Product Manual

SmarTac/PIB
Standard version/S4Cplus

503 497-502
2001-03-06

Mounting in the robot control cabinet

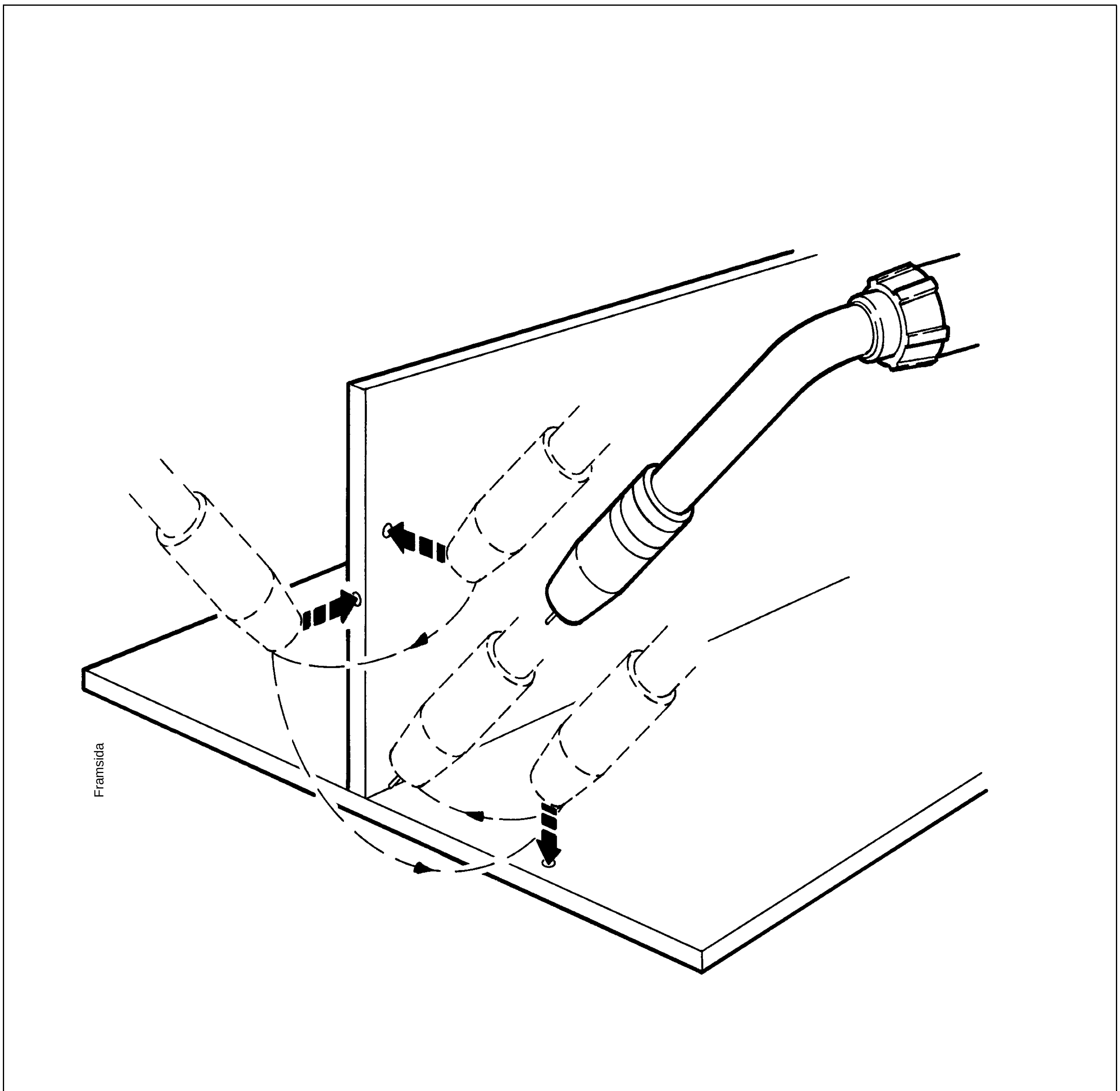


ABB Welding Systems

Article number: 503 497-502

Date: 2001-03-06

ABB Welding Systems AB

S-695 82 Laxå

Sweden

LIST OF CONTENTS

1 Introduction

1.1 Product Overview

SmarTac is a tactile sensor used to find the location of inconsistent weld joints and offset the programmed points in a weld program. The main component is an electronic sensor board, which detects contact with the part feature to be located. The SmarTac board is supplied as an add-on unit and installed in the robot cabinet. A RAPID system module, SmarTac.sys, provided by ABB WSD supports powerful programming tools explained in section 5, “*The User's Guide.*” SmarTac searching can be added to programs while programming a part, or it can be added to a pre-existing weld routine.

1.2 Operation Overview

With SmarTac a part feature may be “searched” using part of the torch. Typically the welding wire or the gas cup is used as the sensing portion of the torch. “Searches” are programmed into a weld sequence. Each search consists of two robtargets; one for the start location and one for the expected location of the part feature. While searching the torch feature (gas cup or wire) is energized with about 40VDC. When the torch feature makes contact with the part (at ground potential) an input is set in the robot controller. When the input is detected, robot location is stored and motion stops.

The Search instructions included in the SmarTac software are designed to return “offset”” information. In other words, the result of a search is the distance between where the original search location was programmed and where the robot has now found the part.

Using SmarTac effectively can dramatically reduce fixturing costs. It can also help account for part variability that can not otherwise be controlled.

1.3 Requirements Overview

1.3.1 System Prerequisites

This SmarTac version is intended for use in arc welding systems incorporating IRB 140,1400, 2400, etc. robots.

BaseWare requirements: 4.0 or higher.

Controller requirements: S4Cplus

The SmarTac package includes one system module that is loaded in the foreground task of the controller. The module, SmarTac.sys, is a stand-alone, read-only, no-step-in, module. Consequently, it is compatible with any RAPID program, assuming the I/O configuration is correct, and no previous version of SmarTac is loaded

1.3.2 User’s Qualifications

Any competent robot programmer (S4 RAPID language) may be self-taught to program and use basic SmarTac searches. Some complex searching techniques are best reserved for those programmers that have attended an advanced programming class offered by ABB, unless the programmer has a solid mathematical background.

For the robotic system operator, the addition of searches is largely transparent and requires no further training.

1.4 Precautions!

The power supply must always be switched off whenever work is carried out in the control cabinet.

Circuit boards - printed circuit boards and components - must never be handled without Electro-Static-Discharge (ESD) protection in order not to damage them. Use the wrist strap located on the inside of the controller door.

All personnel working with the robot system must be very familiar with the safety regulations outlined in the chapter on Safety in the robot controller User's Guide. Incorrect operation can damage the robot or injure someone

2 Technical description

2.1 General

SmarTac/PIB is a further development of the ABB joint search device SmarTac. Mechanically and electrically it is integrated with the ABB welding interface PIB (Process Interface Board).

The unit has two sensor inputs, which can be activated one at a time or simultaneously.

The unit is a so-called "Add-on" unit and is connected to the PIB by way of a 32-pole connector of the Euro type, see Figure 1.

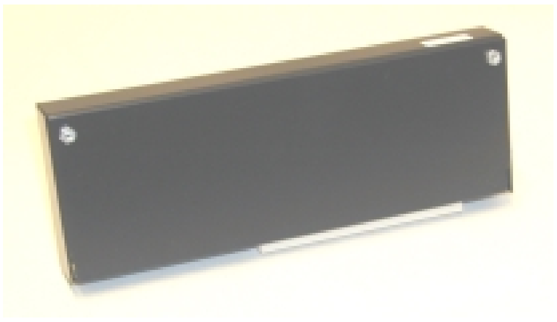

smartac

Figure 1

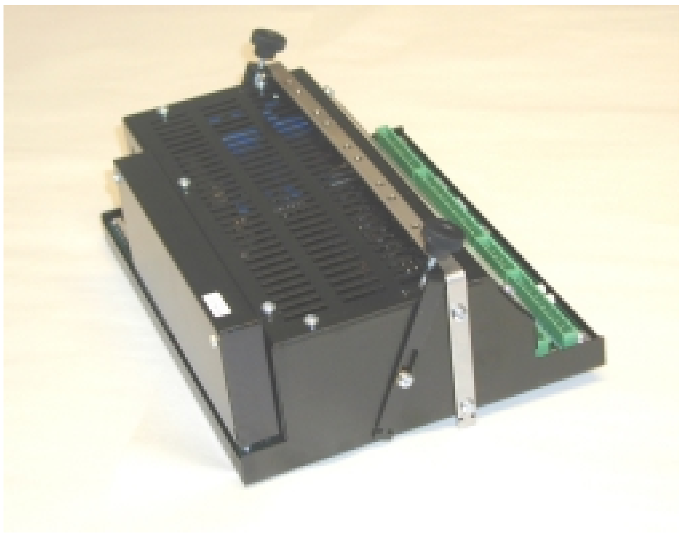

smartac-pub

Figure 2

The search properties of SmarTac/PIB are determined by two of the adjustable parameters, Voltage Valid Limit and Sensor Detection Sensitivity. They are transferred from the robot together with other PIB configuration data. See *Product Manual, Welding Equipment/S4Cplus* for Configuration Parameters.

The search properties can thereby be adapted to the existing circumstances of the search circuit.

2.2 Sensors

In the welding system A314/A324 containing PIB, the input for sensor 1 is connected to the gas cup of the welding gun, whereas sensor input 2 is connected to the welding nozzle for searching by way of the welding wire.

Using sensor 2 it is usually necessary to disconnect the welding circuit to avoid current divertion through the power source resulting in too low search voltage.

Searching with sensor 2 is only used for special applications.

2.3 Function Description - Searching

The search of the joint is usually done using a search routine in the robot program. The following description assumes that the ABB Flexible Automation signal names and robot configuration apply.

In deliveries containing SmarTac, programs for the search routine and configuration parameters are pre-loaded. When SmarTac is delivered as an option, a diskette containing the corresponding data comes with the delivery.

Note.
Although the pre-loaded configuration parameters for SmarTac in most cases fit, a change may be needed for the actual application. The procedure for this is the same as for all configuration parameters in PIB. See *Product Manual, Welding Equipment/S4Cplus*.

In the event the configuration parameters must be modified, the same conditions apply as for PIB. See the section Configuration.

Activating the Sensor (Sensor 1)

The sensor is activated by a message from the robot to the PIB, doSE1_SEL=1, applying the search voltage to the gas cup of the welding gun.

The search voltage connected between the gas cup and the object to be searched is generated by a voltage source galvanically separated from other current circuits.

Checking the Sensor (Voltage Valid Limit)

When a sensor is activated the search voltage will depend on the insulating properties of the open search circuit.

Low insulation value between the sensor and the parts having electrical contact with the object to be searched will reduce the search voltage, due to for example the passage of current through the water when a water-cooled welding gun is used, soot formation, etc.

Increased contact resistance due to oxide layers, oil film, soot, etc. in combination with decreased search voltage makes it more difficult to achieve reliable contact between the sensor and the search object.

Using the adjustable parameter Voltage Valid Limit a level can be set under which the search shall not continue.

Configuration range: 0 – 40V in steps of 1V.
Preconfigured value: 8

If the present search voltage is higher than the Voltage Valid Limit, the message diSe_Valid=1 will be sent from the PIB to the robot giving the robot the signal for carrying on the search.

Sensitivity (Sensor Detection Sensitivity)

The adjustable parameter Sensor Detection Sens determines the sensitivity of the sensor. Configuration range: 0-10V in steps of 0.1V.

The SmarTac trigger level is locked by the message doSE_REF=1 from the robot according to the following:
Trigger level = the present search voltage - the Sensor Detection Sens value.

Under normal conditions reliable search is achieved using values > = 1V.
Preconfigured value: 10

Detection

When during the search the gas cup gets into contact with the search object the sensor input is exposed to voltage drop.

If the voltage drops below the trigger level the PIB will send the search stop message diSE1_DET=1 to the robot, and the co-ordinates of the search object can be registered.

2.4 Delivery

SmarTac is delivered as SmarTac complete, article no. 503500-880, consisting of:

- SmarTac unit
- Software, contained in the system diskette when a complete system is delivered, and in a separate diskette when SmarTac is delivered separately
- User’s Guide with program description and examples.

2.5 Technical Data

Accuracy: Max. deviation ± 0.25 mm at a search speed of 20 mm/sec.

Marking: See Figure 3.

Mechanical Data

Weight: 0.220 kg
Dimensions: 22x65x185 mm (see Figure 3)
Enclosure class: IP 20.

Electrical Data:

Max. serach voltage: 40V
Max. search current: 4.3 mA.

Environmental data: See point 6.16.3

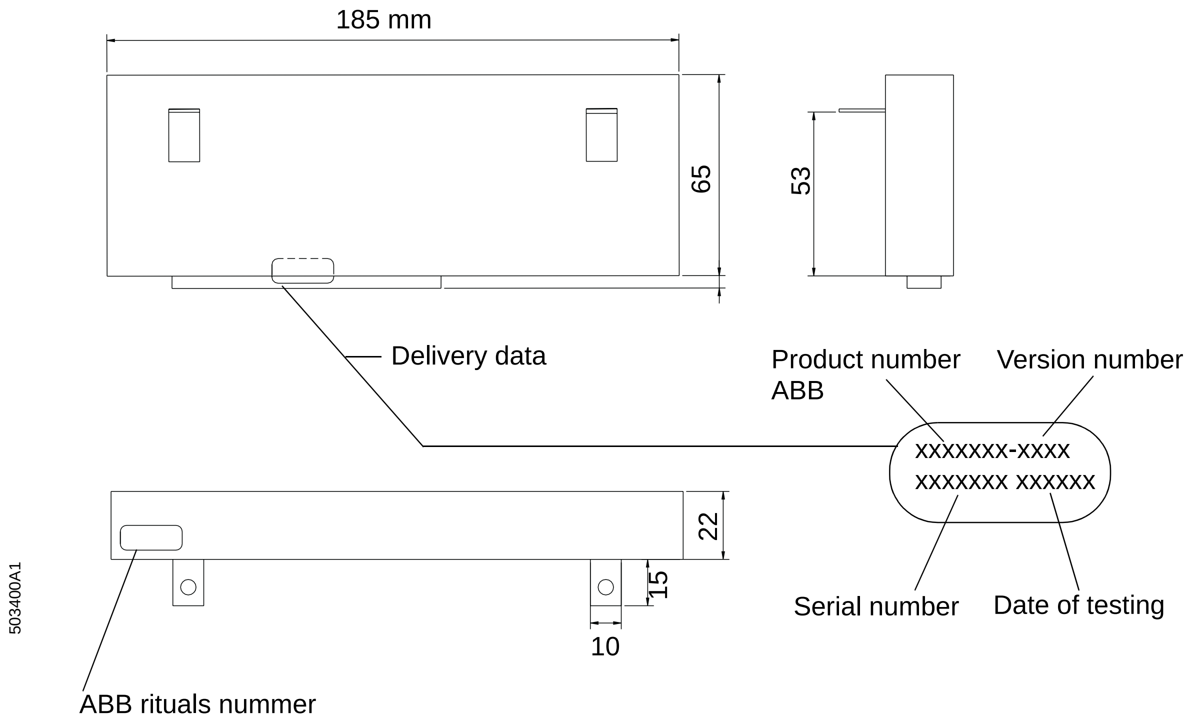

185 mm
65
53
Delivery data
Product number ABB
Version number
xxxxxxx-xxxx
xxxxxxx xxxxxx
22
15
10
Serial number
Date of testing
503400A1
ABB rituals nummer

Figure 3

3 Installation

3.1 Components List

The following items are supplied with the SmarTac option:

- SmarTac “Add on” unit.
- SmarTac.sys RAPID system module.
- Relevant electrical schematics.

3.2 Tools Required

The following tools are required to install the SmarTac option:

- Torx screwdriver.

3.3 Hardware set-up

3.3.1 SmarTac Board Installation

1. Turn off incoming power to S4Cplus controller.

2. When handling the SmarTac board, avoid electrical discharge damage by wearing a grounding wrist strap or similar Electro-Static-Discharge (ESD) protection.

3. Install the SmarTac “Add on” unit on the PIB, see Figure 2.

3.4 Software set-up

3.4.1 Compatibility

This SmarTac version is intended for use in arc welding systems incorporating IRB 140,1400, 2400, etc. robots.

