# SCARA Robot

A SCARA (Selective Compliance Assembly Robot Arm) is a 2-link industrial robot arm used in a variety of assembly tasks. Each joint, the shoulder and elbow, are controlled by a separate motor. The tool tip moves through a sequence of commands that rotate the links connected to motor shafts. An image of an industrial SCARA robot and corresponding schematic are shown in Figure 1.

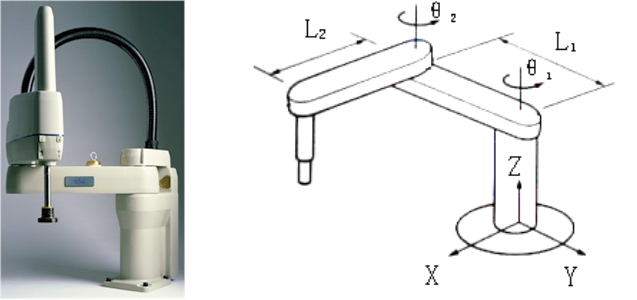


Figure 1: SCARA Robot

You can check out SCARA robots at these sites:

Epson: <https://epson.com/scara-robots>

Kuka: <https://www.kuka.com/en-ca/products/robotics-systems/industrial-robots/kr-scara-robot>

## Background

When you program a SCARA robot, you specify points in Cartesian space (**x-y** coordinates). The robot must then calculate how to position its rotary joints (using rotation angles , ) so the endpoint (tool) is at the desired **x-y** position. This is known as ***inverse kinematics***. Most robots implement these calculations as **C** code routines in the main robot controller. The robot also needs to calculate the tool position **x-y** given the joint positions , . This is known as ***forward*** ***kinematics***.

Figure 2 below is a plan view schematic of the SCARA robot arm. To compute the endpoint position, , we use a standard Cartesian (**x-y**) coordinate system with the origin at the robot base (the 'shoulder'). and are the inner and outer link lengths, and and are the shoulder and elbow joint angles ().   
**Notice that is measured from the inner link position**.

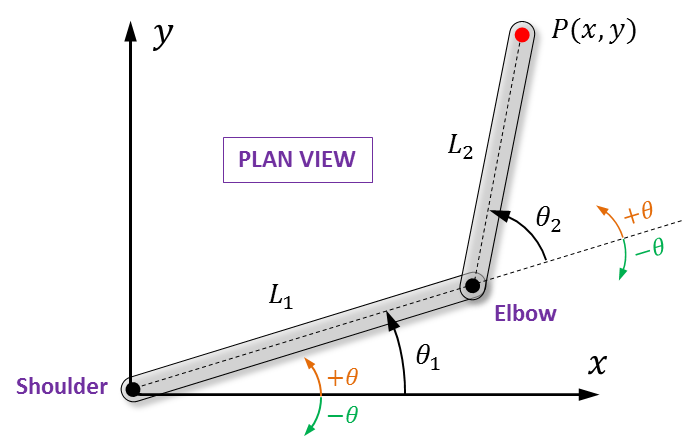


Figure 2: SCARA Robot Plan View Schematic

## Forward Kinematics Calculations

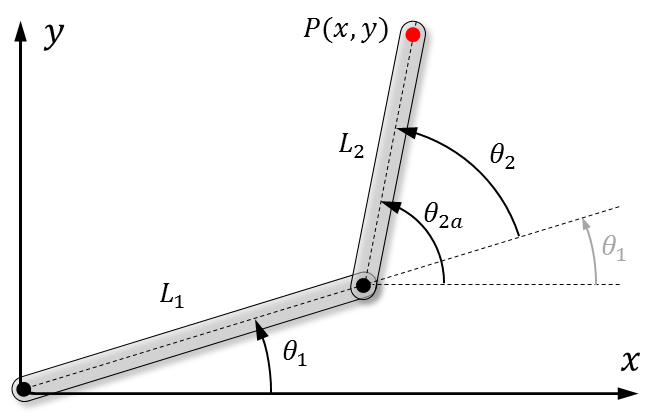


Figure 3: Forward Kinematics Diagram

When the two joint angles, and , are given, **forward kinematic** calculations are used to get the endpoint position, . From Figure 3 above,

Where is the ***absolute*** angle of the outer link. From the diagram, , therefore,

(1a)

(1b)

## Inverse Kinematics Calculations

When controlling the robot, we want to specify the tool position, not the joint angles. Since we are controlling the joint motors to move the links, we will have to use ***inverse kinematics*** to find the joint angles corresponding to the desired tool position. When the two links are not inline, there are two solutions for a given tool position. One solution is a '**right arm**' configuration (relative link positions are like your right arm) and the other is a '**left arm**' configuration. Both configurations are shown in the right-hand diagram in Figure 4 below.

