The Euler-Lagrangian of the Ein Concept LaTeX

Robert Brothers Mechanical Engineering Student @ UTSA

November 9, 2014

Abstract

Using the Denavit-Hartenberg Parameters the Lagrangian of Ein was Calculated to be L

1 Question: The Lagrangian

1.1 question about the lagrangian

when i take the derivative of the lagrangian w/ respect to q do i take the derivative of each q? and are the dependent on one another? if i differentiate $q_1 q_2$ with respect to q_1 what do i get?

$$M_i = \begin{bmatrix} m_i & 0 & 0 \\ 0 & m_i & 0 \\ 0 & 0 & m_i \end{bmatrix} \tag{1}$$

$$I_{b,i} = \begin{bmatrix} I_x & 0 & 0 \\ 0 & I_y & 0 \\ 0 & 0 & I_z \end{bmatrix} \tag{2}$$

$$L = K - P \tag{3}$$

$$K = \frac{M_i V_i^2}{2} = \frac{1}{2}\dot{q}^T \left[\sum_{i=1}^{n} M_i J_{v,i}^T J_{v,i} + J_{\omega,i}^T R_i I_{b,i} R_i^T J_{\omega,i} \right] \dot{q} \tag{4}$$

$$J_i = \begin{bmatrix} J_{v,i} \\ J_{\omega,i} \end{bmatrix} = \begin{bmatrix} J_{v_0} & J_{v_1} & \dots & J_{v_n} \\ J_{\omega_0} & J_{\omega_1} & \dots & J_{\omega_n} \end{bmatrix} \tag{5}$$

$$P = M_i g h = \sum_{i=1}^{n} \vec{g}^T M_i \vec{r}_{c,i} \tag{6}$$

2 Results

2.1 Denavit-Hartenberg Parameters

Link$_i$	a$_i$	α_i	d$_i$	θ_i
1	0	$\frac{\pi}{2}$	L$_t$	θ_t
2	0	0	L$_s$	0

2.2 Jacobian

2.2.1 Jacobian of Link One's Center of Mass

$$J_1 = \begin{bmatrix} 0.5l_1 \sin(q_1) & 0 \\ -0.5l_1 \cos(q_1) & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \end{bmatrix} \tag{7}$$

2.2.2 Jacobian of Link Two's Center of Mass

$$J_2 = \begin{bmatrix} 0 & 1.0 \sin(q_1) \\ 0 & -1.0 \cos(q_1) \\ 1 & 6.12323399573677 \cdot 10^{-17} \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{8}$$

2.3 Lagrangian Results

$$L = 0.5gl_1m_1 \sin(q_1) + gm_2 \left(0.5l_2\right.$$
$$+ 1.0q_2 \cos(q_1) + 0.5m_2\dot{q}_2 \left(6.12323399573677 \cdot 10^{-17}\dot{q}_1\right.$$
$$+ 1.0\dot{q}_2 + 0.5\dot{q}_1 \left(6.12323399573677 \cdot 10^{-17}m_2\dot{q}_2\right.$$
$$+ \dot{q}_1 \left(0.25l_1^2m_1 + 1.04719755119713m_1r_1^2\right.$$
$$+ 1.0m_2$$

