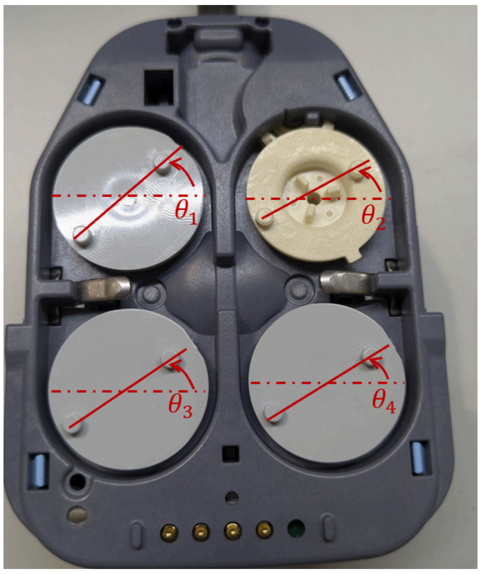
Kinematics of the da Vinci large needle driver

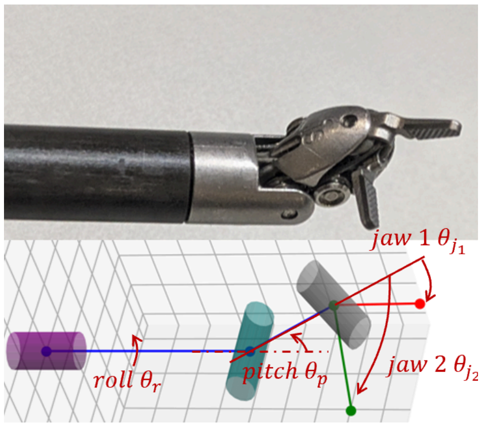
I Objective

This document aims to develop a kinematic model for the da Vinci Large Needle Driver (Model No. 420006). The proposed model will facilitate the establishment of both forward and inverse kinematics principles, which are essential for designing a control system capable of operating the device.

II Kinematic model



a) Input conventions.



b) Kinematic model of the end effector "output" angles

Figure 1: Kinematic model for the daVinci large needle driver.

Figure 1 represent the angle naming conventions of the proposed model. To account for the mechanical stops at the tip, the following "output" angles boundary conditions are proposed:

$$-110° \leq \theta_{j_1} \leq 110°$$

$$-110° \leq \theta_{j_2} \leq 110°$$

$$\theta_{j_2} \geq \theta_{j_1}$$

III Forward kinematics

III.1 V2: Simplified: non linearities handle by border conditions

Observation The non-linearities that occur at the stops can be seen as modifying the available domains for the inputs. This allows the use of a linear invertible domain with non constant borders. This way, all non linearities caused by collisions and stops can be handled at the feasability check, while the behaviour at a stop can be described as changing the boundaries of other inputs.

Note on simultaneous movements As seen on Figure 3, the domain of accepted inputs is convex. Since simultaneous movements of the output, in that domain, can be seen as a straight line (linear combination), this movement will stay inside the domain by convexity. Thus, provided the movement is synchronized, no stops should be hit unwillingly.

III.1.a Geometric relationships

$$\theta_r = \frac{270}{170}\theta_2$$
$$\theta_p = -\frac{80}{80}\theta_1$$
$$\theta_{j_1} = \frac{110}{90}\left(\theta_3 - \frac{70}{80}\theta_1\right)$$
$$\theta_{j_2} = \frac{110}{90}\left(\theta_4 - \frac{70}{80}\theta_1\right)$$

with and aditionnal boundary conditions, on the input:

$$\theta_3 > \theta_4$$
$$\theta_1 \text{ in } [-80 - \Delta_b, 80 + \Delta_b] \text{ where } \Delta_b = \begin{pmatrix} \frac{80}{60}(\theta_3 + 90°) \text{ if } \ \theta_3 < -90° \\ \frac{80}{60}(\theta_4 - 90°) \text{ if } \ \theta_4 > 90° \end{pmatrix}$$

and on the output:

$$\theta_r = \frac{270}{170}\theta_2$$
$$\theta_p = -\frac{80}{80}\theta_1$$
$$\theta_{j_1} = \frac{110}{90}\left(\theta_3 - \frac{70}{80}\theta_1\right)$$
$$\theta_{j_2} = \frac{110}{90}\left(\theta_4 - \frac{70}{80}\theta_1\right)$$

III.1.b python code:

Following is an example python code handling the forward kinematics.

```python
roll = 270/170*theta_2

pitch = - 80/80 * theta_1

jaw1 = 110/90*(theta_3 - 70/80*theta_1)
jaw2 = 110/90*(theta_4 - 70/80*theta_1)


...
conditions_output = { # 'message' : bool not in bound
    "$\\theta_p$ not in $[-80, 80]$": np.degrees(pitch) < -80 or np.degrees(pitch) > 80,
    # "$\\theta_r$ not in $[-270, 270]$": np.degrees(roll) < -270 or np.degrees(roll) > 270,
    "$\\theta_{j1}$ not in $[-110, 110]$": np.degrees(jaw1) < -110 or np.degrees(jaw1) > 110,
    "$\\theta_{j2}$ not in $[-110, 110]$": np.degrees(jaw2) < -110 or np.degrees(jaw2) > 110,
    "Should not have $\\theta_{j1} < \\theta_{j2}$": jaw2 < jaw1,
}
delta_boundary = 0
if theta_3 < theta_4 < -90:
    delta_boundary += 80/60*(theta_3 + np.radians(90))
elif 90 < theta_3 < theta_4:
    delta_boundary += 80/60*(theta_4 - np.radians(90))
conditions_input = {
    "Should not have $\\theta_4 < \\theta_3$": theta_3 > theta_4,
    f"$\\theta_1$ not in $[-{80 + delta_boundary}, {80 + delta_boundary}] (dynamic
boundaries)$": theta_1 < -np.radians(80) + delta_boundary or theta_1 > np.radians(80) +
delta_boundary,
}
true_keys = [key for key, value in conditions_output.items() if value] + [key for key, value in
conditions_input.items() if value]
```

III.2 Limit values

Thanks to the results represented on Figure 2, the following extreme input values were found:

- $\theta_1 : [-78.5°, 78.5°]$
- $\theta_2 : [-170°, 170°]$
- $\theta_3 : [-152.3°, 152.3°]$
- $\theta_4 : [-152.3°, 152.3°]$

These angles should be taken into account for choosing the control system's actuators.

IV Inverse kinematics

With the simplified model, the inverse kinematics is linear and can be solved with a matrix inversion. The non-linearities are handled by the border conditions.

IV.1 Geometric relationships

$$-\theta_2 = \frac{170}{270}\theta_r$$

$$\begin{pmatrix} \theta_p \\ \theta_{j_1} \\ \theta_{j_2} \end{pmatrix} = \begin{pmatrix} -\frac{80}{80} & 0 & 0 \\ -\frac{70}{80} * \frac{110}{90} & \frac{110}{90} & 0 \\ -\frac{70}{80} * \frac{110}{90} & 0 & \frac{110}{90} \end{pmatrix}^{-1} \begin{pmatrix} \theta_1 \\ \theta_3 \\ \theta_4 \end{pmatrix} = \frac{1}{88} \begin{pmatrix} -80 & 0 & 0 \\ -77 & 72 & 0 \\ -77 & 0 & 72 \end{pmatrix} \begin{pmatrix} \theta_1 \\ \theta_3 \\ \theta_4 \end{pmatrix}$$

with the same border conditions described in Section III.1.a.

IV.2 python code

Following is an example python code handling the backward kinematics.

```
theta_2 = 170/270*roll

out_vec3 = np.array([pitch, jaw1, jaw2])
rel_matrix = np.array([
    [-80/80       , 0       , 0     ],
    [-70/80*110/90, 110/90  , 0     ],
    [-70/80*110/90, 0       , 110/90],
])
in_vec3 = np.linalg.solve(rel_matrix, out_vec3)
theta_1, theta_3, theta_4 = in_vec3[0], in_vec3[1], in_vec3[2]
```

V Domains

The following plots reprensent domains defined by the boundary conditions. These were computed by sampling the 3D space (input excluding θ_2 or output θ_r respectively).

