Nama : Nurul Amelia

NIM : 1103194032

[Symforce]

+ Code + Text

```
pip install symforce
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: symforce in /usr/local/lib/python3.8/dist-packages (0.7.0)
Requirement already satisfied: black in /usr/local/lib/python3.8/dist-packages (from symforce) (22.10.0)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.8/dist-packages (from symforce) (2.11.3)
Requirement already satisfied: symforce-sym==0.7.0 in /usr/local/lib/python3.8/dist-packages (from symforce) (0.7.0)
Requirement already satisfied: sympy~=1.11.1 in /usr/local/lib/python3.8/dist-packages (from symforce) (1.11.1)
Requirement already satisfied: skymarshal==0.7.0 in /usr/local/lib/python3.8/dist-packages (from symforce) (0.7.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.8/dist-packages (from symforce) (1.21.6)
Requirement already satisfied: graphviz in /usr/local/lib/python3.8/dist-packages (from symforce) (0.10.1)
Requirement already satisfied: clang-format in /usr/local/lib/python3.8/dist-packages (from symforce) (15.0.4)
Requirement already satisfied: scipy in /usr/local/lib/python3.8/dist-packages (from symforce) (1.7.3)
Requirement already satisfied: ply in /usr/local/lib/python3.8/dist-packages (from skymarshal==0.7.0->symforce) (3.11)
Requirement already satisfied: six in /usr/local/lib/python3.8/dist-packages (from skymarshal==0.7.0->symforce) (1.15.0)
Requirement already satisfied: argh in /usr/local/lib/python3.8/dist-packages (from skymarshal==0.7.0->symforce) (0.26.2)
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.8/dist-packages (from sympy~=1.11.1->symforce) (1.2.1)
Requirement already satisfied: click>=8.0.0 in /usr/local/lib/python3.8/dist-packages (from black->symforce) (8.1.3)
Requirement already satisfied: typing-extensions>=3.10.0.0 in /usr/local/lib/python3.8/dist-packages (from black->symforce) (4.1.1)
Requirement already satisfied: platformdirs>=2 in /usr/local/lib/python3.8/dist-packages (from black->symforce) (2.5.4)
Requirement already satisfied: pathspec>=0.9.0 in /usr/local/lib/python3.8/dist-packages (from black->symforce) (0.10.2)
Requirement already satisfied: tomli>=1.1.0 in /usr/local/lib/python3.8/dist-packages (from black->symforce) (2.0.1)
Requirement already satisfied: mypy-extensions>=0.4.3 in /usr/local/lib/python3.8/dist-packages (from black->symforce) (0.4.3)
Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python3.8/dist-packages (from jinja2->symforce) (2.0.1)
```

[3]
```
# Configuration (optional)
import symforce

symforce.set_symbolic_api("sympy") # The sympy API is the default and pure python.
symforce.set_log_level("warning")
# symforce.notebook_util - helpers for interactive use in a Jupyter notebook with an IPython kernel.
# display - display the given expressions in latex, or print if not an expression.
# print_expression_tree - print a SymPy expression tree, ignoring node attributes.
from symforce.notebook_util import display, print_expression_tree
```

[4]
```
# algebraic symbols.
import symforce.symbolic as sf

x = sf.Symbol("x")
y = sf.Symbol("y")
```

```
# build a symbolic expression.
expr = x ** 2 + sf.sin(y) / x ** 2
display(expr)
```

$$x^2 + \frac{\sin(y)}{x^2}$$

[6]
```
# this expression object is a tree of operations and arguments.
print_expression_tree(expr)
```

```
Add: x**2 + sin(y)/x**2
+-Pow: x**2
| +-Symbol: x
| +-Integer: 2
+-Mul: sin(y)/x**2
  +-Pow: x**(-2)
  | +-Symbol: x
  | +-Integer: -2
  +-sin: sin(y)
    +-Symbol: y
```

[7]
```
# we can evaluate this numerically by plugging in values.
display(expr.subs({x: 1.2, y: 0.4}))
```

1.71042940438101

[8]
```
# we can perform symbolic manipulation like differentiation, integration, simplifiation, etc..
display(expr.diff(y))
```

$$\frac{\cos(y)}{x^2}$$

```
display(sf.series(expr, y))
```

$$\frac{y}{x^2} - \frac{y^3}{6x^2} + \frac{y^5}{120x^2} + x^2 + O\left(y^6\right)$$

[Geometry]

+ Code + Text

```
pip install symforce
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: symforce in /usr/local/lib/python3.8/dist-packages (0.7.0)
Requirement already satisfied: scipy in /usr/local/lib/python3.8/dist-packages (from symforce) (1.7.3)
Requirement already satisfied: clang-format in /usr/local/lib/python3.8/dist-packages (from symforce) (15.0.4)
Requirement already satisfied: black in /usr/local/lib/python3.8/dist-packages (from symforce) (22.10.0)
Requirement already satisfied: skymarshal==0.7.0 in /usr/local/lib/python3.8/dist-packages (from symforce) (0.7.0)
Requirement already satisfied: symforce-sym==0.7.0 in /usr/local/lib/python3.8/dist-packages (from symforce) (0.7.0)
Requirement already satisfied: graphviz in /usr/local/lib/python3.8/dist-packages (from symforce) (0.10.1)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.8/dist-packages (from symforce) (2.11.3)
Requirement already satisfied: numpy in /usr/local/lib/python3.8/dist-packages (from symforce) (1.21.6)
Requirement already satisfied: sympy~=1.11.1 in /usr/local/lib/python3.8/dist-packages (from symforce) (1.11.1)
Requirement already satisfied: argh in /usr/local/lib/python3.8/dist-packages (from skymarshal==0.7.0->symforce) (0.26.2)
Requirement already satisfied: ply in /usr/local/lib/python3.8/dist-packages (from skymarshal==0.7.0->symforce) (3.11)
Requirement already satisfied: six in /usr/local/lib/python3.8/dist-packages (from skymarshal==0.7.0->symforce) (1.15.0)
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.8/dist-packages (from sympy~=1.11.1->symforce) (1.2.1)
Requirement already satisfied: platformdirs>=2 in /usr/local/lib/python3.8/dist-packages (from black->symforce) (2.5.4)
Requirement already satisfied: mypy-extensions>=0.4.3 in /usr/local/lib/python3.8/dist-packages (from black->symforce) (0.4.3)
Requirement already satisfied: click>=8.0.0 in /usr/local/lib/python3.8/dist-packages (from black->symforce) (8.1.3)
Requirement already satisfied: tomli>=1.1.0 in /usr/local/lib/python3.8/dist-packages (from black->symforce) (2.0.1)
Requirement already satisfied: typing-extensions>=3.10.0.0 in /usr/local/lib/python3.8/dist-packages (from black->symforce) (4.1.1)
Requirement already satisfied: pathspec>=0.9.0 in /usr/local/lib/python3.8/dist-packages (from black->symforce) (0.10.2)
Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python3.8/dist-packages (from jinja2->symforce) (2.0.1)
```

[3]
```python
# Setup
import symforce

symforce.set_symbolic_api("sympy")
symforce.set_log_level("warning")

from symforce.notebook_util import display
import symforce.symbolic as sf
from symforce import ops
```

[4]
```python
# Identity definition
display(sf.Rot3())
```

```
<Rot3 <Q xyzw=[0, 0, 0, 1]>>
```

+ Code + Text RAM Disk Editing

```python
# Symbolic definition
display(sf.Rot3.symbolic("R"))
```

```
<Rot3 <Q xyzw=[R_x, R_y, R_z, R_w]>>
```

[7]
```python
# To/From rotation matrix

# Rotate about x-axis
theta = sf.Symbol("theta")
R_mat = sf.Matrix(
    [
        [1, 0, 0],
        [0, sf.cos(theta), -sf.sin(theta)],
        [0, sf.sin(theta), sf.cos(theta)],
    ]
)
R = sf.Rot3.from_rotation_matrix(R_mat)

display(R_mat)
display(R)  # Note the additional expressions required to avoid numerical errors
display(R.to_rotation_matrix())
```

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{bmatrix}$$

```
<Rot3 <Q xyzw=[(1 - Max(0, sign(Max(1, 2*cos(theta) + 1, cos(theta)) - 1)))*(sqrt(Max(0, 2 - 2*cos(theta)))/2 + Max(0, sign(Max(1, 2*cos(theta) + 1, cos(theta)) - 1))/2) + 2*sin(theta)*Min(1 - Max(0, sign(-2*cos(theta) + Max(1,
2*cos(theta) + 1, cos(theta)) - 1)), 1 - Max(0, 1 - Max(0, sign(Max(1, 2*cos(theta) + 1, cos(theta)) - 1)), Min(1 - Max(0, sign(-cos(theta) + Max(1, 2*cos(theta) + 1, cos(theta)))), Max(1 - Max(0, 1 - Max(0, sign(Max(1, 2*cos(theta)
+ 1, cos(theta)) - 1))), 1 - Max(0, 1 - Max(0, sign(Max(1, 2*cos(theta) + 1, cos(theta)) - 1)), Min(1 - Max(0, 1 - Max(0, sign(Max(1, 2*cos(theta) + 1, cos(theta)) - 1))), 1 - Max(0, sign(-cos(theta) + Max(1, 2*cos(theta) + 1,
cos(theta))))))))))/(2*sqrt(Max(0, 2*cos(theta) + 2)) - 2*Min(1 - Max(0, sign(-2*cos(theta) + Max(1, 2*cos(theta) + 1, cos(theta)) - 1)), 1 - Max(0, 1 - Max(0, sign(Max(1, 2*cos(theta) + 1, cos(theta)) - 1)), Min(1 - Max(0, sign(-
cos(theta) + Max(1, 2*cos(theta) + 1, cos(theta)))), Max(1 - Max(0, 1 - Max(0, sign(Max(1, 2*cos(theta) + 1, cos(theta)) - 1))), 1 - Max(0, 1 - Max(0, sign(Max(1, 2*cos(theta) + 1, cos(theta)) - 1)), Min(1 - Max(0, 1 - Max(0,
sign(Max(1, 2*cos(theta) + 1, cos(theta)) - 1))), 1 - Max(0, sign(-cos(theta) + Max(1, 2*cos(theta) + 1, cos(theta))))))))) + 2), (1/2 - Min(1 - Max(0, 1 - Max(0, sign(Max(1, 2*cos(theta) + 1, cos(theta)) - 1))), 1 - Max(0, sign(-
cos(theta) + Max(1, 2*cos(theta) + 1, cos(theta)))))/2)*Min(1 - Max(0, 1 - Max(0, sign(Max(1, 2*cos(theta) + 1, cos(theta)) - 1))), 1 - Max(0, sign(-cos(theta) + Max(1, 2*cos(theta) + 1, cos(theta))))), (1/2 - Min(1 - Max(0, sign(-
cos(theta) + Max(1, 2*cos(theta) + 1, cos(theta)))), 1 - Max(0, 1 - Max(0, sign(Max(1, 2*cos(theta) + 1, cos(theta)) - 1)), Min(1 - Max(0, 1 - Max(0, sign(Max(1, 2*cos(theta) + 1, cos(theta)) - 1))), 1 - Max(0, sign(-cos(theta) +
Max(1, 2*cos(theta) + 1, cos(theta))))))/2)*Min(1 - Max(0, sign(-cos(theta) + Max(1, 2*cos(theta) + 1, cos(theta)))), 1 - Max(0, 1 - Max(0, sign(Max(1, 2*cos(theta) + 1, cos(theta)) - 1)), Min(1 - Max(0, 1 - Max(0, sign(Max(1,
2*cos(theta) + 1, cos(theta)) - 1))), 1 - Max(0, sign(-cos(theta) + Max(1, 2*cos(theta) + 1, cos(theta)))))), 2*(1 - Max(0, sign(Max(1, 2*cos(theta) + 1, cos(theta)) - 1)))*sin(theta)/(2*sqrt(Max(0, 2 - 2*cos(theta))) + 2*Max(0,
sign(Max(1, 2*cos(theta) + 1, cos(theta)) - 1))) + (sqrt(Max(0, 2*cos(theta) + 2))/2 - Min(1 - Max(0, sign(-2*cos(theta) + Max(1, 2*cos(theta) + 1, cos(theta)) - 1)), 1 - Max(0, 1 - Max(0, sign(Max(1, 2*cos(theta) + 1, cos(theta)) -
1)), Min(1 - Max(0, sign(-cos(theta) + Max(1, 2*cos(theta) + 1, cos(theta)))), Max(1 - Max(0, 1 - Max(0, sign(Max(1, 2*cos(theta) + 1, cos(theta)) - 1))), 1 - Max(0, 1 - Max(0, sign(Max(1, 2*cos(theta) + 1, cos(theta)) - 1)), Min(1 -
Max(0, 1 - Max(0, sign(Max(1, 2*cos(theta) + 1, cos(theta)) - 1))), 1 - Max(0, sign(-cos(theta) + Max(1, 2*cos(theta) + 1, cos(theta)))))))))/2 + 1/2)*Min(1 - Max(0, sign(-2*cos(theta) + Max(1, 2*cos(theta) + 1, cos(theta)) - 1)), 1
- Max(0, 1 - Max(0, sign(Max(1, 2*cos(theta) + 1, cos(theta)) - 1)), Min(1 - Max(0, sign(-cos(theta) + Max(1, 2*cos(theta) + 1, cos(theta)))), Max(1 - Max(0, 1 - Max(0, sign(Max(1, 2*cos(theta) + 1, cos(theta)) - 1))), 1 - Max(0, 1 -
Max(0, sign(Max(1, 2*cos(theta) + 1, cos(theta)) - 1)), Min(1 - Max(0, 1 - Max(0, sign(Max(1, 2*cos(theta) + 1, cos(theta)) - 1))), 1 - Max(0, sign(-cos(theta) + Max(1, 2*cos(theta) + 1, cos(theta)))))))))]>>
```

