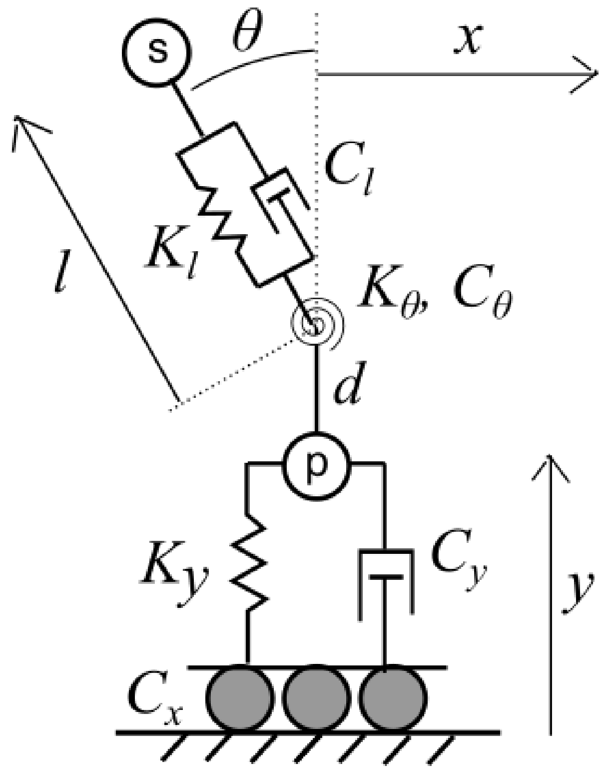
Equations of Motion in dRuBbLe



Setup

First we import the NumPy and SymPy mechanics modules in Python. NumPy provides an extensive set of numerical analysis funtions, while SymPy is a symbolic coding environment with the mechanics module standardizing various kinematic transformations and and equation of motion generation. The IPython display modules are imported to generate typeset equations suitable for document publication. Turning on `mechanics_printing()` will generate more compact typeset equations, where $\frac{\partial x}{\partial t}$ will be written $\dot{x}$, for example.

In [1]:
```
import numpy as np
from sympy import *
from sympy.physics.mechanics import *
from IPython.display import display, Latex
# init_printing(use_latex='mathjax')
mechanics_printing()
```

Generalized Coordinates

To begin the dynamics model formulation, we first define the generalized coordinates. The player is defined as a point mass located at point p, with positions (x, y) defined relative to the origin in the horizontal and vertical directions, respectively. The stool is also defined as a point mass located at point s, which is translated and rotated relative to the player with a transformation (to be described later) requiring an arm extension length l and stool tilt angle θ. These four variables form the generalized coordinate vector $\boldsymbol{q}$.

The SymPy Mechanics `dynamicsymbols` function is used to define these as symbolic variables that are time-varying, and therefore candidate degrees-of-freedom (DOFs). Notably, the left hand side of the equation are the variables to be used in the source code, where `th` is used in place of θ. When typing code that references the tilt angle, `th` will be used, whereas when the `display` function is called to typeset the equation, the more visually appealing θ is used, as seen below. Also note that the display adds the (t) to each variable, indicating it is time-varying. The chained assignment operators (i.e. `a = b = c`) are used such that in the source code the variables `generalized_coords` and `q` can be used interchangeably.

In [55]:
```
x, y, l, th = dynamicsymbols('x y l θ')
generalized_coords = q = Matrix([x, y, l, th])
display(Latex('$\mathbf{q} = \,$' + q._repr_latex_()))
```

$$\mathbf{q} = \left[\begin{matrix} x(t) \\ y(t) \\ l(t) \\ \theta(t) \end{matrix}\right]$$

We will also need the derivatives of these coordinates with respect to time, which are obtained by calling the `dynamicsymbols` function with the second argument `1` indicating the first derivative, and `2` indicating the second derivative.

In [61]:
```
dxdt, dydt, dldt, dthdt = dynamicsymbols('x y l θ', 1)
ddt_generalized_coords = dqdt = Matrix([dxdt, dydt, dldt, dthdt])
d2xdt2, d2ydt2, d2ldt2, d2thdt2 = dynamicsymbols('x y l θ', 2)
d2dt2_generalized_coords = d2qdt2 = Matrix([d2xdt2, d2ydt2, d2ldt2, d2th
dt2])
display(Latex('$\dot{\mathbf{q}} = \,$' + dqdt._repr_latex_()))
display(Latex('$\ddot{\mathbf{q}} = \,$' + d2qdt2._repr_latex_()))
```

$$\dot{\mathbf{q}} = \begin{bmatrix} \frac{d}{dt} x(t) \\ \frac{d}{dt} y(t) \\ \frac{d}{dt} l(t) \\ \frac{d}{dt} \theta(t) \end{bmatrix}$$

$$\ddot{\mathbf{q}} = \begin{bmatrix} \frac{d^2}{dt^2} x(t) \\ \frac{d^2}{dt^2} y(t) \\ \frac{d^2}{dt^2} l(t) \\ \frac{d^2}{dt^2} \theta(t) \end{bmatrix}$$

Initial values for the generalized coordinates are represented by a subscript 0, where $q_0 = (x_0, y_0, l_0, \theta_0)^T$. In SymPy, these are defined using a call to the `symbols` function, which declares them as constants. As before, a slight variation is used for the variable representation in the source code versus in typesetting, where for the latter the underscore is used to ensure that these are typeset as a subscript.

