Introduction to Robotics
Assignment #3

Caus Danu 7014833   Massimo Innocentini 7016313   Daniel Bremer 6834136

Task 3.1 (4 points) Screw Cap:

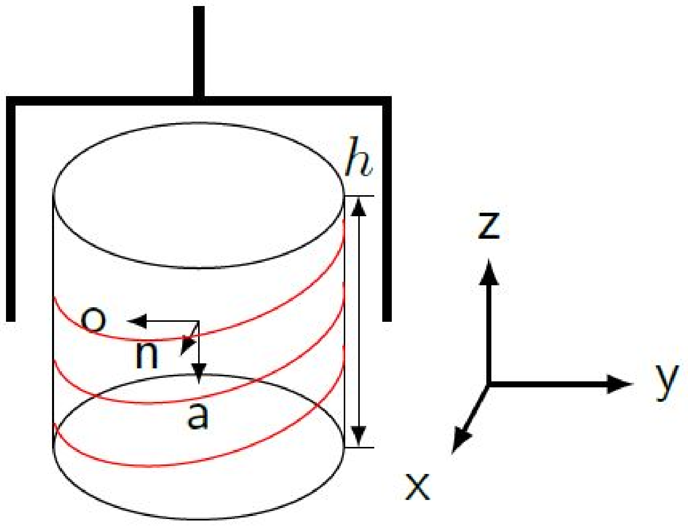

Figure 1: Loosening of a screw cap

The angle θ of rotation depends on time and can be computed as follows:

$$\theta(t) = \omega_z * t$$

Hence, all the sines and cosines that have as argument θ will use the above equivalence instead:

$$n(t) = \begin{bmatrix} cos(\omega_z * t) \\ sin(\omega_z * t) \\ 0 \end{bmatrix}$$

$$o(t) = \begin{bmatrix} cos(\omega_z * t - \frac{\pi}{2}) \\ sin(\omega_z * t - \frac{\pi}{2}) \\ 0 \end{bmatrix} = \begin{bmatrix} sin(\omega_z * t) \\ -cos(\omega_z * t) \\ 0 \end{bmatrix}$$

$$a(t) = \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix}$$

$\omega_z = 2\pi f = \frac{2\pi}{T} => T = \frac{2\pi}{\omega_z}$
$v_z = \frac{threadheight}{T} = \frac{h/\#rot}{T} = \frac{h}{\#rot} * \frac{\omega_z}{2\pi}$
$d_z = v_z * t = \frac{h\omega_z}{2\pi * \#rot} * t$
Therefore

$$d(t) = \begin{bmatrix} 0 \\ 0 \\ \frac{h\omega_z}{2\pi * \#rot} * t \end{bmatrix}$$

As a result, the time-dependent homogeneous transformation looks as follows:

$$T(t) = \begin{bmatrix} cos(\omega_z * t) & sin(\omega_z * t) & 0 & 0 \\ sin(\omega_z * t) & -cos(\omega_z * t) & 0 & 0 \\ 0 & 0 & -1 & \frac{h\omega_z}{2\pi * \#rot} * t \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Task 3.2 (4 points) DH-Parameters:

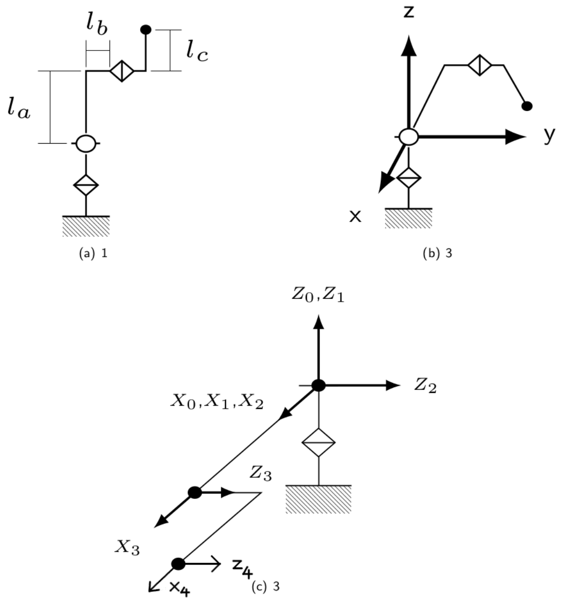

Figure 2: Robot workspace

3.2.1 (2 points): The DH-parameters can be found in Table 1:

Link	θ_i	d_i	a_i	α_i
1	θ_1	0	0	0
2	θ_2	0	0	-90
3	θ_3	0	l_a	0
4	0	l_b	l_c	0

Table 1: DH-parameters for the 3 degrees of freedom manipulator

3.2.2 (2 points): Homogeneous transformations determined with the DH parameters from Table 1:

$$^0A_1 = \begin{bmatrix} C_1 & -S_1 & 0 & 0 \\ S_1 & C_1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad ^1A_2 = \begin{bmatrix} C_2 & 0 & -S_2 & 0 \\ S_2 & 0 & C_2 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$^2A_3 = \begin{bmatrix} C_3 & -S_3 & 0 & l_aC_3 \\ S_3 & C_3 & 0 & l_aS_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad ^3A_4 = \begin{bmatrix} 1 & 0 & 0 & l_c \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & l_b \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$^{0}T_1 = {}^{0}A_1 \tag{1}$$

$$\begin{aligned}
^{0}T_2 &= {}^{0}A_1{}^{1}A_2 \\
&= \begin{bmatrix} C_1C_2 - S_1S_2 & 0 & -C_2S_1 - C_1S_2 & 0 \\ S_1C_2 + C_1S_2 & 0 & C_1C_2 - S_1S_2 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} C_{12} & 0 & -S_{12} & 0 \\ S_{12} & 0 & C_{12} & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
\end{aligned} \tag{2}$$

$$\begin{aligned}
^{0}T_3 &= {}^{0}A_1{}^{1}A_2{}^{2}A_3 \\
&= \begin{bmatrix} C_3C_{12} & S_3 - C_{12} & -S_{12} & l_aC_3C_{12} \\ C_3S_{12} & S_3 - S_{12} & C_{12} & l_aC_3S_{12} \\ -S_3 & -C_3 & 0 & -l_aS_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}
\end{aligned} \tag{3}$$

Task 3.3 (4 points) Grasping from Camera:

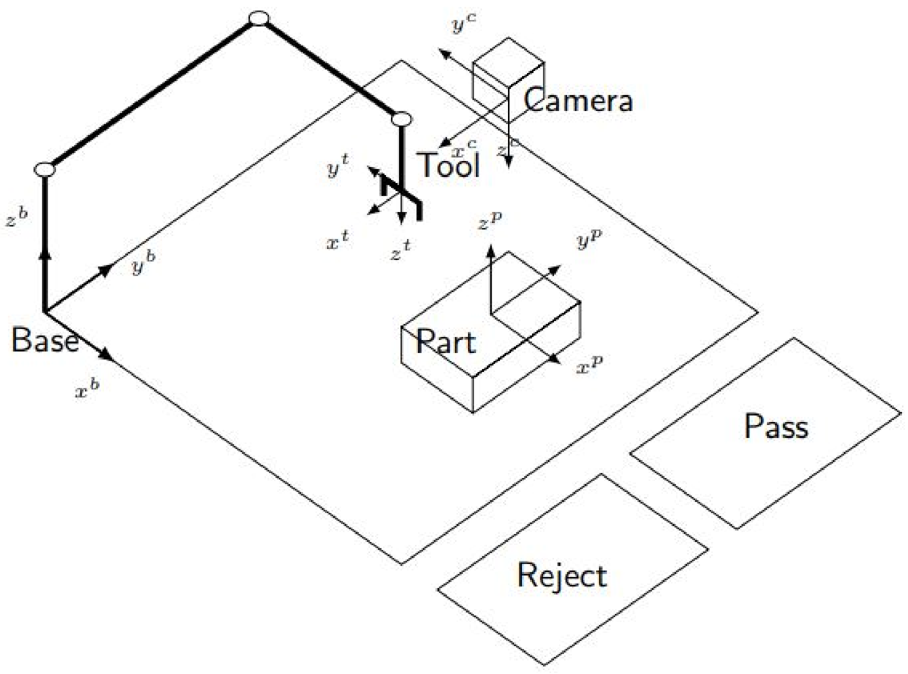

Figure 3: Robot workspace

3.3.1 (2 points):