2.4 Torque: Derived from the Lagrangian

$$\tau_k = -0.5gl_1m_1 \cos(q_1) + gm_2 \left(0.5l_2\right.$$
$$+ 1.0q_2 \sin(q_1) \;-\; 1.0gm_2 \cos(q_1)$$
$$+ 1.0m_2\ddot{q}_2 + \ddot{q}_1 \left(0.25l_1^2m_1\right.$$
$$+ 1.04719755119713m_1r_1^2 + 1.0m_2$$

```latex
\documentclass[10pt]{article}
\usepackage{setspace}
\usepackage{amsmath,amsfonts,amsthm,amssymb}
\usepackage{color}
\usepackage{fancyhdr}
\usepackage{chngpage}
\usepackage{enumerate}
\usepackage{graphicx}
\usepackage{boxedminipage}

\title{The Euler-Lagrangian of the Ein Concept \LaTeX}
\author{Robert Brothers Mechanical Engineering Student @ UTSA}
\begin{document}
\doublespacing
\footnotesize
\maketitle
\date

\abstract{Using the Denavit-Hartenberg Parameters the Lagrangian of Ein was Calculat
ed to be L}

\section{Question: The Lagrangian}
\subsection{question about the lagrangian}
when i take the derivative of the lagrangian w/ respect to q
do i take the derivative of each q?
and are the dependent on one another?
if i differentiate $q_{1}q_{2}$ with respect to $q_{1}$ what do i get?

\begin{equation}
M_{i} =
\begin{bmatrix}
m_{i} & 0 & 0 \\
0 & m_{i} & 0 \\
0 & 0 & m_{i} \\
\end{bmatrix}
\end{equation}

\begin{equation}
I_{b,i} =
\begin{bmatrix}
I_{x} & 0 & 0 \\
0 & I_{y} & 0 \\
0 & 0 & I_{z} \\
\end{bmatrix}
\end{equation}



\begin{equation}
L & = & K & - & P
\end{equation}

\begin{equation}
K & = & \frac{M_{i}V_{i}^{2}}{2} =
\frac{1}{2}\dot{q^{T}}\left[\sum\limits_{i=1}^{n} M_{i}J_{v,i}^{T}J_{v,i} + \\
J_{\omega,i}^{T}R_{i}I_{b,i}R_{i}^{T}J_{\omega,i}\right]\dot{q}
\end{equation}

\begin{equation}
J_{i} =
\begin{bmatrix}
J_{v,i} \\
J_{\omega,i}\\
\end{bmatrix}
=
\begin{bmatrix}
J_{v_{0}} & J_{v_{1}} & \dots & J_{v_{n}} \\
J_{\omega_{0}} & J_{\omega_{1}} & \dots & J_{\omega_{n}}
\end{bmatrix}
\end{equation}

\begin{equation}
P & = & M_{i}gh\\
```

```latex
= \sum\limits_{i=1}^{n} \vec{g}^{T} M_{i} \vec{r}_{c,i}
\end{equation}


\section{Results}

\subsection{Denavit-Hartenberg Parameters}
\begin{center}
\begin{tabular}{| 1 | p{1cm} | p{1cm} | p{1cm} | p{1cm} |}
\hline
Link_{i} & a_{i} & $\alpha_{i}$ & d_{i} & $\theta_{i}$ \\ \hline
1 & 0 & $\frac{\pi}{2}$ & L_{t} & $\theta_{t}$ \\ \hline
2 & 0 & 0 & L_{s} & 0 \\ \hline
\end{tabular}
\end{center}

\subsection{Jacobian}

\subsubsection{Jacobian of Link One's Center of Mass}
\begin{equation}
J_{1} =
\left[\begin{matrix}
0.5 l_{1} \sin{\left (q_{1} \right )} & 0\\
- 0.5 l_{1} \cos{\left (q_{1} \right )} & 0\\
0 & 0\\
0 & 0\\
0 & 0\\
1 & 0
\end{matrix}\right]
\end{equation}

\subsubsection{Jacobian of Link Two's Center of Mass}
\begin{equation}
J_{2} =
\left[\begin{matrix}
0 & 1.0 \sin{\left (q_{1} \right )}\\
0 & - 1.0 \cos{\left (q_{1} \right )}\\