In [67]:
```
x0, y0, l0, th0 = symbols('x_0 y_0 l_0 θ_0')
q0 = Matrix([x0, y0, l0, th0])
display(Latex('$\mathbf{q}_0 = \,$' + q0._repr_latex_()))
```

$$\mathbf{q}_0 = \begin{bmatrix} x_0 \\ y_0 \\ l_0 \\ \theta_0 \end{bmatrix}$$

Constants

Next we define the constants, including masses m_p and m_s of the player and stool, an offset distance d between the player center of mass and the point about which the stool rotates, the gravitational constant $'g'$ (represented by `grav` in source code).

In [5]:
```
mp, ms, d, grav, t = symbols('m_p m_s d g t')
```

The spring coefficients K_y, K_l, and K_θ introduce generalized forces into the system proportional to the subscripted coordinate minus its initial value.

In [6]:
```
Ky, Kl, Kt = symbols('K_y K_l K_θ')
```

Likewise, the damping constants C_x, C_y, C_l, and C_θ induce generalized forces proportional to derivatives with respect to time of the generalized coordinates.

In [68]:
```
Cx, Cy, Cl, Cth = symbols('C_x C_y C_l C_θ')
generalized_damping_forces = Qdamping = Matrix([- Cx * dxdt, - Cy * dydt
, - Cl * dldt, - Cth * dthdt])
display(Latex('$\mathbf{Q}_{damping} = \,$' + Qdamping._repr_latex_()))
```

$$\mathbf{Q}_{damping} = \begin{bmatrix} -C_x \frac{d}{dt} x(t) \\ -C_y \frac{d}{dt} y(t) \\ -C_l \frac{d}{dt} l(t) \\ -C_\theta \frac{d}{dt} \theta(t) \end{bmatrix}$$

The generalized input forces Q_x, Q_y, Q_l, and Q_θ represent the control variables, from either player inputs or the computer control logic algorithm.

In [69]:
```
Qx, Qy, Ql, Qth = symbols('Q_x Q_y Q_l Q_θ')
generalized_input_forces = Qinput = Matrix([Qx, Qy, Ql, Qth])
display(Latex('$\mathbf{Q}_{input} = \,$' + Qinput._repr_latex_()))
```

$$\mathbf{Q}_{input} = \begin{bmatrix} Q_x \\ Q_y \\ Q_l \\ Q_\theta \end{bmatrix}$$

Kinematics

The inertial reference frame is designated as `R`. The default convention in SymPy Mechanics is to represent the reference frame `R` using the unit vectors $\hat{\mathbf{r}}_x$, $\hat{\mathbf{r}}_y$, and $\hat{\mathbf{r}}_z$, however, these default symbols are replaced with unit vectors $\hat{\mathbf{i}}$, $\hat{\mathbf{j}}$, and $\hat{\mathbf{k}}$ for typesetting purposes, where $\hat{\mathbf{i}}$ is horizontal to the right side of the screen, $\hat{\mathbf{j}}$ is vertical and upward, and $\hat{\mathbf{k}}$ is out of the page. It is understood these are spatially fixed such that $\partial\hat{\mathbf{i}}/\partial t = \partial\hat{\mathbf{j}}/\partial t = \partial\hat{\mathbf{k}}/\partial t = 0$. The stool frame is designated as `E`, with the default unit vectors provided by SymPy Mechanics of $\hat{\mathbf{e}}_x$, $\hat{\mathbf{e}}_y$, and $\hat{\mathbf{e}}_z$, which are oriented relative to the reference frame `R` by a rotation of θ about the $\hat{\mathbf{k}}$ axis, where $\hat{\mathbf{k}}$ is accessed in the source code as `R.z`. The direction cosine matrix (DCM) that transforms from the `R` to the `E` frame is given by $\mathbf{R}_e$.

In [66]:
```
R = ReferenceFrame('R')
R.pretty_vecs = ['i', 'j', 'k']
R.latex_vecs = ['\\mathbf{\\hat{i}}', '\\mathbf{\\hat{j}}', '\\mathbf{\\hat{k}}']
E = ReferenceFrame('E')
E.orient(R, 'Axis', [th, R.z])
Re = E.dcm(R)
display(Latex('$\mathbf{R}_e = \,$' + Re._repr_latex_()))
```

$$\mathbf{R}_e = \begin{bmatrix} \cos(\theta(t)) & \sin(\theta(t)) & 0 \\ -\sin(\theta(t)) & \cos(\theta(t)) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The origin is defined as a `Point` designated `O`, and its velocity is set to zero using the `.set_vel()` method. The ensuing points of interest will be defined relative to the origin.

In [10]:
```
origin = Point('O')
origin.set_vel(R, 0)
```

Positions

The player's center of mass is located at the point `p`, which has the position vector $\mathbf{r}_p$ given by the generalized coordinates x and y, measured relative to the origin in the `R` frame. The `.set_pos()` method is used to define this position, with the source code `R.x` and `R.y` to access the unit vectors $\hat{\mathbf{i}}$ and $\hat{\mathbf{j}}$, respectively.