BaseWare requirements: 4.0 or higher.

Controller requirements: S4Cplus

The SmarTac package includes one system module that is loaded in the foreground task of the controller. The module, SmarTac.sys, is a stand-alone, read-only, no-step-in, module. Consequently, it is compatible with any RAPID program, assuming the I/O configuration is non-conflicting

3.4.2 System Parameters

Board	**Channel**	**Designation**	**Association**
B_PROC_30	**Input Ch 10**	**diSE_VALID**	**SmarTac**
B_PROC_30	**Output Ch 12**	**doSE1_SEL**	**SmarTac**
B_PROC_30	**Output Ch 14**	**doSE_REF**	**SmarTac**
B_PROC_30	**Output Ch 7**	**diSE1_DET**	**SmarTac**

3.4.3 Loading Software

1. The software will be loaded together with the robot aw configuration option “3HEA 50385088”.

4 Applications Guide

4.1 Searching Conditions

SmarTac is intended for use in the following conditions:

- In applications where surfaces are free from rust, mill scale, paint, or other electrically-insulating layer or coating.
- If the gas nozzle is used for a search probe, it must be cleaned at regular intervals.
- If a water-cooled torch is used, the quality of the cooling water is very important. Impure water, e.g. containing salt solution, will act as a load that will reduce the sensitivity and/or reduce the sensing voltage below SmarTac working range. Distilled water or a non-conductive coolant such as ethylene glycol is recommended as gun coolant solution. “Tap” water is unacceptable.

4.2 Programming Limitations

The use of searches ranges from very simplistic to very complex. In some instances, very complex searching techniques must be used to adequately determine weld seam locations. In such instances, assistance from an experienced ABB technician may be required.

4.3 Signals and Connections

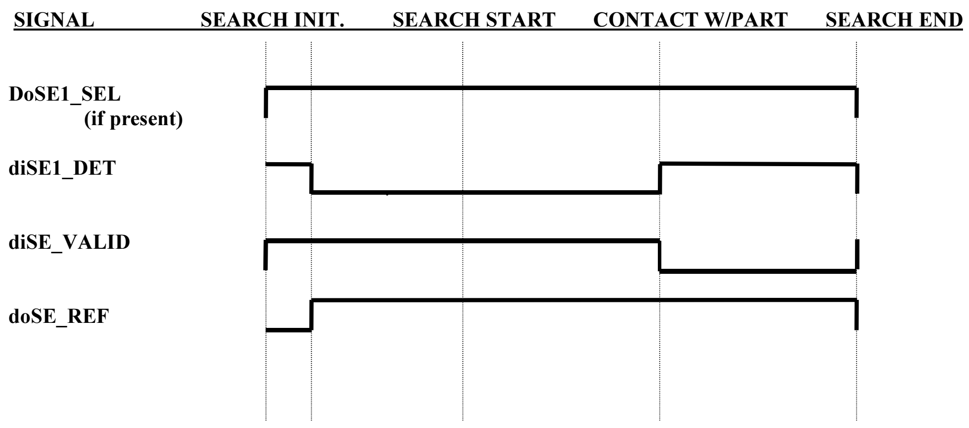
SmarTac is operated by signals from the robot. Signals and connections are described below.

SmarTac I/O:

There are 4 I/O signals used by SmarTac.

diSE_DET	Input used for surface detection.
diSE_VALID	Input used for validation of the search voltage.
doSE_REF	Output used to set the sensing reference voltage.
doSE1_SEL	Output used to set the search detection signal.

4.3.1 I/O Timeline



5 User's Guide

5.1 Safety

Failure to follow safety guidelines presented throughout this manual can result in property damage, serious injury, or death!

Consequently, the control cabinet door(s) should always be closed when the control cabinet is turned on. Only qualified technicians should ever attempt trouble shooting.

The SmarTac sensing voltage applied at the torch when searching is supplied by a 40VDC, low current source. This sensing current is harmless.

The programmer must take all safety precautions presented in the Robot Controller's User's Guide while operating the robotic system.

5.2 Introduction

The SmarTac system available from ABB Flexible automation, is a very versatile tool for accurately locating weld seams. The system module, SmarTac.sys, included in the package, contains useful search instructions that simplify the programming. The module also includes mathematical functions that are useful in advanced searching techniques. All of these are discussed in this section.

Before tactile searching can be used effectively, you need to be able to answer these questions:

1. How do my parts deviate?

Knowing where the parts move, and where they don't is critical for determining what features to search. Searching takes time. Unnecessary searches increase cycle time and programming complexity. In this manual, the simplest cases will be handled first. In many cases these techniques will be enough. In some situations where part fit-up and/or fixtures is poor, you'll need to understand all the tricks described in the manual.

2. What is a *frame*?

A good understanding of *work objects* and *displacement frames* is the key to successful programming with SmarTac searching. This guide will start with a discussion of these important elements. As the programming examples in the guide become more complex, this topic will be further elaborated on.

3. What are the RAPID instructions and how are they used to manipulate my weld routines?

In this guide we will look at several search techniques and the instructions required. Detailed examples will be provided for each of these. Plus the appendices, *SmarTac Instructions* and *SmarTac Functions*, provide detailed explanations of the instructions and functions included with the SmarTac software.

5.3 What is a *Frame*?

The ABB controller utilizes a number of frames to describe the relationship of the robot tool to the world. The following is a very brief synopsis of what is described in the *S4 Robot Controller User's Guide*. The next page or so may be difficult to understand at first. Don't worry about it. The exercises later on will help to solidify your understanding of these concepts.

Frames are coordinate systems. In the robot controller, frames are described by location and orientation. The location of a frame is defined by three dimensions, x, y, and z. These three dimensions dictate where the origin of the frame is located. The orientation of a frame is described by four quaternions.

Frames are used in tool definitions, *tooldata*, robot targets, *robtargets*, work object definitions, *wobjdata*, and program displacement definitions, *pose*. The tool frame definition describes how the Tool Center Point (TCP) is related to the end of the robot arm, which is related to the *world* frame through the base of the robot. A *robtarget* is related to a *program displacement frame*, which is related to the *object frame* of a work object, which is related to the *user frame* of the work object, which is related to the *world frame*. Take a look at this figure from the *Robot Controller User's Guide*:

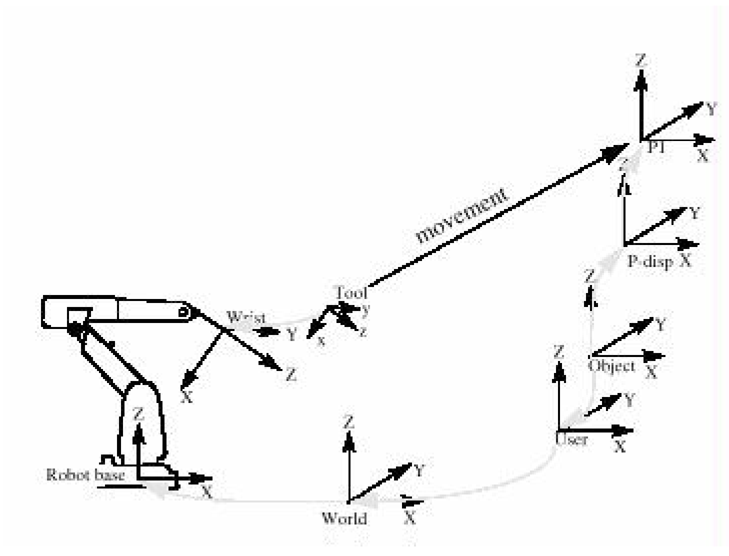


Figure 4.

In the figure, *P1* is a robtarget frame. *Pdisp* is the current program displacement frame. *Object* is the object frame of the current work object. *User* is the user frame of the current work object. (Note: Work objects have two frames: *object* and *user*.) *World* is the frame that all the other frames are related to in one way or another. The *Robot base* is defined as the *baseframe* of the robot.

In the figure above, if any of the frames are changed, the relationship of the TCP to the *robtarget* changes. If the *user* frame of the work object is changed by 10mm in 'X', in relation to the *World* frame, then the *robtarget* will move 10mm in 'X', in relation to the *World* frame. If the *program displacement* is changed by 10mm in 'X', in relation to the *object* frame, then the *robtarget* will move 10mm in 'X', in relation to the *object* frame.

5.3.1 Exercise 1 Program Displacement

This exercise demonstrates how a program displacement works. The exercises later in this guide will not be as detailed as this one. Please take the time to understand this exercise before attempting others.

If the SmarTac module has not been loaded, instructions are found in the Installation and Maintenance section of this manual. All exercises assume that the SmarTac software and hardware is installed and working properly.

1. Create a new program module to work in. Call it "ST_TEST". Consult the *Controller User's Guide* if you need help.

2. Create a new routine in that module and call it "disp_ex1".

3. If not already done, define the tool using the five point method or Bullseye. Call the tool "tWeldGun". Consult *the Controller User's Guide*, or *Bullseye* manual if you need help.

Tip: To make programming easier you may want to add in these commands into one of your "Most Common" pick lists:

PDispAdd

PDispOff

PDispSet

Search_1D

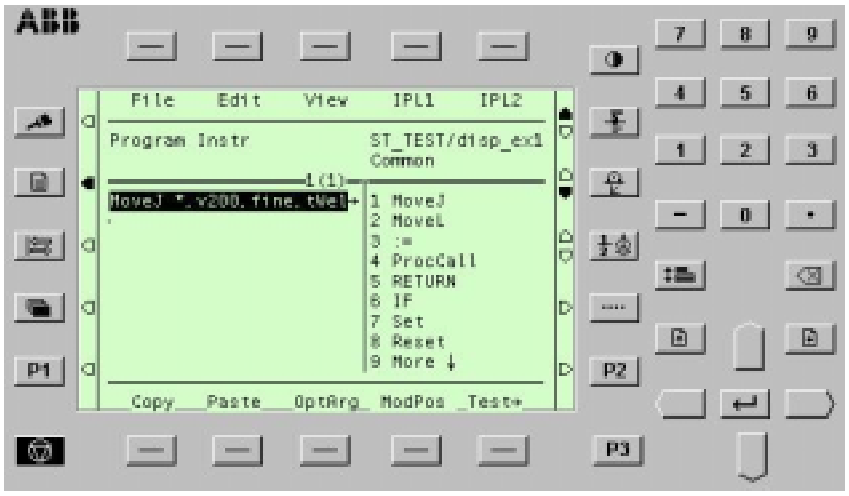
Search_Groove

Search_Part

4. Tape a piece of paper to a table, or similar surface, within the robot's reach. On the paper draw a rectangle. (Perfection is not required here.)

5. View the modules, select the new module, "ST_TEST", and select the new routine, "disp_ex1".

6. Jog the robot so that the torch is pointing at the rectangle you drew. The tip of the torch should be a few inches above the rectangle. Create a MoveJ at this point using tWeldGun and no work object selected:

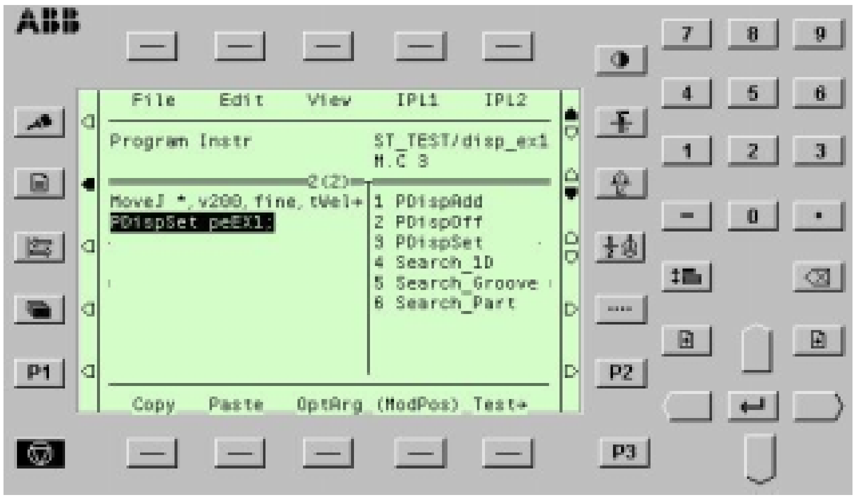


```
  File     Edit     View     IPL1     IPL2

Program Instr                  ST_TEST/disp_ex1
                               Common
------------------------1(1)--------------------
MoveJ *,v200,fine,tWel→ 1 MoveJ
                        2 MoveL
                        3 :=
                        4 ProcCall
                        5 RETURN
                        6 IF
                        7 Set
                        8 Reset
                        9 More ↓

  Copy    Paste    OptArg   ModPos   Test→
```

7. Now insert the instruction, *PDispSet*. This is a RAPID command that will be found on one of the standard instruction pick-lists. Here, you will see that we used a custom "Most Common" pick-list. The PDispSet instruction requires one argument: a *Displacement Frame*. When prompted for this data, select *new*… and create a new *pose* data type called "peEX1":



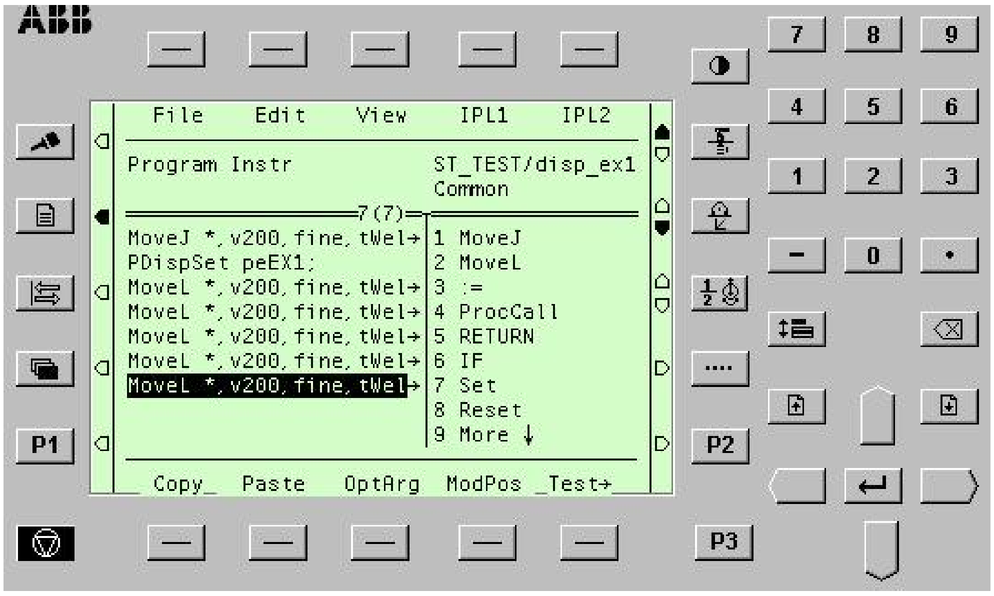
```
  File     Edit     View     IPL1     IPL2

Program Instr                  ST_TEST/disp_ex1
                               M.C 3
------------------------2(2)--------------------
MoveJ *,v200,fine,tWel→ 1 PDispAdd
PDispSet peEX1;         2 PDispOff
                        3 PDispSet
                        4 Search_1D
                        5 Search_Groove
                        6 Search_Part

  Copy    Paste    OptArg  (ModPos)  Test→
```

8. Now jog the robot down to the rectangle so that the tip is just above one of the rectangle corners. Create a MoveL at this position using tWeldGun and no work object selected.

