UiO : Department of Technology Systems
University of Oslo

11. Mobile robots II

Kim Mathiassen

Lecture overview

- Planning (11.5)
- Motion Control (11.6)
- Odometric Localization (11.7)

11.5 Planning

- Planning a trajectory can be divided into two parts
  - Finding a path
  - Finding a timing law
- Want to find

$$\boldsymbol{q}(t), \text{ for } t \in [t_i, t_f]$$
$$\boldsymbol{q}(t_i) = \boldsymbol{q}_i \qquad \boldsymbol{q}(t_f) = \boldsymbol{q}_f$$

- Then $\boldsymbol{q}(t)$ can be broken into
  - Geometric path $\boldsymbol{q}(s), \text{ with } d\boldsymbol{q}(s)/ds \neq 0$
  - Timing law $s = s(t)$ where $s(t_i) = s_i \quad s(t_f) = s_f$

$$\dot{s}(t) \geq 0, \text{ for } t \in [t_i, t_f]$$

- One possible choice is $s_i = 0$ and $s_f = L$, where L is the path length

11.5.1 Path and timing law

- The space-time separation implies

$$\dot{\boldsymbol{q}} = \frac{d\boldsymbol{q}}{dt} = \frac{d\boldsymbol{q}}{ds}\,\dot{s} = \boldsymbol{q}'\dot{s},$$

- The non-holonomic constraint can then be written as

$$\boldsymbol{A}(\boldsymbol{q})\dot{\boldsymbol{q}} = \boldsymbol{A}(\boldsymbol{q})\boldsymbol{q}'\dot{s} = \boldsymbol{0}.$$

- If $\dot{s}(t) > 0$, for $t \in [t_i, t_f]$, then

$$\boldsymbol{A}(\boldsymbol{q})\boldsymbol{q}' = \boldsymbol{0}. \qquad (11.40)$$

Path and timing law

- The geometric admissible path can be defined as the solution to the nonlinear system

$$\boldsymbol{q}' = \boldsymbol{G}(\boldsymbol{q})\widetilde{\boldsymbol{u}}, \qquad (11.41)$$

- Where $\widetilde{\boldsymbol{u}}$ is a vector of geometric inputs

$$\boldsymbol{u}(t) = \widetilde{\boldsymbol{u}}(s)\dot{s}(t)$$

- Once the geometric inputs are assign, the path is uniquely determined
- The choice of timing law will then identify a particular trajectory

Path and timing law - unicycle

- Non-holonomic constraint

$$\boldsymbol{A}(\boldsymbol{q})\boldsymbol{q}' = \boldsymbol{0} \qquad [\sin\theta \quad -\cos\theta \quad 0]\,\boldsymbol{q}' = x'\sin\theta - y'\cos\theta = 0,$$

- This expresses the fact that the tangent to the Cartesian path must be aligned with the driving direction of the robot
- This means that there cannot be any discontinuities in the path, unless $\dot{s} = 0$ where the robots turns on the spot
- Geometric admissible paths are the solution to

$$\begin{aligned} x' &= \widetilde{v}\cos\theta \\ y' &= \widetilde{v}\sin\theta \\ \theta' &= \widetilde{\omega}, \end{aligned} \qquad (11.42)$$

$$v(t) = \widetilde{v}(s)\dot{s}(t) \qquad (11.43)$$

$$\omega(t) = \widetilde{\omega}(s)\dot{s}(t). \qquad (11.44)$$

11.5.2 Flat outputs

- Many kinematic models of mobile robots exhibit a property known as *differential flatness*
  - This includes the unicycle and the bicycle
- A non-linear system $\dot{x} = f(x) + G(x)u$ is differential flat if
  - Both the state vector x and the control input u can be expressed algebraically as a function of y and its derivatives

$$x = x(y, \dot{y}, \ddot{y}, \ldots, y^{(r)})$$
$$u = u(y, \dot{y}, \ddot{y}, \ldots, y^{(r)}).$$

- This means that once a trajectory is assign for y, both the state vector x and the control input u is uniquely determined

Flat outputs – unicycle case

- Assume that the flat outputs $\boldsymbol{y}$ is the Cartesian path $(x(s), y(s))$
- The state trajectory is $\boldsymbol{q}(s) = [\, x(s) \quad y(s) \quad \theta(s) \,]^T$ where

$$\theta(s) = \text{Atan2}\,(y'(s), x'(s)) + k\pi \qquad k = 0, 1. \qquad (11.45)$$

- The control input is

$$\widetilde{v}(s) = \pm\sqrt{(x'(s))^2 + (y'(s))^2} \qquad (11.46)$$

$$\widetilde{\omega}(s) = \frac{y''(s)x'(s) - x''(s)y'(s)}{(x'(s))^2 + (y'(s))^2}. \qquad (11.47)$$

- This shows that the unicycle has the differential flatness property

Flat outputs – unicycle case remarks

- K can be 0 or 1 depending on the direction of the robot motion (forward or backwards)
- The sign of $\widetilde{v}(s)$ also depend on the direction of the motion
- If $x'(\bar{s}) = y'(\bar{s}) = 0$ for some $\bar{s} \in [s_i, s_f]$ one has $\widetilde{v}(\bar{s}) = 0$ This happens when there are motion inversions in the path
- Then (11.45) does not define the orientation
- The possibility of reconstructing θ and $\widetilde{\omega}$ are lost if the Cartesian trajectory degenerates to a point

Flat outputs – Chain form

- In driftless dynamic systems differential flatness is a necessary and sufficient condition for transformability to the chained form
- Chained form of the unicycle is

$$\begin{aligned} z_1 &= \theta \\ z_2 &= x\cos\theta + y\sin\theta \\ z_3 &= x\sin\theta - y\cos\theta \end{aligned} \quad (11.23) \qquad \begin{aligned} v &= v_2 + z_3 v_1 \\ \omega &= v_1, \end{aligned} \quad (11.24) \qquad \begin{aligned} \dot{z}_1 &= v_1 \\ \dot{z}_2 &= v_2 \\ \dot{z}_3 &= z_2 v_1. \end{aligned} \quad (11.25)$$

- Using y = (z_1, z_3) yields

$$z_2 = \frac{\dot{z}_3}{\dot{z}_1} \qquad v_1 = \dot{z}_1 \qquad v_2 = \frac{\dot{z}_1 \ddot{z}_3 - \ddot{z}_1 \dot{z}_3}{\dot{z}_1^2}.$$

11.5.3 Path planning

- Path can be planned efficiently for robots that have the differential flatness property
- Any interpolation scheme can be used, as long as the boundary conditions are satisfied
- The configuration variables and control input can then be calculated from $\boldsymbol{y}(s)$
- The resulting configuration space task will automatically satisfy the non-holonomic constraints

Planning via Cartesian polynomials

- The problem can be solved by interpolating the initial values x_i, y_i and the final values x_f, y_f of the flat output x, y
- Let $s_i = 0$ and $s_f = 1$
- One may use the following cubic polynomial

$$x(s) = s^3 x_f - (s-1)^3 x_i + \alpha_x s^2(s-1) + \beta_x s(s-1)^2$$
$$y(s) = s^3 y_f - (s-1)^3 y_i + \alpha_y s^2(s-1) + \beta_y s(s-1)^2,$$

- With the additional boundary conditions

$$x'(0) = k_i \cos\theta_i \qquad x'(1) = k_f \cos\theta_f$$
$$y'(0) = k_i \sin\theta_i \qquad y'(1) = k_f \sin\theta_f,$$

- Where $k_i \neq 0,\ k_f \neq 0$ are free parameters

Planning via Cartesian polynomials

- The additional boundary conditions are necessary to guarantee that the unicycle arrives with the same kind of motion which it leaves the initial configuration
- Since a cubic function is used the Cartesian path does not contain motion inversion
- For example, letting $k_i = k_f = k > 0$ yields

$$\begin{bmatrix} \alpha_x \\ \alpha_y \end{bmatrix} = \begin{bmatrix} k\cos\theta_f - 3x_f \\ k\sin\theta_f - 3y_f \end{bmatrix} \qquad \begin{bmatrix} \beta_x \\ \beta_y \end{bmatrix} = \begin{bmatrix} k\cos\theta_i + 3x_i \\ k\sin\theta_i + 3y_i \end{bmatrix}.$$