In [70]:
```
player_cm = Point('p')
player_cm.set_pos(origin, x * R.x + y * R.y)
rp = player_cm.pos_from(origin)
display(Latex('$\mathbf{r}_p = \,$' + rp._repr_latex_()))
```

$$\mathbf{r}_p = x\hat{\mathbf{i}} + y\hat{\mathbf{j}}$$

The stool center of mass is located at the point `s`, which has the position vector $\mathbf{r}_s$, which is offset vertically from `p` by a constant distance d in the $\partial\hat{\mathbf{j}}$ direction, and by $l(t)$ in the $\hat{\mathbf{e}}_y$ direction (accessed as `E.y` in the source code).

In [73]:
```
rotation_center = Point('d')
rotation_center.set_pos(player_cm, d * R.y)
stool_cm = Point('s')
stool_cm.set_pos(rotation_center, l * E.y)
rs = stool_cm.pos_from(origin)
display(Latex('$\mathbf{r}_s = \,$' + rs._repr_latex_() + '$\, = \,$' + rs.express(R)._repr_latex_()))
```

$$\mathbf{r}_s = l\hat{\mathbf{e}}_\mathbf{y} + x\hat{\mathbf{i}} + (d + y)\hat{\mathbf{j}} = (-l\sin(\theta) + x)\hat{\mathbf{i}} + (d + l\cos(\theta) + y)\hat{\mathbf{j}}$$

Velocities

The velocity of the point `p`, given by $\mathbf{v}_p$, is calculated by taking its time derivative evaluated in the frame `R`.

In [74]:
```
vp = rp.dt(R)
player_cm.set_vel(R, vp)
display(Latex('$\mathbf{v}_p = \,$' + vp._repr_latex_()))
```

$$\mathbf{v}_p = \dot{x}\hat{\mathbf{i}} + \dot{y}\hat{\mathbf{j}}$$

Similarly, the velocity of the point `s`, given by $\mathbf{v}_s$ is calculated in the same way. By default, the velocity of `s` relative to `p` is given in the rotated stool frame, with a tangential term $l\dot{\theta}\hat{\mathbf{e}}_x$ and radial term $\dot{l}\hat{\mathbf{e}}_y$. The `.express()` method is used to give the equivalent velocity in the inertial $\hat{\mathbf{i}}$ and $\hat{\mathbf{j}}$ coordinates.

In [75]:
```
vs = rs.dt(R)
stool_cm.set_vel(R, vs)
display(Latex('$\mathbf{v}_s = \,$' + vs._repr_latex_() + '$\, = \,$' +
vs.express(R)._repr_latex_()))
```

$$\mathbf{v}_s = -l\dot{\theta}\hat{\mathbf{e}}_{\mathbf{x}} + \dot{l}\hat{\mathbf{e}}_{\mathbf{y}} + \dot{x}\hat{\mathbf{i}} + \dot{y}\hat{\mathbf{j}} = (-l\cos(\theta)\dot{\theta} - \sin(\theta)\dot{l} + \dot{x})\hat{\mathbf{i}} + (-l\sin(\theta)\dot{\theta} + \cos(\theta)\dot{l} + \dot{y})\hat{\mathbf{j}}$$

Lagrange's Method

Energy

The kinetic energy, $T = \frac{1}{2}m_p\mathbf{v}_p \cdot \mathbf{v}_p + \frac{1}{2}m_s\mathbf{v}_s \cdot \mathbf{v}_s$, is calculated where m_p and m_s are the point masses assigned to the player and stool, and $(\cdot)$ represents the dot product.

In [15]:
```
T = simplify(0.5 * mp * dot(vp, vp) + 0.5 * ms * dot(vs, vs))
display(Latex('$T = $'), T)
```

$T =$

$$0.5m_p\dot{x}^2 + 0.5m_p\dot{y}^2 + 0.5m_sl^2\dot{\theta}^2 - 1.0m_sl\sin(\theta)\dot{y}\dot{\theta} - 1.0m_sl\cos(\theta)\dot{x}\dot{\theta} - 1.0m_s\sin(\theta)\dot{l}\dot{x} + 1$$

The potential energy V is the sum of the gravitational potential $(m_p\mathbf{r}_p + m_s\mathbf{r}_\mathbf{s}) \cdot g\hat{\mathbf{j}}$ plus spring potential $K_y(y - y_0)^2$, $K_l(l - l_0)^2$, and $K_\theta\theta^2$.

