Operating Principle of RobotMotion Library

CCK-Explorer*

1 Introduction

This library is written specifically for the this type of SCARA robot[1] which has one of its motor not located in its joint for stability purpose as in figure 1. Hence, this robot has its special DH parameters which will be explained in Section 2.

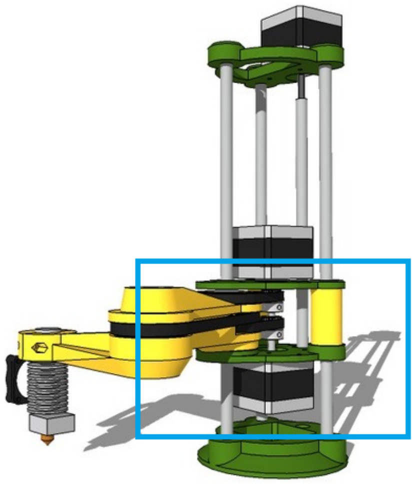


Figure 1: The blue boxed region indicates that the stepper motor responsible for link 2 (between joint 2 and 3) is not placed on the joint 2 and the rotation motion is transmitted using timing belt.

This library consists of two main functions, i.e.

i int cooordinates2step()

ii int motion(void(*speedTimeFcn)(float, float, float, float, float, float, int), void(*endEffectorFcn)(int))

As their names implied, function (i) converts the user-defined coordinates of the end-effector to the number of steps required by the stepper motors to transition from one coordinate to the other. Meanwhile, function (ii) controls the time for the stepper motor to move each step, so that a smooth motion can

*With the help of my coursework's partners in building this hardware.

[1] `https://www.thingiverse.com/thing:1241491`, thanks to Idegraaf for open sourcing it.

be executed. Both of these functions will be briefly explained in Section 3 and 4 respectively.

2 SCARA Robot Specification

Skeleton diagram in figure 2 shows the essential parameters involved in determining the frame orientation and the global position of the end-effector with respect to inertial reference frame or frame 0.

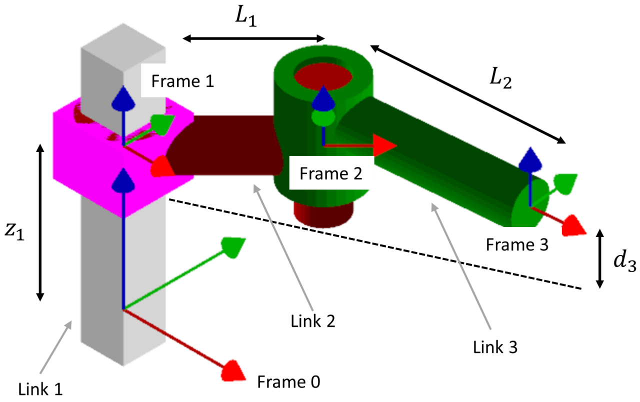


Figure 2: Skeleton diagram for determining the DH parameters.[2]

Nonetheless, since the motor controlling link 3 (arm 2) is placed along the z-axis of frame 1 but not on joint 2, then by supposing θ_1 and θ_2 represents the orientations of frames 2 and 3 with respect to frame 1 respectively, we have the DH parameters listed out in table 1.

Table 1: DH parameters of the SCARA Robot

Joint	Link	θ_i	α_i	r_i	d_i
0 - 1	1	0	0	0	z_1
1 - 2	2	θ_1	0	L_1	0
1 - 2	2	$-\theta_1$	0	0	0
2 - 3	3	θ_2	0	L_2	0

Notice that the extra third row in table 1 causes it to have special DH parameters. Then, the homogenous transformation matrix of joint 3 relative to joint 0, ${}^0T_3$ is derived, which the orientation of joint 3 depends only on θ_2 but not $\theta_1 + \theta_2$.

$$
{}^0T_3 = \begin{pmatrix} \cos\theta_2 & -\sin\theta_2 & 0 & L_1\cos\theta_1 + L_2\cos\theta_2 \\ \sin\theta_2 & \cos\theta_2 & 0 & L_1\sin\theta_1 + L_2\sin\theta_2 \\ 0 & 0 & 1 & z_1 + d_3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{1}
$$

