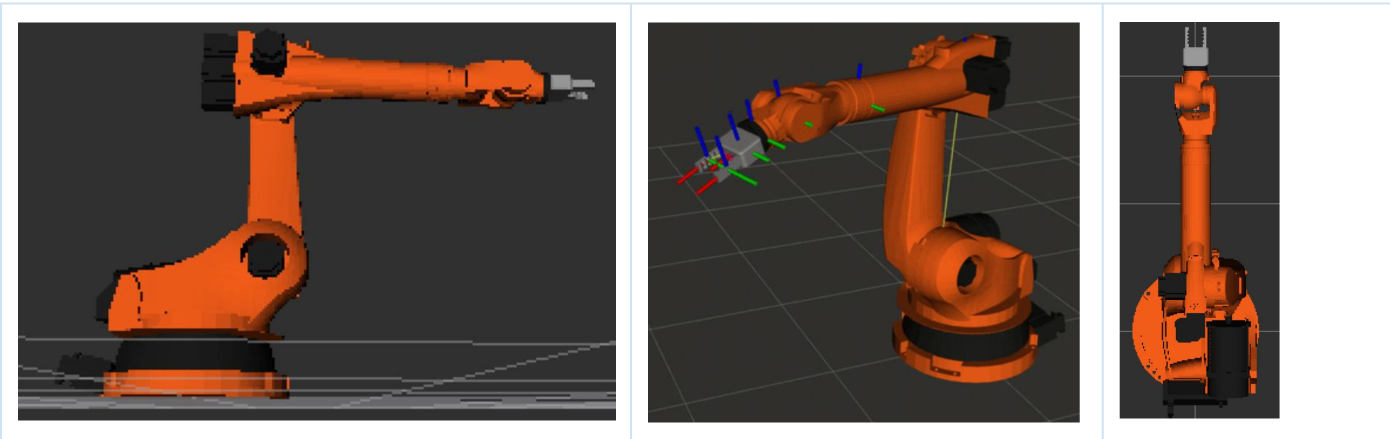
PROJECT 2: PICK AND PLACE, INVERSE KINEMATICS

Mithi Sevilla, July 3 2017, Udacity Robotics Nanodegree http://medium.com/@mithi http://github.com/mithi