In [78]:
```
V = dot(mp * rp + ms * rs, grav * R.y) + 0.5 * ( Ky * (y - y0)**2 + Kl *
(l - l0)**2 + Kt * th**2 )
V = simplify(V)
display(Latex('$V = $'), V)
```

$V =$

$$0.5K_l(l_0 - l)^2 + 0.5K_y(y_0 - y)^2 + 0.5K_\theta\theta^2 + gm_sl\cos(\theta) + g\left(m_py + m_s(d + y)\right)$$

The Lagrangian $L = T - V$

In [17]:
```
L = T - V
display(Latex('$L = T - V = $'), L)
```

$L = T - V =$

$$-0.5K_l(l_0 - l)^2 - 0.5K_y(y_0 - y)^2 - 0.5K_\theta\theta^2 - gm_sl\cos(\theta) - g\left(m_py + m_s(d + y)\right) + 0.5$$

$$- 1.0m_sl\cos(\theta)\dot{x}\dot{\theta} - 1.0m_s\sin(\theta)\dot{l}\dot{x} + 1.0m_s\cos(\theta)\dot{l}\dot{y} + 0.5m_s\dot{l}^2 + 0.5m_s\dot{x}^2 + 0.5m_s\dot{y}^2$$

Equations of Motion

Equations of motion are calculated using Lagrange's method where $\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\mathbf{q}}}\right) - \frac{\partial L}{\partial \mathbf{q}} = \mathbf{Q}_{damping} + \mathbf{Q}_{input}$, or equivalently $\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\mathbf{q}}}\right) - \frac{\partial L}{\partial \mathbf{q}} - \mathbf{Q}_{damping} - \mathbf{Q}_{input} = \mathbf{0}$.

In [83]:
```
# Initialize the eom matrix
eom = Matrix([None, None, None, None])

for k in range(4):
    # Apply Lagrange's method
    tmp = diff(diff(L, dqdt[k]), t) - diff(L, q[k]) - Qdamping[k] - Qinp
ut[k]

    # Simplify and insert into eom matrix
    eom[k] = simplify(tmp)

    # Display result
    display(Latex('EOM for ' + q[k]._repr_latex_() + ' is : '), eom[k])
```

EOM for $x(t)$ is :

$$C_x \dot{x} - Q_x + 1.0 m_p \ddot{x} + 1.0 m_s l \sin(\theta)\dot{\theta}^2 - 1.0 m_s l \cos(\theta)\ddot{\theta} - 1.0 m_s \sin(\theta)\ddot{l} - 2.0 m_s \cos(\theta)\dot{l}\dot{\theta}$$

EOM for $y(t)$ is :

$$C_y \dot{y} - 1.0 K_y (y_0 - y) - Q_y + g\left(m_p + m_s\right) + 1.0 m_p \ddot{y} - 1.0 m_s l \sin(\theta)\ddot{\theta} - 1.0 m_s l \cos(\theta)\dot{\theta}^2 -$$

EOM for $l(t)$ is :

$$1.0 C_l \dot{l} - 1.0 K_l l_0 + 1.0 K_l l - 1.0 Q_l + 1.0 g m_s \cos(\theta) - 1.0 m_s l \dot{\theta}^2 - 1.0 m_s \sin(\theta)\ddot{x} + 1.0 m_s \mathrm{c}$$

EOM for $\theta(t)$ is :

$$C_\theta \dot{\theta} + 1.0 K_\theta \theta - Q_\theta - g m_s l \sin(\theta) + 1.0 m_s l^2 \ddot{\theta} - 1.0 m_s l \sin(\theta)\ddot{y} - 1.0 m_s l \cos(\theta)\ddot{x} + 2.0 m_s l \dot{l}$$

Integrable Form

Next solve the equations of motion for the accelerations, $\partial \boldsymbol{q}/\partial t$, using the SymPy `solve` function. This returns a dictionary where the keys are the symbolic variables for $\partial \mathbf{q}/\partial t$. The loop simplifies the solution equations, and places them into a list that is given in the original order of `eom`.

In [89]:

```python
# Solve for the accelerations
eom_solved = solve(eom, d2qdt2)

# Initialize the eoms matrix
eoms = Matrix([None, None, None, None])

for k in range(4):
    # Insert the equation from the eom_solved dictionary indexed using d
2qdt2
    eoms[k] = simplify(eom_solved[d2qdt2[k]])

    # Display the equation
    print(' ')
    display(Latex(d2qdt2[k]._repr_latex_() + '$\, = \,$'), eoms[k])
```

$$\frac{d^2}{dt^2}x(t) =$$

$$\frac{-C_l l \sin(\theta)\dot{l} - C_x l\dot{x} - C_\theta \cos(\theta)\dot{\theta} + K_l l_0 l \sin(\theta) - K_l l^2 \sin(\theta) - K_\theta \theta \cos(\theta) + Q_l l \sin(\theta) +}{m_p l}$$

$$\frac{d^2}{dt^2}y(t) =$$

$$\frac{-gm_p l + \left(-C_\theta\dot{\theta} - K_\theta\theta + Q_\theta\right)\sin(\theta) + \left(C_l \cos(\theta)\dot{l} - C_y\dot{y} - K_l l_0 \cos(\theta) + K_l l \cos(\theta) + K}{m_p l}$$

$$\frac{d^2}{dt^2}l(t) =$$

$$\frac{m_p m_s l\dot{\theta}^2 + m_p\left(-C_l\dot{l} + K_l l_0 - K_l l + Q_l\right) + m_s\left(-C_l\dot{l} - C_x \sin(\theta)\dot{x} + C_y \cos(\theta)\dot{y} + K_l l_0 - K_l l - K_y y_0 \cos(\theta) + K_y y \cos(\theta) + Q_l +}{m_p m_s}$$

$$\frac{d^2}{dt^2}\theta(t) =$$

$$\frac{-C_x m_s l \cos(\theta)\dot{x} - C_y m_s l \sin(\theta)\dot{y} - C_\theta m_p\dot{\theta} - C_\theta m_s\dot{\theta} + K_y m_s y_0 l \sin(\theta) - K_y m_s l y \sin(\theta) - + Q_\theta m_p + Q_\theta m_s - 2.0 m_p m_s l\dot{l}\dot{\theta}}{m_p m_s l^2}$$

