

# Project: Kinematics Pick & Place

## Rubric Points

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

### Kinematic Analysis

#### 1. Run the forward\_kinematics demo and evaluate the kr210.urdf.xacro file to perform kinematic analysis of Kuka KR210 robot and derive its DH parameters.

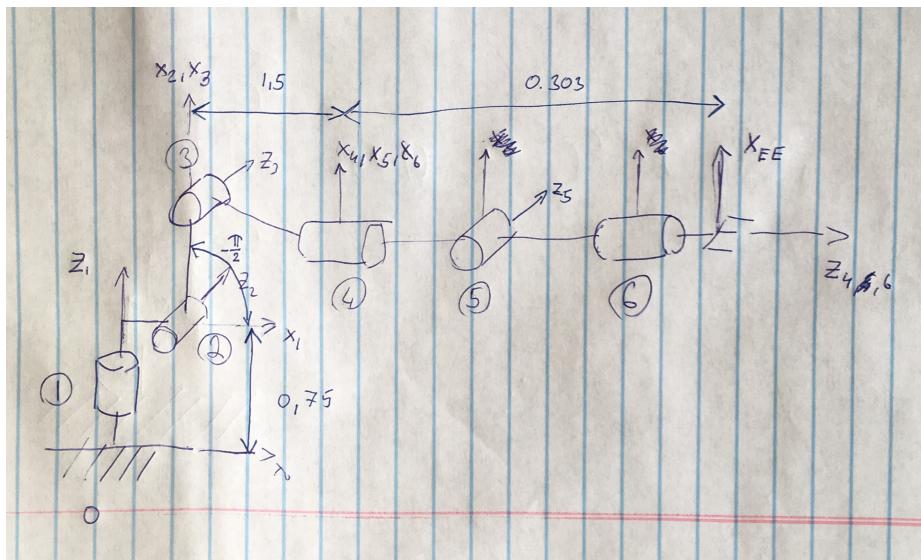
During this part we derived description of the robot (kinematic chain). The reason we us DH is that it is a system that ensures that if people use it the same way they can easily exchange information about robots etc. Also it simplifies the implementation since the transformation from DH table to transformation matrices is very simple.

Here is a DH parameters table.

Links	alpha(i-1)	a(i-1)	d(i-1)	theta(i)
0->1	0	0	0.75	q1
1->2	- pi/2	0.35	0	-pi/2 + q2
2->3	0	1.25	0	q3
3->4	-pi/2	-0.054	1.5	q4
4->5	pi/2	0	0	q5
5->6	-pi/2	0	0	q6
6->EE	0	0	0.303	0

Let's walk through derivation briefly

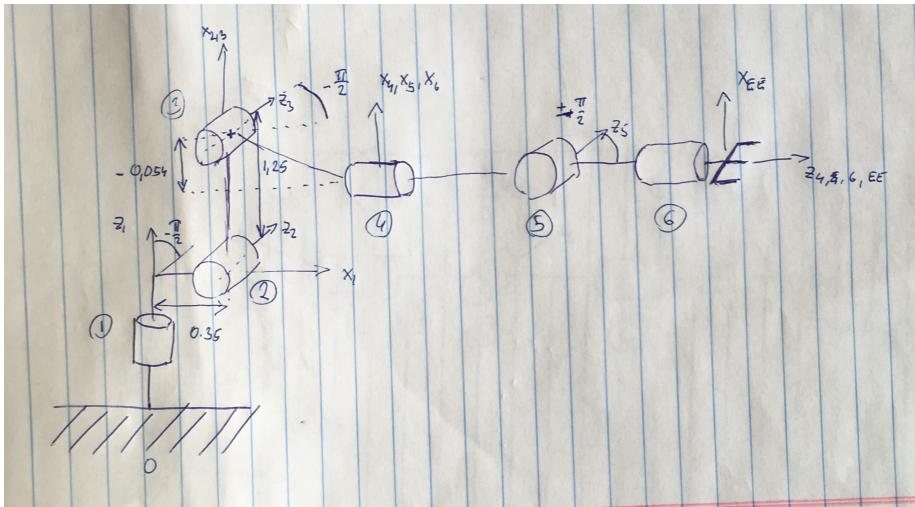
Here is pictures of the kinematic joints. Important is to pick z and x axes for the joints.



In this first picture I highlighted d and theta parameters.

$d$  is defined as distance between neighbouring  $x$  axes ( $x_i$  and  $x_{i+1}$ ) along the  $z_{i+1}$  axes. In the picture are 3 instances. The values can be found in the xacro files. Note that although joints 4,5,6 are drawn far apart their  $x$  axes are actually coincident. This simplifies some values + it actually fits the physical implementation of the wrist.

$\theta$  are defined as angle between  $x_i$  and  $x_{i+1}$  along  $z_{i+1}$  in the right hand sense. Since all joints are revolute this parameters are all variables but we need to augment one angle by a value  $-\pi/2$  for it to work (between  $x_2, x_3$ ).



In this picture I did the same for the parameters alpha and  $a$ .

$a$  is defined as distance between  $z_i$  and  $z_{i+1}$  along  $x_i$ . There are three nonzero places marked in the picture. Note maybe the -0.054. It is zero since it goes down from 3 to 4.

The alphas are defined as angle between  $z_i$  and  $z_{i+1}$  about  $x_i$  again in the right hand sense. There is a lot of these values in the picture since we have adjacent joints usually rotated by 90 degrees. I marked several values in the picture. Notice  $z_4-z_5$  and  $z_5-z_6$ . They are marked at the same spot but it has different signs.