0s completed at 5:44 AM

[8]
```python
# To/From Euler angles
R = sf.Rot3.from_yaw_pitch_roll(0, 0, theta)  # Yaw rotation only
ypr = R.to_yaw_pitch_roll()

display(R)
display(ops.StorageOps.simplify(list(ypr)))  # Simplify YPR expression
```

```
<Rot3 <Q xyzw=[sin(theta/2), 0, 0, cos(theta/2)]>>
```

$[0,\ 0,\ \operatorname{atan}_2(\sin(\theta), \cos(\theta))]$

[9]
```python
# From axis-angle representation

# Rotate about x-axis
R = sf.Rot3.from_angle_axis(angle=theta, axis=sf.Vector3(1, 0, 0))

display(R)
```

```
<Rot3 <Q xyzw=[sin(theta/2), 0, 0, cos(theta/2)]>>
```

[10]
```python
# Rotation defining orientation of body frame wrt world frame
world_R_body = sf.Rot3.symbolic("R")

# Point written in body frame
body_t_point = sf.Vector3.symbolic("p")

# Point written in world frame
world_t_point = world_R_body * body_t_point

display(world_t_point)
```

$$\begin{bmatrix} p_0\left(-2R_y^2 - 2R_z^2 + 1\right) + p_1\left(-2R_wR_z + 2R_xR_y\right) + p_2\cdot\left(2R_wR_y + 2R_xR_z\right) \\ p_0\cdot\left(2R_wR_z + 2R_xR_y\right) + p_1\left(-2R_x^2 - 2R_z^2 + 1\right) + p_2\left(-2R_wR_x + 2R_yR_z\right) \\ p_0\left(-2R_wR_y + 2R_xR_z\right) + p_1\cdot\left(2R_wR_x + 2R_yR_z\right) + p_2\left(-2R_x^2 - 2R_y^2 + 1\right) \end{bmatrix}$$

[11]
```python
body_R_cam = sf.Rot3.symbolic("R_cam")
world_R_cam = world_R_body * body_R_cam

# Rotation inverse = negate vector part of quaternion
cam_R_body = body_R_cam.inverse()
display(body_R_cam)
display(cam_R_body)
```

```
<Rot3 <Q xyzw=[R_cam_x, R_cam_y, R_cam_z, R_cam_w]>>
<Rot3 <Q xyzw=[-R_cam_x, -R_cam_y, -R_cam_z, R_cam_w]>>
```

[12]
```python
world_R_body_numeric = sf.Rot3.from_yaw_pitch_roll(0.1, -2.3, 0.7)
display(world_t_point.subs(world_R_body, world_R_body_numeric))
```

$$\begin{bmatrix} -0.662947416398295p_0 - 0.554353314451006p_1 - 0.503182994394693p_2 \\ -0.0665166116342196p_0 + 0.713061539471145p_1 - 0.697938952419008p_2 \\ 0.74570521217672p_0 - 0.429226797490819p_1 - 0.509596009450867p_2 \end{bmatrix}$$

Poses

Poses are defined as a rotation plus a translation, and are constructed as such. We use the notation world_T_body to represent a pose that transforms from the body frame to the world frame.

[13]
```python
# Symbolic construction
world_T_body = sf.Pose3.symbolic("T")
display(world_T_body)
```

```
<Pose3 R=<Rot3 <Q xyzw=[T.R_x, T.R_y, T.R_z, T.R_w]>>, t=(T.t0, T.t1, T.t2)>
```

[14]
```python
# Construction from a rotation and translation

# Orientation of body frame wrt world frame
world_R_body = sf.Rot3.symbolic("R")

# Position of body frame wrt world frame written in the world frame
world_t_body = sf.Vector3.symbolic("t")

world_T_body = sf.Pose3(R=world_R_body, t=world_t_body)
display(world_T_body)
```

```
<Pose3 R=<Rot3 <Q xyzw=[R_x, R_y, R_z, R_w]>>, t=(t0, t1, t2)>
```

[15]
```python
# Compose pose with a pose
body_T_cam = sf.Pose3.symbolic("T_cam")
world_T_cam = world_T_body * body_T_cam

# Compose pose with a point
body_t_point = sf.Vector3.symbolic("p")  # Position in body frame
# Equivalent to: world_R_body * body_t_point + world_t_body
world_t_point = world_T_body * body_t_point
display(world_t_point)
```

$$\begin{bmatrix} p_0\left(-2R_y^2 - 2R_z^2 + 1\right) + p_1\left(-2R_wR_z + 2R_xR_y\right) + p_2\cdot\left(2R_wR_y + 2R_xR_z\right) + t_0 \\ p_0\cdot\left(2R_wR_z + 2R_xR_y\right) + p_1\left(-2R_x^2 - 2R_z^2 + 1\right) + p_2\left(-2R_wR_x + 2R_yR_z\right) + t_1 \\ p_0\left(-2R_wR_y + 2R_xR_z\right) + p_1\cdot\left(2R_wR_x + 2R_yR_z\right) + p_2\left(-2R_x^2 - 2R_y^2 + 1\right) + t_2 \end{bmatrix}$$

[16]
```python
# Invert a pose
body_T_world = world_T_body.inverse()
display(world_T_body)
display(body_T_world)
```

```
<Pose3 R=<Rot3 <Q xyzw=[R_x, R_y, R_z, R_w]>>, t=(t0, t1, t2)>
<Pose3 R=<Rot3 <Q xyzw=[-R_x, -R_y, -R_z, R_w]>>, t=(-t0*(-2*R_y**2 - 2*R_z**2 + 1) - t1*(2*R_w*R_z + 2*R_x*R_y) - t2*(-2*R_w*R_y + 2*R_x*R_z), -t0*(-2*R_w*R_z + 2*R_x*R_y) - t1*(-2*R_x**2 - 2*R_z**2 + 1) - t2*(2*R_w*R_x +
2*R_y*R_z), -t0*(2*R_w*R_y + 2*R_x*R_z) - t1*(-2*R_w*R_x + 2*R_y*R_z) - t2*(-2*R_x**2 - 2*R_y**2 + 1))>
```

+ Code + Text

Vectors and Matrices

Vectors and matrices are all represented using subclasses of sf.Matrix class, and can be constructed in several different ways as shown below.

[17]
```
# Matrix construction. The statements below all create the same 2x3 matrix object

# Construction from 2D list
m1 = sf.Matrix([[1, 2, 3], [4, 5, 6]])

# Construction using specified size + data
m2 = sf.Matrix(2, 3, [1, 2, 3, 4, 5, 6])

# sf.MatrixNM creates a matrix with shape NxM (defined by default for 6x6
# matrices and smaller)
m3 = sf.Matrix23(1, 2, 3, 4, 5, 6)
m4 = sf.Matrix23([1, 2, 3, 4, 5, 6])

# Construction using aliases
m5 = sf.M([[1, 2, 3], [4, 5, 6]])
m6 = sf.M(2, 3, [1, 2, 3, 4, 5, 6])
m7 = sf.M23(1, 2, 3, 4, 5, 6)
m8 = sf.M23([1, 2, 3, 4, 5, 6])

# Construction from block matrices of appropriate dimensions
m9 = sf.Matrix23.block_matrix([[sf.M13([1, 2, 3])], [sf.M13([3, 4, 5])]])
```

[18]
```
# Vector constructors. The statements below all create the same 3x1 vector object

# Construction from 2D list
v1 = sf.Matrix([[1], [2], [3]])

# Construction from 1D list. We assume a 1D list represents a column vector.
v2 = sf.Matrix([1, 2, 3])

# Construction using aliases (defined by default for 9x1 vectors and smaller)
v3 = sf.Matrix31(1, 2, 3)
v4 = sf.M31(1, 2, 3)
v5 = sf.Vector3(1, 2, 3)
v6 = sf.V3(1, 2, 3)
```

+ Code + Text

[19]
```
# Matrix of zeros
z1 = sf.Matrix23.zero()
z2 = sf.Matrix.zeros(2, 3)

# Matrix of ones
o1 = sf.Matrix23.one()
o2 = sf.Matrix.ones(2, 3)
```

[21]
```
zero_matrix = sf.Matrix33.zero()
identity_matrix = sf.Matrix33.eye()

# Could also write:
zero_matrix = ops.GroupOps.identity(sf.Matrix33)

display(zero_matrix)
display(identity_matrix)
```

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

$$\begin{bmatrix}1 & 0 & 0\\0 & 1 & 0\\0 & 0 & 1\end{bmatrix}$$

[22]
```
# Matrix multiplication
m23 = sf.M23.symbolic("lhs")
m31 = sf.V3.symbolic("rhs")
display(m23 * m31)
```

$$\begin{bmatrix}lhs_{00}rhs_0 + lhs_{01}rhs_1 + lhs_{02}rhs_2\\lhs_{10}rhs_0 + lhs_{11}rhs_1 + lhs_{12}rhs_2\end{bmatrix}$$

+ Code + Text

```
[23] # Vector operations
     norm = m31.norm()
     squared_norm = m31.squared_norm()
     unit_vec = m31.normalized()
     display(unit_vec)
```

$$\begin{bmatrix} \frac{rhs_0}{\sqrt{rhs_0^2 + rhs_1^2 + rhs_2^2}} \\ \frac{rhs_1}{\sqrt{rhs_0^2 + rhs_1^2 + rhs_2^2}} \\ \frac{rhs_2}{\sqrt{rhs_0^2 + rhs_1^2 + rhs_2^2}} \end{bmatrix}$$

```
[24] m33 = 5 * sf.Matrix33.eye()  # Element-wise multiplication with scalar
     display(m33.inv())  # Matrix inverse
```

$$\begin{bmatrix} \frac{1}{5} & 0 & 0 \\ 0 & \frac{1}{5} & 0 \\ 0 & 0 & \frac{1}{5} \end{bmatrix}$$

