Introduction to Cognitive Robotics

Module 3: Mobile Robots

Lecture 7: the go-to-position and go-to-pose problems; MIMO controller

David Vernon
Carnegie Mellon University Africa

www.vernon.eu

Go-to-Position as a Control Problem

Solution 2: MIMO Controller

2D solution to a 2D problem ⇒ MIMO strategy

Multiple Input, Multiple Output

Compute forward and rotation velocities

If required, convert them to left and right wheel angular velocities

$$v = K_p^{pos}\, e_{pos}$$

$$\omega = K_p^{h}\, e_h$$

(K_p^{pos} and K_p^{h} gains have different values than Solution 1)

- Translation velocity v depends on how far the robot is from the goal
- Rotation velocity ω depends on how much it is heading away from the goal
- We limit v by an upper bound v_{max}

Go-to-Position as a Control Problem

Solution 2: MIMO Controller

If required, convert them to left and right wheel angular velocities ...

to do this we need the inverse kinematics

- We need to convert (v, ω) to $(\dot{\phi}_1, \dot{\phi}_2)$

- That is, we need to find f_L, f_R such that:

$$\dot{\phi}_1 = f_R(v, \omega)$$
$$\dot{\phi}_2 = f_L(v, \omega)$$

Recall: Forward Kinematics

The motion of the robot in the local robot frame of reference R due to the rotation of the wheels is given by:

Also see Siegwart et al. (2004, 2011)

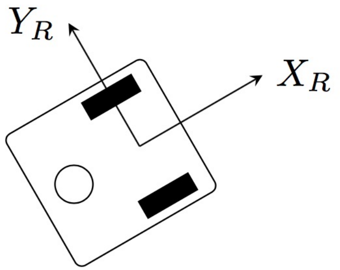
Instantaneous velocity v in the X_R direction

$$\dot{\xi}_R = \begin{bmatrix} \frac{r\dot{\phi}_1}{2} + \frac{r\dot{\phi}_2}{2} \\ 0 \\ \frac{r\dot{\phi}_1}{2l} + \frac{-r\dot{\phi}_2}{2l} \end{bmatrix}$$

Instantaneous velocity in the Y_R direction

Angular velocity ω about the vertical

Rate of change of pose
in the robot frame of reference



Inverse Kinematics

That is

$$\begin{aligned} v &= \frac{r\dot{\phi}_1}{2} + \frac{r\dot{\phi}_2}{2} \\ \omega &= \frac{r\dot{\phi}_1}{2l} + \frac{-r\dot{\phi}_2}{2l} \end{aligned}$$

Thus

$$\begin{aligned} \dot{\phi}_1 &= \frac{v}{r} + \frac{\omega l}{r} \\ \dot{\phi}_2 &= \frac{v}{r} - \frac{\omega l}{r} \end{aligned}$$

Forward kinematics vs. inverse kinematics

– Joint space
  - configurations of the movable joints of the robot
  - here, change in configuration given by ($\dot{\phi}_1$, $\dot{\phi}_2$)

– Work space
  - configurations of the robot in the environment
  - here, change in configuration given by (v, ω)

– Direct (forward) kinematics
  - transformation from joint space to work space

– Inverse kinematics
  - transformation from work space to joint space

Equivalently,
multiplying by time elapsed
and the radius of the wheels,
the distance travelled by the
right and left wheels: d_R and d_L

Equivalently, multiplying by time elapsed,
the distance travelled by the robot
and the robot's rotation: d and $\Delta\theta$

Go-to-Position as a Control Problem

Solution 2: MIMO Controller

Algorithm goto2(x_g, y_g) using proportional control

Global variables: the current robot position and orientation (x_r, y_r, θ_r)

Arguments: the goal position of the robot (x_g, y_g)

the proportional gains for controlling position K_p^{pos} and orientation K_p^h

the tolerances on position error Δ_{pos}

Remember: you will need to use different gain values for this controller

Go-to-Position as a Control Problem

Solution 2: MIMO Controller

Do

Compute the current position of the robot (x_r, y_r)

Compute the distances from the robot position to the target position (d_x, dy)

$$d_x = x_g - x_r$$
$$d_y = y_g - y_r$$

Compute the position and heading errors(e_{pos}, e_h)

$$e_{pos} = \text{sqrt}\,(d_x^2 + d_y^2)$$
$$e_h = \text{atan2}\,(d_y, d_x) - \theta_r$$

The difference between the desired heading and the current robot orientation

Compute the forward velocity and angular velocity (v, ω)

$$v = K_p^{pos}\, e_{pos}$$
$$\omega = K_p^h\, e_h$$

If required, convert (v, ω) to $(\dot{\phi}_1, \dot{\phi}_2)$ using inverse kinematics

