Chapter 7
Control

Part 2

7.2 State Space Control

Carnegie Mellon
THE ROBOTICS INSTITUTE

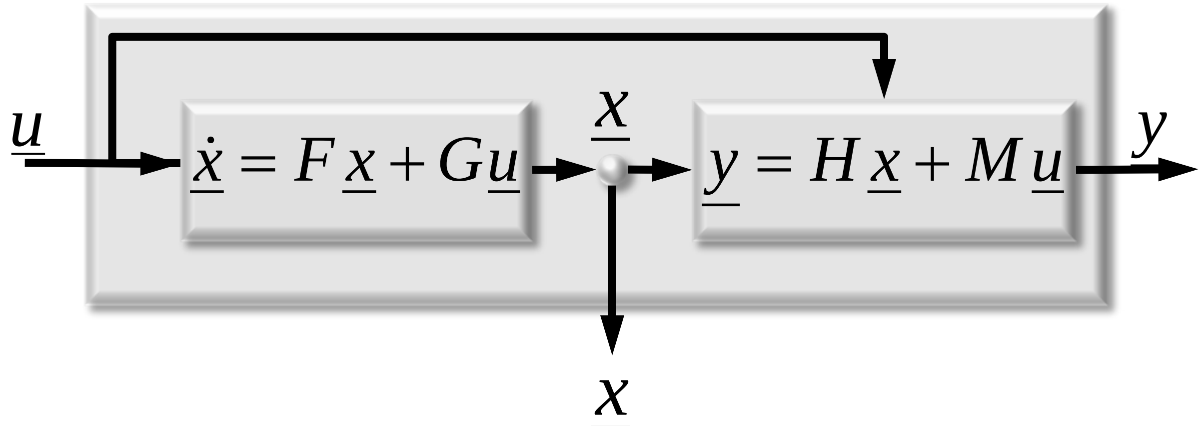
Outline

- 7.2 State Space Control
  - 7.2.1 Introduction
  - 7.2.2 State Space Feedback Control
  - 7.2.3 Example: Robot Trajectory Following
  - 7.2.4 Perception Based Control
  - 7.2.5 Steering Trajectory Generation
  - Summary

Outline

- 7.2 State Space Control
  - <u>7.2.1 Introduction</u>
  - 7.2.2 State Space Feedback Control
  - 7.2.3 Example: Robot Trajectory Following
  - 7.2.4 Perception Based Control
  - 7.2.5 Steering Trajectory Generation
  - Summary

7.2 State Space Control



- Looks deeper at system behavior by exposing the states
- Tries to control the entire state vector

7.2.1 Introduction
(State Space Model)

- Recall the linear TI case:

$$\dot{\underline{x}}(t) = F(t)\underline{x}(t) + G(t)\underline{u}(t)$$

$$\underline{y}(t) = H(t)\underline{x}(t) + M(t)\underline{u}(t)$$

- x is n X 1
- u is r X 1
- y is m X 1

7.2.1.1 Controllability

- Completely controllable if there is a u(t) that drives the system:
  - From any x(t1)
  - To any x(t2)
  - In finite time Dt = t2-t1.
- Totally controllable if Dt can be made as small as desired.
- Some use the word reachability for this concept.

7.2.1.1 Controllability

(Controllability Condition)

- Totally controllable iff this n X nm matrix:

$$Q = [G|FG|FFG|\dots F^{n-1}G]$$

- … is of full rank.
- If F(t) and G(t) depend on time, Q can only lose rank at isolated points in time.

7.2.1.2 Observability

- Completely observable if $\underline{x}(t_1)$ is fully determined by knowledge of
  - $\underline{u}(t)$ and
  - $\underline{y}(t)$
  - on an interval $[t_1, t_2]$ where $t_2 > t_1$:
  - In finite time $\Delta t = t_2 - t_1$.
- Totally observable if $\Delta t = t_2 - t_1$ can be made as small as desired.

7.2.1.2 Observability
(Observability Condition)

- Totally observable iff this mn X n matrix:

$$P = \begin{bmatrix} H \\ HF \\ HFF \\ \cdots \\ HF^{n-1} \end{bmatrix}$$

- … is of full rank.
- If F(t) and G(t) depend on time, P can only lose rank at isolated points in time.

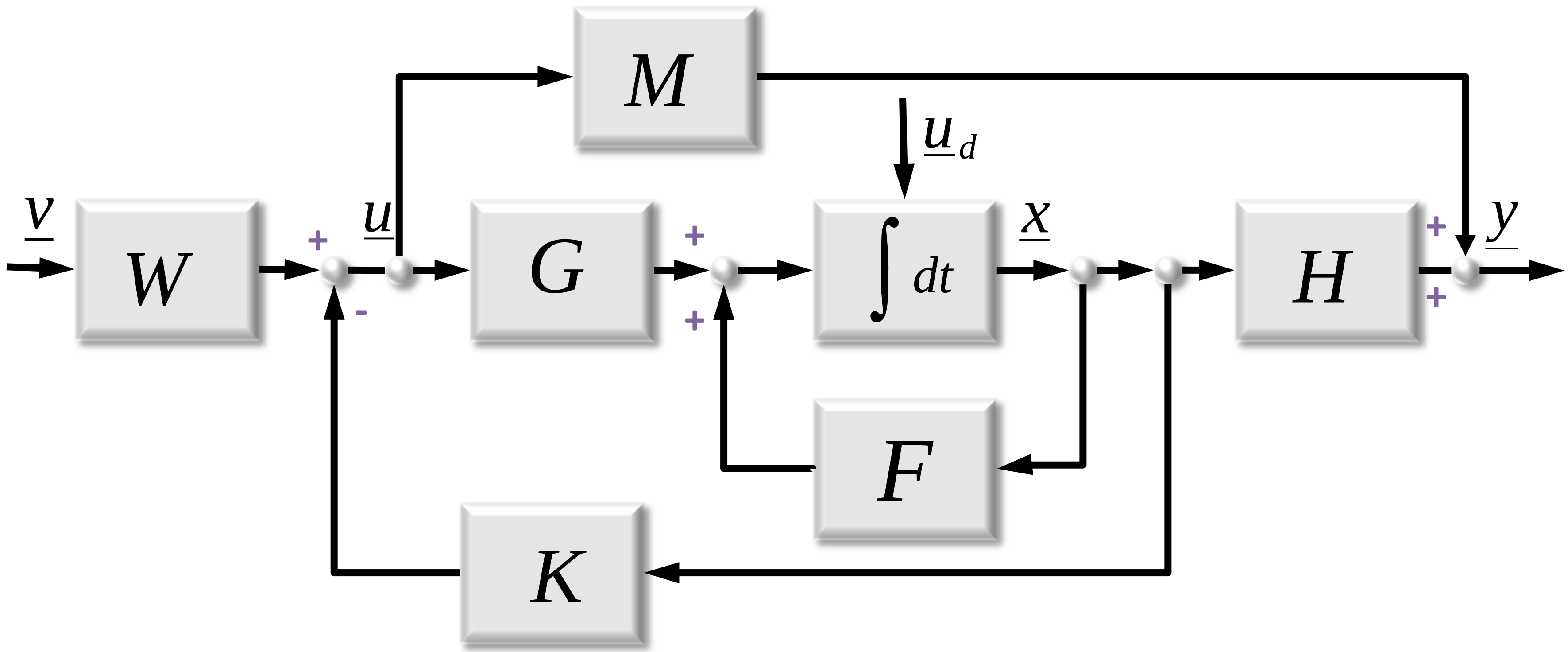
Outline

- 7.2 State Space Control
  - 7.2.1 Introduction
  - 7.2.2 State Space Feedback Control
  - 7.2.3 Example: Robot Trajectory Following
  - 7.2.4 Perception Based Control
  - 7.2.5 Steering Trajectory Generation
  - Summary

Feedback Control in State Space

- Two options: state and output feedback.
- Only the second seems relevant (since only y is accessible by definition)
- However:
  - Full state feedback is theoretically relevant
  - Sometimes you can measure all states
  - Often, you can reconstruct the states.

7.2.2.1 State Feedback

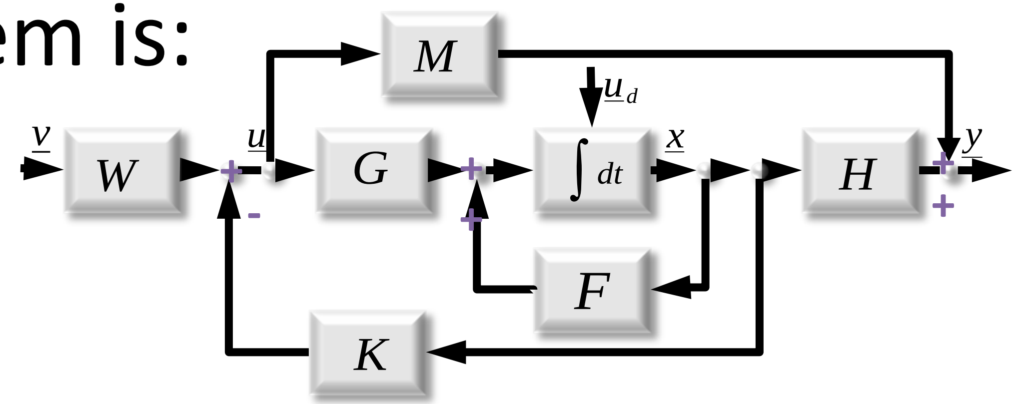


$$\underline{u}(t) = W\underline{v}(t) - K\underline{x}(t)$$

7.2.2.1 State Feedback

- Upon substitution,
- the new linear system is:

$$\underline{\dot{x}} = [F - GK]\underline{x} + GW\underline{v}$$
$$\underline{y} = [H - MK]\underline{x} + MW\underline{v}$$



- Can show:
  - Controllability is unaltered if W is full rank
  - Observability can be altered or even lost (i.e if H=MK).

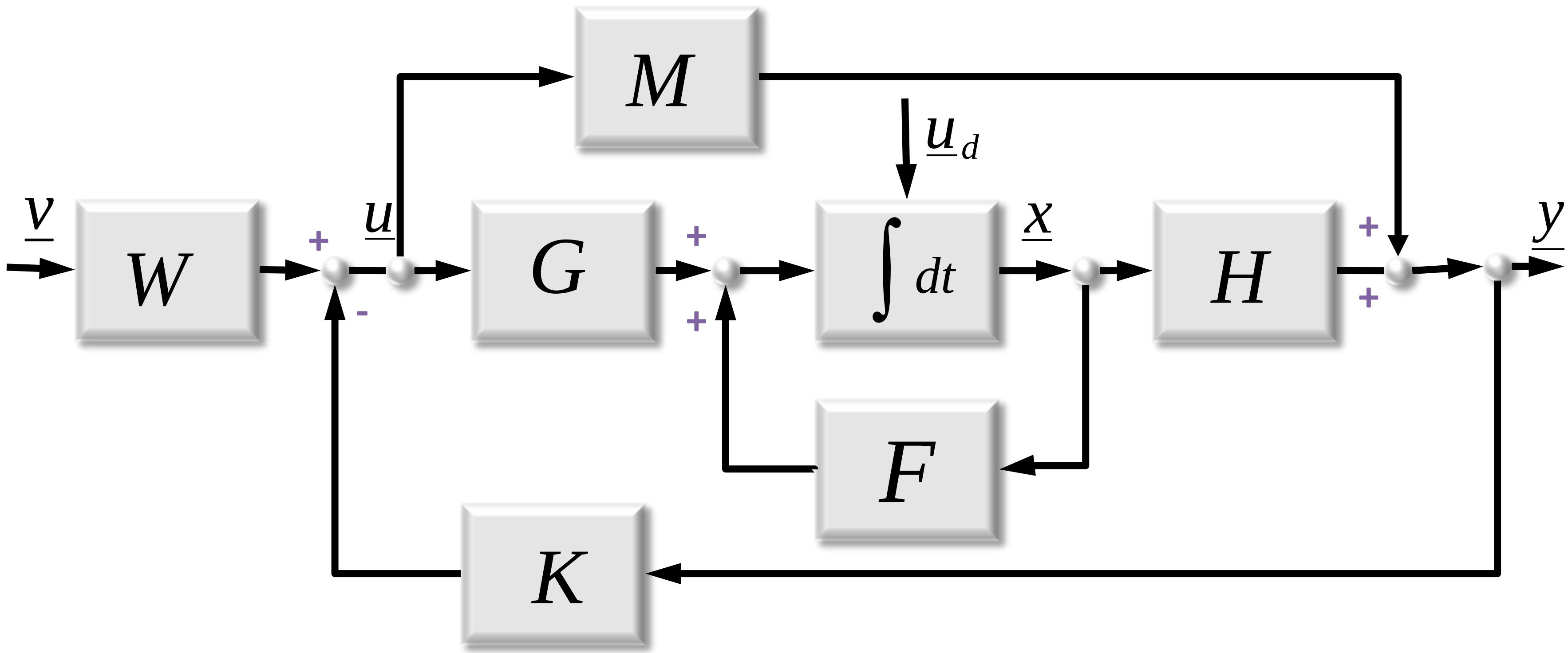
7.2.2.2 Eigenvalue Assignment

- Equivalent to pole placement.
- Consider time invariant case. Stability depends on eigenvalues of new dynamics matrix F-GK:

$$det(\lambda I - F + GK) = 0$$

- If original system is controllable, these e-values can be placed arbitrarily by suitable choice of the gain matrix K.