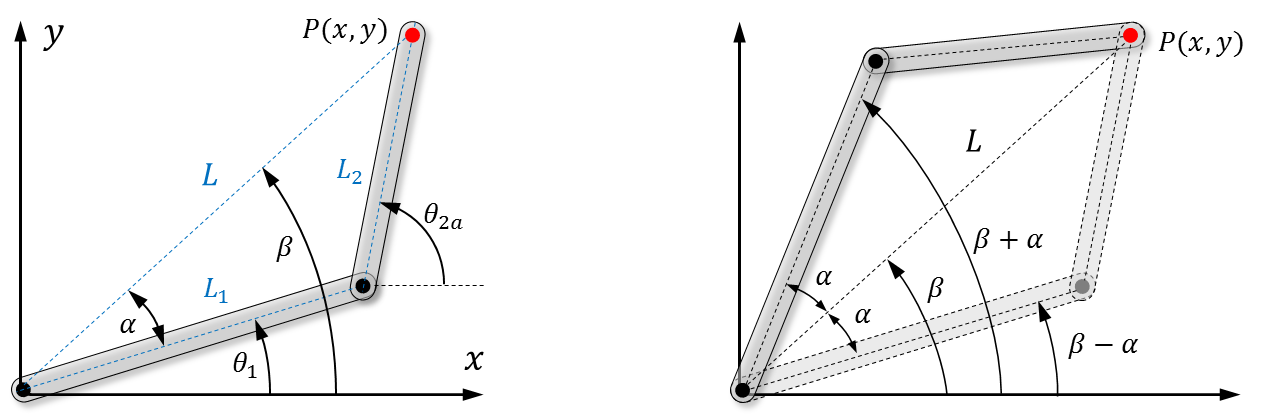


Figure 4: Inverse Kinematics Diagrams

From the diagrams above,

From the cosine law:

From the geometry:

(2a)

This gives two values for ( for a 'left arm' solution, for a 'right arm' solution). We can also determine from the geometry,

We can now calculate using the geometrical relation: ,

(2b)