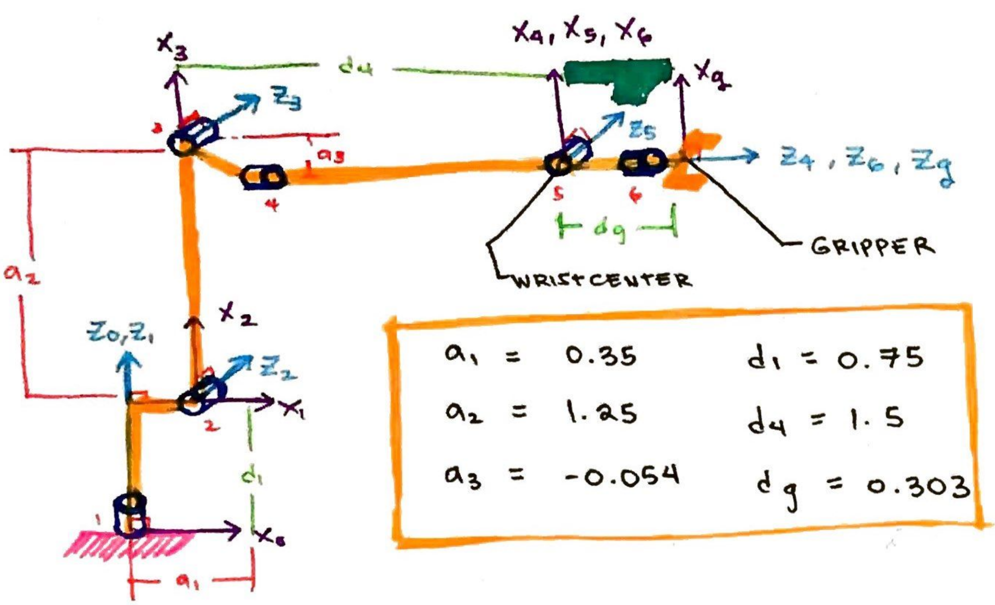
KUKA KR210: SIX DEGREE-OF-FREEDOM RRRRRR SERIAL MANIPULATOR



MODIFIED DH-PARAMETERS

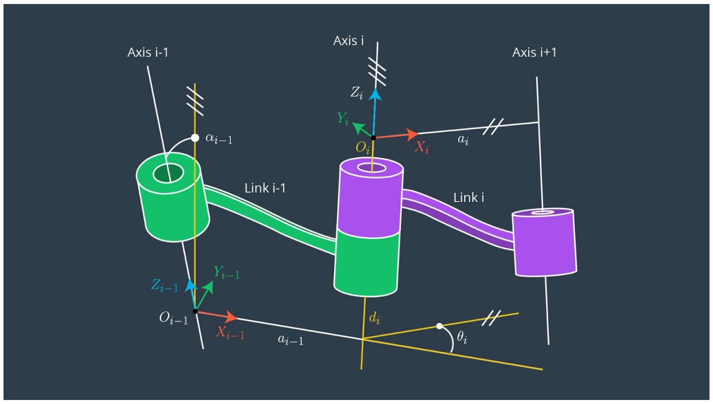
The Kuka KR210's modified DH parameters are the following:

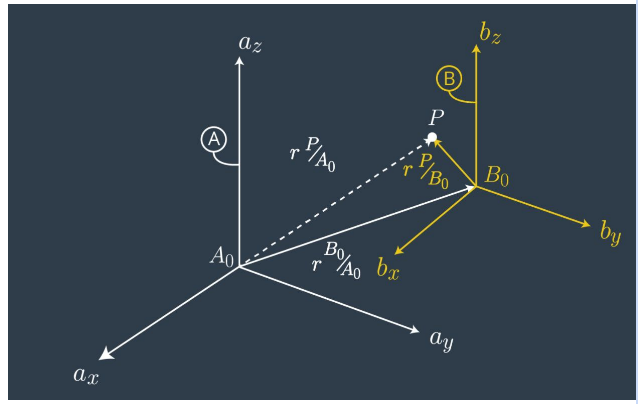
i	alpha[i-1]	a[i-1]	d[i]	theta[i]
1	0	0	d1 = 0.75	q1
2	-pi/2	a1 = 0.35	0	q2 - pi/2
3	0	a2 = 1.25	0	q3
4	-pi/2	a3 = -0.54	d4 = 1.50	q4
5	pi/2	0	0	q5
6	-pi/2	0	0	q6
g	0	0	dg = 0.303	0



DEFINITIONS:

`alpha[i-1]`	TWIST ANGLE	Angle between axis `z[i-1]` and axis `z[i]` measured about axis `x[i-1]`
`a[i-1]`	LINK LENGTH	Distance from axis `z[i-1]` to axis `z[i]` measured along axis `x[i-1]`
`d[i]`	LINK OFFSET	Distance from axis `x[i-1]` to axis `x[i]` measured along axis `z[i]`
`theta[i]`	JOINT ANGLE	Angle between axis `x[i-1]` and axis `x[i]` measured about axis `z[i]`





Each joint must rotate about the z-axis and each frame's z-axis must intersect and be perpendicular to the previous frame's z-axis

TRANSFORMATION MATRICES ABOUT EACH JOINT

DEFINITION:

Tab	**`T[a, b]`**	Pose *(homogeneous transformation matrix)* of coordinate-frame of joint b with respect to coordinate- frame of joint a which represents both rotation and translation

Tab

lx	mx	nx	px
ly	my	ny	py
lz	mz	nz	pz
0	0	0	1

Rab	**pab**
0 0 0	1

Note in the diagram above: ***pab = r Bo/Ao***

`Tac = Tab * Tbc`

`Rab = [l, m, n]`	The rotation pose of frame b with respect to frame a
`Pab = [px, py, pz].T`	The origin of frame b relative to origin of frame a expressed in coordinates of frame a

