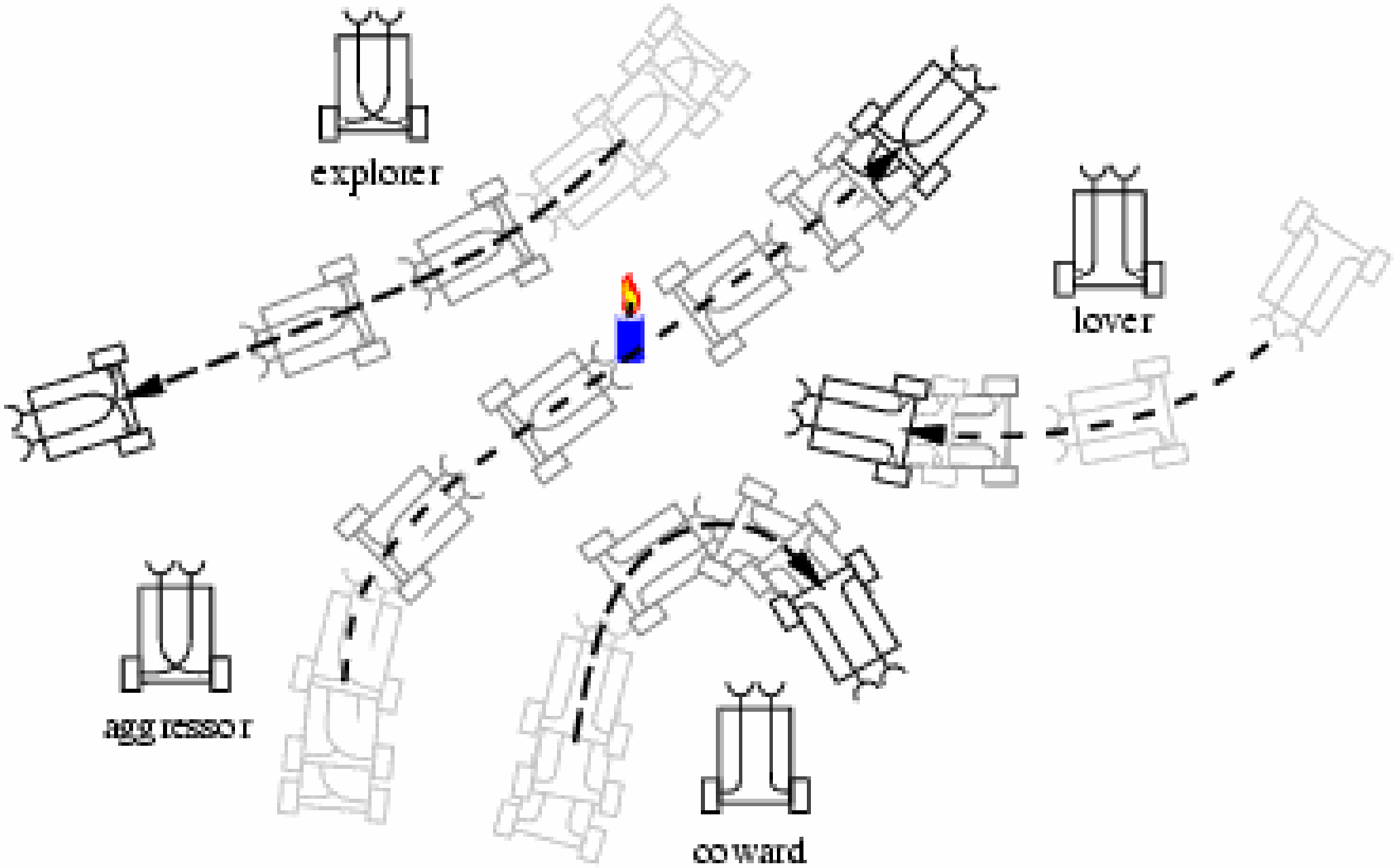


Light Sensor
Temperature Sensor
Organic Material Sensor
Oxygen Sensor

Different sensors:

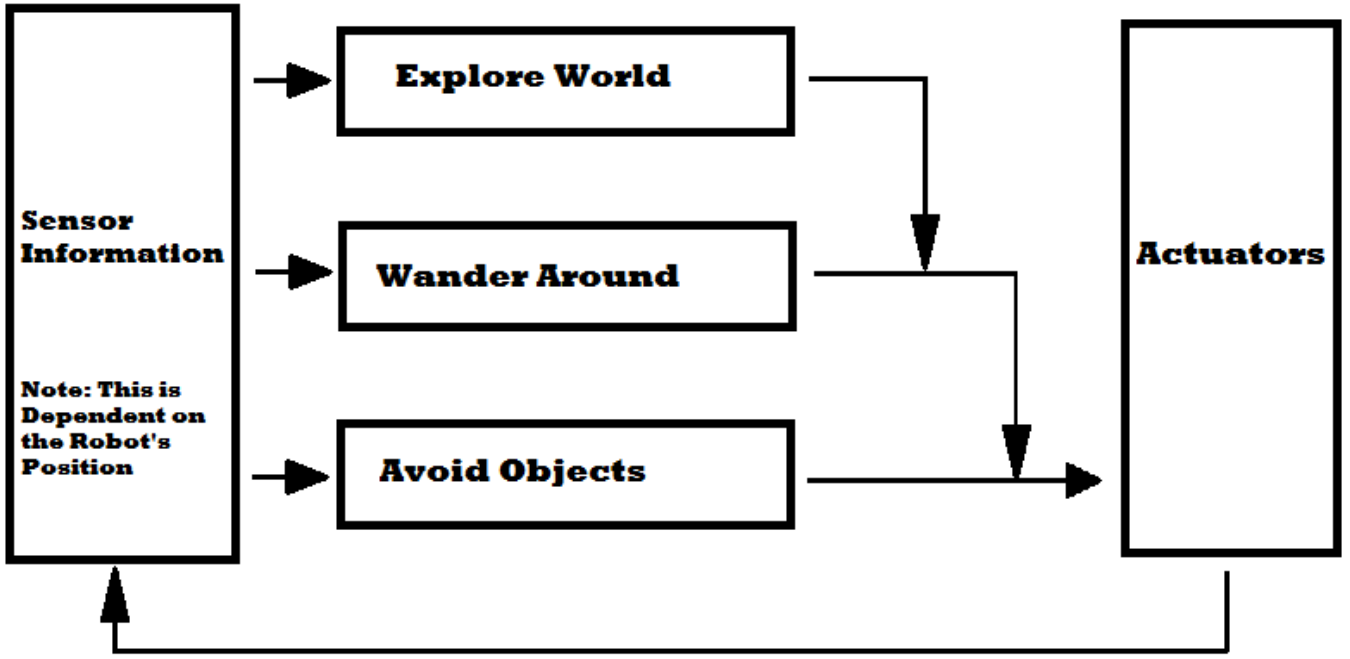
Turns towards light,
doesn’t like heat,
loves organic material,
searches for best Oxygen

Emergent Behavior: Performs the brain function of simplest living beings


explorer
lover
aggressor
coward

Subsumption architecture

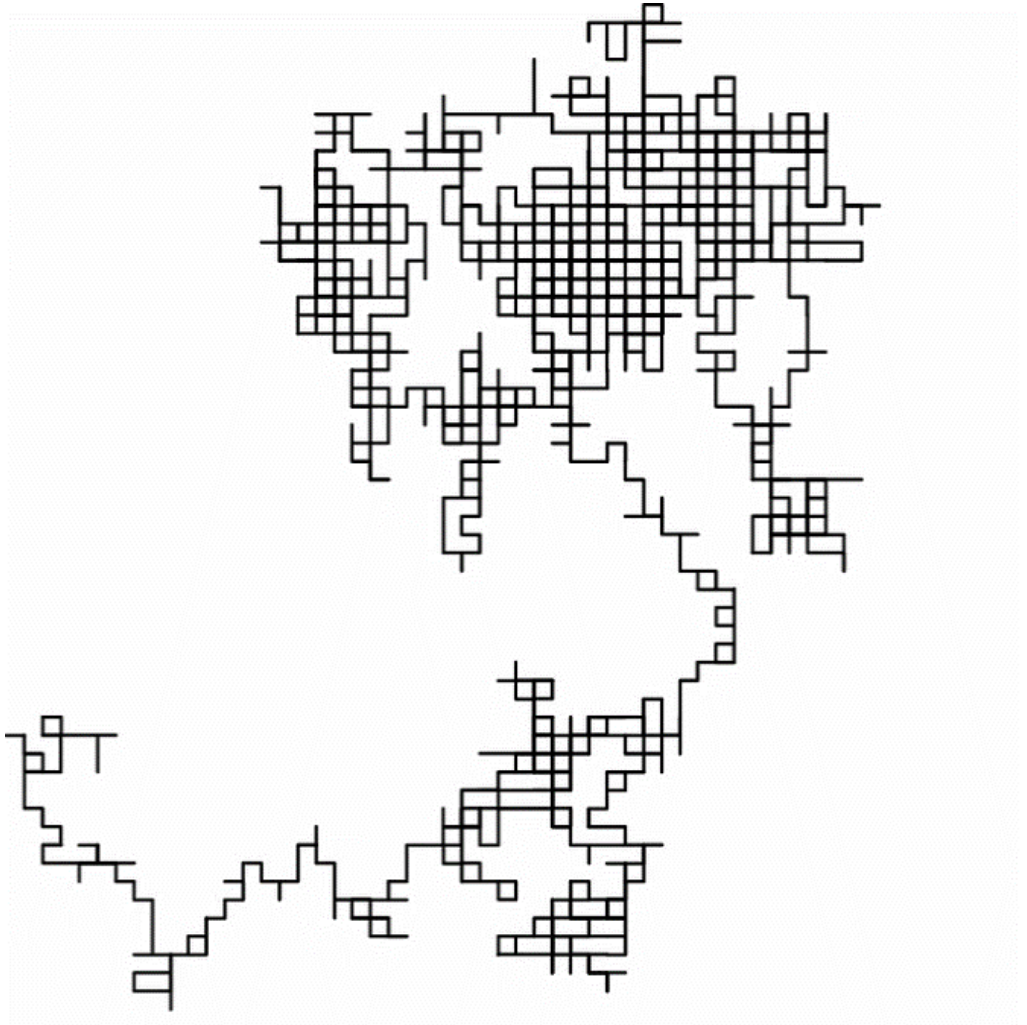
- Rodney Brooks (MIT) 1991:
  - “The world is its own best model” =>
  - “Intelligence without representation”
- Emergent behaviors
- Conclusions:
  - Emergent behaviors quite interesting/ impressive.
  - More complex tasks often need more intelligence.
  - => Behaviors good for low level tasks.