- The choice of k influence the path

$$\widetilde{v}(0) = k_i \qquad \widetilde{v}(1) = k_f.$$

Planning via Cartesian polynomials

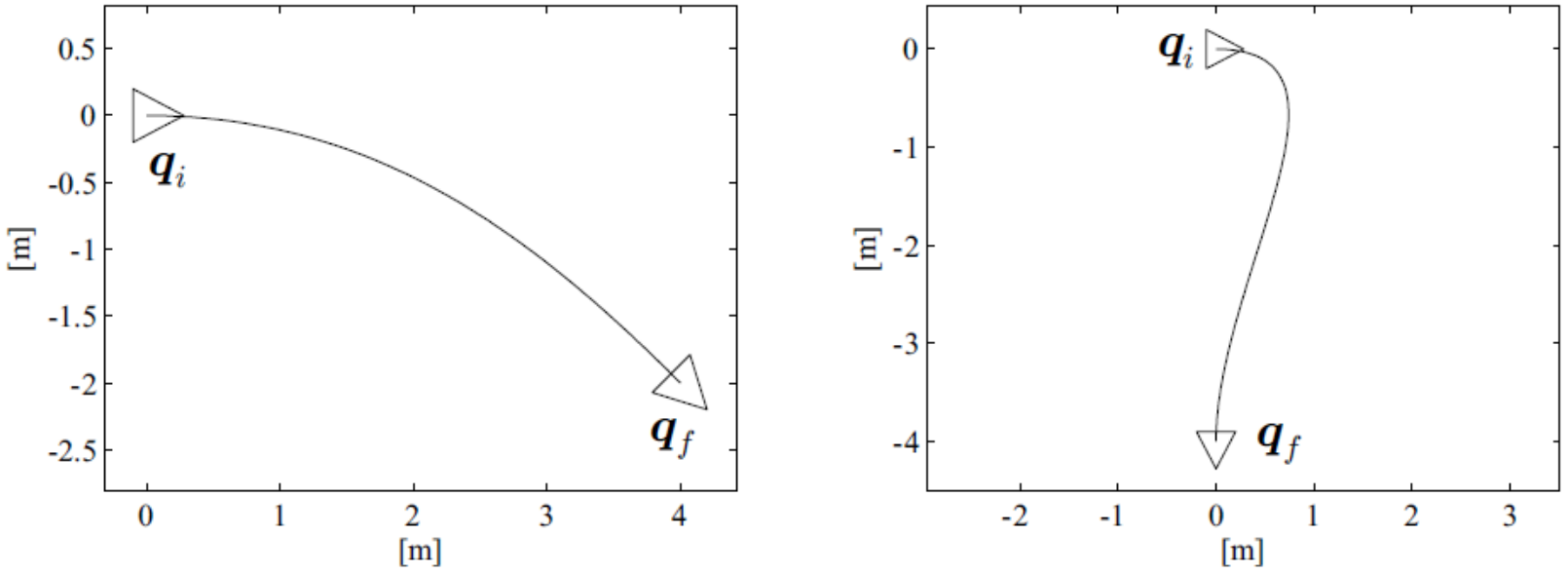


Fig. 11.5. Two parking manoeuvres planned via cubic Cartesian polynomials; in both cases $k = 5$ has been used

Planning via Cartesian polynomials

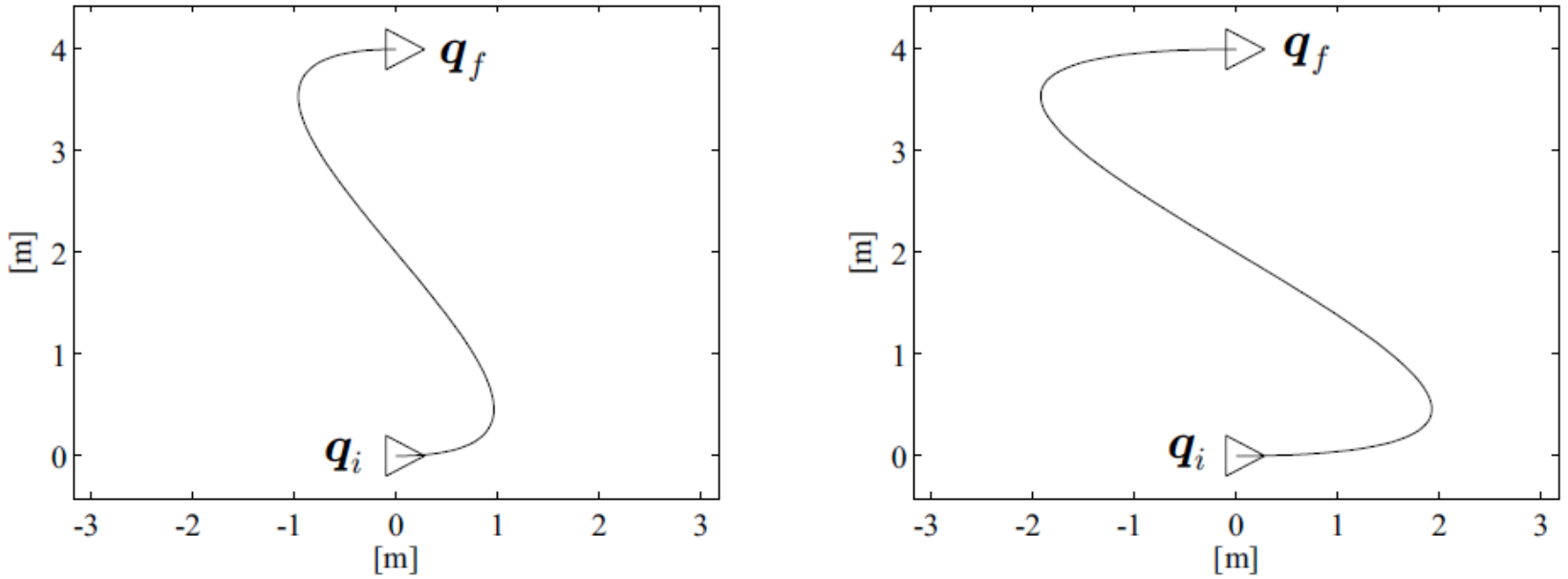


Fig. 11.6. Planning a parallel parking manoeuvre via cubic Cartesian polynomials; *left*: with $k = 10$, *right*: with $k = 20$

Planning via Cartesian polynomials

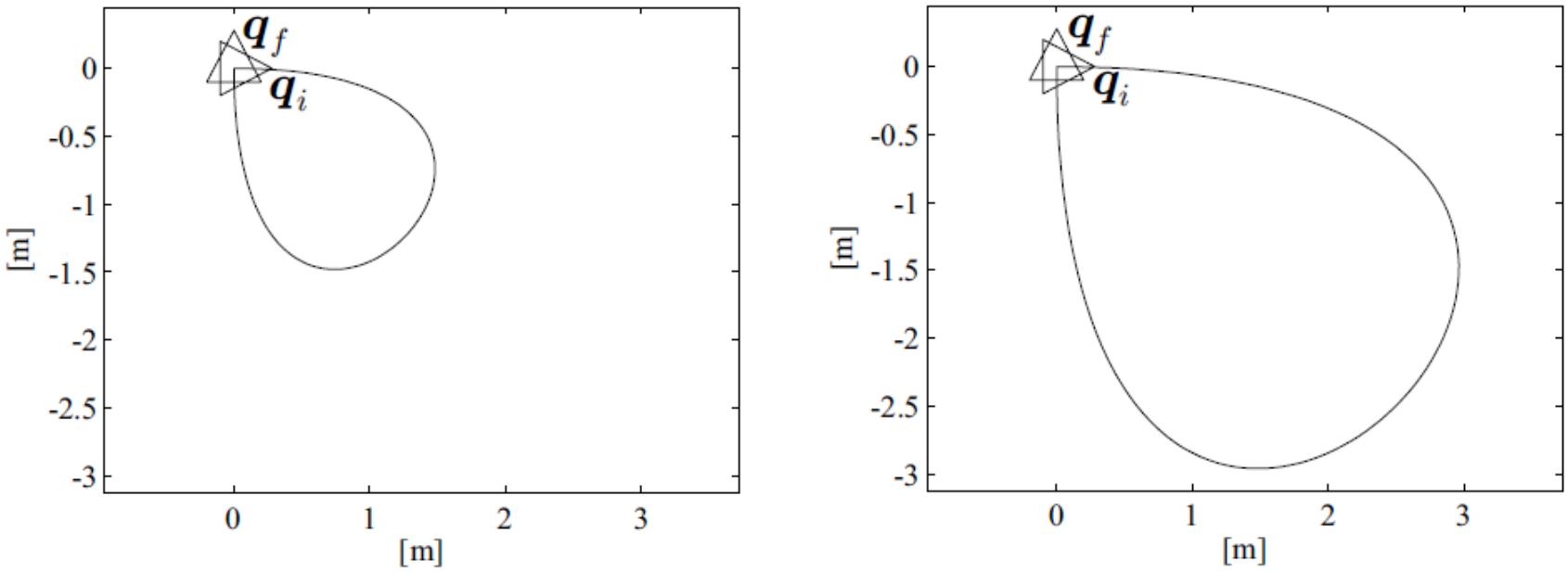


Fig. 11.7. Planning a pure reorientation manoeuvre via cubic Cartesian polynomials; *left*: with $k = 10$, *right*: with $k = 20$

Planning via the chained form

- Planning in chained form coordinates z
- Can be generalized to other kinematic models (bicycle)
- First the initial and final values z_i and z_f must be computed from q_i and q_f
- Then it is sufficient to interpolate the initial and final values of z_1 and z_3 with the appropriate boundary conditions

$$z_2 = z_3'/z_1'$$

Planning via the chained form

- We use a cubic function to solve the problem, under the assumption $z_{1,i} \neq z_{1,f}$

$$z_1(s) = z_{1,f}s - (s-1)z_{1,i}$$
$$z_3(s) = s^3 z_{3,f} - (s-1)^3 z_{3,i} + \alpha_3 s^2(s-1) + \beta_3 s(s-1)^2,$$