`l = [lx, ly, lz].T`	The x axis of frame b expressed in frame a coordinates
`m = [mx, my, mz].T`	The y axis of frame b expressed in frame a coordinates
`n = [nx, ny, nz].T`	The z axis of frame b expressed in frame a coordinates

`Pa = Tab * Pb`	**`Pb = r P/Bo is the point P expressed in the coordinates of frame b`** **`Pa = r P/Ao is the point P expressed in the coordinates of frame a`**

Given the modified DH parameters the pose of a joint frame `i` with respect to the previous joint frame `i-1` can be constructed as a sequence of four basic transformations:

T[i-1, i]	**R(x[i-1], alpha[i-1]) * D(x[i-1], a[i-1]) * R(z[i], theta[i]) * D(z[i], d[i])**

1. First, a rotation about `x[i-1]` by `alpha[i-1]`
2. Then, a translation along `x[i-1]` by `a[i-1]`
3. Then, a rotation about resulting axis `z[i]` by `theta[i]`
4. Then, a translation along axis `z[i]` by `d[i]`

Which results to the following matrix:

cos(theta[i])	-sin(theta[i])	0	a[i-1]
sin(theta[i])*cos(alpha[i-1])	cos(theta[i])*cos(alpha[i-1])	-sin(alpha[i-1])	-d[i]*sin(alpha[i-1])
sin(theta[i])*sin(alpha[i-1])	cos(theta[i])*sin(alpha[i-1])	cos(alpha[i-1])	d[i]*cos(alpha[i-1])
0	0	0	1

Substituting this matrix to the modified DH Parameters from the table above, we get the following transformation matrices about each joint with respect to the previous joint:

T01	**T12**
$\begin{bmatrix} \cos(q1) & -\sin(q1) & 0 & 0 \\ \sin(q1) & \cos(q1) & 0 & 0 \\ 0 & 0 & 1 & 0.75 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} \sin(q2) & \cos(q2) & 0 & 0.35 \\ 0 & 0 & 1 & 0 \\ \cos(q2) & -\sin(q2) & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

T23	**T34**
$\begin{bmatrix} \cos(q3) & -\sin(q3) & 0.0 & 1.25 \\ \sin(q3) & \cos(q3) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} \cos(q4) & -\sin(q4) & 0 & -0.054 \\ 0 & 0 & 1 & 1.5 \\ -\sin(q4) & -\cos(q4) & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

T45	**T56**
$\begin{bmatrix} \cos(q5) & -\sin(q5) & 0 & 0 \\ 0 & 0 & -1 & 0 \\ \sin(q5) & \cos(q5) & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} \cos(q6) & -\sin(q6) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -\sin(q6) & -\cos(q6) & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

T6g	
$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0.303 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	

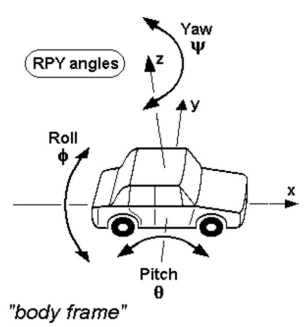
Note that I used the following code to get these matrices:

```
# get the pose (homogenous transforms) of each joint wrt to previous joint
q1, q2, q3, q4, q5, q6= symbols('q1:7')
d90 = pi / 2

T01 = pose(q1, 0, 0, 0.75)
T12 = pose(q2 - d90, -d90, 0.35, 0)
T23 = pose(q3, 0, 1.25, 0)
T34 = pose(q4, -d90, -0.054, 1.5)
T45 = pose(q5, d90, 0, 0)
```

```
T56 = pose(q6, -d90, 0, 0)
T6g = pose(0, 0, 0, 0.303)
```

```
def pose(theta, alpha, a, d):
  # This function returns the pose T of one joint frame i with respect to the previous joint frame (i - 1)
  # given the parameters:
  # theta: theta[i]
  # alpha: alpha[i-1]
  # a: a[i-1]
  # d: d[i]

  r11, r12 = cos(theta), -sin(theta)
  r23, r33 = -sin(alpha), cos(alpha)
  r21 = sin(theta) * cos(alpha)
  r22 = cos(theta) * cos(alpha)
  r31 = sin(theta) * sin(alpha)
  r32 = cos(theta) * sin(alpha)

  x = a
  y = -d * sin(alpha)
  z = d * cos(alpha)

  T = Matrix([
    [r11, r12, 0.0, x],
    [r21, r22, r23, y],
    [r31, r32, r33, z],
    [0.0, 0.0, 0.0, 1]])

  return simplify(T)
```

