%% [markdown]

Linearization: Crane System Study

In [ ]:

```
# %%

import sympy
from sympy import *
from sympy.physics.vector.printing import vlatex
from IPython.display import Math, display

init_printing()

def dotprint(expr):
    display(Math(vlatex(expr)))
```

In [ ]:

```
# %%

t = symbols('t')
# Generalized coordinates
z, h, theta = symbols(r'z, h, \theta', cls=Function)
z = z(t)
h = h(t)
theta = theta(t)

z_dot = z.diff(t)
h_dot = h.diff(t)
theta_dot = theta.diff(t)

z_ddot = z.diff(t,2)
h_ddot = h.diff(t,2)
theta_ddot = theta.diff(t,2)

mc, mr, Jc, d, mu, g, F, tau = symbols(r'm_c, m_r, J_c, d, \mu, g, F, \tau', real=True)
```

In [ ]:

```
# %%

M = Matrix([[mc+2*mr, 0, 0],
            [0, mc+2*mr, 0],
            [0, 0, Jc+2*mr*d**2]])

RHS = Matrix([[-F*sin(theta)-mu*z_dot],
              [-(mc+2*mr)*g+F*cos(theta)],
              [tau]])

EOM_dd = M.inv()*RHS
dynamics = Eq(Matrix([z_ddot, h_ddot, theta_ddot]), EOM_dd)
dotprint(dynamics)
```

$\displaystyle \left[\begin{matrix}\ddot{z}\\\ddot{h}\\\ddot{\theta}\end{matrix}\right] = \left[\begin{matrix}\frac{- F \sin{\left(\theta \right)} - \mu \dot{z}}{m_{c} + 2 m_{r}}\\\frac{F \cos{\left(\theta \right)} + g \left(- m_{c} - 2 m_{r}\right)}{m_{c} + 2 m_{r}}\\\frac{\tau}{J_{c} + 2 d^{2} m_{r}}\end{matrix}\right]$

%% [markdown]

Deriving Nonlinear State Space Equations

Solve for our highest derivatives, $\ddot{z}, \ddot{h}, \ddot{\theta}$:

In [ ]:

```
# %%

solve_dict = solve(dynamics, (z_ddot, h_ddot, theta_ddot), simplify=True, dict=True)[0]
dotprint(solve_dict)
```

$\displaystyle \left\{ \ddot{\theta} : \frac{\tau}{J_{c} + 2 d^{2} m_{r}}, \ \ddot{h} : \frac{F \cos{\left(\theta \right)} - g m_{c} - 2 g m_{r}}{m_{c} + 2 m_{r}}, \ \ddot{z} : - \frac{F \sin{\left(\theta \right)} + \mu \dot{z}}{m_{c} + 2 m_{r}}\right\}$

%% [markdown]

Create our state vector $x = [x_1, x_2, x_3, x_4, x_5, x_6]$:

In [ ]:

```
# %%

state = MatrixSymbol('x', 6, 1)
dotprint(state)
```

$\displaystyle x$

%% [markdown]

We have $x = [x_1, x_2, x_3, x_4, x_5, x_6] = [z, h, \theta, \dot{z}, \dot{h}, \dot{\theta}]$, so $\dot{x} = [\dot{z}, \dot{h}, \dot{\theta}, \ddot{z}, \ddot{h}, \ddot{\theta}]$. Let's put that in a vector:

In [ ]:

```
# %%

state_deriv = Matrix([
    z_dot,
    h_dot,
    theta_dot,
    solve_dict[z_ddot],
    solve_dict[h_ddot],
    solve_dict[theta_ddot]
])
dotprint(state_deriv)
```

$\displaystyle \left[\begin{matrix}\dot{z}\\\dot{h}\\\dot{\theta}\\- \frac{F \sin{\left(\theta \right)} + \mu \dot{z}}{m_{c} + 2 m_{r}}\\\frac{F \cos{\left(\theta \right)} - g m_{c} - 2 g m_{r}}{m_{c} + 2 m_{r}}\\\frac{\tau}{J_{c} + 2 d^{2} m_{r}}\end{matrix}\right]$