- Note that $z_1'(s)$ is constant, equal to $z_{1,f} - z_{1,i} \neq 0$
- The unknowns $\alpha_3,\ \beta_3$ must be determined using the boundary conditions on z_2

$$\frac{z_3'(0)}{z_1'(0)} = z_{2i} \qquad \frac{z_3'(1)}{z_1'(1)} = z_{2f},$$

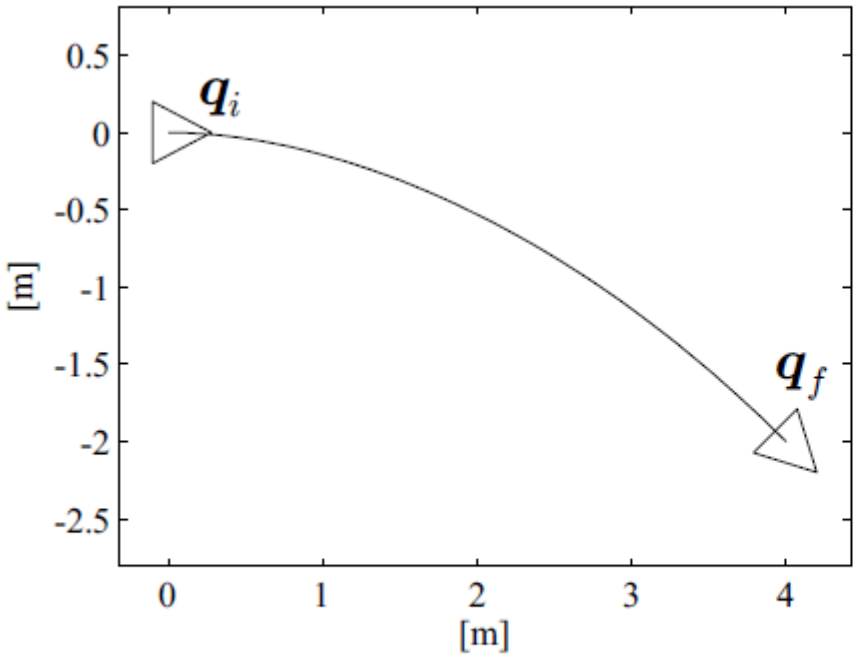
Planning via the chained form

- Solving for α_3, β_3 yields

$$\alpha_3 = z_{2,f}(z_{1,f} - z_{1,i}) - 3z_{3,f}$$
$$\beta_3 = z_{2,i}(z_{1,f} - z_{1,i}) + 3z_{3,i}.$$

- This scheme does not handle $z_{1,i} = z_{1,f}$, i.e., when $\theta_i = \theta_f$
- To overcome this one can introduce a via point
- Another solution is to set $z_{1,f} = z_{1,i} + 2\pi$, which will cause the robot to perform a complete rotation, but it will have the same final configuration

Planning via the chained form



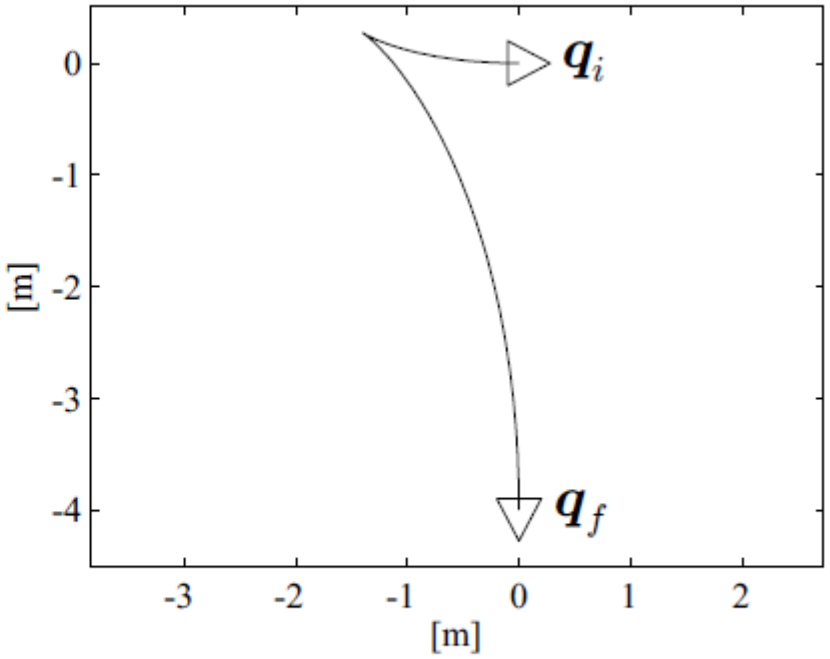


Fig. 11.8. Two parking manoeuvres planned via the chained form

Planning via the chained form

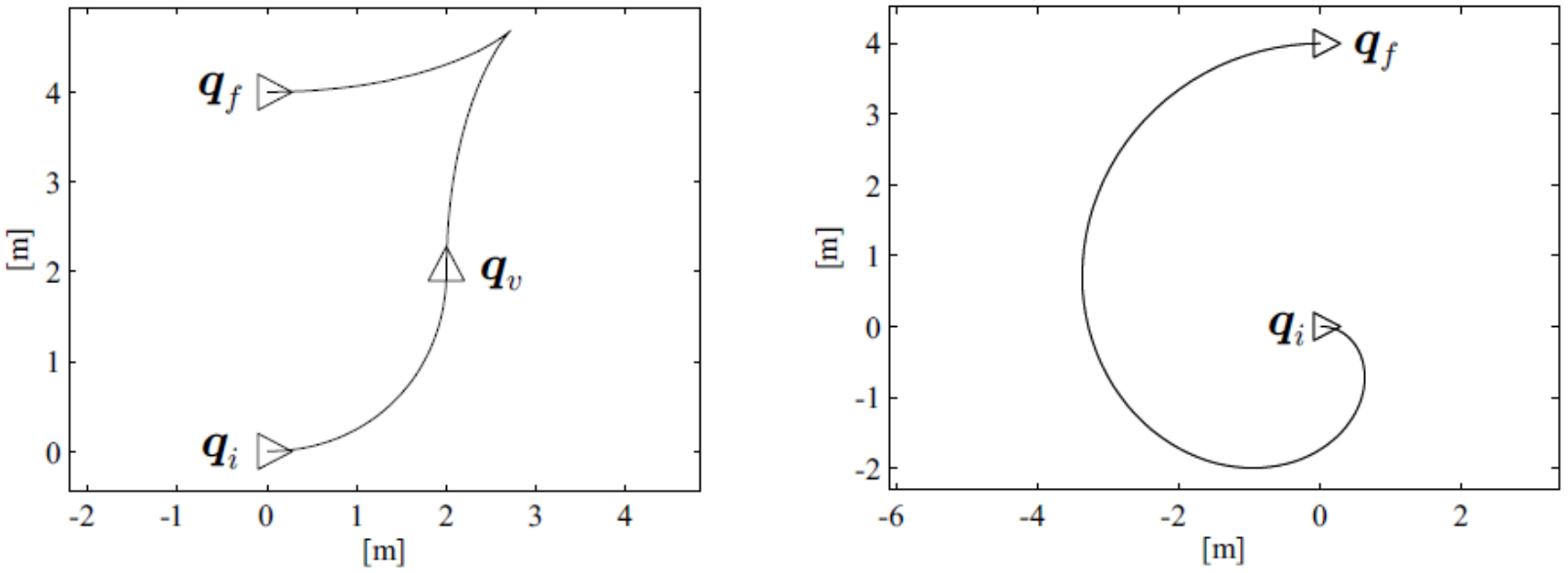


Fig. 11.9. Planning a parallel parking manoeuvre via the chained form; *left*: adding a via point q_v, *right*: letting $\theta_f = \theta_i + 2\pi$

Planning via the chained form

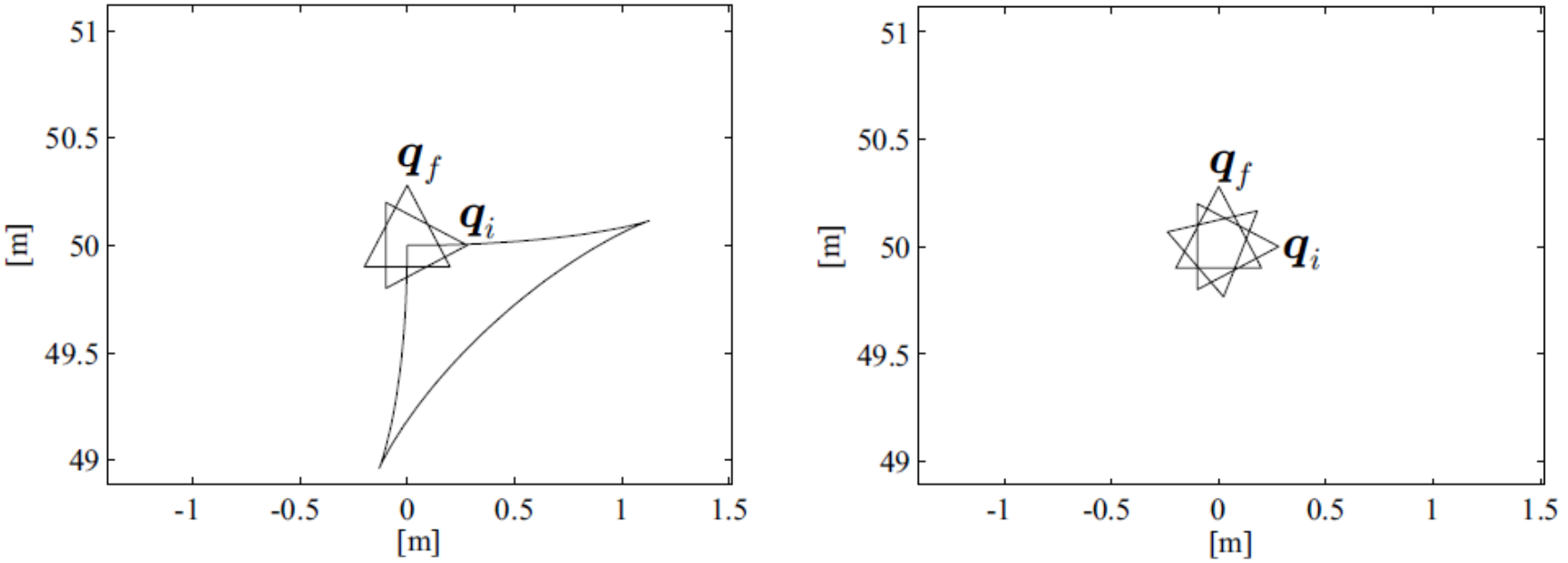


Fig. 11.10. Planning a pure reorientation manoeuvre via the chained form; *left*: with the coordinate transformation (11.23), *right*: with the coordinate transformation (11.52)