THE TRANSFORMATION MATRIX BETWEEN THE GRIPPER FRAME AND THE BASE FRAME GIVEN THE POSITION AND ORIENTATION OF THE GRIPPER



From Wikipedia:

> Any orientation can be achieved by composing three elemental rotations, starting from a known standard orientation. Equivalently, any rotation matrix R can be decomposed as a product of three elemental rotation matrices. For instance:
>
> $$R = X(\alpha)Y(\beta)Z(\gamma)$$
>
> is a rotation matrix that may be used to represent a composition of extrinsic rotations about axes *z,y,x*, (in that order) , or a composition of intrinsic rotations about axes *x-y′-z″*(in that order).

If we are given the position and orientation of the gripper `(px, py, pz, roll, pitch, yaw)`, assuming that this is with respect to the gripper frame from our DH parameter. The roll, pitch, yaw are extrinsic rotations, we can use consecutive x, y, z rotations such as tait-bryan angles. The resulting homogeneous transform is the following below:

THE POSE OF GRIPPER WRT TO THE BASE FRAME

```
Matrix([
[cos(pitch)*cos(yaw), sin(pitch)*sin(roll)*cos(yaw) - sin(yaw)*cos(roll), sin(pitch)*cos(roll)*cos(yaw) + sin(roll)*sin(yaw), px],
[sin(yaw)*cos(pitch), sin(pitch)*sin(roll)*sin(yaw) + cos(roll)*cos(yaw), sin(pitch)*sin(yaw)*cos(roll) - sin(roll)*cos(yaw), py],
[        -1.0*sin(pitch),                                   sin(roll)*cos(pitch),                                   cos(pitch)*cos(roll), pz],
[                    0,                                                    0,                                                     0,  1]])
```

Rotation matrices in x, y, z axes

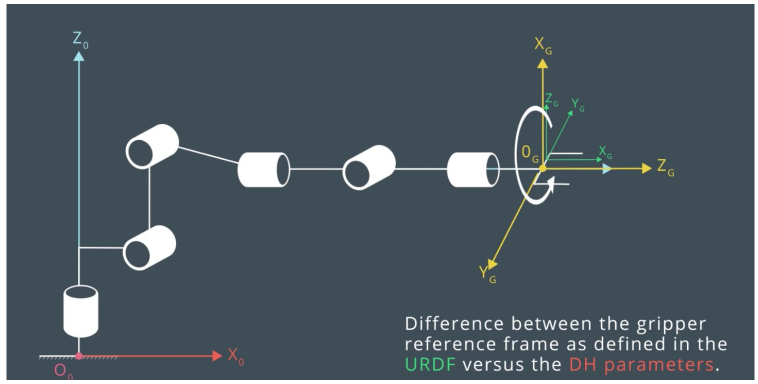
```
def rotx(q):

  sq, cq = sin(q), cos(q)

  r = Matrix([
    [1., 0., 0.],
    [0., cq,-sq],
    [0., sq, cq]])

  return r
```

```
def roty(q):

  sq, cq = sin(q), cos(q)

  r = Matrix([
    [ cq, 0., sq],
    [ 0., 1., 0.],
    [-sq, 0., cq]])

  return r
```

```
def rotz(q):

  sq, cq = sin(q), cos(q)

  r = Matrix([
    [cq,-sq, 0.],
    [sq, cq, 0.],
    [0., 0., 1.]])

  return r
```

Note that, I used the following code to get these matrices:

```
roll, pitch, yaw = symbols('roll pitch yaw')
px, py, pz = symbols('px py pz', real = True)

R = rotz(yaw) * roty(pitch) * rotx(roll)
T = Matrix([
  [R[0, 0], R[0, 1], R[0, 2], px],
  [R[1, 0], R[1, 1], R[1, 2], py],
  [R[2, 0], R[2, 1], R[2, 2], pz],
  [0, 0, 0, 1]
])

print(simplify(trigsimp(T)))
```



