ROBOTICS

Application manual

SmarTac

Trace back information:
Workspace Main version a564
Checked in 2024-02-07
Skribenta version 5.5.019

Application manual
SmarTac

SmarTac for OmniCore

Document ID: 3HAC087219-001

Revision: A

Table of contents

Overview of this manual

About this manual

This manual explains the basics of when and how to use the option SmarTac:

- Product overview
- Operation overview
- Requirements overview
- Software set-up
- Software reference, RAPID

Usage

This manual can be used either as a reference to find out if an option is the right choice for solving a problem, or as a description of how to use an option. Detailed information regarding syntax for RAPID routines, and similar, is not described here, but can be found in the respective reference manual.

Who should read this manual?

This manual is intended for:

- installation personnel
- robot programmers

Prerequisites

The reader should...

- be familiar with industrial robots and their terminology
- be familiar with the RAPID programming language
- be familiar with system parameters and how to configure them.

Reference documents

References	Document ID
Technical reference manual - RAPID Overview	*3HAC065040-001*
Technical reference manual - RAPID Instructions, Functions and Data types	*3HAC065038-001*
Operating manual - OmniCore	*3HAC065036-001*
Technical reference manual - System parameters	*3HAC065041-001*
Operating manual - RobotStudio	*3HAC032104-001*
Application manual - Product.ProductName	*3HAC084370-001*

Revisions

Revision	Description
A	Released with RobotWare 7.13.

1 Introduction

1.1 Product overview

General

SmarTac is a software package used to find the location of inconsistent weld joints and offset the programmed points in a weld program.

SmarTac uses the torch or welding wire as a tactile sensor and can be used with any modern power source that provide a *Touch Sense* functionality.

Searching with SmarTac

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

General

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 a certain voltage. The voltage level can be different and depend on the used power source brand. When the torch feature contacts 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.

Search instructions

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.

Why use SmarTac?

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

1.3 System requirements

Introduction

This SmarTac version is intended for use in arc welding systems incorporating IRB 1600, 1660, 2600, etc. robots.

- RobotWare: 7.13
- Controller requirements: OmniCore
- Option: 3416-2 Arc Welding Premium

SmarTac package

The SmarTac package includes software that is loaded into all arc welding motion tasks, when the option is installed.

Process configuration parameters are used to connect real I/O signals and to modify the default settings.

2 Installation

2.1 Safety instructions

DANGER

Before doing any work inside the cabinet, disconnect the mains power.

ELECTROSTATIC DISCHARGE (ESD)

The components are sensitive to ESD. Always use ESD protection when handling them. Use the wrist strap located on the controller.

DANGER

All personnel working with the robot system must be very familiar with the safety regulations. Incorrect operation can damage the robot or injure someone.

CAUTION

Danger from unexpected electric shock.

When touch sensing is activated, a voltage of approximately 70 V (voltage depends on the power source brand and can be higher) is applied to the wire electrode/gas nozzle.

If touched, a harmless but perceptible electric shock can be transmitted. An involuntary reaction to this shock can cause injuries.

Do not touch the wire electrode and the torch body (gas nozzle, contact tip, etc.) when touch sensing is active.

2.2 Hardware installation

Components

The SmarTac Add-In does not consist of any additional hardware needed for the OmniCore controller. It utilizes the provided *Touch Sense* functionality of a power source.

Additional hardware can be required depending on the used power source brand to get the torch/gas cup energized, for example, for the Fronius TPS/i power source the Fronius option *Opt/I WF Gas nozzle position 4,100,855,IK* is mandatory to search with the gas cup.

For more information, contact your local welding equipment representative.

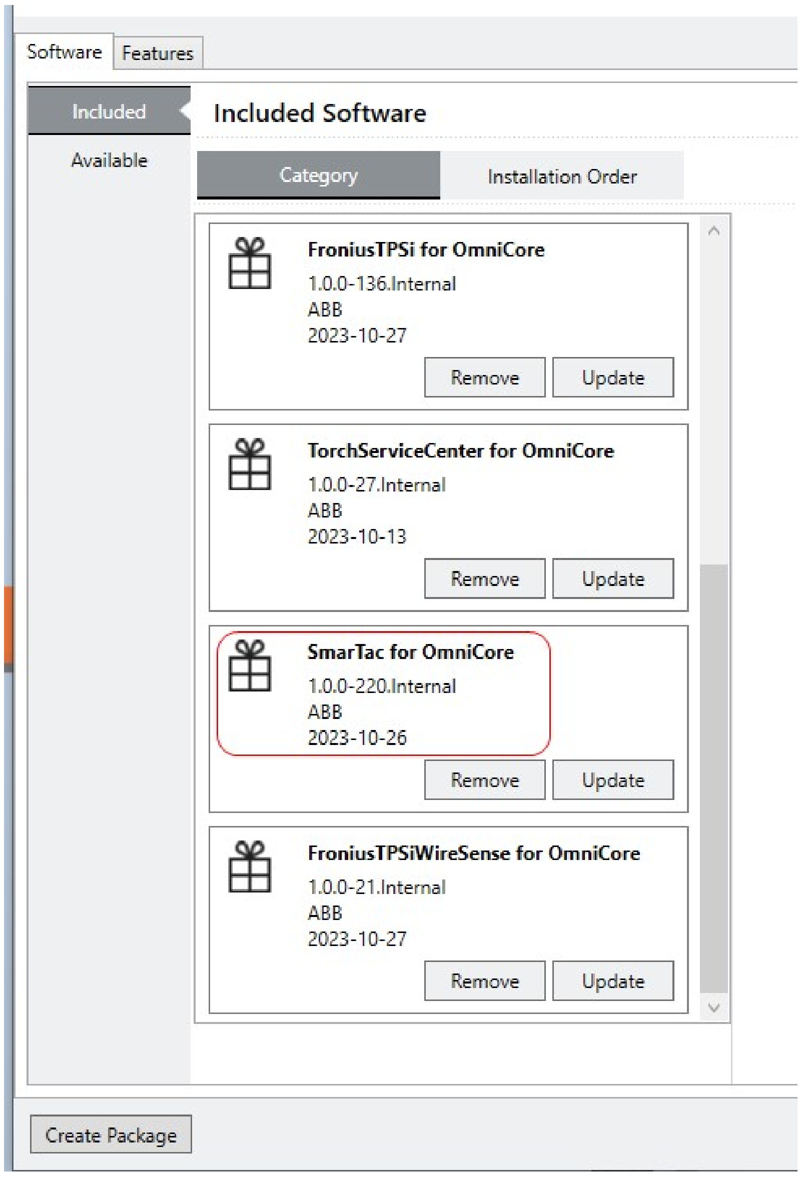
2.3 Software installation

2.3.1 About SmarTac Add-in

Installation

The SmarTac function package is provided as an Add-In and is installed on the robot controller using Robotstudio, the **Modify Installation** function.

Add the option to the system.



FroniusTPSi for OmniCore
1.0.0-136.Internal
ABB
2023-10-27

TorchServiceCenter for OmniCore
1.0.0-27.Internal
ABB
2023-10-13

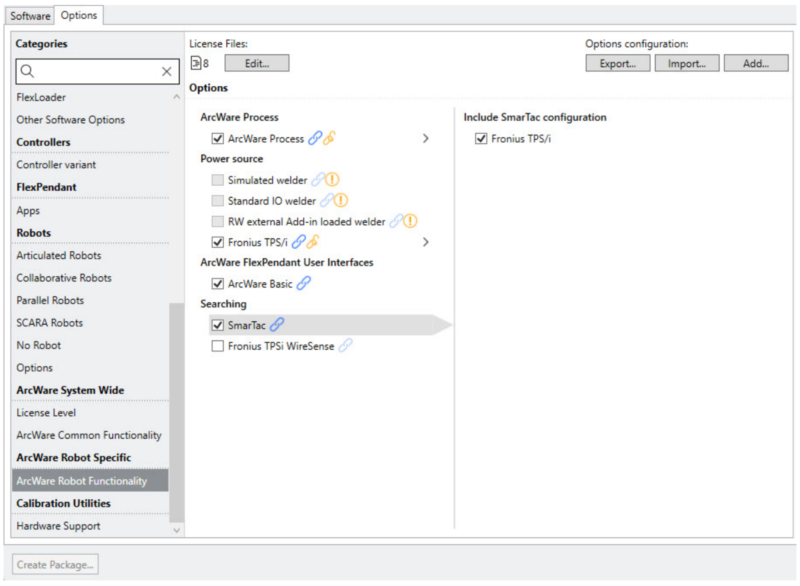
SmarTac for OmniCore
1.0.0-220.Internal
ABB
2023-10-26

FroniusTPSiWireSense for OmniCore
1.0.0-21.Internal
ABB
2023-10-27

xx2300001807

Continues on next page

In the **Feature** section, browse to **ArcWare Robot Functionality - Searching** and select **SmarTac** (if not already selected). Pre-defined configuration settings might be available, which automatically will install the power source-related configuration.



