ME451
Kinematics and Dynamics of Machine Systems

Cam-Follower Constraints – 3.3

Driving Constraints – 3.5

October 11, 2011

"Computers are useless. They can only give you answers."
Pablo Picasso

Before we get started…

- Last Time
  - Covered composite joints and cam-follower

- Today:
  - Wrap up kinematic constraints (Cam-flat follower & Point-Follower)
  - Start cover driving constraints

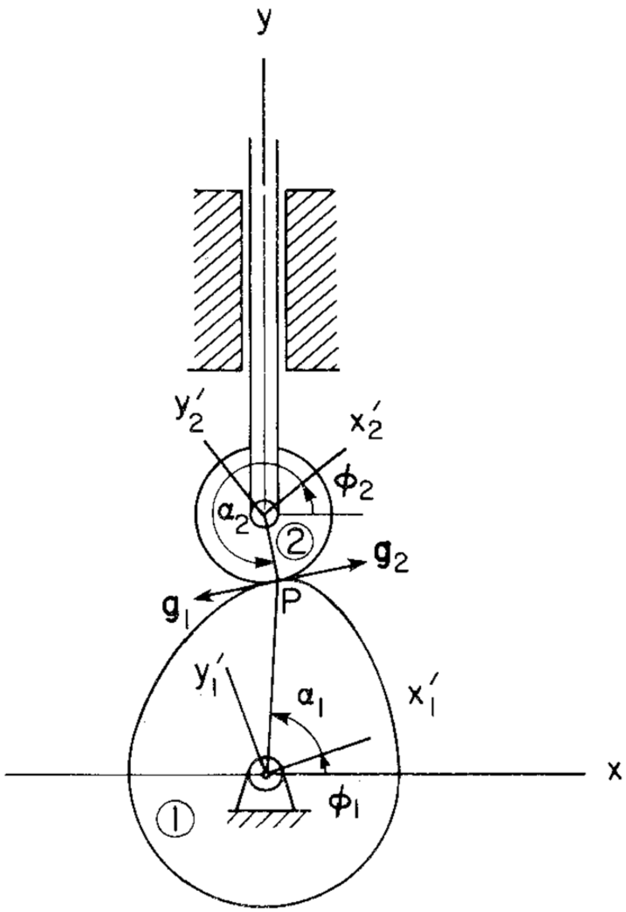
- Final Project proposal due on 10/28
  - Can be based on your simEngine2D
  - Can represent some project you undertake in ADAMS
  - I'm open to other suggestions…

- October 20 lecture: dedicated to visualization (post-processing)
  - Learn how generate an animation of the dynamics of your mechanism (generate an avi file)

- Syllabus was updated on the class website

Example

- Determine the expression of the tangents $\mathbf{g}_1$ and $\mathbf{g}_2$



$$\rho_1(\alpha_1) = \begin{cases} -\frac{1}{4}\cos 3\alpha_1 + \frac{5}{4} & \text{if} \quad 0 \le \alpha_1 < \frac{2\pi}{3} \\ 1 & \text{if} \quad \frac{2\pi}{3} \le \alpha_1 \le 2\pi \end{cases}$$

$$\rho_2(\alpha_2) = \frac{1}{4}$$

Cam flat-faced-follower Pair

- A particular case of the general cam-follower pair
  - Cam stays just like before
  - Flat follower
  - Typical application: internal combustion engine
  - Not covered in detail, HW touches on this case

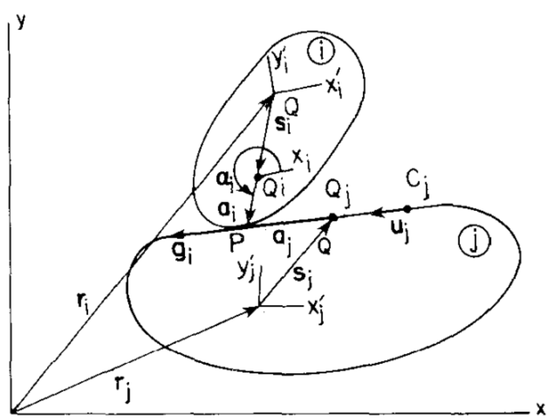


Figure 3.4.10 Cam–flat-faced follower pair.

