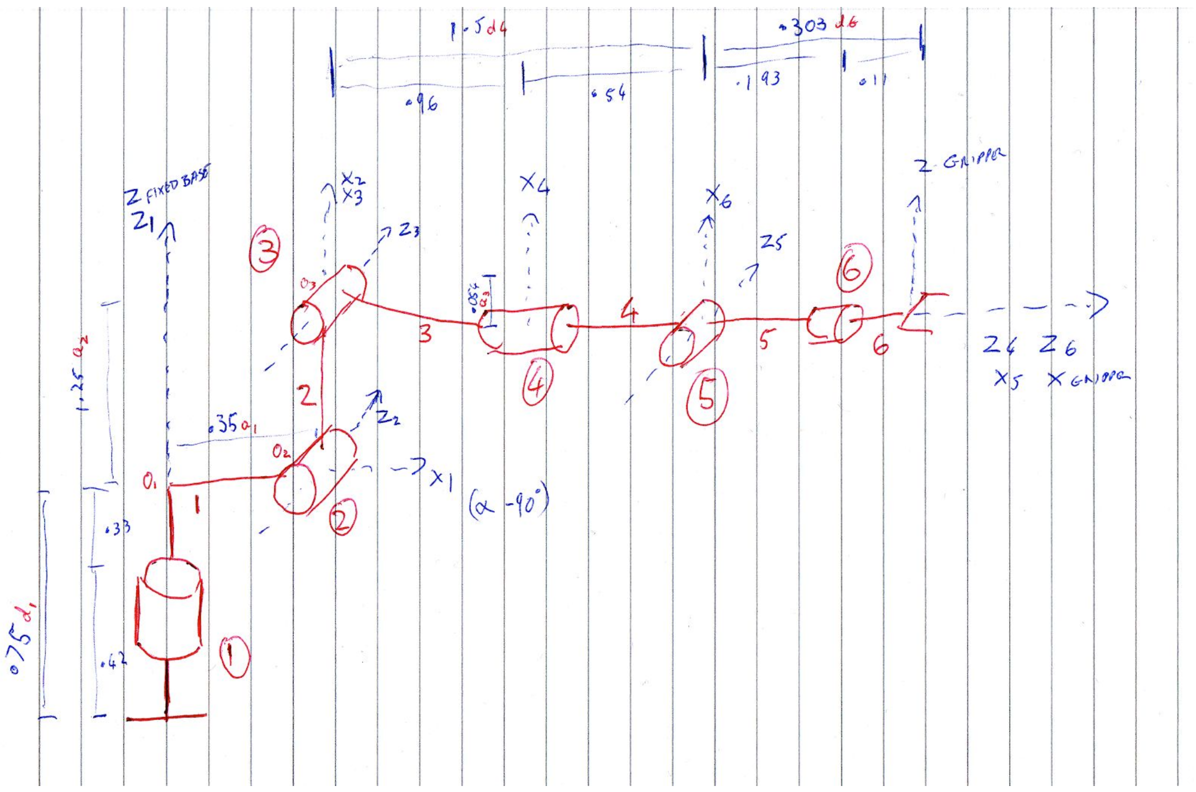
RoboND - Kinematics Project

Fergal Toohey

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.



Links	α (i-1)	a(i-1)	d(i)	θ (i)
0 -> 1	0	0	.75	q1
1 -> 2	- π /2	0.35	0	- π /2 + q2
2 -> 3	0	1.25	0	q3
3 -> 4	- π /2	-.056	1.5	q4

4 -> 5	π /2	0	0	q5
5 -> 6	- π /2	0	0	q6
6 -> EE	0	0	.303	0

The DH table was created from the RViz model, and the URDF file, according to the modified DH parameter convention used for this course.

Adjustments of note from the URDF:
Origins of Joints 4, 5, and 6 were set to be coincident, at Joint 5, thereby facilitating a spherical wrist model.
The gripper link (6->EE) was set therefore set to be the full length from Joint 5 to the gripper (.303)
Origin of Joint 1 was moved up its Z axis, so that be level with Joint 1
The rotation at Joint 2 was adjusted by -90 degrees to bring it in line with the DH convention.

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.

The homogenous transform from frame *i*-1, to frame *i* under the DH convention is defined as:

$$
{}^{i-1}_{\;\;i}\mathrm{T} = \begin{bmatrix} cos(q_i) & -sin(q_i) & 0 & a_{i-1} \\ sin(q_i)cos(\alpha_{i-1}) & cos(q_i)cos(\alpha_{i-1}) & -sin(\alpha_{i-1}) & -sin(\alpha_{i-1})d_i \\ sin(q_i)sin(\alpha_{i-1}) & cos(q_i)sin(\alpha_{i-1}) & cos(\alpha_{i-1}) & cos(\alpha_{i-1})d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

Thus, the transformation matrixes for each individual joint, using the parameters from the DH table, are as follows:

$$
{}^{0}_{1}\mathrm{T} = \begin{bmatrix} cos(q_1) & -sin(q_1) & 0 & a_0 \\ sin(q_1)cos(\alpha_0) & cos(q_1)cos(\alpha_0) & -sin(\alpha_0) & -sin(\alpha_0)d_1 \\ sin(q_1)sin(\alpha_0) & cos(q_1)sin(\alpha_0) & cos(\alpha_0) & cos(\alpha_0)d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

$$
{}^{1}_{2}\mathrm{T} = \begin{bmatrix} cos(q_2) & -sin(q_2) & 0 & a_1 \\ sin(q_2)cos(\alpha_1) & cos(q_2)cos(\alpha_1) & -sin(\alpha_1) & -sin(\alpha_1)d_2 \\ sin(q_2)sin(\alpha_1) & cos(q_2)sin(\alpha_1) & cos(\alpha_1) & cos(\alpha_1)d_2 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

$$
{}^{2}_{3}\mathrm{T} = \begin{bmatrix} cos(q_3) & -sin(q_3) & 0 & a_2 \\ sin(q_3)cos(\alpha_2) & cos(q_3)cos(\alpha_2) & -sin(\alpha_2) & -sin(\alpha_2)d_3 \\ sin(q_3)sin(\alpha_2) & cos(q_3)sin(\alpha_2) & cos(\alpha_2) & cos(\alpha_2)d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