Options

ArcWare Process
- [✓] ArcWare Process

Power source
- [ ] Simulated welder
- [ ] Standard IO welder
- [ ] RW external Add-in loaded welder
- [✓] Fronius TPS/i

ArcWare FlexPendant User Interfaces
- [✓] ArcWare Basic

Searching
- [✓] SmarTac
- [ ] Fronius TPSi WireSense

Include SmarTac configuration
- [✓] Fronius TPS/i

xx2300001808

2.3.2 System parameters

SmarTac Settings

Parameter	Value	Description
Name	T_ROB1 - T_ROB3	Parameter used for the selected *Task*. Type: `string`
Uses Signals	Default value: smtsig1	Reference to the parameter *SmarTac Standard Signals*. Default value will be automatically updated if a supported power source is selected with **Installation Editor**. Type: `string`
Uses Speeds	Default value: smtspeedstd	Reference to to the parameter *SmarTac Speeds*. Type: `string`
Use EIO Interface	Default value: FALSE	Parameter to enable the EIO interface. Status and error codes will be sent via a configurable interface. Type: bool
Disable Errorhandler	Default value: FALSE	Parameter to disable SmarTac internal error handler. All errors must be handled on user level.

SmarTac Speeds

Parameter	Value	Description
Name	Default value: Smtspeedstd	Parameter defines the default search speed. Type: string
Main Search speed	Default value: 20 mm/s Min value: 1 mm/s Max value: 80 mm/s	Default search speed for the instructions `Search_1D` and `Search_Part`. Type: num
Grove Search Speed	Default value: 15 mm/s	Default search speed for the instruction `Search_Groove`. Type: num

SmarTac Standard Signals

Parameter	Signal type	Description
Name	Default value: smtsig1	Name of the parameter instance. Type: string
Detection input	Digital Input	Input to indicate that the part feature is found. Wire/Gas cup had contact with the part. This input will stop the search movement.
Wire Select Output	Digital Output	Output to selected between searching with the wire or the gas cup.
Sensor On Output	Digital Output	Output to activate the touch sense functionality for the power source. This wire/gas cup will be energized with voltage.

Continues on next page

SmarTac Errorhandler IO

Parameter	Signal type	Description
Name		Name of the parameter in-stance.
Active Dialog Type	Group Output	Indicate the active dialog type number.
Dialog active	Digital Output	Indicates an active dialog on the FlexPendant.
Dialog Acknowledge	Digital Input	Response from remote device.
Response	Group Input	Response from remote device.
Error Type	Group Output	Type of error.
Error Number	Group Output	Not yet used.
SmarTac Error	Digital Output	Error status indication.

For more information on how to use the SmarTac Errorhandler IO, see *Search Error Recovery I/O interface on page 19*.

2.4 Search Error Recovery I/O interface

General information

SmarTac provides an I/O based interface to communicate with an external device, mainly a PLC, to indicate an active user dialog on the FlexPendant that needs attention, and to remotely control that user dialog. The I/O interface supports all search instructions provided by the SmarTac software package.

The Search Error Recovery I/O interface behaves like the Weld Error Recovery in ArcWare, as it follows the same concept. If the Weld Error Recovery in ArcWare is configured the same way, signals can be configured to remotely control the SmarTac user interface.

Note

The bit mapping (length) for the group outputs/inputs might be changed if the Weld Error Recovery I/O interface in ArcWare is used in combination with the Search Error Recovery I/O interface in SmarTac.

Additional information for the Weld Error Recovery I/O interface can be found in *Application manual - Product.ProductName*.

The internal error handling in SmarTac can be switched off if wanted. All SmarTac related errors must be handled on user level by adding an error handler.

Note

If the internal error handler is switched off, the Search Error Recovery I/O interface cannot be used.

Usage

The Search Error Recovery dialogs presented on the FlexPendant may be acknowledged from a remote source through an optional I/O interface. This is necessary if a PLC or other remote computer is used for the primary operator interface while running production.

Architecture

All I/O signals used with the Search Error Recovery I/O interface must be configured.

In a MultiMove system, each welding robot will have its own Search Error Recovery I/O interface with separate I/O signals. The end user can specify his own signal names for each welding robot in the system parameters (topic Process). To simplify this document, the signal names will here be described as `signalname_x`. For example: `diSMT_Ack_X`, where x specifies the welding robot number. The I/O interface will be activated if all signals for each welding robot are defined in the system, otherwise, the I/O interface will be disabled.

Continues on next page

The Search Error Recovery I/O interface signal definition (X represents robot number 1-4)

Signal common name	Signal definition name	Description
Application Error	`doSMT_Error_X`	Indicate a general SmarTac error. This output can be used if the internal error handler is switched off. It will work independent from the I/O interface. Type: Digital Output
Prompt Acknowledge	`diSMT_Ack_X`	Allows the remote device to acknowledge a Weld Error Recovery prompt. Type: Digital Input
Dialog Active	`doSMT_Dialog_X`	Indicates to a remote device that a dialog is active and is awaiting a response. Type: Digital Output
Active Dialog Type	`goSMT_Dialog_X`	Indicates to the remote device that the Dialog Type prompt is active (Type 9 to Type 12). **Note** Type 1 to 8 are reserved/used for the Weld Error Recovery I/O interface in RobotWare Arc. Type: Group Output
Response	`giMT_Response_X`	Allows the remote device to communicate a response. The context of the response is dictated by the active dialog type: **Active Dialog 9** 1 Retry 2 Return 3 Abort (Raise) **Active Dialog 10** 1 Return 2 Abort (Raise) **Active Dialog 11** 1 Retry 2 Detect 3 Reject 4 Abort (Raise) **Active Dialog 12** 1 OK 2 Abort (Raise) Type: Group Input

Continues on next page

Signal common name	Signal definition name	Description
Error Type	`goSMT_ErrType_X`	Indicates the SmarTac error type to the remote device. Valid output data range: 30-36 0 = No active error type (30) `SMT_ACT_ERR` (31) `SMT_SENON_ERR` (32) `SMT_NOGROOVE` (33) `SMT_GROOVESEARCH` (34) `SMT_LIMIT` (35) `SMT_CONFLICT_ERR` (36) `ERR_WHLSEARCH` Type: Group Output 6 bit
Error Number	`goSMT_ErrNum_X`	Not yet used since SmarTac does not create specific error codes. The only thing that is logged is a SmarTac search override warning. Type: Group Output

Sequence

The I/O sequence is as follows:

1. A search error occurs triggering a Search Error Recovery prompt to be displayed. The Search Error Recovery will set `doSMT_Dialog_X` high to indicate an active prompt. The Search Error Recovery will also set `goSMT_Dialog_X` to indicate the type of prompt. If the prompt is an error type, an error type will be supplied on group outputs `goSMT_ErrType_X`.
2. The remote device interprets the information. If the dialog prompt type requires a numeric response, the remote device supplies the value on `giSMT_Response_X`.
3. The remote device acknowledges the prompt by pulsing the `diSMT_Ack_X` signal. The Search Error Recovery responds by closing the prompt on the FlexPendant.

The Weld Error Recovery I/O interface will be inoperable until the `diSMT_Ack_X` signal is reset.