Planning via the chained form

- The last solution uses a different transform
- The problem with the transform is that $\theta_i \neq \theta_f$ implies $z_{2,i} \neq z_{2,f}$ and $z_{3,i} \neq z_{3,f}$ even if $x_i = x_f$, $y_i = y_f$
- Instead using the transform

$$z_1 = \theta - \theta_f$$
$$z_2 = (x - x_i)\cos\theta + (y - y_i)\sin\theta \qquad (11.52)$$
$$z_3 = (x - x_i)\sin\theta - (y - y_i)\cos\theta,$$

- This transform places the origin of the reference frame (z_2, z_3) of the initial position of the unicycle

Planning via parameterized inputs

- A conceptually different approach is to write the input, rather than the path, in parameterized form
- For the system

$$\begin{aligned} z_1' &= \widetilde{v}_1 \\ z_2' &= \widetilde{v}_2 \\ z_3' &= z_2 \widetilde{v}_1 \\ &\vdots \\ z_n' &= z_{n-1} \widetilde{v}_1. \end{aligned}$$

- Let the geometric input be chosen as

$$\widetilde{v}_1 = \operatorname{sgn}(\Delta) \tag{11.48}$$

$$\widetilde{v}_2 = c_0 + c_1 s + \ldots + c_{n-2} s^{n-2}, \tag{11.49}$$

$$\Delta = z_{1,f} - z_{1,i} \text{ and } s \in [s_i, s_f] = [0, |\Delta|]$$

Planning via parameterized inputs

- The geometric inputs must satisfy the boundary condition $\boldsymbol{z}(s_f) = \boldsymbol{z}_f$
- This condition can be expressed as the a linear system of equations

$$\boldsymbol{D}(\Delta) \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_{n-2} \end{bmatrix} = \boldsymbol{d}(\boldsymbol{z}_i, \boldsymbol{z}_f, \Delta) \qquad (11.50)$$

- Where $\boldsymbol{D}(\Delta)$ is invertible if $\Delta \neq 0$

Planning via parameterized inputs - unicycle

- For the unicycle case this will be

$$\boldsymbol{D} = \begin{bmatrix} |\Delta| & \frac{\Delta^2}{2} \\ \operatorname{sgn}(\Delta)\frac{\Delta^2}{2} & \frac{\Delta^3}{6} \end{bmatrix} \quad \boldsymbol{d} = \begin{bmatrix} z_{2,f} - z_{2,i} \\ z_{3,f} - z_{3,i} - z_{2,i}\Delta \end{bmatrix}. \qquad (11.51)$$

- Still $z_{1,i} = z_{1,f}$ yields a singular case
- After fining the parameters both $\boldsymbol{z}(s)$ and $\widetilde{\boldsymbol{v}}(s)$ must be converted to $\boldsymbol{q}(s)$ and $\widetilde{\boldsymbol{u}}(s)$
- s don't represent arc length
- The method does not use flat outputs directly, but relies on closed-form integrability of the chained form (equivalent to differential flattness)

11.5.4 Trajectory planning

- Once the path $\boldsymbol{q}(s),\ s \in [s_i, s_f]$ has been planned, it is possible to choose a timing law $s = s(t)$
- Usually the velocities are subject to constraints

$$|v(t)| \leq v_{\max} \qquad |\omega(t)| \leq \omega_{\max} \qquad \forall t, \tag{11.53}$$

- It is necessary to verify that the velocities along the trajectory are admissible
- It is possible to slow down the timing

$$v(t) = \widetilde{v}(s)\frac{ds}{d\tau}\frac{d\tau}{dt} = \widetilde{v}(s)\frac{ds}{d\tau}\frac{1}{T} \tag{11.54}$$

$$\omega(t) = \widetilde{\omega}(s)\frac{ds}{d\tau}\frac{d\tau}{dt} = \widetilde{\omega}(s)\frac{ds}{d\tau}\frac{1}{T}, \tag{11.55}$$

$$\tau = t/T, \text{ with } T = t_f - t_i$$

11.5.5 Optimal trajectories

- Utilize Pontriagin minimum principle, or other optimization techniques to find an optimal trajectory

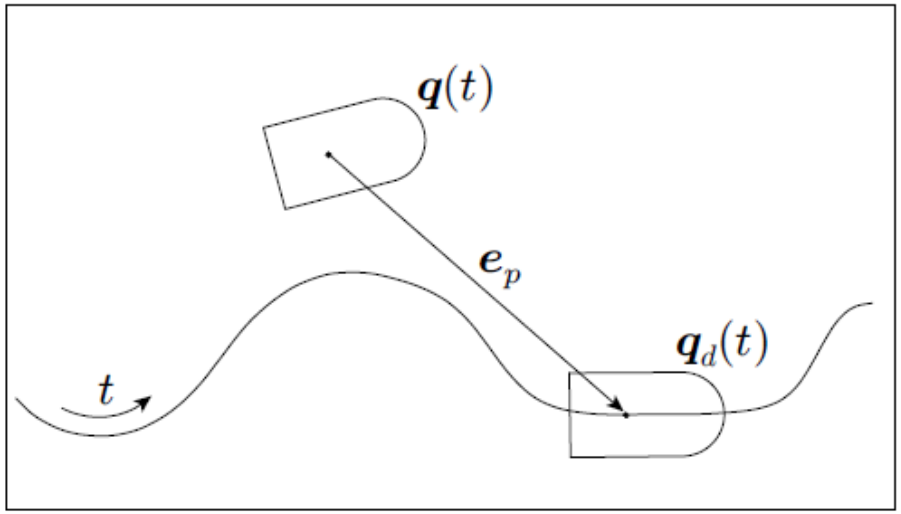
- Not part of the curriculum

11.6 Motion control

- Motion control of mobile robots are generally formulated with respect to the kinematic model
- We assume that the control input $\dot{q}$ is the input to our system
- There are two reasons for this simplification
  - In the dynamics section state feedback cancelled the dynamic effects, reducing the system to a first order kinematic model
  - The majority of mobile robots already have low-level controllers implemented from the producers

Motion control

- Trajectory tracking
  Asymptotically track a Cartesian trajectory $(x_d(t), y_d(t))$
- Posture regulation
  Asymptotically reach a desired posture $\boldsymbol{q}_d$



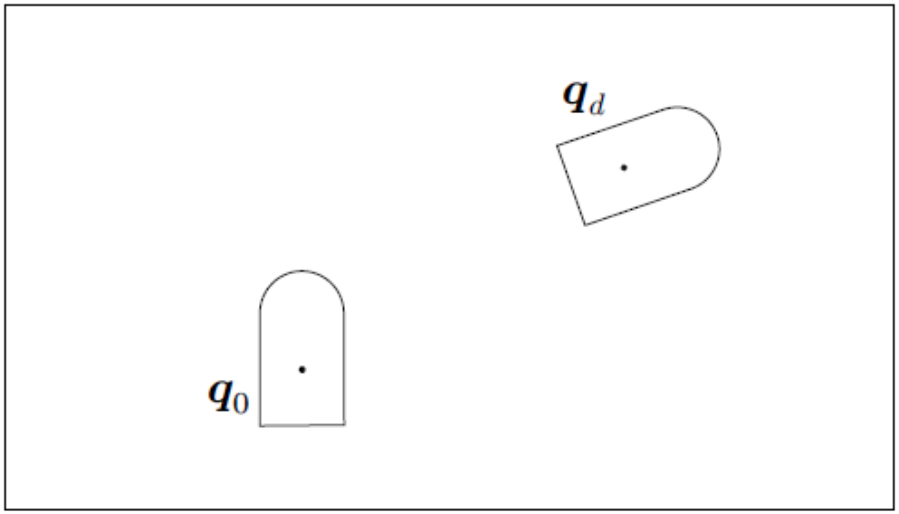


Fig. 11.13. Control problems for a unicycle; *left*: trajectory tracking, *right*: posture regulation

11.6.1 Trajectory tracking

- Assuming that the trajectory is admissible, i.e. satisfies the equation

$$\begin{aligned} \dot{x}_d &= v_d \cos\theta_d \\ \dot{y}_d &= v_d \sin\theta_d \\ \dot{\theta}_d &= \omega_d \end{aligned} \qquad (11.57)$$

- The planned trajectories from chapter 11.5 satisfies these conditions

Trajectory tracking – error definition

- Comparing the desired state $\boldsymbol{q}_d(t)$ and the actual measured state $\boldsymbol{q}(t)$ in a reference frame aligned with the robot

$$\boldsymbol{e} = \begin{bmatrix} e_1 \\ e_2 \\ e_3 \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_d - x \\ y_d - y \\ \theta_d - \theta \end{bmatrix}$$

- Differentiating with regards to time, and using the kinematic model vields

$$\begin{aligned} \dot{e}_1 &= v_d \cos e_3 - v + e_2\,\omega \\ \dot{e}_2 &= v_d \sin e_3 - e_1\,\omega \\ \dot{e}_3 &= \omega_d - \omega. \end{aligned} \qquad (11.61)$$

Trajectory tracking – error definition

- Using the input transform

$$v = v_d \cos e_3 - u_1 \tag{11.62}$$

$$\omega = \omega_d - u_2, \tag{11.63}$$

- Yields the following tracking error dynamics

$$\dot{\boldsymbol{e}} = \begin{bmatrix} 0 & \omega_d & 0 \\ -\omega_d & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \boldsymbol{e} + \begin{bmatrix} 0 \\ \sin e_3 \\ 0 \end{bmatrix} v_d + \begin{bmatrix} 1 & -e_2 \\ 0 & e_1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}. \tag{11.64}$$