NAVIGATION: SIMPLE SEARCH

Random Walk

- 2D random walk => randomly decide direction
- Very simple – yet effective
- Good chance that everything will be explored – no guarantee
- E.g.: used in vacuum cleaning robots



Random walk in a 2D grid
(Wikipedia)

Homework: Wall following (Right Hand)

- Find the exit of a maze by following the right hand wall
  - “Keep your right hand on the wall”
- Guaranteed to work on:
  - Simply connected maze:
    - all walls connected together or to maze outer wall
    - => no islands
- Simple algorithm:
  - Only state variable: Robot Pose
  - No planning needed; No mapping needed

- Used by fireman if there is no visibility (smoke)

Right Hand Wall Following: Overview

- Input:
  - Map (2D grid)
    - Only use: Find out if cell on the right or in front is occupied or free
  - Start pose
- Output:
  - Start pose + all intermediate poses + end pose
- Algorithm:
  - 1) Find Wall:
    - Go forward until you either:
    - Reach an exit
    - Have a wall to the right side
    - Have a wall in front of you
      - In this case turn left once
  - 2) Follow Wall:
    - Do until you reach an exit:
      - If no wall on your right:
        - Turn right
        - Step forward
      - Else if wall in front of you:
        - Turn left
      - Else:
        - Go forward

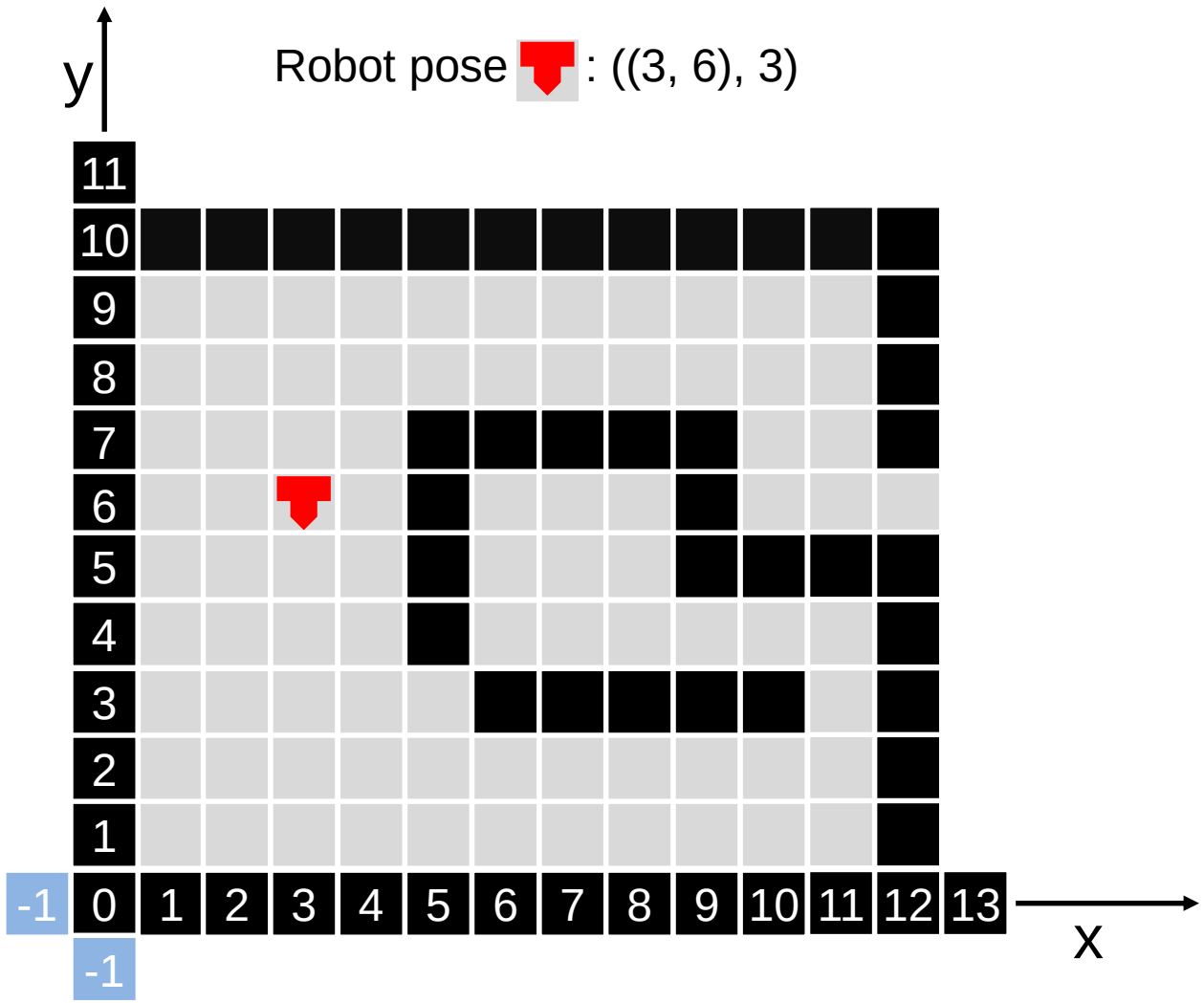
Suggested Helper Functions

- Map class
  - Get(position):
    - Returns 2 if position is outside of map (exit reached)
    - Returns 0 or 1 if position is inside of map – depending on saved value in the map

- Turn Left
- Turn Right
- Step Forward

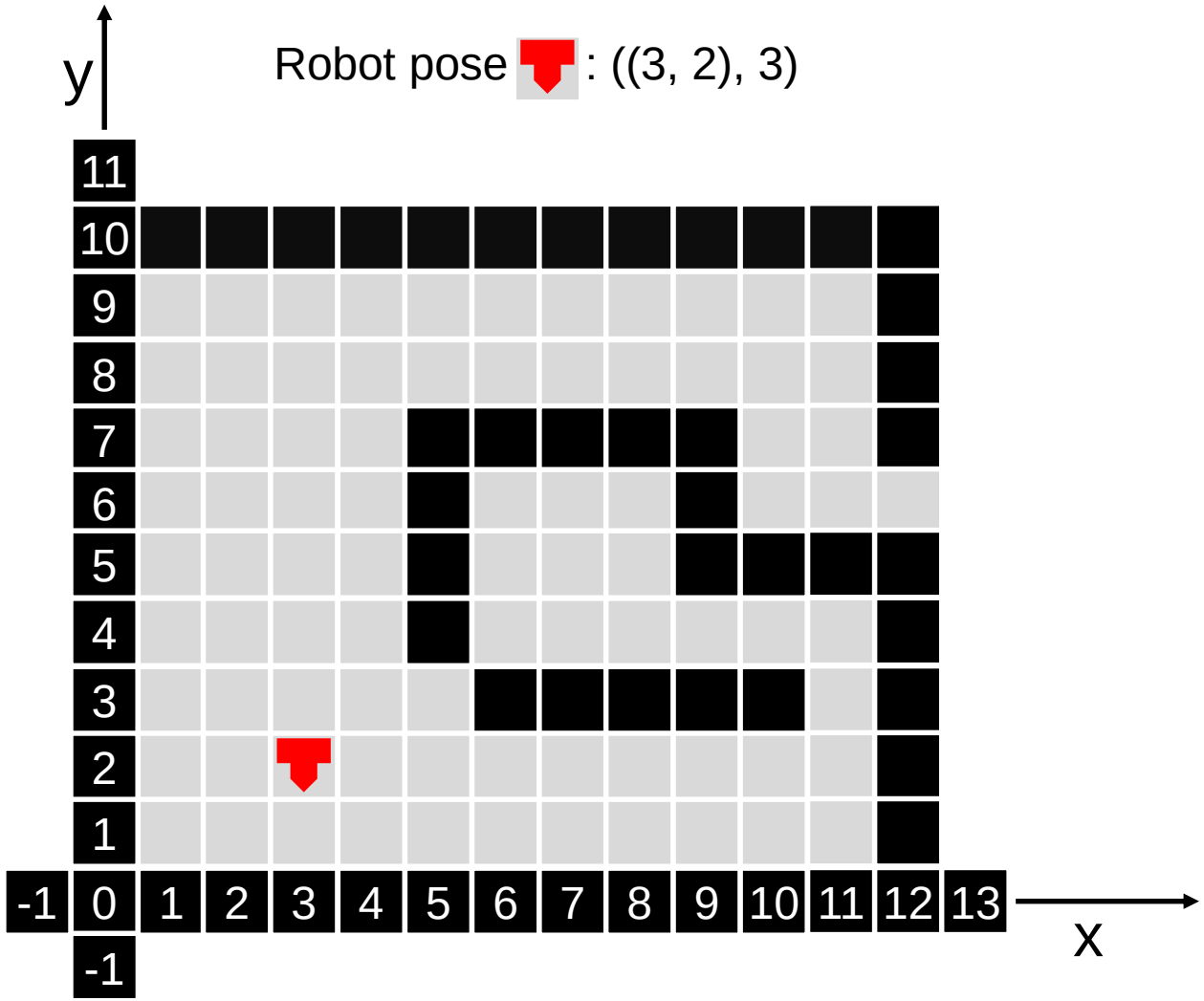
- Is Wall in Front
- Is Wall on Right
  - Those two might make use of the functions above

Conventions



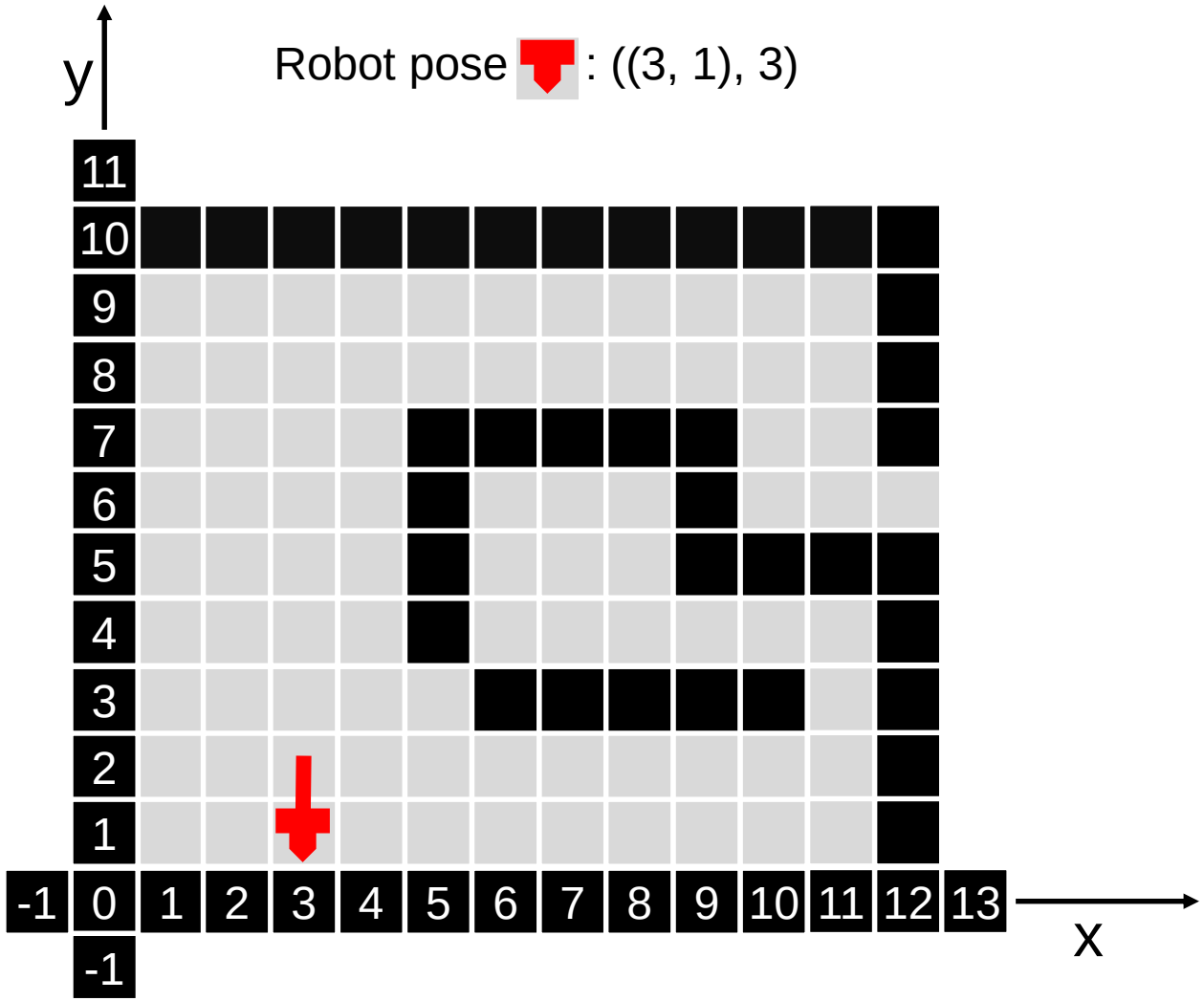
- Cell Values:
  - Black (1): occupied; Gray (0): free
- Coordinate system:
  - Starts lower left corner
- Orientation o :
  - 0: direction positive x →
  - 1: direction positive y ↑
  - 2: direction negative x ←
  - 3: direction negative y ↓
  - Theta (rotation around z-axis): $\theta = \frac{o\,\pi}{2}$
- Pose:
  - Tuple of position (x, y) and orientation o: ((x, y), o)

