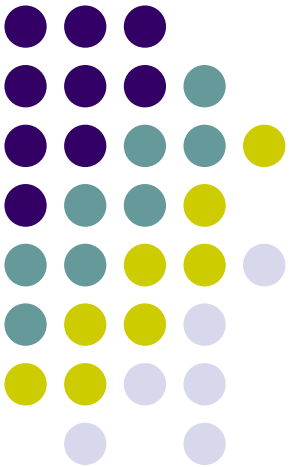
ME451
Kinematics and Dynamics of Machine Systems

Kinematic Analysis

Wrecker boom example

Matlab Implementation Issues.

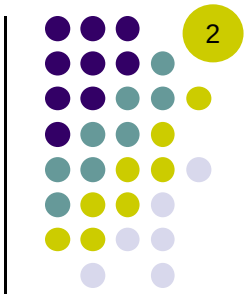
October 28, 2014



Dan Negrut
ME451, Fall 2014
University of Wisconsin-Madison

- Quote of the day: “To succeed in life, you need two things: ignorance and confidence.”
-- Mark Twain

Before we get started…



- Last time
  - Wrap up, MATLAB implementation issues
  - Singular configurations in Kinematic analysis
  - Example, wrecker boom
- Today
  - Wrap up, wrecker boom example
  - `simEngine2D`-related MATLAB implementation issues
    - Focus on assembling `Phi`, `Nu`, `Gamma`, and `Jac`
- Midterm exam: 11/04/2014 (during regular class hours, same room)
  - Open everything
  - Midterm Review: on 11/03/2014
    - Starts at 7:15 pm, runs as long as needed
    - Room TBA
- Project 1 made available online
  - Requires you to use `simEngine2D` in conjunction with excavator example discussed in class
  - Due on Nov 11 at 11:59 PM

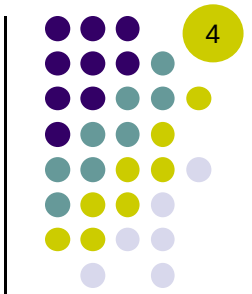
[handout]

Wrecker-Boom Example

- We are interested in the KINEMATICS of this mechanism
  - That is, we are interested in how this mechanism moves in response to a set of *two* kinematic drivers (motions) applied to it
  - Relatively straight forward to check that this mechanism has NDOF=0
  - Using full set of generalized coordinates to warm up for `simEngine2D` design

- Recall what we have to do:
  - **Step A**: Identify *all* physical *joints* and *drivers* present in the system
  - **Step B**: Identify the corresponding set of constraint equations $\mathbf{\Phi}(\mathbf{q}, t) = \mathbf{0}$
  - **Step C**: Compute $\mathbf{\Phi_q}$: needed for the Position Analysis
  - **Step D**: Compute $\boldsymbol{\nu}$: needed for the Velocity Analysis
  - **Step E**: Compute $\boldsymbol{\gamma}$: needed for the Accelerations Analysis

simEngine2D: Implementation Aspects

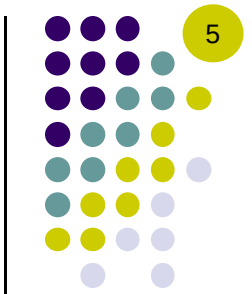


- Nomenclature issue
  - Kinematic constraints: revolute joint, distance constraint, cam and follower constraint, translational joint, absolute orientation constraint, etc.

- Some kinematic constraints induce one constraint equation while others introduce two or maybe three equations
    - Example, three equations: cam and follower constraint
    - Example, two equations: revolute and translational joints
    - Example, one equation: distance constraint, relative orientation constraint, etc.