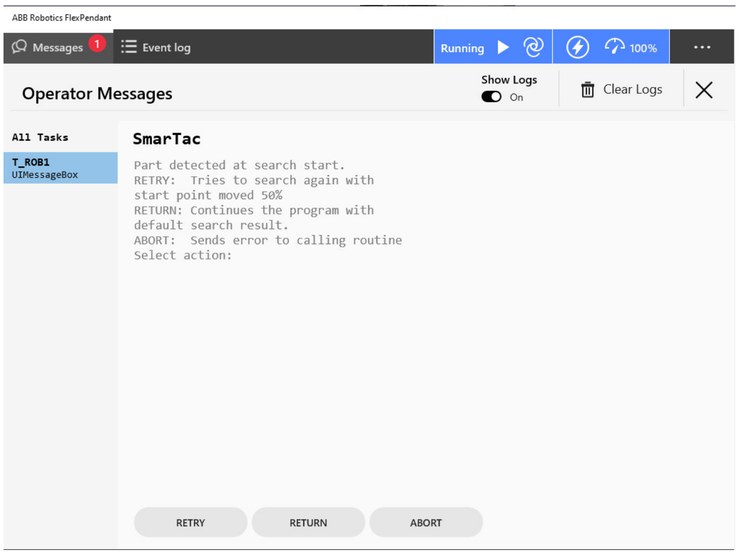
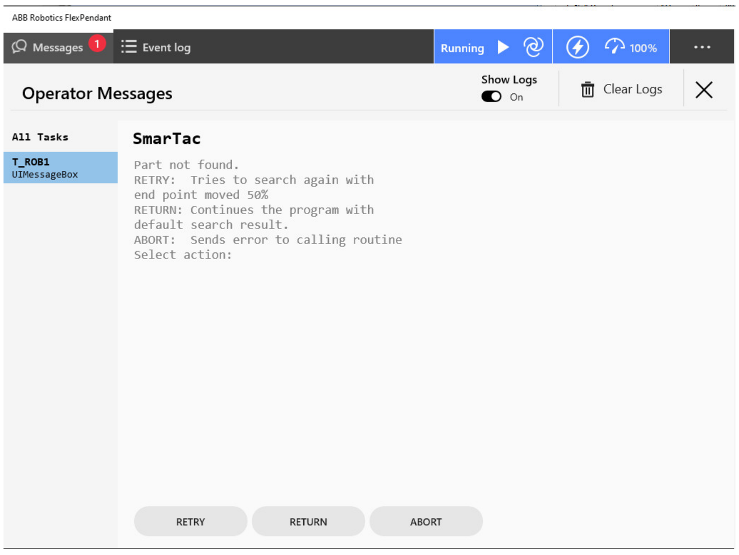
A warning will be written in the event log if `diSMT_Ack_X` was active before the user dialog was active. In such a case, the group outputs `goSMT_ErrType_X` and `goSMT_Dialog_X` remain 0. The output `doSMT_Dialog_X` is still set to indicate a necessary user action on the FlexPendant.

Active dialog types

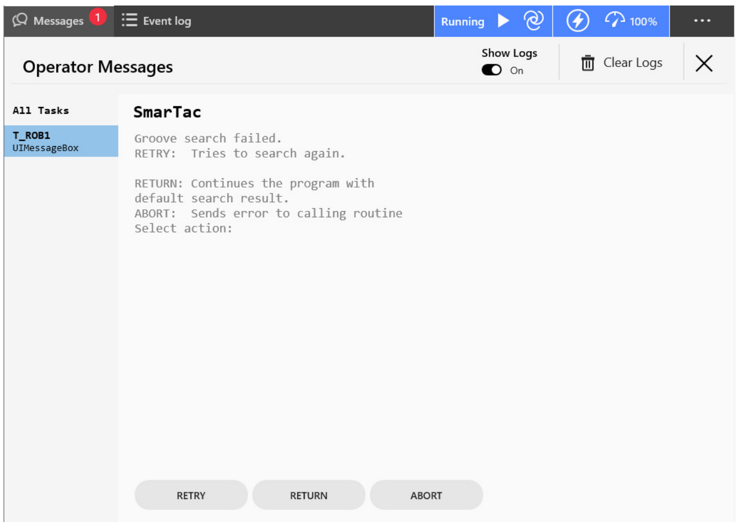
There are four possible dialog prompts from the Search Error Recovery. When one of the four prompts are active, the digital output `doSMT_Dialog_X` will be high. The prompts require a numeric response from `giSMT_Response_X` followed by an acknowledgment from `diSMT_Ack_X`.

Continues on next page

Dialog type 9

Error	User dialog
Part detected at search start	 ABB Robotics FlexPendant Messages 1 Event log Running 100% Operator Messages Show Logs On Clear Logs All Tasks T_ROB1 UIMessageBox RETRY RETURN ABORT SmarTac Part detected at search start. RETRY: Tries to search again with start point moved 50% RETURN: Continues the program with default search result. ABORT: Sends error to calling routine Select action: xx2300001810
Part not found	 ABB Robotics FlexPendant Messages 1 Event log Running 100% Operator Messages Show Logs On Clear Logs All Tasks T_ROB1 UIMessageBox RETRY RETURN ABORT SmarTac Part not found. RETRY: Tries to search again with end point moved 50% RETURN: Continues the program with default search result. ABORT: Sends error to calling routine Select action: xx2300001811

Continues on next page

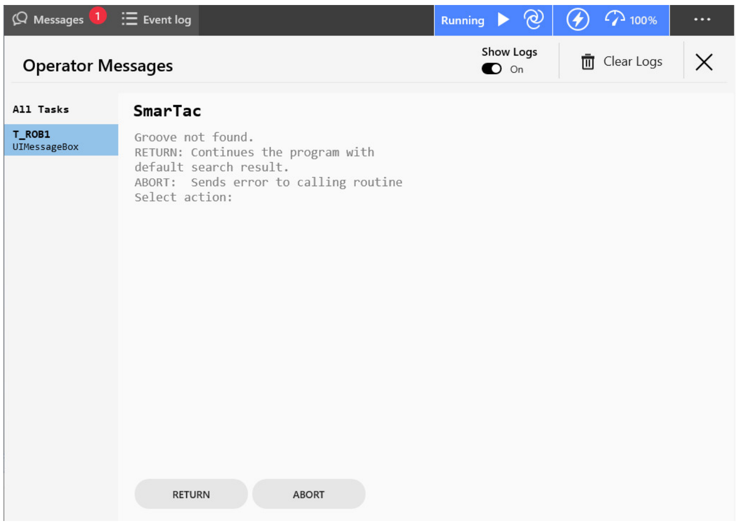
Error	User dialog
Groove search failed	 SmarTac Groove search failed. RETRY: Tries to search again. RETURN: Continues the program with default search result. ABORT: Sends error to calling routine Select action: Messages Event log Running 100% Operator Messages Show Logs On Clear Logs All Tasks T_ROB1 UIMessageBox RETRY RETURN ABORT xx2300001809

When one of the dialogs above is active, the signal `doSMT_Dialog_X` will be high and `goSMT_Dialog_X` will be set to 9. The remote device may respond to the dialog by setting `giSMT_Response_X` to a value from the list below, followed by pulsing `diSMT_Ack_X`.

Response value	Description
(1)	Retry
(2)	Return
(3)	Abort (Raise)

Dialog type 10

The dialog prompt type 10 is only used within the RAPID instruction `Search_Groove`.

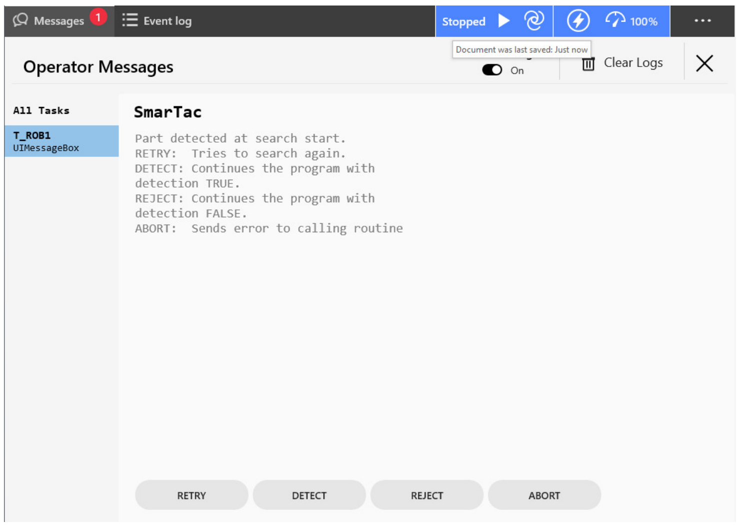
Error	User dialog
Groove not found	 SmarTac Groove not found. RETURN: Continues the program with default search result. ABORT: Sends error to calling routine Select action: Messages Event log Running 100% Operator Messages Show Logs On Clear Logs All Tasks T_ROB1 UIMessageBox RETURN ABORT xx2300001812

Continues on next page

Response value	Description
(1)	Return
(2)	Abort (Raise)

Dialog type 11

The dialog prompt type 11 is only used within the RAPID instruction `Search_Part`.

Error	User dialog
Part detected at search start	 SmarTac Part detected at search start. RETRY: Tries to search again. DETECT: Continues the program with detection TRUE. REJECT: Continues the program with detection FALSE. ABORT: Sends error to calling routine xx2300001813

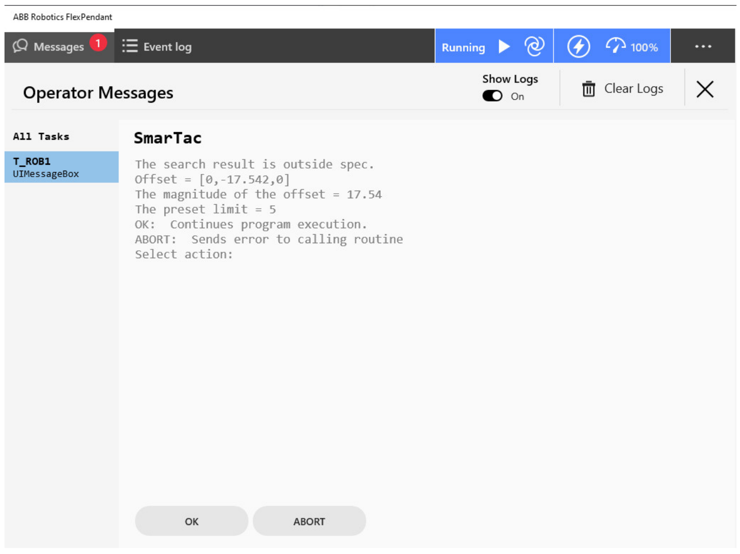
When one of the dialogs above is active, the signal `doSMT_Dialog_X` will be high and `goSMT_Dialog_X` will be set to 11. The remote device may respond to the dialog by setting `giSMT_Response_X` to a value from the list below, followed by pulsing `diSMT_Ack_X`.