Find wall



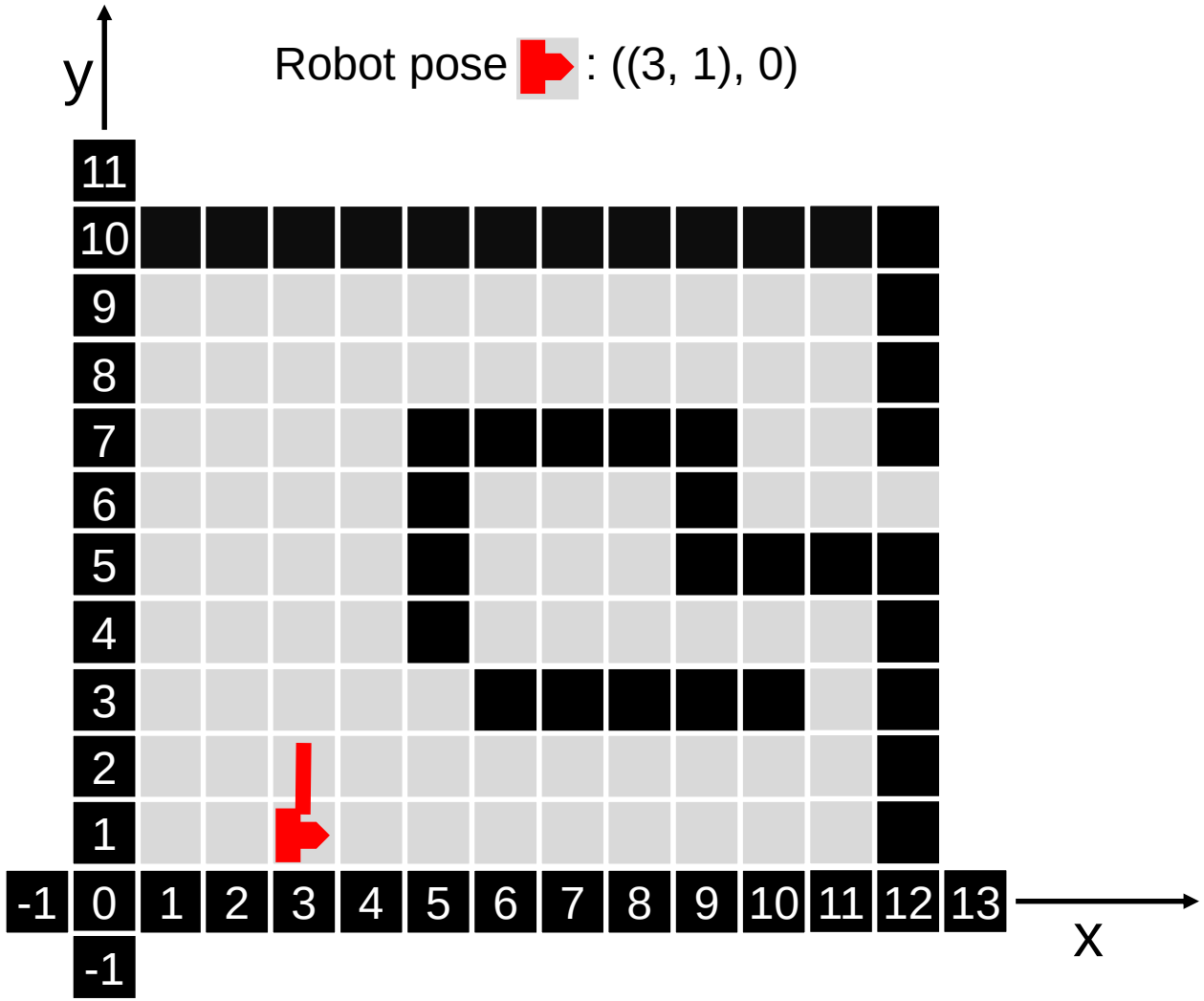
- 1) Find Wall:
  - **Go forward until** you either:
  - Reach an exit
  - Have a wall to the right side
  - Have a wall in front of you
    - In this case turn left once
- **=> go forward to ((3, 1), 3)**

Find wall



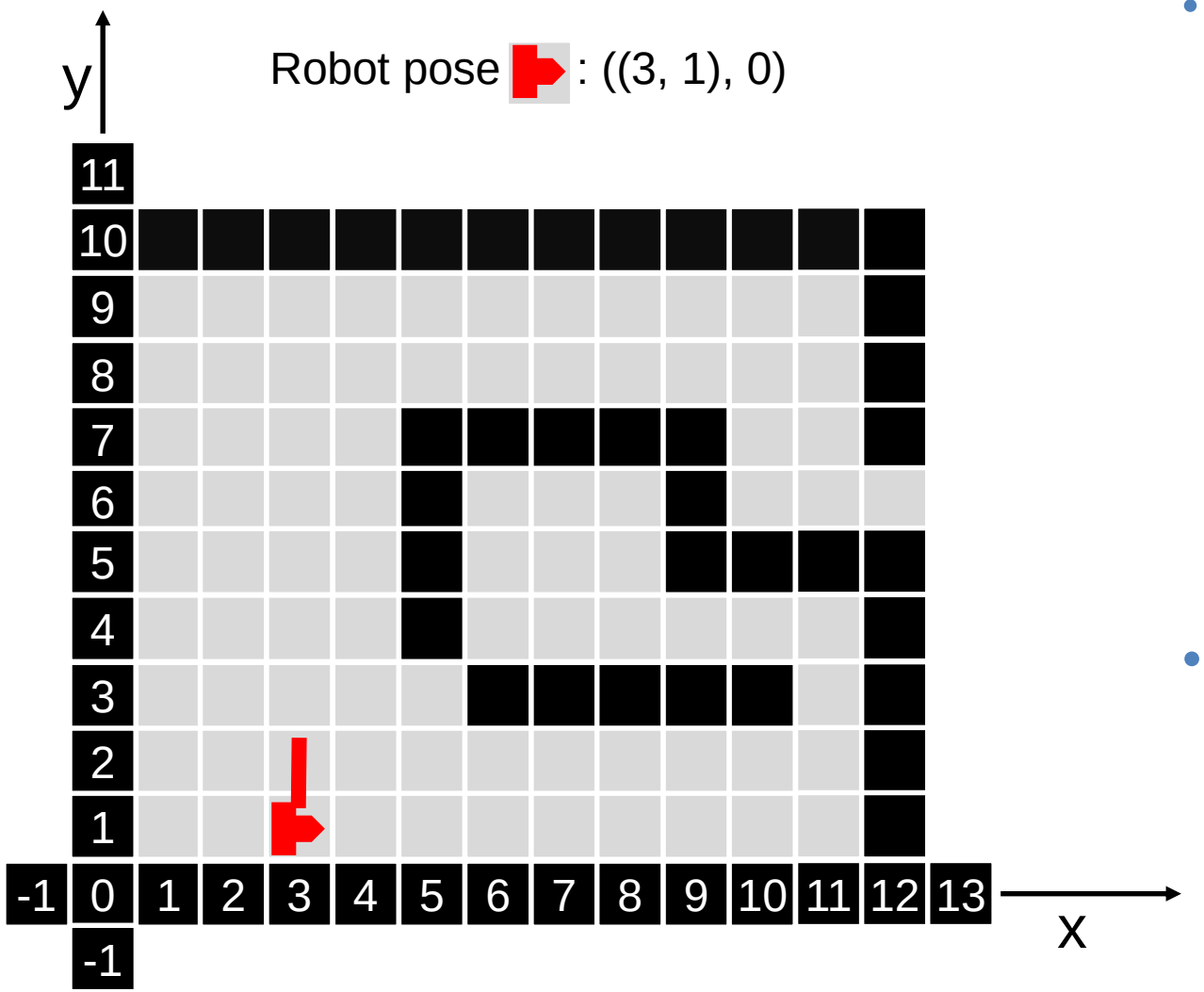
- 1) Find Wall:
  - Go forward until you either:
  - Reach an exit
  - Have a wall to the right side
  - **Have a wall in front of you**
    - In this case turn left once
- **=> turn left to ((3, 1), 0)**

Find wall



- 1) Find Wall:
  - Go forward until you either:
  - Reach an exit
  - Have a wall to the right side
  - **Have a wall in front of you**
    - **In this case turn left once**
- **Done**

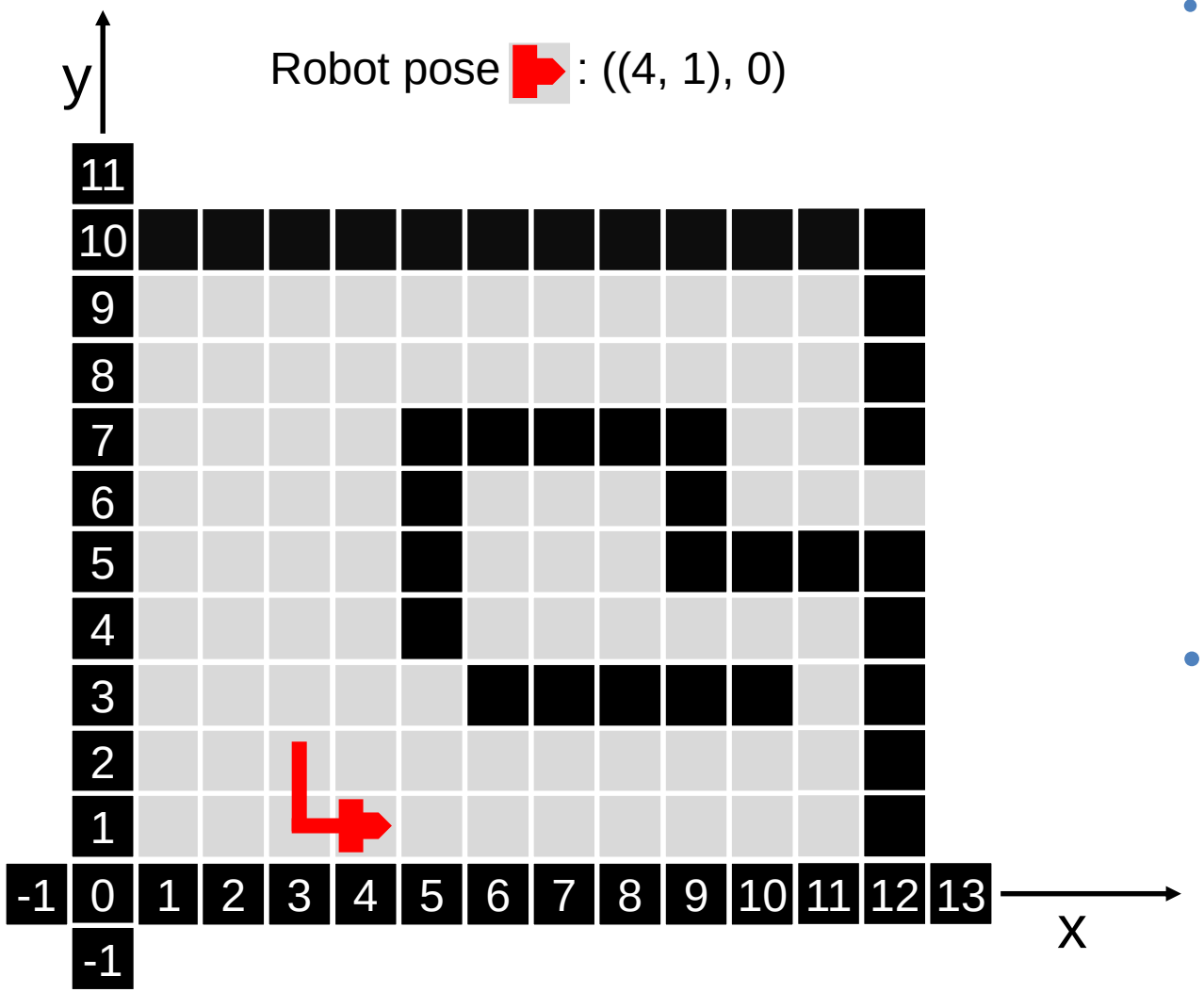
Find wall



- Follow Wall:
  - Do until you reach an exit:
    - If no wall on your right:
      - Turn right
      - Step forward
    - Else if wall in front of you:
      - Turn left
    - **Else:**
      - **Go forward**