- First term is linear, while second and third are nonlinear

Trajectory tracking – approximate linearization

- The simplest approach is to linearize around the reference trajectory, which means that $\boldsymbol{e} = \mathbf{0}$
- Setting $\sin e_3 = e_3$ and evaluating the input matrix on the trajectory yields

$$\dot{\boldsymbol{e}} = \begin{bmatrix} 0 & \omega_d & 0 \\ -\omega_d & 0 & v_d \\ 0 & 0 & 0 \end{bmatrix} \boldsymbol{e} + \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}. \qquad (11.65)$$

Trajectory tracking – approximate linearization

- Now consider the linear feedback

$$u_1 = -k_1\, e_1 \tag{11.66}$$

$$u_2 = -k_2\, e_2 - k_3\, e_3 \tag{11.67}$$

- This leads to the closed-loop linearized dynamics

$$\dot{\boldsymbol{e}} = \boldsymbol{A}(t)\,\boldsymbol{e} = \begin{bmatrix} -k_1 & \omega_d & 0 \\ -\omega_d & 0 & v_d \\ 0 & -k_2 & -k_3 \end{bmatrix} \boldsymbol{e}. \tag{11.68}$$

- With the characteristic polynomial of matrix $\boldsymbol{A}$

$$p(\lambda) = \lambda(\lambda + k_1)(\lambda + k_3) + \omega_d^2(\lambda + k_3) + v_d k_2(\lambda + k_1).$$

Trajectory tracking – approximate linearization

- To simplify the tuning and the characteristic polynomial, let

$$k_1 = k_3 = 2\zeta a \qquad k_2 = \frac{a^2 - \omega_d^2}{v_d}, \tag{11.69}$$

- Not the characteristic polynomial is reduced to

$$p(\lambda) = (\lambda + 2\zeta a)(\lambda^2 + 2\zeta a\lambda + a^2).$$
$$\text{with } \zeta \in (0,1) \text{ and } a > 0$$

- Now we have three constant eigenvalues
  - One real negative in $-2\zeta a$
  - Two complex conjugates with a real negative part, damping coefficient ζ and natural frequency a

Trajectory tracking – approximate linearization

- Since the system is time-varying, asymptotical stability is not guarantied
- Asymptotical stability is guarantied when the desired inputs are constants, this reduce the system to be time-invariant
- When the tracking error vanishes the control inputs will be the same as the desired control inputs (reduced to a feed forward action)
- Note that k_2 diverges when v_d goes to zero
- This means that this controller cannot handle motion inversions

Trajectory tracking – nonlinear control

- Consider the tracking error dynamics from (11.64), rewritten as

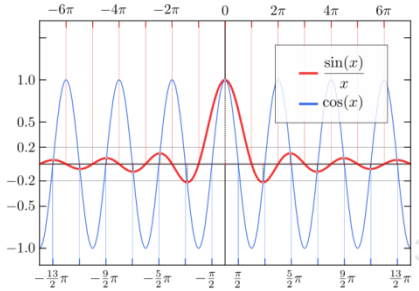
$$\begin{aligned}\dot{e}_1 &= e_2\,\omega + u_1\\ \dot{e}_2 &= v_d \sin e_3 - e_1\omega \qquad (11.70)\\ \dot{e}_3 &= u_2,\end{aligned}$$

- Using the nonlinear control law based in the approximate linearization

$$u_1 = -k_1(v_d, \omega_d)\, e_1 \qquad (11.71)$$

$$u_2 = -k_2\, v_d\, \frac{\sin e_3}{e_3}\, e_2 - k_3(v_d, \omega_d)\, e_3, \qquad (11.72)$$

$$k_1(\cdot,\cdot) > 0 \text{ and } k_3(\cdot,\cdot) > 0 \qquad k_2 > 0$$



Trajectory tracking – nonlinear control

- Both $k_1(\cdot,\cdot)$ and $k_3(\cdot,\cdot)$ are bounded functions with bounded derivatives
- k_2 is constant
- If the reference input v_d and ω_d are also bounded, and with bounded derivatives, and do not both converge to zero the tracking error converges to zero
- This can be proven using the Lyapunov method

Trajectory tracking – input/output linearization

- Consider the following outputs

$$y_1 = x + b\cos\theta$$
$$y_2 = y + b\sin\theta, \qquad b \neq 0$$

- They represent the Cartesian coordinates of a point B

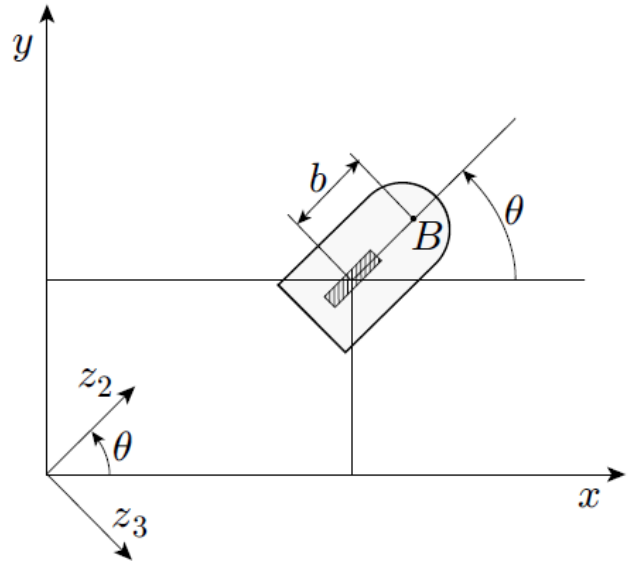


Fig. 11.3. Generalized coordinates for a unicycle

- The time derivatives are

$$\begin{bmatrix} \dot{y}_1 \\ \dot{y}_2 \end{bmatrix} = \begin{bmatrix} \cos\theta & -b\sin\theta \\ \sin\theta & b\cos\theta \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} = \boldsymbol{T}(\theta) \begin{bmatrix} v \\ \omega \end{bmatrix}. \qquad (11.74)$$

Trajectory tracking – input/output linearization

- The matrix $\boldsymbol{T}(\theta)$ has the determinant b, and is therefore invertible
- Using the transform

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \boldsymbol{T}^{-1}(\theta) \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta/b & \cos\theta/b \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

- Yields

$$\begin{aligned} \dot{y}_1 &= u_1 \\ \dot{y}_2 &= u_2 \\ \dot{\theta} &= \frac{u_2 \cos\theta - u_1 \sin\theta}{b}. \end{aligned} \qquad (11.75)$$

Trajectory tracking – input/output linearization

- This system is linear with regards to y_1, y_2 and u_1, u_2
- A simple linear controller on the following form guarantees exponential convergence

$$u_1 = \dot{y}_{1d} + k_1(y_{1d} - y_1) \qquad (11.76)$$

$$u_2 = \dot{y}_{2d} + k_2(y_{2d} - y_2), \qquad (11.77)$$

- Note that the orientation is not controlled
- In fact this tracking scheme does not use orientation error, it is based on the output error rather than the state error
- The reference Cartesian trajectory for the point B can be arbitrary (i.e. may have discontinuities without requiring the robot to stop and reorient itself)

Trajectory tracking - simulations

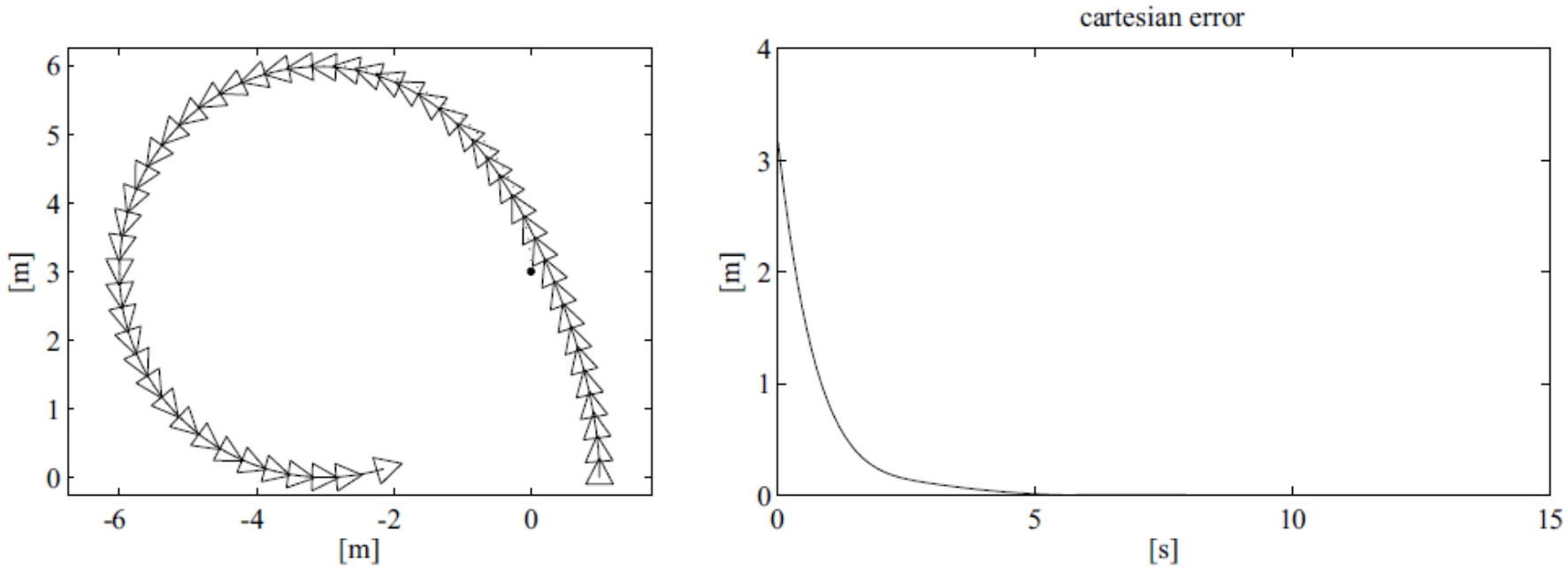


Fig. 11.14. Tracking a circular reference trajectory (*dotted*) with the controller based on approximate linearization; *left*: Cartesian motion of the unicycle, *right*: time evolution of the norm of the Cartesian error $\boldsymbol{e}_p$