Response value	Description
(1)	Retry
(2)	Detect
(3)	Reject
(4)	Abort (Raise)

Continues on next page

Dialog type 12

The dialog prompt type 12 is only used within the RAPID instruction `Search_1D`.

Error	User dialog
Limit error This error only occurs when the optional argument `[\Limit]` is used with the `Search_1D` instruction.	 ABB Robotics FlexPendant Messages 1 Event log Running 100% Operator Messages Show Logs On Clear Logs All Tasks T_ROB1 UIMessageBox OK ABORT SmarTac The search result is outside spec. Offset = [0,-17.542,0] The magnitude of the offset = 17.54 The preset limit = 5 OK: Continues program execution. ABORT: Sends error to calling routine Select action: xx2300001814

When the dialog above is active, the signal `doSMT_Dialog_X` will be high and `goSMT_Dialog_X` will be set to 12. The remote device may respond to the dialog by setting `giSMT_Response_X` to a value from the list below, followed by pulsing `diSMT_Ack_X`.

Response value	Description
(1)	OK
(2)	Abort (Raise)

Error type

The error type will be sent on `goSMT_ErrType_X`. The following table displays a list of possible error types from SmarTac:

ERRNO	Description	Error type
SMT_ACT_ERR	SmarTac activation error	30
SMT_SENON_ERR	Part detected prior search	31
SMT_NOGROOVE	Groove not found	32
SMT_GROOVESEARCH	Groove search error	33
SMT_LIMIT	Limit Error (`Search_1D`)	34
SMT_CONFLICT_ERR	Conflict error	35
ERR_WHLSEARCH	Error during search	36

Error number

The group output `goSMT_ErrNum_X` is not yet used and is reserved for future usage. In the current version, SmarTac does not provide any error codes related to search errors. The output will bet set to 0.

2.5 Configure the Search Error Recovery I/O interface

Description

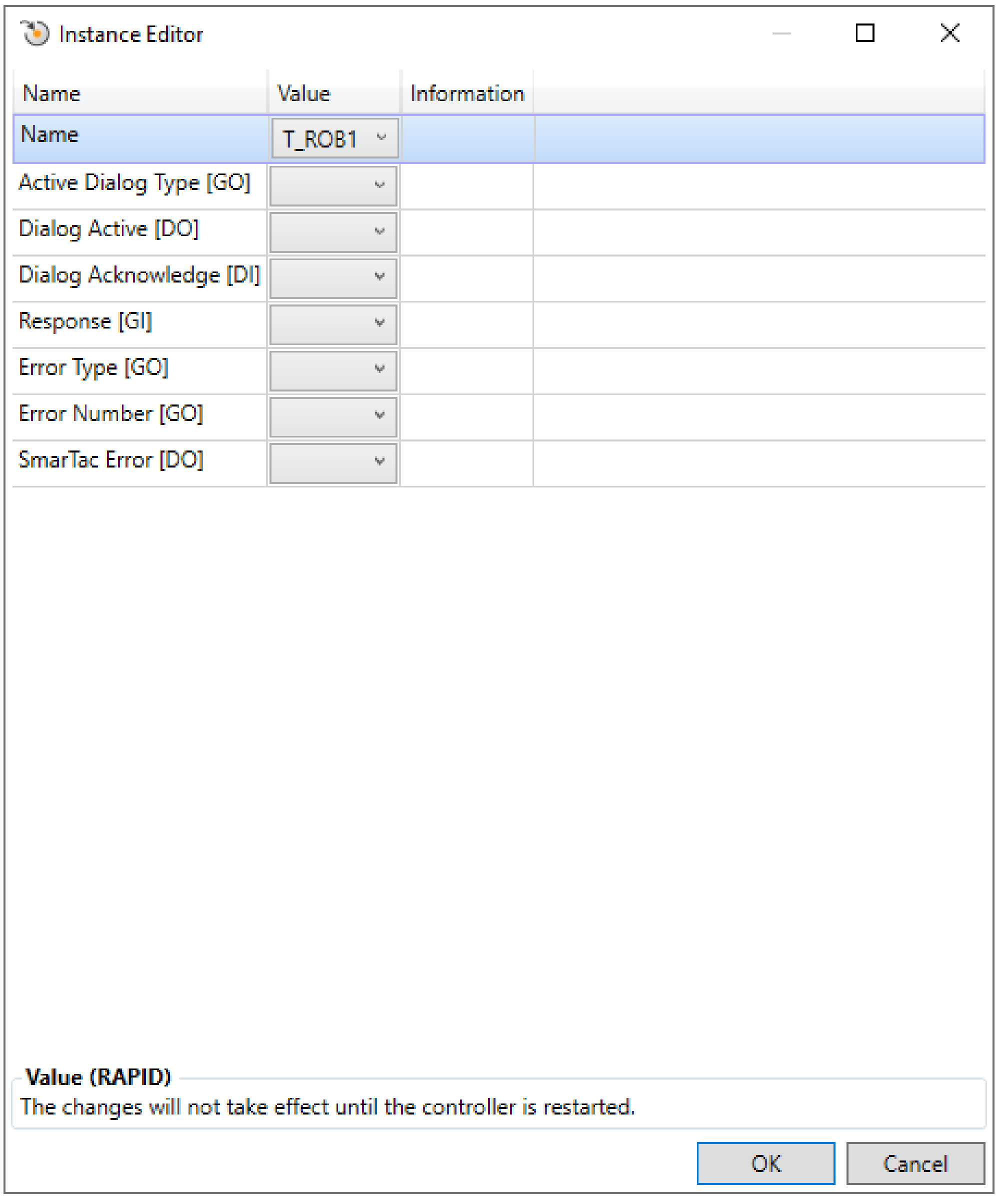
The Search Error Handler I/O configures the Search Error Recovery I/O part of the Search Error Recovery feature in SmarTac.

The configuration parameters can be found in the **Configuration Editor**, topic **Process**, type `Arc Error Handler I/O`, in RobotStudio.

In order to use the Search Error Handler I/O interface, the parameter **Use EIO Interface** must be defined as active in the SmarTac settings.

Examples

The default configuration has the following definition and can be found in the process configuration database **PROC/SmarTac Errorhandler IO**.