- NOTE:
  - A kinematic constraint is a physical thing
  - Kinematic constraint equation[s]: how a kinematic constraint ends up being modeled
  - The number of kinematic constraint equations always larger than the number of kinematic constraints
    - Because you have kinematic constraints that induce two or three equations (ex: revolute joint)

simEngine2D: Implementation Aspects



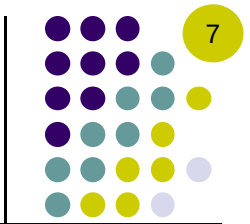
- MATLAB variables of relevance
  - `crntT` – the current time at which the analysis takes place (≥0)
  - `NKC` – number of kinematic constraints
  - `NKCE` – number of kinematic constraint equations
  - `Nbodies` – number of 2D bodies present in the mechanism
    - Note: The ground is not a body, it introduces no generalized coordinates
  - `KinCon` – an array containing all the kinematic constraints present in the system
    - There are exactly `NKC` entries in this array
  - `BodyArray` – an array containing all the bodies present in the model
  - `qState` – array of size 3*`Nbodies` that stores all body positions
  - `qdState` – array of size 3*`Nbodies` that stores all body velocities
  - `qddState` – array of size 3*`Nbodies` that stores all body accelerations

`simEngine2D`: Implementation Aspects

- MATLAB variables of relevance
  - `Phi` – array of size `NKCE` that holds the Φ of all kinematic constraints present in the model
  - `Nu` – array of size `NKCE` that holds the ν of all kinematic constraints present in the model
  - `Gamma` - array of size `NKCE` that holds the γ of all kinematic constraints present in the model
  - `Jac` – matrix that has `NKCE` rows and 3*`Nbodies` columns

- Notes:
  - Blue font used herein for MATLAB variables
  - Pseudo code provided next is not meant to be MATLAB copy-and-paste ready
    - Used for illustration, many variables are not defined or set

Last Lecture:
Generic implementation, Kin Analysis



```
% Generic code for position analysis.
% Performs Newton iterations as long as the correction is too large,
% but do not exceed a preset number of iterations.
normCorrection = 1.0;
tolerance = 1.E-8;
max_iterations = 20;
flags.PhiNeeded   = 1;
flags.JacNeeded   = 1;
flags.NuNeeded    = 0;
flags.GammaNeeded = 0;
Tend = 5.0;
deltaT = 0.05;
crntT = 0;
qState = getInitialGuess();
while crntT<Tend
    for it = 1:max_iterations
        updatePhiNuGammaJac(kinCon,crntT,qState,qdState,BodyArray,flags,Phi,Nu,Gamma,Jac);
        correction = Jac\Phi;

        normCorrection = norm(correction);
        fprintf('    norm correction = %g\n', normCorrection);
        if normCorrection <= tolerance
            break;
        end
        qState = qState - correction;
    end
    % save qState in some array that you can access later for plotting/animation;
    % might want to do velocity and acceleration analysis here
    crntT = crntT + deltaT;
end
```

How "updatePhiNuGammaJac" might look like

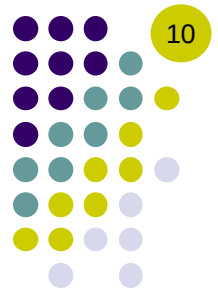
```
for i=1:NKC
         switch kinCon(i).type
          case {'revolute'}
            if flags.phiNeeded==1
                updatePhiREV(kinCon(i), crntT, qState, BodyArray, Phi);
            end
            if flags.JacNeeded==1
                updateJacREV(kinCon(i), crntT, qState, BodyArray, Jac);
            end
            if flags.NuNeeded==1
                updateNuREV(kinCon(i), crntT, qState, BodyArray, Nu);
            end
            if flags.GammaNeeded==1
                updateGammaREV(kinCon(i), crntT, qState, qdState, BodyArray, Gamma);
            end
          case 'distanceConstraint'
            if flags.phiNeeded==1
                updatePhiDISTCON(kinCon(i), crntT, qState, BodyArray, Phi);
            end
            if flags.JacNeeded==1
                updateJacDISTCON(kinCon(i), crntT, qState, BodyArray, Jac);
            end
            if flags.NuNeeded==1
                updateNuDISTCON(kinCon(i), crntT, qState, BodyArray, Nu);
            end
            if flags.GammaNeeded==1
                updateGammaDISTCON(kinCon(i), crntT, qState, qdState, BodyArray, Gamma);
            end
          case 'Orientation'
            % include here code for orientation
            % and then keep going through other constraint
          otherwise
            disp('Unknown kinematic constraint. Stopping execution...');
            exit;
        end
end
```