One of the most powerful operations can use matrices for is to compute jacobians with respect to other geo objects. By default compute jacobians with respect to the tangent space of the given object.

```
[25] R0 = sf.Rot3.symbolic("R0")
     R1 = sf.Rot3.symbolic("R1")
     residual = sf.M(R0.local_coordinates(R1))
     display(residual)
```

$$\begin{bmatrix} \frac{2\cdot(2\min(0,\operatorname{sign}(R_{0w}R_{1w}+R_{0x}R_{1x}+R_{0y}R_{1y}+R_{0z}R_{1z}))+1)(R_{0w}R_{1x}-R_{0x}R_{1w}-R_{0y}R_{1z}+R_{0z}R_{1y})\operatorname{acos}(\min(1.0,|R_{0w}R_{1w}+R_{0x}R_{1x}+R_{0y}R_{1y}+R_{0z}R_{1z}|))}{\sqrt{1-\min(1.0,|R_{0w}R_{1w}+R_{0x}R_{1x}+R_{0y}R_{1y}+R_{0z}R_{1z}|)^2}} \\ \frac{2\cdot(2\min(0,\operatorname{sign}(R_{0w}R_{1w}+R_{0x}R_{1x}+R_{0y}R_{1y}+R_{0z}R_{1z}))+1)(R_{0w}R_{1y}+R_{0x}R_{1z}-R_{0y}R_{1w}-R_{0z}R_{1x})\operatorname{acos}(\min(1.0,|R_{0w}R_{1w}+R_{0x}R_{1x}+R_{0y}R_{1y}+R_{0z}R_{1z}|))}{\sqrt{1-\min(1.0,|R_{0w}R_{1w}+R_{0x}R_{1x}+R_{0y}R_{1y}+R_{0z}R_{1z}|)^2}} \\ \frac{2\cdot(2\min(0,\operatorname{sign}(R_{0w}R_{1w}+R_{0x}R_{1x}+R_{0y}R_{1y}+R_{0z}R_{1z}))+1)(R_{0w}R_{1z}-R_{0x}R_{1y}+R_{0y}R_{1x}-R_{0z}R_{1w})\operatorname{acos}(\min(1.0,|R_{0w}R_{1w}+R_{0x}R_{1x}+R_{0y}R_{1y}+R_{0z}R_{1z}|))}{\sqrt{1-\min(1.0,|R_{0w}R_{1w}+R_{0x}R_{1x}+R_{0y}R_{1y}+R_{0z}R_{1z}|)^2}} \end{bmatrix}$$

```
[26] jacobian = residual.jacobian(R1)
     # The jacobian is quite a complex symbolic expression, so we don't display it for
     # convenience.
     # The shape is equal to (dimension of residual) x (dimension of tangent space)
     display(jacobian.shape)
```

$(3, 3)$

+ Code + Text

General properties of geo objects

Storage operations

All geometric types implement the “Storage” interface. This means that they can:

1. Be serialized into a list of scalar expression(to_storage())
2. Be reconstructed from a list of scalar expressions (.from_storage())
3. Use common symbolic operations (symbolic construction, substitution, simplification, etc.)

```
[27] # Serialization to scalar list
     rot = sf.Rot3()
     elements = rot.to_storage()
     assert len(elements) == rot.storage_dim()
     display(elements)
```

$[0, 0, 0, 1]$

```
[28] # Construction from scalar list
     rot2 = sf.Rot3.from_storage(elements)
     assert rot == rot2
```

```
[29] # Symbolic operations
     rot_sym = sf.Rot3.symbolic("rot_sym")
     rot_num = rot_sym.subs(rot_sym, rot)

     display(rot_sym)
     display(rot_num)
     display(rot_num.simplify())  # Simplify internal symbolic expressions
     display(rot_num.evalf())  # Numerical evaluation
```

```
<Rot3 <Q xyzw=[rot_sym_x, rot_sym_y, rot_sym_z, rot_sym_w]>>
<Rot3 <Q xyzw=[0, 0, 0, 1]>>
<Rot3 <Q xyzw=[0, 0, 0, 1]>>
<Rot3 <Q xyzw=[0, 0, 0, 1.00000000000000]>>
```

+ Code + Text

Group operations

All geometric types also implement the “Group” interface, meaning that geometric objects:

1. Can be composed with objects of the same type to produce an object of the same type (.compose())
2. Have an identity element (.identity())
3. Can be inverted (.inverse())
4. Can be created to represent the relation between two other objects of the same type (.between())

[30]
```
# Construct two random rotations
R1 = sf.Rot3.random()
R2 = sf.Rot3.random()

# Composition
display(R1.compose(R2))  # For rotations this is the same as R1 * R2
```

```
<Rot3 <Q xyzw=[0.653265442882310, 0.188414896040808, 0.552681423650694, -0.481961961193156]>>
```

[31]
```
# Identity
R_identity = sf.Rot3.identity()
display(R1)
display(R_identity * R1)
```

```
<Rot3 <Q xyzw=[-0.728809474196865, 0.220562873248606, -0.103322716860027, 0.639932172535395]>>
<Rot3 <Q xyzw=[-0.728809474196865, 0.220562873248606, -0.103322716860027, 0.639932172535395]>>
```

[32]
```
# Inverse
R1_inv = R1.inverse()
display(R_identity)
display(R1_inv * R1)
```

```
<Rot3 <Q xyzw=[0, 0, 0, 1]>>
<Rot3 <Q xyzw=[0, 0, 0, 1.00000000000000]>>
```

+ Code + Text

[33]
```
# Between
R_delta = R1.between(R2)
display(R1 * R_delta)
display(R2)
```

```
<Rot3 <Q xyzw=[-0.0745814111614632, -0.108426628661147, 0.585285669293584, -0.800076224259854]>>
<Rot3 <Q xyzw=[-0.0745814111614631, -0.108426628661147, 0.585285669293584, -0.800076224259854]>>
```

Lie Group operations

Rotations, poses, and matrices all implement the “LieGroup” interface, meaning that they each have a tangent space. There are many great references on Lie groups out there already, so instead of introducing them here, we recommend checking out Frank Dellaert’s, Ethan Eade’s, or JL Blanco’s tutorials. In SymForce, objects which are a Lie Group can:

1. Be used to compute the tangent space vector about the identity element (.to_tangent())
2. Be constructed from a tangent space vector about the identity element (.from_tangent())
3. Be perturbed by a tangent space vector about the given element (.retract())
4. Be used to compute the tangent space perturbation needed to obtain another given element (.local_coordinates())
5. Be used to compute a jacobian describing the relation between the underlying data of the object (e.g. a quaternion for a rotation) and the tangent space vector about the given element (.storage_D_tangent())

[34]
```
# To/From tangent space vector about identity element
R1 = sf.Rot3.random()
tangent_vec = R1.to_tangent()
R1_recovered = sf.Rot3.from_tangent(tangent_vec)

assert len(tangent_vec) == R1.tangent_dim()
display(R1)
display(R1_recovered)
```

```
<Rot3 <Q xyzw=[0.0294442699046180, -0.423525123320371, 0.508118659749276, -0.749383034569671]>>
<Rot3 <Q xyzw=[-0.0294442699046180, 0.423525123320371, -0.508118659749276, 0.749383034569671]>>
```

[35]

```python
# Tangent space perturbations

# Perturb R1 by the given vector in the tangent space around R1
R2 = R1.retract([0.1, 2.3, -0.5])

# Compute the tangent vector pointing from R1 to R2, in the tangent space
# around R1
recovered_tangent_vec = R1.local_coordinates(R2)

display(recovered_tangent_vec)
```

$[0.1, \ 2.3, \ -0.5]$

[36]

```python
# Jacobian of storage w.r.t tangent space perturbation

# We chain storage_D_tangent together with jacobians of larger symbolic
# expressions taken with respect to the symbolic elements of the object (e.g. a
# quaternion for rotations) to compute the jacobian wrt the tanget space about
# the element.
# I.e. residual_D_tangent = residual_D_storage * storage_D_tangent

jacobian = R1.storage_D_tangent()
assert jacobian.shape == (R1.storage_dim(), R1.tangent_dim())
```

[Ops]

+ Code + Text

```
[2] pip install symforce
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: symforce in /usr/local/lib/python3.8/dist-packages (0.7.0)
Requirement already satisfied: clang-format in /usr/local/lib/python3.8/dist-packages (from symforce) (15.0.4)
Requirement already satisfied: skymarshal==0.7.0 in /usr/local/lib/python3.8/dist-packages (from symforce) (0.7.0)
Requirement already satisfied: symforce-sym==0.7.0 in /usr/local/lib/python3.8/dist-packages (from symforce) (0.7.0)
Requirement already satisfied: sympy~=1.11.1 in /usr/local/lib/python3.8/dist-packages (from symforce) (1.11.1)
Requirement already satisfied: graphviz in /usr/local/lib/python3.8/dist-packages (from symforce) (0.10.1)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.8/dist-packages (from symforce) (2.11.3)
Requirement already satisfied: scipy in /usr/local/lib/python3.8/dist-packages (from symforce) (1.7.3)
Requirement already satisfied: numpy in /usr/local/lib/python3.8/dist-packages (from symforce) (1.21.6)
Requirement already satisfied: black in /usr/local/lib/python3.8/dist-packages (from symforce) (22.10.0)
Requirement already satisfied: six in /usr/local/lib/python3.8/dist-packages (from skymarshal==0.7.0->symforce) (1.15.0)
Requirement already satisfied: ply in /usr/local/lib/python3.8/dist-packages (from skymarshal==0.7.0->symforce) (3.11)
Requirement already satisfied: argh in /usr/local/lib/python3.8/dist-packages (from skymarshal==0.7.0->symforce) (0.26.2)
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.8/dist-packages (from sympy~=1.11.1->symforce) (1.2.1)
Requirement already satisfied: pathspec>=0.9.0 in /usr/local/lib/python3.8/dist-packages (from black->symforce) (0.10.2)
Requirement already satisfied: mypy-extensions>=0.4.3 in /usr/local/lib/python3.8/dist-packages (from black->symforce) (0.4.3)
Requirement already satisfied: tomli>=1.1.0 in /usr/local/lib/python3.8/dist-packages (from black->symforce) (2.0.1)
Requirement already satisfied: platformdirs>=2 in /usr/local/lib/python3.8/dist-packages (from black->symforce) (2.5.4)
Requirement already satisfied: click>=8.0.0 in /usr/local/lib/python3.8/dist-packages (from black->symforce) (8.1.3)
Requirement already satisfied: typing-extensions>=3.10.0.0 in /usr/local/lib/python3.8/dist-packages (from black->symforce) (4.1.1)
Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python3.8/dist-packages (from jinja2->symforce) (2.0.1)
```

```
[3] # Setup
import symforce

symforce.set_symbolic_api("sympy")
symforce.set_log_level("warning")

from symforce.notebook_util import display
import symforce.symbolic as sf
from symforce.values import Values
from symforce.ops import StorageOps, GroupOps, LieGroupOps
```

+ Code + Text

StorageOps

StorageOps: Data type that can be serialized to and from a vector of scalar quantities.

Methods: .storage_dim(), .to_storage(), .from_storage(), .symbolic(), .evalf(), .subs(), .simplify()

Storage operations are used extensively for marshalling and for operating on each scalar in a type.

```
[4] # Number of scalars used to represent a Pose3 (4 quaternion + 3 position)
display(StorageOps.storage_dim(sf.Pose3))
```

7

```
[5] # Because we are using concepts, we can operate on types that aren't subtypes of symforce
display(StorageOps.storage_dim(float))
```

1

```
[6] # Element-wise operations on lists of objects
display(StorageOps.storage_dim([sf.Pose3, sf.Pose3]))
```

14

```
[7] # Element-wise operations on Values object with multiple types of elements
values = Values(
    pose=sf.Pose3(),
    scalar=sf.Symbol("x"),
)
display(StorageOps.storage_dim(values))  # 4 quaternion + 3 position + 1 scalar
```

8

```
[8] # Serialize scalar
display(StorageOps.to_storage(5))
```

$[5]$

```
[9] # Serialize vector/matrix
display(StorageOps.to_storage(sf.V3(sf.Symbol("x"), 5.2, sf.sqrt(5))))
```

$[x,\ 5.2,\ \sqrt{5}]$

+ Code + Text

[10]
```python
# Serialize geometric type and reconstruct
T = sf.Pose3.symbolic("T")
T_serialized = StorageOps.to_storage(T)
T_recovered = StorageOps.from_storage(sf.Pose3, T_serialized)
display(T_serialized)
display(T_recovered)
```

$[T.R_x,\ T.R_y,\ T.R_z,\ T.R_w,\ T.t0,\ T.t1,\ T.t2]$

```
<Pose3 R=<Rot3 <Q xyzw=[T.R_x, T.R_y, T.R_z, T.R_w]>>, t=(T.t0, T.t1, T.t2)>
```

GroupOps

GroupOps: Mathematical group that implements closure, associativity, identity and invertibility.

Methods: .identity(), .inverse(), .compose(), .between()

Group operations provide the core methods to compare and combine types.