Built-in Methods

Generalized Speeds

To facilitate equation of motion generation using Kane's method, we will also define a set of generalized speeds. In this case, we will use substition variables $u = \frac{\partial x}{\partial t}$, $v = \frac{\partial y}{\partial t}$, $w = \frac{\partial l}{\partial t}$, and $\omega = \frac{\partial \theta}{\partial t}$.

In [90]:
```
u, v, w, omeg = dynamicsymbols('u v w ω')
generalized_speeds = s = Matrix([u, v, w, omeg])
dudt, dvdt, dwdt, domegdt = dynamicsymbols('u v w ω', 1)
ddt_generalized_speeds = dsdt = Matrix([dudt, dvdt, dwdt, domegdt])
display(Latex('$\mathbf{s} = \,$' + generalized_speeds._repr_latex_()))
```

$$\mathbf{s} = \begin{bmatrix} u(t) \\ v(t) \\ w(t) \\ \omega(t) \end{bmatrix}$$

A kinematic equation will be used in Kane's method to define the equality $\mathbf{s} = \dot{\mathbf{q}}$, equivalently $\mathbf{s} - \dot{\mathbf{q}} = \mathbf{0}$. The latter convention is commonly used by the SymPy solvers, where a matrix representing a system of equations will be assumed to be in equilibrium, or in other words equal to a matrix of zeros with the same dimension.

In [21]:
```
kinematic_equation = s - dqdt
display(kinematic_equation)
```

$$\begin{bmatrix} u - \dot{x} \\ v - \dot{y} \\ w - \dot{l} \\ \omega - \dot{\theta} \end{bmatrix}$$

Bodies

The player and stool are both defined as `Particle` objects, in other words point masses. The first argument to the class is the name of the particle, second article is a `Point` object corresponding to the centers of mass, and third object is the mass, in this case symbolic. These are added to the `bodies` list.

In [22]:
```
player = Particle('player', player_cm, mp)
stool = Particle('stool', stool_cm, ms)
bodies = [player, stool]
```

Loads

The bodies are acted on by various forces and moments, including from gravity, spring extension, damping, and player input.

In [95]:
```
gravitational_forces = Fg = - mp * grav * R.y, -ms * grav * R.y
spring_forces = Fk = - Ky * (y - y0) * R.y + Kl * (l - l0) * E.y, - Kl *
(l - l0) * E.y
spring_moments = Mk = Kt * th * E.z, - Kt * th * E.z
damping_forces = Fd = - Cx * dxdt * R.x - Cy * dydt * R.y + Cl * dldt *
E.y, - Cl * dldt * E.y
damping_moments = Md = Cth * dthdt * E.z, - Cth * dthdt * E.z
input_forces = Fi = Qx * R.x + Qy * R.y - Ql * E.y, Ql * E.y
input_moments = Mi = - Qth * E.z, Qth * E.z
loads = [(player_cm, Fg[0] + Fk[0] + Fd[0] + Fi[0]),  # Forces on the pl
ayer
         (stool_cm, Fg[1] + Fk[1] + Fd[1] + Fi[1]),   # Forces on the st
ool
         (R, Mk[0] + Md[0] + Mi[0]),                  # Moments on the p
layer, R-frame
         (E, Mk[1] + Md[1] + Mi[1])]                  # Moments on the s
tool, E-frame
display(Latex('Gravitational Forces : '), Fg)
display(Latex('Spring Forces : '), Fk)
display(Latex('Spring Moments : '), Mk)
display(Latex('Damping Forces : '), Fd)
display(Latex('Damping Moments : '), Md)
display(Latex('Input Forces : '), Fi)
display(Latex('Input Moments : '), Mi)
display(Latex('Net Forces and Moments: '), loads)
```

Gravitational Forces :

$$\left(-gm_p\hat{\mathbf{j}},\ -gm_s\hat{\mathbf{j}}\right)$$

Spring Forces :

$$\left(-K_y\left(-y_0+y\right)\hat{\mathbf{j}}+K_l\left(-l_0+l\right)\hat{\mathbf{e}}_{\mathbf{y}},\ -K_l\left(-l_0+l\right)\hat{\mathbf{e}}_{\mathbf{y}}\right)$$

Spring Moments :

$$\left(K_\theta\theta\hat{\mathbf{e}}_{\mathbf{z}},\ -K_\theta\theta\hat{\mathbf{e}}_{\mathbf{z}}\right)$$

Damping Forces :

$$\left(-C_x\dot{x}\hat{\mathbf{i}}-C_y\dot{y}\hat{\mathbf{j}}+C_l\dot{l}\hat{\mathbf{e}}_{\mathbf{y}},\ -C_l\dot{l}\hat{\mathbf{e}}_{\mathbf{y}}\right)$$