Comments, `updatePhiREV`

- These comments apply to `updatePhiBLAH` , no matter what BLAH is: revolute, translational, etc.
- The comments also apply to `updateNuBLAH` and `updateGammaBLAH`
- Each BLAH will have to evaluate its Φ. For instance
  - If BLAH is REV, then implement in MATLAB Eq. 3.3.10 (or 3.3.11)
    - Two entries will get updated in the overall `Phi` array
  - If BLAH is TRAN, then implement in MATLAB Eq. 3.3.13
    - Two entries will get updated in the overall `Phi` array
  - If BLAH is DISTCON, then implement in MATLAB Eq. 3.3.7
    - One entry will get updated in the overall `Phi` array
  - Etc.
- The same goes for updating the arrays `Nu` and `Gamma`

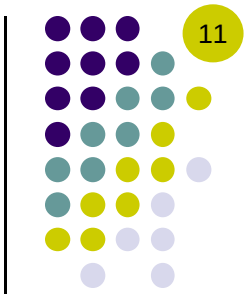
Argument List Discussion:

`updatePhiBLAH(kinCon(i), CrntT, qState, BodyArray, Phi)`



- `kinCon(i)` – a MATLAB structure that contains all the attributes of i^{th} kinematic constraint so that you can populate `Phi` with the right entries
  - Make sure you understand what a MATLAB data "structure" is
- `crntT` – you should know the time at which you are evaluating Φ
  - Think about handling a motion, for instance
- `qState` – you need the position of all bodies
- `BodyArray` - you need access to the collection of bodies in the model since you have markers on bodies that you need access to
- `Phi` – you need this array to populate it with the value[s] you compute
  - For instance, for a distance constraint you compute a number that you should deposit in the right position in Φ
- NOTE: When calling `updateGammaBLAH`, you need to also pass `qdState`, since computation of `Gamma` depends on velocities

Assembly Issues – Part 1: `Phi`, `Nu`, `Gamma`, and `Jac`



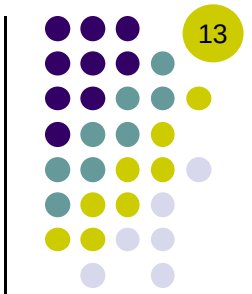
- You will have to generate an array of integers `nOfEqs`, that each kinematic constraint induces
  - The entry `nOfEqs(i)` of this array gives the number of equations (or rows) that the i^{th} kinematic constraint will introduce in `Phi`, `Nu`, and `Gamma`

- A prefix scan on the array `nOfEqs(i)` will produce the offsets that each kinematic constraint needs to observe when injecting its entries in `Phi`, `Nu`, and `Gamma`
  - Save this offset in a field of the structure `kinCon(i)`
    - For instance, `kinCon(i).rowEntry = 4`

Assembly Issues – Part 1: `Phi`, `Nu`, `Gamma`, and `Jac`

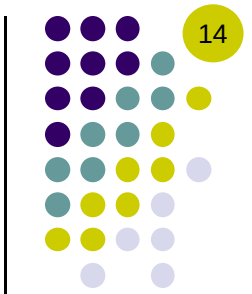
- Row offset example
  - Assume you have four kinematic constraints that are stored in `kinCon`, such that `kinCon(1).type='revolute'`, `kinCon(2).type='distanceConstraint'`, `kinCon(3).type='translational'`, and `kinCon(4).type='orientation'`
  - Then, `nOfEqs(1)=2`, `nOfEqs(2)=1`, `nOfEqs(3)=2`, `nOfEqs(4)=1`
  - Therefore, `kinCon(1).rowEntry=1`, `kinCon(2).rowEntry=3`, `kinCon(1).rowEntry=4`, `kinCon(1).rowEntry=5`
  - These numbers are obtained doing what is called a "prefix scan" on the array that starts with offset 1
    - Our example, input array: 2, 1, 2, 1
    - Our example, output array: 1, 1+2, 1+2+1, 1+2+1+2
  - Note that the row dimension of `Phi`, `Nu`, `Gamma`, and `Jac` is 6 (that is, 2+1+2+1)