%% [markdown]

Now we can substitute in our $x_1, x_2, x_3, x_4, x_5, x_6$ values:

In [ ]:

```
# %%

# Dictionary for substitutions
subs_dict = {
    z: state[0],
    h: state[1],
    theta: state[2],
    z_dot: state[3],
    h_dot: state[4],
    theta_dot: state[5]
}

f_expr = state_deriv.subs(subs_dict)
dotprint(f_expr)
```

$\displaystyle \left[\begin{matrix}{x}_{3,0}\\{x}_{4,0}\\{x}_{5,0}\\- \frac{F \sin{\left({x}_{2,0} \right)} + \mu {x}_{3,0}}{m_{c} + 2 m_{r}}\\\frac{F \cos{\left({x}_{2,0} \right)} - g m_{c} - 2 g m_{r}}{m_{c} + 2 m_{r}}\\\frac{\tau}{J_{c} + 2 d^{2} m_{r}}\end{matrix}\right]$

%% [markdown]

We now have our nonlinear state space equations!

Linearization

First, we need to find an equilibrium point. We can either do this by hand or use Sympy's `solve` function to do this.

Note: It's a good idea to practice doing this by hand in addition to using Sympy!

In [ ]:
```
# %%

# This equation represents f(x,u) = 0
equilibrium_equation = Eq(f_expr, Matrix([0, 0, 0, 0, 0, 0]))

eq_solve_dict = solve(equilibrium_equation, (state[0], state[1], state[2], state[3], state[4], state[5], F, tau), simplify=True, dict=True)[0]
dotprint(eq_solve_dict)
```

$\displaystyle \left\{ F : - g \left(m_{c} + 2 m_{r}\right), \ \tau : 0, \ {x}_{2,0} : \pi, \ {x}_{3,0} : 0, \ {x}_{4,0} : 0, \ {x}_{5,0} : 0\right\}$

%% [markdown]

We can see that at equilibrium:

1. z and h can be any value
2. $\theta = 0$ or π (we'll use 0)
3. All velocities are zero
4. $F = (m_c + 2m_r)g$
5. $\tau = 0$

Below we define our equilibrium points $x_e = [z_e, h_e, \theta_e, \dot{z}_e, \dot{h}_e, \dot{\theta}_e]$ and $u_e = [F_e, \tau_e]$.

In [ ]:
```
# %%

z_e, h_e = symbols('z_e h_e')
u_eq = Matrix([(mc + 2*mr)*g, 0])
theta_eq = 0
z_dot_eq = h_dot_eq = theta_dot_eq = 0
```

%% [markdown]

Define A, B Jacobians

We can use Sympy's `jacobian` function to find the jacobians of `f(x,u)`.

First we find $A = \frac{\partial f}{\partial x}$:

In [ ]:
```
# %%

A = f_expr.jacobian(state)
dotprint(A)
```

$\displaystyle \left[\begin{matrix}0 & 0 & 0 & 1 & 0 & 0\\0 & 0 & 0 & 0 & 1 & 0\\0 & 0 & 0 & 0 & 0 & 1\\0 & 0 & - \frac{F \cos{\left({x}_{2,0} \right)}}{m_{c} + 2 m_{r}} & - \frac{\mu}{m_{c} + 2 m_{r}} & 0 & 0\\0 & 0 & - \frac{F \sin{\left({x}_{2,0} \right)}}{m_{c} + 2 m_{r}} & 0 & 0 & 0\\0 & 0 & 0 & 0 & 0 & 0\end{matrix}\right]$

%% [markdown]

We can evaluate at our specific (x_e, u_e) point using `subs`.