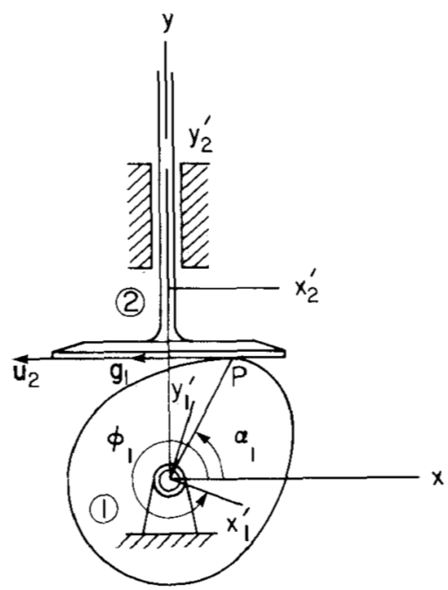


Figure 3.4.11 Cam–flat-faced follower in an internal combustion engine.

Point-Follower Pair

- Framework (Step 1):
  - Pin P is attached to body i and can move in slot attached to body j.
  - NOTE: the book forgot to mention what $\mathbf{g}_j$ is (pp.85, eq. 3.4.32)
    - It represents the tangent to the slot in which P is allowed to move

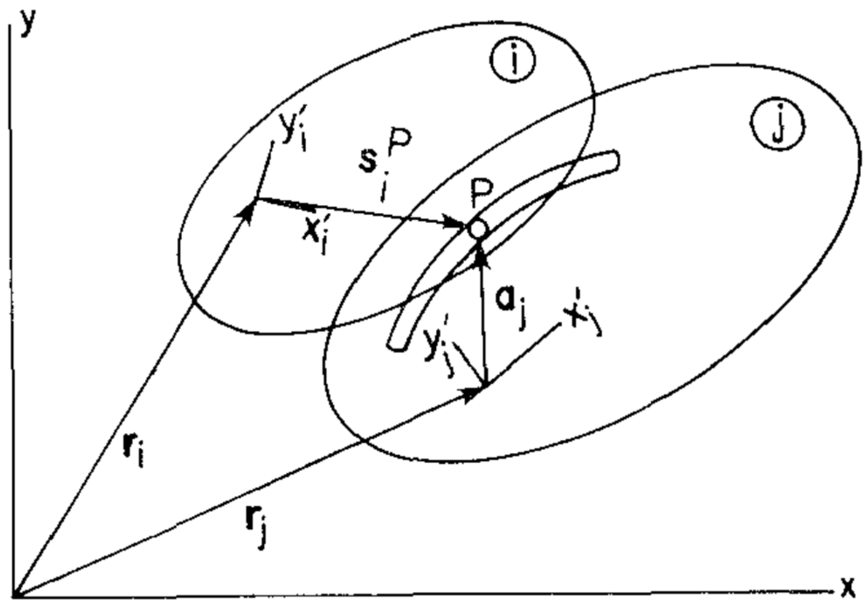


Figure 3.4.12 Point–follower pair.

- The location of point P in slot attached to body j is captured by angle α_j that parameterizes the slot.
- Therefore, when dealing with a point-follower we'll be dealing with the following set of generalized coordinates:
  - Body i: x_i, y_i, ϕ_i,
  - Body j: x_j, y_j, ϕ_j, α_j

Point-Follower Pair

- Step 2: Constraint Equations $\Phi(\mathbf{q},t)$ = ?

- Step 3: Constraint Jacobian $\Phi_{\mathbf{q}}$ = ?

- Step 4: ν = ?

- Step 5: γ = ?