Assembly Issues – Part 2: `Jac`



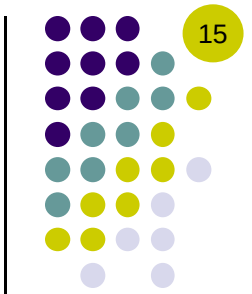
- The Part 1 discussed what row you place your information in when you do assembly of `Phi`, `Nu`, and `Gamma`
- Part 2 discusses how to figure out the column in which you should deposit values that you computed
- This is relevant only in conjunction with the Jacobian `Jac` computation
- `Phi`, `Nu`, and `Gamma`, are vectors, so only row entry information is necessary to figure out where you need to inject the contributions of a kinematic constraint

Assembly Issues – Part 2: `Jac`



- Question at hand: you have `kinCon(k)` that captures information about the k^{th} kinematic constraint. In what columns of the Jacobian matrix `Jac` should you place the partial derivatives that you compute for `kinCon(k)`?
- For instance, if `kinCon(k).type='revolute'` you compute two sets of partial derivatives, see textbook Eq. 3.3.12
  - Where do you put the 2 by 3 matrix $\Phi_{q_i}^{revolute}$?
  - Where do you put the 2 by 3 matrix $\Phi_{q_j}^{revolute}$?
- Important information that should be available: who body i and body j are for `kinCon(k)`
- These two bodies dictate the columns of `Jac` where the $\Phi_{q_i}^{revolute}$ and $\Phi_{q_j}^{revolute}$ submatrices will go

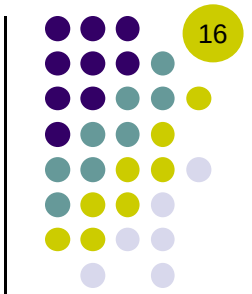
Assembly Issues – Part 2: `Jac`



- Column offset example
  - Assume in this example that you have four kinematic constraints stored in `kinCon`, such that `kinCon(1).type='revolute'`, `kinCon(2).type='distanceConstraint'`, `kinCon(3).type='translational'`, and `kinCon(4).type='Orientation'`
  - Assume also in this example that the mechanism has three bodies and that:
    - `kinCon(1).BodyI=2` and `kinCon(1).BodyJ=1` (goes with $\Phi^{revolute}$)
    - `kinCon(2).BodyI=1` and `kinCon(2).BodyJ=0` (goes with $\Phi^{distance}$)
    - `kinCon(3).BodyI=3` and `kinCon(3).BodyJ=2` (goes with $\Phi^{translational}$)
    - `kinCon(3).BodyI=3` and `kinCon(3).BodyJ=0` (goes with $\Phi^{orientation}$)

  - Then, here's how the assembly works
    - The 2 by 3 sub-block $\Phi_{q2}^{revolute}$ goes in rows 1 and 2, columns 4, 5, and 6
    - The 2 by 3 sub-block $\Phi_{q1}^{revolute}$ goes in rows 1 and 2, columns 1, 2, and 3
    - The 1 by 3 sub-block $\Phi_{q1}^{distance}$ goes in row 3, columns 1, 2, and 3
    - The 2 by 3 sub-block $\Phi_{q3}^{translatioal}$ goes in rows 4 and 5, columns 7, 8, and 9
    - The 2 by 3 sub-block $\Phi_{q2}^{translational}$ goes in rows 4 and 5, columns 4, 5, and 6
    - The 1 by 3 sub-block $\Phi_{q3}^{orientation}$ goes in row 6, columns 7, 8, and 9