…

$$
{}^{6}_{EE}\mathrm{T} = \begin{bmatrix} cos(q_7) & -sin(q_7) & 0 & a_6 \\ sin(q_7)cos(\alpha_6) & cos(q_7)cos(\alpha_6) & -sin(\alpha_6) & -sin(\alpha_6)d_7 \\ sin(q_7)sin(\alpha_6) & cos(q_7)sin(\alpha_6) & cos(\alpha_6) & cos(\alpha_6)d_7 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

Generalized homogenous transform

The generalized homogenous transform between the base pose and the End Effector pose is equal to the end effector roll, pitch, and yaw supplied by ROS (*r, p, y*), post-multiplied by correction rotations about the z axis (180 degrees) and y axis (-90 degrees), to remove the discrepancy between Gazebo and the DH parameters. This corrected rotation matrix was created in python. [In IK_server.py, it is the sympy matrix referred to as R_EE]

A homogenous transformation can then be derived by combining the corrected rotation matrix with the translation from End Effector position relative to the base (pos_{xyz}):

$$
{}^{0}_{EE}\mathrm{T} = \begin{bmatrix} s(p)c(r)c(y) + s(r)s(y) & -s(p)s(r)c(y) + s(y)c(r) & c(p)c(y) & pos_x \\ s(p)s(y)c(r) - s(r)c(y) & -s(p)s(r)s(y) - c(r)c(y) & s(y)c(p) & pos_y \\ c(p)c(r) & -s(r)c(p) & -s(p) & pos_z \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

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

The last three joints of the Kuka 210 define a Spherical Wrist, as they are revolute and their joint axes intersect at a single point, Joint 5, the Wrist Centre.

This means that the manipulator is solvable in closed form, and the position and orientation of the end effector are kinematically decoupled from one another, into the Inverse Position and Inverse Orientation problems, respectively.

This reduces the values that need to be determined for the last three joints:
- the coordinates of the Wrist Centre, and
- the composition of rotations required to orient the end effector.

Once the Wrist Centre is obtained, the angles of Joints 1-3 can be obtained by trigonometry.

Coordinates of the Wrist Centre:

We have already calculated the End Effector's corrected rotation, so to get the wrist position, we translate back from the End Effector along the End Effector's Z axis, by *d7*, to arrive at Joint 5, the Wrist Centre:

$$
\begin{aligned}
WC_x &= pos_x - d_7 \cdot n_x \\
WC_y &= pos_y - d_7 \cdot n_y \\
WC_z &= pos_z - d_7 \cdot n_z
\end{aligned}
$$

Where *pos* is the End Effector position, *d7* is the value from the DH table, and n_{xyz} is the Z component of the EE's corrected rotation.

Theta 1, 2, 3:

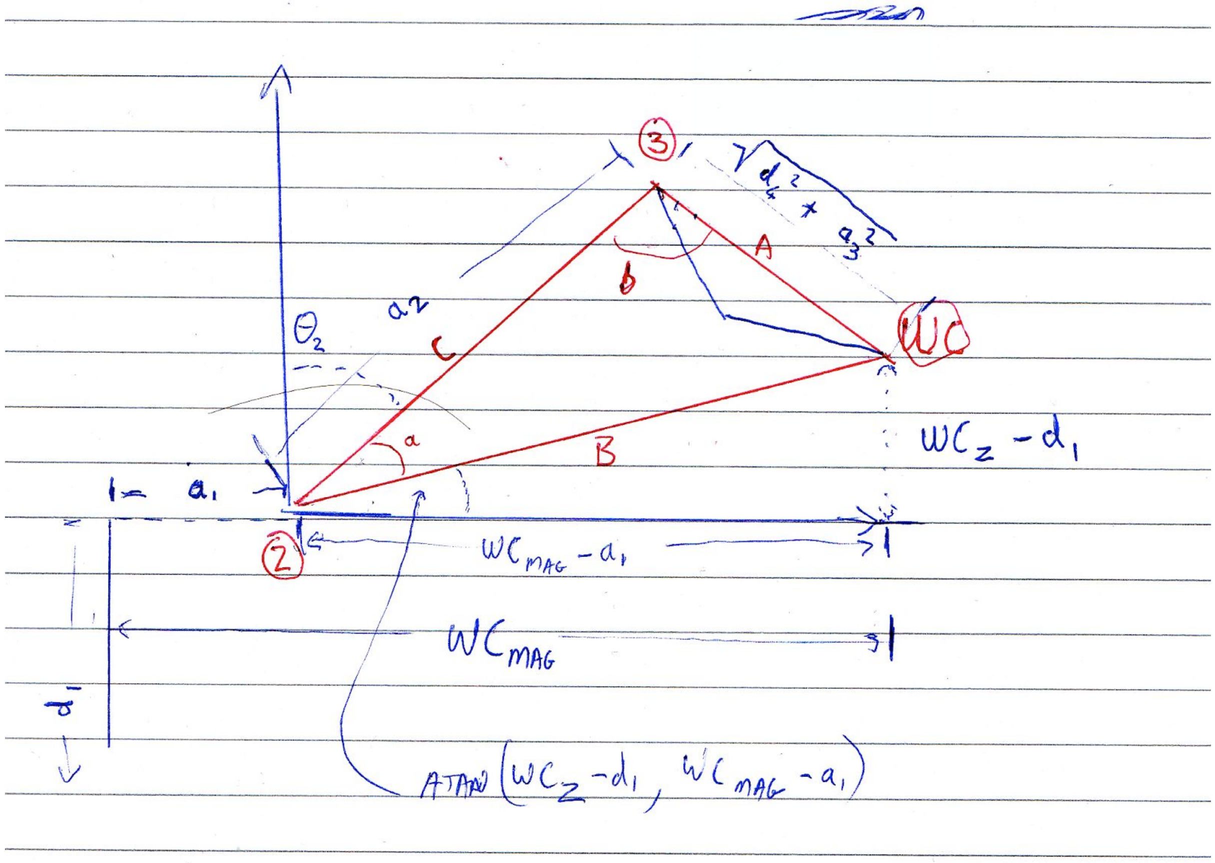
From this, the value for θ_1 can be directly deduced: θ_1 is the horizontal rotation of the base joint, and can be deduced by the XY component of the wrist centre:

$$\theta_1 = atan2(WC_y, WC_x)$$

The distance on the base link's XY plane, from the wrist centre to the origin is required to derive the other angles:

$$WC_{mag} = \sqrt{{WC_x}^2 \cdot {WC_y}^2}$$

Thus the positions of WC and Joint 2 are known, as are the lengths of the links between them. This means that θ_2 and θ_3 can be determined by analysing the triangle created by Joint 2, Joint 3, and the Wrist Centre, as shown below:.