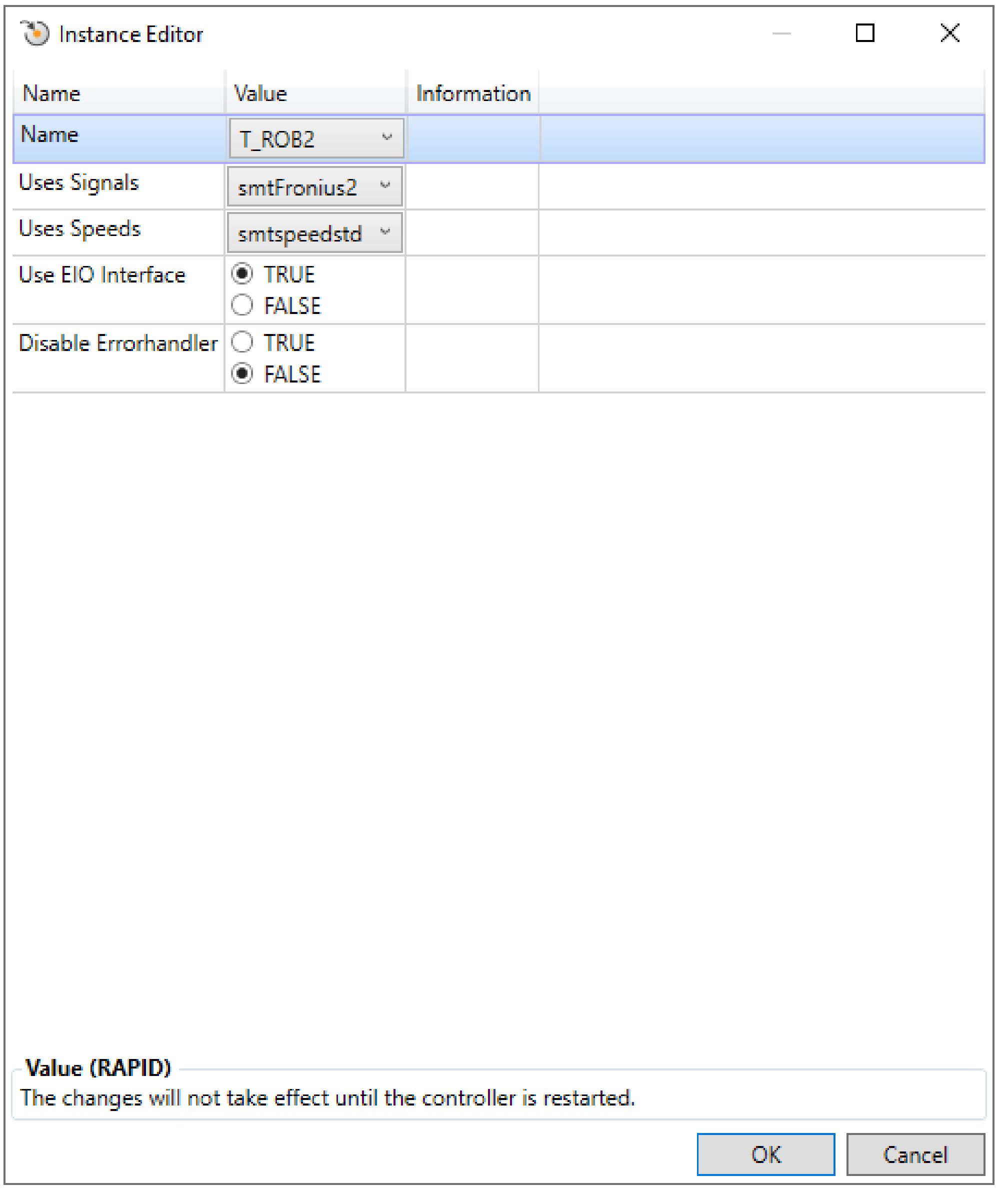
Name	Value	Information
Name	T_ROB1	
Active Dialog Type [GO]		
Dialog Active [DO]		
Dialog Acknowledge [DI]		
Response [GI]		
Error Type [GO]		
Error Number [GO]		
SmarTac Error [DO]		

Value (RAPID)
The changes will not take effect until the controller is restarted.

xx2000002104

Continues on next page

To activate the Search Error Handler I/O interface, set field **Use EIO Interface** to TRUE. The parameter can be found in the process configuration database **PROC/SmarTac Settings.**



Name	Value	Information
Name	T_ROB2	
Uses Signals	smtFronius2	
Uses Speeds	smtspeedstd	
Use EIO Interface	[✓ TRUE] FALSE	
Disable Errorhandler	TRUE [✓ FALSE]	

Value (RAPID)
The changes will not take effect until the controller is restarted.

xx2000002105

Parameters

Parameter	Description	Data Type
Name	The name of the instance SMT_ERR_HNDL_IO. Must be T_ROB1-T_ROB4.	type: StringNormal
goWER_Dialog	The signal name for Active Dialog Type.	go
doWER_Dialog	The signal name for Dialog Active.	do
diWER_Ack	The signal name for Prompt Acknowledge.	di
giWER_Response	The signal name for Response.	gi

Continues on next page

Parameter	Description	Data Type
goWER_ErrType	The signal name for Error Type.	go
goWER_ErrNum	The signal name for Error Number.	go
doSMT_Error	The signal name for general Error.	do

2.6 User defined error handling

Description

The internal error handler in SmarTac can be switched off to handle all possible errors on user level. The error handler can be switched off in the process configuration database.

The parameter **Disable Errorhandler** can be found in the process configuration database **PROC/SmarTac Settings**.

The Search Error Handler I/O interface will remain inactive as long as the internal error handler of SmarTac is disabled even if the Search Error Handler I/O interface is configured.

The output `doSMT_Error_X` is set if a search error or activation error occur. The signal is set to 0 with the next execution of a SmarTac search instruction.

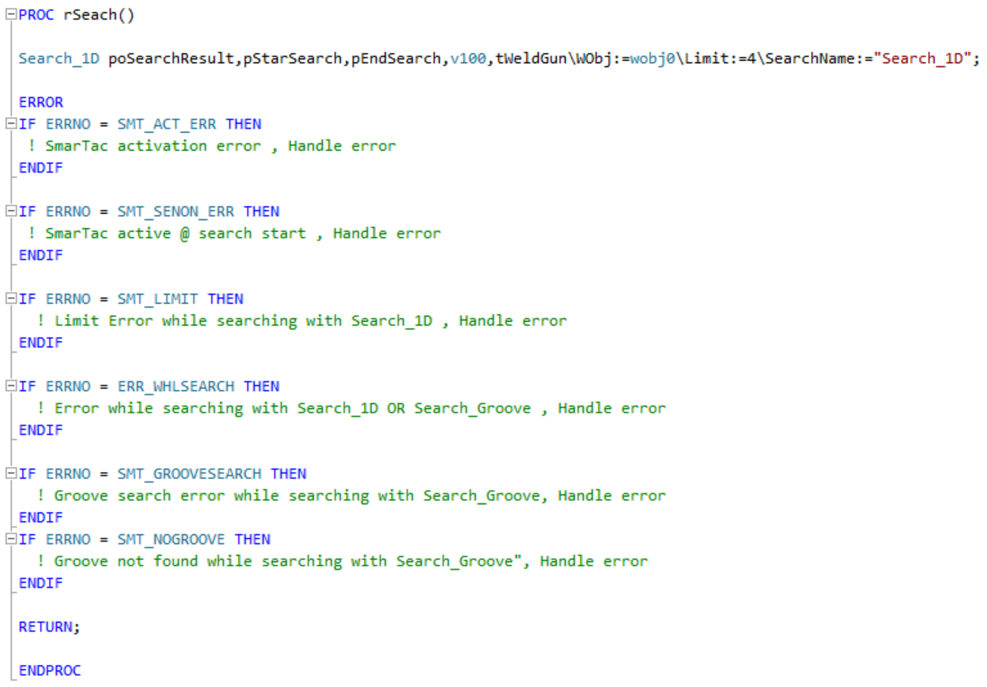
Error handling

The following errors can be handled by the internal error handler:

ERRNO	Description
SMT_ACT_ERR	SmarTac activation error
SMT_SENON_ERR	Part detected prior search
SMT_NOGROOVE	Groove not found
SMT_GROOVESEARCH	Groove search error
SMT_LIMIT	Limit Error (Seach_1D)
SMT_CONFLICT_ERR	Conflict error
ERR_WHLSEARCH	Error during search

Continues on next page

Example



```
PROC rSeach()

Search_1D poSearchResult,pStarSearch,pEndSearch,v100,tWeldGun\WObj:=wobj0\Limit:=4\SearchName:="Search_1D";

ERROR
IF ERRNO = SMT_ACT_ERR THEN
 ! SmarTac activation error , Handle error
ENDIF

IF ERRNO = SMT_SENON_ERR THEN
 ! SmarTac active @ search start , Handle error
ENDIF

IF ERRNO = SMT_LIMIT THEN
  ! Limit Error while searching with Search_1D , Handle error
ENDIF

IF ERRNO = ERR_WHLSEARCH THEN
  ! Error while searching with Search_1D OR Search_Groove , Handle error
ENDIF

IF ERRNO = SMT_GROOVESEARCH THEN
  ! Groove search error while searching with Search_Groove, Handle error
ENDIF
IF ERRNO = SMT_NOGROOVE THEN
  ! Groove not found while searching with Search_Groove", Handle error
ENDIF

RETURN;

ENDPROC
```

xx2000002118

3 Application guide

3.1 Searching conditions

Introduction

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.

3.2 Programming limitations

Searching with welding wire

In systems where searching with the welding wire is needed, a wire trimmer is necessary to ensure a known wire stick-out. A wire trimmer is a hardware device that requires extra I/O. This option may be purchased though ABB.

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 User's guide

4.1 Introduction

Note

All exercises assume that the SmarTac software and hardware are installed and working properly.

Questions

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 do not, 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 the part fit-up and/or fixture is poor, you will 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. See *Operating manual - OmniCore* and *Operating manual - RobotStudio*.
3. What are the RAPID instructions and how are they used in my weld routines?

   In this guide we will look at several search techniques, with detailed examples. The SmarTac instructions and functions are described in *RAPID reference on page 79*.

4.2 Exercise 1: program displacement

About the exercise

This exercise demonstrates how a program displacement works.