Assembly Issues – Part 2: `Jac`



- One more place where the Body I and Body J information is important: picking up the generalized coordinates and velocities to inject entries in `Phi`, `Nu`, `Gamma`, and `Jac`. In our example,
  - `kinCon(1).type='revolute'`, `kinCon(2).type='distanceConstraint'`, `kinCon(3).type='translational'`, and `kinCon(4).type='Orientation'`

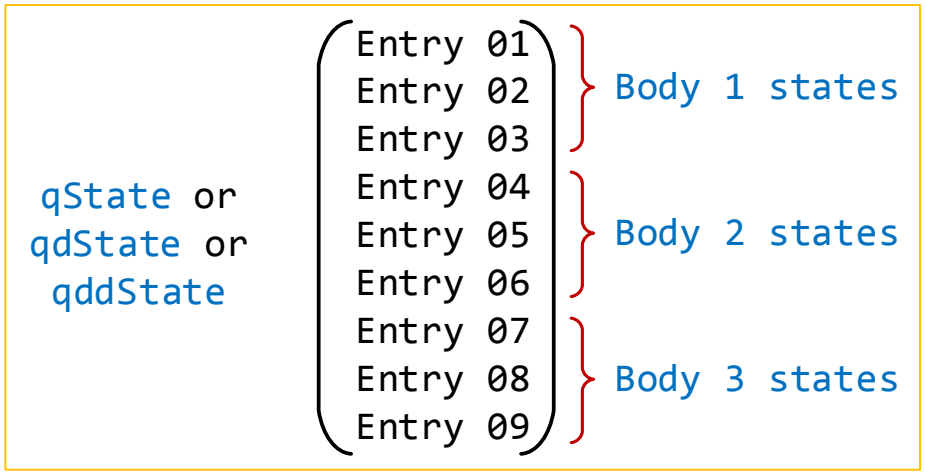
  - `kinCon(1).BodyI=2` and `kinCon(1).BodyJ=1` (goes with $\Phi^{revolute}$)
  - `kinCon(2).BodyI=1` and `kinCon(2).BodyJ=0` (goes with $\Phi^{distance}$)
  - `kinCon(3).BodyI=3` and `kinCon(3).BodyJ=2` (goes with $\Phi^{translational}$)
  - `kinCon(3).BodyI=3` and `kinCon(3).BodyJ=0` (goes with $\Phi^{orientation}$)

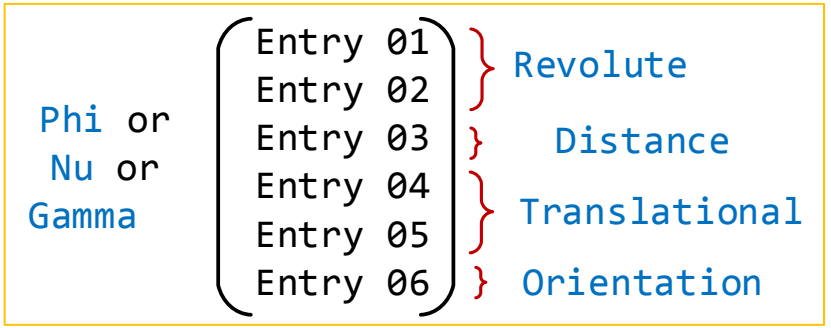
    - The role of $\boldsymbol{q}_i$ in Eq. 3.3.10 is played by the entries 4, 5, and 6 in the array of generalized `qState`
    - The role of $\boldsymbol{q}_j$ in Eq. 3.3.10 is played by the entries 1, 2, and 3 in the array of generalized `qState`
    - The role of $\boldsymbol{q}_i$ in Eq. 3.3.7 is played by the entries 1, 2, and 3 in the array of generalized `qState`
    - The role of $\boldsymbol{q}_i$ in Eq. 3.3.13 is played by the entries 7, 8, and 9 in the array of generalized `qState`
    - The role of $\boldsymbol{q}_j$ in Eq. 3.3.13 is played by the entries 4, 5, and 6 in the array of generalized `qState`
    - The role of $\boldsymbol{q}_i$ in Eq. 3.2.6 is played by the entries 7, 8, and 9 in the array of generalized `qState`