Side A: Distance between Joint 3, and Wrist Centre:

$$A = \sqrt{{a_3}^2 \cdot {d_4}^2}$$

Side B: Distance between Wrist Centre and Joint 2:

$$B = \sqrt{(WC_{mag} - a_1)^2 \cdot (WC_z - d_1)^2}$$

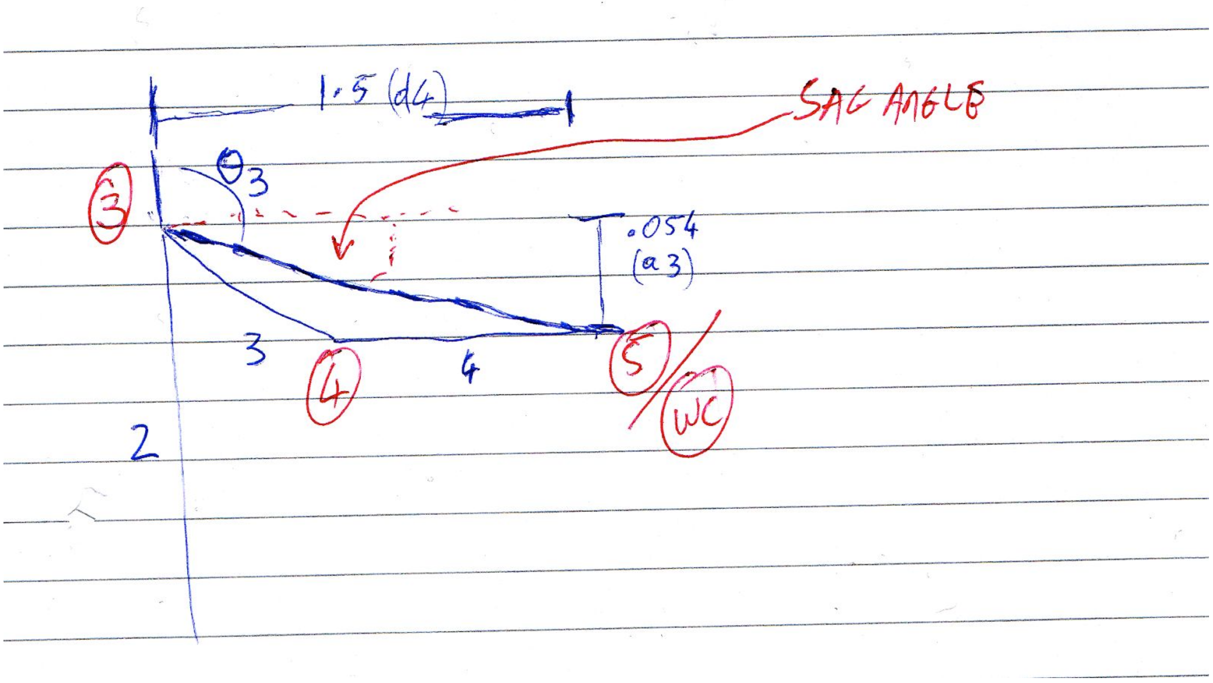
Side C: Distance between Joints 2 and 3:

$C = a_2$

As all sides are known, apply Cosine rules to determine angles.

$$a = acos((B^2 + C^2 - A^2)/2BC)$$
$$b = acos((A^2 + C^2 - B^2)/2AC)$$

Account for Sag at between Joint 3 and the wrist centre:



$$sag = atan2(a_3, d_4)$$

Use these angles to determine thetas 2 and 3

$$\theta_2 = \pi/2 - a - atan2(WC_z - d_1, WC_{mag} - a_1)$$
$$\theta_3 = \pi/2 - sag - b$$

Theta 4, 5, 6

Composition of extracted rotation matrices for rotations 0 -> 3

$$^0_3R = ^0_1T \cdot ^1_2T \cdot ^2_3T$$

Get rotation of spherical wrist joints by applying end effector rotation matrix to inverse of R0_3

$$^3_6R = ^0_3R^{-1} \cdot ^0_{EE}R$$

Extract euler angles:

To identify the symbolic relationship of the Euler angles, R3_EE was calculated symbolically in Python.

$$ {}^{3}_{EE}R = {}^{3}_{4}R \cdot {}^{4}_{5}R \cdot {}^{5}_{6}R \cdot {}^{6}_{EE}R $$

Producing the following matrix:

$$ {}^{3}_{EE}R = \begin{bmatrix} -s(\theta_4)s(\theta_6) + c(\theta_4)c(\theta_5)c(\theta_6) & -s(\theta_4)c(\theta_6) - s(\theta_6)c(\theta_4)c(\theta_5) & -s(\theta_5)c(\theta_4) \\ s(\theta_5)c(\theta_6) & -s(\theta_5)s(\theta_6) & c(\theta_5) \\ -s(\theta_4)c(\theta_5)c(\theta_6) - s(\theta_6)c(\theta_4) & s(\theta_4)s(\theta_6)c(\theta_5) - c(\theta_4)c(\theta_6) & s(\theta_4)s(\theta_5) \end{bmatrix} $$

Which can be solved for $\theta_{4,5,6}$ using arctan2:

$$ atan2(sin\theta, cos\theta) = \theta $$

Therefore,

$$ \theta_5 = atan2(\sqrt{r_{3,3}{}^2 + -r_{1,3}{}^2}, r_{2,3}) $$
$$ \theta_4 = atan2(r_{3,3}, -r_{1,3}) $$
$$ \theta_6 = atan2(-r_{2,2}, r_{2,1}) $$

Note:

Multiple solutions:

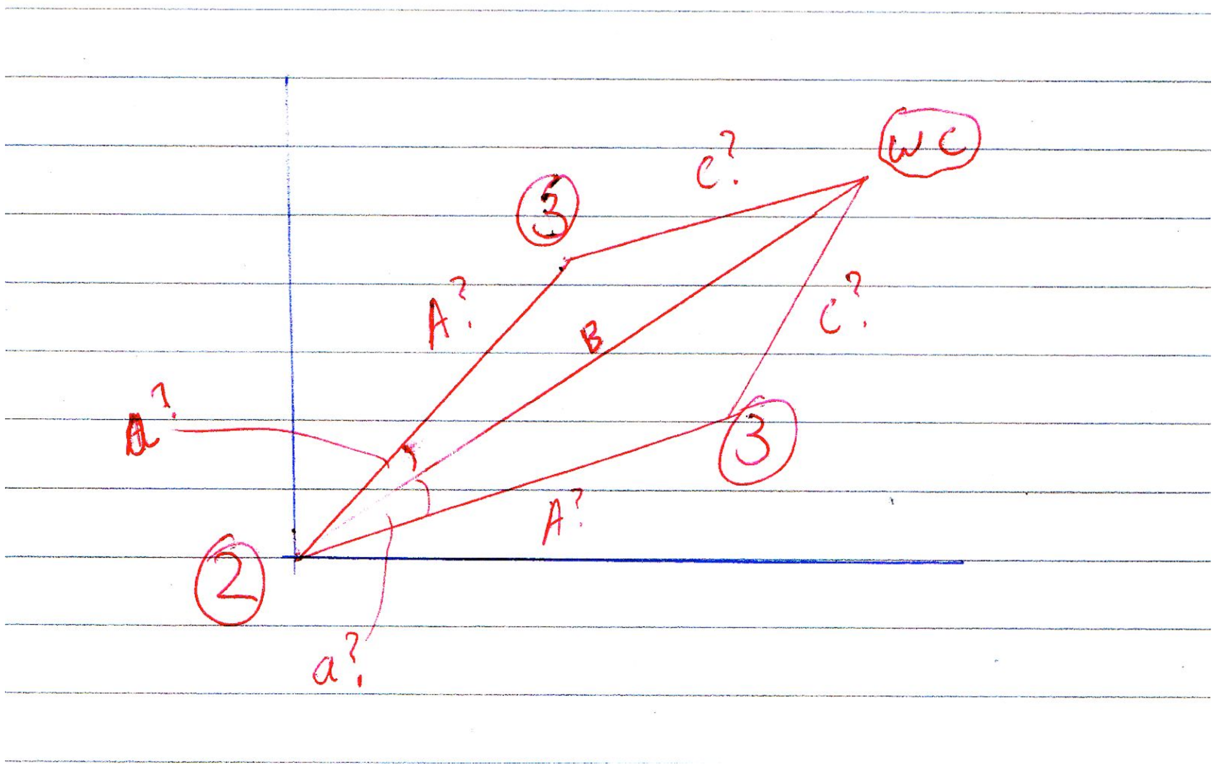
The inputs to θ_5 cleanly cancel down to [Sin θ_5, Cos θ_5].

However, the inputs to θ_4 and θ_6 do not solely contain functions of their respective angles, instead they are also functions of Sin θ_5 which means that their solution is sensitive to the sign of Sin θ_5.

Therefore, if Sin θ_5 is < 0, the correct equations for θ_4 and θ_6 are:

$$ \theta_4 = atan2(-r_{3,3}, r_{1,3}) $$
$$ \theta_6 = atan2(r_{2,2}, -r_{2,1}) $$