Note

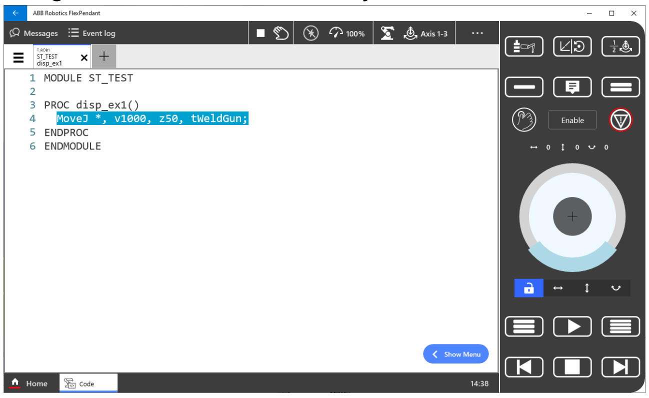
The exercises later in this guide will not be as detailed as this one. Please take the time to understand this exercise before attempting others.

Instruction

	Action
1	Create a new program module. Name it `ST_TEST`.
2	Create a new routine in that module and name it `disp_ex1`.

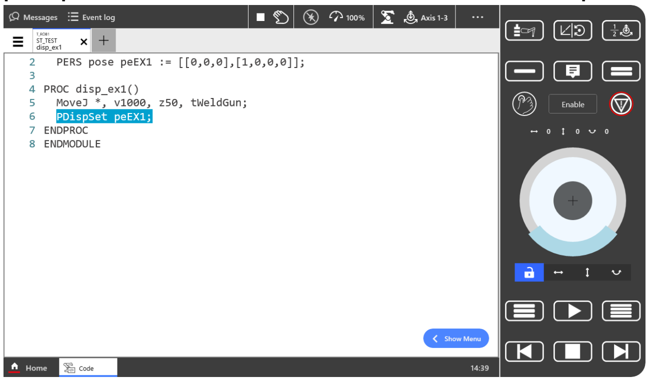
3	If not already done, define the tool using the five point method or BullsEye. Name the tool `tWeldGun`.
	To make programming easier you can add in these instructions into one of your *Most Common* pick lists:
	`PDispAdd`
	`PDispOff`
	`PDispSet`
	`Search_1D`
	`Search_Groove`
	`Search_Part`

4	Tape a piece of paper to a table, or similar surface, within the robot's reach. On the paper draw a rectangle.
5	View the modules, select the new module `ST_TEST`, and select the new routine, `disp_ex1`.

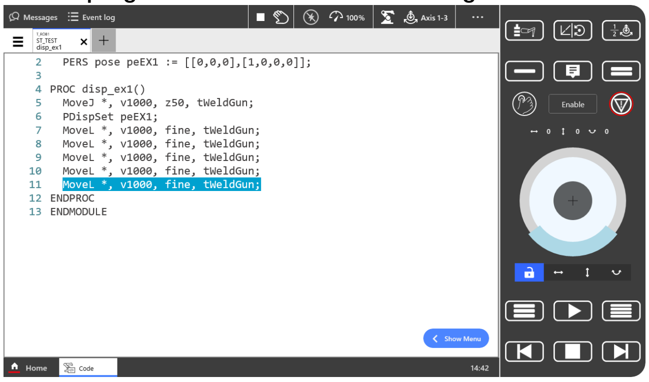
6	Jog the robot so that the torch is pointing at the rectangle on the paper. The tip of the torch should be a few inches above the rectangle. Create a `MoveJ` at this point using `tWeldGun` and no work object selected.
	
	 ABB Robotics FlexPendant
	Messages
	Event log
	100%
	Axis 1-3
	T_ROB1
	ST_TEST
	disp_ex1
	Enable
	Show Menu
	Home
	Code
	14:38
	`MODULE ST_TEST`
	
	`PROC disp_ex1()`
	`  MoveJ *, v1000, z50, tWeldGun;`
	`ENDPROC`
	`ENDMODULE`
	xx2300000290

Continues on next page

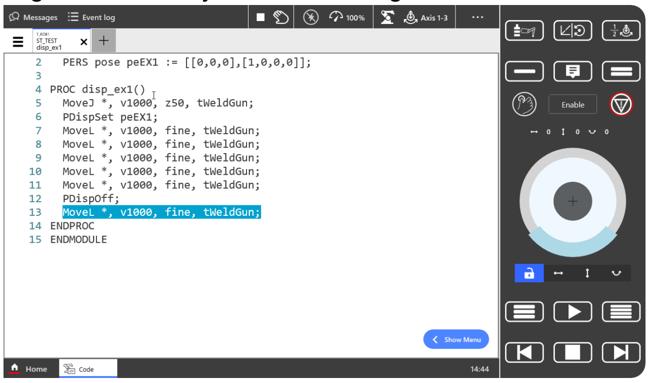
	Action

7	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`.
	 Messages
	Event log
	100%
	Axis 1-3
	T_ROB1 ST_TEST disp_ex1
	Enable
	Show Menu
	Home
	Code
	14:39
	`  PERS pose peEX1 := [[0,0,0],[1,0,0,0]];`
	` `
	`PROC disp_ex1()`
	`  MoveJ *, v1000, z50, tWeldGun;`
	`  PDispSet peEX1;`
	`ENDPROC`
	`ENDMODULE`
	xx2300000291

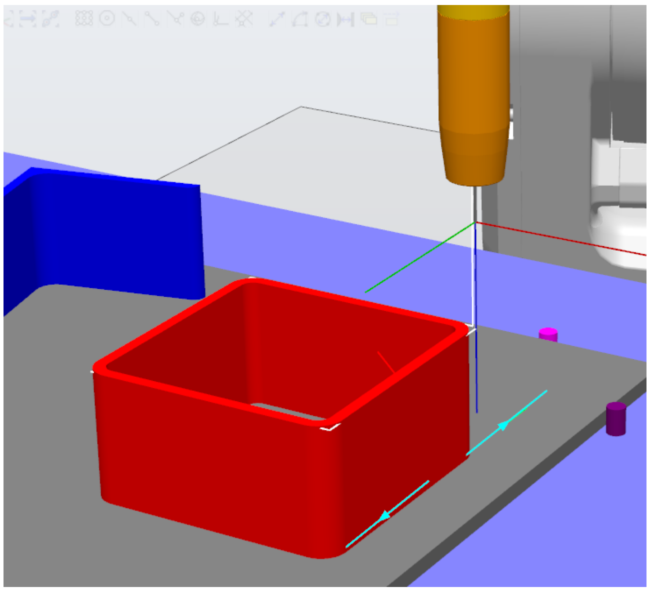
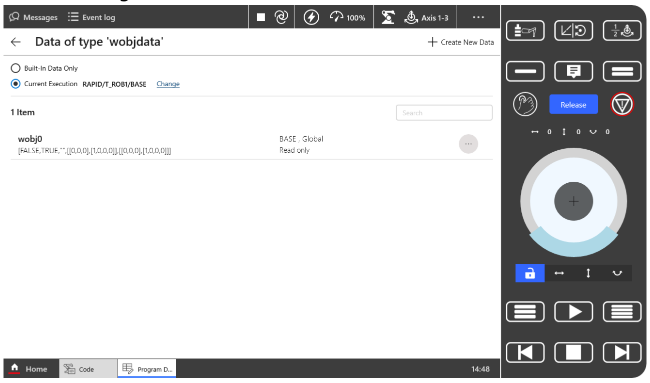
8	Jog the robot down to the rectangle so that the tip is just above one of the rectangle corners. Create a `MoveL` at this position using `tWeldGun` and no work object selected.

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.
	 Messages
	Event log
	100%
	Axis 1-3
	T_ROB1 ST_TEST disp_ex1
	Enable
	Show Menu
	Home
	Code
	14:42
	`  PERS pose peEX1 := [[0,0,0],[1,0,0,0]];`
	` `
	`PROC disp_ex1()`
	`  MoveJ *, v1000, z50, tWeldGun;`
	`  PDispSet peEX1;`
	`  MoveL *, v1000, fine, tWeldGun;`
	`  MoveL *, v1000, fine, tWeldGun;`
	`  MoveL *, v1000, fine, tWeldGun;`
	`  MoveL *, v1000, fine, tWeldGun;`
	`  MoveL *, v1000, fine, tWeldGun;`
	`ENDPROC`
	`ENDMODULE`
	xx2300000292

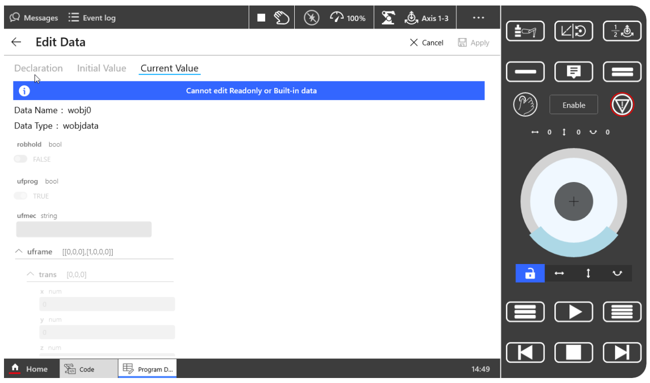
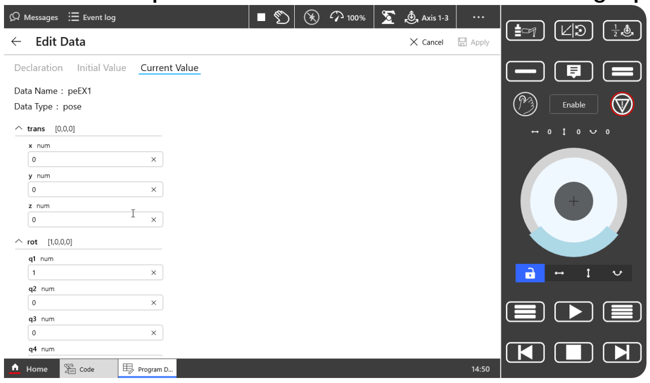
10	Insert the instruction `PDispOff`.