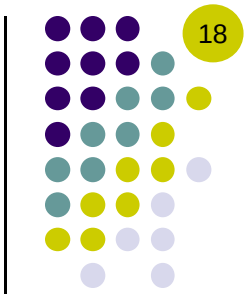
How Various Quantities Look Like

- `qState`, `qdState`, `qddState` have all the same dimension: 9 entries
- `Phi`, `Nu`, `Gamma` have the same dimension: 6
- `Jac` is of dimension 6 by 9

- SIDEBAR: Three more constraints (kinematic or motions) should be prescribed for this example to lead to a NDOF count of zero





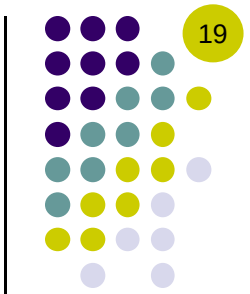


Assembly Issues, Wrap Up

- Why was this discussion needed?
  - Before today, we only looked at one kinematic constraint at a time
  - Never discussed how you put it all together, how to assemble the pieces
  - This assembly is required by `simEngine2D`

- Your `simEngine2D` implementation and the variables you use could be quite different than what we discussed here
  - Adopt and adapt something that makes sense to you

- At this point we covered:
  - How to do kinematic analysis (look also at the MATLAB code provided in lecture of 10/21 and 10/23)
  - How to assemble the entities that are needed to carry out kinematic analysis: `Phi`, `Nu`, `Gamma`, `Jac`



Quick Advice

- This is as heavy duty as it gets in terms of MATLAB use
  - Revisit quickly the MATLAB material we covered in class in ME451 at beginning of semester
  - Learn the basic of MATLAB structures, see for instance
    - http://www.cs.utah.edu/~germain/PPS/Topics/Matlab/structures.html
  - More advanced info about structures, see for instance
    - http://www.mathworks.com/help/matlab/structures.html
- Look at previous years' solution
  - But don't spend too much time there since it's not going to help a lot
- Start simple: implement `simEngine2D` for a simple pendulum and compare against the simple pendulum example in the textbook
  - Move then to mechanism with two bodies for which you have the solution in a textbook example
  - See examples on next two slides, we discussed them in class
- Post questions on the forum

[handout]

Newton-Raphson:
Position Analysis of Mechanism

- Example based on mechanism of Problem 3.5.6, modeled with reduced set of generalized coordinates

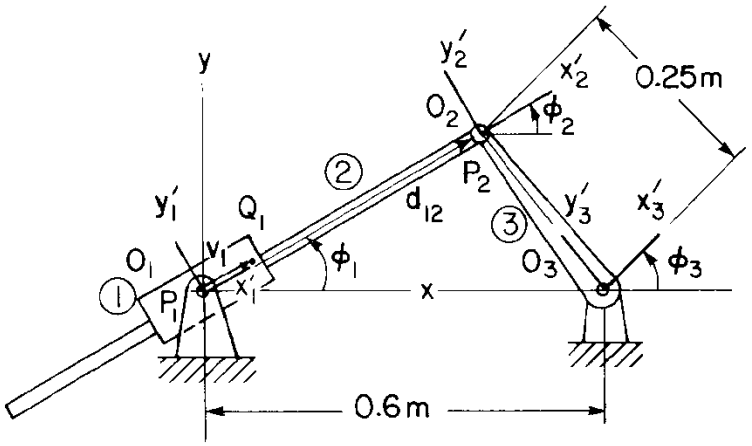


Figure P3.5.6

$$\mathbf{q} = \begin{bmatrix} x_2 \\ y_2 \\ \phi_3 \end{bmatrix}$$

$$\mathbf{\Phi}(\mathbf{q}, t) = \begin{bmatrix} q_1 + 0.25\sin q_3 - 0.6 \\ q_2 - 0.25\cos q_3 \\ q_1^2 + q_2^2 - (t/10 + 0.4)^2 \end{bmatrix} = \mathbf{0}$$