**Note:** There is one for each given by equation (2a).

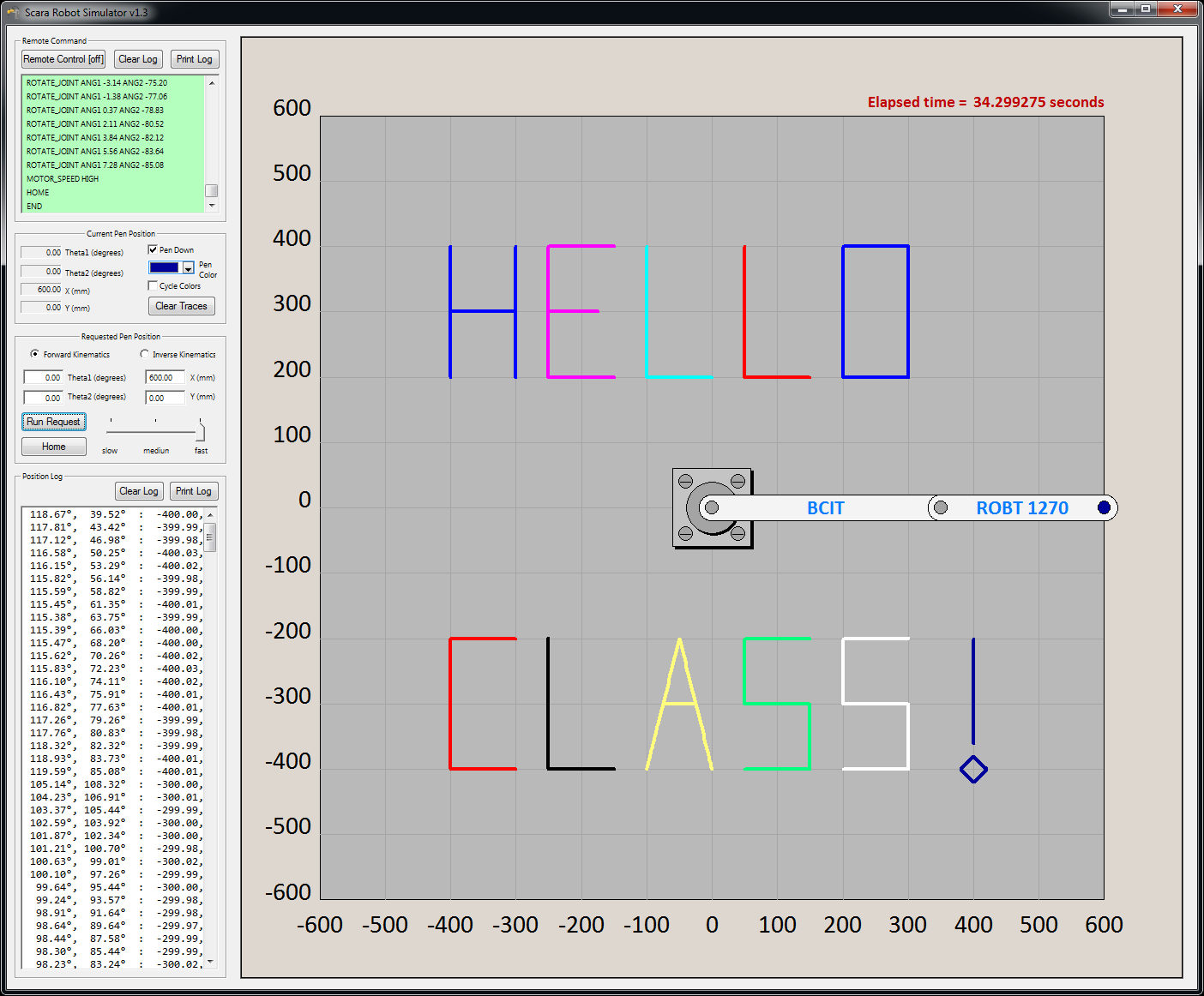
**More Notes**

In C, use atan2 for inverse tangent calculations. The standard library function atan2(y,x) returns an angle in the range which allows you to resolve any x,y coordinate into the correct one of four quadrants. Conversely, the standard library function atan(y/x), returns angles only in the first and fourth quadrants . This is because ((-y)/(-x)) has an identical value as ((+y)/(+x)), therefore atan(y/x) cannot distinguish the third quadrant from the first quadrant. Similarly, atan(y/x) cannot distinguish the second quadrant from the fourth quadrant.

**Even More Notes**

There are limitations on how large the joint angles and can be due to electrical cables housed within the robot links (otherwise the cables will become too twisted). Also, the angle between the inner and outer links is limited so that the tool does not contact the inner link. In the following section you will be introduced to a SCARA robot simulator. These angular restrictions are built into the simulator.

# SCARA Robot Simulator



A SCARA robot simulator has been developed for testing control code on a real robot. The simulated robot's tool is a pen that can be used to draw lines of assorted colors and thicknesses. The user controls the pen color and line thickness. These functions mimic industrial robots such as cake decorators or laser/pen plotters. The simulator can either be controlled manually through a control panel on the software interface or remotely through commands sent via a TCP/IP network interface. Remote commands are issued from a separate control program, such as a C program that you will write (the program can be running on a separate computer).

A screenshot of the simulator is shown in Figures 4 and 5. Figure 5 has been annotated to illustrate the different interface components. You will become familiar with controlling the robot, applying more sophisticated controls as you learn more facets of **C** programming. The simulator can be used in stand-alone manual mode using the control panel interface. Using the interface, you can position the pen by specifying *either* the two joint angles (forward kinematics) *or* the **x-y** position of the tool (inverse kinematics). When you control the simulator from remote commands sent from a control program,  
**THE ROBOT CAN ONLY BE POSITIONED BY SPECIFYING THE JOINT ANGLES**. If you want to move to a specific **x-y** position using remote commands, you need to apply inverse kinematics to first calculate the joint angles and then send those joint angles as a command to the simulator.

Notice that pen traces are not straight lines. To move from one point to another, the robot uses what is termed **joint interpolated motion**. This is done by rotating both the shoulder and elbow joints at constant (and likely different) speeds so that they finish rotating at the same time. This, in general, creates a curved pen trace. This means that if we wanted to move from point A to point B in a **straight line**, we will **not** be able to do so using a single move command. To get the tooltip to move along a desired path, we need to break up the path into a series of small movements so the trace will approximate the desired path. The smaller the movements, the better the approximation.