What is interesting about this particular setup is that there is many parallel/collinear links which significantly simplifies derivation of parameters. Also the wrist center (4-6) is in one place again causing a lot of parameters to be zero.

Derivation of DH parameters is not difficult but definitely requires practice to be able to perform it without errors. One place where I struggled in this part was the selection of axis between points 1-2 which leads to additional  $\pi/2$ . Only later I realized that I did not follow the DH rules properly and this orientation of this axis is actually forced by definition based on selection of previous axes.

After acquainting ourself with the environment I derived the DH parameters and implemented forward and backward kinematics using the IK. Here is the output of the debug script which was essential to make it work.

```

Theta 6 error is: 3.1441136
rThese theta errors may not be a correct representation of your code, due to the fact
that the arm can have multiple positions. It is best to add your forward kinematics to
confirm whether your code is working or not.

End effector error for x position is: 0.00000303
End effector error for y position is: 0.00000033
End effector error for z position is: 0.00000260
Overall end effector offset is: 0.00000000 units

robot@daclty:~/catkin_ws/src/RoboND-Kinematics-Project$ ./IK_debug3.py
angle_a: 0.6774383653549
angle_r: 0.0000000000000
angle_c: 0.55700232089768
l1:12266057632267 + pi/2
l2: 93284637944608 + pi/2
l3: 78802272347388
l4: 4074823592328954
l5: 0.0000000000000
l6: 0.0000000000000
l7: 0.0000000000000
l8: 0.0000000000000
l9: 0.0000000000000
l10: 0.0000000000000
l11: 0.0000000000000
l12: 0.0000000000000
l13: 0.0000000000000
l14: 0.0000000000000
l15: 0.0000000000000
l16: 0.0000000000000
l17: 0.0000000000000
l18: 0.0000000000000
l19: 0.0000000000000
l20: 0.0000000000000
l21: 0.0000000000000
l22: 0.0000000000000
l23: 0.0000000000000
l24: 0.0000000000000
l25: 0.0000000000000
l26: 0.0000000000000
l27: 0.0000000000000
l28: 0.0000000000000
l29: 0.0000000000000
l30: 0.0000000000000
l31: 0.0000000000000
l32: 0.0000000000000
l33: 0.0000000000000
l34: 0.0000000000000
l35: 0.0000000000000
l36: 0.0000000000000
l37: 0.0000000000000
l38: 0.0000000000000
l39: 0.0000000000000
l40: 0.0000000000000
l41: 0.0000000000000
l42: 0.0000000000000
l43: 0.0000000000000
l44: 0.0000000000000
l45: 0.0000000000000
l46: 0.0000000000000
l47: 0.0000000000000
l48: 0.0000000000000
l49: 0.0000000000000
l50: 0.0000000000000
l51: 0.0000000000000
l52: 0.0000000000000
l53: 0.0000000000000
l54: 0.0000000000000
l55: 0.0000000000000
l56: 0.0000000000000
l57: 0.0000000000000
l58: 0.0000000000000
l59: 0.0000000000000
l60: 0.0000000000000
l61: 0.0000000000000
l62: 0.0000000000000
l63: 0.0000000000000
l64: 0.0000000000000
l65: 0.0000000000000
l66: 0.0000000000000
l67: 0.0000000000000
l68: 0.0000000000000
l69: 0.0000000000000
l70: 0.0000000000000
l71: 0.0000000000000
l72: 0.0000000000000
l73: 0.0000000000000
l74: 0.0000000000000
l75: 0.0000000000000
l76: 0.0000000000000
l77: 0.0000000000000
l78: 0.0000000000000
l79: 0.0000000000000
l80: 0.0000000000000
l81: 0.0000000000000
l82: 0.0000000000000
l83: 0.0000000000000
l84: 0.0000000000000
l85: 0.0000000000000
l86: 0.0000000000000
l87: 0.0000000000000
l88: 0.0000000000000
l89: 0.0000000000000
l90: 0.0000000000000
l91: 0.0000000000000
l92: 0.0000000000000
l93: 0.0000000000000
l94: 0.0000000000000
l95: 0.0000000000000
l96: 0.0000000000000
l97: 0.0000000000000
l98: 0.0000000000000
l99: 0.0000000000000
l100: 0.0000000000000
l101: 0.0000000000000
l102: 0.0000000000000
l103: 0.0000000000000
l104: 0.0000000000000
l105: 0.0000000000000
l106: 0.0000000000000
l107: 0.0000000000000
l108: 0.0000000000000
l109: 0.0000000000000
l110: 0.0000000000000
l111: 0.0000000000000
l112: 0.0000000000000
l113: 0.0000000000000
l114: 0.0000000000000
l115: 0.0000000000000
l116: 0.0000000000000
l117: 0.0000000000000
l118: 0.0000000000000
l119: 0.0000000000000
l120: 0.0000000000000
l121: 0.0000000000000
l122: 0.0000000000000
l123: 0.0000000000000
l124: 0.0000000000000
l125: 0.0000000000000
l126: 0.0000000000000
l127: 0.0000000000000
l128: 0.0000000000000
l129: 0.0000000000000
l130: 0.0000000000000
l131: 0.0000000000000
l132: 0.0000000000000
l133: 0.0000000000000
l134: 0.0000000000000
l135: 0.0000000000000
l136: 0.0000000000000
l137: 0.0000000000000
l138: 0.0000000000000
l139: 0.0000000000000
l140: 0.0000000000000
l141: 0.0000000000000
l142: 0.0000000000000
l143: 0.0000000000000
l144: 0.0000000000000
l145: 0.0000000000000
l146: 0.0000000000000
l147: 0.0000000000000
l148: 0.0000000000000
l149: 0.0000000000000
l150: 0.0000000000000
l151: 0.0000000000000
l152: 0.0000000000000
l153: 0.0000000000000
l154: 0.0000000000000
l155: 0.0000000000000
l156: 0.0000000000000
l157: 0.0000000000000
l158: 0.0000000000000
l159: 0.0000000000000
l160: 0.0000000000000
l161: 0.0000000000000
l162: 0.0000000000000
l163: 0.0000000000000
l164: 0.0000000000000
l165: 0.0000000000000
l166: 0.0000000000000
l167: 0.0000000000000
l168: 0.0000000000000
l169: 0.0000000000000
l170: 0.0000000000000
l171: 0.0000000000000
l172: 0.0000000000000
l173: 0.0000000000000
l174: 0.0000000000000
l175: 0.0000000000000
l176: 0.0000000000000
l177: 0.0000000000000
l178: 0.0000000000000
l179: 0.0000000000000
l180: 0.0000000000000
l181: 0.0000000000000
l182: 0.0000000000000
l183: 0.0000000000000
l184: 0.0000000000000
l185: 0.0000000000000
l186: 0.0000000000000
l187: 0.0000000000000
l188: 0.0000000000000
l189: 0.0000000000000
l190: 0.0000000000000
l191: 0.0000000000000
l192: 0.0000000000000
l193: 0.0000000000000
l194: 0.0000000000000
l195: 0.0000000000000
l196: 0.0000000000000
l197: 0.0000000000000
l198: 0.0000000000000