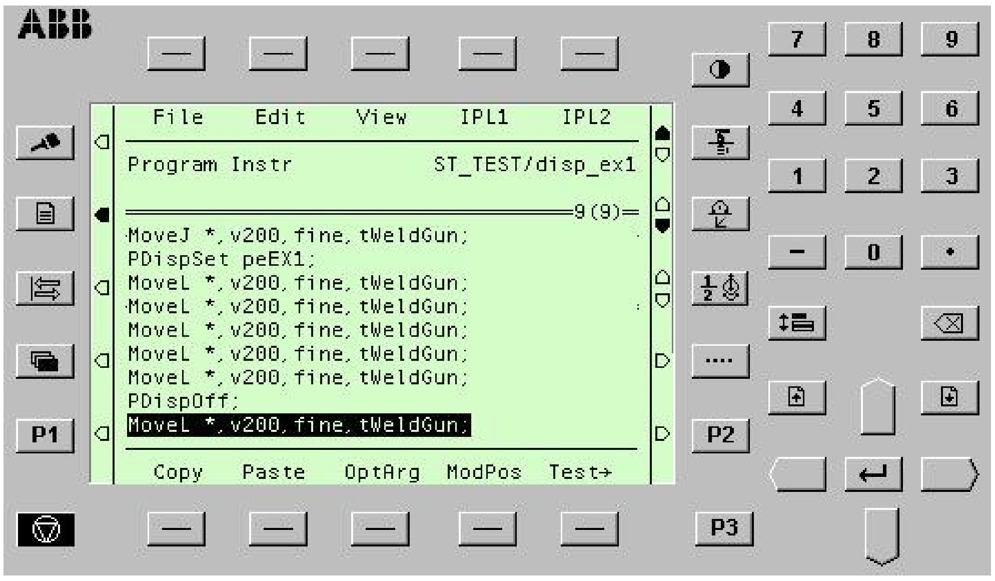
9. Do the same for the rest of the corners, returning to the first corner, so that you have a short program that traces out the rectangle:



```
Program Instr                    ST_TEST/disp_ex1
                                 Common
=========================7(7)=
MoveJ *,v200,fine,tWel→ | 1 MoveJ
PDispSet peEX1;         | 2 MoveL
MoveL *,v200,fine,tWel→ | 3 :=
MoveL *,v200,fine,tWel→ | 4 ProcCall
MoveL *,v200,fine,tWel→ | 5 RETURN
MoveL *,v200,fine,tWel→ | 6 IF
MoveL *,v200,fine,tWel→ | 7 Set
                        | 8 Reset
                        | 9 More ↓
```

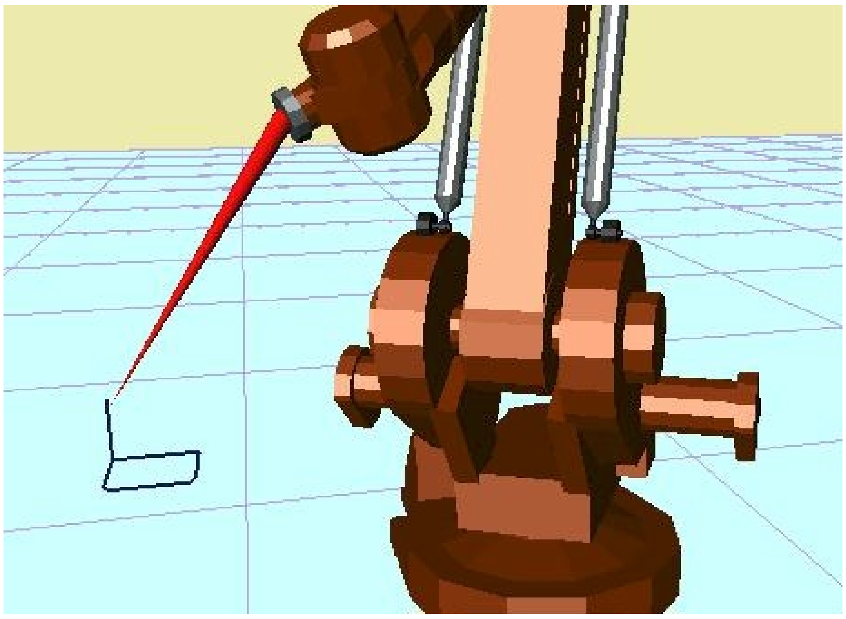
10. Now insert the instruction: *PDispOff*.

11. Jog the robot away from the rectangle a few inches and insert a final MoveL:



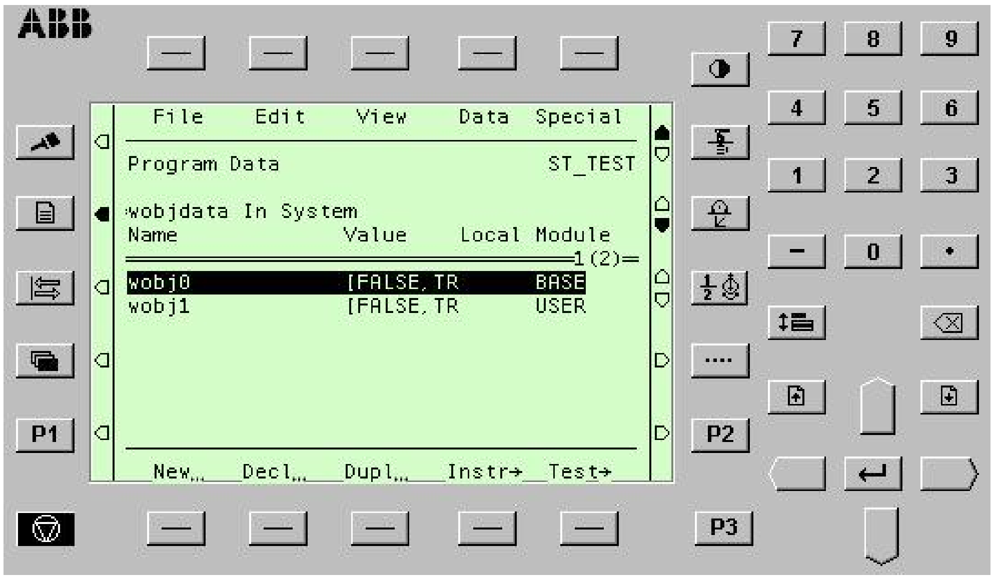
```
Program Instr                    ST_TEST/disp_ex1
===================================9(9)=
MoveJ *,v200,fine,tWeldGun;
PDispSet peEX1;
MoveL *,v200,fine,tWeldGun;
MoveL *,v200,fine,tWeldGun;
MoveL *,v200,fine,tWeldGun;
MoveL *,v200,fine,tWeldGun;
MoveL *,v200,fine,tWeldGun;
PDispOff;
MoveL *,v200,fine,tWeldGun;
```

12. Execute this routine from the beginning in manual mode to be sure the program works correctly. The path should look something like this:



This routine executes a simple movement path. Each of the robtargets in the MoveL instructions is related to the World frame through the chain of frames discussed earlier. In this case, however, the work object has not been specified, so wobj0 is used by default. This work object is the same as the *world* frame.

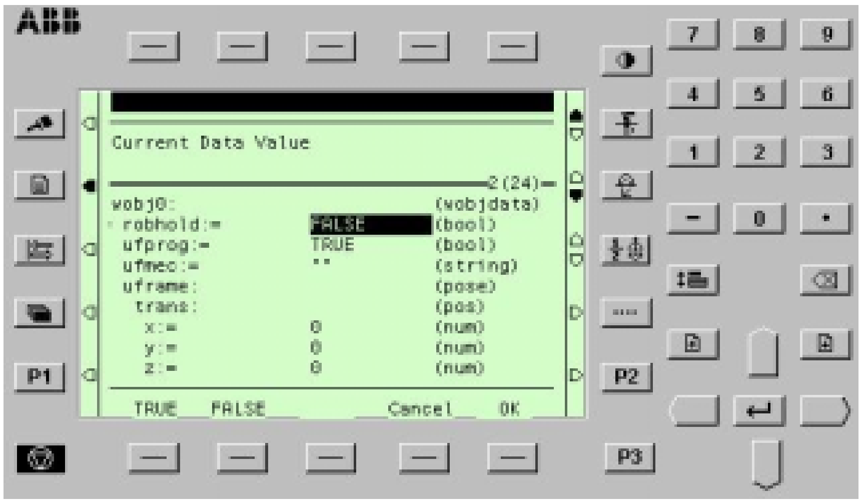
13. Take a look at the values in wobj0 by selecting *View* and "Data Types". Then choose *wobjdata* off the list. The list of work objects defined in the system is shown:



```
 File    Edit    View    Data  Special
Program Data                   ST_TEST
wobjdata In System
Name               Value     Local Module
══════════════════════════════════1(2)═
wobj0              [FALSE,TR       BASE
wobj1              [FALSE,TR       USER

 New...   Decl...   Dupl...   Instr→  Test→
```

14. Highlight *wobj0* and hit enter. The following will be visible:



```
Current Data Value
                                         2(24)
wobj0:                                  (wobjdata)
 robhold:=          FALSE               (bool)
 ufprog:=           TRUE                (bool)
 ufmec:=            ""                  (string)
 uframe:                                (pose)
  trans:                                (pos)
   x:=              0                   (num)
   y:=              0                   (num)
   z:=              0                   (num)
 TRUE   FALSE                 Cancel    OK
```

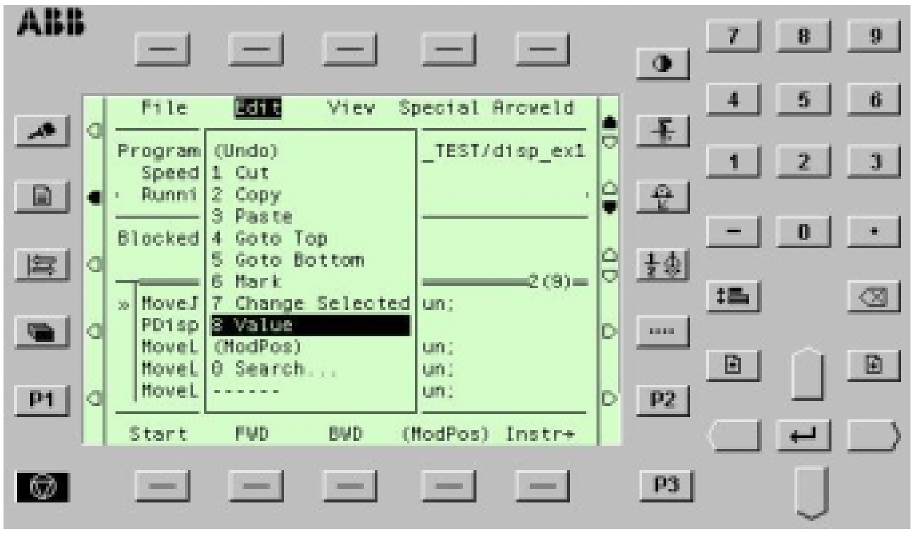
Cursor down and look at the data that is present. Note that a work object has two frames, the user frame, *uframe*, and the object frame, *oframe*. Also note that the values are all zero for the locations, and ones and zeros for the orientations. That's why the work object has no affect on our program. It is the same as the *World* frame.

The only other frame that can change the robtarget positions in our "rectangle" routine, is the Displacement Frame.

15. Hit "Cancel" to get out of the work object window. Then hit "test" to return to the test screen.

16. Move the cursor to the PDispSet instruction, and over to the argument, peEX1. Only peEX1 should be highlighted.

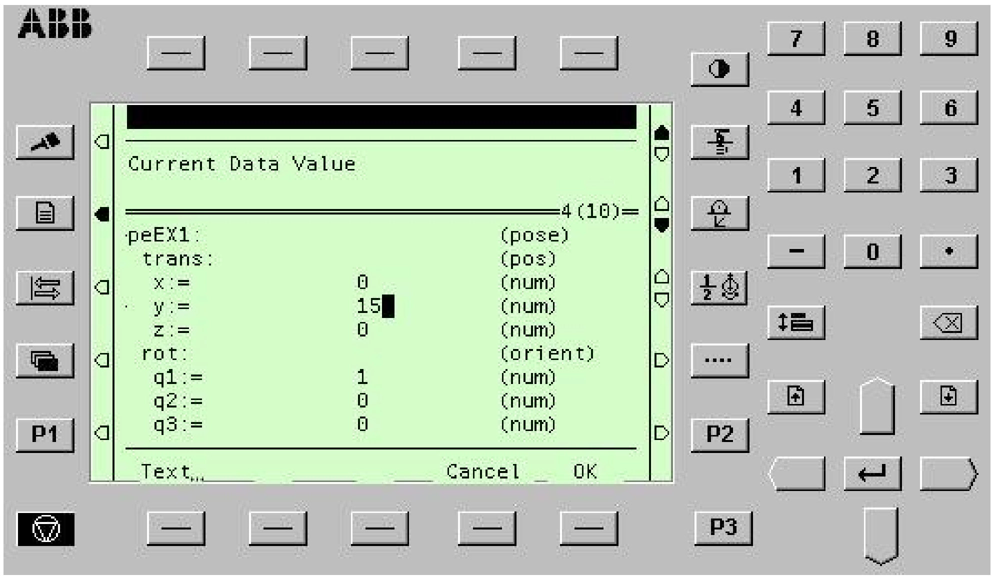
17. View the data in peEX1 by selecting "Edit" and "value":



```
  File     Edit     View  Special Arcweld
 Program (Undo)              _TEST/disp_ex1
   Speed 1 Cut
   Runni 2 Copy
         3 Paste
 Blocked 4 Goto Top
         5 Goto Bottom
         6 Mark                          2(9)
 » MoveJ 7 Change Selected   un;
   PDisp 8 Value
   MoveL (ModPos)            un;
   MoveL 0 Search...         un;
   MoveL ------              un;
  Start    FWD     BWD   (ModPos) Instr+
```

A screen will appear showing the values present in this data type, *pose*. (Note: This feature works for all data types.) X, Y, and Z will all be zero. This displacement frame did not alter the rectangle program at all.

18. Move the cursor to the Y value and change the number to 15:


```
Current Data Value
                                     4(10)=
peEX1:                         (pose)
  trans:                       (pos)
    x:=          0             (num)
    y:=          15            (num)
    z:=          0             (num)
  rot:                         (orient)
    q1:=         1             (num)
    q2:=         0             (num)
    q3:=         0             (num)
  Text...                Cancel    OK
```

19. Hit OK and run the routine, *disp_ex1*, again.

What happens? The moves are shifted 15mm in the positive Y direction. That’s 15mm in Y relative to the work object *object frame*. And, as discussed earlier, the object frame and user frame in wobj0 are the *same* as the world frame. So the rectangle moved 15mm relative to the *world*, as well.

This is what Program Displacement Frames do. A change in the displacement frame changes the location of the robtargets. Displacement Frames can be turned on and off using *PDispSet* and *PDispOff*. Similarly, changes in the work object will move the robtargets as well (Work object manipulation will be discussed later).

Try making other changes in X, Y, and Z of *peEX1*. Remember, positive Z will move the rectangle up. Don’t use a negative Z, or you’ll crash the tool. Use the technique described in step #17.

Important: Do not make changes to the four quaternions, q1…q4. If quaternions are changed manually, errors could occur. Quaternions must be normalized, so it is not possible to choose numbers randomly. This will be discussed later.

Advanced: Look at the *robtarget* data using the same technique described in step #17 for looking at *pose* data. Notice that the robtarget data does not change when the rectangle is moved using the displacement frame. Do you understand why?

5.4 Using SmarTac to modify a Displacement Frame.

SmarTac programming tools provide a simple way to search a part feature and apply the search results to a program displacement frame. As Exercise 1 demonstrated, using program displacements is an easy way of shifting programmed robtargets. The most basic search is a one-dimensional search. A 1-D search finds an offset in one direction.

Search_1D is an instruction that is included in the SmarTac system module. Of the three search instructions included in the module, this is the most common. It is useful in a variety of situations and will be present in most of the exercises and examples from this point on.

The *Search_1D* instruction has the following structure:

```
Search_1D peOffset,p1,p2,v200,tWeldGun;
```

Search_1D has five required arguments:

Argument	**Example Name**	**Data Type**
1. Result	peOffset	pose
2. StartPoint	p1	robtarget
3. SearchPoint	p2	robtarget
4. Speed	v200	speeddata
5. Tool	tWeldGun	tooldata

When executed, the robot makes an 'L' move to the *StartPoint*, 'p1'. The TCP velocity described in *Speed* determines the speed of this first move. The SmarTac board is activated and motion starts towards the *SearchPoint*, 'p2'. The robot will continue past the search point for a total search distance described by twice the distance between *StartPoint* and *SearchPoint*, or until the part feature is sensed. Once the part feature is sensed, motion stops, and the displacement data, *Result*, is stored.

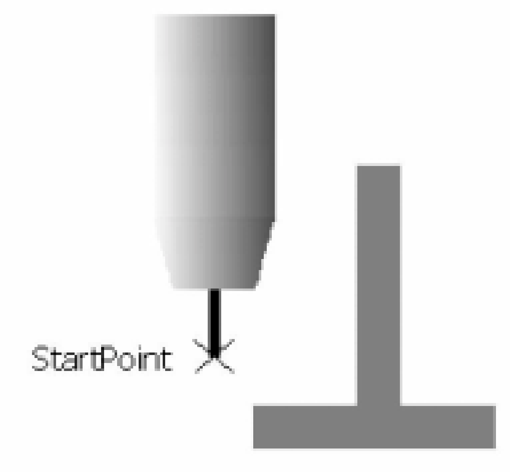


Figure 5.