[2] red-axis = x-axis, green-axis = y-axis, blue-axis = z-axis. Also, joints 0, 1, 2 and 3 are located at frame 0, 1, 2 and 3 respectively.
(Library's public parameters) L_1 = armLength1, L_2 = armLength2, d_3 = zOffset

3 int cooordinates2step()

Suppose the global coordinate of the end effector is (x, y, z) with respect to frame 0, then this function will convert this coordinate to the variables θ_1 , θ_2 and z_1 from Eq. 1, through inverse kinematic process, and this function can be divided into three sub functions, i.e.

i void offsetZaxis()

ii void coordinates2angle()

iii void angle2step()

Function (i) removes the offset, d_3 from z, while function (ii) converts x and y to θ_1 and θ_2.

Lastly, the differences of the next and present states of θ_1, θ_2 and z_1 will be computed by function (iii), which are then further converted to the number of steps required for each of the stepper motors to move.

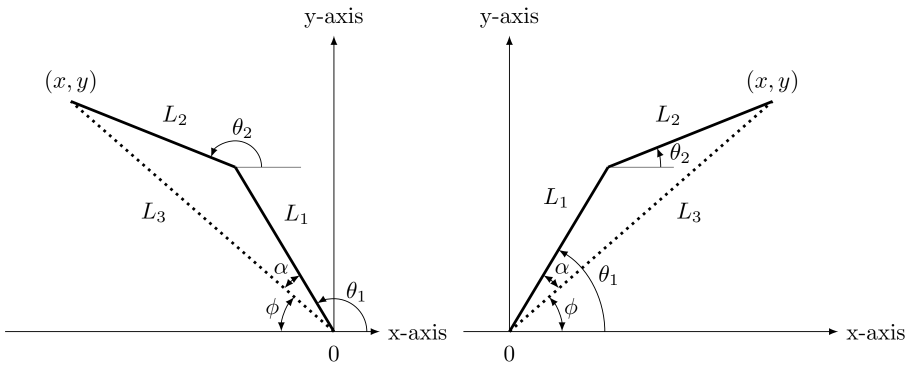
3.1 void offsetZaxis()

z_1[3] can be easily obtained through Eq. 1, which results in Eq. 2.

$$z_1 = z - d_3 \tag{2}$$

3.2 void coordinates2angle()

Meanwhile, θ_1 and θ_2[4] (both in radian) can be solved from the triangular solution in figure 3.



(a) Negative x-region

(b) Positive x-region

Figure 3: For x-y Cartesian coordinate system with respect to frame 0, the workspace of the robotic arm is restricted inside the 1^{st} and 2^{nd} quadrants.

---

[3](Library's protected parameter) z_1 = zNew

[4](Library's protected parameters) θ_1 = angleMotor1, θ_2 = angleMotor2

The following equations are deduced,

$$L_3 = \sqrt{x^2 + y^2} \tag{3}$$

$$\alpha = \cos^{-1} \frac{L_1^2 + L_3^2 - L_2^2}{2L_1L_3} \tag{4}$$

$$\phi = \left| \tan^{-1} \frac{y}{x} \right| \tag{5}$$

$$\theta_1 = \begin{cases} \phi + \alpha & x \geq 0 \\ \pi - (\phi + \alpha) & x < 0 \end{cases} \tag{6}$$

$$\theta_2 = \sin^{-1} \frac{y - \ L_1 \sin\theta_1}{L_2} \tag{7}$$

The sine function is chosen in Eq. 7 since it is positive in both first and second quadrants which simplifies the calculation.

One important sidenote, based on figure 3, only the positive y-axis is involved, hence positive values with zero inclusive must be solely used for y, else calculation errors will be arised.

3.3 void angle2step()

Before proceeding to any explanation about the calculation, it is important to understand three parameters, i.e. teeth ratio, lead and step per revolution.