In [ ]:

```
# %%

A_subs = {
    state[0]: z_e,
    state[1]: h_e,
    state[2]: theta_eq,
    state[3]: z_dot_eq,
    state[4]: h_dot_eq,
    state[5]: theta_dot_eq,
    F: u_eq[0],
    tau: u_eq[1]
}

A_eq = A.subs(A_subs)
dotprint(A_eq)
```

$\displaystyle \left[\begin{matrix}0 & 0 & 0 & 1 & 0 & 0\\0 & 0 & 0 & 0 & 1 & 0\\0 & 0 & 0 & 0 & 0 & 1\\0 & 0 & - g & - \frac{\mu}{m_{c} + 2 m_{r}} & 0 & 0\\0 & 0 & 0 & 0 & 0 & 0\\0 & 0 & 0 & 0 & 0 & 0\end{matrix}\right]$

%% [markdown]

Now we do a similar process to find $B = \frac{\partial f}{\partial u}$

In [ ]:

```
# %%

B = f_expr.jacobian(Matrix([F, tau]))
dotprint(B)
```

$\displaystyle \left[\begin{matrix}0 & 0\\0 & 0\\0 & 0\\- \frac{\sin{\left({x}_{2,0} \right)}}{m_{c} + 2 m_{r}} & 0\\\frac{\cos{\left({x}_{2,0} \right)}}{m_{c} + 2 m_{r}} & 0\\0 & \frac{1}{J_{c} + 2 d^{2} m_{r}}\end{matrix}\right]$

%% [markdown]

Like for the A matrix, we need to substitute in our equilibrium values.

In [ ]:

```
# %%

B_subs = {
    state[0]: z_e,
    state[1]: h_e,
    state[2]: theta_eq,
    state[3]: z_dot_eq,
    state[4]: h_dot_eq,
    state[5]: theta_dot_eq,
    F: u_eq[0],
    tau: u_eq[1]
}

B_eq = B.subs(B_subs)
dotprint(B_eq)
```

$\displaystyle \left[\begin{matrix}0 & 0\\0 & 0\\0 & 0\\0 & 0\\\frac{1}{m_{c} + 2 m_{r}} & 0\\0 & \frac{1}{J_{c} + 2 d^{2} m_{r}}\end{matrix}\right]$

%% [markdown]

Decoupled Dynamics

We can separate the longitudinal and lateral dynamics:

In [ ]:

%%

A_lin_lon = A_eq[[1, 4], [1, 4]]
B_lin_lon = B_eq[[1, 4], [0]]

print("Linearized A matrix for longitudinal dynamics:")
dotprint(A_lin_lon)

print("Linearized B matrix for longitudinal dynamics:")
dotprint(B_lin_lon)

A_lin_lat = A_eq[[0, 2, 3, 5], [0, 2, 3, 5]]
B_lin_lat = B_eq[[0, 2, 3, 5], [1]]

print("Linearized A matrix for lateral dynamics:")
dotprint(A_lin_lat)

print("Linearized B matrix for lateral dynamics:")
dotprint(B_lin_lat)

Linearized A matrix for longitudinal dynamics:
$\displaystyle \left[\begin{matrix}0 & 1\\0 & 0\end{matrix}\right]$
Linearized B matrix for longitudinal dynamics:
$\displaystyle \left[\begin{matrix}0\\\frac{1}{m_{c} + 2 m_{r}}\end{matrix}\right]$
Linearized A matrix for lateral dynamics:
$\displaystyle \left[\begin{matrix}0 & 0 & 1 & 0\\0 & 0 & 0 & 1\\0 & - g & - \frac{\mu}{m_{c} + 2 m_{r}} & 0\\0 & 0 & 0 & 0\end{matrix}\right]$
Linearized B matrix for lateral dynamics:
$\displaystyle \left[\begin{matrix}0\\0\\0\\\frac{1}{J_{c} + 2 d^{2} m_{r}}\end{matrix}\right]$

%% [markdown]

Final Equations

To obtain our final equations of motion, we define new variables measuring our offset from the equilibrium:

$$ \begin{align*} \tilde{x} &= x - x_e \\ \tilde{u} &= u - u_e \end{align*} $$

Our final equations of motion are:

$$ \dot{\tilde{x}} = A\tilde{x} + B\tilde{u} $$

Where A and B are the linearized matrices we derived above, and the system is decoupled into longitudinal and lateral dynamics.