Approximate linearization – Constant velocities

Trajectory tracking - simulations

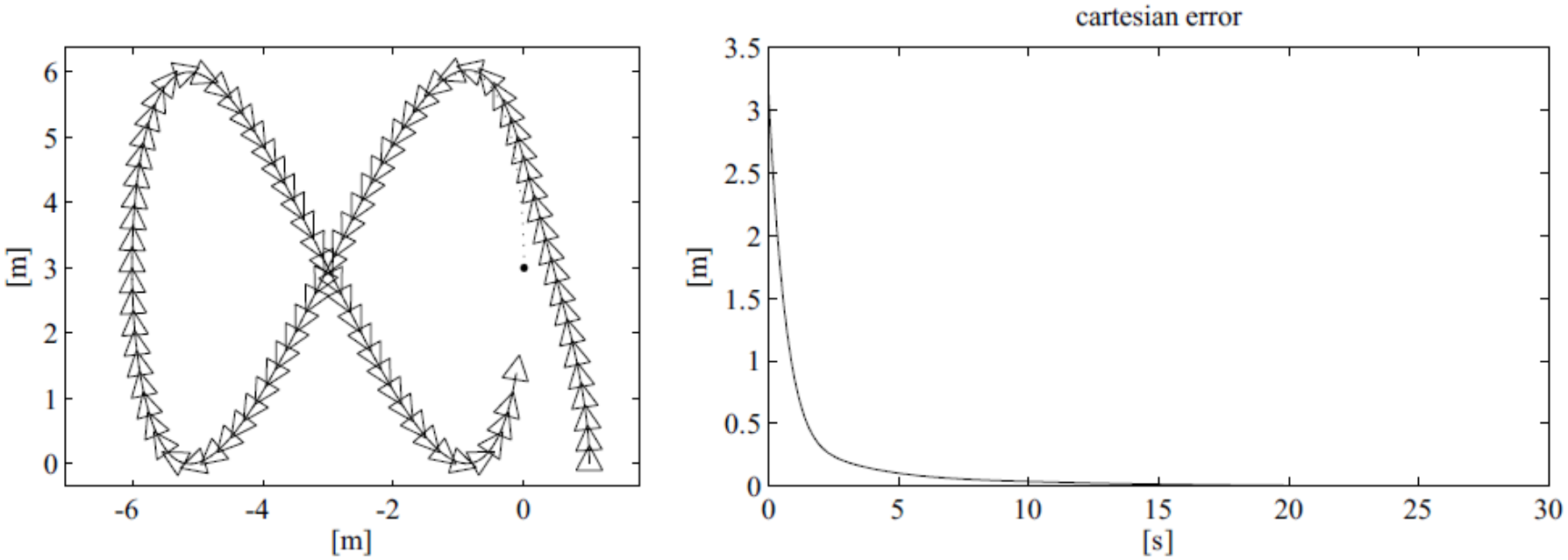


Fig. 11.15. Tracking a figure of eight-shaped reference trajectory (*dotted*) with the nonlinear controller; *left*: Cartesian motion of the unicycle, *right*: time evolution of the norm of the Cartesian error $\boldsymbol{e}_p$

Nonlinear control – Time varying v_d

Trajectory tracking - simulations

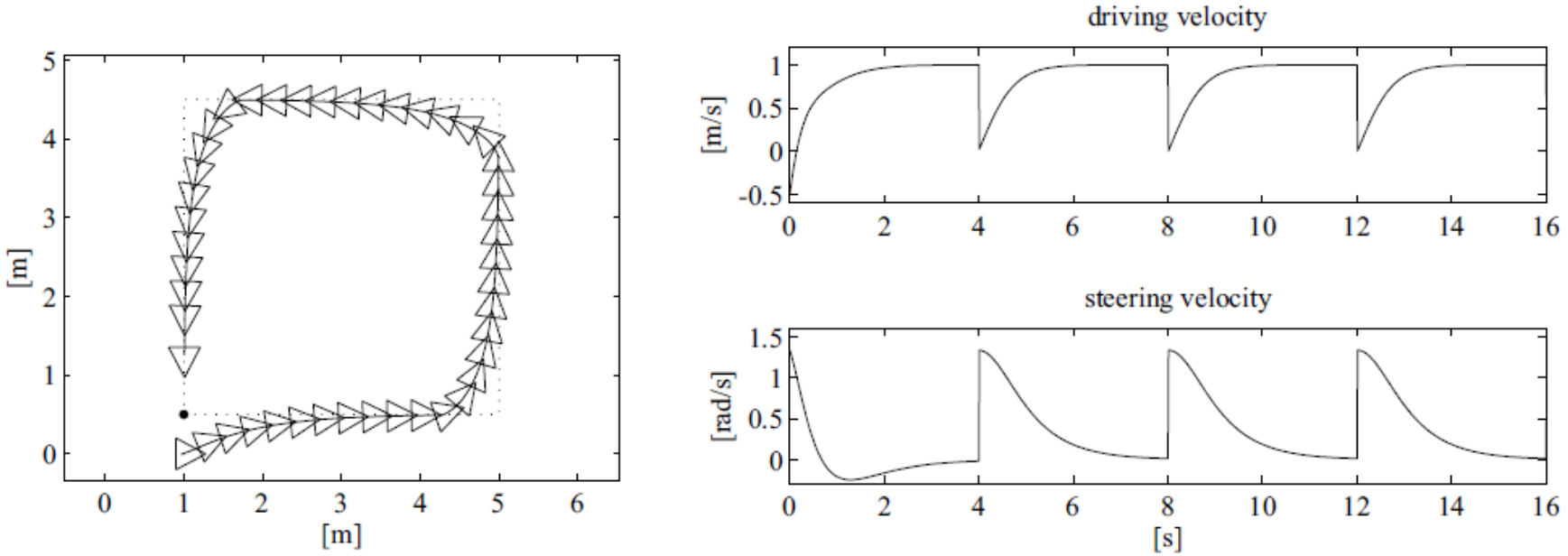


Fig. 11.16. Tracking a square reference trajectory (*dotted*) with the controller based on input/output linearization, with $b = 0.75$; *left*: Cartesian motion of the unicycle, *right*: time evolution of the velocity inputs v and ω

Input/output linearization – Constant velocity in a square. Cannot stop to reorient, which means that the trajectory cannot be followed perfectly

Trajectory tracking - simulations

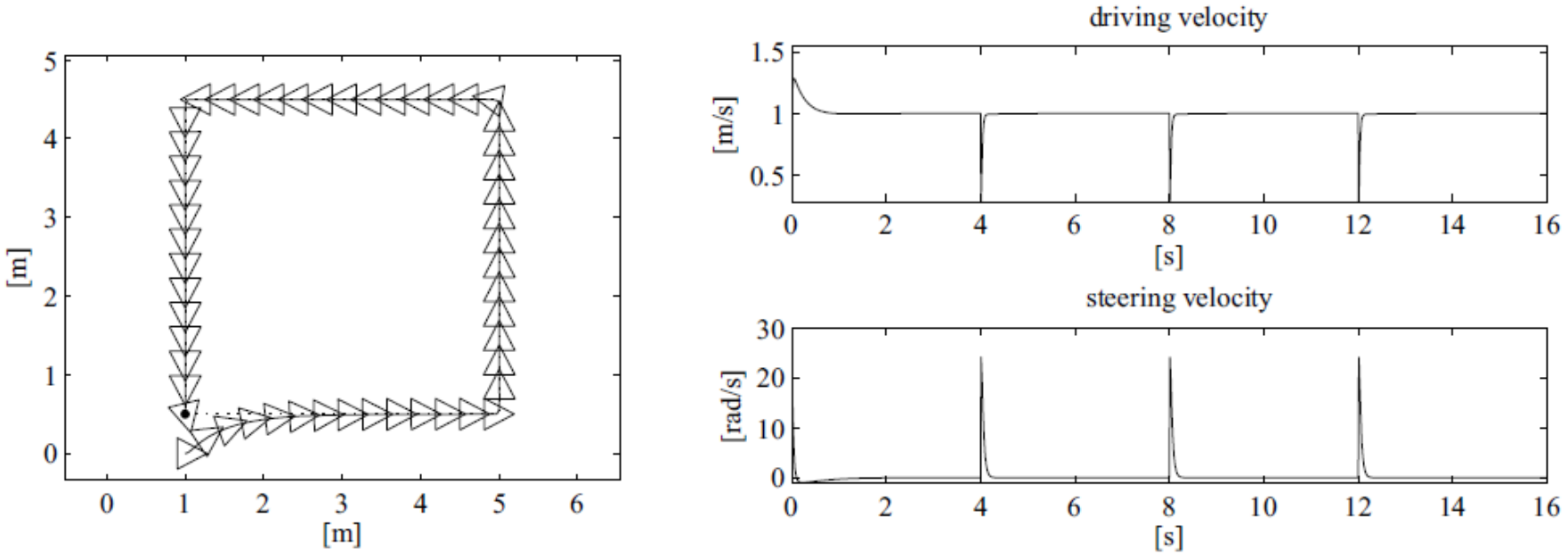


Fig. 11.17. Tracking a square reference trajectory (*dotted*) with the controller based on input/output linearization, with $b = 0.2$; *left*: Cartesian motion of the unicycle, *right*: time evolution of the velocity inputs v and ω