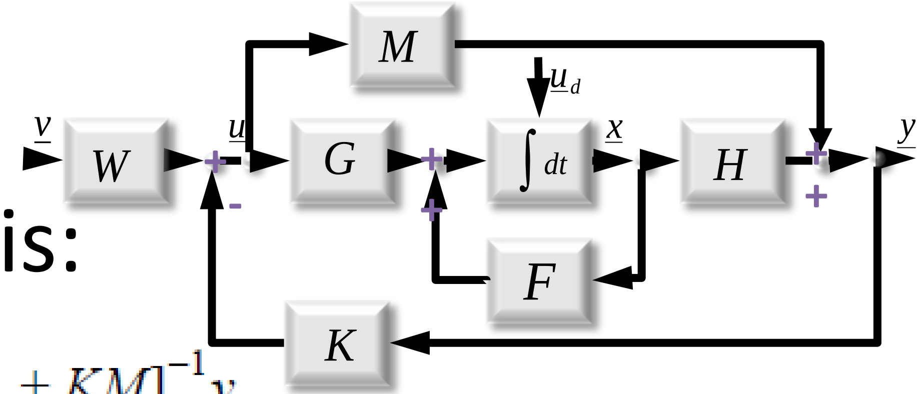
7.2.2.3 Output Feedback



$$\underline{u}(t) = W\underline{v}(t) - K\underline{y}(t)$$

Carnegie Mellon
THE ROBOTICS INSTITUTE

7.2.2.3 Output Feedback



- Upon substitution,
- the new linear system is:

$$\dot{\underline{x}} = [F - GK[I_m + MK]^{-1}H]\underline{x} + GW[I_r + KM]^{-1}\underline{v}$$

$$\underline{y} = [I_m + MK]^{-1}[H\underline{x} + MW\underline{v}]$$

- Can show:
  - Controllability is unaltered if W and [Ir + KM]-1 are of full rank
  - Observability is preserved.

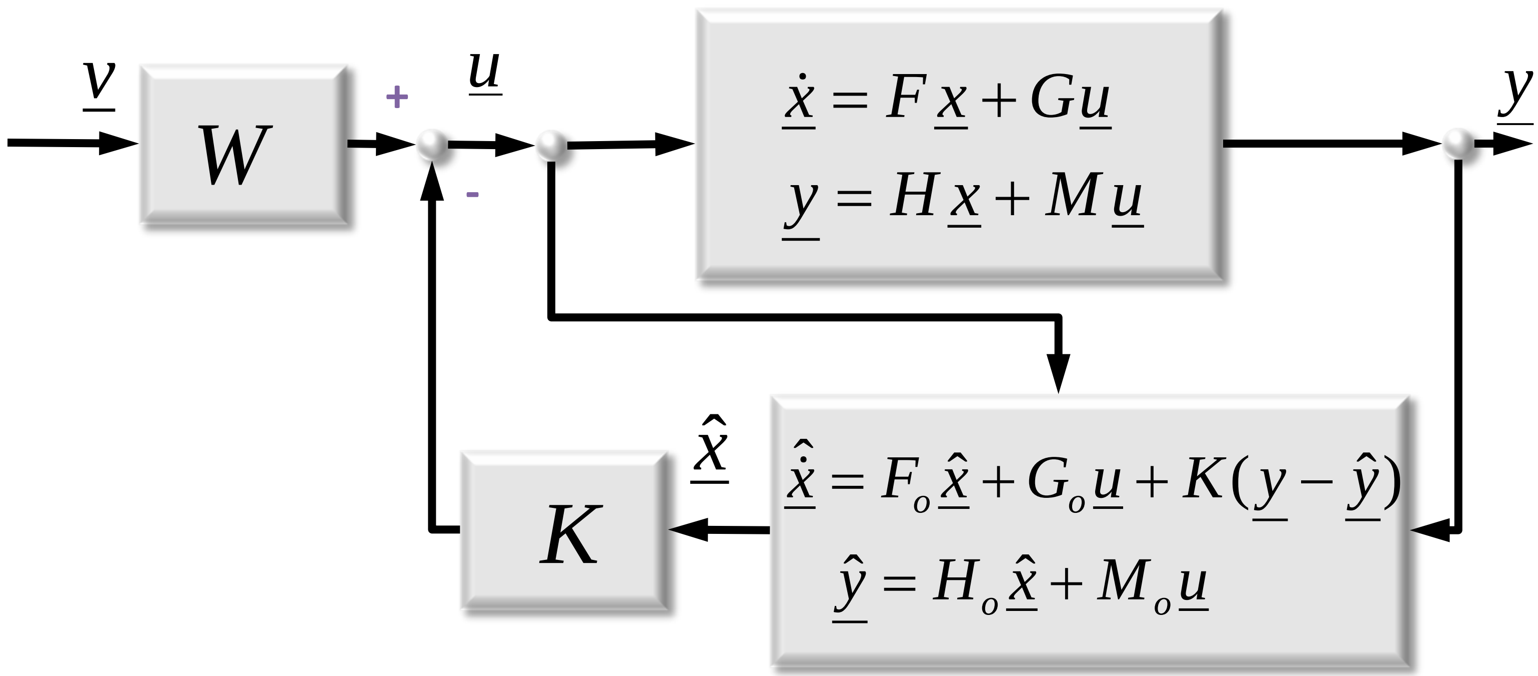
7.2.2.4 Eigenvalue Assignment

- If original system …
  - is completely controllable
  - and H is full rank
- m of these e-values can be placed arbitrarily by suitable choice of the gain matrix K.

7.2.2.5 Observers

- Clearly, output feedback is inferior.
- Can we:
  - reconstruct the state from the measurements?
  - Use the reconstructed state as state feedback.
- Surprisingly, yes.

7.2.2.5 Observers



- Almost every mobile robot looks like this.
- The observer is the state estimation system.
  - Hence, the Kalman filter.

7.2.2.5 Observers

- The predicted output is:

$$\hat{\underline{y}} = H_o \hat{\underline{x}} + M_o \underline{u}$$

- The observer dynamics have one extra input – the output prediction error:

$$\frac{d}{dt}\hat{\underline{x}} = F_o \hat{\underline{x}} + G_o \underline{u} + K_o(\underline{y} - \hat{\underline{y}})$$

Ko is TBD

- Subtract this from real system dynamics (assuming matrices match)

$$\frac{d}{dt}(\underline{x} - \hat{\underline{x}}) = F(\underline{x} - \hat{\underline{x}}) - K_o(\underline{y} - \hat{\underline{y}})$$

Same form as a controller!

7.2.2.5 Observers

- If the error dynamics are controllable, we can drive the state estimate to agree with the state in an arbitrarily short period of time.

$$\frac{d}{dt}(\underline{x} - \hat{\underline{x}}) = F(\underline{x} - \hat{\underline{x}}) - K_o(\underline{y} - \hat{\underline{y}})$$

Same form as a controller!

  – Assuming measurements and matrices are known perfectly.

7.2.2.6 Control of Nonlinear Systems

- All mobile robots are nonlinear
  - Because the sensors and actuators are on the robot.
- So…. Why study all this linear system stuff?

7.2.2.6 Control of Nonlinear Systems

- All mobile robots are nonlinear
  - Because the sensors and actuators are on the robot.
- So…. Why study all this linear system stuff?
  - Control the linearized (error) dynamics.

7.2.2.6 Control of Nonlinear Systems
(Two DOF Design)

- Aka Feedforward with feedback trim….
  - Feed forward the reference trajectory open loop control.
  - Use feedback to reject disturbances etc.

$$
\begin{aligned}
\dot{\underline{x}}(t) &= \underline{f}(\underline{x}(t), \underline{u}(t)) \\
\underline{y}(t) &= \underline{h}(\underline{x}(t), \underline{u}(t))
\end{aligned}
$$

Nonlinear System

$$
\begin{aligned}
\delta\dot{\underline{x}}(t) &= F(t)\delta\underline{x}(t) + G(t)\delta\underline{u}(t) \\
\delta\underline{y}(t) &= H(t)\delta\underline{x}(t) + M(t)\delta\underline{u}(t)
\end{aligned}
$$

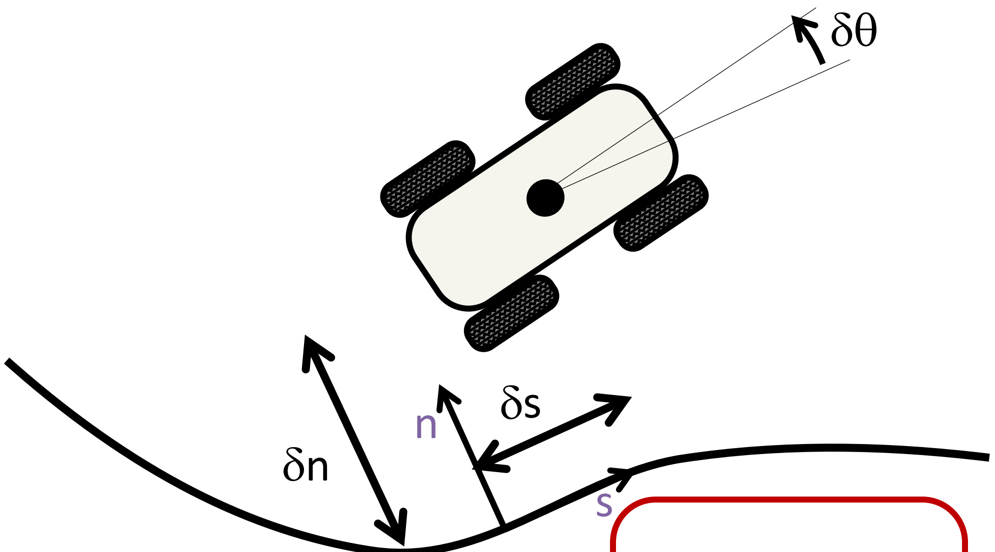
Linearized Error Dynamics

Outline

- 7.2 State Space Control
  - 7.2.1 Introduction
  - 7.2.2 State Space Feedback Control
  - <u>7.2.3 Example: Robot Trajectory Following</u>
  - 7.2.4 Perception Based Control
  - 7.2.5 Steering Trajectory Generation
  - Summary

7.2.3.1 Representing Trajectories

- Assume velocity is fwd along body x only
- States: $\underline{x} = \begin{bmatrix} x & y & \theta \end{bmatrix}^T$
- Inputs: $\underline{u} = \begin{bmatrix} \kappa & V \end{bmatrix}^T$
- Nonlinear state space model in terms of time.

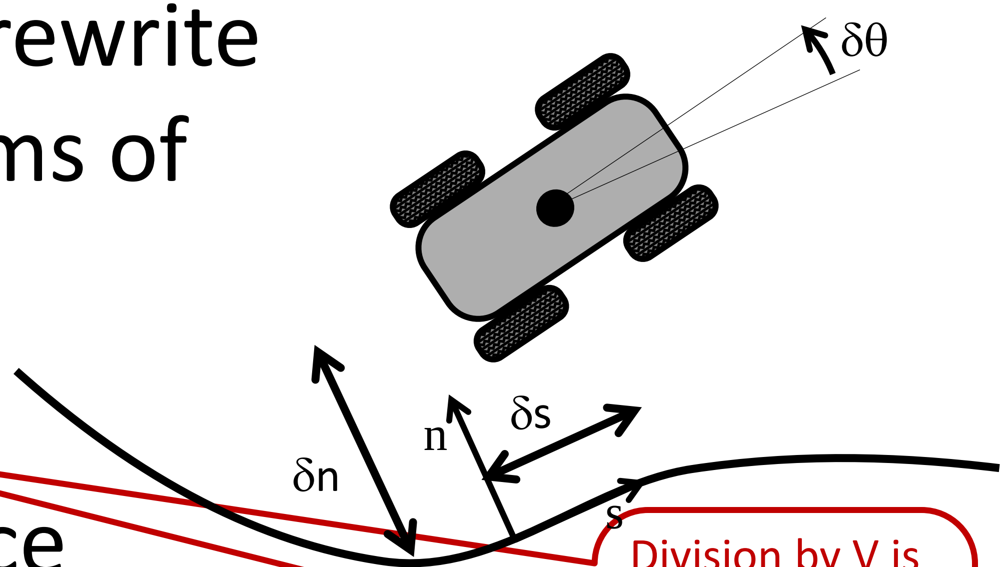


But notice speed V can be factored out.

$$\frac{d}{dt}\begin{bmatrix} x \\ y \\ \psi \end{bmatrix} = \begin{bmatrix} \cos\psi \\ \sin\psi \\ \kappa \end{bmatrix} v \qquad \begin{bmatrix} x(t) \\ y(t) \\ \psi(t) \end{bmatrix} = \begin{bmatrix} x(0) \\ y(0) \\ \psi(0) \end{bmatrix} + \int_0^t \begin{bmatrix} \cos\psi \\ \sin\psi \\ \kappa \end{bmatrix} V dt$$

7.2.3.1 Representing Trajectories

- This means we can rewrite the dynamics in terms of distance



$$\frac{dx}{dt} / V = \frac{dx}{dt} / \frac{ds}{dt} = \frac{dx}{ds}$$

- Nonlinear state space model in terms of distance.