- **=> go forward to ((4, 1), 0)**

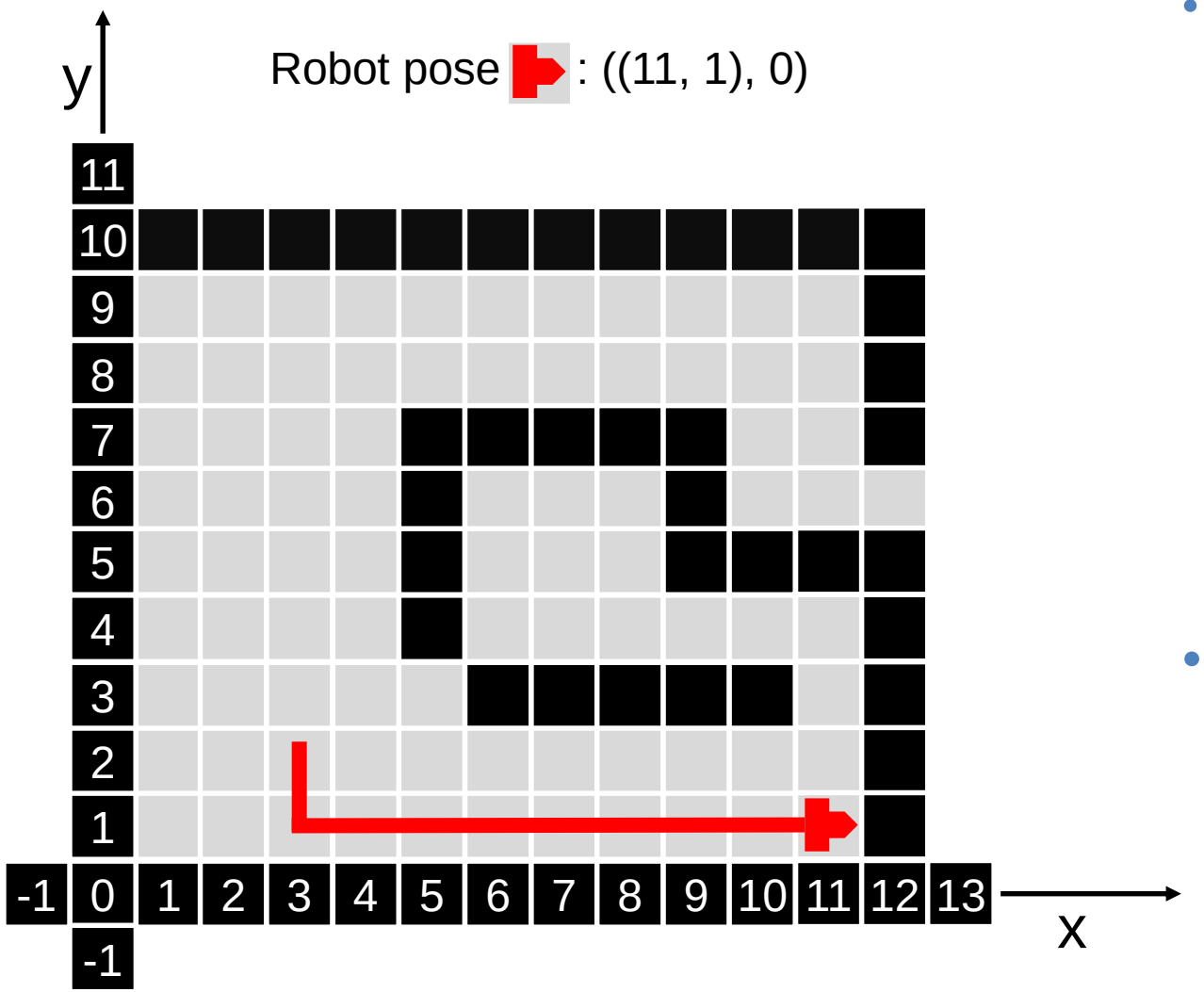
Find wall



- Follow Wall:
  - Do until you reach an exit:
    - If no wall on your right:
      - Turn right
      - Step forward
    - Else if wall in front of you:
      - Turn left
    - **Else:**
      - **Go forward**

- **=> go forward …**

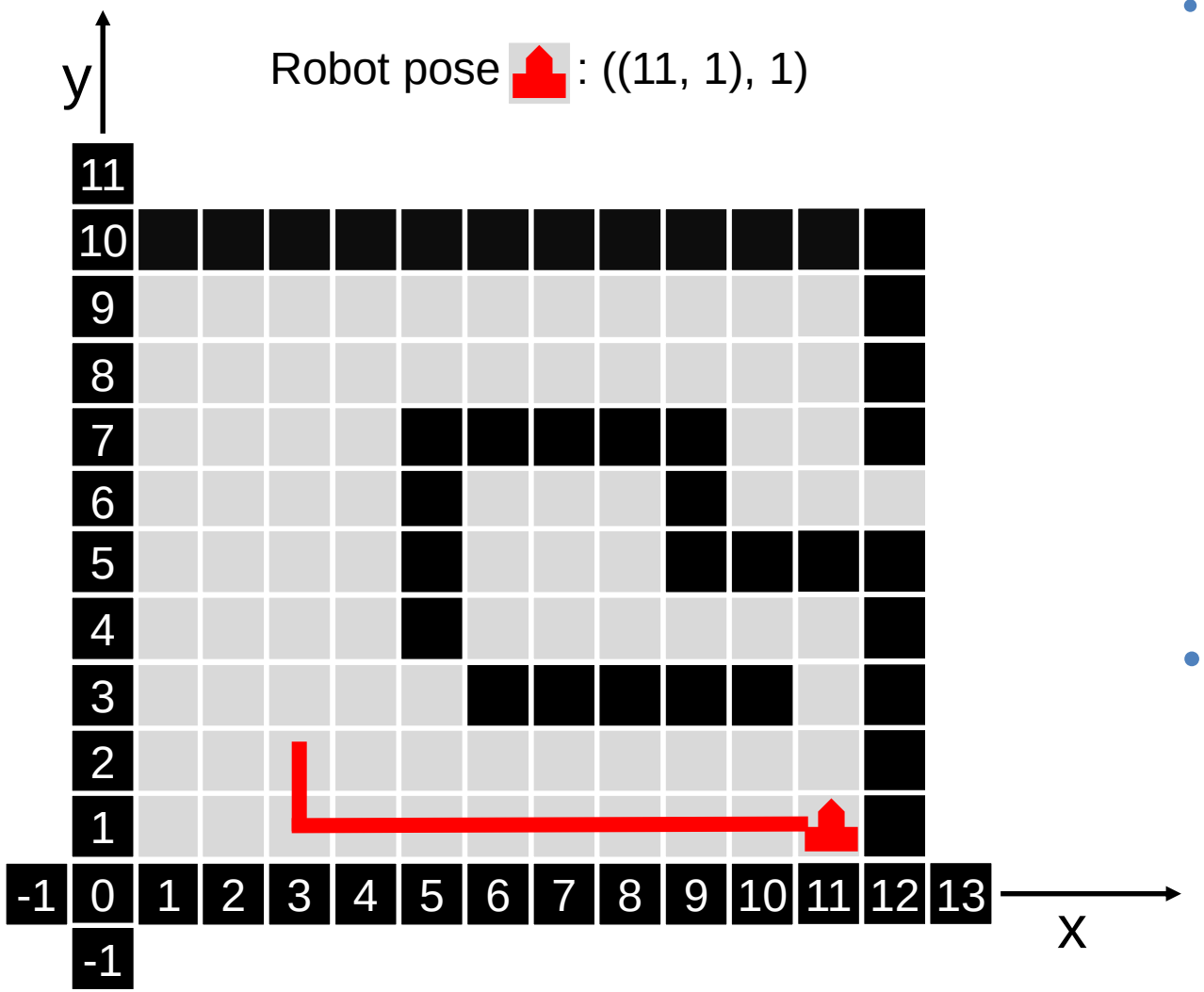
Find wall



- Follow Wall:
  - Do until you reach an exit:
    - If no wall on your right:
      - Turn right
      - Step forward
    - **Else if wall in front of you:**
      - **Turn left**
    - Else:
      - Go forward