Driving Constraints

- The context
  - Up until now, we only discussed time invariant kinematic constraints
  - Normally the mechanism has a certain number of DOFs
  - Some additional time dependent constraints ("drivers") are added to control these "unoccupied" DOFs
    - You thus control the motion of the mechanism
    - For Kinematics Analysis, you need NDOF=0

Kinematic Drivers

Absolute Coordinate Drivers

- Absolute x-coordinate driver
- Absolute y-coordinate driver
- Absolute angle driver

Relative Coordinate Drivers

- You see these more often…
  - Relative distance driver
  - Revolute-rotational driver
  - Translational-distance driver

Absolute Driving Constraints

- Indicate that the coordinate of a point expressed in the global reference frame assumes a certain value that changes with time

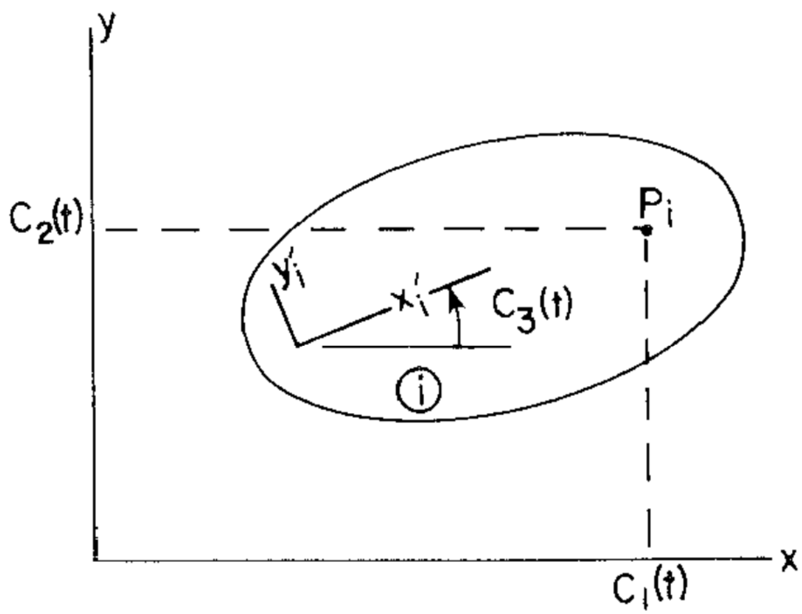


Figure 3.5.1 Absolute coordinate drivers.