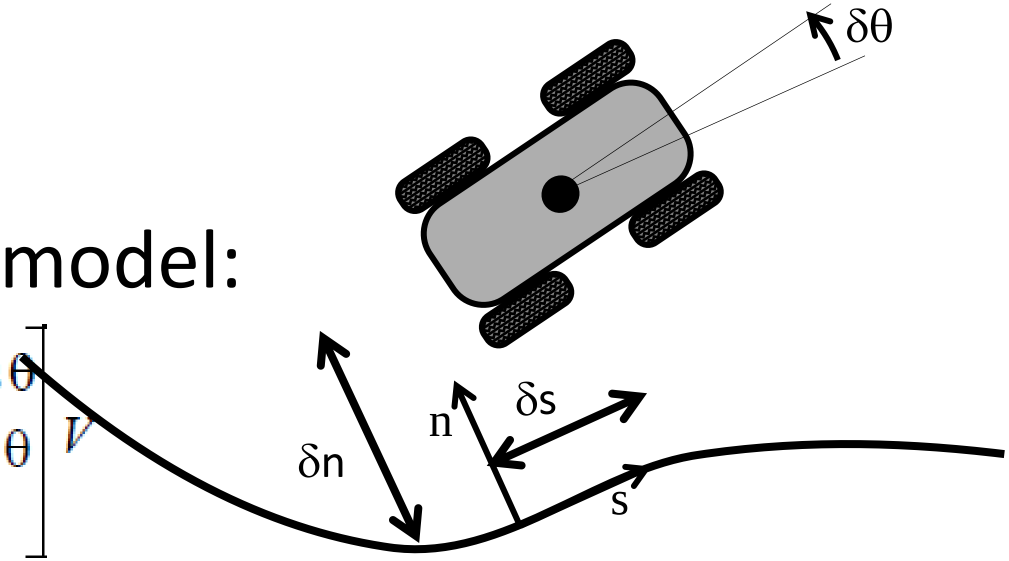
$$\frac{d}{ds}\begin{bmatrix} x \\ y \\ \psi \end{bmatrix} = \begin{bmatrix} \cos\psi \\ \sin\psi \\ \kappa \end{bmatrix} \qquad \begin{bmatrix} x(s) \\ y(s) \\ \psi(s) \end{bmatrix} = \begin{bmatrix} x(0) \\ y(0) \\ \psi(0) \end{bmatrix} + \int_0^t \begin{bmatrix} \cos\psi \\ \sin\psi \\ \kappa \end{bmatrix} ds$$

Division by V is only a problem if you insist on a curvature for every time.

7.2.3.2 Robot Trajectory Following

- States: $\underline{x} = \begin{bmatrix} x & y & \theta \end{bmatrix}^T$
- Inputs: $\underline{u} = \begin{bmatrix} \kappa & V \end{bmatrix}^T$
- Nonlinear state space model:

$$\frac{d}{dt}\begin{bmatrix} x \\ y \\ \theta \end{bmatrix} = \begin{bmatrix} \cos\theta \\ \sin\theta \\ \kappa \end{bmatrix} V$$



- Suppose a trajectory generator has produced a reference:

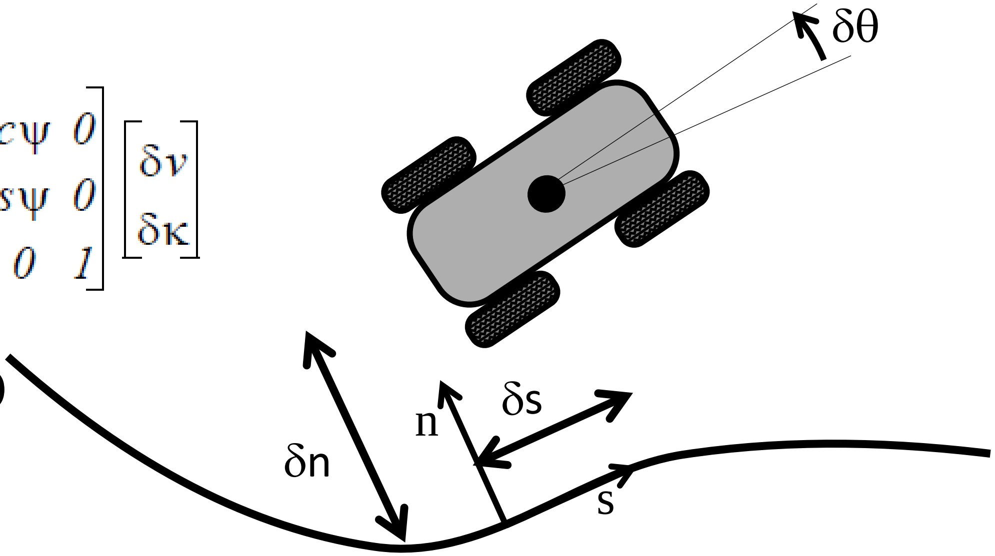
$$[\underline{u}_r(t), \underline{x}_r(t)]$$

- Assume full state feedback.

7.2.3.2.1 Linearization and Controllability

- Linearized Dynamics:

$$\frac{d}{dt}\begin{bmatrix}\delta x\\ \delta y\\ \delta\psi\end{bmatrix} = \begin{bmatrix}0 & 0 & -vs\psi\\ 0 & 0 & vc\psi\\ 0 & 0 & 0\end{bmatrix}\begin{bmatrix}\delta x\\ \delta y\\ \delta\psi\end{bmatrix} + \begin{bmatrix}c\psi & 0\\ s\psi & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}\delta v\\ \delta\kappa\end{bmatrix}$$



- Convert coordinates to path tangent frame.

$$\begin{bmatrix}\delta\dot{s}\\ \delta\dot{n}\\ \delta\dot{\theta}\end{bmatrix} = \begin{bmatrix}0 & 0 & 0\\ 0 & 0 & V\\ 0 & 0 & 0\end{bmatrix}\begin{bmatrix}\delta s\\ \delta n\\ \delta\theta\end{bmatrix} + \begin{bmatrix}1 & 0\\ 0 & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}\delta V\\ \delta\kappa\end{bmatrix}$$

Very Simple System

Linear

$$\underline{s} = \begin{bmatrix}s & n & \theta\end{bmatrix}^T$$

Time Invariant for constant V

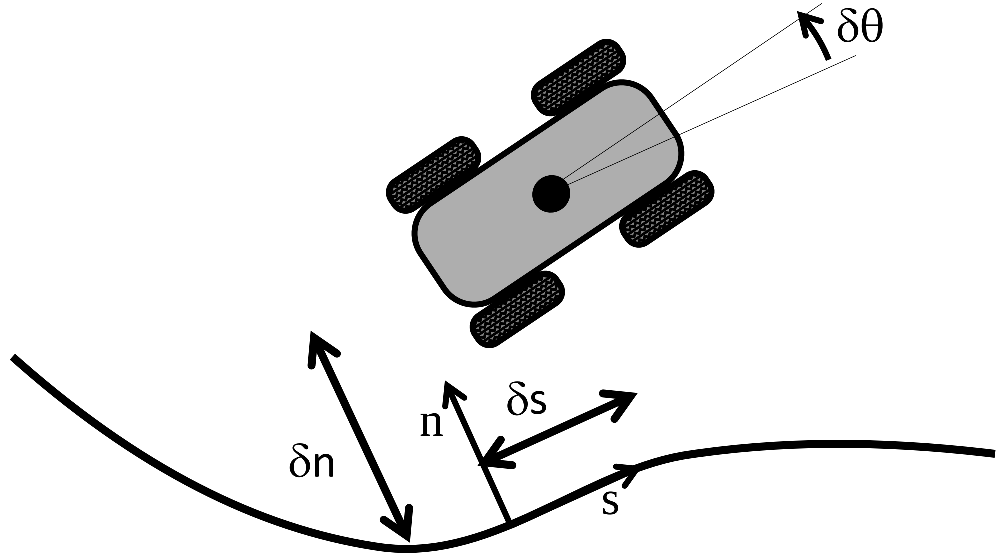
- Of the form:

$$\delta\dot{\underline{s}}(t) = F(t)\delta\underline{s}(t) + G(t)\delta\underline{u}(t)$$

7.2.3.2.1 Linearization and Controllability

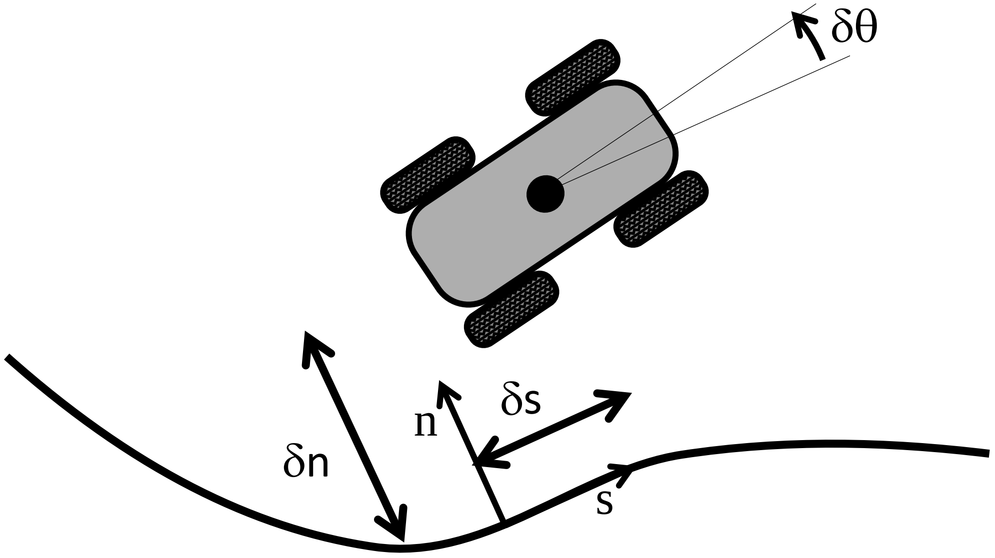
- Controllable?

$$Q = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & V & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$



7.2.3.2.1 Linearization and Controllability

- Controllable?



$$Q = \begin{bmatrix} \boxed{1} & \boxed{0} & 0 & \boxed{0} & 0 & 0 \\ \boxed{0} & \boxed{0} & 0 & \boxed{V} & 0 & 0 \\ \boxed{0} & \boxed{1} & 0 & \boxed{0} & 0 & 0 \end{bmatrix}$$

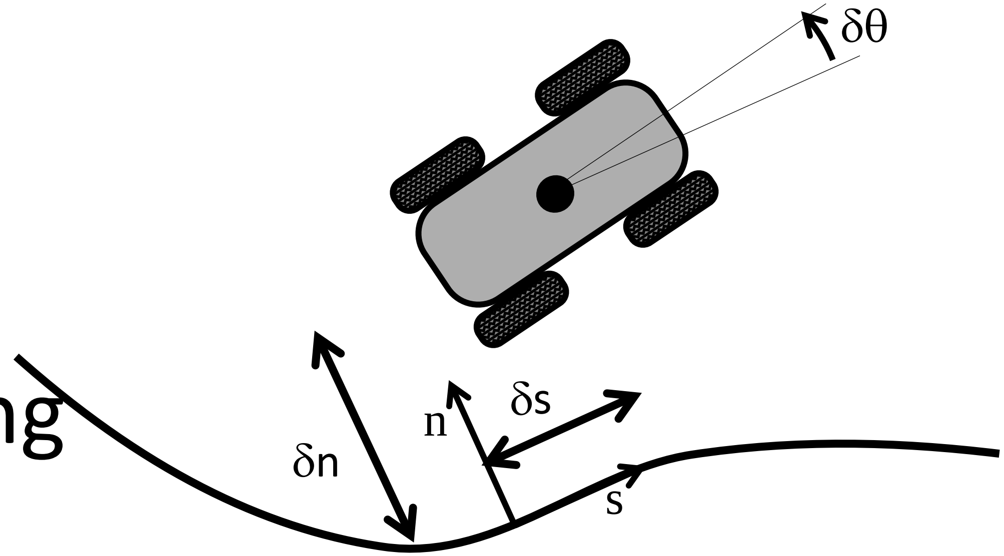
7.2.3.2.2 State Feedback Control Law

- State Feedback

$$\delta \underline{u}(t) = -K \delta \underline{s}(t)$$

- We know speed can control s and steering can control n and θ so:

$$K = \begin{bmatrix} k_s & 0 & 0 \\ 0 & k_n & k_\theta \end{bmatrix}$$



7.2.3.2.2 State Feedback Control Law

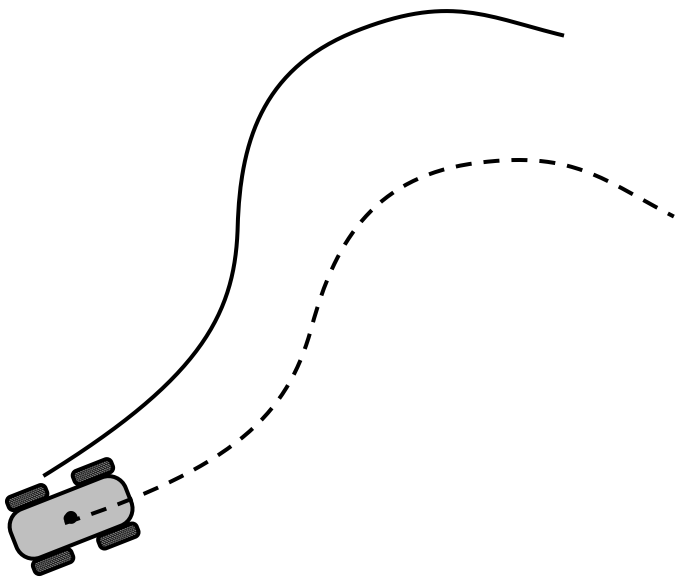
(Total Control Law)

$$\underline{u}(t) = \underline{u}_r(t) + \delta\underline{u}(t) = \underline{u}_r(t) - \begin{bmatrix} k_s & 0 & 0 \\ 0 & k_n & k_\psi \end{bmatrix} \begin{bmatrix} \delta s \\ \delta n \\ \delta\psi \end{bmatrix}$$

Feedforward (path curvature and speed)

Feedback P and PD controllers

7.2.3.2.3 Behavior
(Open Loop Servo Execution)



$$\kappa_d(s) = \kappa(s_{measured})$$

- While simple in principle, open loop execution does not reject disturbances.
- Even a single initial error can grow forever if not compensated.