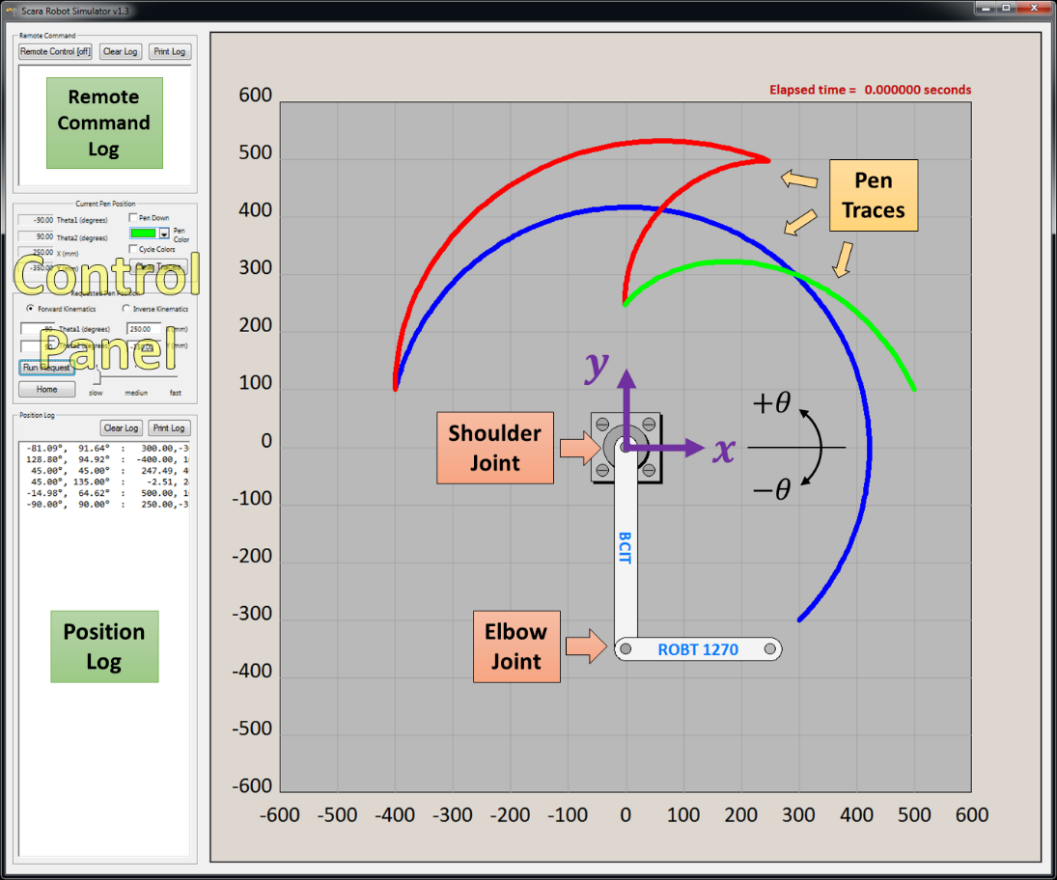


Figure 5

The lengths of the simulator's inner and outer links are =350mm and =250mm, respectively. The shoulder joint angle has a range of **±150°**, while the relative joint angle is limited to **±170°**. This implies that the two links can fold to a minimum angle of **10°**, as shown in Figure 6 below. **Note:** The outer link can be rotated clockwise or counterclockwise to the minimum angle.

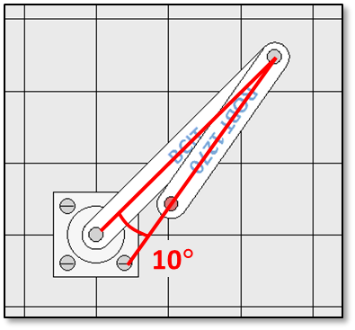


Figure 6

The full range of motion (i.e., all possible **x-y** locations of the tool) of the robot is called the *work envelope* and is given by the union of the shaded regions in Figure 7 below. Due to the restrictions on the angular rotation of each link given above, there are different work sub-envelopes for the robot in the **right arm** configuration than in the **left arm** configuration. The arm configuration is given by the sign of (see Figure 2). Positive values of put the robot in **right arm** configuration and negative values put the robot in **left arm** configuration.

For example, the robot cannot reach x,y=-300,+300 in the left arm configuration nor can it reach x,y=-300,-300 in the right arm configuration.

