Industrial SCARA Robot Adept

Robotics Practicals
Group 29

Autors:
Xavier BAILLY
Philipp SPIESS
Marc URAN

1 April 2019

Contents

1 Preface

In a first time we need to express in a cartesian reference (x, y, z) the coordinates of the *Adept Cobra s350s* robot (J_1, J_2, J_3, J_4) in which the controler executes the motion (cf. Figure 1).

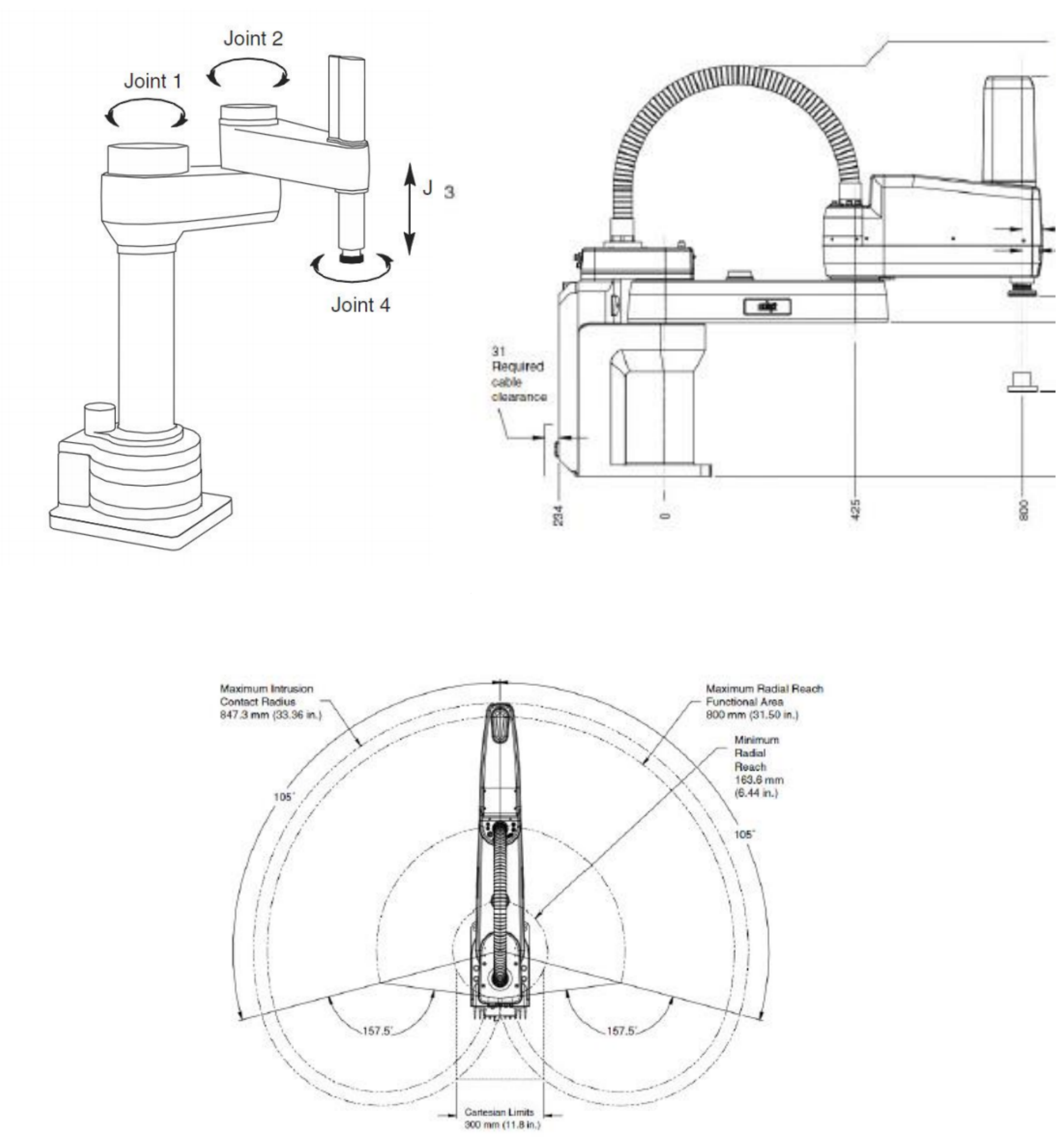
Joint 2
Joint 1
J 3
Joint 4
31 Required cable clearance
234
0
425
800
Maximum Intrusion Contact Radius 847.3 mm (33.36 in.)
Maximum Radial Reach Functional Area 800 mm (31.50 in.)
Minimum Radial Reach 163.6 mm (6.44 in.)
105°
105°
157.5°
157.5°
Cartesian Limits 300 mm (11.8 in.)