1 & 6.12323399573677 \cdot 10^{-17}\\
0 & 0\\
0 & 0\\
0 & 0
\end{matrix}\right]
\end{equation}

\subsection{Lagrangian Results}
\begin{equation}
\begin{align}
L  &= 0.5 g l_{1} m_{1} \sin{\left (q_{1} \right )} + g m_{2} \left(0.5 l_{2}\nonumb
er\\
&\qquad + 1.0 q_{2}\right) \cos{\left (q_{1} \right )} + 0.5 m_{2} \dot{q}_{2} \left
(6.12323399573677 \cdot 10^{-17} \dot{q}_{1}\nonumber\\
&\qquad + 1.0 \dot{q}_{2}\right) + 0.5 \dot{q}_{1} \left(6.12323399573677 \cdot 10^{
-17} m_{2} \dot{q}_{2}\nonumber\\
&\qquad + \dot{q}_{1} \left(0.25 l_{1}^{2} m_{1} + 1.04719755119713 m_{1} r_{1}^{2}\
nonumber\\
&\qquad + 1.0 m_{2}\right)\right)
\end{align}
\end{equation}

\subsection{Torque: Derived from the Lagrangian}
\begin{equation}
\begin{align}
\tau_{k} &= - 0.5 g l_{1} m_{1} \cos{\left (q_{1} \right )} + g m_{2} \left(0.5 l_{2
}\nonumber\\
&\qquad + 1.0 q_{2}\right) \sin{\left (q_{1} \right )} - 1.0 g m_{2} \cos{\left (q_{
1} \right )}\nonumber\\
&\qquad + 1.0 m_{2} \ddot{q}_{2} + \ddot{q}_{1} \left(0.25 l_{1}^{2} m_{1}\nonumber\
\
&\qquad + 1.04719755119713 m_{1} r_{1}^{2} + 1.0 m_{2}\right)
\end{align}
\end{equation}

\end{document}
```

```
#!/usr/bin/env python
import sys
sys.path.append(r"/Users/robertbrothers/Desktop/Fall 2014/Fundamentals_of_Robotics/r
obo_git/python/")
import robotics_functions as rf, numpy as np, scipy as sp, sympy as sy

[l1, l2, l3, t1, t2, t3, a1, a2, a3, d1, d2, d3] = sy.symbols("l1 l2 l3 t1 t2 t3 a1
a2 a3 d1 d2 d3")
[q1, q2, qdot1, qdot2, qddot1, qddot2, m1, m2, r1, r2] = sy.symbols("q1 q2 qdot1 qdo
t2 qddot1 qddot2 m1 m2 r1 r2")

link_list_cm = [[
    [0, np.pi/2, q1, 0],
    [0, 0, q2, 0]
    ],
    [ sy.Matrix([[-l1/2],[0],[0],[1]]),
      sy.Matrix([[0],[0],[l2/2],[1]])
      ]
    ]

m = np.array([m1, m2])
l = np.array([l1, l2])
r = np.array([r1, r2])


M = [sy.Matrix([
  [m[i],0,0],
  [0,m[i],0],
  [0,0,m[i]]
  ]) for i in range(len(m))]
I = [sy.Matrix([
  [m1*l[0]**2/3,0,0],
  [0,m1*np.pi*r[0]**2/3, 0],
  [0, 0, m1*l[0]**2/3]
  ]),
  sy.Matrix([
    [m2*l[1]**2/3,0,0],
    [0,m2*l[1]**2/3,0],
    [0,0,m2*np.pi*r[1]**2/3]
    ])
  ]
q = sy.Matrix([
  [q1],
  [q2]
  ])
qdot = sy.Matrix([
  [qdot1],
  [qdot2]
  ])
tdv_vec = [
    (qdot1,qddot1),
    (qdot2,qddot2),
    (q1, qdot1),
    (q2, qdot2),
    ]
if __name__ == "__main__":
  print sy.pprint(sy.simplify(sy.trigsimp(rf.sym_pt_jacobian(link_list_cm)[1])))
```

```
#!/usr/bin/env python
import sys
sys.path.append(r"/Users/robertbrothers/Desktop/Fall 2014/Fundamentals_of_Robotics/r
obo_git/python/")
import robotics_functions as rf, numpy as np, scipy as sp, sympy as sy

[l1, l2, l3, t1, t2, t3, a1, a2, a3, d1, d2, d3] = sy.symbols("l1 l2 l3 t1 t2 t3 a1
a2 a3 d1 d2 d3")