However, in our case the orientation is given in the URDF frame, so you have to perform a 180 degree counterclockwise rotation about the current z axis and then a 90 degree clockwise rotation about the resulting y axis. IE

```
R' = rotz(yaw) * roty(pitch) * rotx(roll) * rotz(pi) * roty(-pi/2)
```

So the resulting transform which is the pose of the gripper's URDF frame wrt to the base frame is as follows:

```
Matrix([
[sin(pitch)*cos(roll)*cos(yaw) + sin(roll)*sin(yaw), -sin(pitch)*sin(roll)*cos(yaw) + sin(yaw)*cos(roll), cos(pitch)*cos(yaw), px],
[sin(pitch)*sin(yaw)*cos(roll) - sin(roll)*cos(yaw), -sin(pitch)*sin(roll)*sin(yaw) - cos(roll)*cos(yaw), sin(yaw)*cos(pitch), py],
[                              cos(pitch)*cos(roll),                              -sin(roll)*cos(pitch),         -sin(pitch), pz],
[                                                 0,                                                   0,                   0,  1]])
```

CALCULATING THE INDIVIDUAL JOINT ANGLES

We are given the following: this is given position and orientation of the gripper wrt to URDFrame

```
px, py, pz = 0.49792, 1.3673, 2.4988
roll, pitch, yaw = 0.366, -0.078, 2.561
gripper_point = px, py, pz
```

ONE: from the poses, store the rotation of joint 3 wrt to the base frame and the transpose, store this, we will need this later.

```
T03 = simplify(T01 * T12 * T23)
R03 = T03[:3, :3]
R03T = R03.T
```

TWO: From the poses, also store the rotation of joint 6 wrt to the joint 3, store this, we will need this later.

```
T36 = simplify(T34 * T45 * T56)
R36 = T36[:3, :3]
```

THREE: The yaw, pitch, and roll is given wrt to the URDF frame. We must convert this to gripper frame by performing:

- a rotation of 180 degrees ccw about the z axis and then
- a rotation of 90 degrees cw about the new y axis

Note: This is the transpose of the rotation of the urdf frame wrt to gripper frame and its transpose which is strangely the same). Store these, we will need this later.

```
Rgu = (rotz(pi) * roty(-pi/2)).T
RguT = Rgu.T
```

FOUR: Get the rotation of the gripper in URDF wrt to base frame. Also get the rotation of the gripper wrt to the base frame from this. Store these, we will need this later.

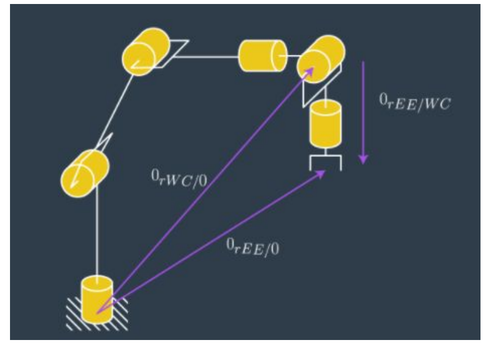
```
R0u_eval = rotz(yaw) * roty(pitch) * rotx(roll)
R0g_eval = R0u_eval * RguT # R0u = R0g * Rgu
```

FIVE: Get the position of the wrist center with respect to the base frame. The following function gets the coordinates of the wrist center wrt to the base frame **(xw, yw, zw)**, given the following info:

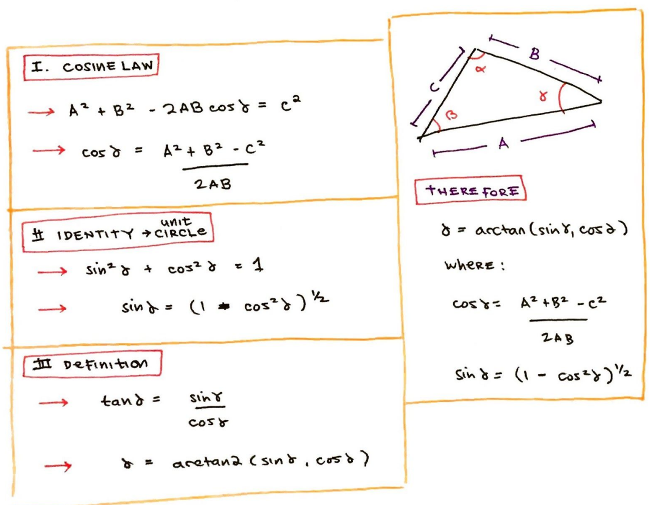
- the coordinates of the gripper (end effector) **(x, y, z)**

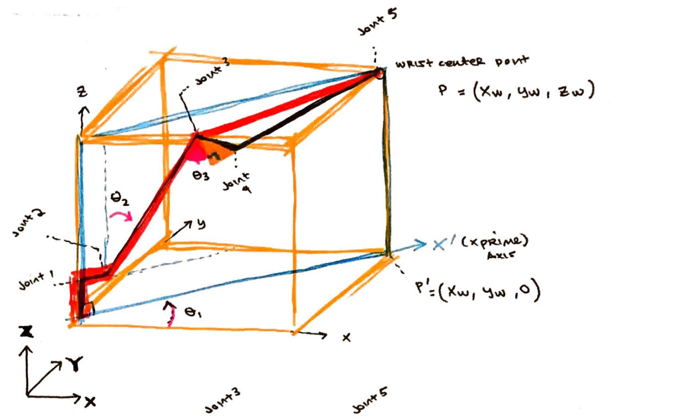
- the evaluated rotation of the gripper in the gripper frame wrt to the base frame **(R0u)**
- the distance between gripper and wrist center **dg** which is along a common **z** axis

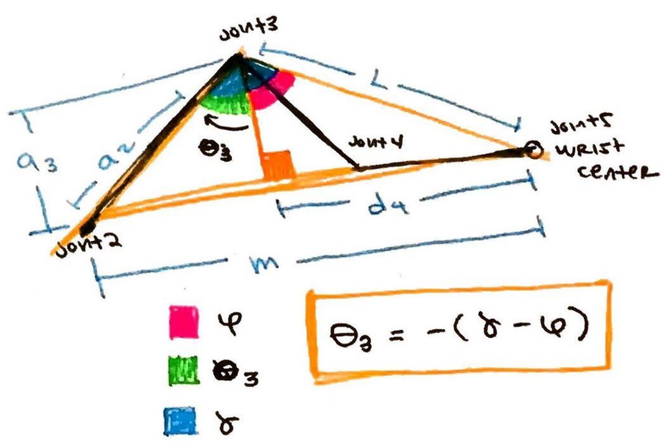
```
def get_wrist_center(gripper_point, R0g, dg = 0.303):

  xg, yg, zg = gripper_point

  nx, ny, nz = R0g[0, 2], R0g[1, 2], R0g[2, 2]
  xw = xg - dg * nx
  yw = yg - dg * ny
  zw = zg - dg * nz

  return xw, yw, zw

#########
wrist_center = get_wrist_center(gripper_point, R0g_eval, dg = 0.303)
```