11.6.2 Regulation

- Design a control law that takes the vehicle to the desired state q_d withour requiring planning
- Two differnet methods
  - Cartesian regulation
  - Posture regulation

Cartesian regulation

- Drive the unicycle to a given position, without specifying the orientation
- Can be used if the robot should visit a set of view points, and has a 360 degree sensor
- Without loss og generality we assume desired Cartesian position is the origin
- The Cartesian error $\boldsymbol{e}_p$ is then $\begin{bmatrix} -x & -y \end{bmatrix}^T$

Cartesian regulation

- Consider the control law

$$v = -k_1(x\cos\theta + y\sin\theta) \tag{11.78}$$

$$\omega = k_2(\mathrm{Atan2}(y,x) - \theta + \pi), \tag{11.79}$$

- The driving velocity v is proportional to the projection of the Cartesian error $\boldsymbol{e}_p$ on the sagittal axis of the unicycle
- The steering velocity ω is proportional to the difference of the orientation of the unicycle and that of the vector $\boldsymbol{e}_p$ (pointing error)

Cartesian regulation

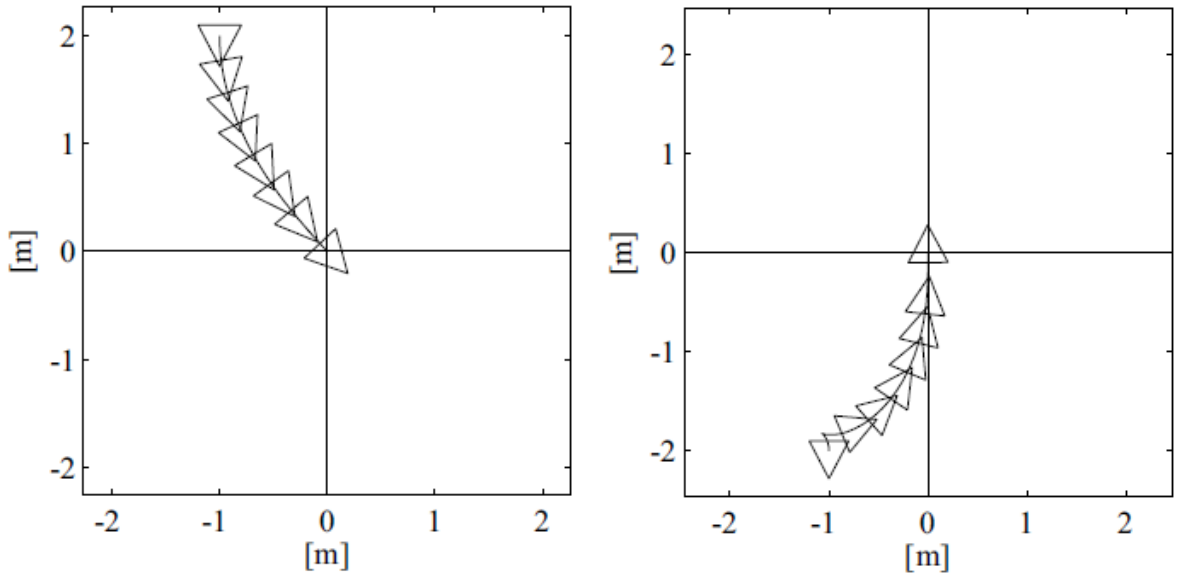
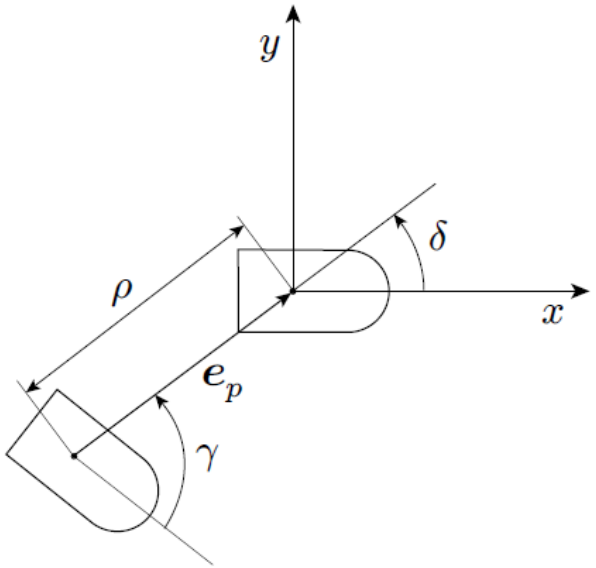


Fig. 11.19. Regulation to the origin of the Cartesian position of the unicycle with the controller (11.78), (11.79), for two different initial configurations

Posture regulation

- Driving the unicycle to a given posture (position and orientation)
- Assume without loss of generality that desired configuration is the origin $\boldsymbol{q}_d = [0 \quad 0 \quad 0]^T$



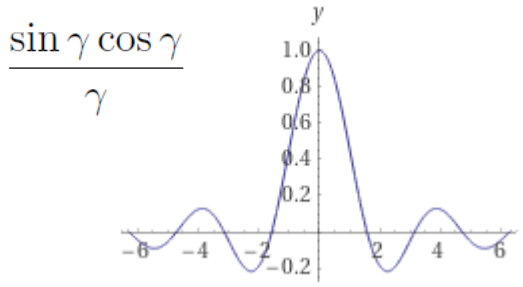
$$\rho = \sqrt{x^2 + y^2}$$
$$\gamma = \text{Atan2}(y, x) - \theta + \pi$$
$$\delta = \gamma + \theta.$$

Fig. 11.18. Definition of polar coordinates for the unicycle

Posture regulation

- Transforming the kinematic model to polar coordinates yields

$$\begin{aligned}\dot{\rho} &= -v\cos\gamma \\ \dot{\gamma} &= \frac{\sin\gamma}{\rho}v - \omega \\ \dot{\delta} &= \frac{\sin\gamma}{\rho}v.\end{aligned} \tag{11.80}$$



Posture regulation

- Consider the control law

$$v = k_1 \rho \cos\gamma \tag{11.81}$$

$$\omega = k_2 \gamma + k_1 \frac{\sin\gamma \cos\gamma}{\gamma} (\gamma + k_3 \delta), \tag{11.82}$$

- The first term drives the robot forward proportional to the distance from the goal and the orientation towards the goal
- The second term rotates the robot towards the goal, and as the robot is pointing at the goal the final orientation influences more

Posture regulation

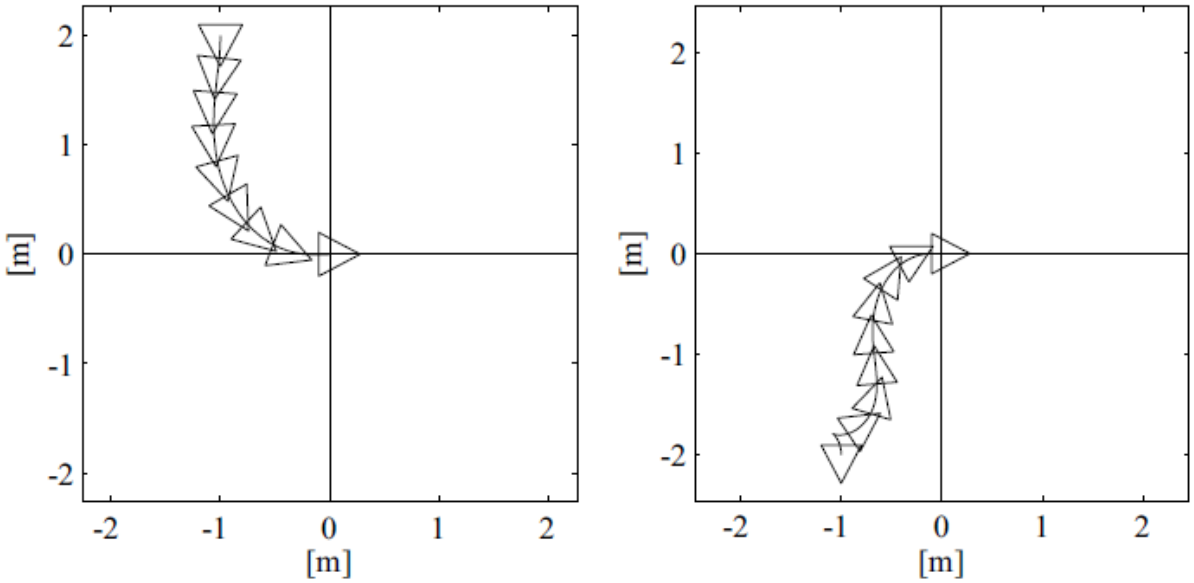


Fig. 11.20. Regulation to the origin of the posture of the unicycle with the controller (11.81), (11.82), for two different initial configurations

11.7 Odometric localization

- The implementation of any feedback controller requires the measured robot configuration
- Unlike manipulators, mobile robots does not measure their configuration variables directly
- Usually wheeled mobile robot are equipped with incremental encoders, that measure the rotation of the wheels
- Therefore it is necessary to have a localization procedure that estimates the robot configuration in real-time

Odometric localization

- Consider a unicycle where the control inputs are constant within each sampling interval
- This implies that the robot moves in an arc of a circle of radius $R = v_k/\omega_k$ (line if $\omega_k = 0$)
- Assume that $\boldsymbol{q}(t_k) = \boldsymbol{q}_k$ is known together with v_k and ω_k
- We would now like to calculate $\boldsymbol{q}_{k+1}$ based on the above values