7.2.3.2.3 Behavior

(Distance Based Open Loop Control)

- Ignore speed by converting to distance as the independent variable.
- Recall, this implies a particular path through space because:

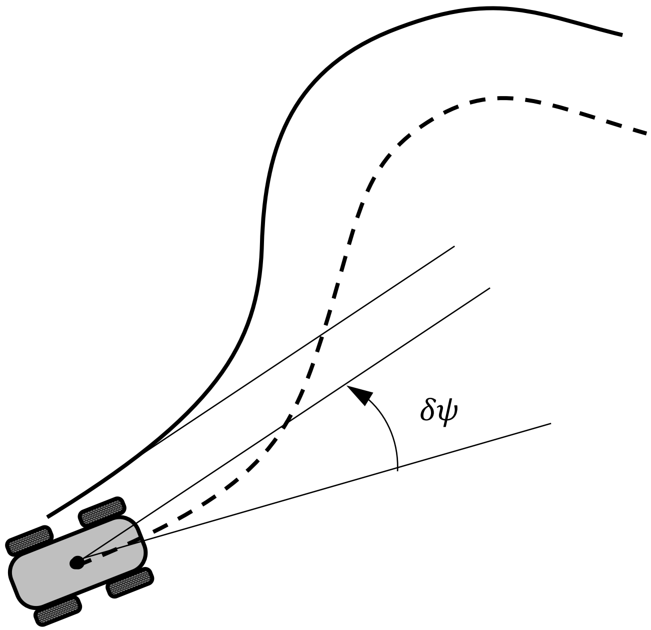
$$\psi(s) = \psi_0 + \int_0^s \kappa \, ds$$

$$x(s) = \int_0^s \cos[\theta(s)] ds$$

$$y(s) = \int_0^s \sin[\theta(s)] ds$$

7.2.3.2.3 Behavior

(Heading Error Compensation)



$$\Delta\kappa = \Delta\psi / L$$

$$\kappa_d(s) = \kappa(s_{measured}) + \Delta\kappa$$

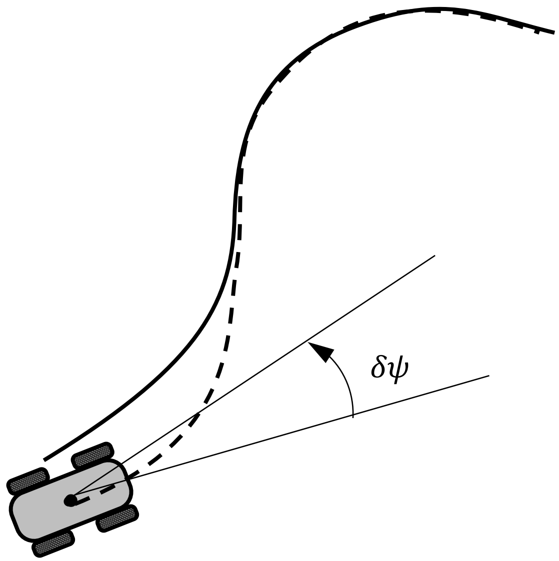
Pass original command to output

Plus a correction

- Passes original command directly to output.
- Bends the response path to be parallel to the desired.
- BUT: Does not move paths together.

7.2.3.2.3 Behavior

(Full Pose Error Compensation)



$$\Delta\kappa = \Delta\psi/L$$

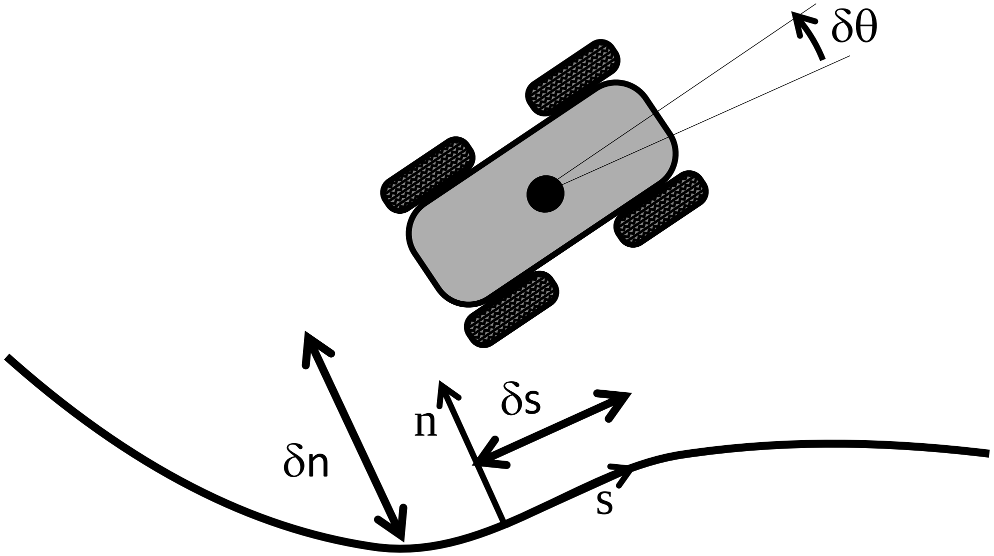
$$\Delta\kappa = 2\Delta x/L^2$$

- Passes original command directly to output.
- Δx is coordinate of closest point in body coords.
- Also adds two corrective amounts intended to remove present error.

7.2.3.2.4 Eigenvalue Placement

- New closed loop dynamics matrix:

$$F - GK = -\begin{bmatrix} k_s & 0 & 0 \\ 0 & 0 & -v \\ 0 & k_n & k_\psi \end{bmatrix}$$



- Characteristic poly:

$$det(\lambda I - F + GK) = \begin{vmatrix} \lambda + k_s & 0 & 0 \\ 0 & \lambda & -v \\ 0 & k_n & \lambda + k_\psi \end{vmatrix}$$

$$\boxed{(\lambda^3 + \lambda^2(k_\psi + k_s) + \lambda(k_n v + k_s k_\psi) + k_s k_n v)}$$

Any coefficients are possible so…..
Any roots are possible.

7.2.3.2.5 Gains

- Both curvature gains can be related to a characteristic length.

$$k_n = \frac{2}{L^2} \qquad k_\psi = \frac{1}{L}$$

- Then $\delta\kappa_\psi = \delta\psi / L$ removes heading error after moving a distance L.
- And $\delta\kappa_n = 2\delta n / L^2$ removes crosstrack error after travelling a distance L.
- Also $\tau_s = 1 / k_s$ is the time constant of speed error response.

Outline

- 7.2 State Space Control
  - 7.2.1 Introduction
  - 7.2.2 State Space Feedback Control
  - 7.2.3 Example: Robot Trajectory Following
  - <u>7.2.4 Perception Based Control</u>
  - 7.2.5 Steering Trajectory Generation
  - Summary

7.2.4 Perception Based Control

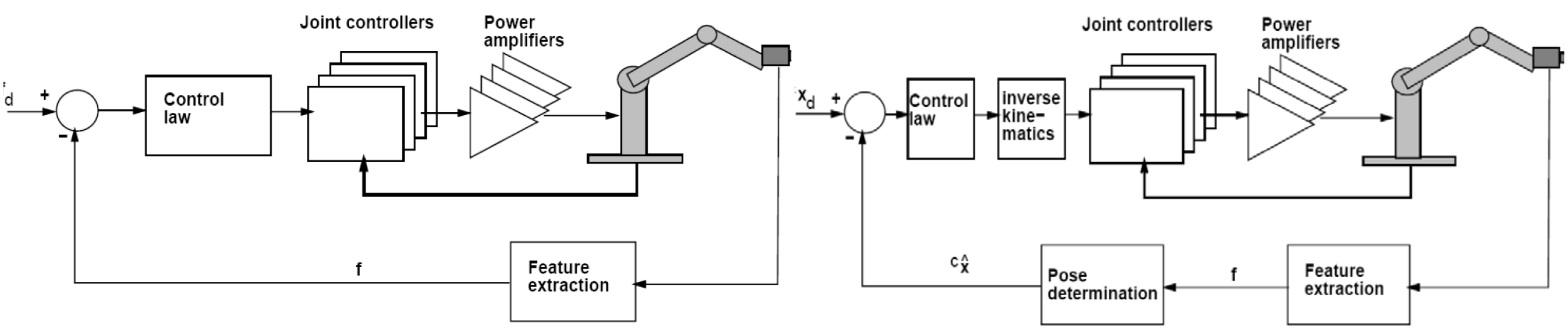
(Visual Servoing)

- Observed feature residuals can be generated by:
  - <u>Perceived errors</u> in pose estimates in a region of overlap (registration) or ...
  - <u>Real errors</u> in pose itself in a positioning task.
- In the latter case, it is natural to close a servo loop and drive the system to move to reduce the error. This is <u>visual servoing</u>.
- Must maintain feature correspondences during motion:
  - Embedded feature tracking problem.

7.2.4 Perception Based Control

(Visual Servoing : Architecture : Errors)

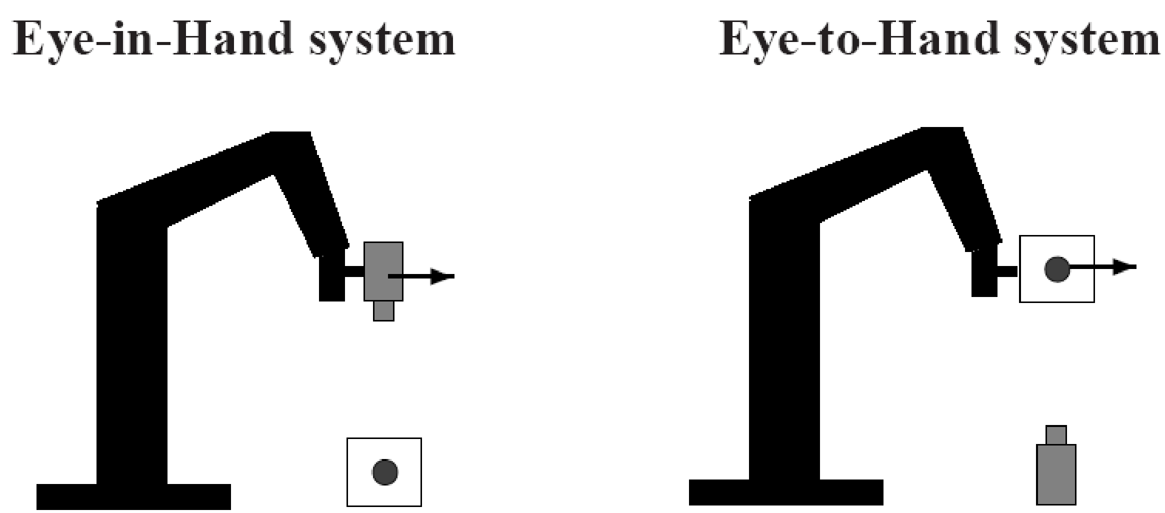
- Image-based control forms errors in image space.
  - Servoing done in image space
- Position-based control forms errors from object poses:
  - Poses derived from image features
  - Servoing done in pose space.