- X-position

$$x^{P_i} - C_1(t) = 0$$

- Y-position

$$y^{P_i} - C_2(t) = 0$$

- Orientation angle

$$\phi_i - C_3(t) = 0$$

Absolute Driver Constraints

- Step 2: Constraint Equations $\Phi(\mathbf{q},t) = ?$

- Step 3: Constraint Jacobian $\Phi_{\mathbf{q}} = ?$

- Step 4: $\nu = ?$

- Step 5: $\gamma = ?$

Absolute Coordinate Drivers

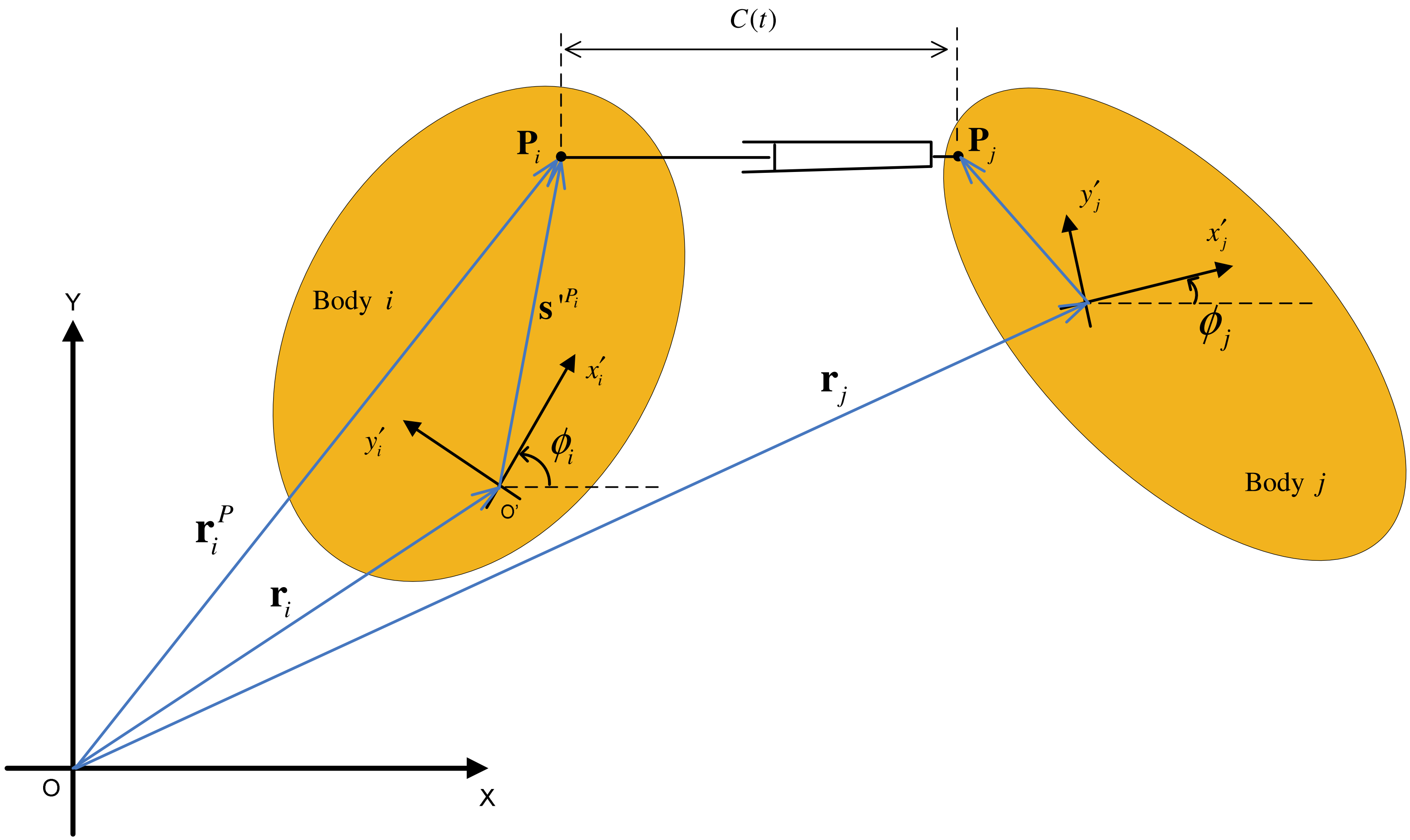
- Very simple to compute this information
  - Add C'(t) to expression of ν
  - Add C''(t) to expression of γ

End Absolute Driving Constraints

Begin Relative Driving Constraints

Relative Distance Driver

- The distance between P_i and P_j is prescribed as a function of time: $||P_i\ P_j||=C(t)$



Relative Distance Driver

- Step 2: Constraint Equations

$$\mathbf{\Phi}^{rdc(i,j)}(\mathbf{q},t) = (\mathbf{r}_i^P - \mathbf{r}_j^P)^T(\mathbf{r}_i^P - \mathbf{r}_j^P) - C^2(t) = 0$$

- Step 3: Constraint Jacobian $\mathbf{\Phi}_{\mathbf{q}}$ = ? (see Eq. 3.3.8)

$$\mathbf{\Phi}_{\mathbf{q}_i}^{rdc(i,j)} = 2(\mathbf{r}_i^P - \mathbf{r}_j^P)^T \begin{bmatrix} \mathbf{I} & \mathbf{B}_i \mathbf{s}_i'^P \end{bmatrix}$$

$$\mathbf{\Phi}_{\mathbf{q}_j}^{rdc(i,j)} = 2(\mathbf{r}_j^P - \mathbf{r}_i^P)^T \begin{bmatrix} \mathbf{I} & \mathbf{B}_j \mathbf{s}_j'^P \end{bmatrix}$$

- Step 4: ν = ?