Send velocities (v, ω) or $(\dot{\phi}_1, \dot{\phi}_2)$ to the robot

Pause some time

while $|e_{pos}| > \Delta_{pos}$

Send velocities $(0, 0)$ to the robot

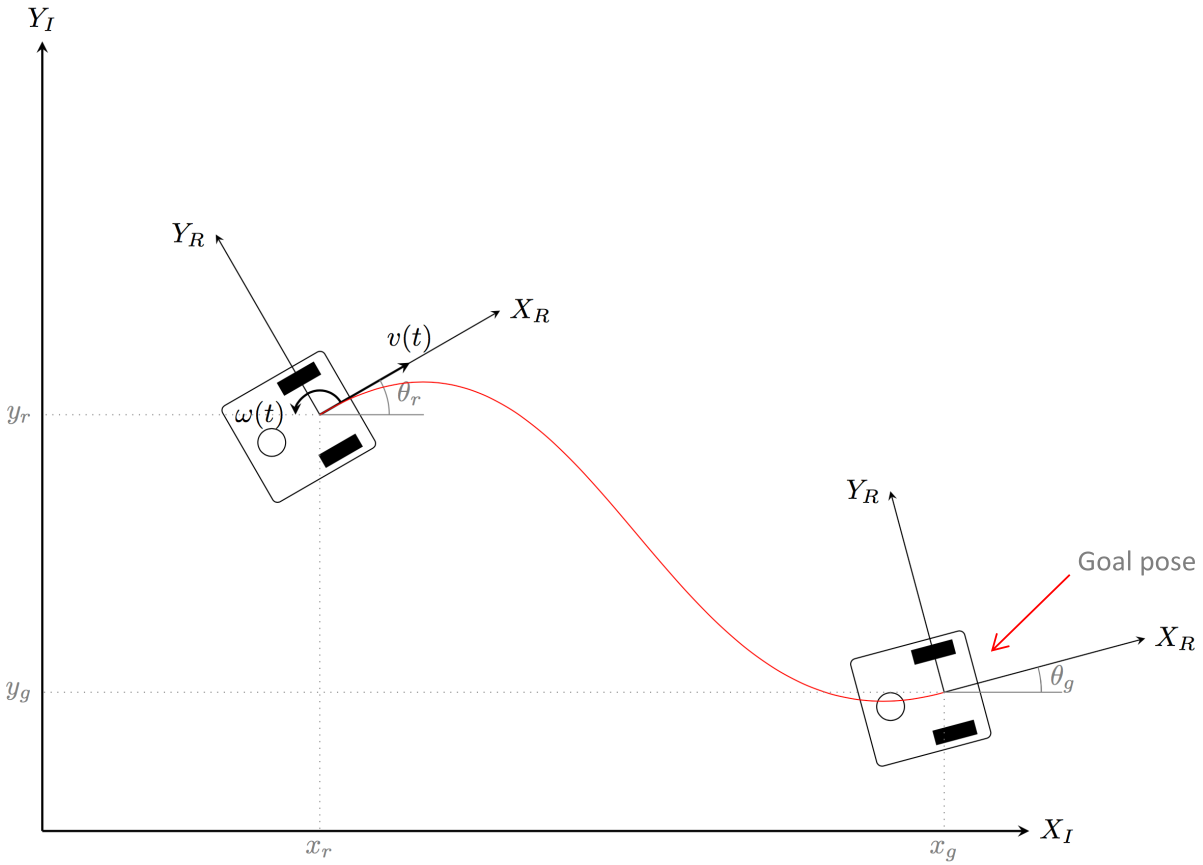
Go-to-Position as a Control Problem

Solution 2: MIMO Controller

As we saw in the part on PID control,
selecting the gain values is crucial for effective control:

- if the ratio of K_p^{pos} to K_p^{h} is too high, the path will overshoot
- if the ratio of K_p^{pos} to K_p^{h} is too low, the path will oscillate