7.2.4 Perception Based Control

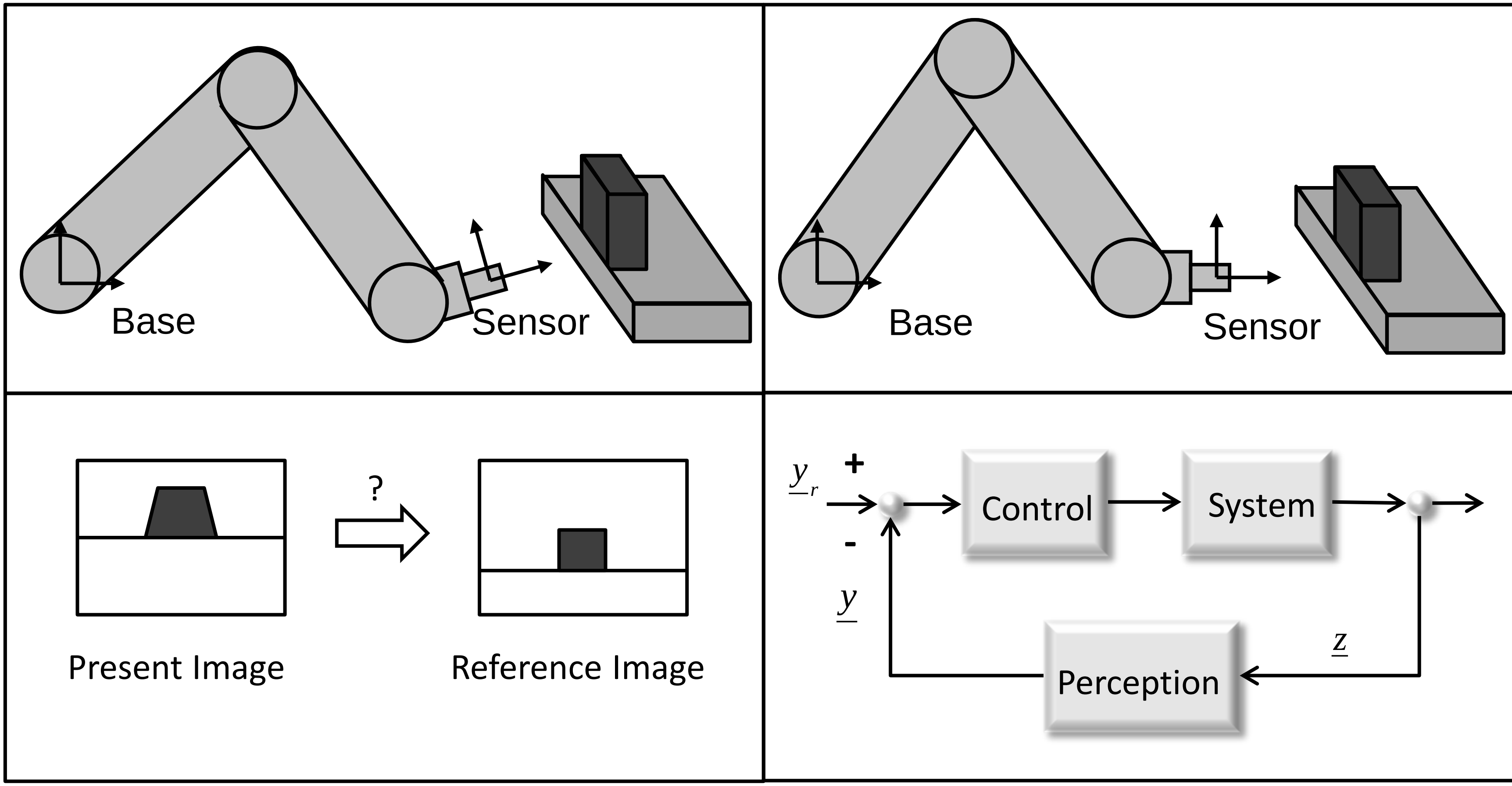
(Visual Servoing : Architecture : Camera Position)

- Camera may be moving or stationary.
- Required motions are reversed with respect to each other.

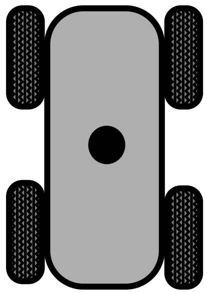


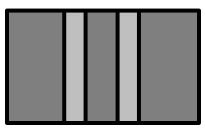
7.2.4.1 Error Coordinates

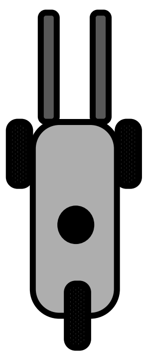
(Image Based Visual Servoing)

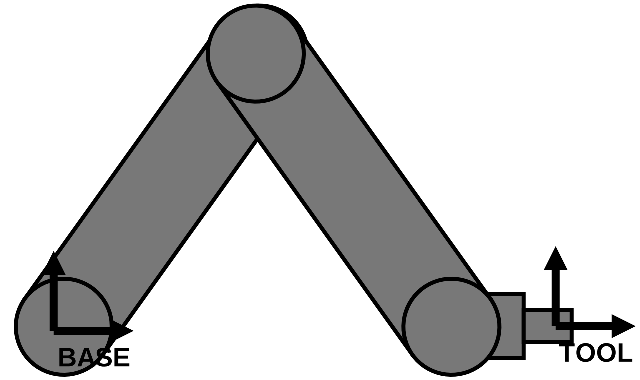


- Problem: drive the system (usually with a camera attached) to turn the present image into the desired image.
- An excellent way to drive up to something with a poor pose estimate.

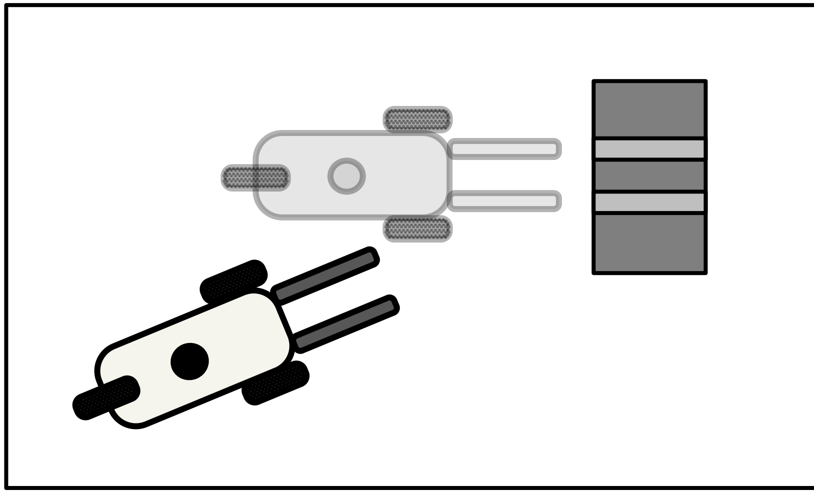


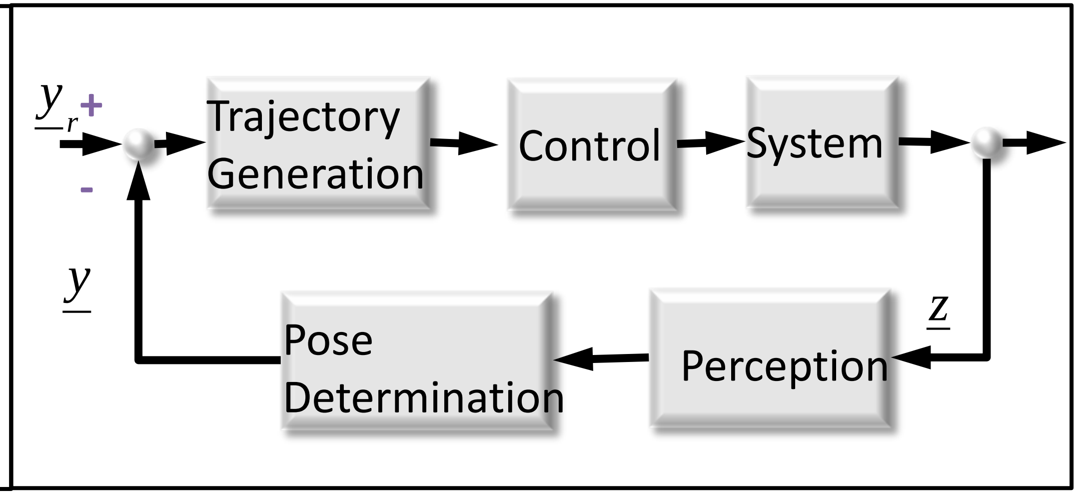





BASE
TOOL

7.2.4.1 Error Coordinates (Image Based Visual Servoing)





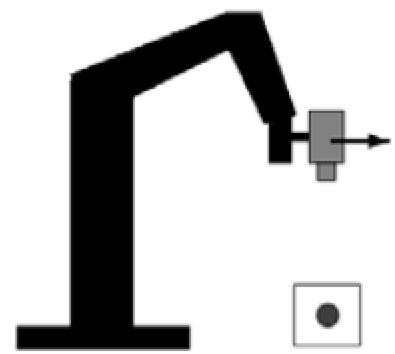
- Explicitly calculate the pose of the object relative to the camera.
- Compute the error in the pose.

7.2.4.2 Visual Servoing

(Image Formation Model)

- Observations / measurements z depend on the pose r of the camera w.r.t. the object.

$$\underline{z} = h(\underline{\rho}(t))$$



7.2.4.2 Visual Servoing
(Image Formation Model)

- k adequate features in $\underline{z}$ with which to control the m dof of the system.
- Usually:

$$k \geq m \qquad m \leq 6$$

- A goal image space configuration zr:
- Regulate the error: $\underline{z}_r - \underline{z}$:
- Typically track features at near video rate.

7.2.4.4 Controller Design
(Basic Controller)

- Observable features z depend in a predictable way on the camera projection model h(_) and the pose $\underline{\rho}(t)$ of the target object wrt the camera:

$$\underline{z} = h(\underline{\rho}(t))$$

- z could also be interpreted simply as the pose relative to the target in a position-based approach.

7.2.4.4 Controller Design
(Basic Controller)

$$\boxed{\underline{z} = h(\underline{\rho}(t))}$$

- Taking the time derivative:

Jacobian L_f is called the interaction matrix.

$$\boxed{\text{Eqn A}} \qquad \dot{\underline{z}} = \frac{\partial h}{\partial \underline{\rho}}\frac{d\underline{\rho}}{dt} = L_f \underline{v}(t)$$

- Also, by definition:

$$\Delta \underline{z} = L_f \Delta \underline{\rho}$$

- We might solve this with the LPI: $\Delta \underline{\rho} = L_f^{+} \Delta \underline{z}$

- If Δz is the feature error, then:

$$\Delta \underline{\rho} = L_f^{+}[\underline{z}_r - \underline{z}]$$

Pose error that explains the feature error to first order.

7.2.4.4 Controller Design (Basic Controller)

$$\Delta\underline{\rho} = L_f^+[\underline{z}_r - \underline{z}]$$

- Divide by a small Dt to get:

Pose velocity

$$\underline{v} = L_f^+ \frac{d}{dt}[\underline{z}_r - \underline{z}]$$

Feature error rate

- Suppose we would like the feature rate to be consistent with nulling the error in t (1/l) seconds.

$$\frac{d}{dt}[\underline{z}_r - \underline{z}] = -\frac{[\underline{z}_r - \underline{z}]}{\tau} = -\lambda[\underline{z}_r - \underline{z}]$$

- Substituting above:

$$\boxed{\underline{v}_c = -\lambda L_f^+[\underline{z}_r - \underline{z}]}$$

A VISUAL SERVO !

7.2.4.4 Controller Design

$$\underline{v}_c = -\lambda L_f^+[\underline{z}_r - \underline{z}]$$

(Basic Controller)

- This is a proportional controller with gain:

$$K_p = \lambda = 1/\tau$$

- Drives observed feature errors exponentially to zero.
- Substituting the control into the dynamics gives the closed loop error dynamics:

$$\dot{\underline{z}} = L_f \underline{v}_c(t) = -\lambda L_f L_f^+[\underline{z}_r - \underline{z}]$$

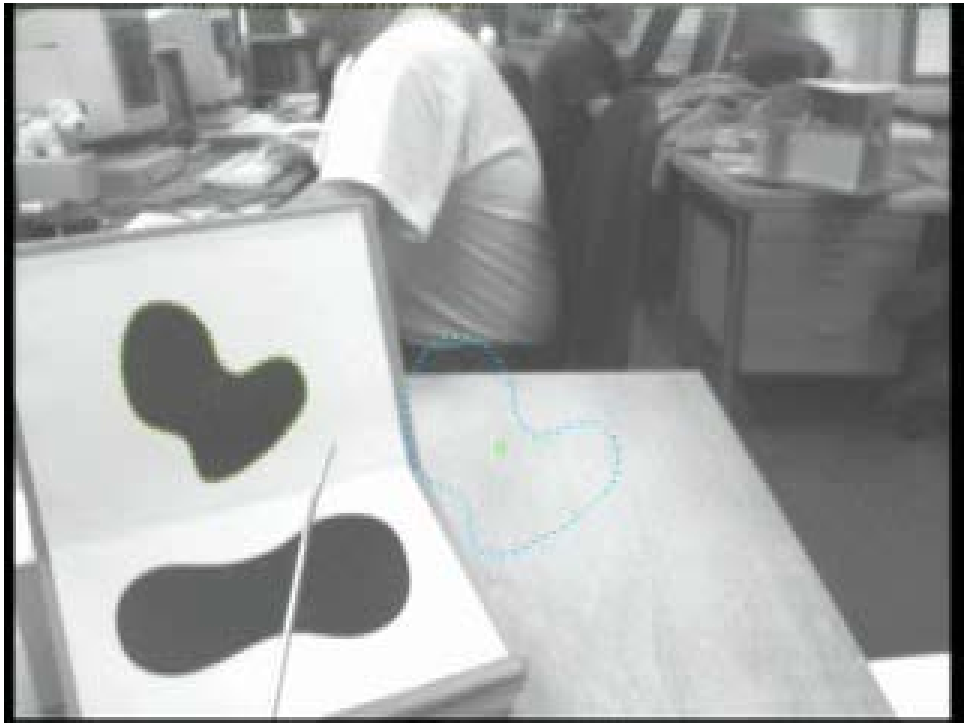
Eqn A

7.2.4.4 Controller Design (Behavior)

$$\dot{\underline{z}} = L_f \underline{v}_c(t) = -\lambda L_f L_f^{+}[\underline{z}_r - \underline{z}]$$

- Case 1: $\mathbf{L_f L_f^{+} = I}$ perfect behavior
- Case 2: $\mathbf{L_f L_f^{+} > 0}$ error decreases
- Case 3: $\mathbf{L_f L_f^{+} < 0}$ error increases
- Ideally:
  - Lf is not singular anywhere.
  - There are no non-optimum local minima.