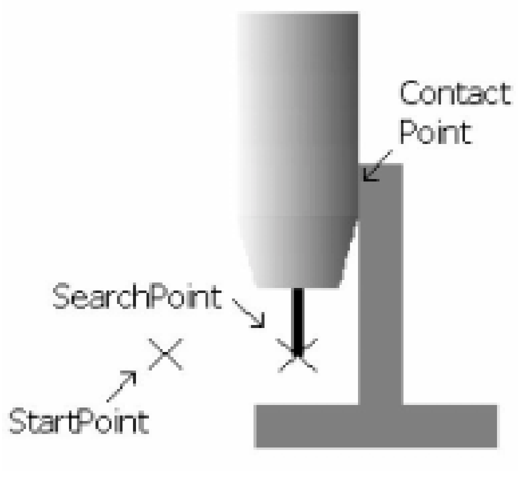


Figure 6.

The *StartPoint* and *SearchPoint* are programmed. The two points determine the direction of the search. The *SearchPoint* is programmed so the torch is touching the part feature. The *Result* is the difference between the programmed *SearchPoint*, and the actual *SearchStop* that is found when a different part is present.

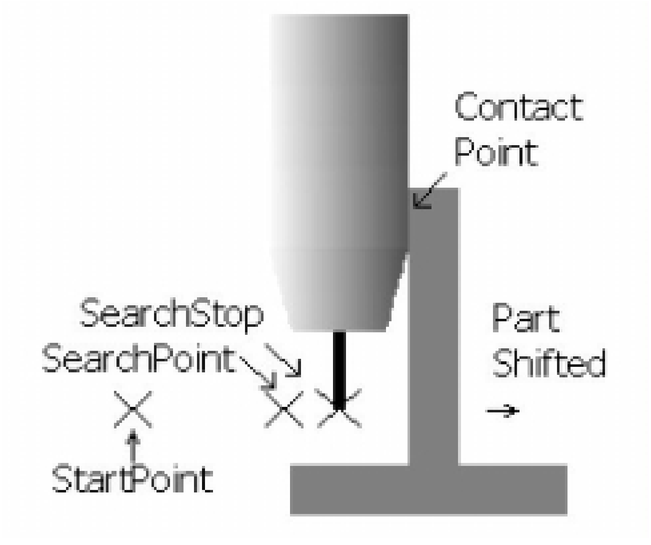


Figure 7.

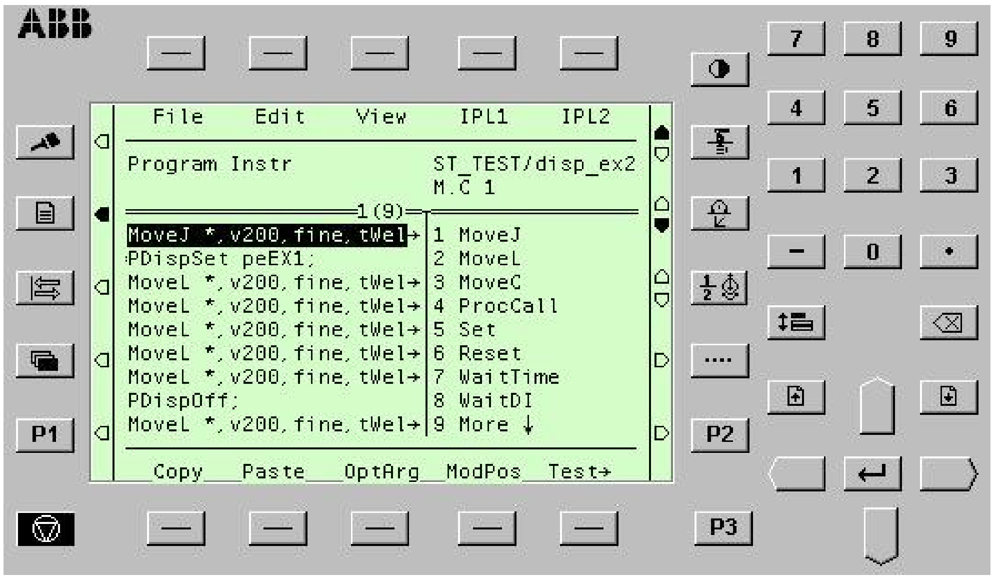
The following exercise demonstrates the instruction, *Search_1D*.

5.4.1 Exercise 2 One Dimensional Search

For this exercise the movement routine used in Exercise 1 is used. If you did not complete Exercise 1, you will need to write a similar routine.

1. View the routines in the module “ST_TEST”.

2. Highlight “disp_ex1” and hit the key marked *Dupl…* at the bottom of the teach pendant screen. The controller will offer the default routine name, “disp_ex2”. Accept this, then look at the instructions in the new routine:



```
  File    Edit    View    IPL1    IPL2

Program Instr             ST_TEST/disp_ex2
                          M.C 1
══════════════════1(9)═╤════════════════
MoveJ *,v200,fine,tWel→│1 MoveJ
PDispSet peEX1;        │2 MoveL
MoveL *,v200,fine,tWel→│3 MoveC
MoveL *,v200,fine,tWel→│4 ProcCall
MoveL *,v200,fine,tWel→│5 Set
MoveL *,v200,fine,tWel→│6 Reset
MoveL *,v200,fine,tWel→│7 WaitTime
PDispOff;              │8 WaitDI
MoveL *,v200,fine,tWel→│9 More ↓

  Copy   Paste   OptArg   ModPos   Test→
```

3. Remove the paper with the rectangle drawn on it.

4. Place a metal plate on the table so that a portion of the plate is overhanging:

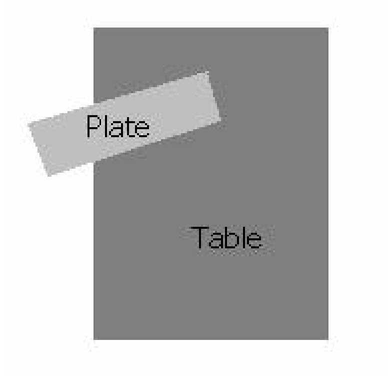


Figure 8. “Top View”

5. Edit the values in peEX1 so that X, Y, and Z equal zero.

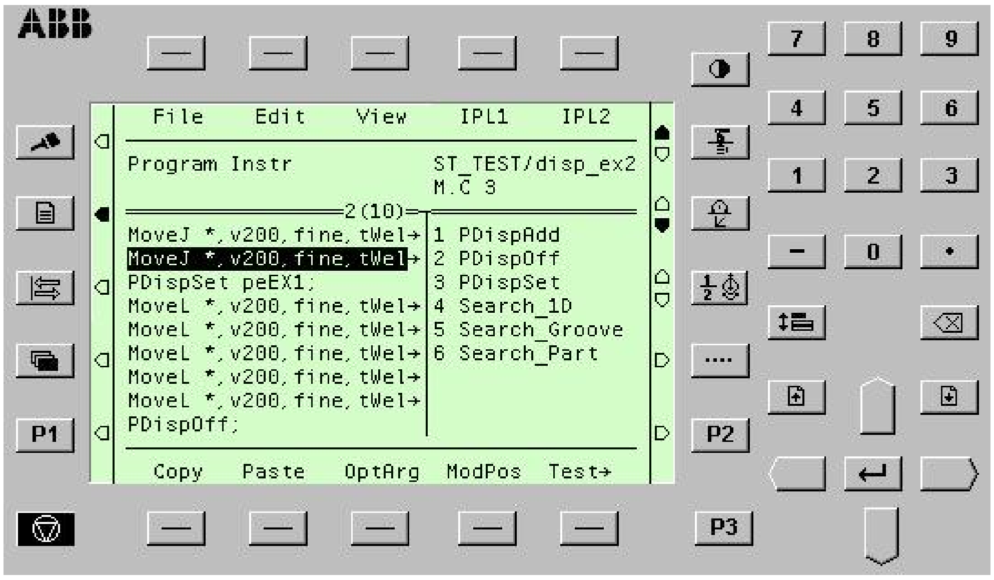
6. Move the Program Pointer to the new routine and toggle the Program Window to *test* mode.

7. Step through the rectangle program and modify each of the points so that they correspond to each of the plate's corners.

8. Continue to step though the routine until the Program Pointer loops back to the beginning.

9. Toggle the Program Window to *instruction* mode. If a “Most Common” pick-list with SmarTac instructions was created, select it.

10. Move the cursor to the first line, *MoveJ*. Using the Copy and Paste keys at the bottom of the screen, copy the *MoveJ* and paste it right below it:



```
Program Instr                ST_TEST/disp_ex2
                             M.C 3
===================2(10)=
MoveJ *,v200,fine,tWel→ 1 PDispAdd
MoveJ *,v200,fine,tWel→ 2 PDispOff
PDispSet peEX1;         3 PDispSet
MoveL *,v200,fine,tWel→ 4 Search_1D
MoveL *,v200,fine,tWel→ 5 Search_Groove
MoveL *,v200,fine,tWel→ 6 Search_Part
MoveL *,v200,fine,tWel→
MoveL *,v200,fine,tWel→
PDispOff;
```

Between the two *MoveJ* instructions we will be adding more moves and a search instruction. The search will collect information about the location of the plate, and store the information in *peEX1*. Our plate-tracing movements will then be shifted accordingly.

11. Jog the robot torch so that it is above and past the edge of the plate. Insert a *MoveL* instruction for this location between the two *MoveJ* instructions.

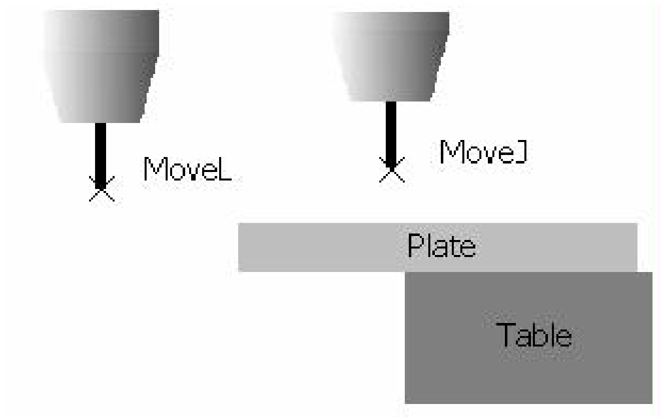
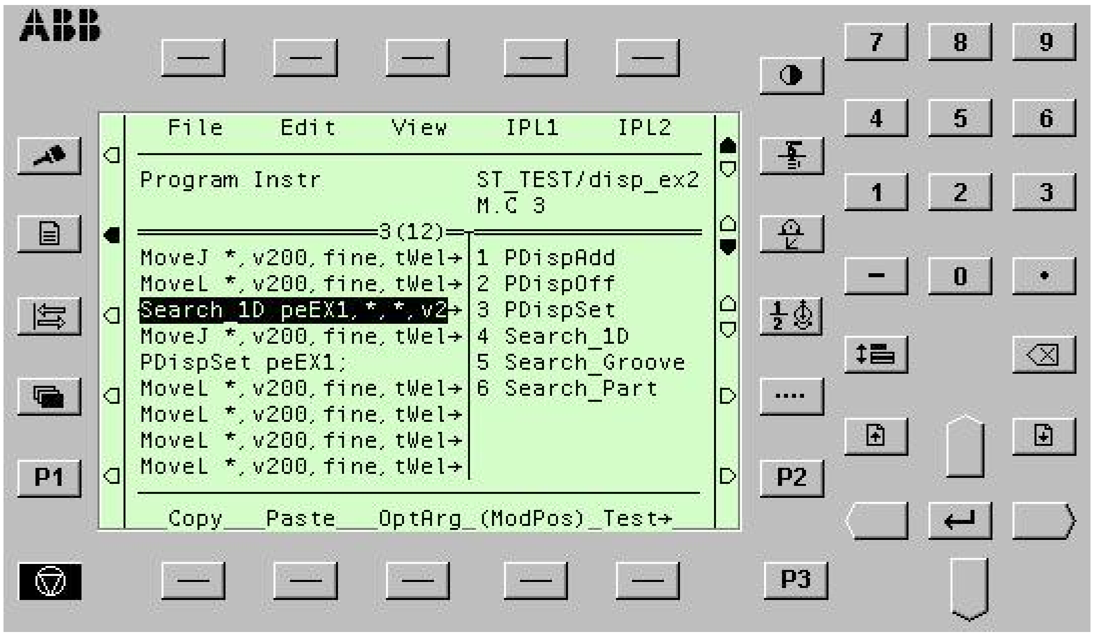


Figure 9.

12. Insert the instruction, *Search_1D*, after the *MoveL*. (If no "Most Common" pick-list was created, use *ProcCall* to find the instruction. When prompted for data, use this data:

Search_1D peEX1,*,*,v200,tWeldGun;

The routine should look like this:



```
Program Instr                ST_TEST/disp_ex2
                             M.C 3
=====================3(12)=====================
MoveJ *,v200,fine,tWel→ 1 PDispAdd
MoveL *,v200,fine,tWel→ 2 PDispOff
Search_1D peEX1,*,*,v2→ 3 PDispSet
MoveJ *,v200,fine,tWel→ 4 Search_1D
PDispSet peEX1;         5 Search_Groove
MoveL *,v200,fine,tWel→ 6 Search_Part
MoveL *,v200,fine,tWel→
MoveL *,v200,fine,tWel→
MoveL *,v200,fine,tWel→
```

13. Jog the robot to the *SearchPoint* and "ModPos" the *second* robtarget in the *Search_1D* instruction. The gas cup should make contact with the plate without deflecting the torch.

14. Jog the robot to the StartPoint and "ModPos" the first robtarget in the Search_1D instruction.

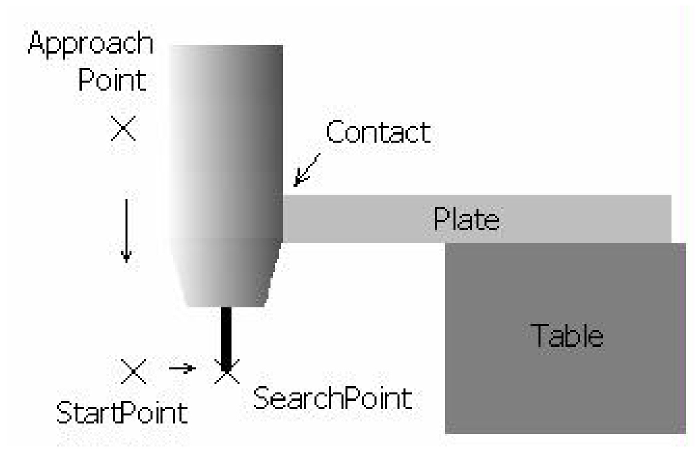


Figure 10.

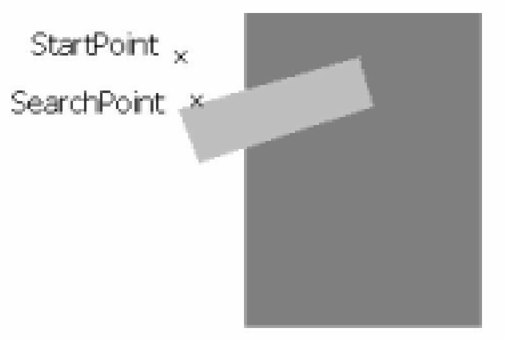


Figure 11. "Top View"

15. Toggle the Program Window to the test mode and move the Program Pointer to the *Search_1D* instruction.

16. Enable the robot and hit the Fwd key. The robot should move to the *StartPoint*, then search towards the *SearchPoint*, until the plate is detected. (See Figure ex2.c)

17. Toggle the Program Window to *instruction* mode and using the Copy and Paste keys, copy the *MoveL* ahead of the *Search_1D* and paste it after the *Search_1D*. Your final routine, *disp_ex2*, should look like this:

```
PROC disp_ex2()
    MoveJ *,v200,fine,tWeldGun;
    MoveL *,v200,fine,tWeldGun;
    Search_1D peEX1,*,*,v200,tWeldGun;
    MoveL *,v200,fine,tWeldGun;
    MoveJ *,v200,fine,tWeldGun;
    PDispSet peEX1;
    MoveL *,v200,fine,tWeldGun;
    MoveL *,v200,fine,tWeldGun;
    MoveL *,v200,fine,tWeldGun;
    MoveL *,v200,fine,tWeldGun;
    MoveL *,v200,fine,tWeldGun;
    PDispOff;
    MoveL *,v200,fine,tWeldGun;
ENDPROC
```

18. Run the routine from the beginning. The torch should search the plate and then trace out the plate.

19. Move the plate about 10mm away from the SearchPoint and try running the routine (See Figure ex2.e). If the plate was moved in the direction of the search, without any rotation, the torch should still trace out the plate correctly.