Figure 1: Kinematics illustration of the *Adept Cobra* robot

With the following notation :

$$\begin{cases} c_1 = cos(J_1) \\ s_1 = sin(J_1) \\ c_2 = cos(J_2) \\ s_2 = sin(J_2) \\ c_{12} = cos(J_1 + J_2) \\ s_{12} = sin(J_1 + J_2) \end{cases}$$

h_2 is the height of Joint 2.

The kinematics model of the *Adept Cobra* robot is given below:

- Joint 1 transformation matrix:

$$A_1(J_1) = \begin{bmatrix} c_1 & -s_1 & 0 & 0 \\ s_1 & c_1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1}$$

- Joint 2 transformation matrix:

$$A_2(J_2) = \begin{bmatrix} c_2 & -s_2 & 0 & l_1(1-c_2) \\ s_2 & c_2 & 0 & -l_1 s_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2}$$

- Joint 3 transformation matrix:

$$A_3(J_3) = \begin{bmatrix} 1 & 0 & 0 & (l_1+l_2)(1-c_4) \\ 0 & 1 & 0 & -(l_1+l_2)s_4 \\ 0 & 0 & 1 & h_2 - J_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3}$$

- Joint 4 transformation matrix:

$$A_4(J_4) = \begin{bmatrix} c_4 & -s_4 & 0 & (l_1+l_2)(1-c_4) \\ s_4 & c_4 & 0 & -(l_1+l_2)s_4 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4}$$

The complete transformation matrix of the robot is finally given by:

$$T = A_1(J_1)A_2(J_2)A_3(J_3)A_4(J_4)$$

And so, with (l_1 and l_2 being respectively the lenghts of arms 1 and 2):

$$\begin{cases} k_x = l_1c_1 + l_2c_{12} - (l_1+l_2)c_{124} \\ k_y = l_1s_1 + l_2s_{12} - (l_1+l_2)s_{124} \\ k_z = h_2 - J_3 \end{cases}$$

We get the homogeneous transformation matrix:

$$T = \begin{bmatrix} c_{124} & -s_{124} & 0 & k_x \\ s_{124} & c_{124} & 0 & k_y \\ 0 & 0 & 1 & k_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5}$$

By moving the point at the initial position of ($x_0 = l_1 + l_2, y_0 = 0, z_0 = 0$) the coordinates are:

$$\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{bmatrix} c_{124} & -s_{124} & 0 & k_x \\ s_{124} & c_{124} & 0 & k_y \\ 0 & 0 & 1 & k_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} l_1+l_2 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} l_1c_1 + l_2c_{12} \\ l_1s_1 + l_2s_{12} \\ h_2 - J_3 \\ 1 \end{pmatrix} \tag{6}$$

We can finally establish the inverse kinematics model:

$$\begin{pmatrix} J_1 \\ J_2 \\ J_3 \end{pmatrix} = \begin{pmatrix} tan^{-1}(\frac{s_1}{c_1}) \\ tan^{-1}(\frac{s_2}{c_2}) \\ h_2 - z \end{pmatrix} = \begin{pmatrix} tan^{-1}(\frac{\pm x l_2 s_2 + y(l_1 + l_2c_2)}{x(l_1+l_2c_2) \mp y(l_2s_2)}) \\ \mp tan^{-1}(\frac{s_2}{c_2}) \\ h_2 - z \end{pmatrix} \tag{7}$$

With $c_2 = \frac{1}{2l_1l_2}(x^2 + y^2 - l_1^2 - l_2^2)$ and $s_2 = \pm\sqrt{1 - c_2^2}$

2 Tic tac toe game

2.1 Preliminary statements

We had to make the robot execute the displacement of some pawns on a grid for a tic tac toe game, according to the following **rules**:

- They are 2 players playing one at time
- Each player has 5 pawns
- The grid is composed of 9 circles
- Each circle can host 1 pawn
- The aim of the game for each player is to align 3 of their pawns
- Each player can put into the computer the number of the grid circle where he wants to place his pawn (each circle has a specific number)

The first step of the programming was to define a specific **frame**, in order to give to the robot the space values required for its displacement (cf. Figure ...):

- We set the origin our frame on the player 1's first pawn
- Our x axis starts at the origin and goes in the direction of player 1's aligned pawns
- Our y axis starts at the origin and goes in the direction of player 2's first pawn
- Our z axis starts at the origin and goes up in the perpendicular direction of plan xy