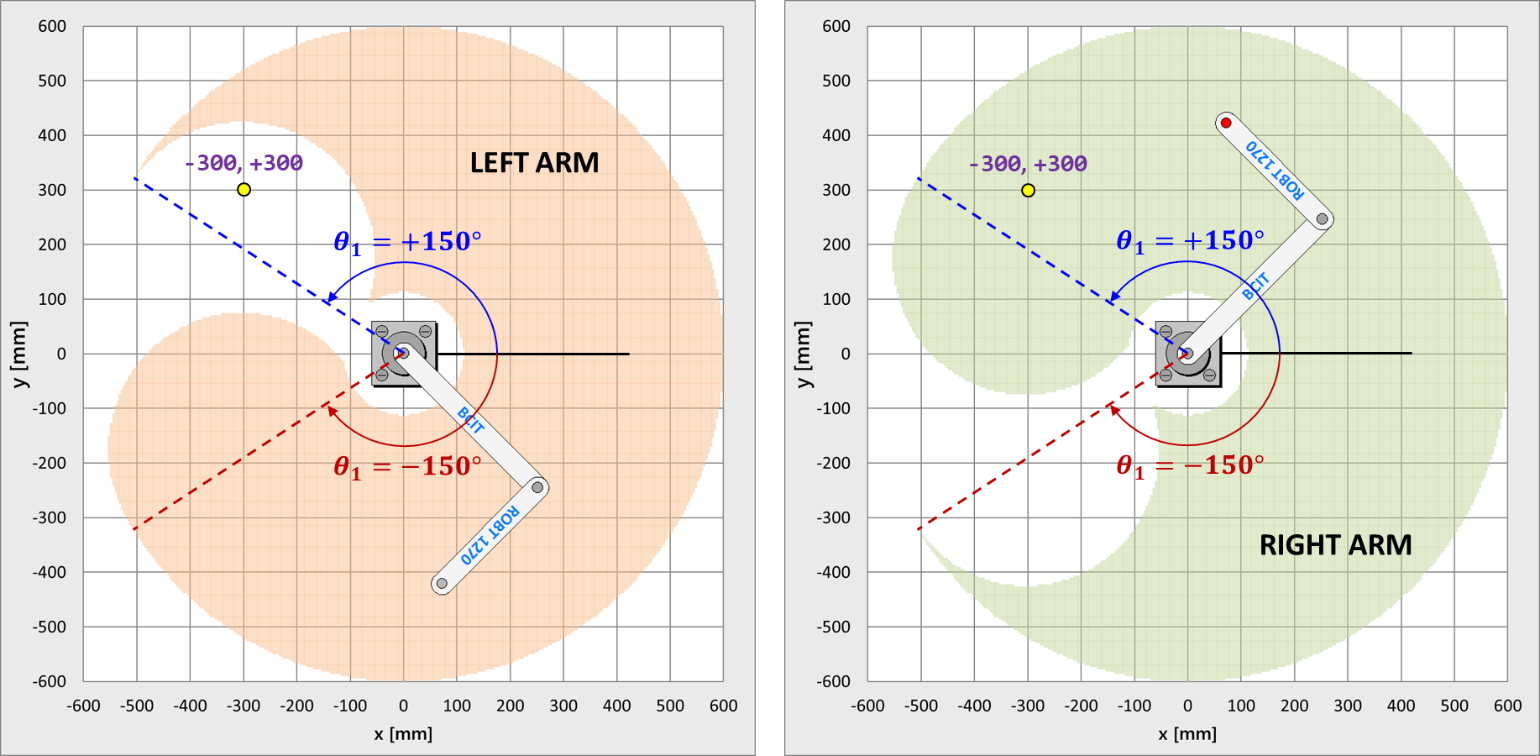


Figure 7

Why do we care about the arm configuration if we can get to an x,y point using either the left or right arm configuration? **The answer is efficiency**. Robots need to execute tasks as quickly as possible (within safety and design margins) so that time is minimized, and profits can be maximized. To illustrate efficiency, let us assume that the pen is at x,y = +300,+300 and the robot is in the left arm configuration. Now let us say we want to move the pen to x,y = -300,+300 in a straight line. From Figure 7, we know that this cannot be done in a left arm configuration alone. We will eventually need to move the arm into the right arm configuration to get to x,y = -300,+300. If we were icing a cake, at the point when we switched from left to right arm, we would have to stop the flow of icing, rotate the arm links to be in the right arm position, and then start the flow of icing again so as not to create an icing 'kink'. Figure 8 below shows the 'kink' that forms at the point we switch from left to right arms while trying to draw a straight line from right to left without lifting the pen.