- **=> turn left to ((11, 1), 1)**

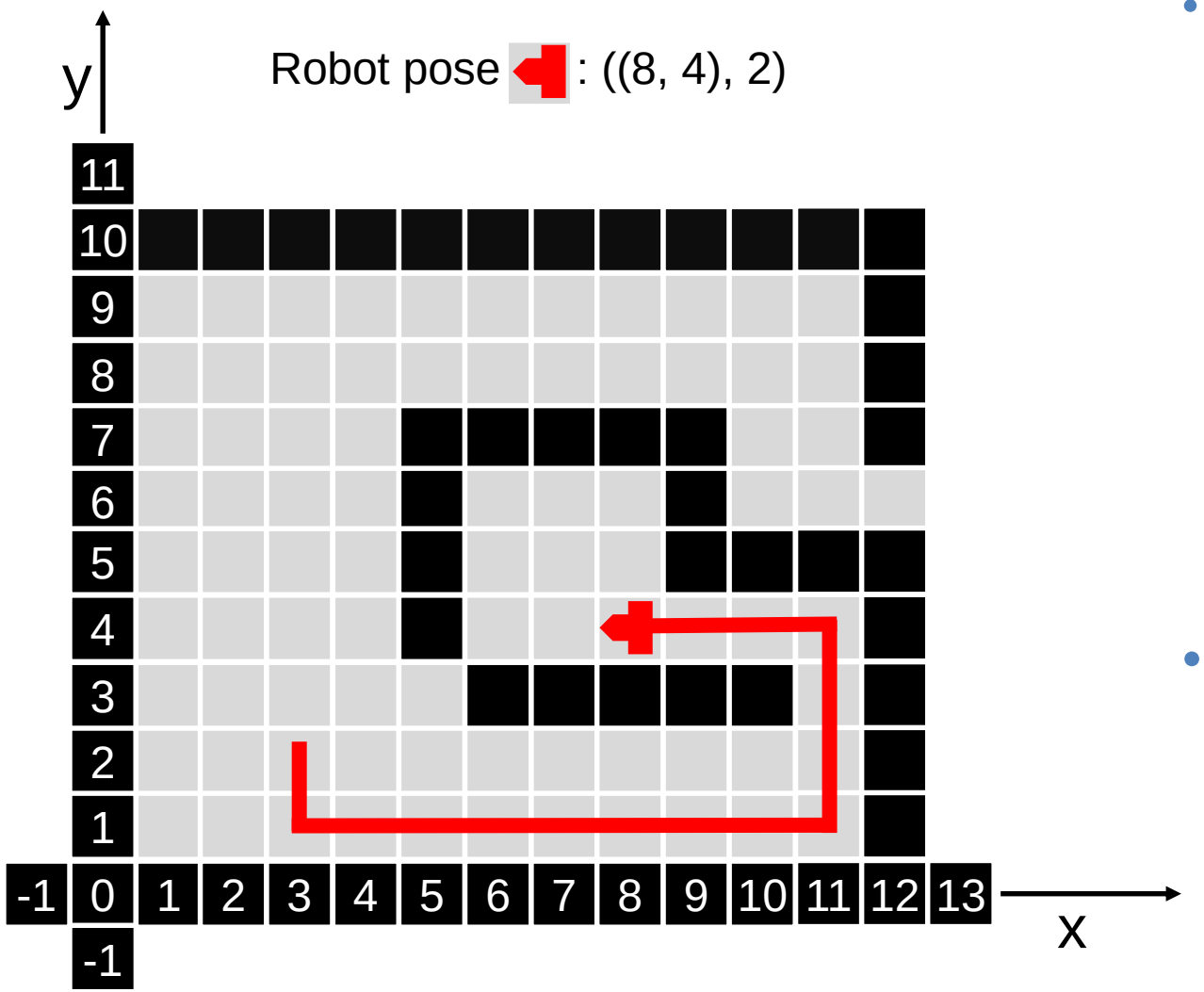
Find wall



- Follow Wall:
  - Do until you reach an exit:
    - If no wall on your right:
      - Turn right
      - Step forward
    - Else if wall in front of you:
      - Turn left
    - Else:
      - **Go forward**

- **=> go forward …**

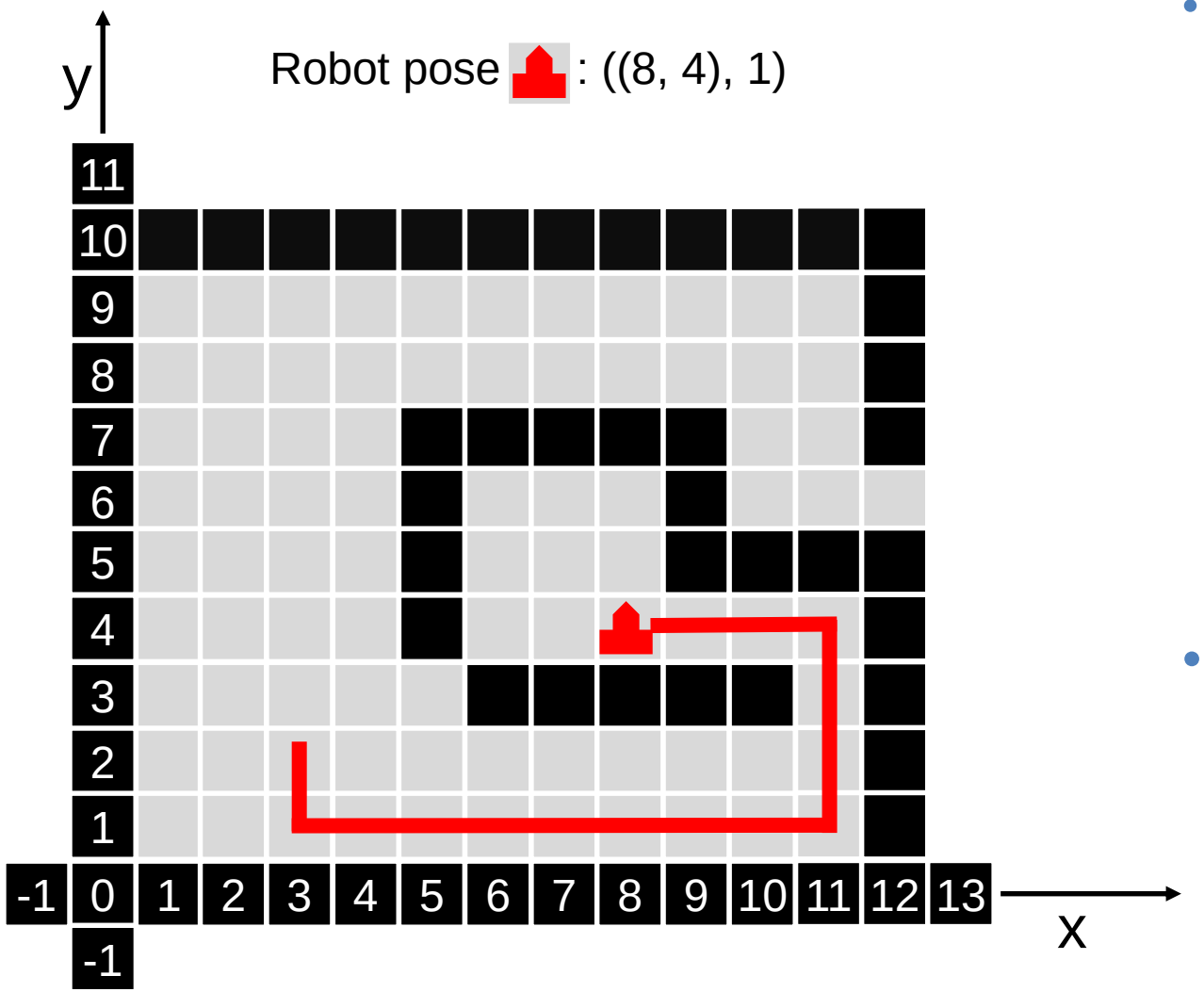
Find wall



- Follow Wall:
  - Do until you reach an exit:
    - **If no wall on your right:**
      - **Turn right**
      - Step forward
    - Else if wall in front of you:
      - Turn left
    - Else:
      - Go forward

- **=> turn right to ((8, 4), 1)**

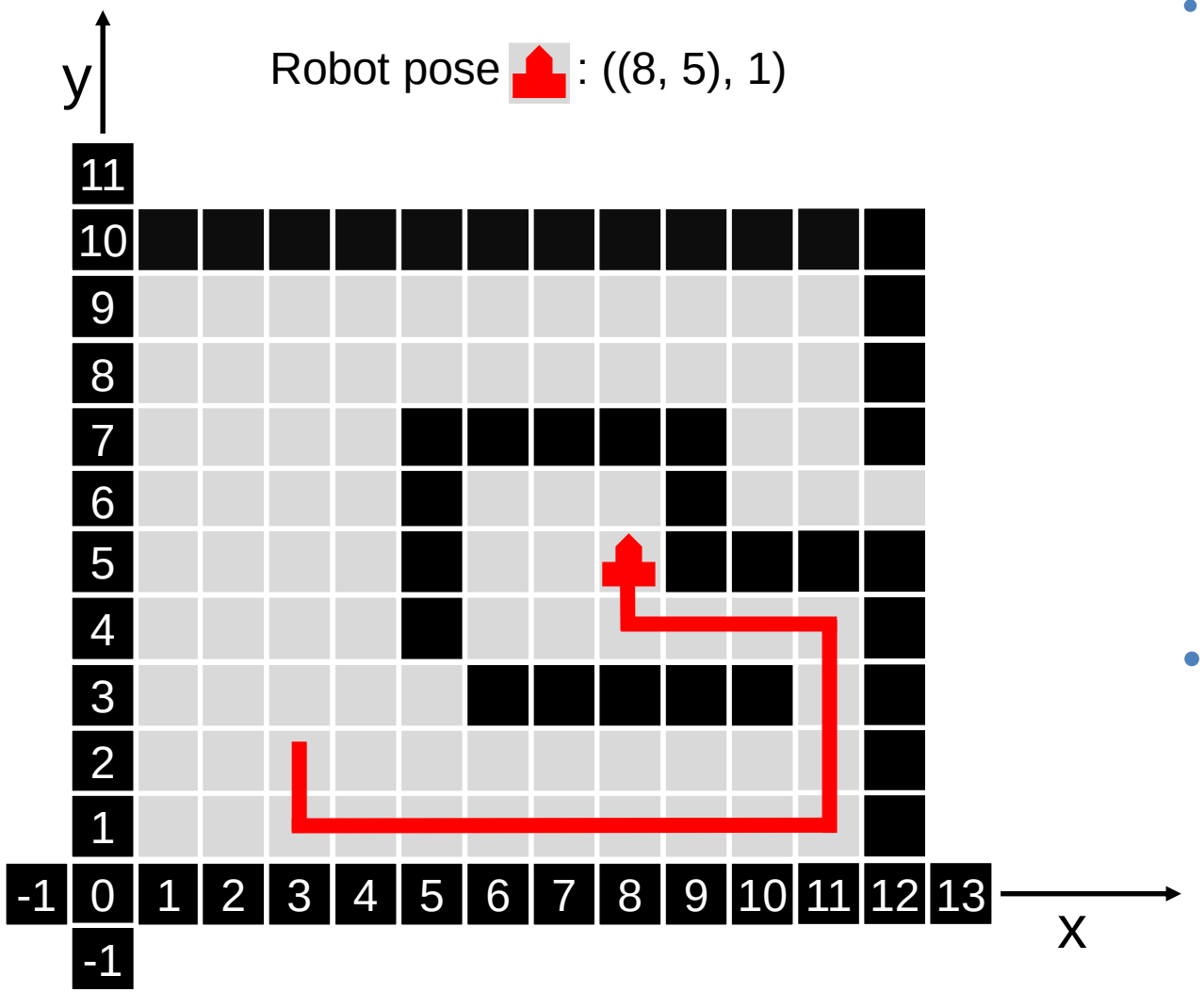
Find wall



- Follow Wall:
  - Do until you reach an exit:
    - **If no wall on your right:**
      - Turn right
      - **Step forward**
    - Else if wall in front of you:
      - Turn left
    - Else:
      - Go forward