Damping Moments :

$$\left(C_\theta\dot{\theta}\hat{\mathbf{e}}_{\mathbf{z}},\ -C_\theta\dot{\theta}\hat{\mathbf{e}}_{\mathbf{z}}\right)$$

Input Forces :

$$\left(Q_x\hat{\mathbf{i}}+Q_y\hat{\mathbf{j}}-Q_l\hat{\mathbf{e}}_{\mathbf{y}},\ Q_l\hat{\mathbf{e}}_{\mathbf{y}}\right)$$

Input Moments :

$$\left(-Q_\theta\hat{\mathbf{e}}_{\mathbf{z}},\ Q_\theta\hat{\mathbf{e}}_{\mathbf{z}}\right)$$

Net Forces and Moments:

```
[(p,
  (-C_x*x' + Q_x)*R.x + (-C_y*y' - K_y*(-y_0 + y) + Q_y - g*m_p)*R.y +
(C_l*l' + K_l*(-l_0 + l) - Q_l)*E.y),
 (s, - g*m_s*R.y + (-C_l*l' - K_l*(-l_0 + l) + Q_l)*E.y),
 (R, (C_θ*θ' + K_θ*θ - Q_θ)*E.z),
 (E, (-C_θ*θ' - K_θ*θ + Q_θ)*E.z)]
```

Kane's Equations

In [24]:
```
kane = KanesMethod(R, q_ind=q, u_ind=s, kd_eqs=kinematic_equation)
fr, frstar = kane.kanes_equations(bodies, loads)
display(Latex('$\mathbf{F}_r = $'), fr)
display(Latex('$\mathbf{F}_r^* = $'), frstar)
display(Latex('$\mathbf{M} = $'), kane.mass_matrix)
display(Latex('$\mathbf{F} = $'), kane.forcing)
display(Latex('where $\mathbf{M}\mathbf{\dot{s}} = \mathbf{F}$'))
```

$\mathbf{F}_r =$

$$\left[\begin{matrix} -C_x\dot{x} + Q_x - \left(-C_l\dot{l} - K_l\left(-l_0 + l\right) + Q_l\right)\sin(\theta) - \left(C_l\dot{l} + K_l\left(-l_0 + l\right) - \right. \\ -C_y\dot{y} - K_y\left(-y_0 + y\right) + Q_y - gm_p - gm_s + \left(-C_l\dot{l} - K_l\left(-l_0 + l\right) + Q_l\right)\cos(\theta) + \left(C_l\dot{l} + \right. \\ -C_l\dot{l} - K_l\left(-l_0 + l\right) + Q_l - gm_s\cos(\theta) \\ -C_\theta\dot{\theta} - K_\theta\theta + Q_\theta + gm_s l\sin(\theta) \end{matrix}\right.$$

$\mathbf{F}_r^* =$

$$\left[\begin{matrix} -m_s l\omega^2\sin(\theta) + m_s l\cos(\theta)\dot{\omega} + 2m_s w\omega\cos(\theta) + m_s\sin(\theta)\dot{w} - \left(m_p + m_s\right)\dot{u} \\ m_s l\omega^2\cos(\theta) + m_s l\sin(\theta)\dot{\omega} + 2m_s w\omega\sin(\theta) - m_s\cos(\theta)\dot{w} - \left(m_p + m_s\right)\dot{v} \\ m_s l\omega^2 + m_s\sin(\theta)\dot{u} - m_s\cos(\theta)\dot{v} - m_s\dot{w} \\ -m_s l^2\dot{\omega} - 2m_s lw\omega + m_s l\sin(\theta)\dot{v} + m_s l\cos(\theta)\dot{u} \end{matrix}\right]$$

$\mathbf{M} =$

$$\left[\begin{matrix} m_p + m_s & 0 & -m_s\sin(\theta) & -m_s l\cos(\theta) \\ 0 & m_p + m_s & m_s\cos(\theta) & -m_s l\sin(\theta) \\ -m_s\sin(\theta) & m_s\cos(\theta) & m_s & 0 \\ -m_s l\cos(\theta) & -m_s l\sin(\theta) & 0 & m_s l^2 \end{matrix}\right]$$

$\mathbf{F} =$

$$\left[\begin{matrix} -C_x\dot{x} + Q_x - m_s l\omega^2\sin(\theta) + 2m_s w\omega\cos(\theta) - \left(-C_l\dot{l} - K_l\left(-l_0 + l\right) + Q_l\right) \\ -C_y\dot{y} - K_y\left(-y_0 + y\right) + Q_y - gm_p - gm_s + m_s l\omega^2\cos(\theta) + 2m_s w\omega\sin(\theta) + \left(-C_l\dot{l} - K_l\right. \\ (\theta) \\ -C_l\dot{l} - K_l\left(-l_0 + l\right) + Q_l - gm_s\cos(\theta) \text{ -} \\ -C_\theta\dot{\theta} - K_\theta\theta + Q_\theta + gm_s l\sin(\theta) - 2m \end{matrix}\right.$$