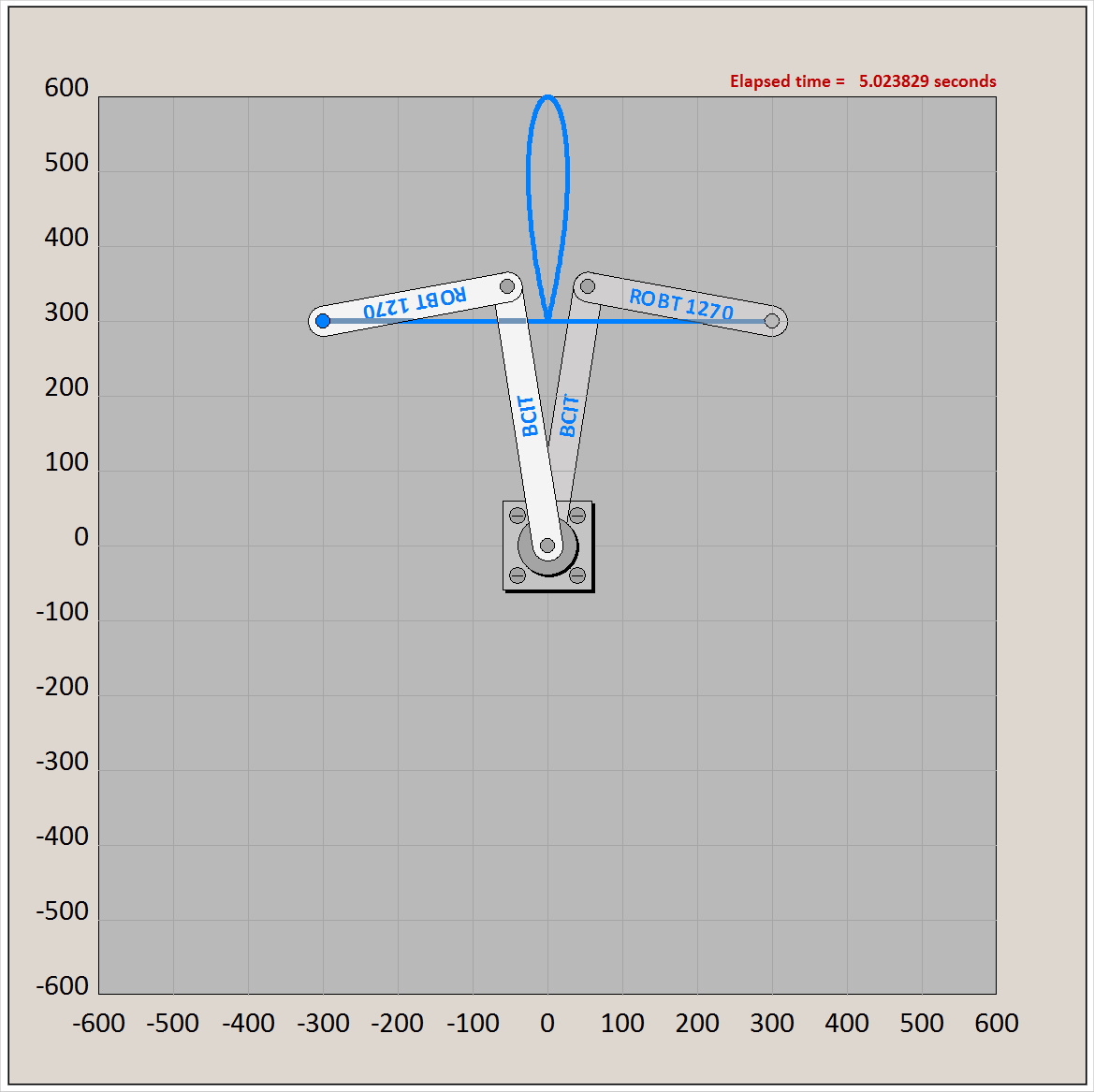


Figure 8

Even if we were moving the robot with the pen up (i.e., not applying icing), the time required to move the pen from point **A** to point **B** will always be greater if we must switch arm configurations during the movement. This is illustrated in Figure 9 below. The movement time from **A** to **B** will be determined by the larger rotation angle between the shoulder and elbow (provided that the motors both have the same maximum rotational speed). In Figure 9 it is easy to see that not switching arms results in less maximum rotation (angle shown by dashed line) and is therefore a more efficient (i.e., faster) path.