- **=> step forward to ((8, 5), 1)**

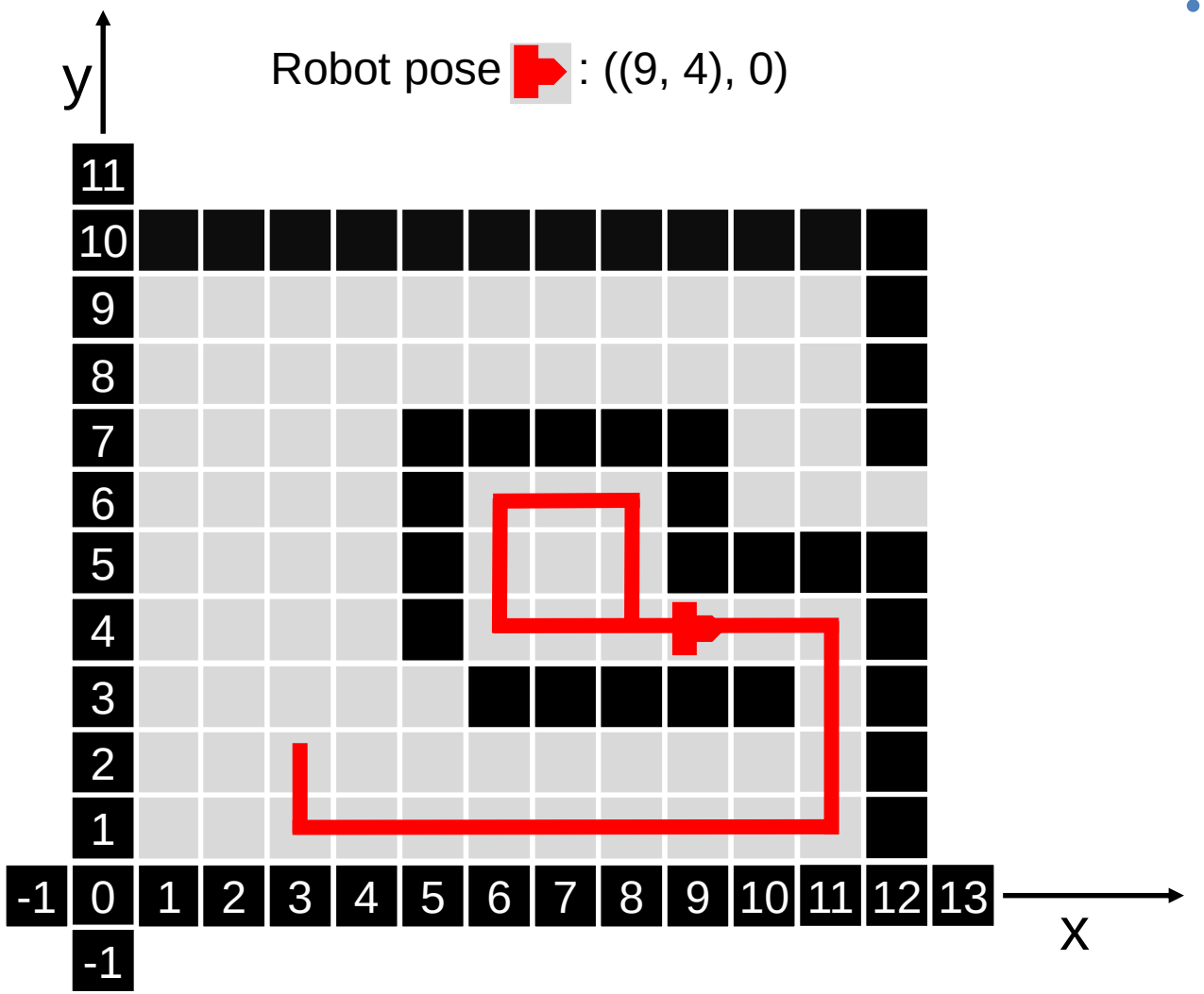
Find wall



- Follow Wall:
  - **Do until you reach an exit:**
    - If no wall on your right:
      - Turn right
      - Step forward
    - Else if wall in front of you:
      - Turn left
    - **Else:**
      - **Go forward**

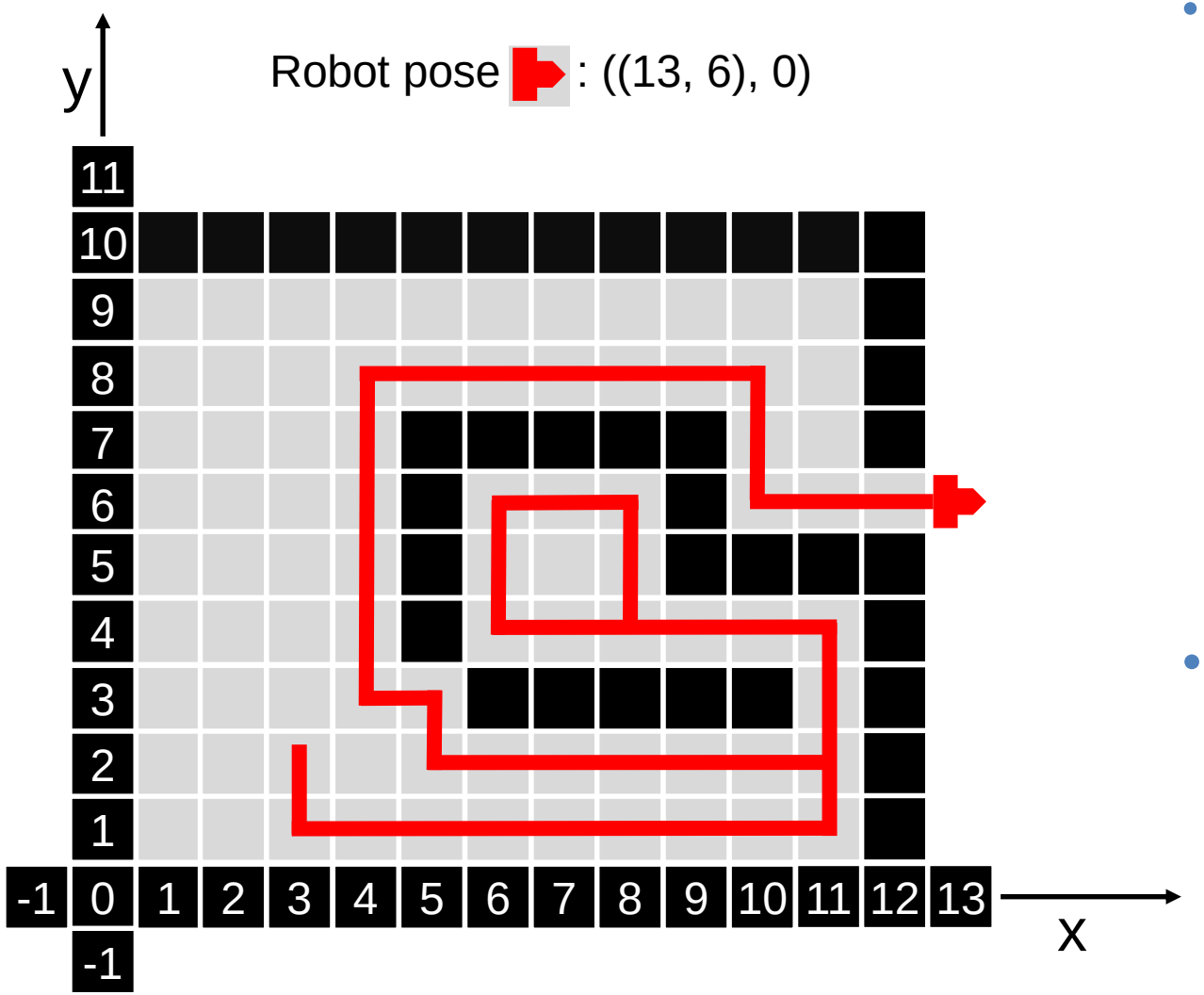
- **=> step forward to ((8, 6), 1)**

Find wall



- Follow Wall:
  - Same position as before, but different orientation => different orientation!
  - State variable is pose (not position) => different state

Find wall



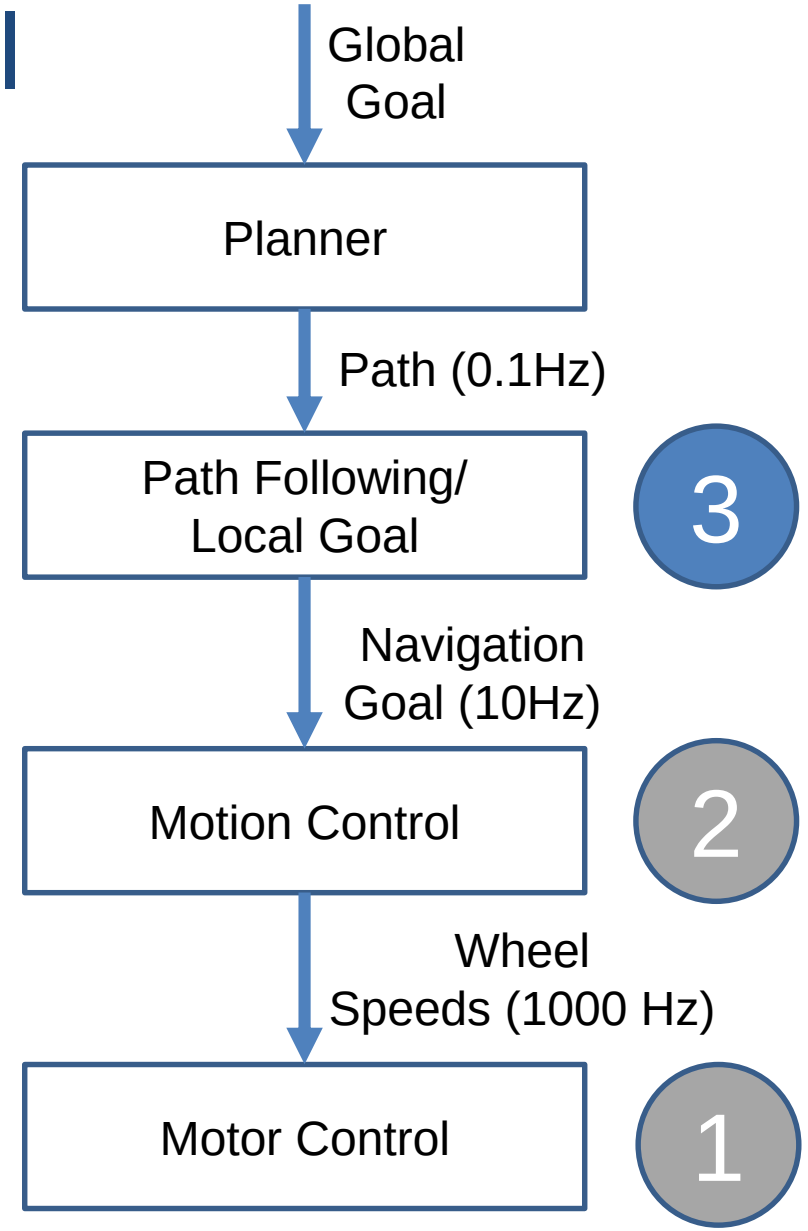
- Follow Wall:
  - Do until you **reach an exit**:
    - If no wall on your right:
      - Turn right
      - Step forward
    - Else if wall in front of you:
      - Turn left
    - Else:
      - Go forward
- **Exit reached: Done!**