Videos



Moment Visual Servoing

Face Visual Servoing

Outline

- 7.2 State Space Control
  - 7.2.1 Introduction
  - 7.2.2 State Space Feedback Control
  - 7.2.3 Example: Robot Trajectory Following
  - 7.2.4 Perception Based Control
  - <u>7.2.5 Steering Trajectory Generation</u>
  - Summary

7.2.5 Steering Trajectory Generation
(Trajectory Generation)

- Trajectory generation is necessary for any kind of precision control of mobile robots.
- The problem occurs in various forms:
  - "Steering" (curvature generation) problem.
  - "Smooth stopping" (velocity profile) problem.
  - Both at once
    - Sometimes in terms of linear and angular velocity.

7.2.5 Steering Trajectory Generation

(Definitions)

- Let a trajectory be a specification of an entire motion.
  - Could be explicitly in terms of state:

$$\{\underline{x}(t) \,|\, (t_0 < t < t_f)\}$$

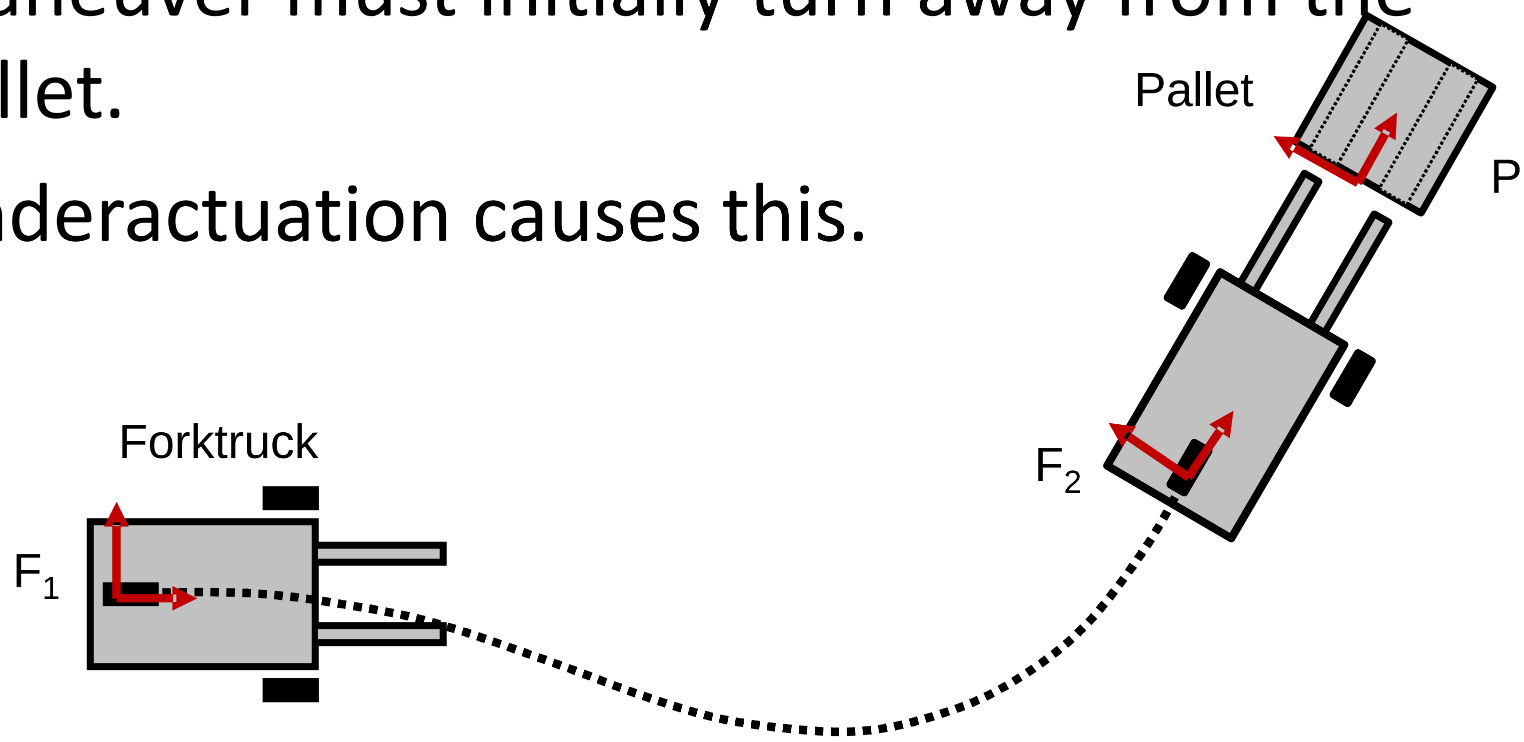
  - Could be implicitly in terms of inputs:

$$\{\underline{u}(t) \,|\, (t_0 < t < t_f)\}$$

  - Need both for 2 dof control
- Both can be visualized as the trajectory followed by the tip of a vector over time.

7.2.5 Steering Trajectory Generation
(Motivation)

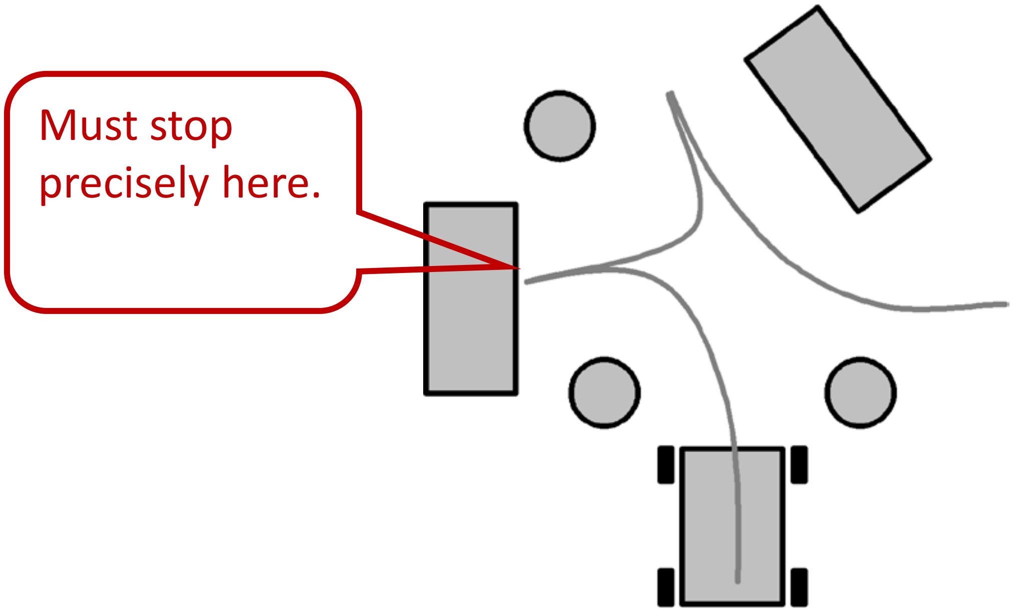
- Load cannot be approached sideways.
- Visualize driving backward from goal.
- Maneuver must initially turn away from the pallet.
- Underactuation causes this.



7.2.5 Steering Trajectory Generation

(Motivation : Precision Control)

- Precision control is necessary:
  - when goals states must be achieved precisely
  - when paths must be followed precisely.
- That happens in cluttered environments, for example.

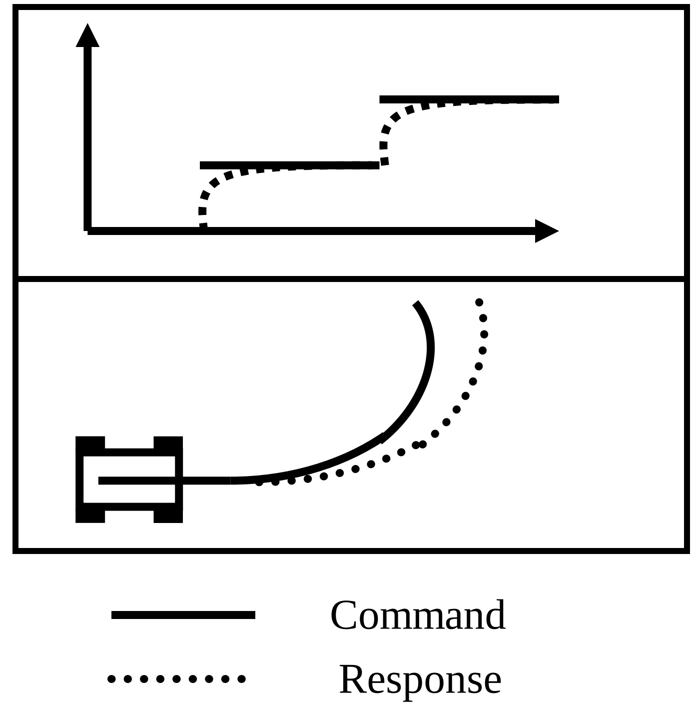


The robot must not only follow the intended curves but it must come to a stop neither too early (which would make achieving the next goal impossible) nor too late (which will cause a collision).

7.2.5 Steering Trajectory Generation

(Motivation : Reduced Following Error)

- Trajectory generation can compensate for the predictable causes of following error.
- Using trajectory generation, you can decide what to ask for, in order to get what you want.

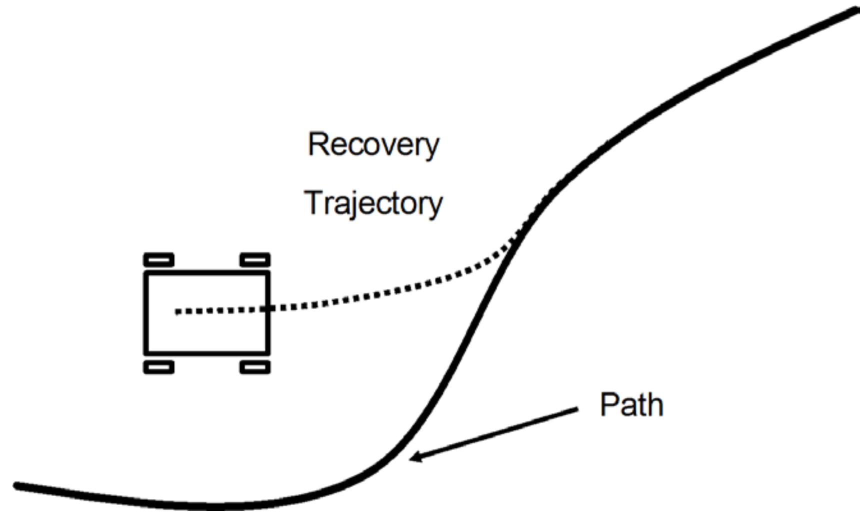


The nonideal response to a discontinuous curvature control can be visualized in either input space or state space.

7.2.5 Steering Trajectory Generation

(Motivation : Corrective Trajectories)

- Control layers above the coordinated control layer….
  - Have easier jobs to do with a trajectory generator to talk to.



The best recovery trajectory is one which reacquires the path at the correct heading and curvature.