Multiple Solutions / Arccos

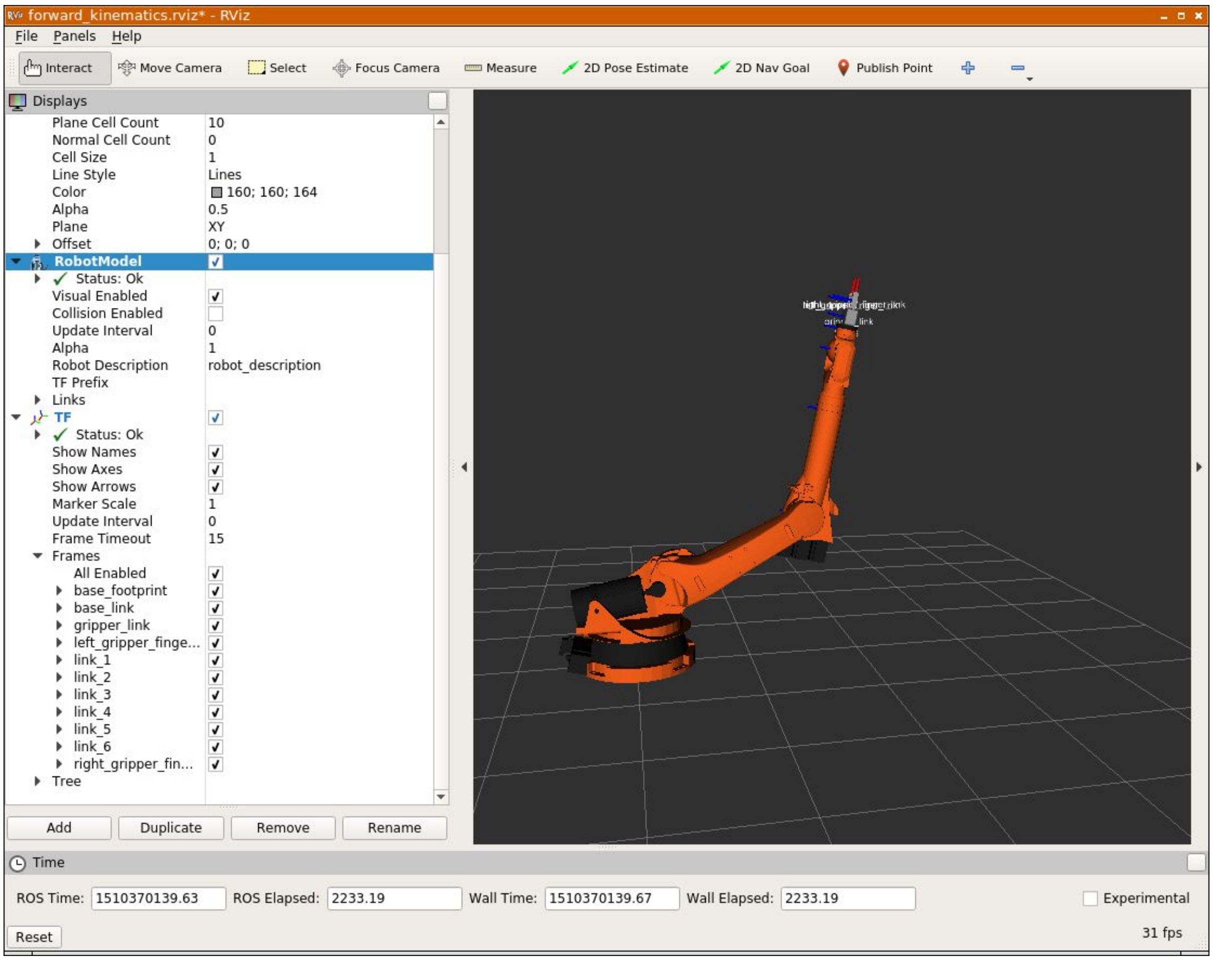


Arccos, as used earlier to find angle *a* and *b* in the process of deriving θ_2 and θ_3, is typically avoided in robotics. It does not supply the direction of the angle, nor can it return angles outside the range 0-180 degrees. This causes the multiple solutions in the illustration above: arccos effectively defines two different triangles. Arctan2 is preferable in most contexts as it avoids this. In this context, however, this ambiguity can be eliminated by simply choosing the upper solution in all scenarios.

In practice, this leads to a specific limitation on the movement range of the robot: neither “Angle a” nor “Angle b” can ever be negative, as they are limited to 0-180 degrees. This means that robot poses with a reflex angle at Joint 3 (i.e. placing it below the line between the Wrist Centre and Joint 2) are not possible.

On exploring the kinematics in Rviz, however, this limitation is not an issue. Whilst there are certain obscure joint arrangements that can’t be reached (which could be important in car assembly, for example [screenshots below]) none of these will be encountered in the Pick and

Place task, of moving an item from the shelf to the bucket.

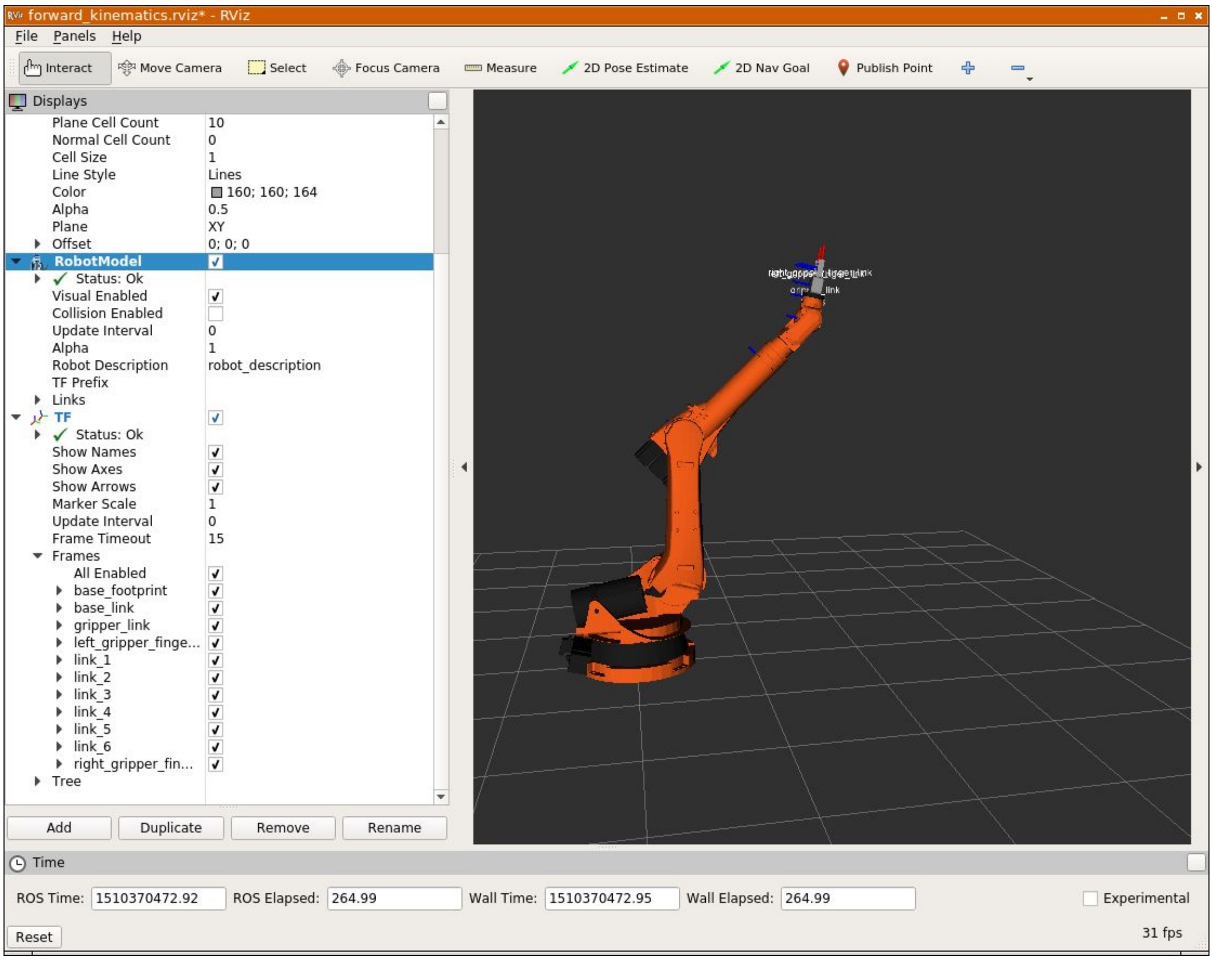


Plane Cell Count	10
Normal Cell Count	0
Cell Size	1
Line Style	Lines
Color	▨ 160; 160; 164
Alpha	0.5
Plane	XY
Offset	0; 0; 0
RobotModel	✓
Status: Ok	
Visual Enabled	✓
Collision Enabled	
Update Interval	0
Alpha	1
Robot Description	robot_description
TF Prefix	
Links	
TF	✓
Status: Ok	
Show Names	✓
Show Axes	✓
Show Arrows	✓
Marker Scale	1
Update Interval	0
Frame Timeout	15
Frames	
All Enabled	✓
base_footprint	✓
base_link	✓
gripper_link	✓
left_gripper_finge...	✓
link_1	✓
link_2	✓
link_3	✓
link_4	✓
link_5	✓
link_6	✓
right_gripper_fin...	✓
Tree	

ROS Time: 1510370139.63 ROS Elapsed: 2233.19 Wall Time: 1510370139.67 Wall Elapsed: 2233.19 Experimental

31 fps

An example of a reflex angle position that can't be achieved with this approach for choosing Joints 2 and 3.