PATH FOLLOWING

Obstacle Avoidance

Navigation, Motion & Motor Control

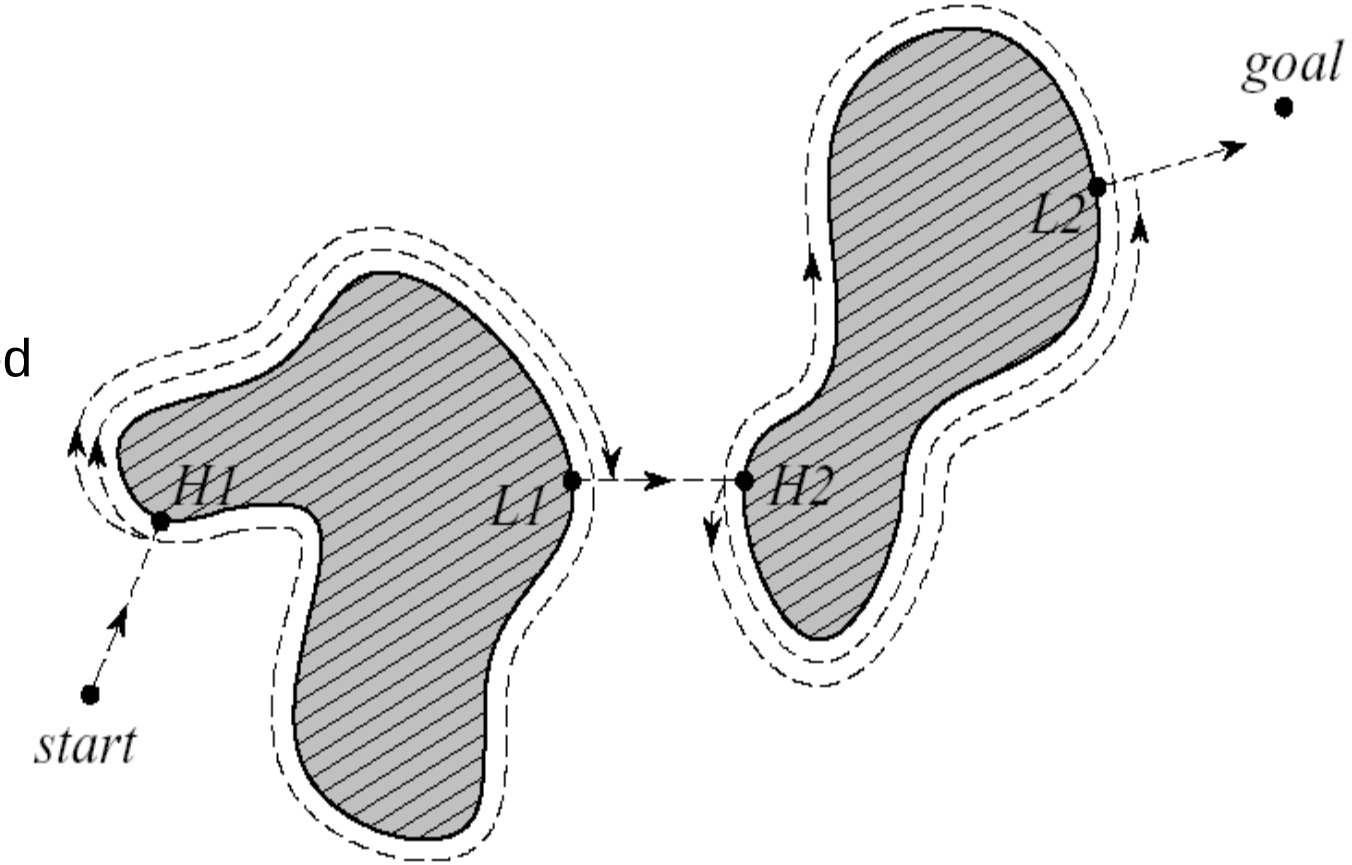
- Navigation/ Motion Control:
  - Where to drive to **next** in order to reach goal
  - Output: motion vector (direction) and speed
  - For example:
    - follow path (Big Model)
    - go to unexplored area (Big Model)
    - drive forward (Small Model)
    - be attracted to goal area (Small Model)
- Motion Control:
  - How use propulsion to achieve motion vector
- Motor Control:
  - How much power to achieve propulsion (wheel speed)



Obstacle Avoidance: Bug1

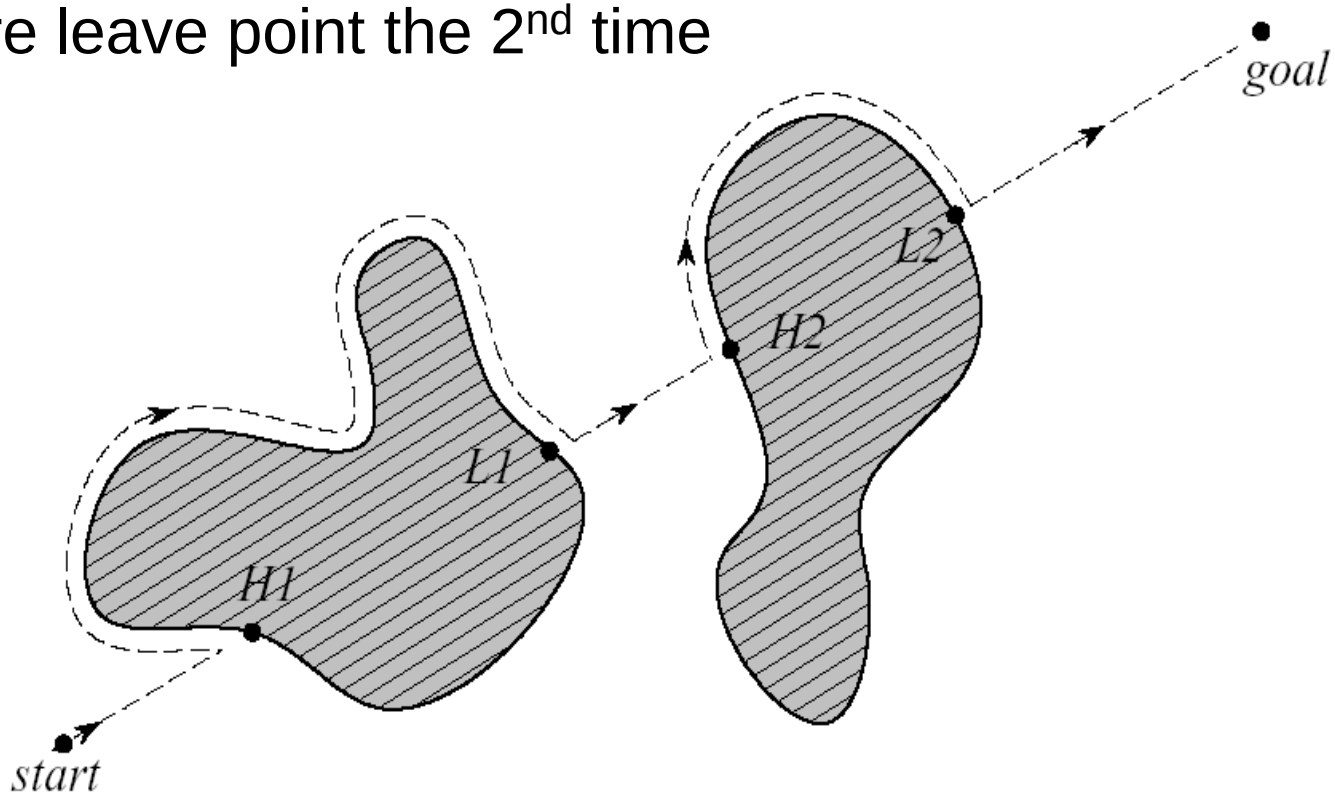
- Following along the obstacle to avoid it
- Each encountered obstacle is once fully circled before it is left at the point closest to the goal

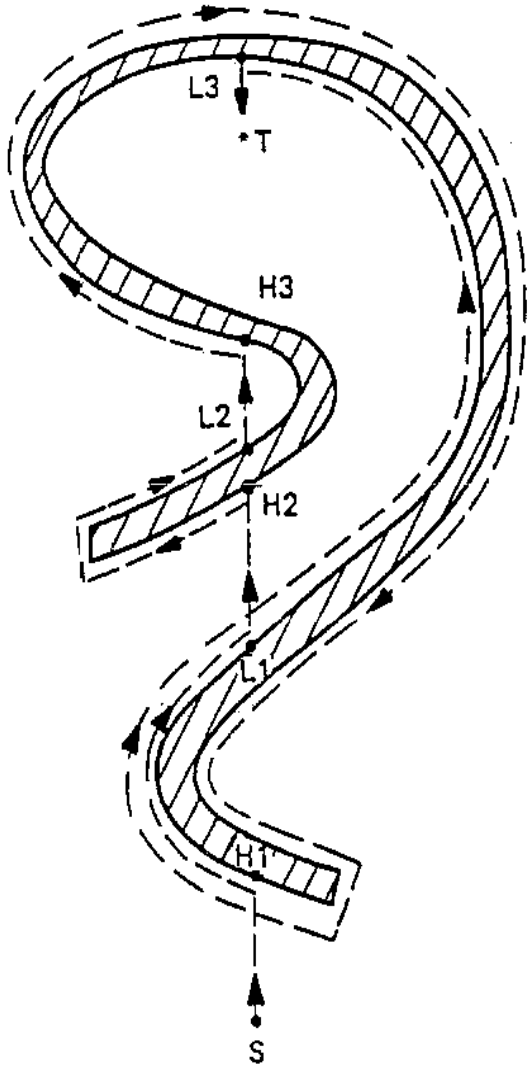
- Advantages
  - No global map required
  - Completeness guaranteed
- Disadvantages
  - Solutions are often highly suboptimal



Obstacle Avoidance: Bug2

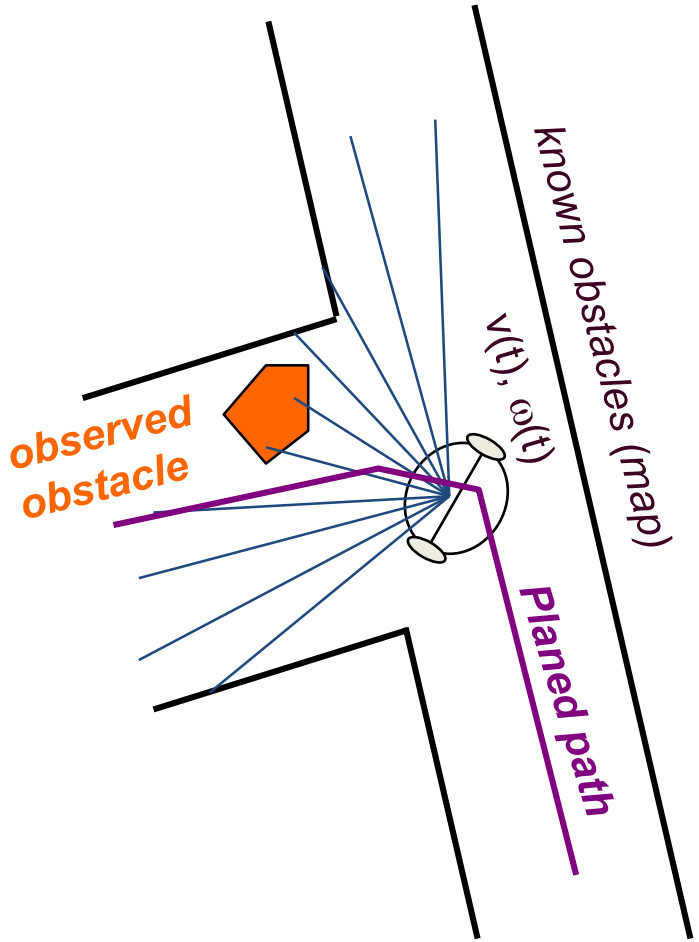
- Right (or left) hand wall following
- Leave obstacle if direct connection between start and goal is crossed
- Ignore leave point the 2nd time





Obstacle Avoidance (Local Path Planning)

- Goal: avoid collisions with obstacles
- Usually based on local map
- Often implemented as independent task
- However, efficient obstacle avoidance should be optimal with respect to
  - the overall goal
  - the actual speed and kinematics of the robot
  - the on boards sensors
  - the actual and future risk of collision



Obstacle Avoidance: Vector Field Histogram (VFH)

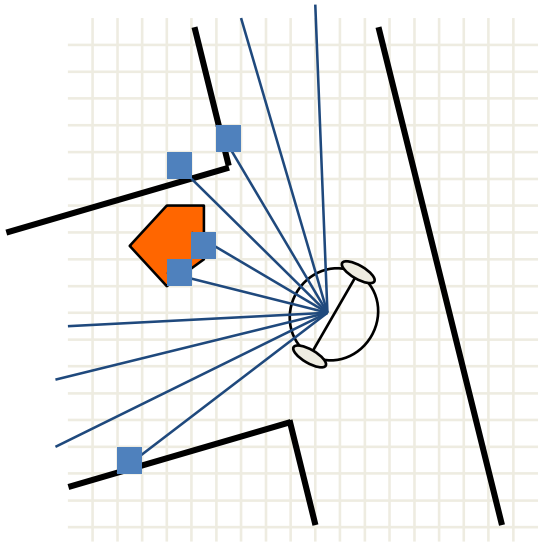
Borenstein et al.