where $\mathbf{M}\mathbf{\dot{s}} = \mathbf{F}$

In [97]:
```
# Solve for the accelerations (derivatives of generalized speeds)
kane_solved = solve(fr + frstar, dsdt)

#from pydy.system import System

# Initialize the kanes matrix
kanes = Matrix([None, None, None, None])

for k in range(4):
    # Insert the equation from the eom_solved dictionary indexed using d
2qdt2
    kanes[k] = simplify(kane_solved[dsdt[k]])

    # Display the equation
    display(Latex(dsdt[k]._repr_latex_() + '\, = \,'), kanes[k])
```

$$\frac{d}{dt}u(t) =$$

$$\frac{-C_l l \sin(\theta)\dot{l} - C_x l\dot{x} - C_\theta \cos(\theta)\dot{\theta} + K_l l_0 l \sin(\theta) - K_l l^2 \sin(\theta) - K_\theta \theta \cos(\theta) + Q_l l \sin(\theta) +}{m_p l}$$

$$\frac{d}{dt}v(t) =$$

$$\frac{-g m_p l + \left(-C_\theta \dot{\theta} - K_\theta \theta + Q_\theta\right) \sin(\theta) + \left(C_l \cos(\theta)\dot{l} - C_y \dot{y} - K_l l_0 \cos(\theta) + K_l l \cos(\theta) + K\right.}{m_p l}$$

$$\frac{d}{dt}w(t) =$$

$$\frac{m_p m_s l\omega^2 + m_p \left(-C_l \dot{l} + K_l l_0 - K_l l + Q_l\right) + m_s \left(-C_l \dot{l} - C_x \sin(\theta)\dot{x} + C_y \cos(\theta)\dot{y} + K_l l_0 - K_l l - K_y y_0 \cos(\theta) + K_y y \cos(\theta) + Q_l + \right.}{m_p m_s}$$

$$\frac{d}{dt}\omega(t) =$$

$$\frac{-C_x m_s l \cos(\theta)\dot{x} - C_y m_s l \sin(\theta)\dot{y} - C_\theta m_p \dot{\theta} - C_\theta m_s \dot{\theta} + K_y m_s y_0 l \sin(\theta) - K_y m_s l y \sin(\theta) - + Q_\theta m_p + Q_\theta m_s - 2 m_p m_s l w \omega}{m_p m_s l^2}$$

In [99]:
```
display(simplify(kanes-eoms).subs(s[2], dqdt[2]).subs(s[3], dqdt[3]))
```

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Code Generation

Next we substitute variables into the equation to create printed text that can be inserted into ordinary Python source code. The printed output represents the code to be implemented in a function of the form `du = f(u)` , where `u` is the list of state variables, and `du` are the derivatives of `u` with respect to time. Additionally, this code presumes the existence of a parameter structure `p` containing the constants, and generalized force variables `Qx, Qy, Ql, Qth` .

This is done in two steps, first using the `.subs()` method which replaces symbolic variables in the equation, but returns an object of the symbolic data type. The second step is to change to a string data type, then use the `.replace()` method to swap in new strings that are shorter and/or more compatible with standard Python source code. It is worth noting that this entire operation could be using the second, string replacement method, but the symbolic substition is inclued for demonstration purposes.

The process begins by defining symbolic or string replacements. The symbols in `dq` are shortened form for the dynamic symbols previously created in the `dqdt` . Where the latter would print long strings intended for typeset display (i.e. `Derivative(x(t), t)` which is typeset as $\frac{\partial}{\partial t} x(t)$), the replacement are 2 or 3 ascii characters (i.e. `dx`).

The list `qstr` are simply the named set of state variables in the list `u` , which are included for readability of the string formatted equations of motion.

The string replacements are defined in `replist` , which is a $N \times 2$ list. In each row, the first entry (`replist[n][0]`) is the "old string", or expected pattern in the original string output, and the second entry (`replist[n][1]`) is the "new string", which replaces it.