7.2.5.1 Problem Specification

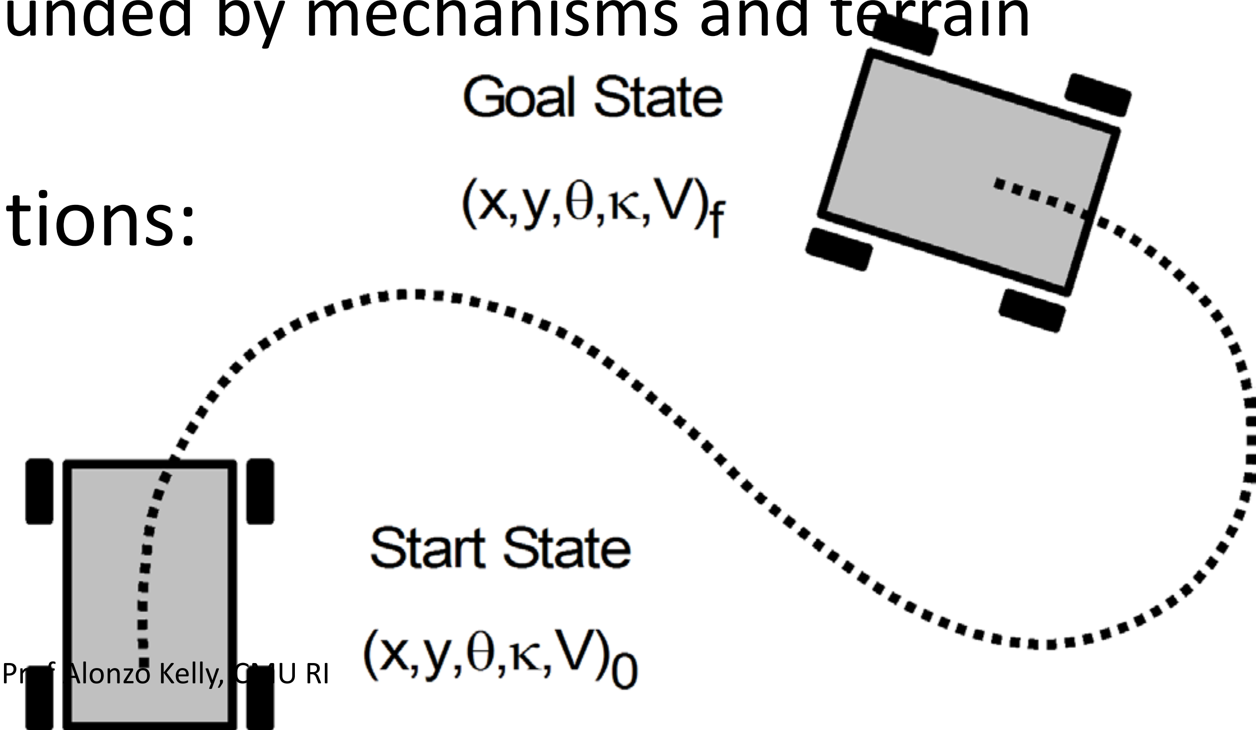
- Dynamics:

$$\dot{\underline{x}} = f(\underline{x}, \underline{u})$$

- Physical constraints: $|\underline{u}(t)| \le \underline{u}_{max}(t)$ $|\dot{\underline{u}}(t)| \le \dot{\underline{u}}_{max}(t)$
  - Turn radius bounded from below
  - Curvature is bounded by mechanisms and terrain friction.
- Boundary conditions:

$$\underline{x}(t_0) = \underline{x}_0$$

$$\underline{x}(t_f) = \underline{x}_f$$



7.2.5.1 Problem Specification

(Constraint : Dynamics)

- For a system:

$$\underline{\dot{x}} = f(\underline{x}, \underline{u})$$

- Problem: determine an entire control function u(t) which generates some desired state trajectory x(t).

$$\longrightarrow \boxed{\underline{f}^{-1}} \longrightarrow$$

- It’s the problem of inverting a differential equation.

7.2.5.2 Formulation as a Rootfinding Problem

(Existence)

- Every u(t) generates some x(t)...

$$x(t) = x(0) + \int_0^t f(\underline{x}, \underline{u}, t)dt$$

- However, many arbitrary x(t)'s represent infeasible motions.
  - Mathematical reasons – underactuation
  - Physics reasons - friction
  - Power related reasons - horsepower

7.2.5.2 Formulation as a Rootfinding Problem

(Parameterization)

- Function space of all $\underline{u}(t)$ is too large to search.
- Parameterize inputs: $\underline{u}(t) \rightarrow \tilde{\underline{u}}(\underline{p}, t)$
- Easy to see by Taylor series that $\underline{p}$ spans all possible $\underline{u}(t)$
  - Pick any $\underline{u}_k(t)$ you like.
  - Write its Taylor series
  - Coefficients $\underline{p}_k$ approximate $\underline{u}_k(t)$ arbitrarily well.
  - But $\underline{u}_k(t)$ was arbitrary too → so $\underline{p}_k$ spans everything!

7.2.5.2 Formulation as a Rootfinding Problem

(Parameterization)

- Now p determines u(t) which determines x(t), so dynamics become:

$$\dot{\underline{x}}(t) = \underline{f}[\underline{x}(\underline{p}, t), \underline{u}(\underline{p}, t), t] = \tilde{\underline{f}}(\underline{p}, t)$$

- The boundary conditions become:

Integrals are suppressed notationally – but they are still there.

$$\underline{g}(\underline{p}, t_0, t_f) = \underline{x}(t_0) + \int_{t_0}^{t_f} \tilde{\underline{f}}(\underline{p}, t)dt = \underline{x}_b$$

- This is conventionally written as:

$$\underline{c}(\underline{p}, t_0, t_f) = \underline{h}(\underline{p}, t_0, t_f) - \underline{x}_b = 0$$

7.2.5.2 Formulation as a Rootfinding Problem

(Parameterization)

- Wait a minute!:

$$\underline{c}(\underline{p}, t_0, t_f) = 0$$

- That is a rootfinding problem!
- Conclusion:
  - the problem of inverting a nonlinear vector differential equation
  - can be converted to a rootfinding problem
  - using parameterization.

7.2.5.3 Steering Trajectory Generation

- Switch to steering problem and ignore velocity.
- Often it is convenient to change the independent variable from time to distance.

$$\underline{q} = [\underline{p}^T, s_f]^T$$

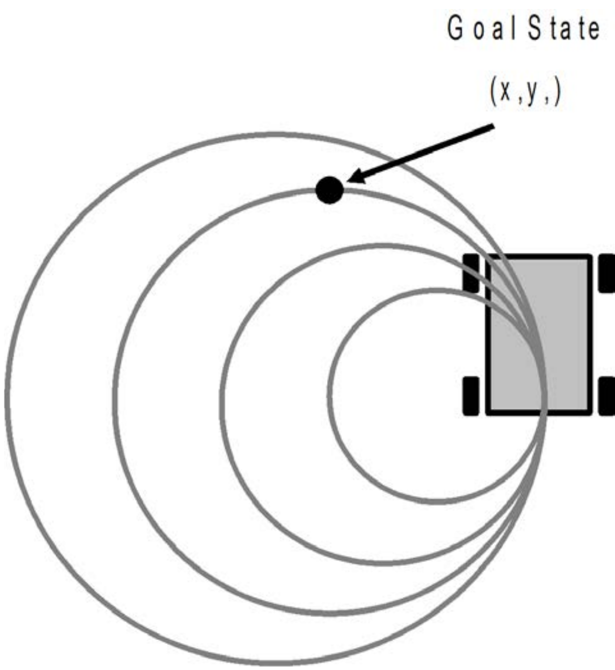
$\underline{p}$ became longer but t_0 and t_f have been eliminated.

- Let the initial distance s0 be zero and absorb the final distance into the parameter vector, thus:

7.2.5.3 Steering Trajectory Generation

(Degrees of Freedom)

- If u(p,sf) has n parameters, these can be used to satisfy n constraints.
- For example, an arc trajectory really has two parameters – radius and length.
- Imagine the circles moving outward until they hit the point.
- Its not too hard to find the radius and distance which hit the point.
- Terminal heading and curvature are beyond your control.

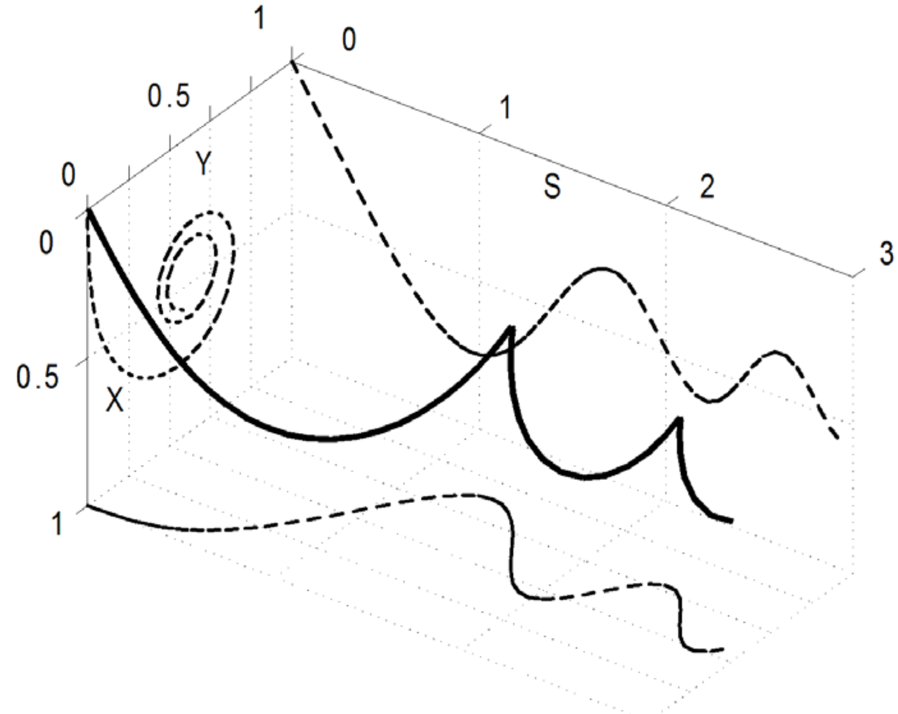


7.2.5.3 Steering Trajectory Generation
(Clothoids)

- Another historically popular curve is the clothoid. It's a linear curvature polynomial:

$$\kappa(s) = a + bs$$

- This has 3 degrees of freedom but its still not enough for some problems.



Carnegie Mellon
THE ROBOTICS INSTITUTE

7.2.5.3 Steering Trajectory Generation

(Polynomial Spirals)

- We can without loss of generality consider the initial pose to be at the origin.
- If initial and final curvature matter, that leaves FIVE constraints:

$$x(s_f) = x_f \qquad \kappa(0) = \kappa_0$$
$$y(s_f) = y_f \qquad \kappa(s_f) = \kappa_f$$
$$\theta(s_f) = \theta_f$$

- A curve with 5 dof is a cubic spiral:

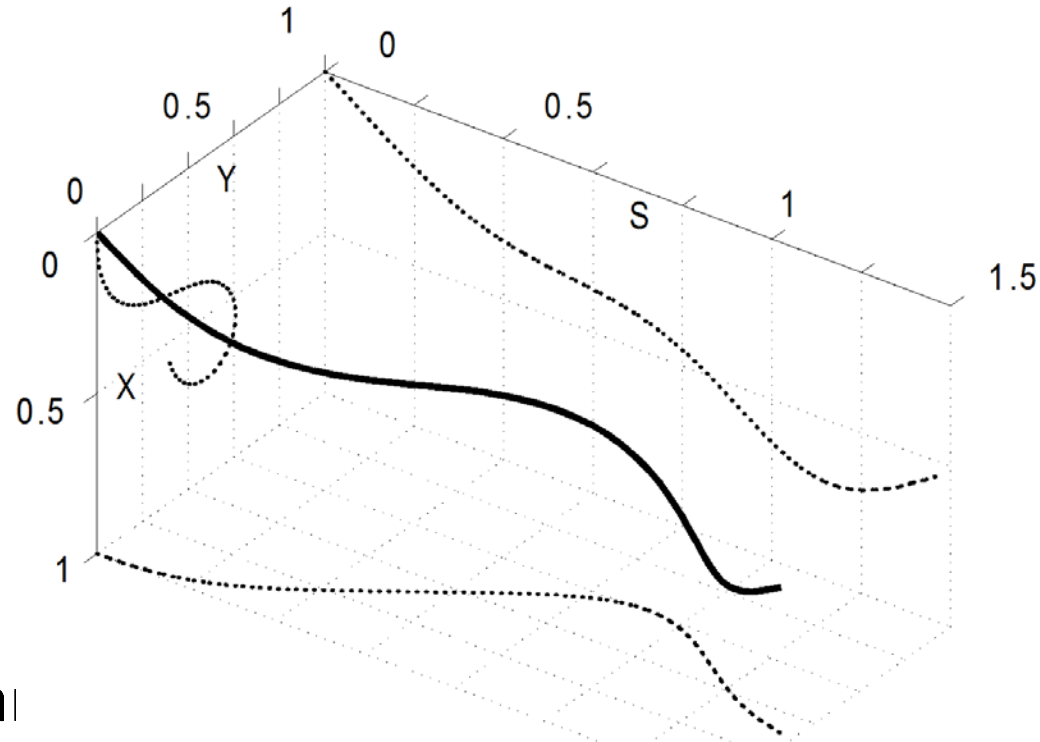
$$\kappa(s) = a + bs + cs^2 + ds^3$$

The parameter s_f must be distinguished from the variable s. The 5th parameter s_f does not appear.