- Environment represented in a grid (2 DOF)
  - cell values: probability of obstacle
- Reduction in to 1 DOF histogram
  - Steering direction algorithm:
    - Find all openings for the robot to pass
    - Lowest cost function G

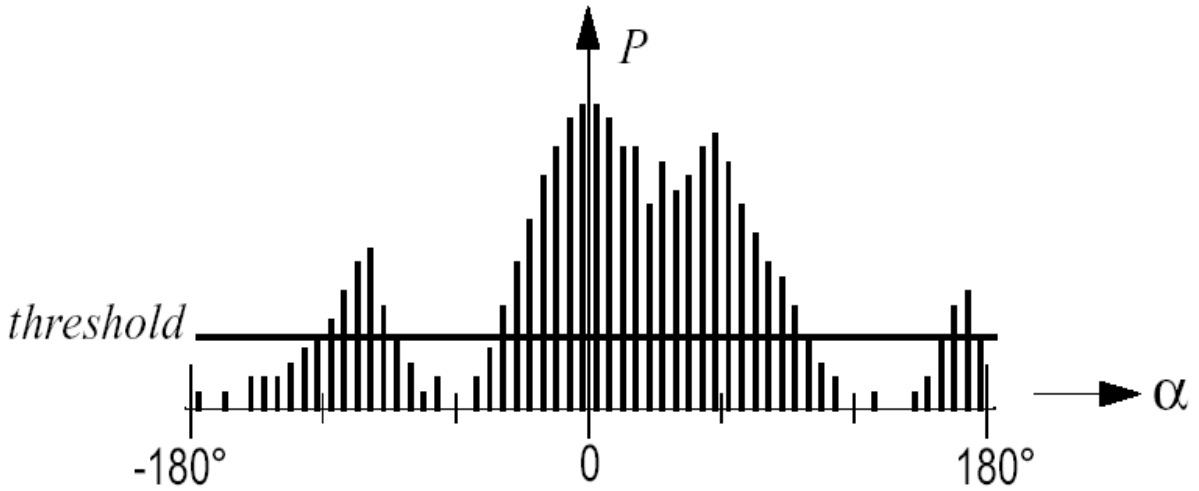
target_direction = alignment of the robot path with the goal

wheel_orientation = difference between the new direction and the currrent wheel orientation

previous_direction = difference between the previously selected direction and the new direction

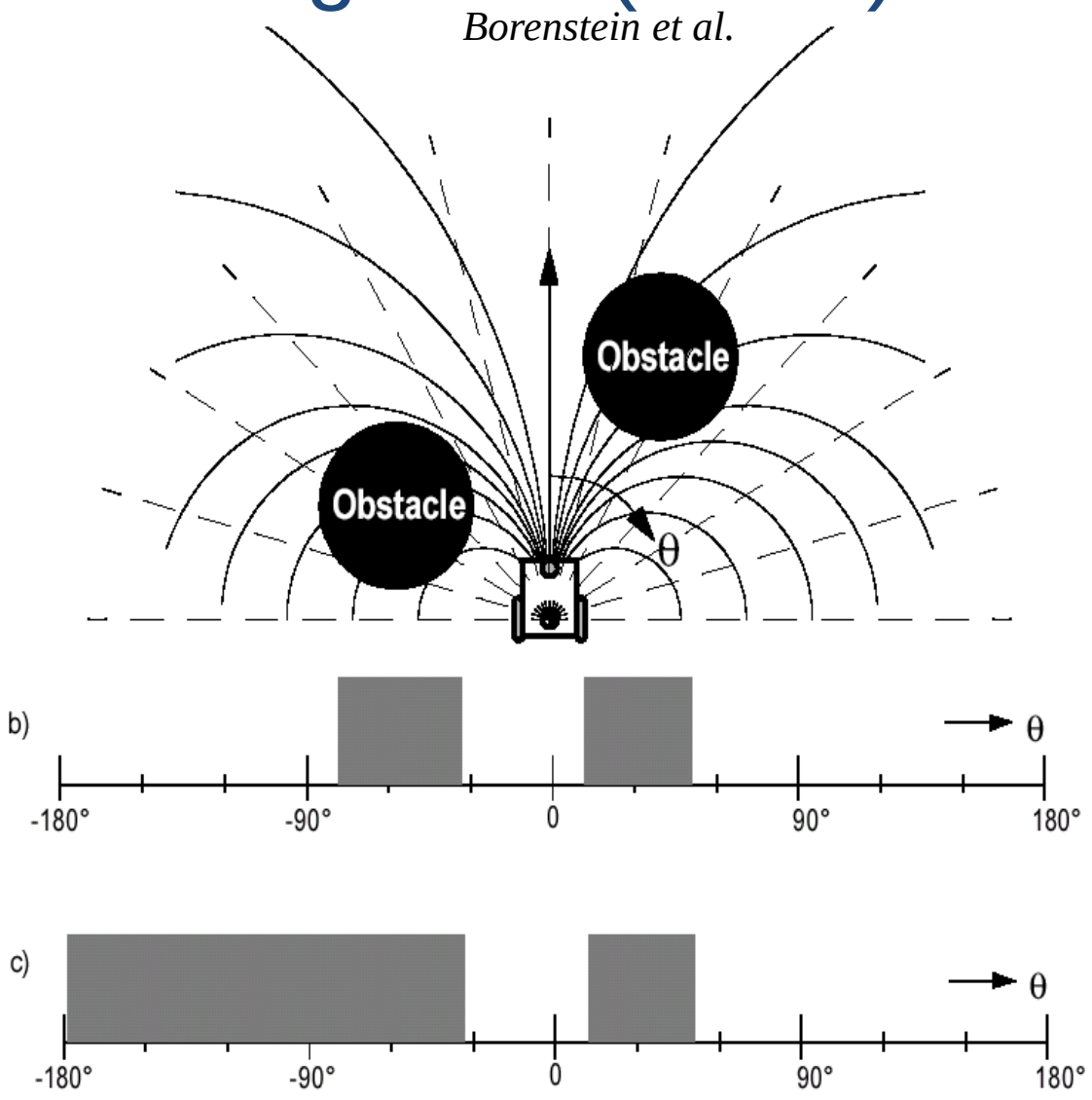


$$G = a \cdot \text{target\_direction} + b \cdot \text{wheel\_orientation} + c \cdot \text{previous\_direction}$$



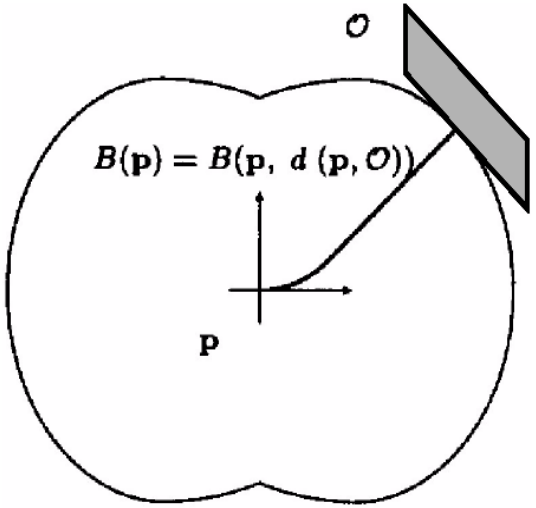
Obstacle Avoidance: Vector Field Histogram + (VFH+)

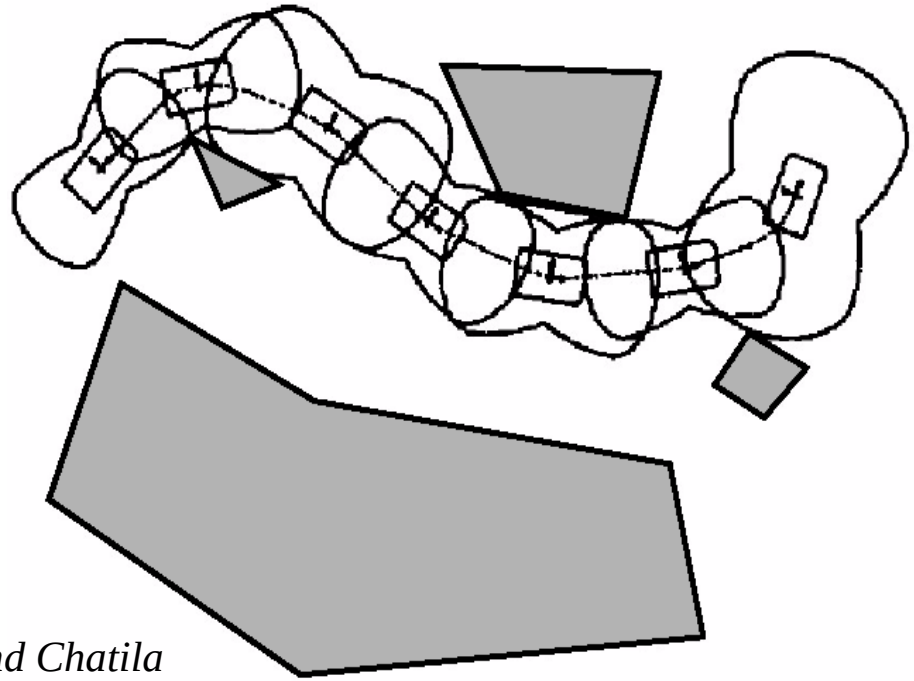
- Accounts for the moving trajectories (dynamics)
  - Robot moving on arcs or straight lines
  - Obstacles blocking a given direction also blocks all the trajectories (arcs) going through this direction
  - Obstacles are enlarged so that all kinematically blocked trajectories are properly taken into account



Obstacle Avoidance: The Bubble Band Concept

- Bubble = Maximum free space which can be reached without any risk of collision
  - generated using the distance to the object and a simplified model of the robot
  - bubbles are used to form a band of bubbles which connects the start point with the goal point

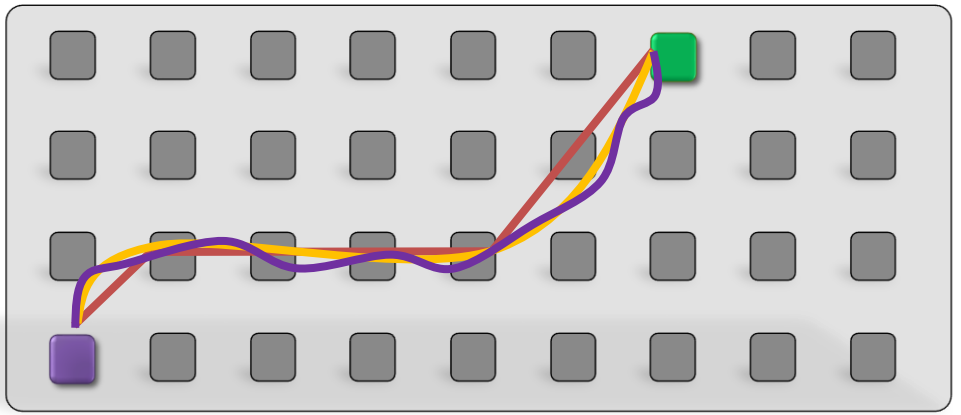




Khatib and Chatila

Path Following

- A path is generated by planning algorithm (next lecture)
- Goal: Drive along that path
  - Path smoothing or
  - Local planner
- Control onto the local path
  - For example:
  - Select goal point one meter ahead of path



Material from Martin Rufli

Dynamic Environments (2/2)

- Dynamic Environments: often Dealt with via Frequent Replanning