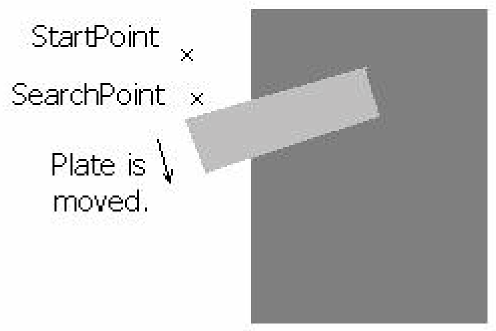


Figure 12.

Questions:

1. Look at the data in peEX1. How does it change after searching different locations?

2. What happens when the plate is moved in other directions?

Advanced:

3. What happens when the search is programmed so that the search direction is not perpendicular to the plate's edge?

4. What errors occur when the plate is moved too far away? Experiment with

the error recovery options to see what they do. Consult Section 8.1 "Instructions" in this manual to see detailed explanations of the error handing capabilities.

5.4.2 Programming Tips

1. Remember that the direction of the search dictates the direction that the resulting program displacement can shift a program.

2. You should almost always search perpendicular to the part feature surface. The search accuracy will suffer if the search direction is at an angle to the feature surface.

3. For a newly programmed search try executing the search using the Forward Step key. When the robot stops motion with the torch touching the part, move the cursor to the *SearchPoint* and "mod-pos" the robtarget. This ensures that a search on a "perfect" part will return a displacement that is very close to zero.

5.5 Using SmarTac for Multi-Dimensional Searching.

As seen in Example 2, a one-dimensional search will determine where a weld seam is if it is constrained to move in only one direction. In some cases this is adequate. More likely, though, a two or three-dimensional search is required. A two-dimensional search would provide information about where a plate is located on a table, for example. A three-dimensional search would also determine how high the table surface was.

In Example 2 you may have noticed that if the plate was rotated when moved, the Displacement Frame would not compensate for the rotation. This is the limitation of single and multi-dimensional searching. These search techniques are relatively easy to master and, despite the limitation, provide accurate search information about the weld seam when used correctly.

To search a part in more than one direction, a combination of one-dimensional searches is used and the result of each search is added together. Example 3 demonstrates this for a two-dimensional search.

5.5.1 Exercise 3 Two Dimensional Search

In this exercise the Search_1D instruction will be used twice to determine a two-dimensional shift in a plate on a table.

1. View the routines in the module, ST_TEST.

2. Highlight *disp_ex2* and duplicate it. Name the new routine *disp_ex3*.

3. Toggle the Program Window to *test* mode.

4. Move the Program Pointer to the new procedure, *disp_ex3*.

5. Move the Program Pointer to the *PDispOff* instruction near the end of the routine.

6. Enable the robot and step forward once to execute this instruction.

7. Make sure the robot can move to the first MoveL that traces out the plate, then move the Program Pointer to this MoveL.

8. Enable the robot and step forward once. Align the plate to the torch tip. Step though the rest of the points to get the plate back to where it was when we first wrote the routine.

9. Move the cursor to the top of the routine, enable the robot, and step forward until the search is complete, and the robot stops at the *MoveL* immediately following the *Search_1D* instruction.

10. Add another *MoveL* here. Its location should be off the end of the plate. You are going to add another search to this routine that will search the end of the plate. This move will provide safe passage.

11. Copy the last *Search_1D* and insert it after the *MoveL* created in step #9.

12. Copy the *MoveL* created in step #9 and insert it after the last *Search_1D*.

13. The routine should now look like this:

```
PROC disp_ex3()
    MoveJ *,v200,fine,tWeldGun;
    MoveL *,v200,fine,tWeldGun;
    Search_1D peEX1,*,*,v200,tWeldGun;
    MoveL *,v200,fine,tWeldGun;
    MoveL *,v200,fine,tWeldGun;  ← New MoveL from step #9
    Search_1D peEX1,*,*,v200,tWeldGun;   ← New Search_1D
    MoveL *,v200,fine,tWeldGun;  ← Copy of MoveL from step #9
    MoveJ *,v200,fine,tWeldGun;
    PDispSet peEX1;
    MoveL *,v200,fine,tWeldGun;
    MoveL *,v200,fine,tWeldGun;
    MoveL *,v200,fine,tWeldGun;
    MoveL *,v200,fine,tWeldGun;
    MoveL *,v200,fine,tWeldGun;
    PDispOff;
    MoveL *,v200,fine,tWeldGun;
ENDPROC
```

14. Modify the robtargets in the new Search_1D to search the end of the plate. The new search will be referred to as "search 2":

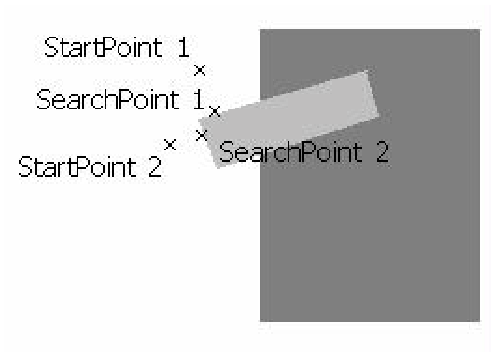
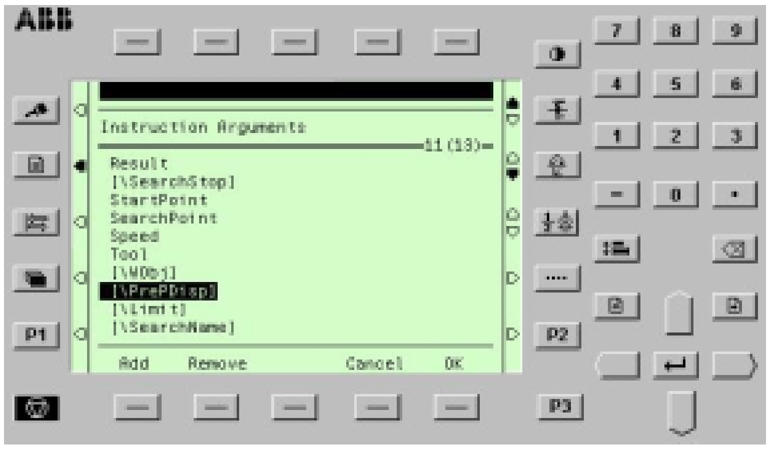


Figure 13.

15. Highlight the second *Search_1D* instruction. And hit the Enter key.

16. Press the OptArg key to look at the optional arguments.

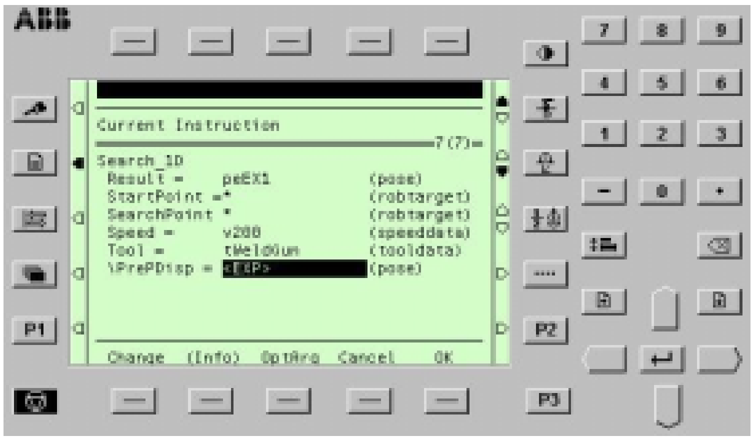
17. Move the cursor down to the optional argument named, [\PrePDisp], and *Add* it.



```
Instruction Arguments
                                   11(13)
 Result
 [\SearchStop]
 StartPoint
 SearchPoint
 Speed
 Tool
 [\WObj]
 [\PrePDisp]
 [\Limit]
 [\SearchName]
```

18. Select OK.

19. Move the cursor to the new argument and hit enter:



```
Current Instruction
                                                  7(7)
Search_1D
  Result =       peEX1            (pose)
  StartPoint =*                   (robtarget)
  SearchPoint *                   (robtarget)
  Speed =        v200             (speeddata)
  Tool =         tWeldGun         (tooldata)
  \PrePDisp =    <EXP>            (pose)

  Change  (Info)  OptArg  Cancel     OK
```

20. From the list of available pose data select *peEX1*. Press OK. The routine should look like this:

```
PROC disp_ex3()
    MoveJ *,v200,fine,tWeldGun;
    MoveL *,v200,fine,tWeldGun;
    Search_1D peEX1,*,*,v200,tWeldGun;
    MoveL *,v200,fine,tWeldGun;
    MoveL *,v200,fine,tWeldGun;
    Search_1D peEX1,*,*,v200,tWeldGun\PrePDisp:=peEX1;
    MoveL *,v200,fine,tWeldGun;
    MoveJ *,v200,fine,tWeldGun;
    PDispSet peEX1;
    MoveL *,v200,fine,tWeldGun;
    MoveL *,v200,fine,tWeldGun;
    MoveL *,v200,fine,tWeldGun;
    MoveL *,v200,fine,tWeldGun;
    MoveL *,v200,fine,tWeldGun;
    PDispOff;
    MoveL *,v200,fine,tWeldGun;
ENDPROC
```

21. Jog the torch so that it is above the plate and execute the routine from the beginning. The torch should trace out the plate.

22. Move the plate about 10mm in any direction and re-execute the routine. The torch should trace out the plate.

Questions:

1. Why is it necessary that the PrePDisp be set to *peEX1* in this example? What happens when a different displacement frame (other than peEX1) is used in the first *Search_1D*?

2. What happens when this optional argument in the second *Search_1D* is not present?

3. Two and three-dimensional searches should almost always use search direc-

tions that are perpendicular to one another. Why?

Advanced:

4. What happens when the plate is rotated slightly? Why?

5. Add the optional argument, [\NotOff], to the first search instruction. And execute the program. What does this do? Hint: Look at the Search_1D section of section 8.1, “Instructions”.

6. What would happen if the [\NotOff] argument was added to the second search and the next section of the routine had a welding instruction? Hint: Look in section 8.1.

7. *Version 7.0 only:* Why must there always be at least one Move instruction between two searches? Hint: What happens when SmarTac is activated while the torch is touching the part?

8. If there is time try to write a three-dimensional search. Do not corrupt disp_ex3, as it will be used later. The 3-D search should look something like this:

```
PROC disp_ex3_3D()
    MoveJ *,v200,fine,tWeldGun;
    MoveL *,v200,fine,tWeldGun;
    Search_1D peEX1,*,*,v200,tWeldGun;
    MoveL *,v200,fine,tWeldGun;
    MoveL *,v200,fine,tWeldGun;
    Search_1D peEX1,*,*,v200,tWeldGun\PrePDisp:=peEX1;
    MoveL *,v200,fine,tWeldGun;
    MoveL *,v200,fine,tWeldGun;
    MoveL *,v200,fine,tWeldGun;
    Search_1D peEX1,*,*,v200,tWeldGun\PrePDisp:=peEX1;
    MoveL *,v200,fine,tWeldGun;
    MoveJ *,v200,fine,tWeldGun;
    PDispSet peEX1;
    MoveL *,v200,fine,tWeldGun;
    MoveL *,v200,fine,tWeldGun;
    MoveL *,v200,fine,tWeldGun;
    MoveL *,v200,fine,tWeldGun;
    MoveL *,v200,fine,tWeldGun;
    PDispOff;
    MoveL *,v200,fine,tWeldGun;
ENDPROC
```

Your routine may have more or less moves to re-orient the torch when searching in the vertical direction:

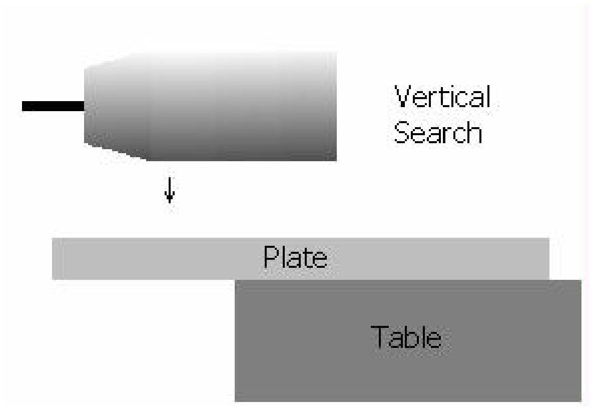
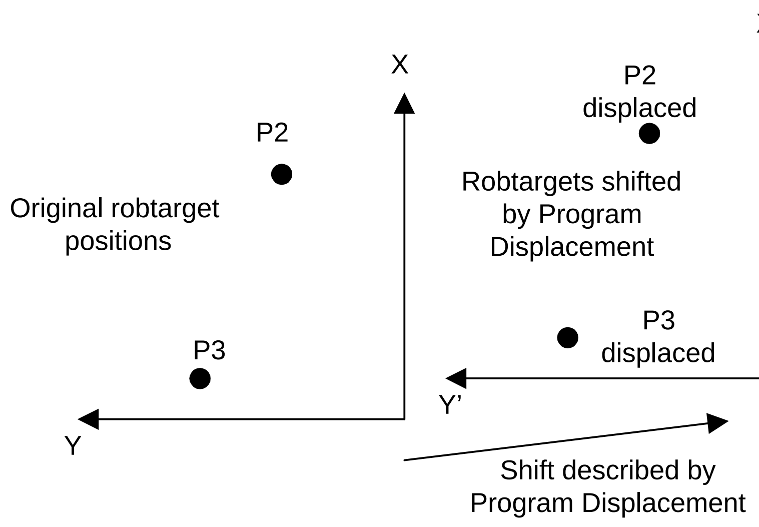


Figure 14.

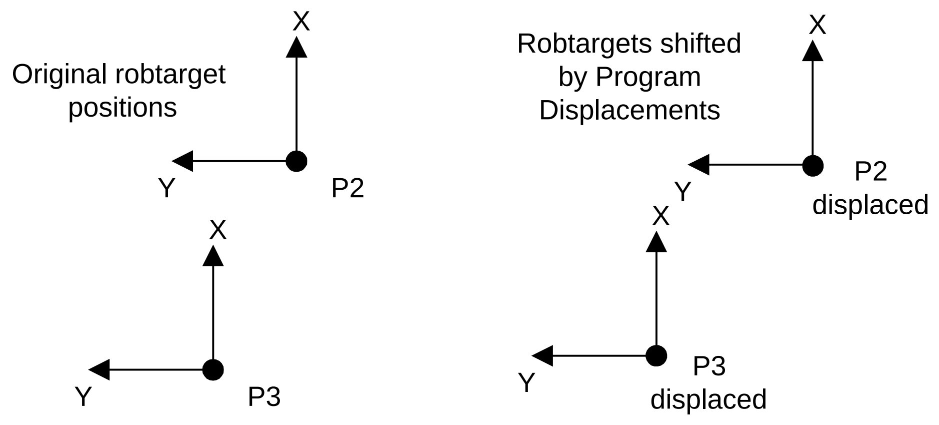
9. In Example 1, what happens if you search the same edge twice, using *PreP-Disp* to add the second search result to the first?

5.6 Using SmarTac to Determine Simple Rotational Changes.

Up to this point basic one dimensional searches have been used to accurately locate part features that have moved in only in translation, not rotation:



Using this same basic concept, it is possible to search a weld seam that moves both in translation and rotation. Imagine that the robtargets in the above sketch, P2 and P3, describe the ArcL\On and ArcL\Off of a weld. If each robtarget is represented by a different Program Displacement then the weld seam can be moved rotation-ally as well:


Original robtarget positions
X
Y
P2
X
Y
P3
Robtargets shifted by Program Displacements
X
Y
P2 displaced
X
Y
P3 displaced

To do this for a real weld seam, the robtargets, P2 and P3, will have to be searched separately and the displacement data stored in two different pose data elements. Example 4 illustrates this technique.