$$^{base}T_{object} = ^{base}T_{camera} * ^{camera}T_{object} = ^{camera}T_{base}^{-1} * ^{camera}T_{object}$$

$$^{camera}T_{base}^{-1} = ^{base}T_{camera} = \begin{bmatrix} 0 & -1 & 0 & 11 \\ -1 & 0 & 0 & 28 \\ 0 & 0 & -1 & 8 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$^{base}T_{object} = \begin{bmatrix} 0 & -1 & 0 & 11 \\ -1 & 0 & 0 & 28 \\ 0 & 0 & -1 & 8 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 0 & -1 & 0 & 17 \\ -1 & 0 & 0 & -7 \\ 0 & 0 & -1 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 18 \\ 0 & 1 & 0 & 11 \\ 0 & 0 & 1 & 5 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3.3.2 (2 points):

$$^{base}T_{tool} = \begin{bmatrix} x^t & y^t & z^t & p^t \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$x^t = -y^p$$

$$y^t = -x^p$$

$$z^t = -z^p$$

Therefore

$$^{base}T_{tool} = \begin{bmatrix} 0 & -1 & 0 & 18 \\ -1 & 0 & 0 & 11 \\ 0 & 0 & -1 & 5 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Task 3.4 (3 points) Repetition precision:

The positioning accuracy depends on the accuracy of the motors themselves, i.e how coarse/fine of a motion they can perform.

Another factor can be the resolution of the motor encoders. Odometry information from motors is used in combination with vision and other sensors to judge/correct the current position of the gripper.

Limitations:

- Vision systems can be limited by the illumination of the environment. Small deviations in the lighting conditions can drastically change the pixel values and therefore create big problems for the position estimation algorithm that operates on those pixels.
- Another limitation of the vision system is that it is hard to estimate depth information using only monocular data. One way would be to apply the parallax principle by moving the camera. Of course the better way would be to use multiple cameras and create a stereo system for depth extraction.
- A third limitation is that it is quite hard to segment certain object pixels from background pixels. If the vision algorithm gets thrown off by an ambiguous scene and wrongly segments the object, the positioning of the gripper will be wrong as well.

Task 3.5 (5 points) Singularities:

There are two main types of singularities:

- Workspace boundary singularities
- Workspace internal singularities

Workspace boundary singularity:

1. An example of a **workspace boundary singularity** is when the length of a robot arm is simply not long enough to reach a certain point in space.
2. Another example is when a joint can not rotate beyond a certain angle, therefore not alowing the robot agent to reach a point that is reachable only via a bigger rotation angle than what the joint permits.

Workspace internal singularity:

1. An example of a **workspace internal singularity** is when a gripper joint for example can rotate around all 3 axes: yaw, pith and roll and 1 or 2 axes get aligned with each other resulting in the loss of one or two degrees of freedom respectively (i.e. Gimbal lock).
2. Workspace internal singularities occur when at least two rotation axes are aligned. Imagine a fully extended planar 2 DOF arm. In this configuration the arm looses a degree of freedom, as movement of the arm results in the same motion direction (see fig. 4). Therefore the TCP velocity in the direction of x cannot be different from 0 and therefore $det(J) = 0$.