The Go-to-Pose Problem



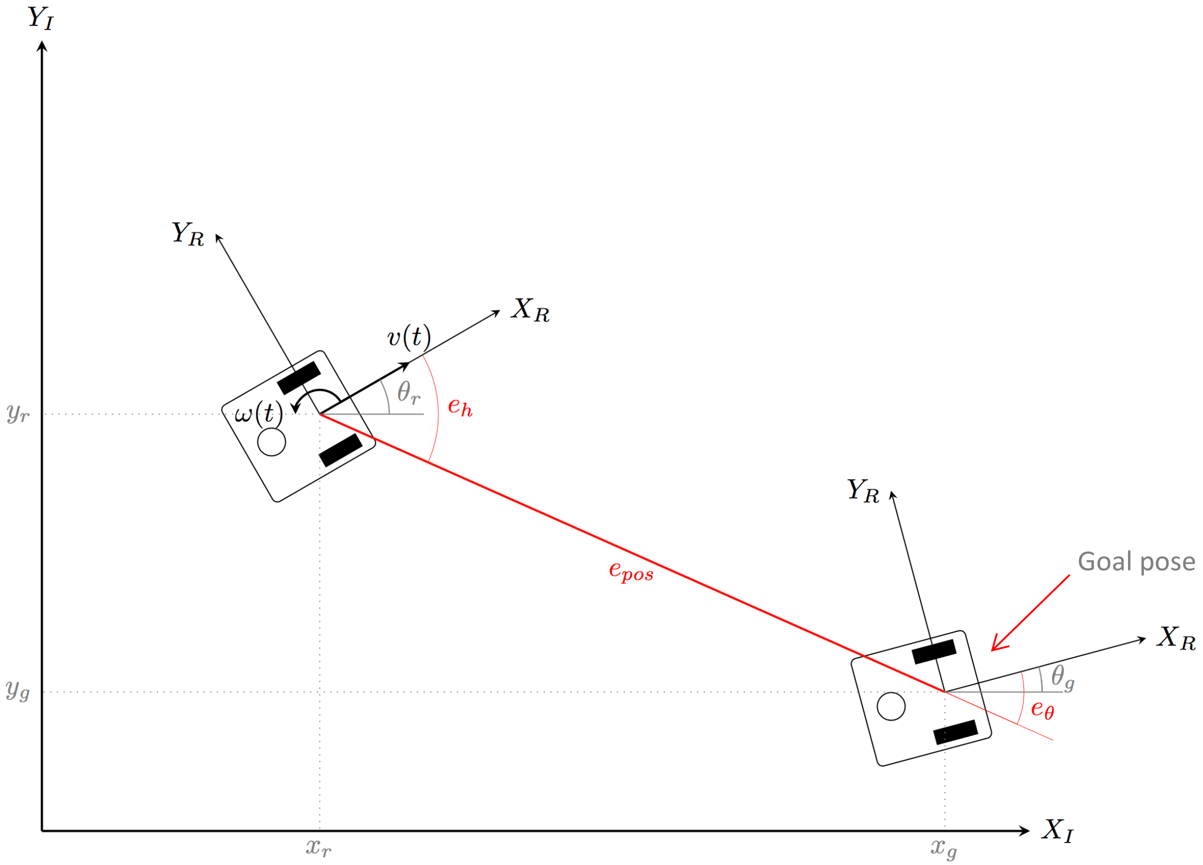
The Go-to-Pose Problem

- The robot knows its own global current position
  - (x_r, y_r, θ_r)

- It knows the global pose of the goal
  - (x_g, y_g, θ_g)

- Compute error in position, heading, and orientation
  - (e_{pos}, e_h, e_θ)

The Go-to-Pose Problem



The Go-to-Pose Problem

- The robot knows its own global current position
  - (x_c, y_c, θ_c)
- It knows the global position of the goal
  - (x_g, y_g, θ_g)
- Compute error in position, heading, and orientation
  - (e_{pos}, e_h, e_θ)
- Reduce all three errors to zero
  - by generating the appropriate forward and angular velocities (v, ω) or, alternatively,
  - by generating the appropriate angular velocities of the wheels, (v_R, v_L) i.e. $(\dot{\phi}_1, \dot{\phi}_2)$

Go-to-Pose as a Control Problem
MIMO Controller

Algorithm goto3(x_g, y_g) using proportional control

Global variables: the current robot position and orientation (x_r, y_r, θ_r)

Arguments: - the goal pose of the robot (x_g, y_g, θ_g)

- the proportional gains for controlling position, heading, and orientation

K_p^{pos}

K_p^{h}

K_p^{θ}

- the tolerance on position error Δ_{pos}

Go-to-Pose as a Control Problem

MIMO Controller

Do

Compute the current position of the robot (x_r, y_r)