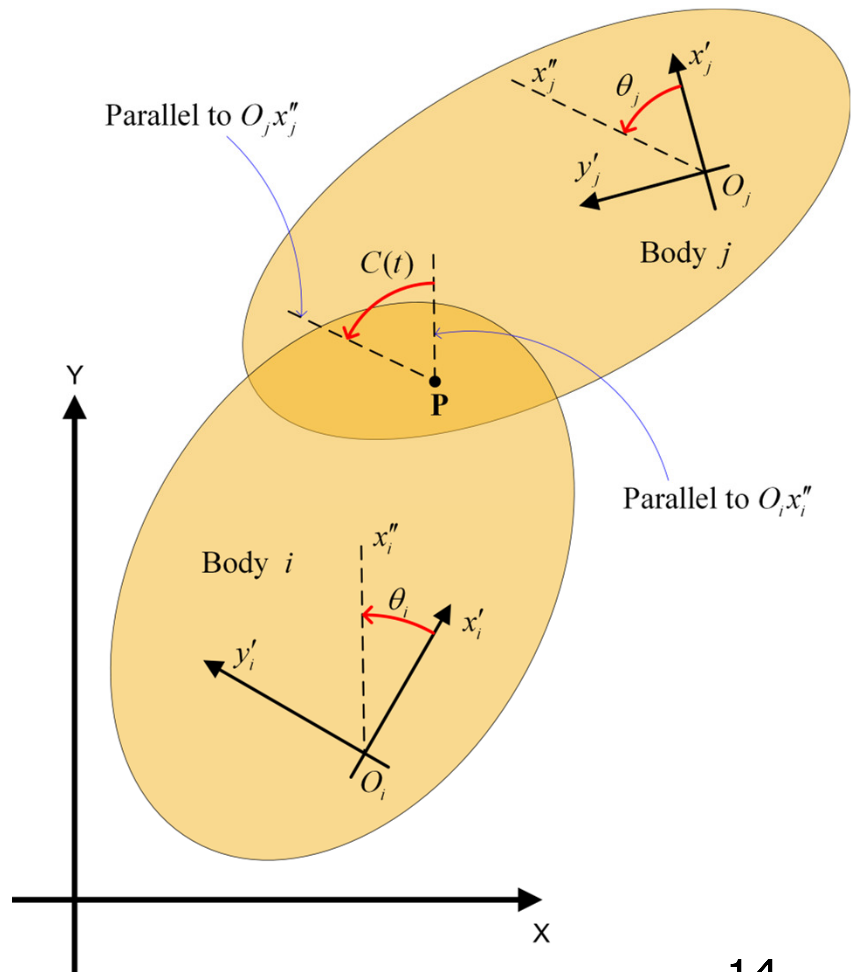
$$\nu = 2C(t)\dot{C}(t)$$

- Step 5: γ = ?

$$\gamma^{rdc(i,j)} = -2(\dot{\mathbf{r}}_i^P - \dot{\mathbf{r}}_j^P)^T(\dot{\mathbf{r}}_i^P - \dot{\mathbf{r}}_j^P) + 2(\mathbf{r}_i^P - \mathbf{r}_j^P)^T(\dot{\phi}_i^2 \mathbf{A}_i \mathbf{s}_i'^P - \dot{\phi}_j^2 \mathbf{A}_j \mathbf{s}_j'^P) + 2C(t)\ddot{C}(t) + 2\dot{C}^2(t)$$

Revolute Rotational Driver

- The framework: at point **P** we have a revolute joint
- It boils down to this: you prescribe the time evolution of the angle in the revolute joint