[11]
```python
# Identity of a pose
display(GroupOps.identity(sf.Pose3))
```

```
<Pose3 R=<Rot3 <Q xyzw=[0, 0, 0, 1]>>, t=(0, 0, 0)>
```

[12]
```python
# Identity of a scalar (under addition)
display(GroupOps.identity(float))
```

0.0

[13]
```python
# Inverse of a vector
display(GroupOps.inverse(sf.V3(1.2, -3, 2)).T)
```

$\begin{bmatrix} -1.2 & 3 & -2 \end{bmatrix}$

[14]
```python
# Compose two vectors (under addition)
display(GroupOps.compose(sf.V2(1, 2), sf.V2(3, -5)))
```

$\begin{bmatrix} 4 \\ -3 \end{bmatrix}$

+ Code + Text

[15]
```python
# Compose a rotation and its inverse to get identity
R1 = sf.Rot3.from_angle_axis(
    angle=sf.Symbol("theta1"),
    axis=sf.V3(0, 0, 1),
)
display(StorageOps.simplify(GroupOps.compose(R1, R1.inverse()).simplify()))
```

```
<Rot3 <Q xyzw=[0, 0, 0, 1]>>
```

[16]
```python
# Relative rotation using `.between()`
R2 = sf.Rot3.from_angle_axis(
    angle=sf.Symbol("theta2"),
    axis=sf.V3(0, 0, 1),
)
R_delta = GroupOps.between(R1, R2)
display(R2)
display(StorageOps.simplify(GroupOps.compose(R1, R_delta)))
```

```
<Rot3 <Q xyzw=[0, 0, sin(theta2/2), cos(theta2/2)]>>
<Rot3 <Q xyzw=[0, 0, sin(theta2/2), cos(theta2/2)]>>
```

LieGroupOps

LieGroupOps: Group that is also a differentiable manifold, such that calculus applies.

Methods: .tangent_dim(), .from_tangent(), to_tangent(), .retract(), .local_coordinates(), .storage_D_tangent()

[17]
```python
# Underlying dimension of a 3D rotation's tangent space
display(LieGroupOps.tangent_dim(sf.Rot3))
```

3

[18]
```python
# Exponential map (tangent space vector wrt identity element) for a 2D rotation
angle = sf.Symbol("theta")
rot2 = LieGroupOps.from_tangent(sf.Rot2, [angle])
display(rot2.to_rotation_matrix())
```

$\begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$

[19]
```python
# Logarithmic map (tangent space wrt identity element -> element) of the rotation
display(LieGroupOps.to_tangent(rot2))
```

$$[\operatorname{atan}_2(\sin(\theta), \cos(\theta))]$$

[20]
```python
# Exponential map of a vector type is a no-op
display(LieGroupOps.from_tangent(sf.V5(), [1, 2, 3, 4, 5]).T)
```

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 \end{bmatrix}$$

[21]
```python
# Retract perturbs the given element in the tangent space and returns the
# updated element
rot2_perturbed = LieGroupOps.retract(rot2, [sf.Symbol("delta")])
display(rot2_perturbed.to_rotation_matrix())
```

$$\begin{bmatrix} -\sin(\delta)\sin(\theta) + \cos(\delta)\cos(\theta) & -\sin(\delta)\cos(\theta) - \sin(\theta)\cos(\delta) \\ \sin(\delta)\cos(\theta) + \sin(\theta)\cos(\delta) & -\sin(\delta)\sin(\theta) + \cos(\delta)\cos(\theta) \end{bmatrix}$$

[22]
```python
# Local coordinates compute the tangent space perturbation between one element
# and another
display(StorageOps.simplify(LieGroupOps.local_coordinates(rot2, rot2_perturbed)))
```

$$[\operatorname{atan}_2(\sin(\delta), \cos(\delta))]$$

[23]
```python
# storage_D_tangent computes the jacobian of the storage space of an object with
# respect to the tangent space around the element.

# A 2D rotation is represented by a complex number, so storage_D_tangent
# represents how that complex number will change given an infinitesimal
# perturbation in the tangent space
display(LieGroupOps.storage_D_tangent(rot2))
```

$$\begin{bmatrix} -\sin(\theta) \\ \cos(\theta) \end{bmatrix}$$

[Cameras]

\+ Code + Text

```
[2] pip install symforce
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: symforce in /usr/local/lib/python3.8/dist-packages (0.7.0)
Requirement already satisfied: skymarshal==0.7.0 in /usr/local/lib/python3.8/dist-packages (from symforce) (0.7.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.8/dist-packages (from symforce) (1.21.6)
Requirement already satisfied: black in /usr/local/lib/python3.8/dist-packages (from symforce) (22.10.0)
Requirement already satisfied: scipy in /usr/local/lib/python3.8/dist-packages (from symforce) (1.7.3)
Requirement already satisfied: clang-format in /usr/local/lib/python3.8/dist-packages (from symforce) (15.0.4)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.8/dist-packages (from symforce) (2.11.3)
Requirement already satisfied: symforce-sym==0.7.0 in /usr/local/lib/python3.8/dist-packages (from symforce) (0.7.0)
Requirement already satisfied: sympy~=1.11.1 in /usr/local/lib/python3.8/dist-packages (from symforce) (1.11.1)
Requirement already satisfied: graphviz in /usr/local/lib/python3.8/dist-packages (from symforce) (0.10.1)
Requirement already satisfied: ply in /usr/local/lib/python3.8/dist-packages (from skymarshal==0.7.0->symforce) (3.11)
Requirement already satisfied: argh in /usr/local/lib/python3.8/dist-packages (from skymarshal==0.7.0->symforce) (0.26.2)
Requirement already satisfied: six in /usr/local/lib/python3.8/dist-packages (from skymarshal==0.7.0->symforce) (1.15.0)
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.8/dist-packages (from sympy~=1.11.1->symforce) (1.2.1)
Requirement already satisfied: pathspec>=0.9.0 in /usr/local/lib/python3.8/dist-packages (from black->symforce) (0.10.2)
Requirement already satisfied: click>=8.0.0 in /usr/local/lib/python3.8/dist-packages (from black->symforce) (8.1.3)
Requirement already satisfied: tomli>=1.1.0 in /usr/local/lib/python3.8/dist-packages (from black->symforce) (2.0.1)
Requirement already satisfied: mypy-extensions>=0.4.3 in /usr/local/lib/python3.8/dist-packages (from black->symforce) (0.4.3)
Requirement already satisfied: typing-extensions>=3.10.0.0 in /usr/local/lib/python3.8/dist-packages (from black->symforce) (4.1.1)
Requirement already satisfied: platformdirs>=2 in /usr/local/lib/python3.8/dist-packages (from black->symforce) (2.5.4)
Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python3.8/dist-packages (from jinja2->symforce) (2.0.1)
```

```
[3] # Setup
import symforce

symforce.set_symbolic_api("sympy")
symforce.set_log_level("warning")

from symforce.notebook_util import display
import symforce.symbolic as sf
```

```
[4] # creating a linear camera calibration object :
linear_camera_cal = sf.LinearCameraCal.symbolic("cal")
display(linear_camera_cal)
```

```
<LinearCameraCal
  focal_length=[cal.f_x, cal.f_y],
  principal_point=[cal.c_x, cal.c_y],
  distortion_coeffs=[]>
```

\+ Code + Text

```
[5] # deproject points written in the camera frame as so:
camera_point = sf.V3.symbolic("p")
camera_ray, _ = linear_camera_cal.camera_ray_from_pixel(camera_point)
display(camera_ray)
```

$$\left[\begin{matrix}\frac{-cal.c_x + p_0}{cal.f_x}\\ \frac{-cal.c_y + p_1}{cal.f_y}\\ 1\end{matrix}\right]$$

```
[8] camera_point_reprojected, _ = linear_camera_cal.pixel_from_camera_point(
        camera_ray,
    )
    display(camera_point_reprojected)
```

$$\left[\begin{matrix}p_0\\ p_1\end{matrix}\right]$$

```
[9] # using camera calibration objects, can create cameras with additional parameters, such as an image size:
linear_camera = sf.Camera(
    calibration=sf.LinearCameraCal(
        focal_length=(440, 400),
        principal_point=(320, 240),
    ),
    image_size=(640, 480),
)
display(linear_camera)
```

```
<Camera
  CameraCal=<LinearCameraCal
  focal_length=[440, 400],
  principal_point=[320, 240],
  distortion_coeffs=[]>
  image_size=[640, 480]>
```

When projecting points into the image frame, can check whether the resulting point is in the bounds determined by image_size:

```python
point_in_FOV = sf.V3(0, 0, 1)
point_outside_FOV = sf.V3(100, 0, 1)
for point in (point_in_FOV, point_outside_FOV):
    pixel, is_valid = linear_camera.pixel_from_camera_point(point)
    print(
        "point={} -> pixel={}, is_valid={}".format(
            point.to_storage(),
            pixel.to_storage(),
            is_valid,
        )
    )
```

```
point=[0, 0, 1] -> pixel=[320, 240], is_valid=1
point=[100, 0, 1] -> pixel=[44320, 240], is_valid=0
```

```python
# create a camera with a given pose:
linear_posed_camera = sf.PosedCamera(
    pose=sf.Pose3(
        # camera is spun 180 degrees about y-axis
        R=sf.Rot3.from_yaw_pitch_roll(0, sf.pi, 0),
        t=sf.V3(),
    ),
    calibration=linear_camera.calibration,
    image_size=(640, 480),
)
display(linear_posed_camera)
```

```
<PosedCamera
  Pose=<Pose3 R=<Rot3 <Q xyzw=[0, 1, 0, 0]>>, t=(0, 0, 0)>
  Camera=<PosedCamera
  CameraCal=<LinearCameraCal
  focal_length=[440, 400],
  principal_point=[320, 240],
  distortion_coeffs=[]>
  image_size=[640, 480]>>
```

```python
# the given pose can be used to transform points between a global frame and the image frame:
global_point = sf.V3(0, 0, -1)
print(
    "point in global coordinates={} (in camera coordinates={})".format(
        global_point.to_storage(),
        (linear_posed_camera.pose * global_point).to_storage(),
    )
)

pixel, is_valid = linear_posed_camera.pixel_from_global_point(global_point)
print(
    "global_point={} -> pixel={}, is_valid={}".format(
        global_point.to_storage(), pixel.to_storage(), is_valid
    )
)
```

```
point in global coordinates=[0, 0, -1] (in camera coordinates=[0, 0, 1])
global_point=[0, 0, -1] -> pixel=[320, 240], is_valid=1
```

```python
# can also transform points in pixel coordinates back into the global frame (given a range):
range_to_point = (global_point - linear_posed_camera.pose.t).norm()
global_point_reprojected, is_valid = linear_posed_camera.global_point_from_pixel(
    pixel, range_to_point=range_to_point
)
display(global_point_reprojected)
```

$$\begin{bmatrix} 0 \\ 0 \\ -1.0 \end{bmatrix}$$

Can warp points between two posed cameras given the location of the pixel in the source camera, the inverse range to the point, and the target camera to warp the point into.