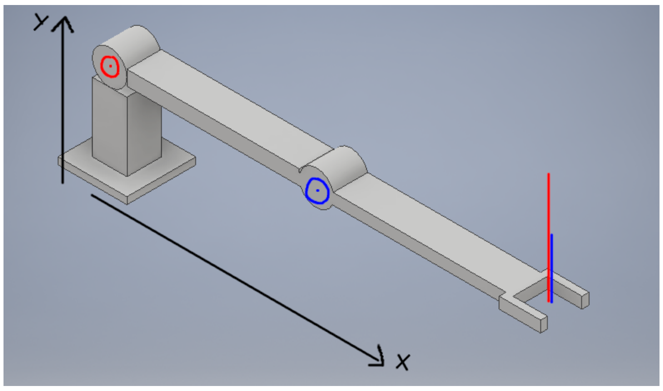

Figure 4: Example for a workspace internal singular configuration

A singularity appears whenever the Jacobian is not invertible, i.e. does not have a full rank, which effectively means that one or more degrees of freedom are lost. A collision on the other hand, does not imply that DOFs are lost. It just means that although the Jacobian allows a certain theoretical trajectory, in practice it is not possible to execute it because of physical constraints like collision of 2 parts of an arm, collision of 2 arms, etc. In other words, the arms would be able to continue the trajectory if only they would not collide with some external part (of the robot itself).

Task 3.6 (4 points) PUMA 560 bonus Task:

To solve this task we make use of the inverse kinematics algebraic solution for the PUMA 560 manipulator, provided in lecture 3. The first part is to find the 3 angles that influence the positioning of the gripper. The next step is to find the 3 orientation angles of the gripper, which are effectively Euler Angles extracted from the final rotation matrix. In order to compute these angles, we will make use of the physical dimensions of the robot from figure 5 and superpose it with figure 6 to get the DH parameters:

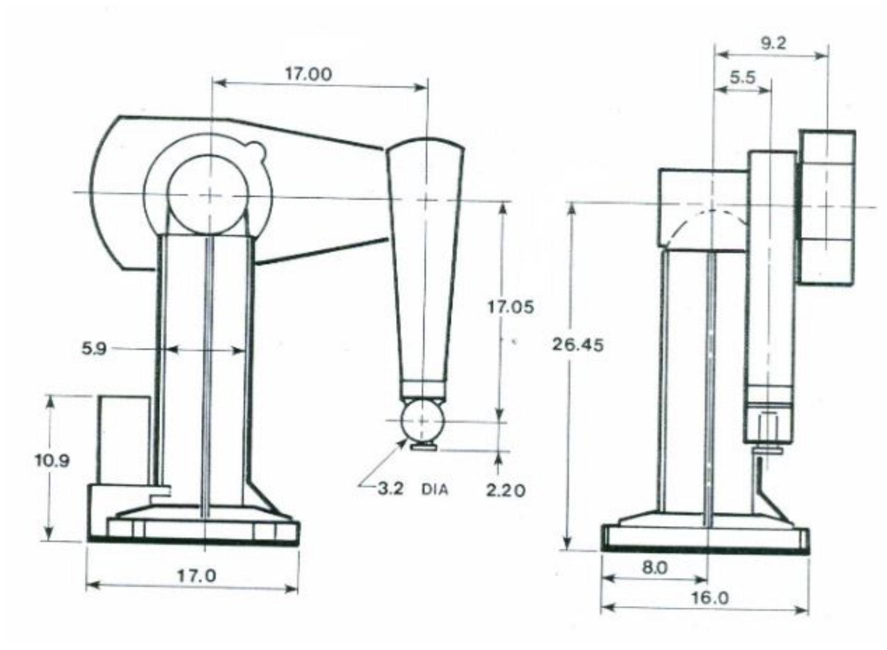

Figure 5: PUMA 560 dimensions

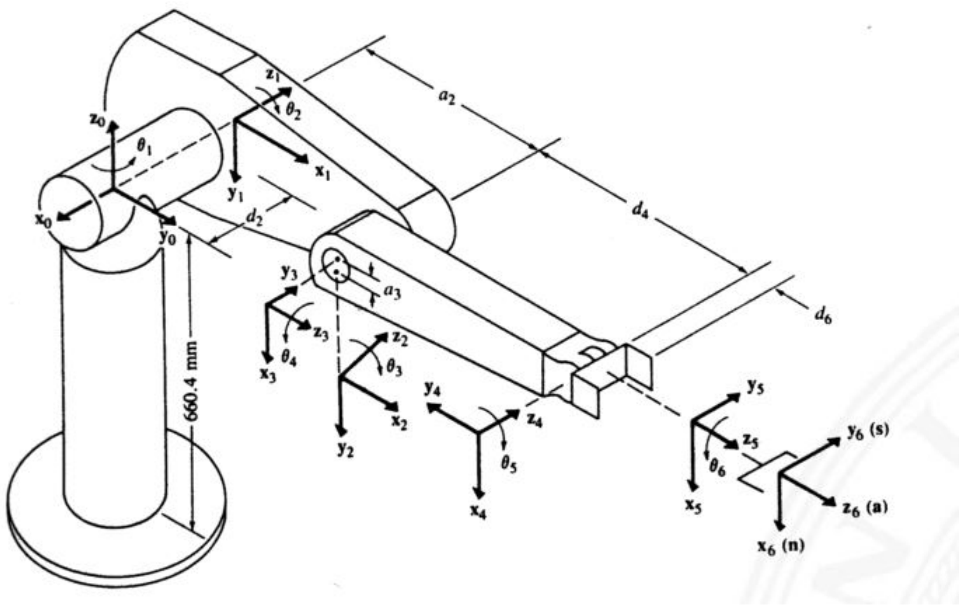

Figure 6: PUMA 560

Results (the exact formulas can be seen in the code appendix below):

$\theta_1 = -10°$, $\theta_2 = 7.79°$, $\theta_3 = -67.89°$, $\theta_4 = -90°$, $\theta_5 = 0°$, $\theta_6 = -180°$

Code Appendix

```
import math

#Position values:
px = 18
py = 11
pz = 5
#Orientation values:
nx=0
ny=-1
nz=0
ax=0
ay=0
ox=-1
oy=0

#From the PUMA 560 robot datasheet we have:
ratio = 2.54 # from inches to cm
a2 = 17 * ratio
a3 = 0 * ratio # approximation
d2 = 5.5 * ratio
d4 = 17.05 * ratio

#Algebraic solution for the first 3 angles:
print("\nPositioning␣angles␣in␣degrees:\n")
theta1 = math.atan2(py*math.sqrt(px*px+py*py-d2*d2)-px*d2, px*math.sqrt(px*px+py*py-d2*d2)+py*d2)
print("theta1=", theta1*180/math.pi)

#-------------------------------------------------------------------------------------------

A3 = 2*a2*a3
B3 = 2*a2*d4
D3 = px*px + py*py + pz*pz - a2*a2 - a3*a3 - d2*d2 - d4*d4

theta3 = math.atan2(A3*math.sqrt(A3*A3+B3*B3-D3*D3) + B3*D3, B3*math.sqrt(A3*A3+B3*B3-D3*D3) + A3*D3)
print("theta3=", theta3*180/math.pi)

#-------------------------------------------------------------------------------------------

A2 = d4*math.cos(theta3) - a3*math.sin(theta3)
B2 = -a3*math.cos(theta3) - d4*math.sin(theta3) - a2

theta2 = math.atan2(B2*math.sqrt(px*px+py*py-d2*d2)+A2*pz, A2*math.sqrt(px*px+py*py-d2*d2)+B2*pz)
print("theta2=", theta2*180/math.pi)

#-------------------------------------------------------------------------------------------
#Orientation solution for the last 3 angles by extracting the Euler angles from the rotation matrix:

print("\nOrientation␣angles␣in␣degrees:\n")
fi = math.atan2(ny, nx)
print("fi=", fi*180/math.pi)

theta = math.atan2(-nz, math.cos(fi)*nx+math.sin(fi)*ay)
print("theta=", theta*180/math.pi)

psi = math.atan2(math.sin(fi)*ax-math.cos(fi)*ay, -math.sin(fi)*ox+math.cos(fi)*oy)
print("psi=", psi*180/math.pi)
```