Plane Cell Count	10
Normal Cell Count	0
Cell Size	1
Line Style	Lines
Color	160; 160; 164
Alpha	0.5
Plane	XY
Offset	0; 0; 0
RobotModel	✓
Status: Ok	
Visual Enabled	✓
Collision Enabled	
Update Interval	0
Alpha	1
Robot Description	robot_description
TF Prefix	
Links	
TF	✓
Status: Ok	
Show Names	✓
Show Axes	✓
Show Arrows	✓
Marker Scale	1
Update Interval	0
Frame Timeout	15
Frames	
All Enabled	✓
base_footprint	✓
base_link	✓
gripper_link	✓
left_gripper_finge...	✓
link_1	✓
link_2	✓
link_3	✓
link_4	✓
link_5	✓
link_6	✓
right_gripper_fin...	✓
Tree	

ROS Time: 1510370472.92 ROS Elapsed: 264.99 Wall Time: 1510370472.95 Wall Elapsed: 264.99 Experimental

31 fps

An example of the solution this implementation would select to achieve the same gripper pose.

Project Implementation

The project was implemented mostly by using the Sympy library. These symbolic matrices allow for the the construction of generic rotation and translation matrices, containing mathematical and trigonometric operators and symbols, that can be manipulated independently of numerical parameters. This permits the construction of symbolic matrices for the relationship between joints, positions, and reference frames, thereby meaning that the inverse kinematics do not need to be completely re-derived for every pose: all that is required is to substitute in the required values for symbols when the function is called.

Preparatory code:

Symbols were instantiated for every element of the DH table, together with symbols for the requested Roll Pitch and Yaw of the end effector.

A corresponding dictionary was created with values for the non-variable symbols, containing the constants, such as link length. The joint angles (i.e. the variables in the system) were not included in this dictionary.

These symbols, and the dictionary, were used to create homogenous transformation matrices for the rotations and translations between each joint in the system.

When handle_calculate_IK() receives a ROS request to calculate Inverse Kinematics, it typically receives a list of many poses to work through. Therefore, it is important to move any code that can be pre-calculated outside the main loop.

In this case: the symbolic composition of the rotations from the Base to Frame 3, and the symbolic rotation matrix for the pose of the End Effector, together with its corrections for the offsets between the DH parameters and Gazebo, are all derived and simplified before looping through each pose. Additionally, pickle is used to store these symbolic matrices to disk, for additional efficiency.

As these operations take approximately over a second to complete each time [`R_EE_sym = simplify(R_EE_sym * R_EE_corr)` took 0.906 seconds when timed in isolation], by removing them from the main loop reduced the calculation of each pose to ~0.05 seconds, as seen in this excerpt of timing printouts from one set of poses:

```
Pose 0: 0.0454950332642
Pose 1: 0.0475261211395
Pose 2: 0.0498931407928
Pose 3: 0.0503730773926
…
```

Main loop

Within the loop for processing responses for each requested pose:

- The requested pose is substituted into the symbolic matrices (position extracted directly, whilst roll, pitch and yaw are converted from the quaternion supplied from ROS)

- The wrist centre is calculated from the End Effector pose, and the DH parameter d7.

- The trigonometric calculations described above are then performed to calculate Thetas 1, 2 and 3. The pre-derived symbolic rotation matrix R0_3 is then evaluated with the values of these first three angles.

- The transposition of R0_3 is then post multiplied by the End Effector rotation, to produce the R3_6 rotation matrix.

- From this matrix, the Euler angles are extracted, again, this follows the formulas outlined in the above section.

Development process:

The code was first implemented in IK_debug.py, and relevant code transferred to IK_server.py on completion.

One difference from the suggested code in the lessons is the use of R0_3.transpose() instead of R0_3.inv('LU'). As pointed out by other students in Slack, .inv('LU') leads to unstable results in this scenario, often leading to a ~.30 offset on EE position.

On replacing this code with transpose(), which in this instance is equivalent, EE offset stays robustly at 0, as shown in the Test Case outputs:

Test Case 1 output (transpose):

```
Wrist error for x position is: 0.00000046
Wrist error for y position is: 0.00000032
Wrist error for z position is: 0.00000545
Overall wrist offset is: 0.00000548 units

Theta 1 error is: 0.00093770
Theta 2 error is: 0.00178633
Theta 3 error is: 0.00206506
Theta 4 error is: 0.00172809
Theta 5 error is: 0.00198404
```

```
        Theta 6 error is: 0.00252923

        End effector error for x position is: 0.00000000
        End effector error for y position is: 0.00000000
        End effector error for z position is: 0.00000000
```
Overall end effector offset is: 0.00000000 units

Test Case 2 output (transpose):

```
        Wrist error for x position is: 0.00002426
        Wrist error for y position is: 0.00000562
        Wrist error for z position is: 0.00006521
        Overall wrist offset is: 0.00006980 units

        Theta 1 error is: 3.14309971
        Theta 2 error is: 0.27927962
        Theta 3 error is: 1.86833314
        Theta 4 error is: 3.08639539
        Theta 5 error is: 0.06340277
        Theta 6 error is: 6.13524929

        End effector error for x position is: 0.00000000
        End effector error for y position is: 0.00000000
        End effector error for z position is: 0.00000000
```
Overall end effector offset is: 0.00000000 units

Test case 3 output (transpose):

```
        Wrist error for x position is: 0.00000503
        Wrist error for y position is: 0.00000512
        Wrist error for z position is: 0.00000585
        Overall wrist offset is: 0.00000926 units

        Theta 1 error is: 0.00136747
        Theta 2 error is: 0.00329800
        Theta 3 error is: 0.00339863
        Theta 4 error is: 6.28213720
        Theta 5 error is: 0.00287049
        Theta 6 error is: 6.28227458

        End effector error for x position is: 0.00000000
        End effector error for y position is: 0.00000000
        End effector error for z position is: 0.00000000
```
Overall end effector offset is: 0.00000000 units

Test case 1 output (inv(‘LU’)):

```
End effector error for x position is: 0.00000000
End effector error for y position is: 0.00000000
End effector error for z position is: 0.00000000
Overall end effector offset is: 0.00000000 units
```