[14]
```
# Perturb second camera slightly from first (small angular change in roll)
perturbed_rotation = linear_posed_camera.pose.R * sf.Rot3.from_yaw_pitch_roll(0, 0, 0.5)
target_posed_cam = sf.PosedCamera(
    pose=sf.Pose3(R=perturbed_rotation, t=sf.V3()),
    calibration=linear_camera.calibration,
)
# Warp pixel from source camera into target camera given inverse range
target_pixel, is_valid = linear_posed_camera.warp_pixel(
    pixel=sf.V2(320, 240),
    inverse_range=1.0,
    target_cam=target_posed_cam,
)
display(target_pixel)
```

$$\begin{bmatrix} 320 \\ 458.520995937516 \end{bmatrix}$$

[15]
```
# In the examples above used a linear calibration, but can use other types of calibrations as well:
atan_cam = sf.ATANCameraCal(
    focal_length=[380.0, 380.0],
    principal_point=[320.0, 240.0],
    omega=0.35,
)
camera_ray, is_valid = atan_cam.camera_ray_from_pixel(sf.V2(50.0, 50.0))
display(camera_ray)
pixel, is_valid = atan_cam.pixel_from_camera_point(camera_ray)
display(pixel)
```

$$\begin{bmatrix} -0.72576759882138 \\ -0.510725347318749 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} 49.9999999999999 \\ 50.0 \end{bmatrix}$$

[Values]

+ Code + Text

```
pip install symforce
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: symforce in /usr/local/lib/python3.8/dist-packages (0.7.0)
Requirement already satisfied: black in /usr/local/lib/python3.8/dist-packages (from symforce) (22.10.0)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.8/dist-packages (from symforce) (2.11.3)
Requirement already satisfied: clang-format in /usr/local/lib/python3.8/dist-packages (from symforce) (15.0.4)
Requirement already satisfied: scipy in /usr/local/lib/python3.8/dist-packages (from symforce) (1.7.3)
Requirement already satisfied: graphviz in /usr/local/lib/python3.8/dist-packages (from symforce) (0.10.1)
Requirement already satisfied: sympy~=1.11.1 in /usr/local/lib/python3.8/dist-packages (from symforce) (1.11.1)
Requirement already satisfied: symforce-sym==0.7.0 in /usr/local/lib/python3.8/dist-packages (from symforce) (0.7.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.8/dist-packages (from symforce) (1.21.6)
Requirement already satisfied: skymarshal==0.7.0 in /usr/local/lib/python3.8/dist-packages (from symforce) (0.7.0)
Requirement already satisfied: six in /usr/local/lib/python3.8/dist-packages (from skymarshal==0.7.0->symforce) (1.15.0)
Requirement already satisfied: argh in /usr/local/lib/python3.8/dist-packages (from skymarshal==0.7.0->symforce) (0.26.2)
Requirement already satisfied: ply in /usr/local/lib/python3.8/dist-packages (from skymarshal==0.7.0->symforce) (3.11)
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.8/dist-packages (from sympy~=1.11.1->symforce) (1.2.1)
Requirement already satisfied: platformdirs>=2 in /usr/local/lib/python3.8/dist-packages (from black->symforce) (2.5.4)
Requirement already satisfied: pathspec>=0.9.0 in /usr/local/lib/python3.8/dist-packages (from black->symforce) (0.10.2)
Requirement already satisfied: mypy-extensions>=0.4.3 in /usr/local/lib/python3.8/dist-packages (from black->symforce) (0.4.3)
Requirement already satisfied: tomli>=1.1.0 in /usr/local/lib/python3.8/dist-packages (from black->symforce) (2.0.1)
Requirement already satisfied: click>=8.0.0 in /usr/local/lib/python3.8/dist-packages (from black->symforce) (8.1.3)
Requirement already satisfied: typing-extensions>=3.10.0.0 in /usr/local/lib/python3.8/dist-packages (from black->symforce) (4.1.1)
Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python3.8/dist-packages (from jinja2->symforce) (2.0.1)
```

[3]
```
# Setup
import symforce

symforce.set_symbolic_api("sympy")
symforce.set_log_level("warning")

import symforce.symbolic as sf
from symforce.values import Values
from symforce.notebook_util import display, display_code, display_code_file
```

[4]
```
inputs = Values(
    x=sf.Symbol("x"),
    y=sf.Rot2.symbolic("c"),
)
display(inputs)
```

```
Values(
  x: x,
  y: <Rot2 <C real=c_re, imag=c_im>>,
)
```

+ Code + Text

[5]
```
# The .add() method can add a symbol using its name as the key:
inputs.add(sf.Symbol("foo"))
display(inputs)
```

```
Values(
  x: x,
  y: <Rot2 <C real=c_re, imag=c_im>>,
  foo: foo,
)
```

[6]
```
# Adding sub-values are well encouraged:
x, y = sf.symbols("x y")
expr = x ** 2 + sf.sin(y) / x ** 2
inputs["states"] = Values(p=expr)
display(inputs)
```

```
Values(
  x: x,
  y: <Rot2 <C real=c_re, imag=c_im>>,
  foo: foo,
  states:   Values(
    p: x**2 + sin(y)/x**2,
  ),
)
```

[7]
```
# A Values serializes to a depth-first traversed list. This means it implements StorageOps:
display(inputs.to_storage())
```

$$\left[x, \ c_{re}, \ c_{im}, \ foo, \ x^{2} + \frac{\sin{\left(y \right)}}{x^{2}}\right]$$

[8]
```
# can also get a flattened lists of keys and values, with . separation for sub-values:
display(inputs.items_recursive())
```

```
[('x', x),
 ('y', <Rot2 <C real=c_re, imag=c_im>>),
 ('foo', foo),
 ('states.p', x**2 + sin(y)/x**2)]
```

Note that there is a .keys_recursive() and a .values_recursive() which return flattened lists of keys and values respectively:

```
display(inputs.keys_recursive())
display(inputs.values_recursive())
```

```
['x', 'y', 'foo', 'states.p']
[x, <Rot2 <C real=c_re, imag=c_im>>, foo, x**2 + sin(y)/x**2]
```

To fully reconstruct the types in the Values from the serialized scalars, need an index that describes which parts of the serialized list correspond to which types. The spec is T.Dict[str, IndexEntry] where IndexEntry has attributes offset, storage_dim, datatype, shape, item_index:

```
[10] index = inputs.index()
     index
```

```
OrderedDict([('x',
  IndexEntry(offset=0, storage_dim=1, _module='builtins', _qualname='float', shape=None, item_index=None)),
 ('y',
  IndexEntry(offset=1, storage_dim=2, _module='symforce.geo.rot2', _qualname='Rot2', shape=None, item_index=None)),
 ('foo',
  IndexEntry(offset=3, storage_dim=1, _module='builtins', _qualname='float', shape=None, item_index=None)),
 ('states',
  IndexEntry(offset=4, storage_dim=1, _module='symforce.values.values', _qualname='Values', shape=None, item_index=OrderedDict([('p', IndexEntry(offset=0, storage_dim=1, _module='builtins', _qualname='float', shape=None, item_index=None))])))])
```

```
[11] # With a serialized list and an index, can get the values back:
     inputs2 = Values.from_storage_index(inputs.to_storage(), index)
     assert inputs == inputs2
     display(inputs)
```

```
Values(
  x: x,
  y: <Rot2 <C real=c_re, imag=c_im>>,
  foo: foo,
  states:   Values(
    p: x**2 + sin(y)/x**2,
  ),
)
```

```
[12] # The item_index is a recursive structure that can contain the index for a sub-values:
     item_index = inputs.index()["states"].item_index
     assert item_index == inputs["states"].index()
```

```
[13] # Can also set sub-values directly with dot notation in the keys. They get split up:
     inputs["states.blah"] = 3
     display(inputs)
```

```
Values(
  x: x,
  y: <Rot2 <C real=c_re, imag=c_im>>,
  foo: foo,
  states:   Values(
    p: x**2 + sin(y)/x**2,
    blah: 3,
  ),
)
```

```
[14] # The .attr field also allows attribute access rather than key access:
     assert inputs["states.p"] is inputs["states"]["p"] is inputs.attr.states.p
     display(inputs.attr.states.p)
```

$$x^2 + \frac{\sin(y)}{x^2}$$

SymForce adds the concept of a name scope to namespace symbols. Within a scope block, symbol names get prefixed with the scope name:

```
[15] with sf.scope("params"):
         s = sf.Symbol("cost")
     display(s)
```

$params.cost$

A common use case is to call a function that adds symbols within name scope to avoid name collisions, also chain name scopes:

```
[16] v = Values()
     v.add(sf.Symbol("x"))
     with sf.scope("foo"):
         v.add(sf.Symbol("x"))
         with sf.scope("bar"):
             v.add(sf.Symbol("x"))
     display(v)
     display(v.attr.foo.bar.x)
```

```
Values(
  x: x,
  foo:   Values(
    x: foo.x,
    bar:     Values(
      x: foo.bar.x,
    ),
  ),
)
```

$foo.bar.x$

The values class also provides a .scope() method that not only applies the scope to symbol names but also to keys added to the Values:

```
[17] v = Values()
     with v.scope("hello"):
         v["y"] = x ** 2
         v["z"] = sf.Symbol("z")
     v
```

```
Values(
  hello:   Values(
    y: x**2,
    z: hello.z,
  ),
)
```

+ Code + Text

This flexible set of features provided by the Values class allows conveniently building up large expressions, and acts as the interface to code generation.

Lie Group Operations

One useful feature of Values objects is that element-wise Lie group operations on can be performed on them.

```
[18] lie_vals = Values()
     lie_vals["scalar"] = sf.Symbol("x")
     lie_vals["rot3"] = sf.Rot3.symbolic("rot")

     sub_lie_vals = Values()
     sub_lie_vals["pose3"] = sf.Pose3.symbolic("pose")
     sub_lie_vals["vec"] = sf.V3.symbolic("vec")

     lie_vals["sub_vals"] = sub_lie_vals

     display(lie_vals)
```

```
Values(
  scalar: x,
  rot3: <Rot3 <Q xyzw=[rot_x, rot_y, rot_z, rot_w]>>,
  sub_vals:   Values(
    pose3: <Pose3 R=<Rot3 <Q xyzw=[pose.R_x, pose.R_y, pose.R_z, pose.R_w]>>, t=(pose.t0, pose.t1, pose.t2)>,
    vec: Matrix([
[vec0],
[vec1],
[vec2]]),
  ),
)
```

```
[19] display(lie_vals.tangent_dim())
     display(len(lie_vals.to_tangent()))
```

13
13

Importantly, can compute the jacobian of the storage space of the object with respect to its tangent space:

```
[20] display(lie_vals.storage_D_tangent())
```

$$\left[\begin{array}{ccccccccccccc}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & \frac{rot_w}{2} & -\frac{rot_z}{2} & \frac{rot_y}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & \frac{rot_z}{2} & \frac{rot_w}{2} & -\frac{rot_x}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -\frac{rot_y}{2} & \frac{rot_x}{2} & \frac{rot_w}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -\frac{rot_x}{2} & -\frac{rot_y}{2} & -\frac{rot_z}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \frac{pose.R_w}{2} & -\frac{pose.R_z}{2} & \frac{pose.R_y}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \frac{pose.R_z}{2} & \frac{pose.R_w}{2} & -\frac{pose.R_x}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -\frac{pose.R_y}{2} & \frac{pose.R_x}{2} & \frac{pose.R_w}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -\frac{pose.R_x}{2} & -\frac{pose.R_y}{2} & -\frac{pose.R_z}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{array}\right]$$

This means that can use the elements of the object to compute a residual, and then compute the jacobian of such a residual with respect to the tangent space of our values object.

```
[21] residual = sf.Matrix(6, 1)
     residual[0:3, 0] = lie_vals["rot3"] * lie_vals["sub_vals.vec"]
     residual[3:6, 0] = lie_vals["sub_vals.pose3"] * lie_vals["sub_vals.vec"]
     display(residual)
```

$$\left[\begin{array}{c}
vec_0\left(-2rot_y^2 - 2rot_z^2 + 1\right) + vec_1\left(-2rot_w rot_z + 2rot_x rot_y\right) + vec_2 \cdot \left(2rot_w rot_y + 2rot_x rot_z\right) \\
vec_0 \cdot \left(2rot_w rot_z + 2rot_x rot_y\right) + vec_1\left(-2rot_x^2 - 2rot_z^2 + 1\right) + vec_2\left(-2rot_w rot_x + 2rot_y rot_z\right) \\
vec_0\left(-2rot_w rot_y + 2rot_x rot_z\right) + vec_1 \cdot \left(2rot_w rot_x + 2rot_y rot_z\right) + vec_2\left(-2rot_x^2 - 2rot_y^2 + 1\right) \\
pose.t0 + vec_0\left(-2pose.R_y^2 - 2pose.R_z^2 + 1\right) + vec_1\left(-2pose.R_w pose.R_z + 2pose.R_x pose.R_y\right) + vec_2 \cdot \left(2pose.R_w pose.R_y + 2pose.R_x pose.R_z\right) \\
pose.t1 + vec_0 \cdot \left(2pose.R_w pose.R_z + 2pose.R_x pose.R_y\right) + vec_1\left(-2pose.R_x^2 - 2pose.R_z^2 + 1\right) + vec_2\left(-2pose.R_w pose.R_x + 2pose.R_y pose.R_z\right) \\
pose.t2 + vec_0\left(-2pose.R_w pose.R_y + 2pose.R_x pose.R_z\right) + vec_1 \cdot \left(2pose.R_w pose.R_x + 2pose.R_y pose.R_z\right) + vec_2\left(-2pose.R_x^2 - 2pose.R_y^2 + 1\right)
\end{array}\right]$$

```
[22] residual_D_tangent = residual.jacobian(lie_vals)
     display(residual_D_tangent.shape)
     display(residual_D_tangent)
```