In [27]:

```
# Define shortened symbols to replace using the subs command
dx, dy, dl, dth = symbols('dx dy dl dθ')
dq = Matrix([dx, dy, dl, dth])

# Define the variable names that are included in the list 'u'
qstr = 'x', 'y', 'l', 'th', 'dx', 'dy', 'dl', 'dth'

# Define the list of variables to replace in the string
replist = [['sin(θ(t))', 's'], ['cos(θ(t))', 'c'],
           ['x(t)', 'x'], ['y(t)', 'y'], ['l(t)', 'l'], ['θ(t)', 'th'],
['dθ', 'dth'],
           ['x_0', 'p.x0'], ['y_0', 'p.y0'], ['l_0', 'p.l0'],
           ['m_p', 'p.mp'], ['m_s', 'p.ms'], ['g', 'p.g'],
           ['K_x', 'p.Kx'], ['K_y', 'p.Ky'], ['K_l', 'p.Kl'], ['K_θ',
'p.Kth'],
           ['C_x', 'p.Cx'], ['C_y', 'p.Cy'], ['C_l', 'p.Cl'], ['C_θ',
'p.Cth'],
           ['Q_x', 'Qx'], ['Q_y', 'Qy'],['Q_l', 'Ql'], ['Q_θ', 'Qth']]

# Initialize the string formatted EOMs
eoms_print = ['du[4] = ', 'du[5] = ', 'du[6] = ', 'du[7] = ']

# Print string that unpacks the state variables
for k in range(4):
    print(qstr[k], '= u[' + str(k) + ']')
    print(' ')

# Print the shortened sine and cosine
print('s = sin(th)')
print(' ')
print('c = cos(th)')
print(' ')

# Print the velocity assignments
for k in range(4):
    print('du[' + str(k) + '] =', qstr[k+4], '= u[' + str(k+4) + ']')
    print(' ')

for k in range(4):
    # Load a copy of the eoms into the tmp_symbolic variable
    tmp_symbolic = eoms[k]

    # Replace the velocity terms with shortened variable
    # names (i.e. Derivative(x(t), t) replaced with dx)
    for j in range(4):
        tmp_symbolic = tmp_symbolic.subs(dqdt[j], dq[j])

    # Create a temporary string formatted equation
    tmp_string = str(tmp_symbolic)

    # Replace old_string with new_string from the replist
    for old_string, new_string in replist:
        tmp_string = tmp_string.replace(old_string, new_string)

    # Replace the print string with tmp, and print result
    eoms_print[k] += tmp_string
```

```
    print(eoms_print[k])
    print(' ')
```

```
x = u[0]

y = u[1]

l = u[2]

th = u[3]

s = sin(th)

c = cos(th)

du[0] = dx = u[4]

du[1] = dy = u[5]

du[2] = dl = u[6]

du[3] = dth = u[7]

du[4] = (-p.Cl*dl*l*s - p.Cx*dx*l - p.Cth*dth*c + p.Kl*p.l0*l*s - p.Kl*
l**2*s - p.Kth*th*c + Ql*l*s + Qx*l + Qth*c)/(p.mp*l)

du[5] = (-p.g*p.mp*l + (-p.Cth*dth - p.Kth*th + Qth)*s + (p.Cl*dl*c -
p.Cy*dy - p.Kl*p.l0*c + p.Kl*l*c + p.Ky*p.y0 - p.Ky*y - Ql*c + Qy)*l)/
(p.mp*l)

du[6] = (dth**2*p.mp*p.ms*l + p.mp*(-p.Cl*dl + p.Kl*p.l0 - p.Kl*l + Ql)
+ p.ms*(-p.Cl*dl - p.Cx*dx*s + p.Cy*dy*c + p.Kl*p.l0 - p.Kl*l - p.Ky*p.
y0*c + p.Ky*y*c + Ql + Qx*s - Qy*c))/(p.mp*p.ms)

du[7] = (-p.Cx*dx*p.ms*l*c - p.Cy*dy*p.ms*l*s - p.Cth*dth*p.mp - p.Cth*
dth*p.ms + p.Ky*p.ms*p.y0*l*s - p.Ky*p.ms*l*y*s - p.Kth*p.mp*th - p.Kth
*p.ms*th + Qx*p.ms*l*c + Qy*p.ms*l*s + Qth*p.mp + Qth*p.ms - 2.0*dl*dth
*p.mp*p.ms*l)/(p.mp*p.ms*l**2)
```

Mass Matrix

The mass matrix is calculated by taking the derivative of each equation of motion with respect to each acceleration term. The primary terms are found along the diagonal, the first three reflect the player and stool mass for the translational DOFs, and the fourth represents the effective moment of inertia for the stool mass eccentricity from its rotation center. The mass matrix inverse is also calculated.

In [28]:
```
M = Matrix([[None, None, None, None],
            [None, None, None, None],
            [None, None, None, None],
            [None, None, None, None]])

for j in range(4):
    for k in range(4):
        M[j, k] = diff(eom[j], d2qdt2[k])

display(Latex('$\mathbf{M} = $'), M)
display(Latex('$\mathbf{M}^{-1} = $'), simplify(M.inv()))
```

$\mathbf{M} =$

$$\begin{bmatrix} 1.0m_p + 1.0m_s & 0 & -1.0m_s\sin(\theta) & -1.0m_s l\cos(\theta) \\ 0 & 1.0m_p + 1.0m_s & 1.0m_s\cos(\theta) & -1.0m_s l\sin(\theta) \\ -1.0m_s\sin(\theta) & 1.0m_s\cos(\theta) & 1.0m_s & 0 \\ -1.0m_s l\cos(\theta) & -1.0m_s l\sin(\theta) & 0 & 1.0m_s l^2 \end{bmatrix}$$

$\mathbf{M}^{-1} =$

$$\begin{bmatrix} \frac{1.0}{m_p} & 0 & \frac{1.0\sin(\theta)}{m_p} & \frac{1.0\cos(\theta)}{m_p l} \\ 0 & \frac{1.0}{m_p} & -\frac{1.0\cos(\theta)}{m_p} & \frac{1.0\sin(\theta)}{m_p l} \\ \frac{1.0\sin(\theta)}{m_p} & -\frac{1.0\cos(\theta)}{m_p} & \frac{1.0}{m_s} + \frac{1.0}{m_p} & 0 \\ \frac{1.0\cos(\theta)}{m_p l} & \frac{1.0\sin(\theta)}{m_p l} & 0 & \frac{1.0(m_p+m_s)}{m_p m_s l^2} \end{bmatrix}$$