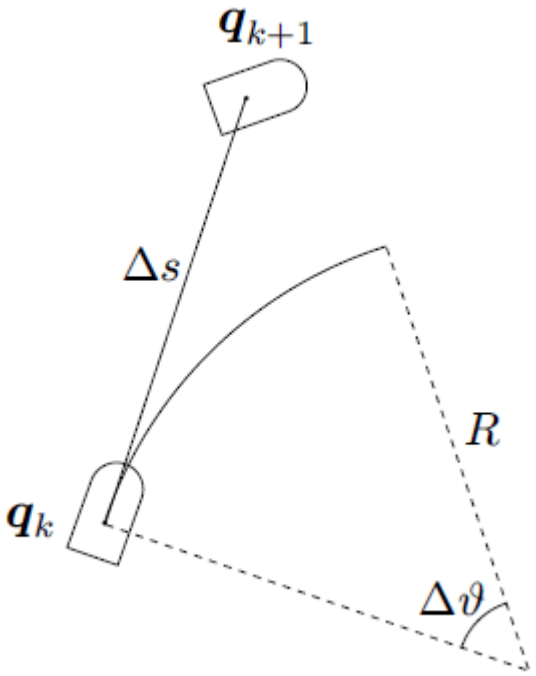
Odometric localization – Eulers method

- Forward integration of the kinematic model yields

$$\begin{aligned} x_{k+1} &= x_k + v_k T_s \cos\theta_k \\ y_{k+1} &= y_k + v_k T_s \sin\theta_k \\ \theta_{k+1} &= \theta_k + \omega_k T_s, \\ T_s &= t_{k+1} - t_k \end{aligned} \qquad (11.83)$$

- This is equivalent with using Euler's method for numerical integration
- An error is introduced in x_{k+1} and y_{k+1} as θ_k is constant throughout the interval

Odometric localization – Eulers method



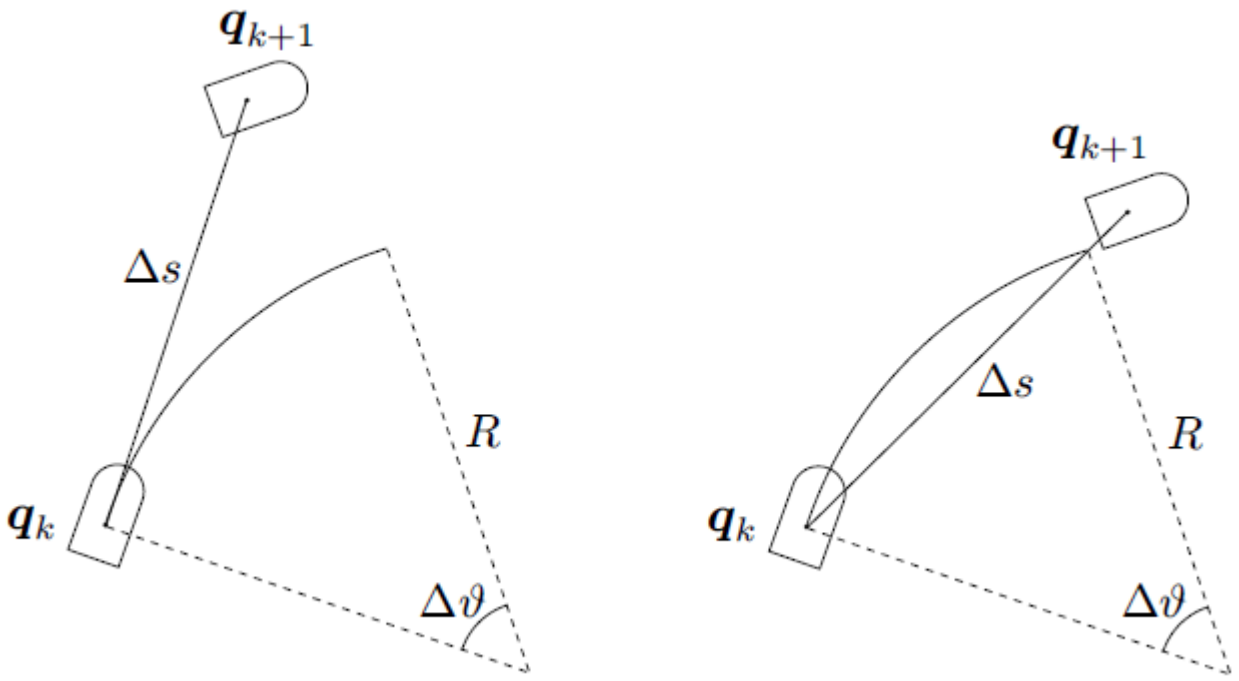
Odometric localization – Runge-Kutta

- Using the second order Runge-Kutta method one obtain

$$
\begin{aligned}
x_{k+1} &= x_k + v_k T_s \cos\left(\theta_k + \frac{\omega_k T_s}{2}\right) \\
y_{k+1} &= y_k + v_k T_s \sin\left(\theta_k + \frac{\omega_k T_s}{2}\right) \\
\theta_{k+1} &= \theta_k + \omega_k T_s,
\end{aligned}
\tag{11.84}
$$

- Now the average value of the orientation is used

Odometric localization – Runge-Kutta



Odometric localization – Exact integration

- Using geometry or the chained for one can calculate the exact development of q_{k+1}

$$
\begin{aligned}
x_{k+1} &= x_k + \frac{v_k}{\omega_k}(\sin\theta_{k+1} - \sin\theta_k) \\
y_{k+1} &= y_k - \frac{v_k}{\omega_k}(\cos\theta_{k+1} - \cos\theta_k) \qquad (11.85) \\
\theta_{k+1} &= \theta_k + \omega_k T_s.
\end{aligned}
$$

- When $\omega_k = 0$ the formula becomes the same as Euler and Runge-Kutta
- When implementing the method this must be handle as a special case

Odometric localization - Comparison

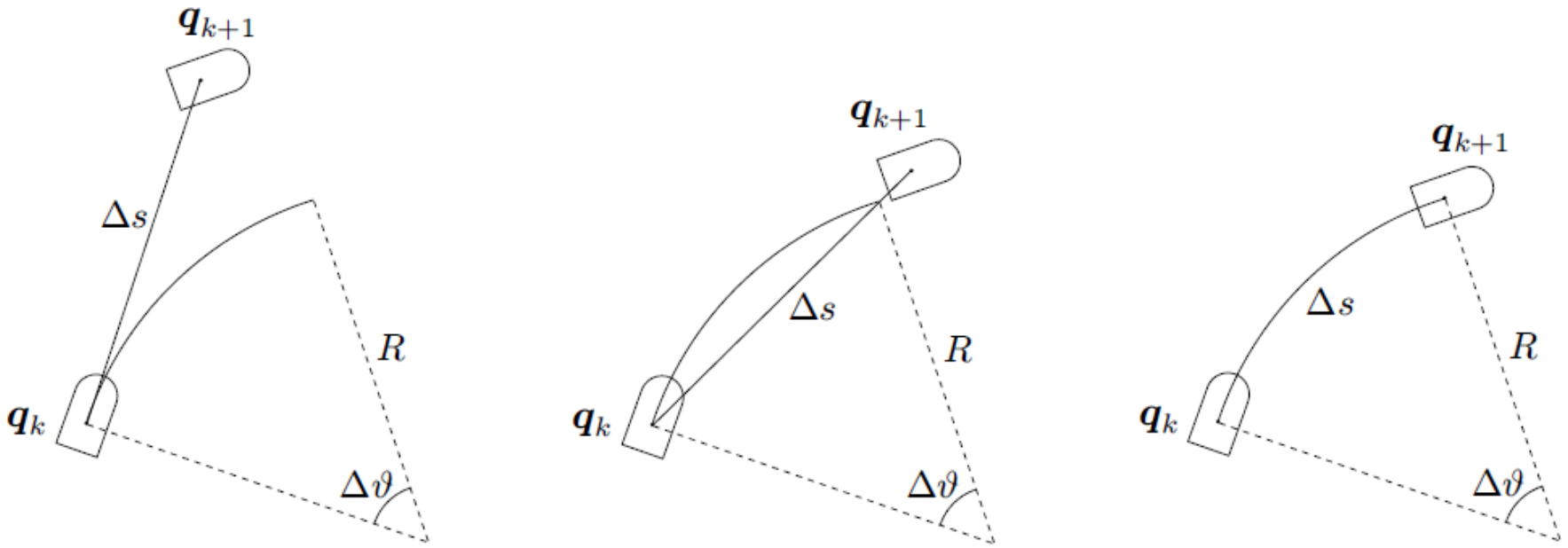


Fig. 11.21. Odometric localization for a unicycle moving along an elementary tract corresponding to an arc of circle; *left*: integration via Euler method, *centre*: integration via Runge–Kutta method, *right*: exact integration

Odometric localization

- For odometric localization it is generally better to rely on measured values rather than commanded values
- Forward integration of the kinematic model based on measurements is referred to as *odometric localization* or *dead reckoning*
- The method rely on iterative updated, therefore q_0 must be known
- Tracking accuracy will never be better than the accuracy of the initial configuration q_0

Odometric localization

- Odometric localization is subject to drift (error that grow over time)
  - Wheel slippage
  - Calibration errors of kinematic parameters
  - Numerical errors
- A better approach is to use exteroceptive sensors
- Then the robot can build up a map during motion and correct the odometric localization based on the estimated position in the map
- Simultaneous localization and mapping (SLAM)

Summary

- Planning (11.5)
  - Cartesian polynomials
  - Chained form
  - Parameterized inputs
- Motion Control (11.6)
  - Trajectory tracking
  - Regulation
- Odometric Localization (11.7)
  - Euler’s method, Runge-Kutta, exact integration