(6, 13)

$$\left[\begin{array}{ccccc}
0 & vec_1 \cdot \left(2rot_w rot_y + 2rot_x rot_z\right) + vec_2 \cdot \left(2rot_w rot_z - 2rot_x rot_y\right) & vec_0\left(-2rot_w rot_y - 2rot_x rot_z\right) + vec_2\left(rot_w^2 + rot_x^2 - rot_y^2 - rot_z^2\right) & vec_0\left(-2rot_w rot_z + 2rot_x rot_y\right) + vec_1\left(-rot_w^2 - rot_x^2 + rot_y^2 + rot_z^2\right) & \\
0 & vec_1\left(-2rot_w rot_x + 2rot_y rot_z\right) + vec_2\left(-rot_w^2 + rot_x^2 - rot_y^2 + rot_z^2\right) & vec_0 \cdot \left(2rot_w rot_x - 2rot_y rot_z\right) + vec_2 \cdot \left(2rot_w rot_z + 2rot_x rot_y\right) & vec_0\left(rot_w^2 - rot_x^2 + rot_y^2 - rot_z^2\right) + vec_1\left(-2rot_w rot_z - 2rot_x rot_y\right) & \\
0 & vec_1\left(rot_w^2 - rot_x^2 - rot_y^2 + rot_z^2\right) + vec_2\left(-2rot_w rot_x - 2rot_y rot_z\right) & vec_0\left(-rot_w^2 + rot_x^2 + rot_y^2 - rot_z^2\right) + vec_2\left(-2rot_w rot_y + 2rot_x rot_z\right) & vec_0 \cdot \left(2rot_w rot_x + 2rot_y rot_z\right) + vec_1 \cdot \left(2rot_w rot_y - 2rot_x rot_z\right) & \\
0 & 0 & 0 & 0 & vec_1 \cdot (2pose.R_w pose.R_y + 2pose. \\
0 & 0 & 0 & 0 & vec_1 (-2pose.R_w pose.R_x + 2pose.R \\
0 & 0 & 0 & 0 & vec_1 (pose.R_w^2 - pose.R_x^2 - pose.R
\end{array}\right.$$

[Codegen]

Codegen Tutorial

One of the most important features of symforce is the ability to generate computationally efficient code from symbolic expressions. Before progressing, first make sure you are familiar with the other symforce tutorials, especially the Values tutorial.

The typical workflow for generating a function is to define a Python function that operates on symbolic inputs to return the symbolic result. Typically this will look like:

1. Define a Python function that operates on symbolic inputs
2. Create a Codegen object using Codegen.function. Various properties of the function will be deduced automatically; for instance, the name of the generated function is generated from the name of the Python function, and the argument names and types are deduced from the Python function argument names and type annotations.
3. Generate the code in your desired language

Alternately, you may want to define the input and output symbolic Values explicitly, with the following steps:

1. Build an input Values object that defines a symbolic representation of each input to the function. Note that inputs and outputs can be Values objects themselves, which symforce will automatically generate into custom types.
2. Build an output Values object that defines the outputs of the function in terms of the objects in the input Values.

3.Generate the code in your desired language

+ Code + Text

[2]
```
pip install symforce
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: symforce in /usr/local/lib/python3.8/dist-packages (0.7.0)
Requirement already satisfied: black in /usr/local/lib/python3.8/dist-packages (from symforce) (22.10.0)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.8/dist-packages (from symforce) (2.11.3)
Requirement already satisfied: clang-format in /usr/local/lib/python3.8/dist-packages (from symforce) (15.0.4)
Requirement already satisfied: scipy in /usr/local/lib/python3.8/dist-packages (from symforce) (1.7.3)
Requirement already satisfied: numpy in /usr/local/lib/python3.8/dist-packages (from symforce) (1.21.6)
Requirement already satisfied: sympy~=1.11.1 in /usr/local/lib/python3.8/dist-packages (from symforce) (1.11.1)
Requirement already satisfied: symforce-sym==0.7.0 in /usr/local/lib/python3.8/dist-packages (from symforce) (0.7.0)
Requirement already satisfied: skymarshal==0.7.0 in /usr/local/lib/python3.8/dist-packages (from symforce) (0.7.0)
Requirement already satisfied: graphviz in /usr/local/lib/python3.8/dist-packages (from symforce) (0.10.1)
Requirement already satisfied: six in /usr/local/lib/python3.8/dist-packages (from skymarshal==0.7.0->symforce) (1.15.0)
Requirement already satisfied: argh in /usr/local/lib/python3.8/dist-packages (from skymarshal==0.7.0->symforce) (0.26.2)
Requirement already satisfied: ply in /usr/local/lib/python3.8/dist-packages (from skymarshal==0.7.0->symforce) (3.11)
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.8/dist-packages (from sympy~=1.11.1->symforce) (1.2.1)
Requirement already satisfied: mypy-extensions>=0.4.3 in /usr/local/lib/python3.8/dist-packages (from black->symforce) (0.4.3)
Requirement already satisfied: pathspec>=0.9.0 in /usr/local/lib/python3.8/dist-packages (from black->symforce) (0.10.2)
Requirement already satisfied: tomli>=1.1.0 in /usr/local/lib/python3.8/dist-packages (from black->symforce) (2.0.1)
Requirement already satisfied: click>=8.0.0 in /usr/local/lib/python3.8/dist-packages (from black->symforce) (8.1.3)
Requirement already satisfied: typing-extensions>=3.10.0.0 in /usr/local/lib/python3.8/dist-packages (from black->symforce) (4.1.1)
Requirement already satisfied: platformdirs>=2 in /usr/local/lib/python3.8/dist-packages (from black->symforce) (2.5.4)
Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python3.8/dist-packages (from jinja2->symforce) (2.0.1)
```

[3]
```python
# Setup
import numpy as np
import os
import symforce

symforce.set_symbolic_api("symengine")
symforce.set_log_level("warning")

# Set epsilon to a symbol for safe code generation.  For more information, see the Epsilon tutorial:
# https://symforce.org/tutorials/epsilon_tutorial.html
symforce.set_epsilon_to_symbol()

from symforce import codegen
from symforce.codegen import codegen_util
from symforce import ops
import symforce.symbolic as sf
from symforce.values import Values
from symforce.notebook_util import display, display_code, display_code_file
```

Generating from a Python function

First, look at using existing python functions to generate an equivalent function using the codegen package. The inputs to the function are automatically deduced from the signature and type annotations. Additionally, can change how the generated function is declared (e.g. whether to return an object using a return statement or a pointer passed as an argument to the function).

[4]
```python
def az_el_from_point(
    nav_T_cam: sf.Pose3, nav_t_point: sf.Vector3, epsilon: sf.Scalar = 0
) -> sf.Vector2:
    """
    Transform a nav point into azimuth / elevation angles in the
    camera frame.

    Args:
        nav_T_cam (sf.Pose3): camera pose in the world
        nav_t_point (sf.Matrix): nav point
        epsilon (Scalar): small number to avoid singularities

    Returns:
        sf.Matrix: (azimuth, elevation)
    """
    cam_t_point = nav_T_cam.inverse() * nav_t_point
    x, y, z = cam_t_point
    theta = sf.atan2(y, x + epsilon)
    phi = sf.pi / 2 - sf.acos(z / (cam_t_point.norm() + epsilon))
    return sf.V2(theta, phi)
```

[5]
```python
az_el_codegen = codegen.Codegen.function(
    func=az_el_from_point,
    config=codegen.CppConfig(),
)
az_el_codegen_data = az_el_codegen.generate_function()

print("Files generated in {}:\n".format(az_el_codegen_data.output_dir))
for f in az_el_codegen_data.generated_files:
    print("  |- {}".format(os.path.relpath(f, az_el_codegen_data.output_dir)))

display_code_file(az_el_codegen_data.generated_files[0], "C++")
```

```
Files generated in /tmp/sf_codegen_az_el_from_point_yyshbzbo:

  |- cpp/symforce/sym/az_el_from_point.h
```

```cpp
// -----------------------------------------------------------------------------
// This file was autogenerated by symforce from template:
//     function/FUNCTION.h.jinja
// Do NOT modify by hand.
// -----------------------------------------------------------------------------

#pragma once

#include <Eigen/Dense>

#include <sym/pose3.h>

namespace sym {

/**
 * Transform a nav point into azimuth / elevation angles in the
 * camera frame.
 *
 * Args:
 *     nav_T_cam (sf.Pose3): camera pose in the world
 *     nav_t_point (sf.Matrix): nav point
 *     epsilon (Scalar): small number to avoid singularities
 *
 * Returns:
 *     sf.Matrix: (azimuth, elevation)
 */
template <typename Scalar>
Eigen::Matrix<Scalar, 2, 1> AzElFromPoint(const sym::Pose3<Scalar>& nav_T_cam,
                                          const Eigen::Matrix<Scalar, 3, 1>& nav_t_point,
                                          const Scalar epsilon) {
  // Total ops: 78

  // Input arrays
  const Eigen::Matrix<Scalar, 7, 1>& _nav_T_cam = nav_T_cam.Data();

  // Intermediate terms (24)
  const Scalar _tmp0 = 2 * _nav_T_cam[0];
  const Scalar _tmp1 = _nav_T_cam[3] * _tmp0;
  const Scalar _tmp2 = 2 * _nav_T_cam[1];
  const Scalar _tmp3 = _nav_T_cam[2] * _tmp2;
  const Scalar _tmp4 = _tmp1 + _tmp3;
  const Scalar _tmp5 = -2 * std::pow(_nav_T_cam[0], Scalar(2));
  const Scalar _tmp6 = 1 - 2 * std::pow(_nav_T_cam[2], Scalar(2));
  const Scalar _tmp7 = _tmp5 + _tmp6;
  const Scalar _tmp8 = _nav_T_cam[1] * _tmp0;
  const Scalar _tmp9 = 2 * _nav_T_cam[2] * _nav_T_cam[3];
  const Scalar _tmp10 = _tmp8 - _tmp9;
```

```
const Scalar _tmp11 = -_nav_T_cam[4] * _tmp10 - _nav_T_cam[5] * _tmp7 - _nav_T_cam[6] * _tmp4 +
                      _tmp10 * nav_t_point(0, 0) + _tmp4 * nav_t_point(2, 0) +
                      _tmp7 * nav_t_point(1, 0);
const Scalar _tmp12 = -2 * std::pow(_nav_T_cam[1], Scalar(2));
const Scalar _tmp13 = _tmp12 + _tmp6;
const Scalar _tmp14 = _tmp8 + _tmp9;
const Scalar _tmp15 = _nav_T_cam[3] * _tmp2;
const Scalar _tmp16 = _nav_T_cam[2] * _tmp0;
const Scalar _tmp17 = -_tmp15 + _tmp16;
const Scalar _tmp18 = -_nav_T_cam[4] * _tmp13 - _nav_T_cam[5] * _tmp14 - _nav_T_cam[6] * _tmp17 +
                      _tmp13 * nav_t_point(0, 0) + _tmp14 * nav_t_point(1, 0) +
                      _tmp17 * nav_t_point(2, 0);
const Scalar _tmp19 = _tmp18 + epsilon;
const Scalar _tmp20 = -_tmp1 + _tmp3;
const Scalar _tmp21 = _tmp12 + _tmp5 + 1;
const Scalar _tmp22 = _tmp15 + _tmp16;
const Scalar _tmp23 = -_nav_T_cam[4] * _tmp22 - _nav_T_cam[5] * _tmp20 - _nav_T_cam[6] * _tmp21 +
                      _tmp20 * nav_t_point(1, 0) + _tmp21 * nav_t_point(2, 0) +
                      _tmp22 * nav_t_point(0, 0);

// Output terms (1)
Eigen::Matrix<Scalar, 2, 1> _res;

_res(0, 0) =
    std::atan2(_tmp11, _tmp19 + epsilon * ((((_tmp19) > 0) - ((_tmp19) < 0)) + Scalar(0.5)));
_res(1, 0) = -std::acos(_tmp23 / (epsilon + std::sqrt(Scalar(std::pow(_tmp11, Scalar(2)) +
                                                             std::pow(_tmp18, Scalar(2)) +
                                                             std::pow(_tmp23, Scalar(2)))))) +
             Scalar(M_PI_2);
```