Teeth ratio refers to the ratio of gear teeth of the arm and the motor. Like the robotic arm in figure 4, the gear on the arm consists of 62 gear teeth while the motor has the 20 teeth, then the ratio will be $\frac{62}{20}$. This ratio for arm 1 (link 2) and arm 2 (link 3) are represented as T_1 and T_2 [5] respectively.

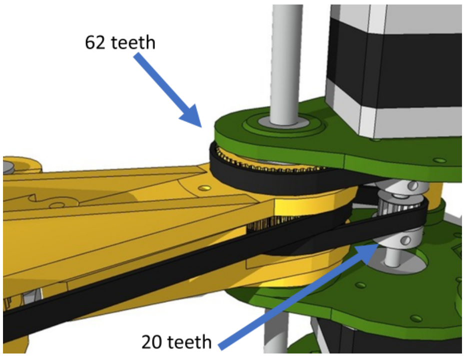


Figure 4: Locations of the gears of the robotic arm and motor

Lead refers to the screw lead. For example, if a nut can travel 0.8 unit of length when the screw performs one revolution, then lead = 0.8 [6]. Let l be the lead of the power screw in link 1 [7].

[5](Library's public parameters) T_1 = teethRatio1 T_2 = teethRatio2
[6]The length's unit must be the same as other parameters which involves the length's unit.
[7](Library's public parameter) l = lead

Step per revolution refers to the number of steps performed by the stepper motor after one revolution. For instance, if a stepper motor undergoes 400 steps to complete a revolution, then this parameter = 400. For the motors controlling link 1, 2, 3, such numbers are represented as S_z, S_1 and S_2 [8] respectively.

Suppose this convention which θ_m^n and θ_m^p represent the next and present angles respectively of link $m+1$ from previous section 3.2. While x_n and x_p represents the next and present states of x, the same applies to y and z global coordinates. Then, this function will first compute the angles and height differences as follows:

For arm 1 or link 2,

$$\Delta\theta_1 = \theta_1^n - \theta_1^p \tag{8}$$

which is bounded in $0 \leq \Delta\theta_1 \leq 2\pi$, but in fact it will be $0 \leq \Delta\theta_1 \leq \pi$ all the time since the robotic arm is only allowed to move in positive y-region with respect to the inertial frame.

For arm 2 or link 3, the difference is first computed,

$$\Delta\theta_2' = \theta_2^n - \theta_2^p \tag{9}$$

which is also bounded in $0 \leq \Delta\theta_2' \leq 2\pi$. Then to avoid any collision between links 2 and 3, an extra operation is implemented as follows:

$$\Delta\theta_2 = \begin{cases} \Delta\theta_2' + 2\pi & x_n < 0 \cap x_p > 0 \cap \Delta\theta_2' < 0 \\ \Delta\theta_2' - 2\pi & x_n \geq 0 \cap x_p < 0 \cap \Delta\theta_2' > 0 \end{cases} \tag{10}$$

For the height difference of next and present states for link 1,

$$\Delta z = z_n - z_p \tag{11}$$

Lastly, the total number of steps, n_t transistioning from one coordinate to the other for motors controlling link 1, link 2 (arm 1), and link 3 (arm 2) are represented in n_{tz}, n_{t1} and n_{t2} [9] with their rotational directions, R stored in R_z, R_1 and R_2 respectively [10].

$$n_{tz} = \frac{|\Delta z|}{l}\ S_z \tag{12}$$

$$n_{t1} = \frac{|\Delta\theta_1|}{2\pi}\ T_1 S_1 \tag{13}$$

$$n_{t2} = \frac{|\Delta\theta_2|}{2\pi}\ T_2 S_2 \tag{14}$$

The value of rotation direction is assigned to HIGH for clockwise and LOW for anticlockwise motions.