l199: 0.0000000000000
l200: 0.0000000000000
l201: 0.0000000000000
l202: 0.0000000000000
l203: 0.0000000000000
l204: 0.0000000000000
l205: 0.0000000000000
l206: 0.0000000000000
l207: 0.0000000000000
l208: 0.0000000000000
l209: 0.0000000000000
l210: 0.0000000000000
l211: 0.0000000000000
l212: 0.0000000000000
l213: 0.0000000000000
l214: 0.0000000000000
l215: 0.0000000000000
l216: 0.0000000000000
l217: 0.0000000000000
l218: 0.0000000000000
l219: 0.0000000000000
l220: 0.0000000000000
l221: 0.0000000000000
l222: 0.0000000000000
l223: 0.0000000000000
l224: 0.0000000000000
l225: 0.0000000000000
l226: 0.0000000000000
l227: 0.0000000000000
l228: 0.0000000000000
l229: 0.0000000000000
l230: 0.0000000000000
l231: 0.0000000000000
l232: 0.0000000000000
l233: 0.0000000000000
l234: 0.0000000000000
l235: 0.0000000000000
l236: 0.0000000000000
l237: 0.0000000000000
l238: 0.0000000000000
l239: 0.0000000000000
l240: 0.0000000000000
l241: 0.0000000000000
l242: 0.0000000000000
l243: 0.0000000000000
l244: 0.0000000000000
l245: 0.0000000000000
l246: 0.0000000000000
l247: 0.0000000000000
l248: 0.0000000000000
l249: 0.0000000000000
l250: 0.0000000000000
l251: 0.0000000000000
l252: 0.0000000000000
l253: 0.0000000000000
l254: 0.0000000000000
l255: 0.0000000000000
l256: 0.0000000000000
l257: 0.0000000000000
l258: 0.0000000000000
l259: 0.0000000000000
l260: 0.0000000000000
l261: 0.0000000000000
l262: 0.0000000000000
l263: 0.0000000000000
l264: 0.0000000000000
l265: 0.0000000000000
l266: 0.0000000000000
l267: 0.0000000000000
l268: 0.0000000000000
l269: 0.0000000000000
l270: 0.0000000000000
l271: 0.0000000000000
l272: 0.0000000000000
l273: 0.0000000000000
l274: 0.0000000000000
l275: 0.0000000000000
l276: 0.0000000000000
l277: 0.0000000000000
l278: 0.0000000000000
l279: 0.0000000000000
l280: 0.0000000000000
l281: 0.0000000000000
l282: 0.0000000000000
l283: 0.0000000000000
l284: 0.0000000000000
l285: 0.0000000000000
l286: 0.0000000000000
l287: 0.0000000000000
l288: 0.0000000000000
l289: 0.0000000000000
l290: 0.0000000000000
l291: 0.0000000000000
l292: 0.0000000000000
l293: 0.0000000000000
l294: 0.0000000000000
l295: 0.0000000000000
l296: 0.0000000000000
l297: 0.0000000000000
l298: 0.0000000000000
l299: 0.0000000000000
l300: 0.0000000000000
l301: 0.0000000000000
l302: 0.0000000000000
l303: 0.0000000000000
l304: 0.0000000000000
l305: 0.0000000000000
l306: 0.0000000000000
l307: 0.0000000000000
l308: 0.0000000000000
l309: 0.0000000000000
l310: 0.0000000000000
l311: 0.0000000000000
l312: 0.0000000000000
l313: 0.0000000000000
l314: 0.0000000000000
l315: 0.0000000000000
l316: 0.0000000000000
l317: 0.0000000000000
l318: 0.0000000000000
l319: 0.0000000000000
l320: 0.0000000000000
l321: 0.0000000000000
l322: 0.0000000000000
l323: 0.0000000000000
l324: 0.0000000000000
l325: 0.0000000000000
l326: 0.0000000000000
l327: 0.0000000000000
l328: 0.0000000000000
l329: 0.0000000000000
l330: 0.0000000000000
l331: 0.0000000000000
l332: 0.0000000000000
l333: 0.0000000000000
l334: 0.0000000000000
l335: 0.0000000000000
l336: 0.0000000000000
l337: 0.0000000000000
l338: 0.0000000000000
l339: 0.0000000000000
l340: 0.0000000000000
l341: 0.0000000000000
l342: 0.0000000000000
l343: 0.0000000000000
l344: 0.0000000000000
l345: 0.0000000000000
l346: 0.0000000000000
l347: 0.0000000000000
l348: 0.0000000000000
l349: 0.0000000000000
l350: 0.0000000000000
l351: 0.0000000000000
l352: 0.0000000000000
l353: 0.0000000000000
l354: 0.0000000000000
l355: 0.0000000000000
l356: 0.0000000000000
l357: 0.0000000000000
l358: 0.0000000000000
l359: 0.0000000000000
l360: 0.0000000000000
l361: 0.0000000000000
l362: 0.0000000000000
l363: 0.0000000000000
l364: 0.0000000000000
l365: 0.0000000000000
l366: 0.0000000000000
l367: 0.0000000000000
l368: 0.0000000000000
l369: 0.0000000000000
l370: 0.0000000000000
l371: 0.0000000000000
l372: 0.0000000000000
l373: 0.0000000000000
l374: 0.0000000000000
l375: 0.0000000000000
l376: 0.0000000000000
l377: 0.0000000000000
l378: 0.0000000000000
l379: 0.0000000000000
l380: 0.0000000000000
l381: 0.0000000000000
l382: 0.0000000000000
l383: 0.0000000000000
l384: 0.0000000000000
l385: 0.0000000000000
l386: 0.0000000000000
l387: 0.0000000000000
l388: 0.0000000000000
l389: 0.0000000000000
l390: 0.0000000000000
l391: 0.0000000000000
l392: 0.0000000000000
l393: 0.0000000000000
l394: 0.0000000000000
l395: 0.0000000000000
l396: 0.0000000000000
l397: 0.0000000000000
l398: 0.0000000000000
l399: 0.0000000000000
l400: 0.0000000000000
l401: 0.0000000000000
l402: 0.0000000000000
l403: 0.0000000000000
l404: 0.0000000000000
l405: 0.0000000000000
l406: 0.0000000000000
l407: 0.0000000000000
l408: 0.0000000000000
l409: 0.0000000000000
l410: 0.0000000000000
l411: 0.0000000000000
l412: 0.0000000000000
l413: 0.0000000000000
l414: 0.0000000000000
l415: 0.0000000000000
l416: 0.0000000000000
l417: 0.0000000000000
l418: 0.0000000000000
l419: 0.0000000000000
l420: 0.0000000000000
l421: 0.0000000000000
l422: 0.0000000000000
l423: 0.0000000000000
l424: 0.0000000000000
l425: 0.0000000000000
l426: 0.0000000000000
l427: 0.0000000000000
l428: 0.0000000000000
l429: 0.0000000000000
l430: 0.0000000000000
l431: 0.0000000000000
l432: 0.0000000000000
l433: 0.0000000000000
l434: 0.0000000000000
l435: 0.0000000000000
l436: 0.0000000000000
l437: 0.0000000000000
l438: 0.0000000000000
l439: 0.0000000000000
l440: 0.0000000000000
l441: 0.0000000000000
l442: 0.0000000000000
l443: 0.0000000000000
l444: 0.0000000000000
l445: 0.0000000000000
l446: 0.0000000000000
l447: 0.0000000000000
l448: 0.0000000000000
l449: 0.0000000000000
l450: 0.0000000000000
l451: 0.0000000000000
l452: 0.0000000000000
l453: 0.0000000000000
l454: 0.0000000000000
l455: 0.0000000000000
l456: 0.0000000000000
l457: 0.0000000000000
l458: 0
```