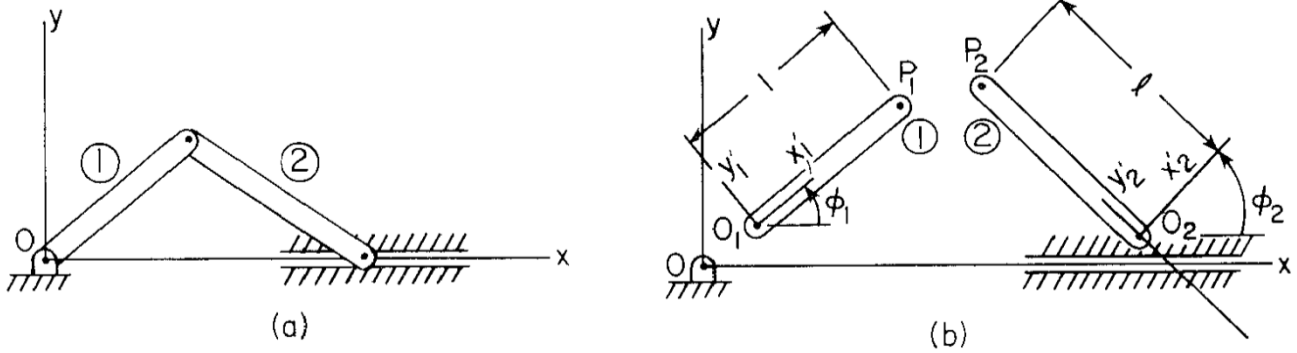
$$\mathbf{\Phi_q}(\mathbf{q}, t) = \begin{bmatrix} 1 & 0 & 0.25\cos q_3 \\ 0 & 1 & 0.25\sin q_3 \\ 2q_1 & 2q_2 & 0 \end{bmatrix}$$

- Newton-Raphson algorithm (at $t = 0$):

$$\mathbf{q}^{(0)} = \begin{bmatrix} 0.2 & 0.2 & \pi/4 \end{bmatrix}^T$$

$$\mathbf{q}^{(i+1)} = \mathbf{q}^{(i)} - \left[\mathbf{\Phi_q}(\mathbf{q}^{(i)}, t)\right]^{-1} \cdot \mathbf{\Phi}(\mathbf{q}^{(i)}, t) \qquad \text{for } i = 1, 2, \ldots$$

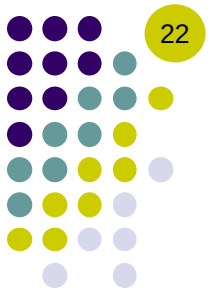
Physical Singular Configurations
[Example 3.7.5]



$$\mathbf{q} = \left[x_2\,,\ \phi_1\,,\ \phi_2\right]$$

$$\mathbf{\Phi}(\mathbf{q},t) = \begin{bmatrix} -x_2 + \cos\phi_1 + l\sin\phi_2 \\ \sin\phi_1 - l\cos\phi_2 \\ \phi_1 - \frac{\pi}{12}t \end{bmatrix} = \mathbf{0} \qquad \nu = \begin{bmatrix} 0 \\ 0 \\ \frac{\pi}{12} \end{bmatrix}$$

$$\mathbf{\Phi_q} = \begin{bmatrix} -1 & -\sin\phi_1 & l\cos\phi_2 \\ 0 & \cos\phi_1 & l\sin\phi_2 \\ 0 & 1 & 0 \end{bmatrix} \qquad \gamma = \begin{bmatrix} \dot{\phi}_1^2\cos\phi_1 + l\dot{\phi}_2^2\sin\phi_2 \\ \dot{\phi}_1^2\sin\phi_1 - l\dot{\phi}_2^2\cos\phi_2 \\ 0 \end{bmatrix}$$



End of Kinematics Analysis

Beginning of Dynamics Analysis