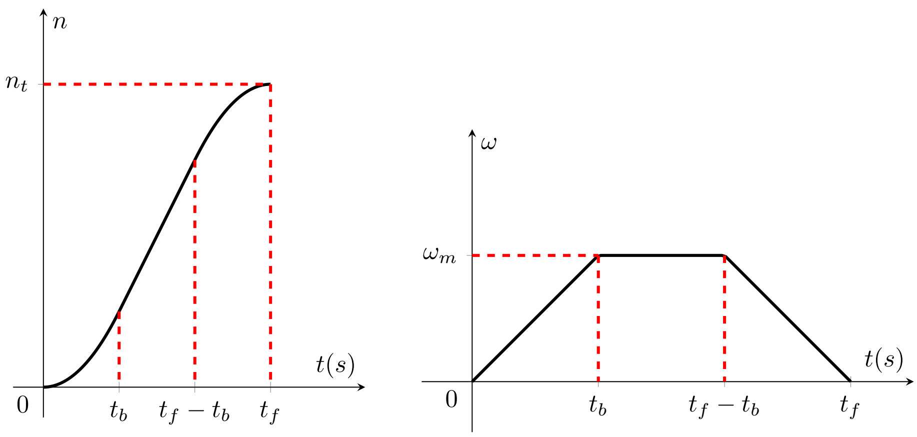
[8] (Library's public parameters) S_1 = stepPerRev1, S_2 = stepPerRev2, S_z = stepPerRevZaxis

[9] (Library's protected parameters) n_{t1} = step1, n_{t2} = step2, n_{tz} = stepZaxis

[10] (Library's protected parameters) R_1 = direction1, R_2 = direction1, R_z = directionZaxis

4 int motion(void(*speedTimeFcn)(float, ...), void(*endEffectorFcn)(int))

Let us conceive a situation that a robotic arm is moving from a constant high speed to a complete halt, a sudden large change of momentum causes the robotic arm failing to stop and continuing its motion, leading to the inaccuracy in the translation of the end effector from this point onwards. This problem can be addressed by varying the speed of the arms gracefully and the implemented solution here is using trapezoidal velocity profile.



(a) Position (step) profile

(b) Velocity (step per second) profile

Figure 5: Each motor will undergo these profiles to complete the angles differences computed in Section 3.3 from one coordinate to the other.[11]

This velocity profile will result in a linear segment with parabolic blends position profile, which can be represented by Eq. 15 and Eq 16 respectively.

$$n = \begin{cases} \frac{\omega_m}{2t_b} t^2 & 0 \le t \le t_b \\ \omega_m t - \frac{\omega_m t_b}{2} & t_b \le t \le t_f - t_b \\ n_t - \frac{\omega_m}{2t_b} \, (t - t_f)^2 & t_f - t_b \le t \le t_f \end{cases} \tag{15}$$

$$\omega = \begin{cases} \frac{\omega_m}{t_b} t & 0 \le t \le t_b \\ \omega_m & t_b \le t \le t_f - t_b \\ -\frac{\omega_m}{t_b}(t - t_f) & t_f - t_b \le t \le t_f \end{cases} \tag{16}$$

The blending time, t_b in Eq. 17 is bounded between 0 and $\frac{t_f}{2}$, which causes ω_m always have this condition in Eq. 18.

$$t_b = t_f - \frac{n_t}{\omega_m} \tag{17}$$

$$\frac{n_t}{t_f} < \omega_m \le \frac{2n_t}{t_f} \tag{18}$$

[11] t = time (second), t_b = blending time, t_f = the final time to arrive the desired angle, n = the number of steps executed, ω = number of steps / second, n_t = the total number of steps as described in Section 3.3, ω_m = maximum ω

Hence, users are required to input a list of times, t_f to reach a list of coordinates from its previous stopping point together with a list of their corresponding maximum cruising speeds, ω_m [12]. Besides, the maximum permissible rotational speed limit, ω_p of the stepper motors also need to be specified [13].

This library will first adjust the suitable ω_m according to t_f. If ω_m is within the limits stated in Eq. 18, then it will utilize the user defined ω_m. Else, it will either use the lower or upper bounds of Eq. 18. However, if ω_m chosen by this library is more than ω_p, then ω_m is first assigned to the value of ω_p, next t_f is redetermined using Eq. 19, which its value will definitely more than the user's assigned t_f.

$$t_f = \frac{2n_t}{\omega_m} \tag{19}$$

With these t_f and ω_m, t_b will then be calculated through Eq. 17. The control of the motor speed is achieved by sending square wave signals to the IO pin [14] based on different time intervals, Δt from the position profile in Fig 5a [15].