Then, the specifications of our **game world** for the robot's motion are (cf. Figure ...):

- The 5 pawns of each player are aligned along the x axis in their respective storage area. Each of them are spaced out of $35mm$
- The 9 circles of the grid are spaced out of $50mm$, along the x and y axis
- The 2 player's pawn storage areas are spaced out of $73.5mm$ along the y axis with the closest x axis line of the grid
- As a consequence, the 2 pawn storage areas are spaced out of $247mm$

2.2 Motion algorithm

In order to command the robot in the simplest possible way, we defined the sequence of instruction for the motion as follows:

- We make sure that the grid is placed in the workspace of the robot.
- We define an offset along the z axis to avoid friction with the floor and collision with other pawns during the motion.
- Assuming the initial position of the robot is random, we ask it to go to the player 1's first pawn location: the origin of our frame.
- The extremity of the robot has then to move down and catch the pawn with the magnet that needs to be turned on.

- We make the extremity of the robot move back up with the pawn and then move to the desired grid location.
- The extremity of the robot is moved down one more time and the magnet is turned off to deposit the pawn on the circle.
- The extremity of the robot finally moves up one last time and the process is repeated for player 2's first pawn, then for player 1's second pawn, etc until victory or draw.

2.3 Robot programming

The language used for programming the SCARA robot is the *V+* language.

The first step of the program was to store in memory all the possible positions for the pawns, on the storage areas and the grid. We start by setting the origin of the grid (bottom left circle $c_1(x_{g,1}, y_{g,1})$) and of the first (reference of the frame $s_{1,1}(x_{p,1}, y_{p,1})$) and the second ($s_{2,1}(x_{p,1}, y_{p,2})$ same x as the frame reference but with an offset along y) storage area.

Knowing the relations between the different pawn locations, we can fill a 2D-table for the grid (x and y positions of the circles) and 2 1D-tables for the storage areas (the y position is constant for each one of them) as shown in Tables 1, 2 3 respectively.

	$y_{g,1}$	$y_{g,2}$	$y_{g,3}$
$x_{g,1}$	c_1	c_2	c_3
$x_{g,2}$	c_4	c_5	c_6
$x_{g,3}$	c_7	c_8	c_9

Table 1: Positions on the grid

	$y_{p,1}$
$x_{p,1}$	$s_{1,1}$
$x_{p,2}$	$s_{1,2}$
$x_{p,3}$	$s_{1,3}$
$x_{p,4}$	$s_{1,4}$
$x_{p,5}$	$s_{1,5}$

Table 2: Positions on storage area 1

	$y_{p,2}$
$x_{p,1}$	$s_{2,1}$
$x_{p,2}$	$s_{2,2}$
$x_{p,3}$	$s_{2,3}$
$x_{p,4}$	$s_{2,4}$
$x_{p,5}$	$s_{2,5}$

Table 3: Positions on storage area 2

Once the game place well defined, we implemented the function *go_and_pic(initial, final)* that executes the **motion algorithm** from an initial position (in one of the storage areas) to a final one (on the grid).

The offset along the z axis is defined with the *DEPART* command, setting the extremity of the robot $20mm$ over the grid. The magnet is activated by giving a positive value $SIGNAL$ 97 and turned off by giving a negative value $SIGNAL$ -97 to the signal command (97 is the number of the magnet)

To make sure that the robot will not interpolate the different spatial positions composing its trajectory, we imposed a *BREAK* command that will stop the robot a few time between two translations.

3 Grid storage

The last task was to implement the function *sort()* that needs to store the grid after the game is over. The algorithm used to perform it operates as follows:

- Each grid circle is covered by the robot, according to the order : $c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9$
- A grid position can have 3 states : occupied by a player 1 pawn (*grid_ value[i] = 1*), occupied by a player 2 pawn (*grid_ value[i] = -1*) or empty (*grid_ value[i] = 0*).
- If the circle is empty, the robots moves on the next grid location.
- If the circle is occupied by a pawn, depending on its owner, the robots has to replace it in the corresponding storage area. The program needs to fill each storage area in the same order as for the go-and-pic operation, in order to place only one pawn in each storage area location.
- Once all the grid circles have been covered, the grid is then stored and a new game can begin.

4 Conclusion

We succeeded in implementing the tic tac toe game. This was confirmed by several games we launched and the storage of the grid was also well achieved, even by increasing the speed of the robot.