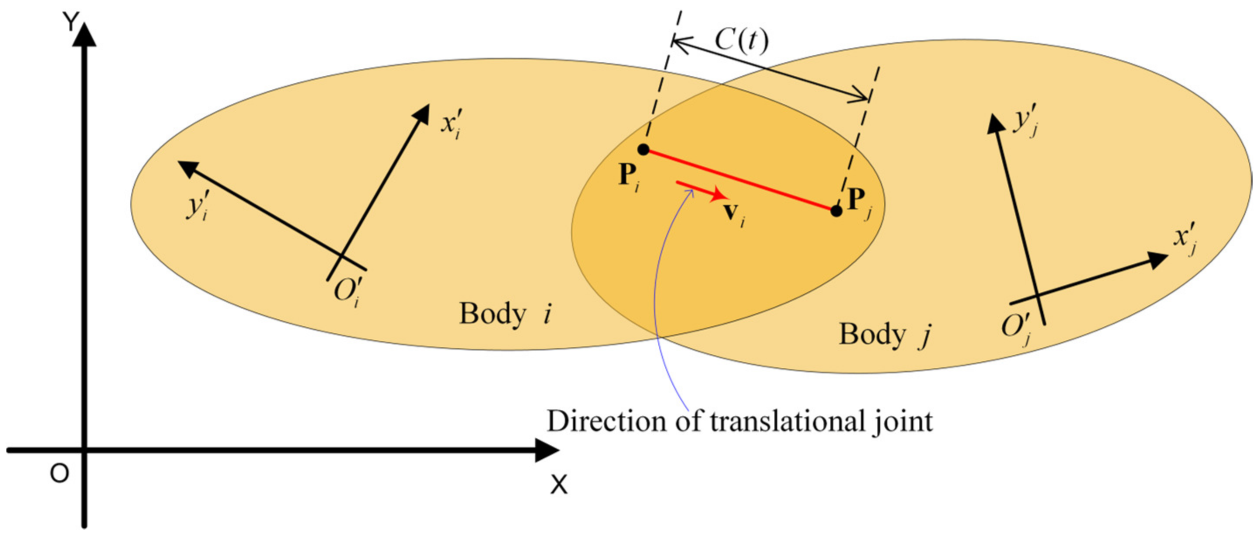
- Driver constraint formulated as

$$\Phi^{rrd(i,j)}(t) = (\phi_j + \theta_j) - (\phi_i + \theta_i) - C(t) = 0$$

- Note that θ_i and θ_j are attributes of the constraint
  - They are constants, they can always be chosen to be zero by slightly modifying C(t)

Translational Distance Driver

- The framework: we have a translational joint between two bodies
  - Direction of translational join on "Body i" is defined by the vector $\mathbf{v}_i$



- This driver says that the distance between point P_i on "Body i" and point P_j on "Body j" measured along in the direction of $\mathbf{v}_i$ changes in time according to a user prescribed function $C(t)$:

$$\frac{\mathbf{v}_i^T \mathbf{d}_{ij}}{v_i} - C(t) = 0$$

Translational Distance Driver (Cntd.)

- The book complicates the formulation with no good reason
  - There is nothing to prevent me to specify the direction $\mathbf{v}_i$ by selecting this quantity to have magnitude 1

- Equivalently, the constraint then becomes

$$\mathbf{v}_i^T \mathbf{d}_{ij} - C(t) = 0$$

- Keep in mind that the direction of translation is indicated now through a unit vector (you are going to get the wrong motion if you work with a $\mathbf{v}_i$ that is not unit length)

- Keep this in mind when working on problem 3.5.6 (assigned on Th)

Driver Constraints, Departing Thoughts

- What is after all a driving constraint?
  - You take your kinematic constraint, which indicates that a certain *kinematic quantity* should stay equal to zero
  - Rather than equating this *kinematic quantity* to zero, you have it change with time…

$$\Phi(\mathbf{q}) = \mathbf{0} \qquad \text{versus} \qquad \Phi(\mathbf{q}) = C(t)$$

- Or equivalently…

$$\underbrace{\Phi(\mathbf{q}) = \mathbf{0}}_{\text{Geometric Constraint}} \qquad \text{versus} \qquad \underbrace{\Phi(\mathbf{q}) - C(t) = \mathbf{0}}_{\text{Driver Constraint}}$$

Driver Constraints, Departing Thoughts (cntd.)

- Notation used:
  - For Kinematic Constraints: $\Phi^K(\mathbf{q})$
  - For Driver Constraints: $\Phi^D(\mathbf{q},t)$
  - Note the arguments (for K, there is no time dependency)

- Correcting the RHS…
  - Computing for Driver Constraints the right hand side of the velocity equation and acceleration equation is straightforward

  - Once you know who to compute these quantities for $\Phi^K(\mathbf{q})$, when dealing with $\Phi^D(\mathbf{q},t)$ is just a matter of correcting…
    - … ν (RHS of velocity equation) with the first derivative of C(t)
    - … γ (RHS of acceleration equation) with second derivative of C(t)
    - Section 3.5.3 discusses these issues