Nevertheless, finding Δt from Eq. 15 will involve square root function, causing inefficient computations for Arduino Uno or Mega, and hence the stepper motor might not able to complete the step within the expected Δt especially for very short translation time. Therefore, linear approximation is used in this library.

Since $\frac{dn}{dt} \approx \frac{\Delta n}{\Delta t}$, it can also be written as $\Delta t \approx \Delta n \left(\frac{dn}{dt}\right)^{-1}$. Using $\frac{dn}{dt}$ or ω from Eq. 16 and setting $\Delta n = 1$, we have

$$\Delta t = \begin{cases} \frac{t_b}{\omega_m t} & 0 \le t \le t_b \\ \frac{1}{\omega_m} & t_b \le t \le t_f - t_b \\ \frac{t_b}{\omega_m (t_f - t)} & t_f - t_b \le t \le t_f \end{cases} \tag{20}$$

which Δt represents the time to send a high pulse after the last high pulse.

Nevertheless, a cycle of square wave must contain a high and a low, hence a low pulse must be sent as well. Then Δt in Eq. 20 must be halved again, producing $\Delta t'$,

$$\Delta t' = \begin{cases} \frac{t_b}{2\omega_m t} & 0 \le t \le t_b \\ \frac{1}{2\omega_m} & t_b \le t \le t_f - t_b \\ \frac{t_b}{2\omega_m (t_f - t)} & t_f - t_b \le t \le t_f \end{cases} \tag{21}$$

which is the time interval to send high pulse and low pulse alternatively.

The first value of t is required to be initialized. From the first function of Eq. 15 and setting $n = \Delta n = \frac{1}{2}$, the initial value of t is $\sqrt{\frac{t_b}{\omega_m}}$. Then, subsequents t are incremented with $\Delta t'$ after sending each high or low pulse alternatively. This whole process continues until n reaches n_t and it is summarized in pseudocode 1[16].

[12](Library's public parameters) t_f = time (struct), ω_m = maxCruisingSpeed (struct)

[13](Library's public parameters) ω_p = limitSpeed_vZaxis (link 1), limitSpeed_v1 (link 2), limitSpeed_v2 (link 3)

[14](Library's public parameters) IO pins = stepperPin_pZaxis (link 1), stepperPin_p1 (link 2), stepperPin_p2(link 3)

[15]Suitable for motor drivers such as DRV8825 and A4988.

[16]This is only for one joint, refer to the .cpp code for three joints' contols.

Algorithm 1 Inner loop of int motion(...) for a single joint

```
Input: t_f > 0, ω_m > 0,                                   ▷ User-defined variables
  n_t > 0, R = HIGH/LOW                                    ▷ From Section 3.3
  speedTimeFcn, endEffectorFcn                             ▷ User-defined functors
Output: Square wave signal to motor stepping input pin
  if ω_m ≤ n_t/t_f then                                    ▷ From lower bound of Eq. 18
     ω_m ← n_t/t_f
  else if ω_m > 2n_t/t_f then                              ▷ From upper bound of Eq. 18
     ω_m ← 2n_t/t_f
  end if
  if ω_m > ω_p then
     ω_m ← ω_p
     calculate t_f                                         ▷ From Eq. 19
  end if
  perform speedTimeFcn
  motor directional input pin ← R
  motor stepping input pin ← LOW
  calculation ← true
  n ← 0
  t ← sqrt(t_b/ω_m)                                        ▷ Initialize t
  t_1st ← present time                                     ▷ Checking present time
  while n ≤ n_t do
     if calculation = true then
        calculate Δt'                                      ▷ From Eq. 21
        t ← t + Δt'
        calculation ← false
     end if
     t_2nd ← present time                                  ▷ Checking present time
     if t_2nd − t_1st ≥ Δt' then
        t_1st ← present time
        if motor stepping input pin = LOW then
           motor stepping input pin ← HIGH
           n ← n + 1
        else
           motor stepping input pin ← LOW
        end if
        calculation ← true
     end if
  end while
  perform endEffectorFcn
```

END