[q1, q2, qdot1, qdot2, qddot1, qddot2, m1, m2, r1, r2] = sy.symbols("q1 q2 qdot1 qdo
t2 qddot1 qddot2 m1 m2 r1 r2")

link_list_cm = [[
    [0, np.pi/2, q1, 0],
    [0, 0, q2, 0]
    ],
    [ sy.Matrix([[-l1/2],[0],[0],[1]]),
      sy.Matrix([[0],[0],[l2/2],[1]])
      ]
    ]

m = np.array([m1, m2])
l = np.array([l1, l2])
r = np.array([r1, r2])


M = [sy.Matrix([
  [m[i],0,0],
  [0,m[i],0],
  [0,0,m[i]]
  ]) for i in range(len(m))]
I = [sy.Matrix([
  [m1*l[0]**2/3,0,0],
  [0,m1*np.pi*r[0]**2/3, 0],
  [0, 0, m1*l[0]**2/3]
  ]),
  sy.Matrix([
    [m2*l[1]**2/3,0,0],
    [0,m2*l[1]**2/3,0],
    [0,0,m2*np.pi*r[1]**2/3]
    ])
  ]
q = sy.Matrix([
  [q1],
  [q2]
  ])
qdot = sy.Matrix([
  [qdot1],
  [qdot2]
  ])
tdv_vec = [
    (qdot1,qddot1),
    (qdot2,qddot2),
    (q1, qdot1),
    (q2, qdot2),
    ]
if __name__ == "__main__":
  print sy.pprint(sy.simplify(sy.trigsimp(rf.sym_pt_jacobian(link_list_cm)[1])))
```

```python
import numpy as np, sympy as sy

gravity = sy.symbols("g")

def symsum( listin):
  val = sy.zeros(listin[0].shape)
  for i in listin:
    val = val+i
  return val


#define numerical rotation matricies
def rotation_z(theta):
  return np.matrix([
    [ np.cos(theta), -np.sin(theta), 0],
    [ np.sin(theta),  np.cos(theta), 0],
    [0, 0, 1]
    ])
def rotation_y(theta):
  return np.matrix([
    [ np.cos(theta), 0,  np.sin(theta)],
    [0, 1, 0],
    [-np.sin(theta), 0,  np.cos(theta)]
    ])
def rotation_x(theta):
  return np.matrix([
    [1, 0, 0],
    [0, np.cos(theta), -np.sin(theta)],
    [0, np.sin(theta),  np.cos(theta)]
    ])

# define symbolic rotation matricies
def sym_rotation_z(theta):
  return sy.Matrix([
    [ sy.cos(theta), -sy.sin(theta), 0],
    [ sy.sin(theta),  sy.cos(theta), 0],
    [0, 0, 1]
    ])
def sym_rotation_y(theta):
  return sy.Matrix([
    [ sy.cos(theta), 0,  sy.sin(theta)],
    [0, 1, 0],
    [-sy.sin(theta), 0,  sy.cos(theta)]
    ])
def sym_rotation_x(theta):
  return sy.Matrix([
    [1, 0, 0],
    [0, sy.cos(theta), -sy.sin(theta)],
    [0, sy.sin(theta),  sy.cos(theta)]
    ])

# define numerical translation matricies
def translation_z( d):
  return np.matrix([
    [1,0,0,0],
    [0,1,0,0],
    [0,0,1,d],
    [0,0,0,1]
    ])
def translation_y( d):
  return np.matrix([
    [1,0,0,0],
    [0,1,0,d],
    [0,0,1,0],
    [0,0,0,1]
    ])
def translation_x( d):
  return np.matrix([
    [1,0,0,d],
    [0,1,0,0],
    [0,0,1,0],
    [0,0,0,1]
    ])
```

```
# define symbolic translation matrices
def sym_translation_z( d):
  return sy.Matrix([
    [1,0,0,0],
    [0,1,0,0],
    [0,0,1,d],
    [0,0,0,1]
    ])
def sym_translation_y( d):
  return sy.Matrix([
    [1,0,0,0],
    [0,1,0,d],
    [0,0,1,0],
    [0,0,0,1]
    ])