5.6.1 Exercise 4 Part feature with simple rotation.

In this example a simple path with two points will be moved in translation as well as in rotation.

1. Create a new routine called, *disp_ex4*, that looks like this:

```
PROC disp_ex4()
   MoveJ *,v200,fine,tWeldGun;
   MoveL *,v200,fine,tWeldGun;
   Search_1D peEX1,*,*,v200,tWeldGun;                    🡐Search 1
   MoveL *,v200,fine,tWeldGun;
   MoveL *,v200,fine,tWeldGun;
   Search_1D peEX2,*,*,v200,tWeldGun\PrePDisp:=peEX1;  🡐Search 2
   MoveL *,v200,fine,tWeldGun;
   MoveL *,v200,fine,tWeldGun;
   Search_1D peEX3,*,*,v200,tWeldGun\PrePDisp:=peEX1; 🡐Search 3
   MoveL *,v200,fine,tWeldGun;
   MoveL *,v200,fine,tWeldGun;
   PDispSet peEX2;
   MoveL P1,v200,fine,tWeldGun;                          🡐Corner 1
   PDispSet peEX3;
   MoveL P2,v20,fine,tWeldGun;                           🡐Corner 2
   PDispOff;
   MoveL *,v200,fine,tWeldGun;
ENDPROC
```

Three displacement frames will be used, peEX1, peEX2, and peEX3. You will need to create these if they do not exist in the system. Be careful that the correct per-displacement is called for each search instruction.

The searches should look similar to those used in Example 2 & 3 (See Figure ex2.c), but positioned around the plate like this:

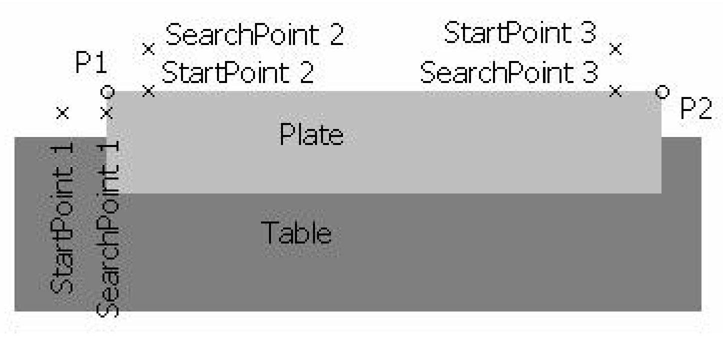


Figure 15.

For the best accuracy, put searches near the corners.

2. Modify P1 and P2 to be at the corners of the plate, as shown above. You will have to create the named robtargets, P1 and P2, if they do not exist in the system. It is not necessary that these two points be named for the test to work. They are named here as a teaching aid only.

3. Modify the "air moves" to clear the plate.

4. Step through the routine to check the positions. If P1 and P2 are out of position, you can jog them into position and mod-pos them with the program displacement turned on.

5. Execute the program from the beginning. And watch the robot trace the edge of the plate.

6. Move the plate in various directions, including rotationally, and execute the routine each time. Does the robot torch follow edge each time? If it does not, there check the program again to be sure all the correct displacement frames are in the right places.

Questions:

1. How is the usage of PrePDisp different from that in Example 3?

2. Look at the values in each of the program displacements, peEX1, peEX2, and peEX3. What the values for the rotation portions, q1…q4?

Advanced:

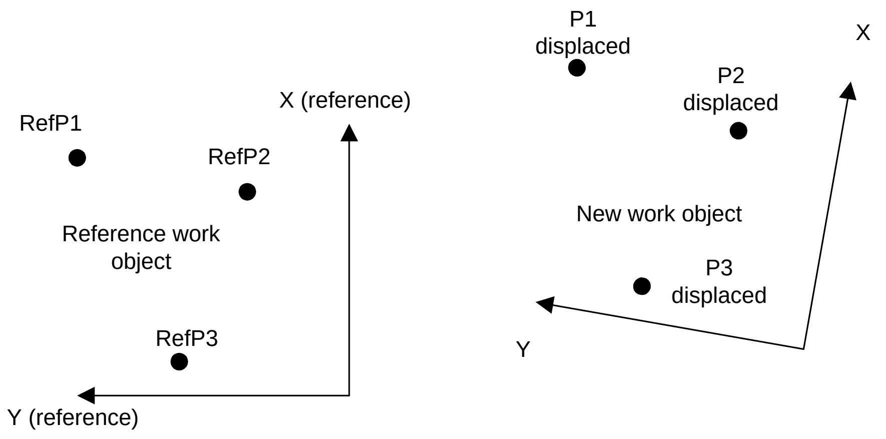
3. When the plate is rotated significantly, do you see any error in the positioning of P1 and P2? Why will large rotations of the plate cause some error in this example?

4. Why would it be difficult to shift an intermittent "stitch" weld in this way?

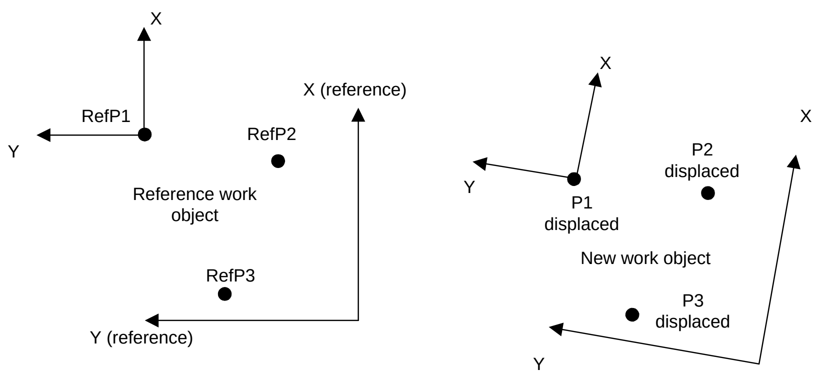
5.7 Using SmarTac with Work Object Manipulation.

Sometimes using multiple displacement frames can not provide an easy way of determining a weld seam's location. In Example 4 we proved that a simple weld path could be moved in translation and rotation using two displacement frames; one for the start and one for the end of the weld path. If the weld were not continuous, i.e. a stitch weld, this would not work. There is no displacement information about the intermediate weld points.

In some instances it is necessary to determine how the *whole* part has moved in translation and rotation. The best way to do this is to use a *work object* to describe where the part is in relation to the *world frame*. Based on search information, the object frame of a work object can be moved in translation and rotation. If the weld sequence in written in this *work object*, the points in the sequence will move with changes to the *work object*.



An important benefit to this technique is that searching and program displacements can still be used for features on the part after the part program has been rotated in the work object.



In this image, robtargets P1, P2, and P3 all move with the work object. In addition, P1 moves with a program displacement frame relative to that work object.

The SmarTac module contains two mathematical functions that can be used in conjunction with the Search_1D instruction to make this searching technique easier.

5.7.1 SmarTac Functions

Two global mathematical functions are provided in the SmarTac module:

PoseAdd

OFrameChange

PoseAdd is a simple function used to add the *trans* portion of two or three displacement frames. The function returns *pose* data. In use it looks like this:

```
peSUM:=PoseAdd(peFIRST,peSECOND);
```

Using *PoseAdd* is similar to using the optional argument *PrePDisp* in a Search_1D.

The second function is *OFrameChange*. This function uses seven arguments, a reference work object, three reference points, and three displacement frames. The function returns *wobjdata*. In use it looks like this:

```
obNEW:=OFrameChange(obREF, p1,p2,p3,pe1,pe2,pe3);
```

Detailed explanations of each of these functions can be found in section 8.2, "Functions."

Exercise 5 will illustrate the usage of these mathematical tools.

5.7.2 Exercise 5 Object Frame Manipulation.

In this exercise a two dimensional example will be used, as in Exercise 4. There will be four searches for this technique, so the plate will have to be clamped so that most of the plate is off the table:

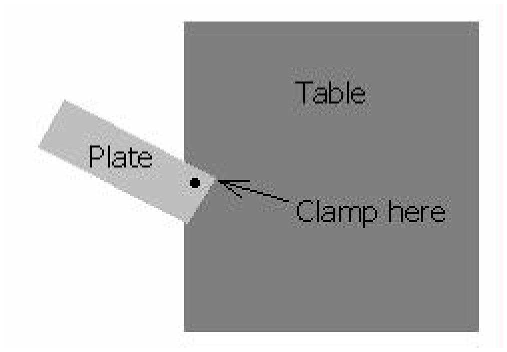


Figure 16.

Refer to this sketch when laying out the points for this exercise:

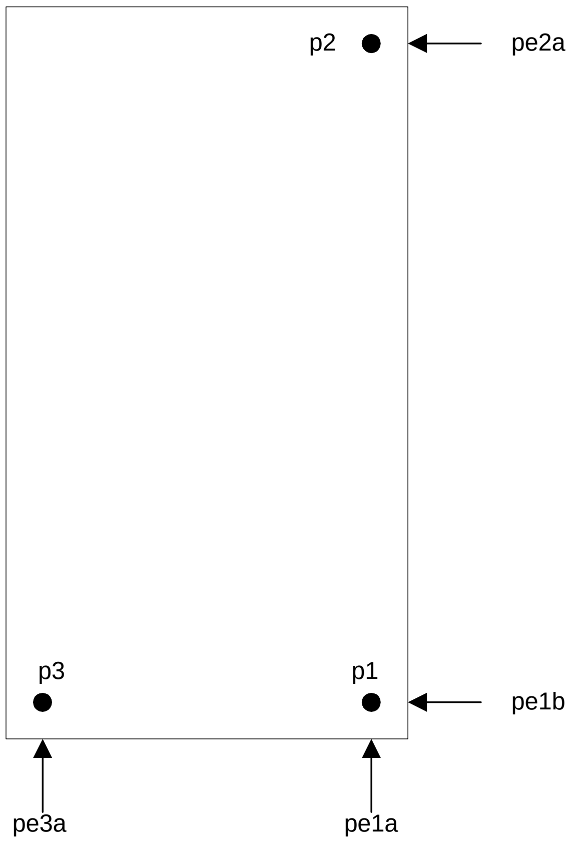
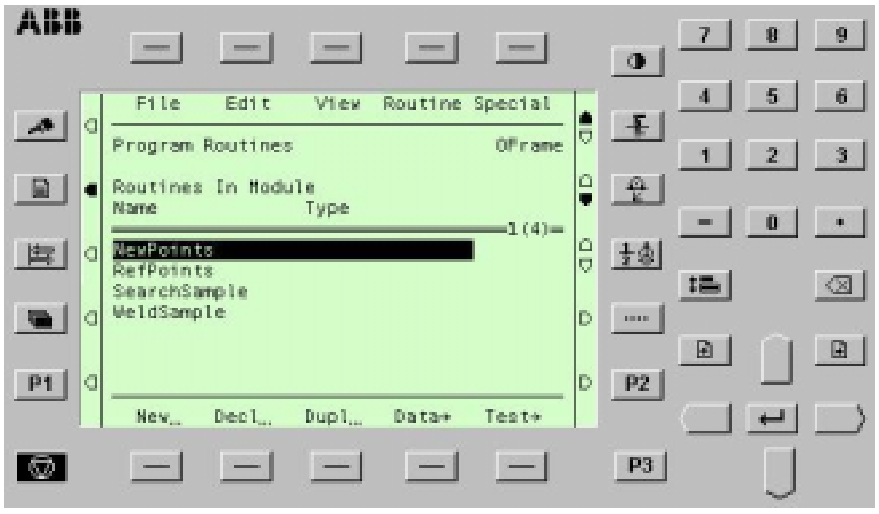


Figure 17.

1.Load the program module, "OFrame", from the provided disk. The disk has been provided to speed up this exercise. If you don't have the disk, you will have to write the routines from scratch using the teach pendant. A printout can be found in section 8.3 of this manual.

2.View the routines in the module. You should see the following routines:



```
File    Edit    View   Routine  Special
Program Routines                 OFrame
Routines In Module
Name               Type
                                1(4)
NewPoints
RefPoints
SearchSample
WeldSample
New...  Decl...  Dupl...  Data+   Test+
```

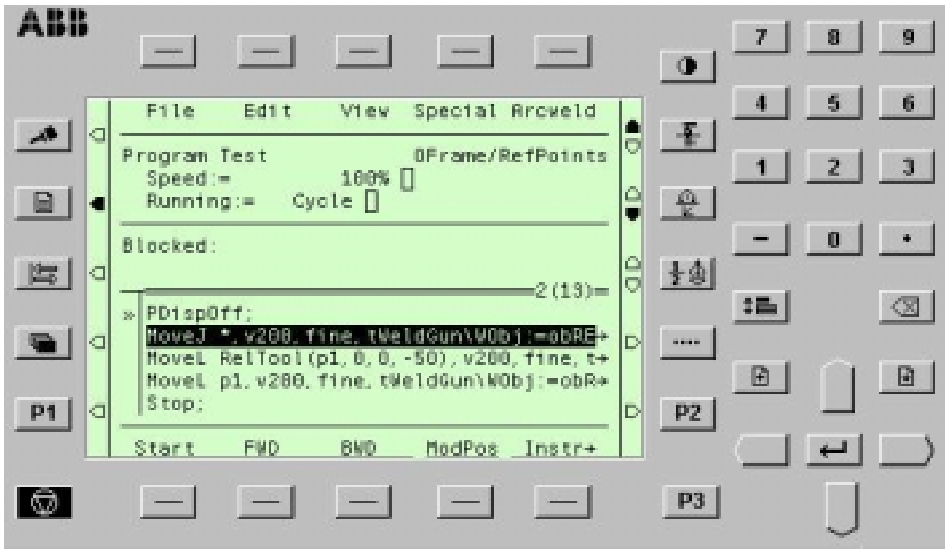
3. Mark three points on the surface of the plate and label them p1, p2, and p3. The location of the points is not critical, but they should be near the corners as shown in Figure ex5.b & ex5.c.

4. Move the program pointer to the procedure called, "RefPoints". *RefPoints* is a routine that, once updated, will point out the three reference points.

5. Execute the instruction: *PDispOff* at the beginning of the procedure using the Forward Step key. (This ensures that no displacements are active.)

6. Jog the robot so that the torch is situated about 150mm above the plate surface and pointing down at the plate.

7. Move the cursor to the first *MoveJ* and modify the position. You may have to change your settings in the Jog Window to reflect the work object change.



```
Program Test                 OFrame/RefPoints
  Speed:=          100%
  Running:=   Cycle

Blocked:

                                        2(13)
» PDispOff;
  MoveJ *,v200,fine,tWeldGun\WObj:=obRE→
  MoveL RelTool(p1,0,0,-50),v200,fine,t→
  MoveL p1,v200,fine,tWeldGun\WObj:=obR→
  Stop;
```

8. Jog the robot so that the torch TCP is just touching the *p1* mark, and modify the second *MoveL*: MoveL p1,v200,fine,tWeldGun\WObj:=obREF;

9. Jog the robot to the *p2* mark and modify the *MoveL*: MoveL p2,v200…

10. Jog the robot to the *p3* mark and modify the *MoveL*: MoveL p3,v200…

11. Jog the robot so the torch is about 150mm above the plate and modify the last *MoveJ*.

12. Move the Program Pointer to the beginning of the routine, and start execution. The robot should go from point to point with the torch TCP, stopping at each point so that the position can be checked. If any positions need to be changed, change them now.

13. Move the Program Pointer to the procedure labeled: "SearchSample".

14. Jog the robot so that the torch TCP is above the plate and off the corner where *p1* is. Modify the first *MoveJ*.

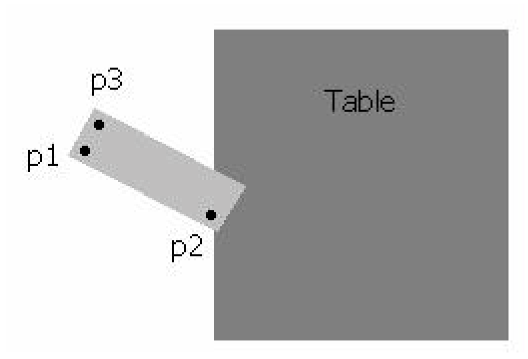


Figure 18.

Jog the robot down so that the torch gas cup is in a position to search the edge of the plate. Refer to Figure ex5.b and modify the points the first *Search_1D*. The search direction is indicated in Figure ex5.b for the displacement frame pe1a (the first search result). Remember that the search direction should be perpendicular to the edge of the part.