11	Jog the robot away from the rectangle a few inches and insert a final `MoveL`.
	 Messages
	Event log
	100%
	Axis 1-3
	T_ROB1 ST_TEST disp_ex1
	Enable
	Show Menu
	Home
	Code
	14:44
	`  PERS pose peEX1 := [[0,0,0],[1,0,0,0]];`
	` `
	`PROC disp_ex1()`
	`  MoveJ *, v1000, z50, tWeldGun;`
	`  PDispSet peEX1;`
	`  MoveL *, v1000, fine, tWeldGun;`
	`  MoveL *, v1000, fine, tWeldGun;`
	`  MoveL *, v1000, fine, tWeldGun;`
	`  MoveL *, v1000, fine, tWeldGun;`
	`  MoveL *, v1000, fine, tWeldGun;`
	`  PDispOff;`
	`  MoveL *, v1000, fine, tWeldGun;`
	`ENDPROC`
	`ENDMODULE`
	xx2300000293

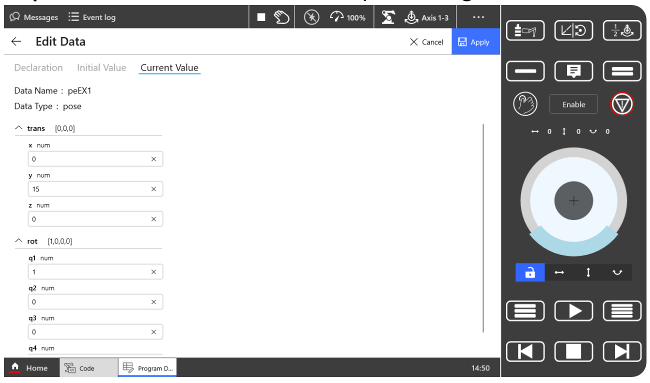
Continues on next page

	Action
12	Execute this routine from the beginning in manual mode to be sure the program works correctly. This routine executes a simple movement path. Each of the robtargets in the `MoveL` instructions is related to the world frame. 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.  xx2300000294
13	Open the Program Data window and take a look at the values in `wobj0`. Then tap to select `wobjdata`.  Messages Event log 100% Axis 1-3 Release Home Code Program D... 14:48 Data of type 'wobjdata' Create New Data Built-In Data Only Current Execution RAPID/T_ROB1/BASE Change 1 Item Search wobj0 [FALSE,TRUE,"",[[0,0,0],[1,0,0,0]],[[0,0,0],[1,0,0,0]]] BASE , Global Read only xx2300000295

Continues on next page

	Action
14	Tap and hold `wobj0`. The following will be visible: Cursor down and look at the data that is present. The only other frame that can change the robtarget positions in our rectangle routine, is the displacement frame. **Note** 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 is why the work object has no affect on our program. It is the same as the World frame.  Messages Event log 100% Axis 1-3 Edit Data Cancel Apply Declaration Initial Value Current Value Enable Home Code Program D... 14:49 Cannot edit Readonly or Built-in data Data Name : wobj0 Data Type : wobjdata robhold bool FALSE ufprog bool TRUE ufmec string uframe [[0,0,0],[1,0,0,0]] trans [0,0,0] x num 0 y num 0 z num xx2300000296
15	Tap **Cancel** to close the work object window. Then select **View Data Types**.
16	Select pose data types.
17	View the data in `peEX1` by selecting it and then select **Edit Value**. A screen will appear showing the values of this data instance. X, Y, and Z will all be zero. This displacement frame did not alter the rectangle program at all.  Messages Event log 100% Axis 1-3 Edit Data Cancel Apply Declaration Initial Value Current Value Enable Home Code Program D... 14:50 Data Name : peEX1 Data Type : pose trans [0,0,0] x num 0 y num 0 z num 0 rot [1,0,0,0] q1 num 1 q2 num 0 q3 num 0 q4 num xx2300000297
18	Move the cursor to the Y value and change the number to 15.

Continues on next page

	Action
19	Tap OK and run the routine, `disp_ex1`, again.  Messages Event log 100% Axis 1-3 Edit Data Cancel Apply Declaration Initial Value Current Value Enable Home Code Program D... 14:50 Data Name : peEX1 Data Type : pose trans [0,0,0] x num 0 y num 15 z num 0 rot [1,0,0,0] q1 num 1 q2 num 0 q3 num 0 q4 num xx2300000298

What happens?

The movements are shifted 15 mm in the positive Y direction. That is 15 mm 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 15 mm 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 modifications are shown in *Exercise 5: using SmarTac with work object manipulation on page 57*).

Try making other changes in X, Y, and Z of `peEX1`. Remember, positive Z will move the rectangle up. Do not use a negative Z, as this will crash the tool.

CAUTION

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.

Advanced

Look at the robtarget data using the same technique for looking at pose data. Note that the robtarget data does not change when the rectangle is moved using the displacement frame.

4.3 Exercise 2: using SmarTac to modify a displacement frame

4.3.1 Introduction

About one-dimensional search

SmarTac programming tools provide a simple way to search a part feature and apply the search results to a program displacement frame. As seen in *Exercise 1: program displacement on page 34*, using program displacements is an easy way of shifting programmed robtargets. The most basic search is a one-dimensional search. A one-dimensional search finds an offset in one direction.

What is Search_1D?

`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 instruction `Search_1D` is described in *Search_1D - One-dimensional search on page 79*.

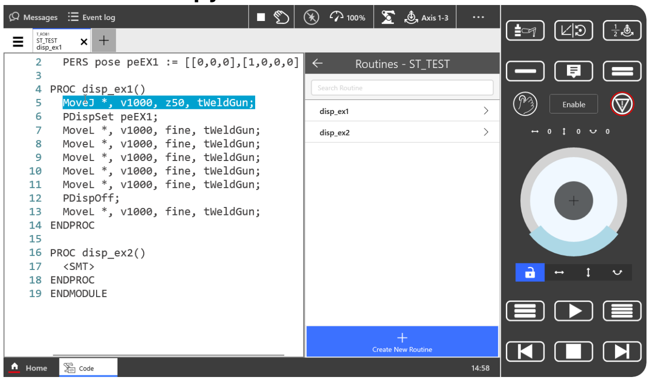
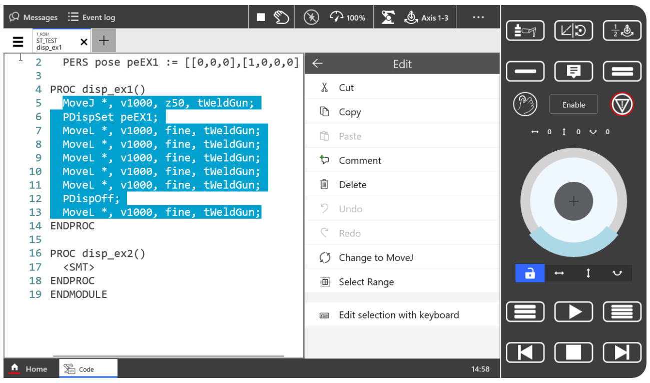
4.3.2 Exercise 2: one-dimensional search

About the exercise

For this exercise the movement routine used in Exercise 1 is used, see *Exercise 1: program displacement on page 34*.