def sym_translation_x( d):
  return sy.Matrix([
    [1,0,0,d],
    [0,1,0,0],
    [0,0,1,0],
    [0,0,0,1]
    ])
# numerically convert 3x3 rotation to 4x4 rotation
def convert3x3to4x4( matrix):
  # add column of zeroes
  matrix = np.hstack((matrix, np.transpose([np.zeros(3)])))
  # add row of 0,0,0,1
  matrix = np.vstack((matrix, np.array([0,0,0,1])))
  return matrix
# symbolically convert 3x3 rotation to 4x4 rotation
def syms_convert3x3to4x4( matrix):
  # add column of zeroes
  matrix = sy.Matrix.hstack(matrix, sy.Matrix(sy.zeros(3)[:,2]))
  # add row of 0,0,0,1
  matrix = sy.Matrix.vstack((matrix, sy.Matrix([0,0,0,1]).T))
  return matrix

def denavit_hartenberg( link):
  return (
    translation_z(link[2])*convert3x3to4x4(rotation_z(link[3]))*
    translation_x(link[0])*convert3x3to4x4(rotation_x(link[1]))
    )
def sym_denavit_hartenberg( link):
  return (
    sym_translation_z(link[2])*sy.Matrix(syms_convert3x3to4x4(sym_rotation_z(link[3]
)))*
    sym_translation_x(link[0])*sy.Matrix(syms_convert3x3to4x4(sym_rotation_x(link[1]
)))
    )

def get_A0n( link_list):
  A0i = np.identity(4)
  A0n = []
  for link in link_list:
    A0i = A0i*denavit_hartenberg( link)
    A0n.append(A0i)
  return A0n

def sym_get_A0n( link_list):
  A0i = np.identity(4)
  A0n = []
  for link in link_list:
    A0i = A0i*sym_denavit_hartenberg( link)
    A0n.append(A0i)
  return A0n

def get_A0i():
  A0i = []
  for link in link_list:
    A0i.append(denavit_hartenberg( link))
  return A0i
```

```python
def sym_get_A0i(link_list):
  A0i = []
  for link in link_list:
    A0i.append(sym_denavit_hartenberg( link))
  return A0i


def end_jacobian( link_list):
  A0n = get_A0n( link_list)
  # rotational vectors
  R = [np.matrix(np.identity(3))]
  R = R + [end[:3,:3] for end in A0n]
  # positions of each end effector
  O = [sy.Matrix([[0],[0],[0]])]
  O = O + [np.matrix(end[:,3][:3]) for end in A0n]
  # unit z vector
  k = np.matrix([[0],[0],[1]])
  J_v = []
  J_w = []
  for i in range(len(A0n)):
    # if theta_i is 0 then joint is prismatic
    if (link_list[i][3] == 0):
      J_v.append(sy.Matrix(R[i]*k))
      J_w.append(sy.Matrix([[0],[0],[0]]))
    # if theta_i is not 0 then joint is revolute
    else:
      J_v.append( np.matrix(np.cross( (R[i]*k).T, (O[-1]-O[i]).T)).T )
      J_w.append(np.matrix(R[i]*k))
  J = [np.vstack( (J_v[i], J_w[i])) for i in range(len(J_v))]
  J = np.hstack(J)
  return J


def sym_end_jacobian( link_list):
  A0n = sym_get_A0n( link_list)
  # rotational vectors
  R = [sy.Matrix(np.identity(3))]
  R = R + [end[:3,:3] for end in A0n]
  # positions of each end effector
  O = [sy.Matrix([[0],[0],[0]])]
  O = O + [sy.Matrix(end[:,3][:3]) for end in A0n]
