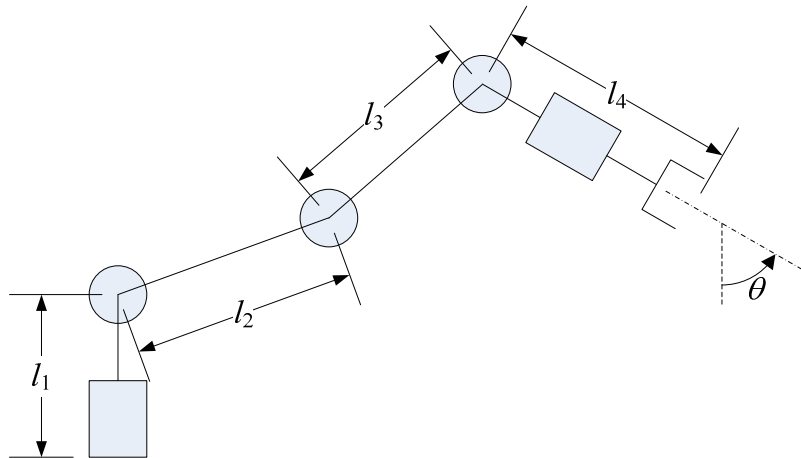


Guideline for Developing the Program for the Servo Robot Ittichote Chuckpaiwong



Writing Programs

We want to compute forward and inverse kinematics by using following parameters.

Joint coordinates

θ_1
 θ_2
 θ_3
 θ_4
 θ_5

Cartesian coordinates

x
 y
 z
 θ
 ϕ

The joint coordinates are defined $\theta_1, \theta_2, \theta_3, \theta_4, \theta_5$, where θ_i is the rotation angle of the joint i . The Cartesian coordinates are defined by x, y, z, θ , and ϕ . x, y, z define a position of end-effector (gripper) position. θ is the elevation angle of the end-effector measure from the vertical axis as shown in the figure. ϕ is the rotation angle of end effector around its own axis. Therefore the angle ϕ can be defined such that $\phi = \theta_5$.

A forward and inverse kinematics must be computed to transform parameters between the joint and Cartesian coordinates. Two computer programs may be written to perform such transformations. A forward kinematics program may be written in form of function as follows:

$$[x, y, z, \text{theta}, \text{phi}] = \text{fwdkin}(q1, q2, q3, q4, q5)$$

This function transforms joint coordinates $q1, q2, q3, q4$ and $q5$ to Cartesian coordinates x, y, z, theta and phi by computing forward kinematics.

Similarly, an inverse kinematics program may be written in form of function as follows:

$$[q1, q2, q3, q4, q5] = \text{invkin}(x, y, z, \text{theta}, \text{phi})$$

This function transforms Cartesian coordinates x, y, z, theta and phi to joint coordinates $q1, q2, q3, q4$ and $q5$ by computing inverse kinematics.

Testing the Programs

Once we have the two functions: `fwdkin()` and `invkin()`, we must check to make sure that our kinematics computations are correct. The idea of checking correctness of the computation is that forward kinematics and inverse kinematics should be inverse transformation of each other. Therefore, we can assume a set of parameters, perform forward kinematics to the set, and then perform inverse kinematics to the output of the forward kinematics. If the programs work correctly, we should yield the original parameters.

However, in the real problem, performing forward kinematics followed by inverse kinematics may not give the original result. This is because the inverse kinematics may have multiple solutions, which forward kinematics always has one solution. Therefore, it is better idea to perform forward kinematics one more time, and compare the Joint coordinates parameters with the former. The following shows the programming pseudo codes to perform the correctness test using the described idea.

```

Initialize Q1
P1 = fwdkin(Q1)
Q2 = invkin(P1)
P2 = fwdkin(Q2)
if P1 = P2
    printf("The result is good")
else
    printf("The result is bad")

```

In this program, we assume a set of joint coordinates (Q1), then perform forward kinematics to obtain corresponding Cartesian coordinates (P1). Then compute the inverse kinematics back again to get the corresponding joint coordinates (Q2). And then compute the forward kinematics one more time to get the corresponding Cartesian coordinates (P2). If our formulas are correct, the former and the later Cartesian coordinates should be identical (P1 = P2). We can test this with multiple sets of coordinates to make sure that we got the right formulas.