Test case 2 output (inv(‘LU’)):

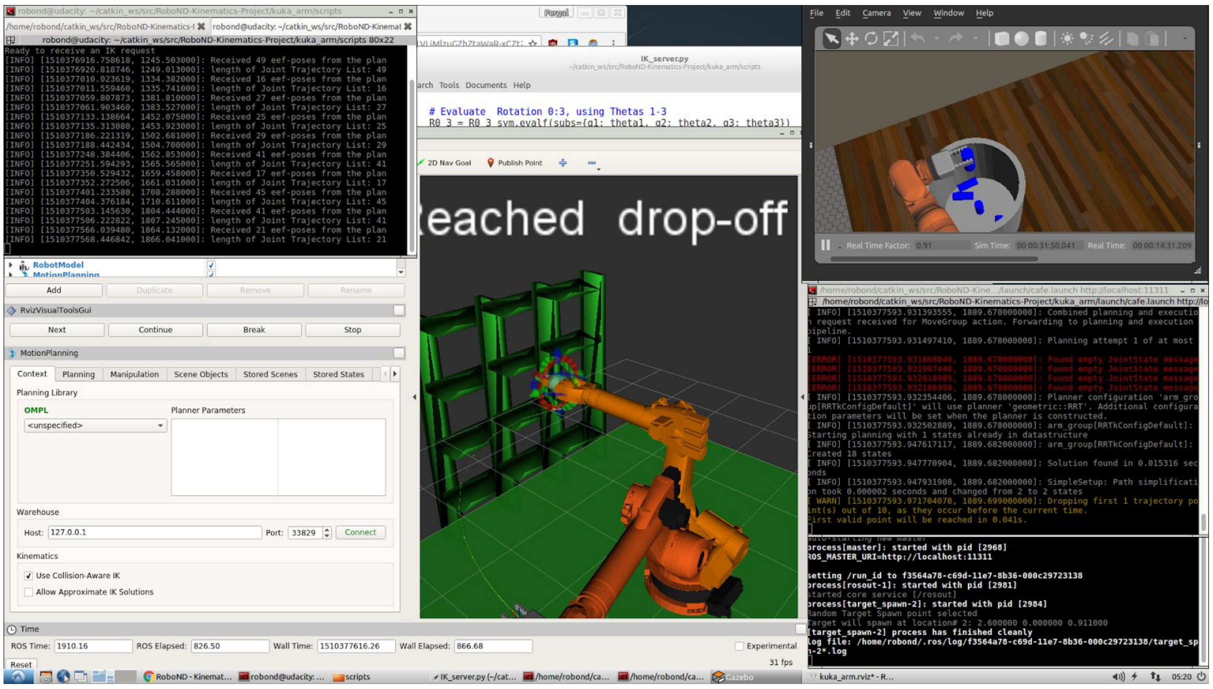
```
End effector error for x position is: 0.00606313
End effector error for y position is: 0.02534562
End effector error for z position is: 0.12809810
Overall end effector offset is: 0.13072217 units
```

Test case 3 output (inv(‘LU’)):

```
End effector error for x position is: 0.07852251
End effector error for y position is: 0.07180827
End effector error for z position is: 0.29378734
Overall end effector offset is: 0.31246314 units
```

Results:

The Robot successfully picks up the object from all tested positions.



```
Ready to receive an IK request
[INFO] [1510376916.758618, 1245.503000]: Received 49 eef-poses from the plan
[INFO] [1510376920.818746, 1249.013000]: length of Joint Trajectory List: 49
[INFO] [1510377010.023619, 1334.382000]: Received 16 eef-poses from the plan
[INFO] [1510377011.559460, 1335.741000]: length of Joint Trajectory List: 16
[INFO] [1510377059.807873, 1381.810000]: Received 27 eef-poses from the plan
[INFO] [1510377061.903460, 1383.527000]: length of Joint Trajectory List: 27
[INFO] [1510377133.138664, 1452.075000]: Received 25 eef-poses from the plan
[INFO] [1510377135.313080, 1453.923000]: length of Joint Trajectory List: 25
[INFO] [1510377186.221319, 1502.681000]: Received 29 eef-poses from the plan
[INFO] [1510377188.442434, 1504.700000]: length of Joint Trajectory List: 29
[INFO] [1510377248.384406, 1562.853000]: Received 41 eef-poses from the plan
[INFO] [1510377251.594293, 1565.565000]: length of Joint Trajectory List: 41
[INFO] [1510377350.529432, 1659.458000]: Received 17 eef-poses from the plan
[INFO] [1510377352.272506, 1661.031000]: length of Joint Trajectory List: 17
[INFO] [1510377401.233580, 1708.288000]: Received 45 eef-poses from the plan
[INFO] [1510377404.376184, 1710.611000]: length of Joint Trajectory List: 45
[INFO] [1510377503.145630, 1804.444000]: Received 41 eef-poses from the plan
[INFO] [1510377506.222822, 1807.245000]: length of Joint Trajectory List: 41
[INFO] [1510377566.039480, 1864.132000]: Received 21 eef-poses from the plan
[INFO] [1510377568.446842, 1866.041000]: length of Joint Trajectory List: 21
```

```
# Evaluate  Rotation 0:3, using Thetas 1-3
R0_3 = R0_3.evalf(subs={q1: theta1, q2: theta2, q3: theta3})
```

Planning Library: OMPL <unspecified>
Host: 127.0.0.1 Port: 33829
[✓] Use Collision-Aware IK
[ ] Allow Approximate IK Solutions
ROS Time: 1910.16 ROS Elapsed: 826.50 Wall Time: 1510377616.26 Wall Elapsed: 866.68
[ ] Experimental 31 fps

```
[ INFO] [1510377593.931393555, 1889.678000000]: Combined planning and executio
n request received for MoveGroup action. Forwarding to planning and execution
pipeline.
[ INFO] [1510377593.931497410, 1889.678000000]: Planning attempt 1 of at most
1
[ERROR] [1510377593.931868840, 1889.678000000]: Found empty JointState message
[ERROR] [1510377593.931907440, 1889.678000000]: Found empty JointState message
[ERROR] [1510377593.932018989, 1889.678000000]: Found empty JointState message
[ERROR] [1510377593.932186998, 1889.678000000]: Found empty JointState message
[ INFO] [1510377593.932354406, 1889.678000000]: Planner configuration 'arm_gro
up[RRTkConfigDefault]' will use planner 'geometric::RRT'. Additional configura
tion parameters will be set when the planner is constructed.
[ INFO] [1510377593.932502889, 1889.678000000]: arm_group[RRTkConfigDefault]:
Starting planning with 1 states already in datastructure
[ INFO] [1510377593.947617117, 1889.682000000]: arm_group[RRTkConfigDefault]:
Created 18 states
[ INFO] [1510377593.947770904, 1889.682000000]: Solution found in 0.015316 sec
onds
[ INFO] [1510377593.947931908, 1889.682000000]: SimpleSetup: Path simplificati
on took 0.000002 seconds and changed from 2 to 2 states
[ WARN] [1510377593.971704078, 1889.699000000]: Dropping first 1 trajectory po
int(s) out of 10, as they occur before the current time.
First valid point will be reached in 0.041s.
```