Generating function jacobians

[6]

```
codegen_with_jacobians = az_el_codegen.with_jacobians(
    # Just compute wrt the pose and point, not epsilon
    which_args=["nav_T_cam", "nav_t_point"],
    # Include value, not just jacobians
    include_results=True,
)

data = codegen_with_jacobians.generate_function()
from symforce.notebook_util import display_code_file

display_code_file(data.generated_files[0], "C++")
```

```
// -----------------------------------------------------------------------------
// This file was autogenerated by symforce from template:
//     function/FUNCTION.h.jinja
// Do NOT modify by hand.
// -----------------------------------------------------------------------------

#pragma once

#include <Eigen/Dense>

#include <sym/pose3.h>

namespace sym {

/**
 * Transform a nav point into azimuth / elevation angles in the
 * camera frame.
 *
 * Args:
 *     nav_T_cam (sf.Pose3): camera pose in the world
 *     nav_t_point (sf.Matrix): nav point
 *     epsilon (Scalar): small number to avoid singularities
 *
 * Returns:
 *     sf.Matrix: (azimuth, elevation)
 *     res_D_nav_T_cam: (2x6) jacobian of res (2) wrt arg nav_T_cam (6)
 *     res_D_nav_t_point: (2x3) jacobian of res (2) wrt arg nav_t_point (3)
 */
template <typename Scalar>
Eigen::Matrix<Scalar, 2, 1> AzElFromPointWithJacobians01(
    const sym::Pose3<Scalar>& nav_T_cam, const Eigen::Matrix<Scalar, 3, 1>& nav_t_point,
    const Scalar epsilon, Eigen::Matrix<Scalar, 2, 6>* const res_D_nav_T_cam = nullptr,
    Eigen::Matrix<Scalar, 2, 3>* const res_D_nav_t_point = nullptr) {
```

```cpp
// Total ops: 289

// Input arrays
const Eigen::Matrix<Scalar, 7, 1>& _nav_T_cam = nav_T_cam.Data();

// Intermediate terms (93)
const Scalar _tmp0 = std::pow(_nav_T_cam[0], Scalar(2));
const Scalar _tmp1 = 2 * _tmp0;
const Scalar _tmp2 = -_tmp1;
const Scalar _tmp3 = std::pow(_nav_T_cam[2], Scalar(2));
const Scalar _tmp4 = 2 * _tmp3;
const Scalar _tmp5 = 1 - _tmp4;
const Scalar _tmp6 = _tmp2 + _tmp5;
const Scalar _tmp7 = 2 * _nav_T_cam[3];
const Scalar _tmp8 = _nav_T_cam[0] * _tmp7;
const Scalar _tmp9 = 2 * _nav_T_cam[2];
const Scalar _tmp10 = _nav_T_cam[1] * _tmp9;
const Scalar _tmp11 = _tmp10 + _tmp8;
const Scalar _tmp12 = _nav_T_cam[6] * _tmp11;
const Scalar _tmp13 = 2 * _nav_T_cam[0] * _nav_T_cam[1];
const Scalar _tmp14 = _nav_T_cam[3] * _tmp9;
const Scalar _tmp15 = -_tmp14;
const Scalar _tmp16 = _tmp13 + _tmp15;
const Scalar _tmp17 = _nav_T_cam[4] * _tmp16;
const Scalar _tmp18 = _tmp11 * nav_t_point(2, 0) + _tmp16 * nav_t_point(0, 0);
const Scalar _tmp19 =
    -_nav_T_cam[5] * _tmp6 - _tmp12 - _tmp17 + _tmp18 + _tmp6 * nav_t_point(1, 0);
const Scalar _tmp20 = std::pow(_nav_T_cam[1], Scalar(2));
const Scalar _tmp21 = 2 * _tmp20;
const Scalar _tmp22 = -_tmp21;
const Scalar _tmp23 = _tmp22 + _tmp5;
const Scalar _tmp24 = _tmp13 + _tmp14;
const Scalar _tmp25 = _nav_T_cam[5] * _tmp24;
const Scalar _tmp26 = _nav_T_cam[1] * _tmp7;
const Scalar _tmp27 = -_tmp26;
const Scalar _tmp28 = _nav_T_cam[0] * _tmp9;
const Scalar _tmp29 = _tmp27 + _tmp28;
const Scalar _tmp30 = _nav_T_cam[6] * _tmp29;
const Scalar _tmp31 = _tmp24 * nav_t_point(1, 0) + _tmp29 * nav_t_point(2, 0);
const Scalar _tmp32 =
    -_nav_T_cam[4] * _tmp23 + _tmp23 * nav_t_point(0, 0) - _tmp25 - _tmp30 + _tmp31;
const Scalar _tmp33 = _tmp32 + epsilon;
const Scalar _tmp34 = _tmp33 + epsilon * ((((_tmp33) > 0) - ((_tmp33) < 0)) + Scalar(0.5));
const Scalar _tmp35 = _tmp2 + _tmp22 + 1;
const Scalar _tmp36 = -_tmp8;
const Scalar _tmp37 = _tmp10 + _tmp36;
const Scalar _tmp38 = _nav_T_cam[5] * _tmp37;
const Scalar _tmp39 = _tmp26 + _tmp28;
const Scalar _tmp40 = _nav_T_cam[4] * _tmp39;
```

```
  const Scalar _tmp41 = _tmp37 * nav_t_point(1, 0) + _tmp39 * nav_t_point(0, 0);
  const Scalar _tmp42 =
      -_nav_T_cam[6] * _tmp35 + _tmp35 * nav_t_point(2, 0) - _tmp38 - _tmp40 + _tmp41;
  const Scalar _tmp43 = std::pow(_tmp19, Scalar(2));
  const Scalar _tmp44 = std::pow(_tmp42, Scalar(2));
  const Scalar _tmp45 = std::sqrt(Scalar(std::pow(_tmp32, Scalar(2)) + _tmp43 + _tmp44));
  const Scalar _tmp46 = _tmp45 + epsilon;
  const Scalar _tmp47 = Scalar(1.0) / (_tmp46);
  const Scalar _tmp48 = std::pow(_nav_T_cam[3], Scalar(2));
  const Scalar _tmp49 = -_tmp0;
  const Scalar _tmp50 = -_tmp20;
  const Scalar _tmp51 = _tmp3 + _tmp48 + _tmp49 + _tmp50;
  const Scalar _tmp52 =
      -_nav_T_cam[6] * _tmp51 - _tmp38 - _tmp40 + _tmp41 + _tmp51 * nav_t_point(2, 0);
  const Scalar _tmp53 = std::pow(_tmp34, Scalar(2));
  const Scalar _tmp54 = Scalar(1.0) / (_tmp43 + _tmp53);
  const Scalar _tmp55 = -_tmp10;
  const Scalar _tmp56 = _tmp36 + _tmp55;
  const Scalar _tmp57 = -_tmp48;
  const Scalar _tmp58 = _tmp3 + _tmp57;
  const Scalar _tmp59 = _tmp0 + _tmp50;
  const Scalar _tmp60 = _tmp58 + _tmp59;
  const Scalar _tmp61 = -_tmp13;
  const Scalar _tmp62 = _tmp14 + _tmp61;
  const Scalar _tmp63 = -_nav_T_cam[4] * _tmp62 - _nav_T_cam[5] * _tmp60 - _nav_T_cam[6] * _tmp56 +
                        _tmp56 * nav_t_point(2, 0) + _tmp60 * nav_t_point(1, 0) +
                        _tmp62 * nav_t_point(0, 0);
  const Scalar _tmp64 = 2 * _tmp19;
  const Scalar _tmp65 = 2 * _tmp42;
  const Scalar _tmp66 = std::pow(_tmp46, Scalar(-2));
  const Scalar _tmp67 = (Scalar(1) / Scalar(2)) * _tmp42 * _tmp66 / _tmp45;
  const Scalar _tmp68 = std::pow(Scalar(-_tmp44 * _tmp66 + 1), Scalar(Scalar(-1) / Scalar(2)));
  const Scalar _tmp69 = _tmp55 + _tmp8;
  const Scalar _tmp70 = -_tmp3;
  const Scalar _tmp71 = _tmp0 + _tmp20 + _tmp57 + _tmp70;
  const Scalar _tmp72 = -_tmp28;
  const Scalar _tmp73 = _tmp27 + _tmp72;
  const Scalar _tmp74 = -_nav_T_cam[4] * _tmp73 - _nav_T_cam[5] * _tmp69 - _nav_T_cam[6] * _tmp71 +
                        _tmp69 * nav_t_point(1, 0) + _tmp71 * nav_t_point(2, 0) +
                        _tmp73 * nav_t_point(0, 0);
  const Scalar _tmp75 = _tmp48 + _tmp70;
  const Scalar _tmp76 = _tmp59 + _tmp75;
  const Scalar _tmp77 =
      -_nav_T_cam[4] * _tmp76 - _tmp25 - _tmp30 + _tmp31 + _tmp76 * nav_t_point(0, 0);
  const Scalar _tmp78 = 2 * _tmp32;
  const Scalar _tmp79 = _tmp20 + _tmp49;
  const Scalar _tmp80 = _tmp75 + _tmp79;
  const Scalar _tmp81 =
      -_nav_T_cam[5] * _tmp80 - _tmp12 - _tmp17 + _tmp18 + _tmp80 * nav_t_point(1, 0);
  const Scalar _tmp82 = _tmp19 / _tmp53;
```

```
  const Scalar _tmp83 = _tmp58 + _tmp79;
  const Scalar _tmp84 = _tmp15 + _tmp61;
  const Scalar _tmp85 = _tmp26 + _tmp72;
  const Scalar _tmp86 = -_nav_T_cam[4] * _tmp83 - _nav_T_cam[5] * _tmp84 - _nav_T_cam[6] * _tmp85 +
                        _tmp83 * nav_t_point(0, 0) + _tmp84 * nav_t_point(1, 0) +
                        _tmp85 * nav_t_point(2, 0);
  const Scalar _tmp87 = Scalar(1.0) / (_tmp34);
```

Code generation using implicit functions

Next, look at generating functions using a list of input variables and output expressions that are a function of those variables. In this case don't need to explicitly define a function in python, but can instead generate one directly using the codegen package.

Let's set up an example for the double pendulum. Will skip the derivation and just define the equations of motion for the angular acceleration of the two links:

[7]
```
# Define symbols
L = sf.V2.symbolic("L").T  # Length of the two links
m = sf.V2.symbolic("m").T  # Mass of the two links
ang = sf.V2.symbolic("a").T  # Angle of the two links
dang = sf.V2.symbolic("da").T  # Angular velocity of the two links
g = sf.Symbol("g")  # Gravity
```

[8]
```
# Angular acceleration of the first link
ddang_0 = (
    -g * (2 * m[0] + m[1]) * sf.sin(ang[0])
    - m[1] * g * sf.sin(ang[0] - 2 * ang[1])
    - 2
    * sf.sin(ang[0] - ang[1])
    * m[1]
    * (dang[1] * 2 * L[1] + dang[0] * 2 * L[0] * sf.cos(ang[0] - ang[1]))
) / (L[0] * (2 * m[0] + m[1] - m[1] * sf.cos(2 * ang[0] - 2 * ang[1])))
display(ddang_0)
```

$$\frac{-gm_1\sin(a_0 - 2a_1) - g(2m_0 + m_1)\sin(a_0) - 2m_1\cdot(2L_0da_0\cos(a_0 - a_1) + 2L_1da_1)\sin(a_0 - a_1)}{L_0\cdot(2m_0 - m_1\cos(2a_0 - 2a_1) + m_1)}$$

[9]
```
# Angular acceleration of the second link
ddang_1 = (
    2
    * sf.sin(ang[0] - ang[1])
    * (
        dang[0] ** 2 * L[0] * (m[0] + m[1])
        + g * (m[0] + m[1]) * sf.cos(ang[0])
        + dang[1] ** 2 * L[1] * m[1] * sf.cos(ang[0] - ang[1])
    )
) / (L[1] * (2 * m[0] + m[1] - m[1] * sf.cos(2 * ang[0] - 2 * ang[1])))
display(ddang_1)
```

$$\frac{2\left(L_0da_0^2(m_0 + m_1) + L_1da_1^2m_1\cos(a_0 - a_1) + g(m_0 + m_1)\cos(a_0)\right)\sin(a_0 - a_1)}{L_1\cdot(2m_0 - m_1\cos(2a_0 - 2a_1) + m_1)}$$