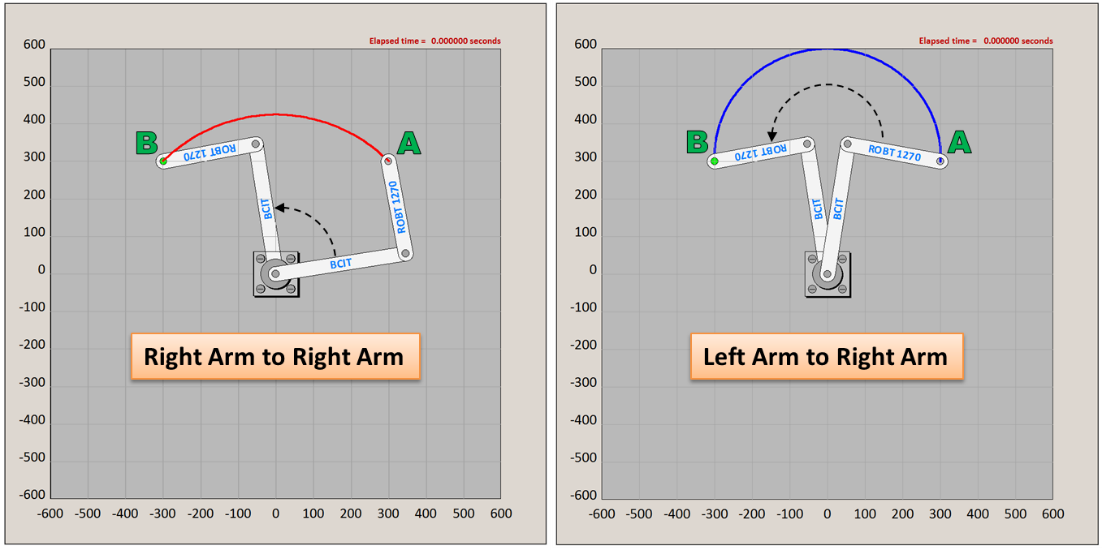


Figure 9

Referring back to Figure 7, we can see that the both arm configurations have identical work sub-envelopes in quadrants 1 and 4. The right arm configuration has a larger work sub-envelope in quadrant 2 (the right arm configuration is actually a superset of the left arm configuration in quadrant 2), while the left arm has greater has a larger work sub-envelope in quadrant 3 (the left arm configuration is a superset of the right arm configuration in quadrant 3). You can use this information to make decisions about how to move the robot more efficiently.

SCARA robots are capable of changing tool tips during operations. Assembly robots use a variety of tool tips during the assembly of a complex part. Changing a tool tip often involves the robot moving to a tool station to get a new tool, a time-consuming process. For optimal efficiency, tool changes must be factored into the overall time taken to complete a task. To simulate a tool change using the SCARA robot simulator, any changes to the pen color will send the robot to the home position () to get a new color pen. This motion must be factored in when trying to minimize task times.

## Remote Commands

As mentioned above, the robot simulator can be controlled remotely through a series of specific commands. These commands are sent as text strings from the control program you will write to the simulator via the network TCP/IP protocol. You **do not** need to care about how the commands get sent, you **do** need to care about how to construct proper commands.

Table 1 below gives a list of all remote commands that can be sent from your controlling program and used to control the simulator. Commands are sent from your program via a **Send()** function that will be provided to you. The **Send()** function takes one parameter in the form of a **C** string. Recall that the first parameter to standard library function **printf** is a string, i.e., printf("I am 20 years old);

You may or may not have learned about how to manage strings at this point. Fortunately, the C standard library contains a function called **sprintf** that mimics the behaviour of **printf** except instead of writing to the screen, the data is written to a string. The Appendix at the end shows you how to create command strings using **sprintf**.