**2. Using the DH parameter table you derived earlier, create individual transformation matrices about each joint. In addition, also generate a generalized homogeneous transform between base\_link and gripper\_link using only end-effector(gripper) pose.**

Benefit of DH parameters is that it simplifies generation of transformation matrices since they can be constructed automatically as seen in the code. By wrapping the matrix generation into a function we can easily write something like

```
def ht_matrix_from_dh(alpha, a, d, q):
    return Matrix([[cos(q), -sin(q), 0, a],
                  [sin(q) * cos(alpha), cos(q)*cos(alpha), -sin(alpha), -sin(alpha) * d],
                  [sin(q) * sin(alpha), cos(q)*sin(alpha), cos(alpha), cos(alpha) * d],
                  [0, 0, 0, 1]])
```

We then can generate individual transformations directly from DH table

```
T0_1 = ht_matrix_from_dh(alpha0, a0, d1, q1).subs(dh_parameters)
..
..
T6_EE
```

And then we can create transformation between arbitrary points.

```
T0_EE = T0_1 * T1_2 .... * T6_EE
```

**3. Decouple Inverse Kinematics problem into Inverse Position Kinematics and inverse Orientation Kinematics; doing so derive the equations to calculate all individual joint angles.**

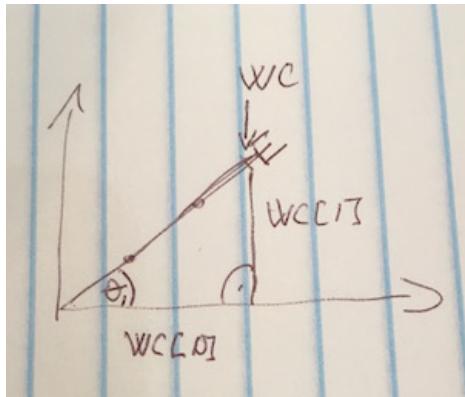
#### Outline of solution

The solution goes like this.