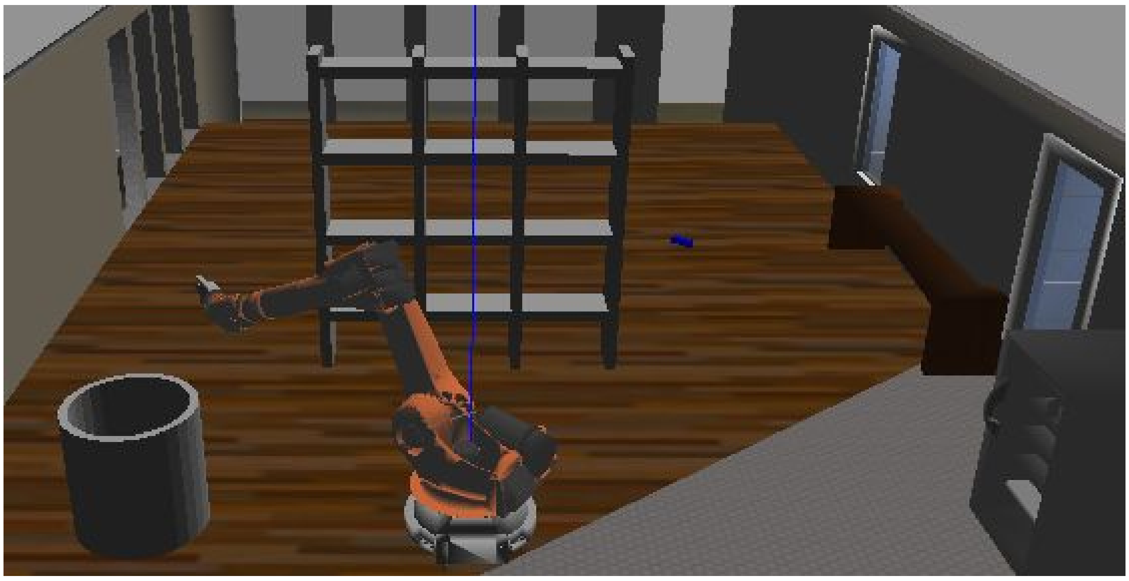
```
process[master]: started with pid [2968]
ROS_MASTER_URI=http://localhost:11311

setting /run_id to f3564a78-c69d-11e7-8b36-000c29723138
process[rosout-1]: started with pid [2981]
started core service [/rosout]
process[target_spawn-2]: started with pid [2984]
Random Target Spawn point selected
Target will spawn at location# 2: 2.600000 0.000000 0.911000
[target_spawn-2] process has finished cleanly
log file: /home/robond/.ros/log/f3564a78-c69d-11e7-8b36-000c29723138/target_sp
n-2*.log
```

Drop-off reached, multiple items in bucket.

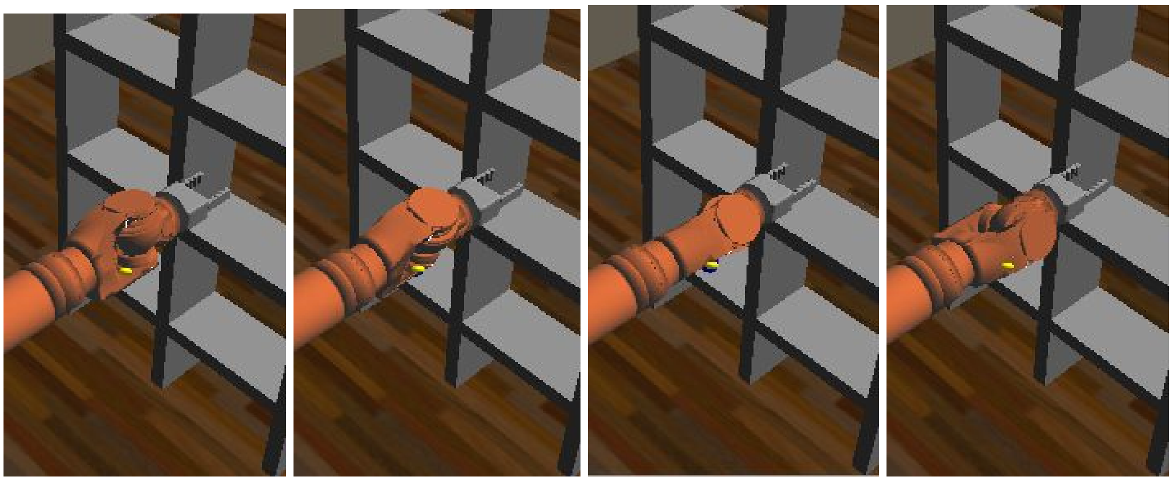
On occasion, however, after the successful pick-up, the path from MoveIt brings the robot too close to the shelves, and the object clips an upright, and gets pulled out of the gripper.

Holding object very close to the shelves...


...seconds later, after the object clipped the shelf

This is behaviour that was demonstrated in the “demo” IK functionality, and thus appears to be a MoveIt bug, rather than a IK bug from this implementation.

The robot also has a tendency to apply a seemingly unnecessary 360 degree rotation to the end effector (or one of the other wrist joints) when making even the slightest adjustments, such as reaching for the middle centre object from the start point. Further analysis is warranted to identify if this is the nature of the robot/MoveIt, or a quirk of this Euler angle implementation.


Snapshots of a seemingly unnecessary ~360 degree wrist rotation.