Modify all the rest of the moves and searches as prescribed by Figure ex5.b.

Test run the *SearchSample* procedure.

Move the Program Pointer to the routine called: “WeldSample”. *WeldSample* does not have any *ArcL* instructions so that ArcWare need not be present to load this module. It has only *MoveL* instructions with slow speeds to simulate welding.

Draw a simulated weld on the surface of the plate using a straight edge and marker (See Figure ex5.d). *WeldSample* has only two segments, you may add more if you desire.

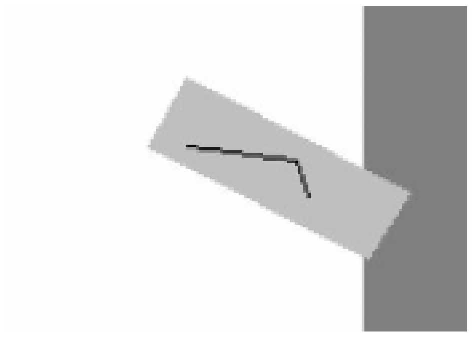


Figure 19.

15. Modify the first point to be above the plate at least 100mm.

16. Modify the second point to be the start of the simulated weld.

17. Modify the third and fourth points to be the middle and end of the weld.

18. Modify the last point to be above the plate at least 100mm.
19. Run *WeldSample* to be sure it follows the line correctly.
20. Run *SearchSample*.
21. Run *NewPoints*. The points p1, p2, and p3 should be pointed out correctly. If not, there is a mistake somewhere. Check your program.
22. Run *WeldSample* again to be sure everything is ok. If the path is not followed, check the program again.
23. Leaving the plate clamped at the corner, move the plate about 10mm at the end.
24. Run *SearchSample*.
25. Run *WeldSample*. The path should follow correctly.
26. Run *NewPoints*. The points should be pointed out correctly.

Questions:

1. What work object is used in *RefPoints* and *SearchSample*?
2. What work object(s) is(are) used in *NewPoints* and *WeldSample*?
3. Is *PDispSet* used in this exercise? Why, or why not?

Look at the last several lines of *SearchSample*:

4. pe1 is the combination of what two searches?
5. pe2 is the combination of what two searches?
6. pe3 is the combination of what two searches?
7. For the best accuracy, there should be two searches for each reference point, located close to each reference point. In this exercise we use only four searches to approximate this. How far do you have to rotate the plate before you notice the inaccuracy?
8. Why does this occur?
9. Would this be a concern for most real-world fixtures?

Advanced:

- Define the object frame of obREF so that the origin is at the corner of the plate where p1 is. Align the object frame with the plate.
- Select obREF as the work object and WObj for the Coordinate System in the Jog Window. You should be able to jog along the edges of the skewed plate with straight deflections of the JoyStick. If not, the object frame was not defined properly.

- Go though Example 5 again with the new work object definition.

10. How might this help when programming?

5.8 Search_Part.

Sometimes it is necessary to search a part feature to determine if it is there or not. Information like this can be used to determine what type of part is present, or if a part is loaded at all. The SmarTac instruction, *Search_Part* is provided for this use.

Search_Part is programmed very much like a *Search_1D* instruction, but it returns a Boolean instead of a Program Displacement. In use it looks like this:

```
Search_Part bPresent,p1,p2,v200,tWeldGun;
```

The robot moves on a path from p1 through p2. If contact is made with the part feature, the Boolean, bPresent, is set to TRUE. If no contact is made, it is set to FALSE.
In this example a weld procedure is selected based on the presence of a particular part feature:

```
PROC Which_Part()
  MoveJ *,v200,z10, tWeldGun;
  MoveJ *,v200,fine, tWeldGun;
  Search_Part bPresent,p1,p2,v200,tWeldGun;
   IF bPresent THEN
    Big_Part;
  ELSE
    Small_Part;
  ENDIF
ENDPROC
```

5.8.1 Exercise 6 Using Search_Part.

1. Create a new procedure in the module, *ST_TEST*. Call it *disp_ex6*.
2. You need only one instruction in this procedure, *Search_Part*. You will have to create Boolean to use as your result. The robtargets need not be named. Search for the edge of the plate such that you can take the plate away later.
3. Run the routine to be sure it works OK.

Questions:

1. With the plate in place, what is the value of the Boolean after searching?
2. With the plate removed, what is the value of the Boolean after searching?

Advanced:

3. What happens when you move the plate so that it is touching the gas cup at the start of the search?

5.9 Conclusion

This overview provides most of the techniques required to use SmarTac searching on the majority of real-world weldments. A number of optional arguments for the search instructions have not been explained here. For more information about these, as well as more examples, see section 8.1 “Instructions”. You will also find an instruction called *PDispAdd* which is used with the same effect as the function *PoseAdd*.

Work objects were discussed briefly throughout this manual. Please consult the Controller User’s Manual for any questions about how to use work objects to simplify programming. Especially for users with coordinated work objects on positioning equipment, a firm understanding of work object user and object frames is critical to writing good weld routines.

6 Software Reference

6.1 Instructions

Search_1D One-dimensional search.

Search_1D is an instruction used for tactilely searching a feature with SmarTac. The search path is described by two required robtargets. The search result is stored as pose data in the required argument ‘Result’. All SmarTac board activation and deactivation is automatically handled.

Example

Search_1D peOffset,p1,p2,v200,tWeldGun;

The robot moves on a path from p1 through p2. When contact is made with the part feature, the difference between the contact location and p2 is stored in peOffset.

Arguments

Search_1D [\NotOff] Result [\SearchStop] StartPoint SearchPoint Speed Tool [\WObj] [\PrePDisp] [\Limit] [\SearchName]

[\NotOff] **Data type: *switch***

If selected, the welding positive lead secondary contact (break box) remains open at the end of the search. Additionally, the SmarTac board remains activated after the search ends. If this switch is selected directly before a welding instruction, welding current will not reach the torch.

Result **Data type: *pose***

The displacement frame that will be updated

[\SearchStop] **Data type: *robtarget***

If selected, this robtarget will be updated as the point where the robot detects the part feature.

StartPoint **Data type: *robtarget***

The start point of the search motion.

SearchPoint **Data type: *robtarget***

The point where the robot expects to touch the part. This robtarget is programmed so that the torch is touching the surface of the part feature.

Speed **Data type: *speeddata***

The speed data used when moving to the *StartPoint*. The velocity of the search motion is unaffected.

Tool **Data type: *tooldata***

The tool used during the search.

[\WObj **Data type: *wobjdata***

The work object used during the search. *WObj* determines what frame *Result* will be related to. If not selected, wobj0 is used.

[\PrePDisp] **Data type: *pose***

If selected, the search will be conducted with this displacement frame active, effectively adding the two displacement frames. This may or may not be the same as the pose data selected for *Result*.

[\Limit] **Data type: *num***

If selected, an error will be flagged if the magnitude of the search result, *Result*, is larger than the value entered for the *Limit*. (millimeters)

[\SearchName] **Data type: *string***

If selected, the search will be assigned this identifying name. The name will accompany any error messages that are written to the *User Error Log*.

Program execution

When executed, the robot makes an ‘L’ move to the start point, *StartPoint*. The SmarTac board is activated and motion starts towards the search point, *SearchPoint*. The robot will continue past the search point for a total search distance described by twice the distance between *StartPoint* and *SearchPoint*. Once the part feature is sensed, motion stops, and the displacement data, *Result*, is stored. This program displacement can later be used to shift programmed points using the RAPID instruction *PDispSet*.

Limitations

If the switch, *NotOff*, is selected, the welding positive lead secondary contact (break box) remains open at the end of the search. If this switch is selected directly before a welding instruction, welding current will not reach the torch and an Arc Ignition Error will occur.

Fault management

If an error occurs when activating the SmarTac board, a menu will appear with the following prompts:Activation of the SmarTac failed

RETRY:	Tries to search again with start point moved 50%
RETURN:	Continues the program with default search result
RAISE:	Sends error to calling routine.

When *RETRY* is selected the start point of the search is shifted farther from the part feature. This may give a good search result in cases where the part feature is unusually close and the torch is touching the part feature at the beginning of a normal search.
When *RETURN* is selected a default search result is used which will include any pre-offset included in the search instruction. A message will be logged in the *User Error Log*.

If an error occurs during the search process, a menu will appear with the following prompts:

Search failed

RETRY:	Tries to search again with end point moved 50%
RETURN:	Continues the program with default search result
RAISE:	Sends error to calling routine.

When *RETRY* is selected the end point of the search is shifted farther into the part feature. This may give a correct search in cases where the part feature is unusually far away from the search.
When *RETURN* is selected a default search result is used which will include any pre-offset included in the search instruction. A message will be logged in the *User Error Log*.

If the torch makes contact with the part before the search begins, the following menu appears:

RETRY:	Tries to search again with start point moved 50%
RETURN:	Continues the program with default search result
RAISE:	Sends error to calling routine.

When *RETRY* is selected the start point of the search is shifted farther from the part feature. This may give a good search result in cases where the part feature is unusually close and the torch is touching the part feature at the beginning of a normal search.
When *RETURN* is selected a default search result is used which will include any pre-offset included in the search instruction. A message will be logged in the *User Error Log*.

If the optional argument *Limit* is selected and the magnitude of *peResult* is larger than the value entered for the *Limit*, the following message appears:

```
The search result is outside spec.
Offset:=  [12.012,3.002,-5.013]
The magnitude of the offset:=  13.34
The preset limit:=  10

OK:           Continue with program execution.
RAISE:        Sends the error to calling routine.
```

When *OK* is selected the search result is accepted regardless of magnitude. A message will be logged in the *User Error Log*.

Examples

Single dimension search in any direction:

```
MoveJ *, vmax,fine, tWeldGun;
Search_1D peOffset,p1,p2,v200,tWeldGun
PDispSet peOffset;ArcL\On,*,vmax,sm1,wd1,wv1,z1,tWeldGun;
ArcL\Off,*, vmax,sm1,wd1,wv1,z1,tWeldGun;
MoveJ *, vmax,z10, tWeldGun;
ArcL\On,*,vmax,sm1,wd1,wv1,z1,tWeldGun;
ArcL\Off,*, vmax,sm1,wd1,wv1,z1,tWeldGun;
PDispOff;
```

Two dimension searching in any direction in a defined work object:

```
MoveJ *, vmax,fine, tWeldGun\WObj:= wobj2;
Search_1D\NotOff, pose1, p1,p2,v200,tWeldGun\WObj:=wobj2;
Search_1D pose1,p3,p4,v200,tWeldGun\WObj:=wobj2\PrePDisp:= pose1;PDisp-
Set pose1;
ArcL\On,*,vmax,sm1,wd1,wv1,z1,tWeldGun\Wobj:= wobj2;
ArcL\Off,*, vmax,sm1,wd1,wv1,z1,tWeldGun\Wobj:= wobj2;
MoveJ *, vmax, z10, tWeldGun\WObj:= wobj2;
ArcL\On,*,vmax,sm1,wd1,wv1,z1,tWeldGun\Wobj:= wobj2;
ArcL\Off,*, vmax,sm1,wd1,wv1,z1,tWeldGun\Wobj:= wobj2;
PDispOff;
```

Important: It is typically unproductive to have two searches in the same direction for the same feature. Multiple searches in the same direction using the PrePDisp option will be averaged. Searches for a single feature should almost always be 90 degrees from each other. This fact implies that usually there should never be more than three searches on any one feature.

Other variations:

One dimensional search with the maximum limit set at 4mm. If the magnitude of the trans portion of peOffset is greater than 4mm an error is flagged:

```
Search_1D,peOffset,p1,p2,v200,tWeldGun\Limit:=4;
```

One dimensional search with the gas cup. The robtarget p3 is updated with the actual search position:

```
Search_1D\SearchStop:=p3,pose1,p1,p2,v200,tWeldGun;
```

One dimensional search with the gas cup. If an error occurs while searching and the operator elects to continue with default results, the name, *First*, will appear along with the error description, in the *User Error Log*. See: *Fault Management* above.

```
Search_1D pose1,p1,p2,v200,tWeldGun\SearchName:=”First”;
```

Syntax

Search_1D

[‘\ ‘ NotOff ‘,’]

[Result ’:=’] < expression (**INOUT**) of *pose* > ’,’

[‘\’ SearchStop ’:=’ < expression (**INOUT**) of *robtarget* >’,’]

[StartPoint ’:=’] < expression (**IN**) of *robtarget* > ’,’

[SearchPoint ’:=’] < expression (**IN**) of *robtarget* > ’,’

[Speed ’:=’] < expression (**IN**) of *speeddata* > ’,’

[Tool ’:=’] < persistent (**PERS**) of *tooldata* >

[‘\’ WObj ’:=’ < persistent (**PERS**) of *wobjdata* >]

[‘\’ PrePDisp‘:=’ < expression (**IN**) of *pose* >]

[‘\’ Limit ‘:=’ < expression (**IN**) of *num* >]

[‘\’ SearchName ‘:=’ < expression (**IN**) of *string* >] ‘;’

Related information

	Described in:
Search_Groove	SmarTac *Instructions*
Search_Part	SmarTac *Instructions*
datatype: *pose*	Rapid Reference *datatypes*
datatype: *wobjdata*	Rapid Reference *datatypes*
datatype: *robtarget*	Rapid Reference *datatypes*

Search_Part — Search for feature presence.

Search_Part is an instruction used for tactilely searching a feature with SmarTac. The search path is described by two required robtargets. If a feature is detected, a required Boolean is set to TRUE, otherwise it is set to FALSE. In either case, program execution continues.

Example

```
Search_Part bPresent,p1,p2,v200,tWeldGun;
```

The robot moves on a path from p1 through p2. If contact is made with the part feature, the Boolean, bPresent, is set to TRUE. If no contact is made, it is set to FALSE.

Arguments

Search_Part [\NotOff] bDetect StartPoint SearchPoint Speed Tool [\WObj]

[\NotOff] Data type: *switch*

If selected, the welding positive lead secondary contact (break box) remains open at the end of the search. Additionally, the SmarTac board remains activated after the search ends. If this switch is selected directly before a welding instruction, welding current will not reach the torch.

bDetect Data type: *bool*

The Boolean that will be updated. TRUE: if the part is sensed, FALSE: if the part is not sensed.

StartPoint Data type: *robtarget*

The start point of the search motion.

SearchPoint Data type: *robtarget*

The point where the robot expects to touch the part. This robtarget is programmed so that the torch is touching the surface of the part feature.

Speed Data type: *speeddata*

The speed data used when moving to the *StartPoint*. The velocity of the search motion is unaffected.

Tool Data type: *tooldata*

The tool used during the search.

[\WObj Data type: *wobjdata*

The work object used during the search. *WObj* determines what frame peResult will be related to. If not selected, *wobj0* is used.

Program execution

When executed, the robot makes an ‘L’ move to the *StartPoint* with the velocity selected in *Speed*. The SmarTac board is activated and motion starts towards the *SearchPoint*. The robot will continue past the search point for a total search distance described by twice the distance between *StartPoint* and *SearchPoint*. If a feature is detected, the required Boolean is set to TRUE, otherwise it is set to FALSE. In either case, program execution continues.

Limitations

If the switch, *NotOff*, is selected, the welding positive lead secondary contact (break box) remains open at the end of the search. If this switch is selected directly before a welding instruction, welding current will not reach the torch and an Arc Ignition Error will occur.

Fault management

If an error occurs during the search process, a menu will appear with the following prompts:

RETRY:	Tries to search again.
DETECT:	Continues the program with detection TRUE
REJECT:	Continues the program with detection FALSE
RAISE:	Sends error to calling routine

If RETRY is selected the robot will move to the *StartPoint*, then to the approach point before searching. When DETECT or REJECT are selected, a message is stored in the *User Error Log*.