- From the planner we receive the poses of the end effector
- Our robot falls into a wide category of robots whose IK can be solved geometrically. This has an advantage that the solver can be very fast since it is in closed form. But we have to find specific solution for this particular kinematic chain
- We get position of the end effector
- From this we compute the position of wrist center
  - This is done by using inverse transformations from the forward kinematics part from wrist center to end effector
- If we know position of our wrist center we can solve theta 1-3 by applying some trigonometric functions
- Theta 4-6 are taken from relation of wrist center to end effector by extracting angles from the rotation matrix

#### Theta 1

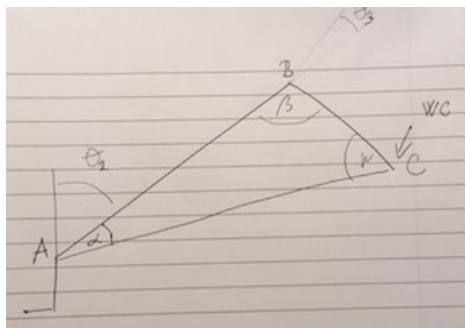
Seen from above we can simply calculate theta 1 by using atan2 directly from the known position of the wrist in x y plane.



```
theta_1 = atan2(wc[0], wc[1])
```

### Triangle ABC

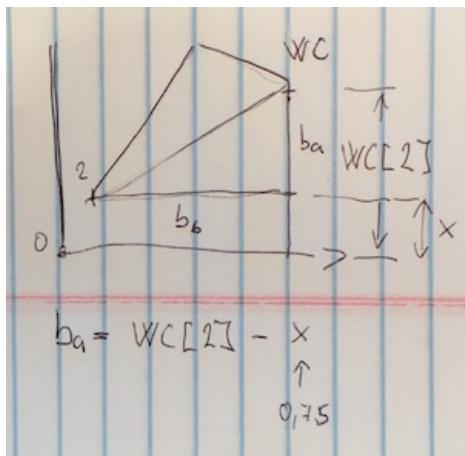
Both theta 2 and theta 3 can be calculated from a triangle A (joint 2), B(joint 3), C (wrist center).



First we calculate the sides of a triangle and then use cosine sentences to get the corresponding angles.

### Side b

First let's grab side b. It can be calculated by pythagorean theorem from sides b\_b and b\_a on the following picture.



**b\_a** we get almost directly from the position of wrist center. **b\_b** can be calculated by another application of pythagoreas from x and y position of wrist center. Both need to be accounted for the fact that they are measured from point 2 and not the origin.

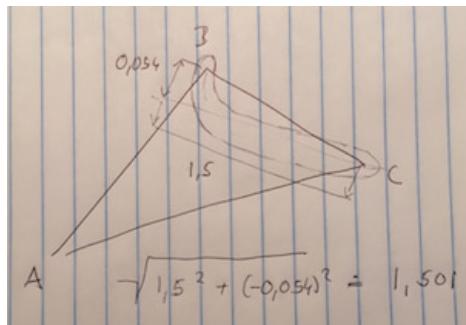
### Side c

Side C can be used directly from the DH params or xarco file.

### Side a

Just taking the side A from DH parameters directly yields decent results but we are introducing systematic error. I noticed after looking at the provided solution. It works roughly like this.

The error comes from the fact that the link represented by c is not a straight link but has a curve or "sag" to it. The corrected length of the side c is simply acquired by computing the hypotenuse of that link for the DH parameters (the sag in the following picture is really exaggerated).



The second correction comes to the angle. And is acquired by for example atan2 from the sides.

$$\text{theta\_2} = \text{atan2}(-0.054, 1.5) = -0.036$$

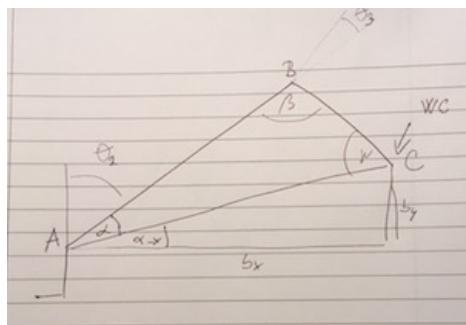
### Theta 2

When we have all the pieces we can simply get theta

$$\text{theta\_2} = \pi/2 - \text{angle\_a} - \text{angle\_a\_x}$$

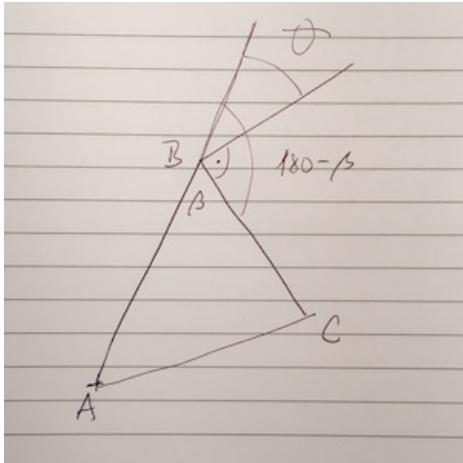
where angle\_a\_x is acquired by

$$\text{angle\_a\_x} = \text{atan2}(b_y, b_x)$$



### Theta 3

Theta 3 can be computed by realizing that Theta 3 and reflected beta has to equal  $\pi/2$



This can be expressed in equation as

$$\begin{aligned} 90 &= 180 - \beta - \theta \\ -90 &= -\beta - \theta \\ \theta &= 90 - \beta \end{aligned}$$

Here in the derivation I omit the small correction of the beta angle discussed before caused by the sag of the link.