  # unit z vector
  k = sy.Matrix([[0],[0],[1]])
  J_v = []
  J_w = []
  for i in range(len(A0n)):
    # if theta_i is 0 then joint is prismatic
    if (link_list[i][3] == 0):
      J_v.append(sy.Matrix(R[i]*k))
      J_w.append(sy.Matrix([[0],[0],[0]]))
    # if theta_i is not 0 then joint is revolute
    else:
      J_v.append((sy.Matrix(R[i]*k).cross(O[-1]-O[i])))
      J_w.append(sy.Matrix(R[i]*k))
  J = [sy.Matrix.vstack( J_v[i], J_w[i]) for i in range(len(J_v))]
  J = sy.Matrix.hstack(J)
  Je= J[0]
  for i in range(len(J)-1):
    j = i+1
    Je = sy.Matrix.hstack(Je, J[j])
  return Je

def sym_cm_jacobian(link_list):
  return 0#Jcm

def sym_pt_jacobian(link_list_position):
  k = sy.Matrix([[0],[0],[1]])
  [link_list, pt_list] = link_list_position
  A0i = sym_get_A0n(link_list)
  O0i = [A0i[i]*pt_list[i] if type(pt_list[i])== type(A0i[i]) else
      sy.Matrix(A0i[i][:,3][:3]) for i in range(len(A0i)) ]
  A0i = [sy.eye(4)]+A0i
  Jpt = []
```

```
  for i in range(len(link_list)):
    j_pt = []
    jci = sy.Matrix(np.zeros(6)).T
    for j in range(len(link_list)):
      if j <= i:
        if not link_list[i][-1] == 0:
          j_pt.append( sy.Matrix.vstack((A0i[j][:3,:3]*k).cross(
            sy.Matrix(O0i[i][:3])-sy.Matrix(A0i[j][:,3][:3])),
            A0i[j][:3,:3]*k)
            )
        else:
          j_pt.append( sy.Matrix.vstack(A0i[j][:3,:3]*k, sy.Matrix([[0],[0],[0]])))
      else:
        j_pt.append( sy.Matrix(np.zeros(6)).T)
    q = j_pt[0]
    for i in range(len(j_pt)-1):
      j = i+1
      q = sy.Matrix.hstack(q, j_pt[j])
    Jpt.append(q)
  return Jpt



#def rotational_velocity_jacobian( link_list)
#def jacobian(link_list)
def D_i( vec):
  [Ji, Mi, Ii, Ri] = vec
  Jv = Ji[:3,:]
  Jw = Ji[3:,:]
  return Jv.T*Mi*Jv + Jw.T*Ri*Ii*Ri.T*Jw


# need a function to return the F from the lagrangian and a list of all the time dep
endent variables
def sym_lagrangian(link_list_cm, M, I, qdot):
  g = sy.Matrix([[0],[gravity],[0]])
  [link_list, Ocm] = link_list_cm
  A = sym_get_A0n(link_list)
  R = [Ai[:3,:3] for Ai in A]
  O = [sy.Matrix(Ai[:,3][:3]) for Ai in A]
  J = sym_pt_jacobian(link_list_cm)
  D = symsum([D_i([ J[i], M[i], I[i], R[i]]) for i in range( len(J))])
  K = .5*(qdot.T*(D)*qdot)
  O0c = [A[i]*Ocm[i] for i in range(len(A))]
  P = symsum( [g.T*M[i]*sy.Matrix(O0c[i][:3]) for i in range(len(J))])
  return K-P

def sym_torque(link_list_cm, M, I, qdot, q, tdv_vec):
  L = sym_lagrangian(link_list_cm, M, I, qdot)[0]
  dLdq_dot = sum([sy.diff(L, qdot[i]) for i in range(len(qdot))])
  dLdq =  sum([sy.diff(L, q[i]) for i in range(len(q))])
  ddtdLdq_dot = sum([sy.diff(dLdq_dot, tdv_vec[i][0])*tdv_vec[i][1] for i in range(l
en(tdv_vec))])
  return ddtdLdq_dot - dLdq
```