Examples

In this example a procedure is selected based on the presence of a particular part feature:

```
PROC Which_Part()
    MoveJ *,v200,z10, tWeldGun;
    MoveJ *,v200,fine, tWeldGun;
    Search_Part bPresent,p1,p2,v200,tWeldGun;
    IF bPresent THEN
        Big_Part;
    ELSE
        Small_Part;
    ENDIF
ENDPROC
```

Other Variations:

Searching with the wire:

```
Search_Part\Wire,bPresent,p1,p2,v200,tWeldGun;
```

Two searches in a work object:

```
Search_Part\NotOff,bPart1,p1,p2,v200,tWeldGun\WObj:=obPart;
Search_Part bPart2,p3,p4,v200,tWeldGun\WObj:=obPart;
```

Syntax

Search_Part
[‘\ ‘ NotOff ‘,’]
[bDetect’:=’] < expression (**INOUT**) of *bool* > ’,
[StartPoint ’:=’] < expression (**IN**) of *robtarget* > ’,’
[SearchPoint ’:=’] < expression (**IN**) of *robtarget* > ’,’
[Speed ’:=’] < expression (**IN**) of *speeddata* > ’,’
[Tool ’:=’] < persistent (**PERS**) of *tooldata* > ’,’
[‘\’ WObj ’:=’ < persistent (**PERS**) of *wobjdata* >] ‘;’

Related information

	Described in:
Search_1D	SmarTac *Instructions*
datatype: *bool*	Rapid Reference *datatypes*

PDispAdd — Add program displacements.

PDispAdd is an instruction used to add a program displacement frame to the current program displacement frame.

Example

PDispAdd pose2;

Pose2 is added to the current displacement frame.

Arguments

PDispAdd Result

Result Data type: *pose*

The displacement frame added to the current program displacement frame.

Program execution

When executed, *Result* is added to the current displacement frame, and the new program displacement frame is activated.

Syntax

```
PDispAdd
      [ Result ':=' ] < expression (IN) of pose > ';'
```

Related information

	Described in:
Search_1D	SmarTac *Instructions*
PoseAdd	SmarTac *Functions*
datatype: *pose*	Rapid Reference *datatypes*

6.2 Functions

PoseAdd — Adds the translation portions of pose data

PoseAdd is a function that requires two or three pose data arguments and returns the summation of the translation portions in pose form. The returned pose data will have the quaternions set to [1,0,0,0].

Example

peSUM:=*PoseAdd* (peFIRST,peSECOND);

peSUM.trans is set equal to peFIRST.trans + peSECOND.trans. The rotational portion of the peSUM is set to [1,0,0,0] by default.

Return value — Data type: *pose*

The displacement frame.

Arguments

PoseAdd (Pose1 Pose2 [\Pose3])

Pose1 Data type: *pose*

Pose data to be added.

Pose2 Data type: *pose*

Pose data to be added.

[\Pose3] Data type: *pose*

Pose data to be added.

Syntax

PoseAdd '('
  [Pose1 ':='] < expression (**IN**) of *pose* > ','
  [Pose2 ':='] < expression (**IN**) of *pose* > ','
  [' \'Pose3 ':=' < expression (**IN**) of *pose* >] ')'

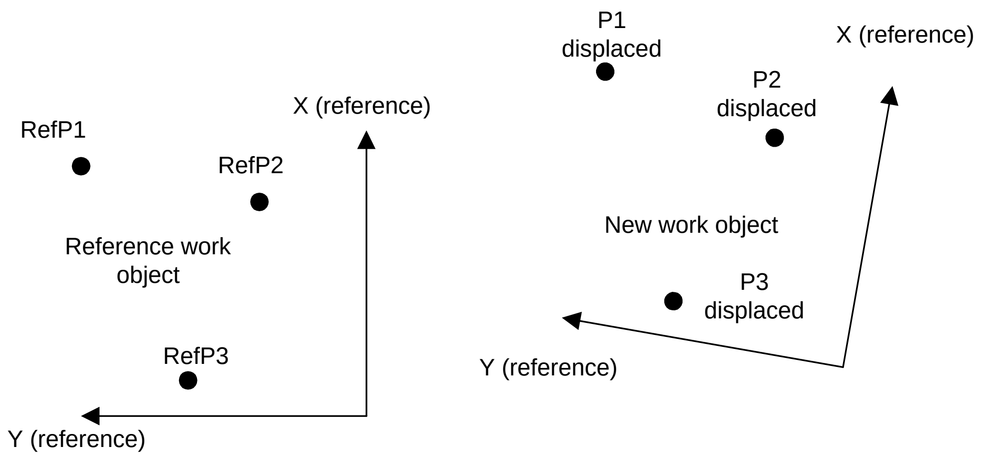
Related information

	Described in:
PDispAdd	SmarTac *Instructions*
datatype: *pose*	Rapid Reference *datatypes*

OFrameChange — Create a new, shifted object frame

OframeChange is a function that returns a work object based on a required reference work object, three reference points, and three corresponding displace-ments,described within the reference work object.

Example



obNEW:=OframeChange(obREF,p1,p2,p3,pe1,pe2,pe3);

The movement of the points *p1, p2,* and *p3* described by displacement frames *pe1, pe2*, and *pe3,* is superimposed over the object frame of the ref-erence work object, *obREF.* The new work object, *obNEW,* has this new object frame and the original user frame information from *obREF.*

Return value — Data type: *wobjdata*

The new work object.

Arguments

OFrameChange (WObj RefP1 RefP2 RefP3 DispP1 DispP2 DispP3)

WObj	Data type: *robtarget*

Reference work object.

RefP1	Data type: *robtarget*

Reference point number one. (Defined in *WObj*)

RefP2	Data type: *robtarget*

Reference point number two. (Defined in *WObj*)

RefP3	Data type: *robtarget*

Reference point number three. (Defined in *WObj*)

DispP1	Data type: *pose*

Displacement frame affecting reference point *RefP1*. (Defined in *WObj*)

DispP2	Data type: *pose*

Displacement frame affecting reference point *RefP2*. (Defined in *WObj*)

DispP3	Data type: *pose*

Displacement frame affecting reference point *RefP3*. (Defined in *WObj*)

Limitations

The reference points can be any three points in space, but they *must* be defined in the reference work object. Similarly, the displacements should be related to the reference work object. The reference points do not have to be the same points as those used in defining the reference work object.

Syntax

OFrameChange ‘(‘
    [WObj ’:=’] < expression (**IN**) of *wobjdata* > ’,’
    [RefP1’:=’] < expression (**IN**) of *robtarget* > ’,’
    [RefP2’:=’] < expression (**IN**) of *robtarget* > ’,’
    [RefP3’:=’] < expression (**IN**) of *robtarget* > ’,’
    [DispP1’:=’] < expression (**IN**) of *pose* > ’,’
    [DispP2’:=’] < expression (**IN**) of *pose* > ’,’
    [DispP3’:=’] < expression (**IN**) of *pose* > ’)’

Related information

	Described in:
PDispAdd	SmarTac *Instructions & Functions*
PoseAdd	SmarTac *Instructions & Functions*
datatype: *pose*	Rapid Reference *datatypes*
datatype: *wobjdata*	Rapid Reference *datatypes*
datatype: *robtarget*	Rapid Reference *datatypes*

6.3 Oframe Module Reference

Exercise 5 uses a program module called "OFrame". The module is included on a floppy disk with the manual. Its purpose is to speed up the training process, whether it be an ABB training course or end-users training themselves. If the disk is not present, use this printout to assist in writing the code. Note: Generic *robtargets* have been reduced to "*" to save space.

```
! Example Module
MODULE OFrame
    PERS wobjdata
    obREF:=[FALSE,TRUE,"",[[0,0,0],[1,0,0,0]],[[0,0,0],[1,0,0,0]]];
    PERS wobjdata
    obNEW:=[FALSE,TRUE,"",[[0,0,0],[1,0,0,0]],[[0,0,0],[1,0,0,0]]];
    PERS robtarget p1:=*;
    PERS robtarget p2:=*;
    PERS robtarget p3:=*;
    PERS pose pe1a:=[[0,0,0],[1,0,0,0]];
    PERS pose pe1b:=[[0,0,0],[1,0,0,0]];
    PERS pose pe2a:=[[0,0,0],[1,0,0,0]];
    PERS pose pe3a:=[[0,0,0],[1,0,0,0]];
    PERS pose pe1:=[[0,0,0],[1,0,0,0]];
    PERS pose pe2:=[[0,0,0],[1,0,0,0]];
    PERS pose pe3:=[[0,0,0],[1,0,0,0]];

 PROC NewPoints()
   PDispOff;
   MoveJ RelTool(p1,0,0,-100),v200,fine,tWeldGun\WObj:=obREF;
   MoveL RelTool(p1,0,0,-50),v200,fine,tWeldGun\WObj:=obNEW;
   MoveL p1,v200,fine,tWeldGun\WObj:=obNEW;
   Stop;
   MoveL RelTool(p2,0,0,-50),v200,fine,tWeldGun\WObj:=obNEW;
   MoveL p2,v200,fine,tWeldGun\WObj:=obNEW;
   Stop;
   MoveL RelTool(p3,0,0,-50),v200,fine,tWeldGun\WObj:=obNEW;
   MoveL p3,v200,fine,tWeldGun\WObj:=obNEW;
   Stop;
   MoveL RelTool(p3,0,0,-50),v200,fine,tWeldGun\WObj:=obNEW;
   MoveJ RelTool(p3,0,0,-100),v200,fine,tWeldGun\WObj:=obREF;
 ENDPROC
```

```
  PROC WeldSample()
    MoveJ *,v200,fine,tWeldGun\WObj:=obNEW;
    !  Simulated weld:
    MoveL *,v200,fine,tWeldGun\WObj:=obNEW;
    MoveL *,v20,z1,tWeldGun\WObj:=obNEW;
    MoveL *,v20,fine,tWeldGun\WObj:=obNEW;
    MoveJ *,v200,fine,tWeldGun\WObj:=obNEW;
  ENDPROC

  PROC SearchSample()
    PDispOff;
    MoveJ *,v200,fine,tWeldGun\WObj:=obREF;
    Search_1D pe1a,*,*,v200,tWeldGun\WObj:=obREF;
    MoveL *,v200,fine,tWeldGun\WObj:=obREF;
    Search_1D pe1b,*,*,v200,tWeldGun\WObj:=obREF;
    MoveL *,v200,fine,tWeldGun\WObj:=obREF;
    Search_1D pe2a,*,*,v200,tWeldGun\WObj:=obREF;
    MoveL *,v200,z10,tWeldGun\WObj:=obREF;
    MoveL *,v200,z10,tWeldGun\WObj:=obREF;
    MoveL *,v200,fine,tWeldGun\WObj:=obREF;
    Search_1D pe3a,*,*,v200,tWeldGun\WObj:=obREF;
    MoveL *,v200,fine,tWeldGun\WObj:=obREF;
    pe1:=PoseAdd(pe1a,pe1b);
    pe2:=PoseAdd(pe1a,pe2a);
    pe3:=PoseAdd(pe1b,pe3a);
    obNEW:=OFrameChange(obREF,p1,p2,p3,pe1,pe2,pe3);
  ENDPROC



  PROC RefPoints()
    PDispOff;
    MoveJ *,v200,fine,tWeldGun\WObj:=obREF;
    MoveL RelTool(p1,0,0,-50),v200,fine,tWeldGun\WObj:=obREF;

    MoveL p1,v200,fine,tWeldGun\WObj:=obREF;
    Stop;
    MoveL RelTool(p2,0,0,-50),v200,fine,tWeldGun\WObj:=obREF;
    MoveL p2,v200,fine,tWeldGun\WObj:=obREF;
    Stop;
    MoveL RelTool(p3,0,0,-50),v200,fine,tWeldGun\WObj:=obREF;
    MoveL p3,v200,fine,tWeldGun\WObj:=obREF;
    Stop;
    MoveL RelTool(p3,0,0,-50),v200,fine,tWeldGun\WObj:=obREF;
    MoveJ *,v200,fine,tWeldGun\WObj:=obREF;
  ENDPROC
ENDMODULE
```

6.4 Mechanical Reference

No mechanical references are included with the SmarTac package.

7 Electrical Reference

SmarTac Reference
Without wire searching capability.
Drawing number: xxxxxxxxxxx

8 Warranty

(a) **EQUIPMENT WARRANTY.** ABB warrants that the equipment shall be free from defects in material and workmanship for the applicable Warranty Period as described below. The Warranty Period shall be either: one year from date of shipment; or if ABB installs the equipment or accepts a standard ABB Robot Installation Report from the installer, for a period of one year from completion of installation, but not to exceed eighteen months from date of shipment; but in any event the Warranty Period shall not exceed 4000 operating hours. Should any failure to conform with the applicable warranty appear during the specified period, ABB shall, if given prompt notice by Buyer, repair, replace, or modify the defective part or parts. New spare parts and refurbished parts shall be subject to the same warranty as original equipment.

Repairs or replacements pursuant to warranty shall not renew or extend the original equipment warranty period; provided, however, that any such repairs or replacements shall be warranted for the time remaining of the original warranty period or 30 days, whichever is longer. ABB’s repair or replacement hereunder shall be exclusive of any removal or installation costs, freight or insurance. ABB shall not be responsible for providing working access to the defect.

(b) **SERVICES.** ABB warrants that the services of its personnel, if provided, will be performed in a workmanlike manner. Should a failure to comply with this warranty appear within six months from the date of completion of such services, ABB shall, if promptly notified in writing, at its option, either provide the services anew or pay Buyer the cost of procuring such services.

(c) **SOFTWARE.** ABB warrants for a period of one (1) year from the date of shipment from ABB that the software furnished under this order will perform in accordance with published or other written specifications prepared, approved, and issued by ABB , when used with specifically identified hardware. In any event, ABB makes no representation or warranty, express or implied, that the operation of the software will be uninterrupted or error free, or that the functions contained in the software will meet or satisfy the Buyer’s intended use or requirements.

(d) **CONDITIONS OF WARRANTY.** This warranty shall not apply to any equipment or parts which (i) have been improperly installed, repaired or altered, (ii) have been subjected to misuse, negligence or accident, or (iii) have been used in a manner contrary to ABB operating and maintenance procedures.

THE ABOVE WARRANTIES AND REMEDIES ARE EXCLUSIVE AND IN LIEU OF ANY AND ALL OTHER REPRESENTATIONS, SPECIFICATIONS, WARRANTIES AND REMEDIES EITHER EXPRESS OR IMPLIED, HEREIN OR ELSEWHERE, OR WHICH MIGHT ARISE UNDER LAW OR EQUITY OR CUSTOM OF TRADE INCLUDING WITHOUT LIMITATION WARRANTIES OF MERCHANTABILITY AND OF FITNESS FOR A SPECIFIED OR INTENDED PURPOSE. THE REMEDY SPECIFIED REPRESENTS THE SOLE LIABILITY OF ABB AND THE SOLE REMEDY OF BUYER WITH RESPECT TO OR ARISING OUT OF THE EQUIPMENT OR SERVICES WHETHER BASED ON CONTRACT, TORT (INCLUDING NEGLIGENCE AND STRICT LIABILITY), OR OTHERWISE.

9 Contacts

Please direct all technical and parts inquiries to:

ABB Flexible Automation.
Welding Systems
695 80 Laxå
Sweden
Ph: +46 (0)584-81666

Please have IRB serial numbers on hand.

10 Glossary

Baseframe	A frame representing the relationship of a robot's base to the world frame.
Displacement	A frame that represents the spatial relationship between a work object and a robtarget. *Displacements* are represented by pose data in RAPID. Search results are typically *displacements*.
Electrode Extension	The location at the tip of the consumable welding wire where metal transfer occurs.
Frame	A coordinate system. Work objects, tool data, displacements, robtargets, etc. are all constructed from *frames*. *Frames* are represented by pose data in RAPID.
Gas Cup	The outer cylindrical portion of a MIG gun that directs the shielding gas over the weld puddle. Often used as a search-sensing surface. Also called a *nozzle*.
Nozzle	See *Gas Cup*.
Pose data	A frame in RAPID. It is represented by Cartesian coordinates and quaternions. (x,y,z,q1,q2,q3,q4)
RAPID	A C-based programming language used for ABB robots.
Robtarget	A frame representing a position in space. Movement instructions like MoveL, MoveJ, ArcL, and Search_1D, all require *robtargets*.
SmarTac	A tactile sensor used to detect inconsistent weld joints and return correctional data to adjust welding position.
Stickout	The distance measure from the contact tip to the weld seam.
TCP	Tool Center Point. The frame representing the spatial relationship between the electrode extension and the wrist of the robot.
Tooldata	A frame representing the TCP in RAPID.
Work object	*wobjdata*. RAPID data type composed of two frames, the "user frame" and "object frame". These two frames together represent the spatial relationship between the World and a program displacement frame.
World	All spatial frame relationships are ultimately related to the *world* frame.

503 497-502
2001-03-06