#### Theta 4-6

From known rotation matrix we want to extract euler's angles. This is done by 2 tricks. Generally we want to use atans as much as possible due to unambiguity compared to arcsin.

The trigonometric tricks are illustrated in the following image.

$$\begin{bmatrix} c_1 c_2 & c_1 s_2 s_3 - c_3 s_1 & s_1 s_3 + c_1 c_3 s_2 \\ c_2 s_1 & c_1 c_3 + s_1 s_2 s_3 & c_3 s_1 s_2 + c_1 s_3 \\ -s_2 & c_2 s_3 & c_2 c_3 \end{bmatrix}$$

$$\begin{aligned} \tan \alpha &= \frac{\sin}{\cos} = \frac{s_2 s_3}{c_2 c_3} = \sqrt{c_2^2 s_3^2 + c_3^2 c_2^2} = \\ &= \sqrt{c_2^2 \cdot (s_3^2 + c_3^2)} = \\ &= c_2^2 \cdot 1 \end{aligned}$$

One is leveraging the identity

$$\tan(3) = \sin(3)/\cos(3) = (c_2 s_3)/(c_2 c_3)$$

The second one is using the idea

```

cos(x) = sqrt(cos^2(x)sin^2(y) + cos^2(x)cos^2(y))
        = sqrt(cos^2(x)) * (sin^2(y) + cos^2(y))

using substitution (sin^2(y) + cos^2(y)) = 1

        = sqrt(cos^2(x)) * 1
        = cos(x)

and similar to the above can write

tg(2) = sin(2)/ sqrt(cos^2(x)sin^2(y) + cos^2(x)cos^2(y))

```

Here the example uses Z Y X matrix. The real code actually uses R3\_EE. First we calculate

```
T3_EE = T3_4 * T4_5 * T5_6 * T6_EE
```

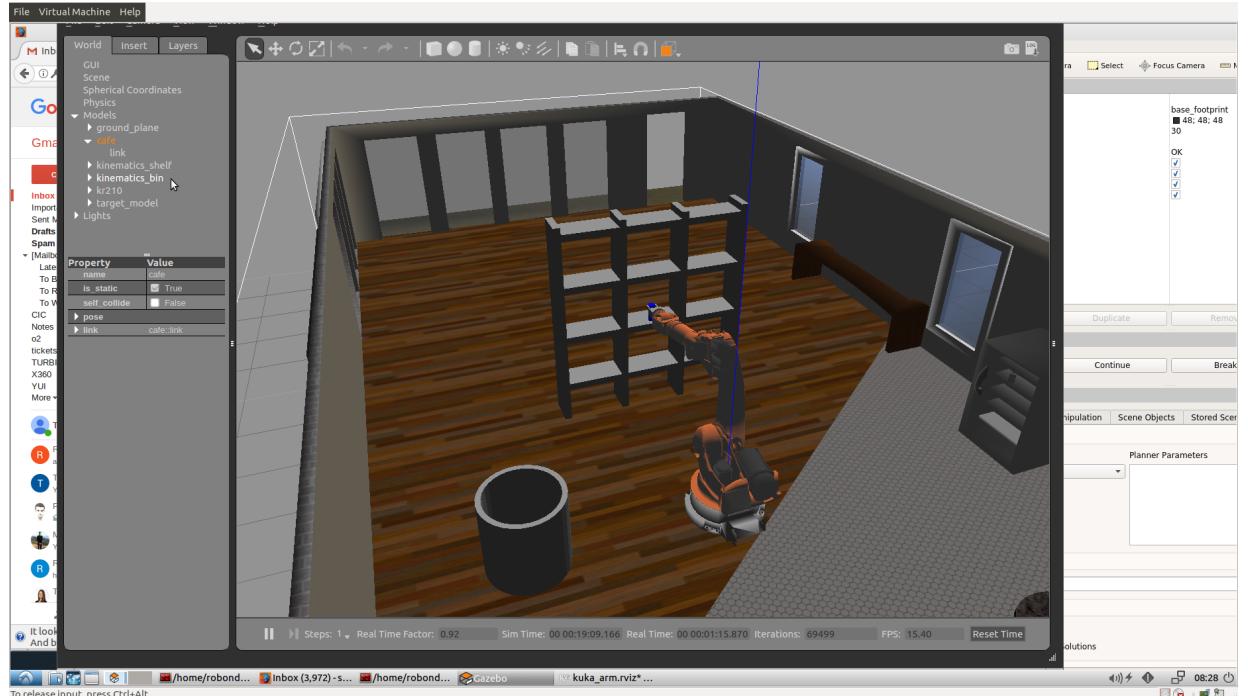
Then we turn into rotational by cropping and use sympy to simplify. Then we apply exactly the same approach as above.