The following values are recommended to be used (as Q1) for testing the correctness of the programs.

Test set #	θ_1	θ_2	θ_3	θ_4	θ_5
1	$\pi/4$	0	$\pi/4$	$\pi/4$	0
2	0	$\pi/4$	$5\pi/6$	$-\pi/4$	$\pi/8$
3	0	$\pi/3$	$\pi/3-0.1$	$-\pi/4$	$\pi/8$
4	0	$\pi/3$	$\pi/3+0.1$	$-\pi/4$	$\pi/8$
5	$2\pi/3$	$\pi/3$	$\pi/3-0.1$	$-\pi/4$	$\pi/8$
6	$2\pi/3$	$\pi/3$	$\pi/3$	$-\pi/4$	$\pi/8$

Troubleshooting

If you have problem in test set #1, it is likely that you made an error somewhere in your calculations or a typo in your program.

It should not be too hard to get your program pass the test set #1. If you have problem with the test set #2, #4, can pass the test set #1, #3, #5 and #6. This is a more difficult problem. This problem can be fixed by choosing the correct sign of

$$\pm\sqrt{p_x^2 + p_y^2}$$

in the computation of θ_2 , where (p_x, p_y, p_z) is the wrist point of the robot. If you choose the positive sign, the problem will occur when the robot wrist point and end-effector are in opposite quadrant (when looking downward) as in Test set #2 and #4. Try to solve this problem by checking the wrist point and end-effector position. Make sure your kinematics program can work correctly with all the Test set given here.

Writing Program to Control Motor

Once your kinematics programs work correctly, you may start to control the robot with confidence. The robot controller receives values to command servo motor between 50 to 250 through serial port. The value 150 means the controller will send pulse width 1.5 ms to a commanded motor, which will make the servo motor rotates to its center position. I prepared a set of Matlab functions to send command to the controller. To use the controller, first you have to initialize the serial port by using command

```
>> s = serinit;
```

After the serial port is initialized. The robot can be moved around. For example, if we want to move the servo motor #3 to the position 160, use the command

```
>> sendcmd(s, 3, 160);
```

When finish using the robot, the serial port must be closed by using command

```
>> serclose(s);
```

Therefore, you must convert the value from angle into the pulse width command before using the function sendcmd().

To control the robot position in joint coordinates (say to position Q) the following pseudocode is performed.

```
cmd = rad2cmd(Q)
sendcmd(Q)
P = fwdkin(Q)
update_cartesian(P)
```

Similarly, to control the robot position in Cartesian coordinates (say to position P) the following pseudocode is performed.

```
Q = invkin(P)
cmd = rad2cmd(Q)
sendcmd(Q)
update_joint(Q)
```

Note that the functions update_cartesian() and update_joint() is for showing current values in both joint and Cartesian coordinates to user. This is necessary when programming a Graphic User Interface (GUI), in which the position in both coordinates must be displayed at all time. Please see a tutorial document on how to program Matlab GUI.

Conversion of Degree to Servo Motor Command

In order to determine relationship between the servo motor angles to the servo motor commands, we measured servo motor command at various angles as shown in the following tables:

Motor 1	Degree	-90°	-45°	0°	+45°	+90°
	Servo Command	75	120	158	205	-

Motor 2	Degree	0°	+45°	+90°	+135°	+180°
	Servo Command	82	120	156	196	-

Motor 3	Degree	-90°	-45°	0°	+45°	+90°
	Servo Command	222	187	152	118	78

Motor 4	Degree	0°	+45°	+90°	+135°	+180°
	Servo Command	-	186	153	120	81

Motor 5	Degree	-90°	-45°	0°	45°	90°
	Servo Command	68	109	150	193	-

From this table, we can create a linear function to compute corresponding command for a given angle, by using linear regression. The formula for each motor should be in the form.

$$\text{degree} = a \times (\text{command}) + b$$

Note that each motor is set to have minimum and maximum command as shown in the following table.

Channel	Motor No.	Min	Max
1	1	69	232
2	2	79	217
3	3	67	236
4	4	67	222
5	5	65	230
6	6	109	191