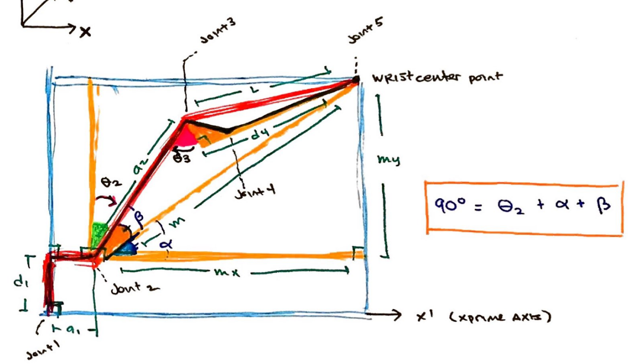


SIX: Now we have the **wrist_center**, we can get the first three joint angles given the geometry of the kuka arm and the cosine rule.










```
def get_hypotenuse(a, b):
  # calculate the longest side given the two shorter sides
  # of a right triangle using pythagorean theorem
  return sqrt(a*a + b*b)

def get_cosine_law_angle(a, b, c):
  # given all sides of a triangle a, b, c
  # calculate angle gamma between sides a and b
  cos_gamma = (a*a + b*b - c*c) / (2*a*b)
  sin_gamma = sqrt(1 - cos_gamma * cos_gamma)
  gamma = atan2(sin_gamma, cos_gamma)
  return gamma
```

```
def get_first_three_angles(wrist_center):

  x, y, z  = wrist_center

  a1, a2, a3 = 0.35, 1.25, -0.054
  d1, d4 = 0.75, 1.5
  l = 1.50097168527591 #get_hypotenuse(d4, -a3)
  phi = 1.53481186671284 # atan2(d4, -a3)

  x_prime = get_hypotenuse(x, y)
```

```
  mx = x_prime -  a1
  my = z - d1
  m = get_hypotenuse(mx, my)
  alpha = atan2(my, mx)

  gamma = get_cosine_law_angle(l, a2, m)
  beta = get_cosine_law_angle(m, a2, l)

  q1 = atan2(y, x)
  q2 = pi/2 - beta - alpha
  q3 = -(gamma - phi)

  return q1, q2, q3

#########
j1, j2, j3 = get_first_three_angles(wrist_center)
```

SEVEN: Finally, We can get the last three angles. You can follow my reasoning on the left below. On the right below is the code.

- **`R0g = R03 * R36 * R6g`**
- **`R6g = I`** frame of joint 6 is the same orientation of gripper frame
- **`R03.T * R0g = R03.T * R03 * R36 * I`**

So therefore:

---> **`R36 = R03.T * R0g`**

Recall we have this expression earlier for **`R03T`**:

```
Matrix([
  [sin(q2 + q3)*cos(q1), sin(q1)*sin(q2 + q3),  cos(q2 + q3)],
  [cos(q1)*cos(q2 + q3), sin(q1)*cos(q2 + q3), -sin(q2 + q3)],
  [            -sin(q1),              cos(q1),             0]])
```

Recall we also have evaluated **`R0g`** earlier.

```
Matrix([
    [0.257143295038827, 0.488872082559650, -0.833595473062543],
    [0.259329420712765, 0.796053601157403, 0.546851822377060],
    [0.930927267496960, -0.356795110642117, 0.0779209320563015]])
```

We also have solved for q1, q2, q3 earlier:

```
q1: 1.01249809363771
q2: -0.275800363737724
q3: -0.115686651053748
```

So we can actually evaluate for **`R36`** because we have numerical values for **`R03.T`** and **`R0g`**

```
def get_last_three_angles(R):

  sin_q4 = R[2, 2]
  cos_q4 =  -R[0, 2]

  sin_q5 = sqrt(R[0, 2]**2 + R[2, 2]**2)
  cos_q5 = R[1, 2]

  sin_q6 = -R[1, 1]
  cos_q6 = R[1, 0]

  q4 = atan2(sin_q4, cos_q4)
  q5 = atan2(sin_q5, cos_q5)
  q6 = atan2(sin_q6, cos_q6)

  return q4, q5, q6

#####
R03T_eval = R03T.evalf(
  subs = {
    q1: j1.evalf(),
    q2: j2.evalf(),
    q3: j3.evalf()}
)

R36_eval = R03T_eval * R0g_eval

j4, j5, j6 = get_last_three_angles(R36_eval)
```

CONCLUSION:

From the gripper position and orientation in the URDF frame:

```
px, py, pz = 0.49792, 1.3673, 2.4988
roll, pitch, yaw = 0.366, -0.078, 2.561
```

We can get the joint angles:

```
q1: 1.01249809363771
q2: -0.275800363737724
q3: -0.115686651053748
q4: 1.63446527240323
q5: 1.52050002599430
q6: -0.815781306199682
```

REFERENCES:

Udacity Course Notes, Various Youtube videos, Students on Udacity-robotics Slack Channel especially @gwwang for posting lots of diagrams.