## Project Implementation

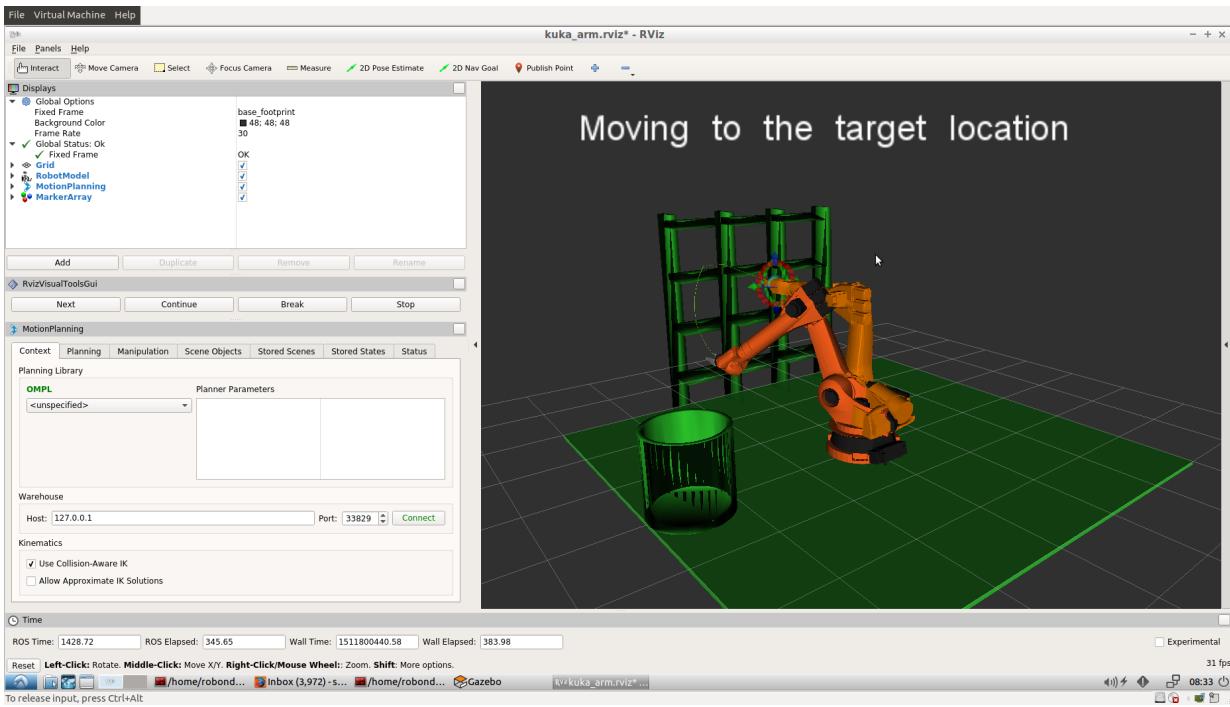
**1. Fill in the `IK_server.py` file with properly commented python code for calculating Inverse Kinematics based on previously performed Kinematic Analysis. Your code must guide the robot to successfully complete 8/10 pick and place cycles. Briefly discuss the code you implemented and your results.**

The code seem to perform well after several iterations. Here are a couple of picture to illustrate it in action.

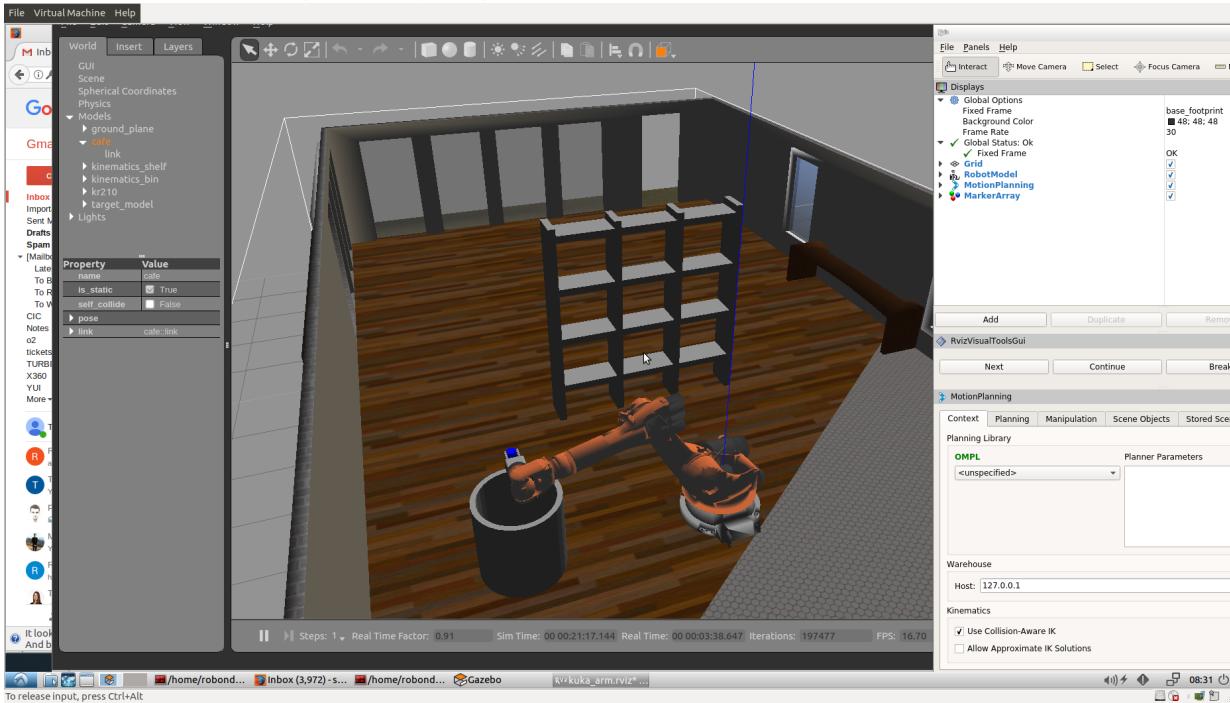
Here the robots is just about to pick up an object.