MATLAB: How to Handle Arbitrary Motions

- The function C(t) should be read from an input file and you need to be able to evaluate it as well as its first and second time derivatives

```
% Suppose you already read from an adm input file the string that defines
% the motion prescribed and that it's stored in "CmotionFunction"
CmotionFunction = '(1.5*sin(t) + 3*t^2)^2'

% This is the relevant/interesting part… NOTE: "eval" and "matlabFunction" are MATLAB native functions
syms t;
cFunction = eval(CmotionFunction)
functionHandleValue = matlabFunction(cFunction ,'vars', [t])

cFunctionPrime = diff(CmotionFunction)
functionHandleFirstDeriv = matlabFunction(cFunctionPrime,'vars', [t])

cFunctionPrimePrime = diff(diff(CmotionFunction))
functionHandleSecondDeriv = matlabFunction(cFunctionPrimePrime,'vars', [t])

[v,f,s] = somePhiConstraint(functionHandleValue, functionHandleFirstDeriv, functionHandleSecondDeriv, 2.3)
```

```
function [value,firstD,secondD] = someDrivingConstraint(f,fPrime,fPrimePrime,t)
% this is where you need to use C(t) and its derivatives…
value = f(t);
firstD = fPrime(t);
secondD = fPrimePrime(t);
```

Example:
Specifying Relative Distance Drivers

- Generalized coordinates: $\mathbf{q} = [\phi_1, x_2, y_2, \phi_2]^T$

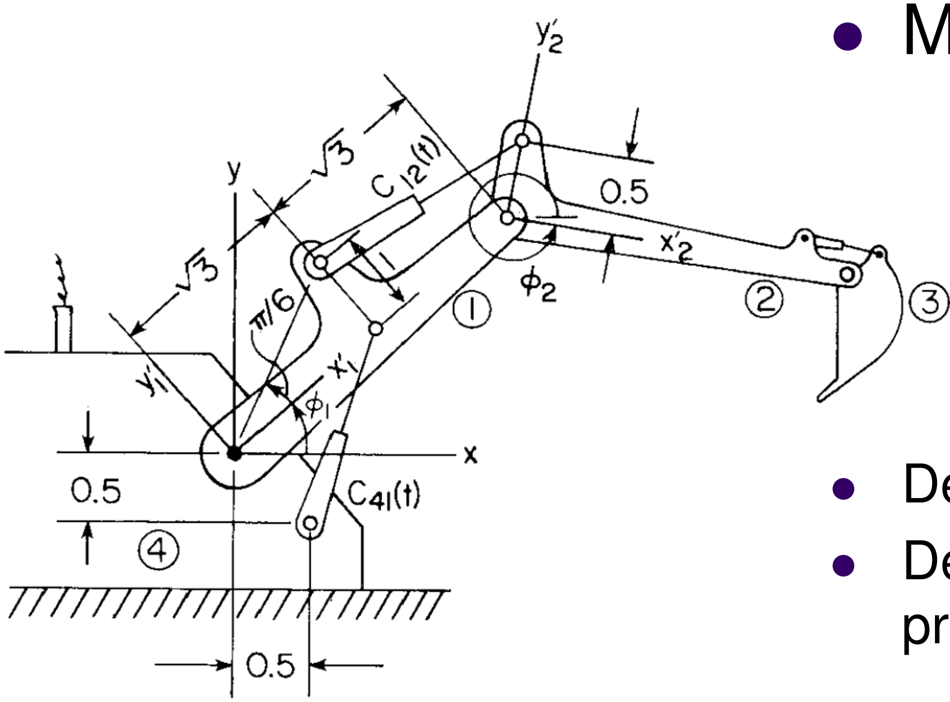


Figure 3.5.6 Excavator boom assembly with two distance drivers.

- Motions prescribed:

$$C_{41}(t) = \tfrac{1}{5}\, t + 1.8$$

$$C_{12}(t) = \tfrac{1}{10}\, t + 1.9$$

- Derive the constraints acting on system
- Derive linear system whose solution provides velocities $\dot{\mathbf{q}} = [\dot{\phi}_1, \dot{x}_2, \dot{y}_2, \dot{\phi}_2]^T$