7.2.5.3 Steering Trajectory Generation

(Polynomial Spirals)

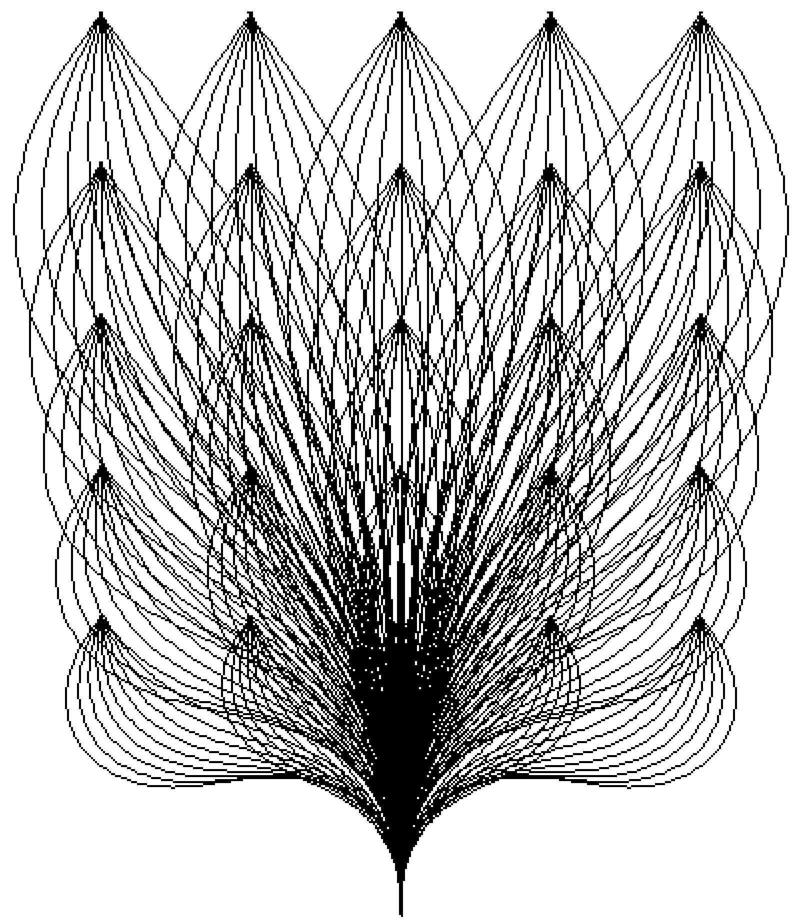
- All cases mentioned so far are special cases of polynomials.
- This form of representation has several advantages:
  - Compact, just store coefficients
  - General, any function can be approximated.
  - Heading can be computed in closed form.



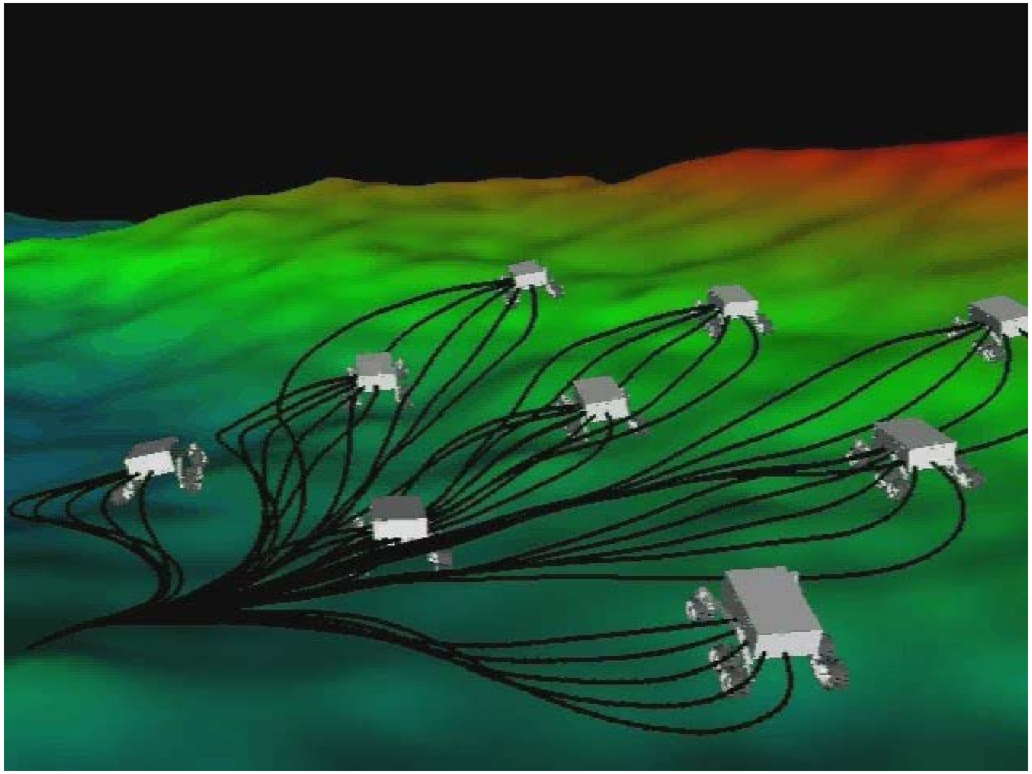
A Cubic Polynomial Spiral

7.2.5.3 Steering Trajectory Generation
(Polynomial Spirals)

- These curves can achieve any terminal posture.



Video



7.2.5.4 Numerical Formulation

(Terminal Posture Acquisition)

- There are 5 constraints:

$$(\kappa_0, x_f, y_f, \theta_f, \kappa_f)$$

- The initial curvature constraint is trivial to satisfy:

$$a = \kappa_0$$

- Denote the remaining 4 parameters by:

$$\underline{q} = \begin{bmatrix} b & c & d & s_f \end{bmatrix}^T$$

7.2.5.4 Numerical Formulation

(Terminal Posture Acquisition)

- Now there are 4 complicated constraints on the 4 parameters:

$$\kappa(\underline{q}) = \kappa_0 + b s_f + c s_f^2 + d s_f^3 = \kappa_f$$

$$\theta(\underline{q}) = \kappa_0 s_f + \frac{b}{2} s_f^2 + \frac{c}{3} s_f^3 + \frac{d}{4} s_f^4 = \theta_f$$

$$x(\underline{q}) = \int_0^{s_f} \cos\left[\kappa_0 s + \frac{b}{2} s^2 + \frac{c}{3} s^3 + \frac{d}{4} s^4\right] ds = x_f$$

$$y(\underline{q}) = \int_0^{s_f} \sin\left[\kappa_0 s + \frac{b}{2} s^2 + \frac{c}{3} s^3 + \frac{d}{4} s^4\right] ds = y_f$$

7.2.5.4 Numerical Formulation
(Linearization)

- Despite the integrals, these are just 4 nonlinear equations of the form:

$$\underline{c}(\underline{q}) = \underline{g}(\underline{q}) - \underline{x}_b = 0$$

- Solve with a rootfinding technique like Newton's method:

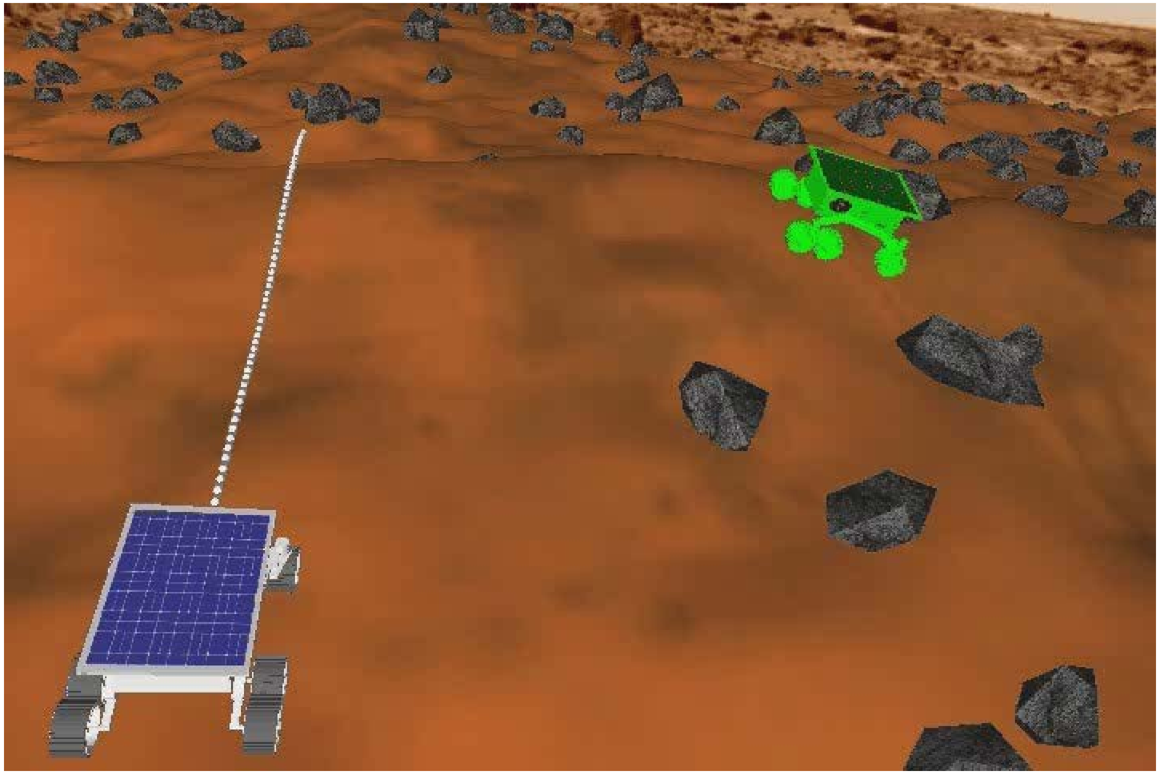
$$\Delta\underline{q} = -\underline{c}_{\underline{q}}^{-1}\underline{c}(\underline{q}) = -\underline{c}_{\underline{q}}^{-1}[\underline{g}(\underline{q}) - \underline{x}_b]$$

Demo

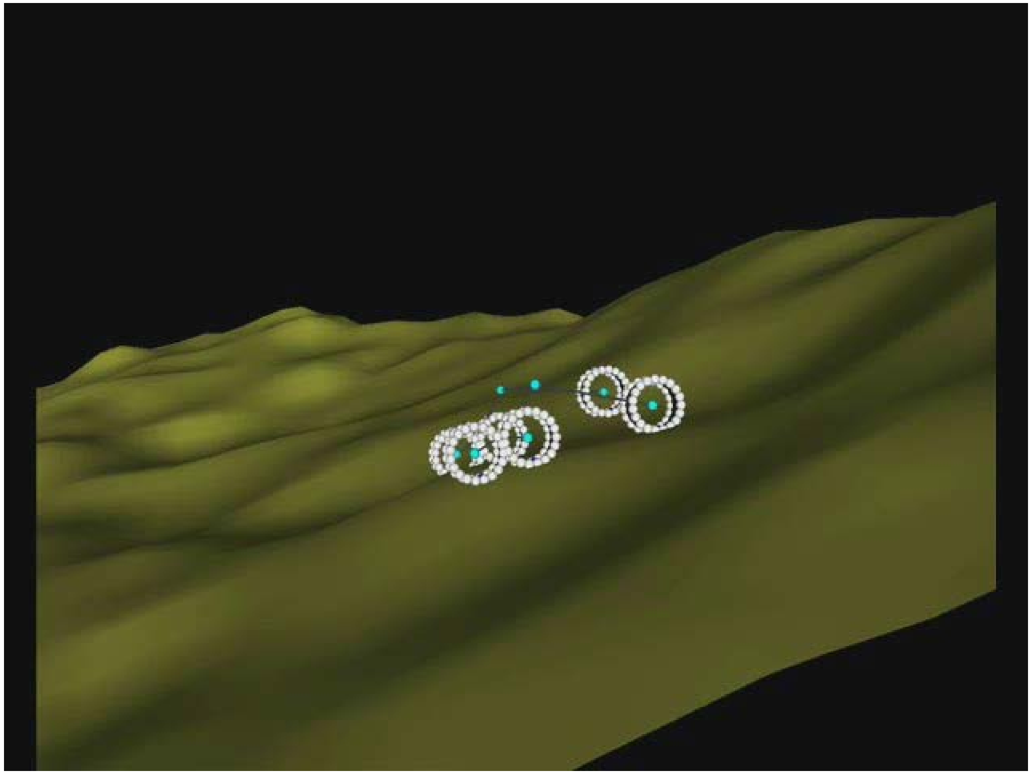
- CuboidDemonstrator1.exe

Right click and open hyperink

Video



Video



Outline

- 7.2 State Space Control
  - 7.2.1 Introduction
  - 7.2.2 State Space Feedback Control
  - 7.2.3 Example: Robot Trajectory Following
  - 7.2.4 Perception Based Control
  - 7.2.4 Steering Trajectory Generation
  - <u>Summary</u>

Carnegie Mellon
THE ROBOTICS INSTITUTE

Summary

- State Space control is more powerful than classical.
  - Theorems provide conditions for arbitrary controllability.
- Observer theory reveals duality of controls and estimation.
- 2 dof control is a good way to follow trajectories.
- Parameterization is a good way to generate them for open loop control.
- Linearization is effective for nonlinear control.

Summary

- Visual servoing implements a closed loop using vision as the feedback sensor.
- A basic version tries to drive an image into coincidence with some reference image by:
  - forming errors in image space.
  - deriving corrective velocity commands from the errors.