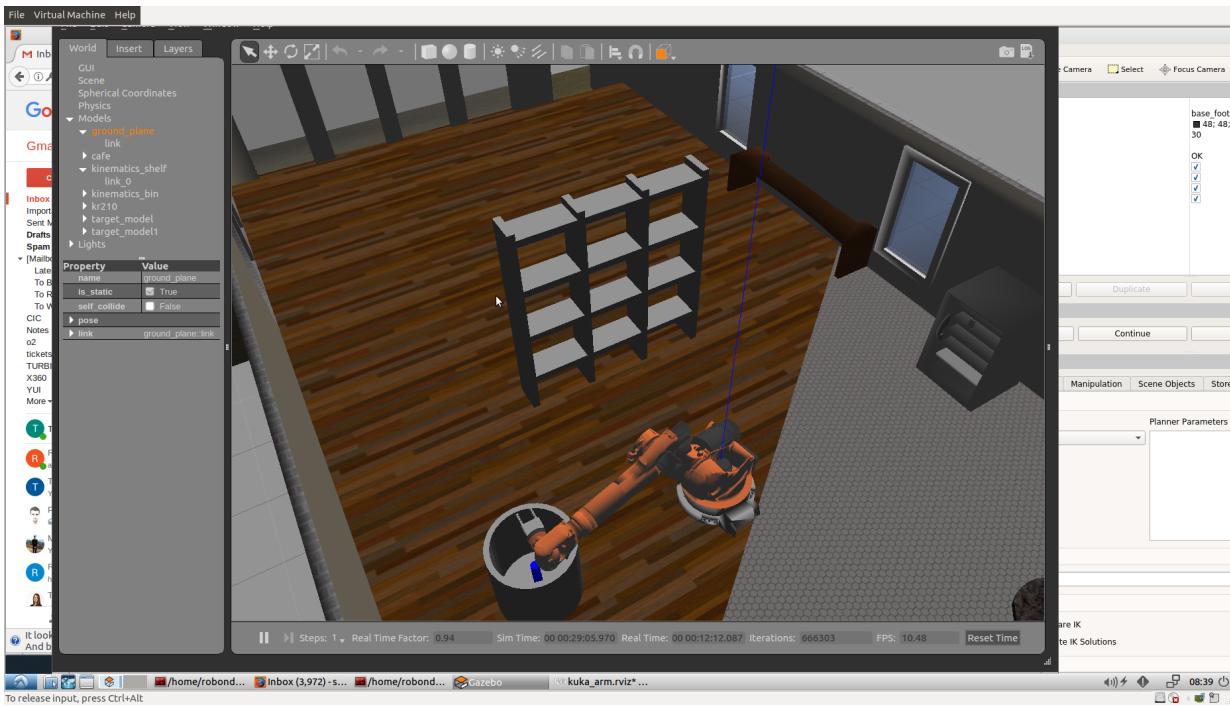


Here it is following the path planned by planner



And here it is before and after drop to the designated place





Video is included in the repo as well in file pick\_cycle.mp4

### Implementation difficulties

The implementation is not that difficult codewise if a person understands the underlying math which at times is a bit difficult since a lot of things are going on. I had the following issues.

#### Inversion stability

During the implementation I had one major issue and that was numerical stability of the implementations of inversion of matrices. I tried several implementations (LU decomposition, inversion, transposition) with different results on different computers. Unfortunately the notes are misleading here. During ~3 days of debugging I switched portions of the implementation for the walkthrough one and it still gave weird results until I switched the inverse for the transpose. Potentially something that might be improved in the future.

How it manifested in the project is that the end effector was spinning wildly. Since it picked cylinders occasionally and followed the path more or less I was not sure what is going on. Then I saw somebody's video posted on slack which did not have this behavior. By experimentation I was able to narrow it down to the inversion.

#### Cosine sentences

I had an issue with cosine sentences. I coded them wrong because of a typo and a complex number was introduced. This coupled with sympy simplification my IK\_debug just hung up and never finished so I was not sure what is going on. Again I was not sure if this is a problem of my computer or maybe issue connected to the VM but I was able to debug the problem in the cosine sentences eventually and got rid of some more expensive sympy operations pushing the debug execution run below a second or two.

#### Theta 4-6

For this part I had to consult the walkthrough. I understood process of extracting the angles from typical matrices but took me a while to realize where the elements are coming from in this particular implementation and why. I saw a comment in the slack giving some tips so after then it was fairly easy to use sympy to come up with the matrix and apply the "tricks" described above to extract the angles.

#### Areas for improvements

Since this project is fairly big the areas are not so much improvements but more areas for further exploration. - I would definitely like to understand the whole chain. Especially how the planner works since sometimes it does not feel like the path planned is optimal - ROS + gazebo seems like a very powerful tool

## **Attributions**

- Portions of the code are used from the project walkthrough