[10]
```
# Organize the input symbols into a Values hierarchy:
inputs = Values()

inputs["ang"] = ang
inputs["dang"] = dang

with inputs.scope("constants"):
    inputs["g"] = g

with inputs.scope("params"):
    inputs["L"] = L
    inputs["m"] = m

display(inputs)
```

```
Values(
  ang: [a0, a1],
  dang: [da0, da1],
  constants:   Values(
    g: g,
  ),
  params:   Values(
    L: [L0, L1],
    m: [m0, m1],
  ),
)
```

[11]
```python
# The output will simply be a 2-vector of the angular accelerations:
outputs = Values(ddang=sf.V2(ddang_0, ddang_1))

display(outputs)
```

```
Values(
  ddang: [(-g*(2*m0 + m1)*sin(a0) + g*sin(-a0 + 2*a1)*m1 + 2*sin(-a0 + a1)*m1*(2*L1*da1 + 2*cos(-a0 + a1)*L0*da0))/(L0*(2*m0 + m1 - cos(-2*a0 + 2*a1)*m1))]
[-2*sin(-a0 + a1)*(g*(m0 + m1)*cos(a0) + (m0 + m1)*L0*da0**2 + cos(-a0 + a1)*m1*L1*da1**2)/(L1*(2*m0 + m1 - cos(-2*a0 + 2*a1)*m1))],
)
```

[12]
```python
# Run code generation to produce an executable module (in a temp directory if none provided):
double_pendulum = codegen.Codegen(
    inputs=inputs,
    outputs=outputs,
    config=codegen.CppConfig(),
    name="double_pendulum",
    return_key="ddang",
)
double_pendulum_data = double_pendulum.generate_function()

# Print what we generated
print("Files generated in {}:\n".format(double_pendulum_data.output_dir))
for f in double_pendulum_data.generated_files:
    print("  |- {}".format(os.path.relpath(f, double_pendulum_data.output_dir)))
```

```
Files generated in /tmp/sf_codegen_double_pendulum_s5dvllzi:

  |- lcmtypes/double_pendulum.lcm
  |- cpp/symforce/sym/double_pendulum.h
```

[13]
```python
display_code_file(double_pendulum_data.function_dir / "double_pendulum.h", "C++")
```

```cpp
// -----------------------------------------------------------------------------
// This file was autogenerated by symforce from template:
//     function/FUNCTION.h.jinja
// Do NOT modify by hand.
// -----------------------------------------------------------------------------

#pragma once

#include <Eigen/Dense>

#include <lcmtypes/sym/constants_t.hpp>
#include <lcmtypes/sym/params_t.hpp>

namespace sym {

/**
 * This function was autogenerated. Do not modify by hand.
 *
 * Args:
 *     ang: Matrix12
 *     dang: Matrix12
 *     constants: Values
 *     params: Values
 *
 * Outputs:
 *     ddang: Matrix21
 */
template <typename Scalar>
Eigen::Matrix<Scalar, 2, 1> DoublePendulum(const Eigen::Matrix<Scalar, 1, 2>& ang,
                                           const Eigen::Matrix<Scalar, 1, 2>& dang,
                                           const sym::constants_t& constants,
                                           const sym::params_t& params) {
  // Total ops: 50

  // Input arrays

  // Intermediate terms (8)
  const Scalar _tmp0 = 2 * ang(0, 1);
  const Scalar _tmp1 = 2 * params.m.data()[0] + params.m.data()[1];
  const Scalar _tmp2 = Scalar(1.0) / (_tmp1 - params.m.data()[1] * std::cos(_tmp0 - 2 * ang(0, 0)));
  const Scalar _tmp3 = -ang(0, 0);
  const Scalar _tmp4 = _tmp3 + ang(0, 1);
  const Scalar _tmp5 = std::cos(_tmp4);
  const Scalar _tmp6 = 2 * std::sin(_tmp4);
  const Scalar _tmp7 = params.m.data()[0] + params.m.data()[1];
```

[13]
```cpp
  // Output terms (1)
  Eigen::Matrix<Scalar, 2, 1> _ddang;

  _ddang(0, 0) =
      _tmp2 *
      (-_tmp1 * constants.g * std::sin(ang(0, 0)) +
       _tmp6 * params.m.data()[1] *
           (2 * _tmp5 * dang(0, 0) * params.L.data()[0] + 2 * dang(0, 1) * params.L.data()[1]) +
       constants.g * params.m.data()[1] * std::sin(_tmp0 + _tmp3)) /
      params.L.data()[0];
  _ddang(1, 0) =
      -_tmp2 * _tmp6 *
      (_tmp5 * std::pow(dang(0, 1), Scalar(2)) * params.L.data()[1] * params.m.data()[1] +
       _tmp7 * constants.g * std::cos(ang(0, 0)) +
       _tmp7 * std::pow(dang(0, 0), Scalar(2)) * params.L.data()[0]) /
      params.L.data()[1];

  return _ddang;
}  // NOLINT(readability/fn_size)

// NOLINTNEXTLINE(readability/fn_size)
}  // namespace sym
```

[14]
```python
# can also generate functions with different function declarations:
# Function using structs as inputs and outputs (returned as pointer arg)
input_values = Values(inputs=inputs)
output_values = Values(outputs=outputs)
namespace = "double_pendulum"
double_pendulum_values = codegen.Codegen(
    inputs=input_values,
    outputs=output_values,
    config=codegen.CppConfig(),
    name="double_pendulum",
)
double_pendulum_values_data = double_pendulum_values.generate_function(
    namespace=namespace,
)

# Print what we generated. Note the nested structs that were automatically
# generated.
print("Files generated in {}:\n".format(double_pendulum_values_data.output_dir))
for f in double_pendulum_values_data.generated_files:
    print("  |- {}".format(os.path.relpath(f, double_pendulum_values_data.output_dir)))
```

[14]
```python
display_code_file(
    double_pendulum_values_data.function_dir / "double_pendulum.h",
    "C++",
)
```

```cpp
#include <lcmtypes/double_pendulum/inputs_t.hpp>
#include <lcmtypes/double_pendulum/outputs_t.hpp>

namespace double_pendulum {

/**
 * This function was autogenerated. Do not modify by hand.
 *
 * Args:
 *     inputs: Values
 *
 * Outputs:
 *     outputs: Values
 */
template <typename Scalar>
void DoublePendulum(const double_pendulum::inputs_t& inputs,
                    double_pendulum::outputs_t* const outputs = nullptr) {
  // Total ops: 50

  // Input arrays

  // Intermediate terms (8)
  const Scalar _tmp0 = 2 * inputs.ang.data()[1];
  const Scalar _tmp1 = 2 * inputs.params.m.data()[0] + inputs.params.m.data()[1];
  const Scalar _tmp2 = Scalar(1.0) / (_tmp1 - inputs.params.m.data()[1] *
                                                std::cos(_tmp0 - 2 * inputs.ang.data()[0]));
  const Scalar _tmp3 = -inputs.ang.data()[0];
  const Scalar _tmp4 = _tmp3 + inputs.ang.data()[1];
  const Scalar _tmp5 = std::cos(_tmp4);
  const Scalar _tmp6 = 2 * std::sin(_tmp4);
  const Scalar _tmp7 = inputs.params.m.data()[0] + inputs.params.m.data()[1];

  // Output terms (1)
  if (outputs != nullptr) {
    double_pendulum::outputs_t& _outputs = (*outputs);

    _outputs.ddang.data()[0] =
        _tmp2 *
        (-_tmp1 * inputs.constants.g * std::sin(inputs.ang.data()[0]) +
         _tmp6 * inputs.params.m.data()[1] *
             (2 * _tmp5 * inputs.dang.data()[0] * inputs.params.L.data()[0] +
              2 * inputs.dang.data()[1] * inputs.params.L.data()[1]) +
         inputs.constants.g * inputs.params.m.data()[1] * std::sin(_tmp0 + _tmp3)) /
        inputs.params.L.data()[0];
```

```
    _outputs.ddang.data()[1] =
        -_tmp2 * _tmp6 *
        (_tmp5 * std::pow(inputs.dang.data()[1], Scalar(2)) * inputs.params.L.data()[1] *
             inputs.params.m.data()[1] +
         _tmp7 * inputs.constants.g * std::cos(inputs.ang.data()[0]) +
         _tmp7 * std::pow(inputs.dang.data()[0], Scalar(2)) * inputs.params.L.data()[0]) /
        inputs.params.L.data()[1];
  }
}  // NOLINT(readability/fn_size)

// NOLINTNEXTLINE(readability/fn_size)
}  // namespace double_pendulum
```

[15]

```
# can generate the same function in other languages as well:
namespace = "double_pendulum"
double_pendulum_python = codegen.Codegen(
    inputs=inputs,
    outputs=outputs,
    config=codegen.PythonConfig(use_eigen_types=False),
    name="double_pendulum",
    return_key="ddang",
)
double_pendulum_python_data = double_pendulum_python.generate_function(
    namespace=namespace,
)

print("Files generated in {}:\n".format(double_pendulum_python_data.output_dir))
for f in double_pendulum_python_data.generated_files:
    print("  |- {}".format(os.path.relpath(f, double_pendulum_python_data.output_dir)))

display_code_file(
    double_pendulum_python_data.function_dir / "double_pendulum.py",
    "python",
)
```

```
        params: values

    Outputs:
        ddang: Matrix21
    """

    # Total ops: 50

    # Input arrays
    if ang.shape == (2,):
        ang = ang.reshape((1, 2))
```

```python
    elif ang.shape != (1, 2):
        raise IndexError(
            "ang is expected to have shape (1, 2) or (2,); instead had shape {}".format(ang.shape)
        )

    if dang.shape == (2,):
        dang = dang.reshape((1, 2))
    elif dang.shape != (1, 2):
        raise IndexError(
            "dang is expected to have shape (1, 2) or (2,); instead had shape {}".format(dang.shape)
        )

    # Intermediate terms (8)
    _tmp0 = 2 * ang[0, 1]
    _tmp1 = 2 * params.m[0] + params.m[1]
    _tmp2 = 1 / (_tmp1 - params.m[1] * math.cos(_tmp0 - 2 * ang[0, 0]))
    _tmp3 = -ang[0, 0]
    _tmp4 = _tmp3 + ang[0, 1]
    _tmp5 = math.cos(_tmp4)
    _tmp6 = 2 * math.sin(_tmp4)
    _tmp7 = params.m[0] + params.m[1]

    # Output terms
    _ddang = numpy.zeros((2, 1))
    _ddang[0, 0] = (
        _tmp2
        * (
            -_tmp1 * constants.g * math.sin(ang[0, 0])
            + _tmp6
            * params.m[1]
            * (2 * _tmp5 * dang[0, 0] * params.L[0] + 2 * dang[0, 1] * params.L[1])
            + constants.g * params.m[1] * math.sin(_tmp0 + _tmp3)
        )
        / params.L[0]
    )
    _ddang[1, 0] = (
        -_tmp2
        * _tmp6
        * (
            _tmp5 * dang[0, 1] ** 2 * params.L[1] * params.m[1]
            + _tmp7 * constants.g * math.cos(ang[0, 0])
            + _tmp7 * dang[0, 0] ** 2 * params.L[0]
        )
        / params.L[1]
    )
    return _ddang
```

```python
[16] constants_t = codegen_util.load_generated_lcmtype(
         namespace, "constants_t", double_pendulum_python_data.python_types_dir
     )

     params_t = codegen_util.load_generated_lcmtype(
         namespace, "params_t", double_pendulum_python_data.python_types_dir
     )

     ang = np.array([[0.0, 0.5]])
     dang = np.array([[0.0, 0.0]])
     consts = constants_t()
     consts.g = 9.81
     params = params_t()
     params.L = [0.5, 0.3]
     params.m = [0.3, 0.2]

     gen_module = codegen_util.load_generated_package(
         namespace, double_pendulum_python_data.function_dir
     )
     gen_module.double_pendulum(ang, dang, consts, params)
```

```
array([[  4.77199518],
       [-22.65691471]])
```