Below the table of commands are examples of valid commands. Note that these do not include the string quotes ("") that you will need for **sprintf**.

|  |  |
| --- | --- |
| **Command** | **Description** |
| ROTATE\_JOINT ANG1 <deg1> ANG2 <deg2> | Rotates the SCARA robot arm links to the desired angles. <deg1> is the inner link angle and <deg2> is the outer link angle, both in degrees (real numbers). Note that the <>'s are not part of the command – they are placeholders to signify variable values. |
| MOTOR\_SPEED HIGH/MEDIUM/LOW | Set the joint motor speeds. The thickness of the pen trace is inversely proportional to the motor speed. **The motor speed for each joint is adjusted so that the rotations finish together**. |
| PEN\_UP | Lifts the pen up so arm movement does not cause writing. |
| PEN\_DOWN | Lowers the pen for writing. |
| PEN\_COLOR <r> <g> <b> | Changes the current pen to a pen with another color. <r>, <g>, and <b> are integers from 0 to 255. They signify the red, green, and blue components of the color. 0 is minimum component intensity, 255 is maximum component intensity. **Note that the robot will be sent home (==) to change the pen and then returned to the position before the pen change**. If the <r>, <g>, <b> values are the same as the current pen, the PEN\_COLOR command will be ignored. |
| CYCLE\_PEN\_COLORS ON/OFF | If ON, the pen color will automatically change between each ROTATE\_JOINT command. If OFF, the pen color will stay fixed at the current color. |
| CLEAR\_TRACE | Clears all pen traces from the display. |
| CLEAR\_REMOTE\_COMMAND\_LOG | Clears the remote command log window. |
| CLEAR\_POSITION\_LOG | Clears the position log window. |
| SHUTDOWN\_SIMULATION | Closes the Simulator. The simulator interface will ask for confirmation of the shutdown. |
| END | Shuts down the remote command link to the Simulator. |
| HOME | Send robot to the home position (==). Note that the pen will be in the PEN\_UP position as the robot links rotate to the home position. |

Table 1

## Examples:

CLEAR\_TRACE

CYCLE\_PEN\_COLORS OFF

CLEAR\_LOG

MOTOR\_SPEED HIGH

PEN\_UP

ROTATE\_JOINT ANG1 47.12 ANG2 -121.55

PEN\_COLOR 255 128 255

PEN\_DOWN

MOTOR\_SPEED LOW

ROTATE\_JOINT ANG1 55.5 ANG2 120.0

PEN\_COLOR 0 0 255

ROTATE\_JOINT ANG1 -20.0 ANG2 -45.0

MOTOR\_SPEED HIGH

HOME

# APPENDIX: Sending Commands to the Simulator using sprintf\_s

To send a command to the robot, use a **sendRobotCommand** function that will be provided to you. This function is not built into **C**, it was written as part of the robot simulator to make it easy for you to communicate with the robot. Examples of using **sendRobotCommand** are shown below:

**sendRobotCommand**("MOTOR\_SPEED MEDIUM\n");

**sendRobotCommand**("PEN\_COLOR 255 128 255\n");

**sendRobotCommand**("PEN\_DOWN\n");

**sendRobotCommand**("ROTATE\_JOINT ANG1 12.50 ANG2 57.34\n");

In the last line of the code example above, the joint angles are written literally in the command string. Although this works, you need to be able to calculate rotation angles in your program and send the calculated angles to the simulator as part of the ROTATE\_JOINT command string. To create a formatted command string from data that is stored in variables, you can use the **sprintf** function.

**sprintf** works like **printf** except that it writes the data to a string instead of to the screen. In **C**, the basis of a string is an array (list) of characters, or **char**s. In **lab2.cpp**, a **char** array named **commandString** is already defined for you. To control the robot, you create a command string using the **sprintf** function and then send the command string to the simulator using the **sendRobotCommand** function with **commandString** as its parameter. Below is an example of how you would use the values stored in two **double** variables **theta1Deg** and **theta2Deg** to move the robot joints to those angles.

char commandString[MAX\_STRING];

double ang1, ang2;

theta1Deg = 35.5;

theta2Deg = -115.6;

sprintf\_s(commandString, MAX\_STRING, "ROTATE\_JOINT ANG1 %lf ANG2 %lf\n",theta1Deg,theta2Deg);

**sendRobotCommand**(commandString);

Of course you will not be hardcoding the angles in your code, but the **sprintf\_s** and **sendRobotCommand** code lines would not change regardless. In the above example, the command that is sent to the simulator is:

**ROTATE\_JOINT ANG1 35.500000 ANG2 -115.600000\n**

**NOTE:** The simulator has been constructed such that **every remote command must end with a newline ('\n') character.** Make sure that all your calls to **sprintf\_s** place a **newline** character at the end of the string.