Compute the distances from the robot position to the target position (d_x, dy)

$$d_x = x_g - x_r$$

$$d_y = y_g - y_r$$

Compute the position, heading, and orientation errors(e_{pos}, e_h, e_θ)

$$e_{pos} = \text{sqrt}(d_x^2 + d_y^2)$$

$$e_h = \text{atan2}(d_y, d_x) - \theta_r$$

$$e_\theta = \theta_g - \text{atan2}(d_y, d_x)$$

Compute the forward velocity and angular velocity (v, ω)

$$v = K_p^{pos} e_{pos}$$

$$\omega = K_p^h e_h + K_p^\theta e_\theta$$

If required, convert (v, ω) to $(\dot{\phi}_1, \dot{\phi}_2)$ using inverse kinematics

Send velocities (v, ω) or $(\dot{\phi}_1, \dot{\phi}_2)$ to the robot

Pause some time

while $|e_{pos}| > \Delta_{pos}$

Send velocities $(0, 0)$ to the robot

The difference between the required heading and the current orientation

The difference between the goal orientation and the required heading

The intuition behind this controller is that the terms $K_p^{pos} e_{pos}$ and $K_p^h e_h$ drive the robot along a line in a heading towards the goal position (same as the go-to-position controller) while the term $K_p^\theta e_\theta$ rotates the line so that the error between the heading and the goal orientation is zero (for details, see P. Corke, Robotics, Vision and Control, Springer, 2017, Section 4.2.4)

Go-to-Pose as a Control Problem

MIMO Controller

Do

Compute the current position of the robot (x_r, y_r)

Compute the distances from the robot position to the target position (d_x, dy)

$d_x = x_g - x_r$

$d_y = y_g - y_r$

Compute the position, heading, and orientation errors(e_{pos}, e_h, e_θ)

$e_{pos} = \text{sqrt}\,(d_x^2 + d_y^2)$

$e_h = \text{atan2}\,(d_y, d_x) - \theta_r$ ← The difference between the required heading and the current orientation

$e_\theta = \theta_g - \text{atan2}\,(d_y, d_x)$ ← The difference between the goal orientation and the required heading

Compute the forward velocity and angular velocity (v, ω)

$v = K_p^{pos}\, e_{pos}$

$\omega = K_p^h\, e_h + K_p^\theta\, e_\theta$ ← Note that K_p^θ is negative so that the robot understeers or oversteers to reposition the robot at a location where the new heading is more closely aligned with the goal orientation (for details, see P. Corke, Robotics, Vision and Control, Springer, 2017, Section 4.2.4)

If required, convert (v, ω) to $(\dot{\phi}_1, \dot{\phi}_2)$ using inverse kinematics

Send velocities (v, ω) or $(\dot{\phi}_1, \dot{\phi}_2)$ to the robot

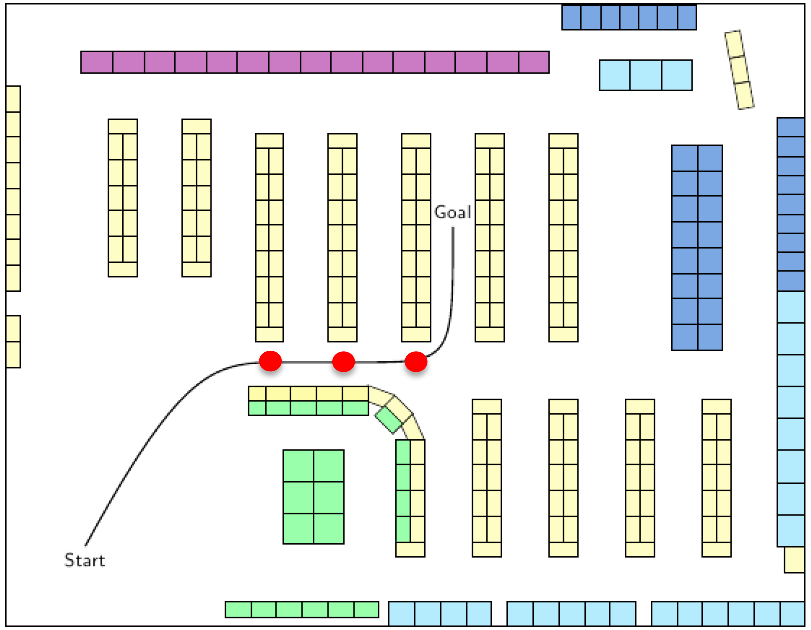
Pause some time

while $|e_{pos}| > \Delta_{pos}$

Send velocities $(0, 0)$ to the robot

Path Tracking Through Waypoints

- The goal position need not be the final goal
- There can also be intermediate goal positions, called waypoints
- We can adapt the go-to-position algorithm to track through a list of waypoints



Path Tracking Through Waypoints

But we will need to modify the velocity control

- Recall: v depends on e_{pos} of the current target
- But we do not want to stop at intermediate points
- Possible solution: use fixed v until the last point