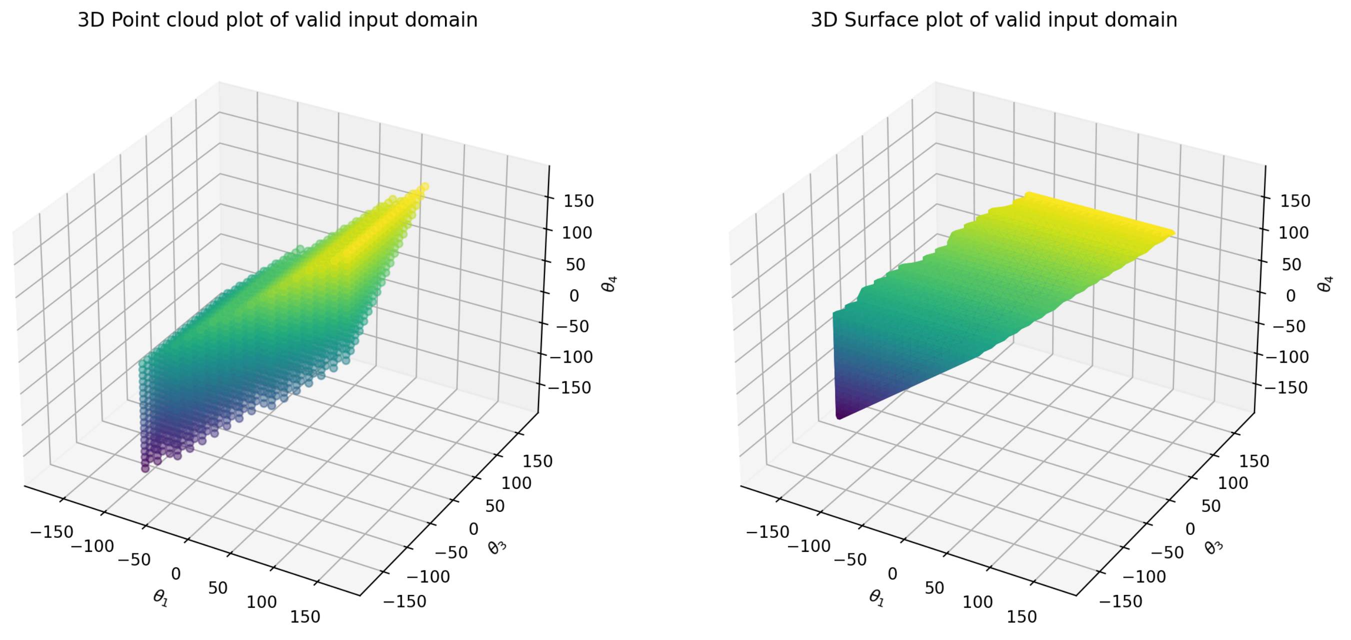


Figure 2: Plot of the available input domain. Left: scatter points, Right: surface plot.

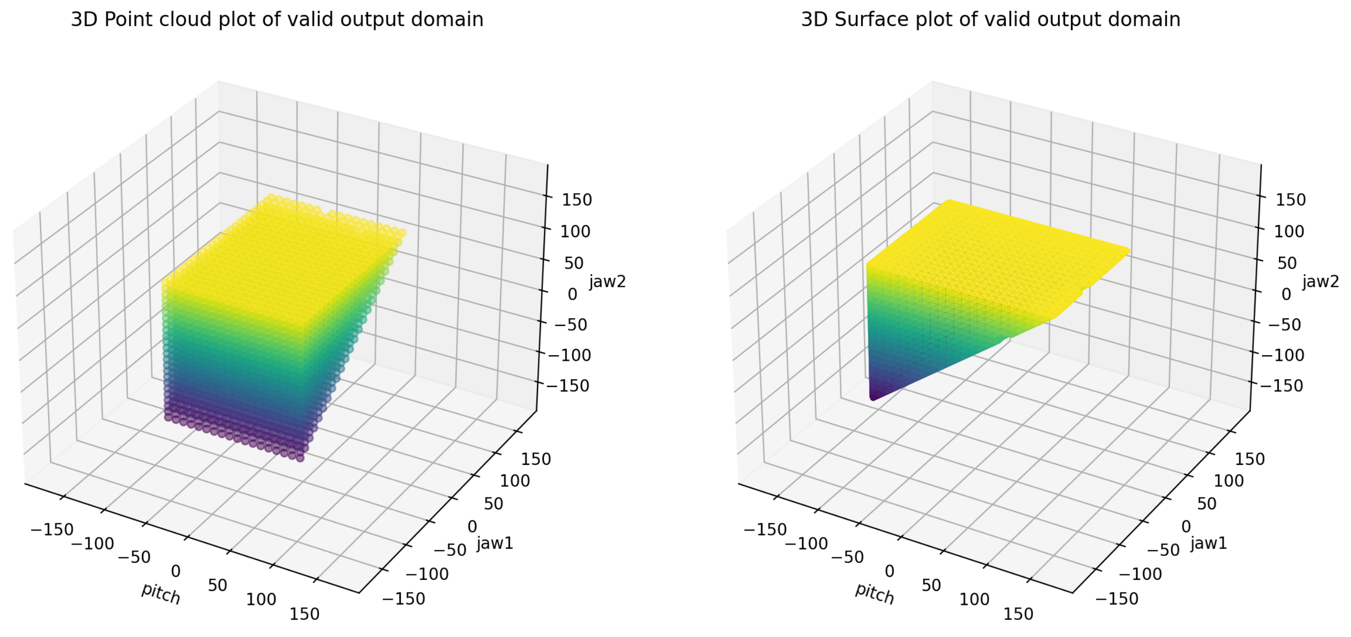


Figure 3: Plot of the available "output" domain. Left: scatter points, Right: surface plot.

This step allowed identifying the limit input values described in Section III.2.

VI Offset consideration

Offset: $\delta\theta_i$ such that at angle $\delta\theta_i$ the position is neutral. Just replace `self.theta_i` by `self.theta_i - delta theta_i` in the equations.