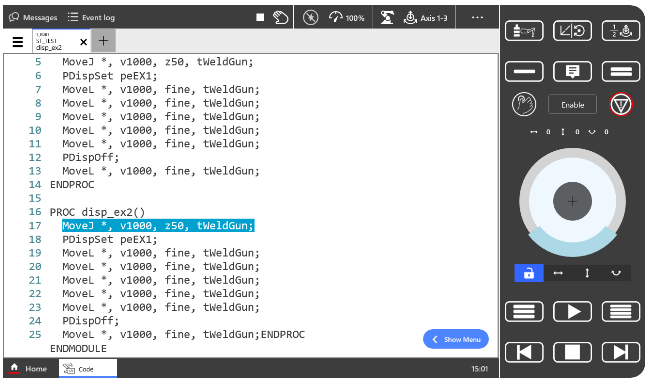
Instruction

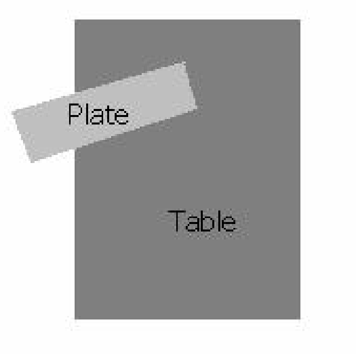
	Action
1	View the routines in the module `ST_TEST`.

2	Select `disp_ex1` and open the **File** menu.
	Select **Create Copy**.
	
	 Messages
	Event log
	100%
	Axis 1-3
	T_ROB1
	ST_TEST
	disp_ex1
	Routines - ST_TEST
	Search Routine
	disp_ex1
	disp_ex2
	Create New Routine
	Home
	Code
	14:58
	Enable
	`  PERS pose peEX1 := [[0,0,0],[1,0,0,0]`
	` `
	`PROC disp_ex1()`
	`  MoveJ *, v1000, z50, tWeldGun;`
	`  PDispSet peEX1;`
	`  MoveL *, v1000, fine, tWeldGun;`
	`  MoveL *, v1000, fine, tWeldGun;`
	`  MoveL *, v1000, fine, tWeldGun;`
	`  MoveL *, v1000, fine, tWeldGun;`
	`  MoveL *, v1000, fine, tWeldGun;`
	`  PDispOff;`
	`  MoveL *, v1000, fine, tWeldGun;`
	`ENDPROC`
	` `
	`PROC disp_ex2()`
	`  <SMT>`
	`ENDPROC`
	`ENDMODULE`
	xx2300000299
	
	 Messages
	Event log
	100%
	Axis 1-3
	T_ROB1
	ST_TEST
	disp_ex1
	Edit
	Cut
	Copy
	Paste
	Comment
	Delete
	Undo
	Redo
	Change to MoveJ
	Select Range
	Edit selection with keyboard
	Home
	Code
	14:58
	Enable
	`  PERS pose peEX1 := [[0,0,0],[1,0,0,0]`
	` `
	`PROC disp_ex1()`
	`  MoveJ *, v1000, z50, tWeldGun;`
	`  PDispSet peEX1;`
	`  MoveL *, v1000, fine, tWeldGun;`
	`  MoveL *, v1000, fine, tWeldGun;`
	`  MoveL *, v1000, fine, tWeldGun;`
	`  MoveL *, v1000, fine, tWeldGun;`
	`  MoveL *, v1000, fine, tWeldGun;`
	`  PDispOff;`
	`  MoveL *, v1000, fine, tWeldGun;`
	`ENDPROC`
	` `
	`PROC disp_ex2()`
	`  <SMT>`
	`ENDPROC`
	`ENDMODULE`
	xx2300000300

Continues on next page

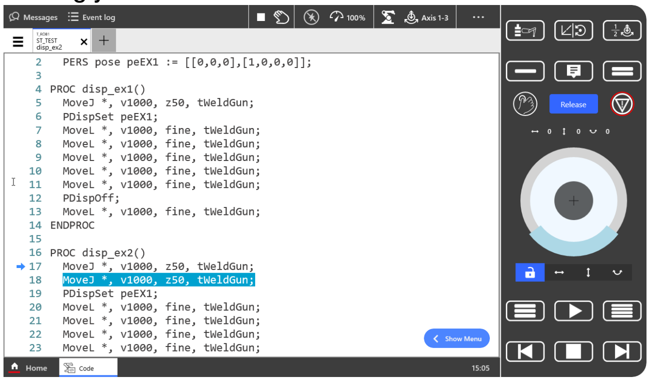
	Action

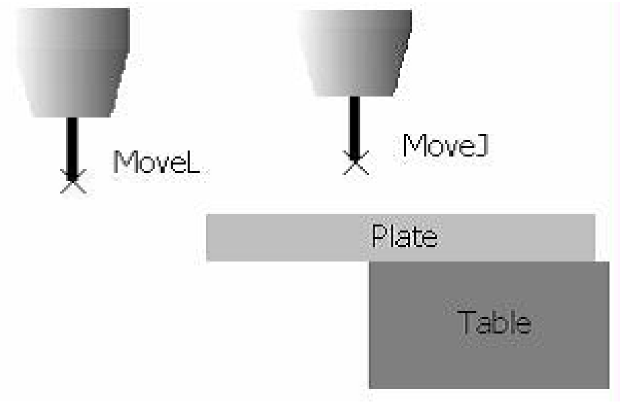
3	Change the default routine name, `disp_ex1Copy`, to `disp_ex2`, then look at the instructions in the new routine.
	
	Messages Event log 100% Axis 1-3
	T_ROB1 ST_TEST disp_ex2
	`  MoveJ *, v1000, z50, tWeldGun;`
	`  PDispSet peEX1;`
	`  MoveL *, v1000, fine, tWeldGun;`
	`  MoveL *, v1000, fine, tWeldGun;`
	`  MoveL *, v1000, fine, tWeldGun;`
	`  MoveL *, v1000, fine, tWeldGun;`
	`  MoveL *, v1000, fine, tWeldGun;`
	`  PDispOff;`
	`  MoveL *, v1000, fine, tWeldGun;`
	`ENDPROC`
	
	`PROC disp_ex2()`
	`  MoveJ *, v1000, z50, tWeldGun;`
	`  PDispSet peEX1;`
	`  MoveL *, v1000, fine, tWeldGun;`
	`  MoveL *, v1000, fine, tWeldGun;`
	`  MoveL *, v1000, fine, tWeldGun;`
	`  MoveL *, v1000, fine, tWeldGun;`
	`  MoveL *, v1000, fine, tWeldGun;`
	`  PDispOff;`
	`  MoveL *, v1000, fine, tWeldGun;ENDPROC`
	`ENDMODULE`
	Show Menu Home Code 15:01
	xx2300000301

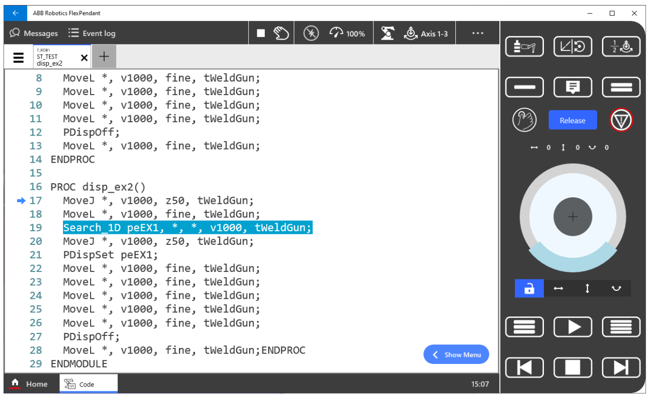
4	Remove the paper with the rectangle drawn on it.
5	Place a metal plate on the table so that a portion of the plate sticks out over the edge of the table.  xx1400001494
6	Edit the values in `peEX1` so that X, Y, and Z equal zero.
7	Move the program pointer to the new routine and toggle the Program Window to test mode.
8	Step through the rectangle program and modify each of the points so that they correspond to each of the plate's corners.
9	Continue to step though the routine until the program pointer loops back to the beginning.
10	Toggle the Program Window to instruction mode. If you have a Most Common picklist with SmarTac instructions, select it.

Continues on next page

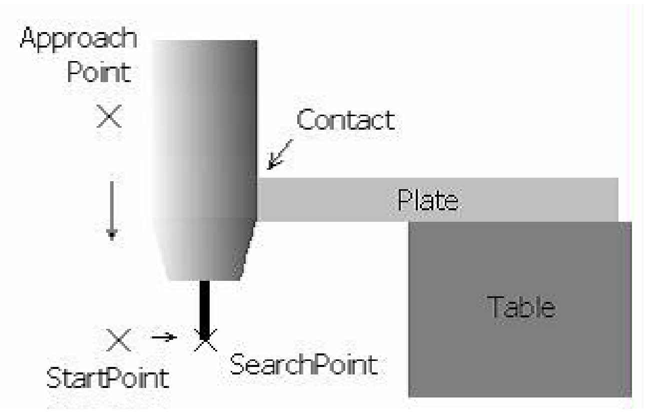
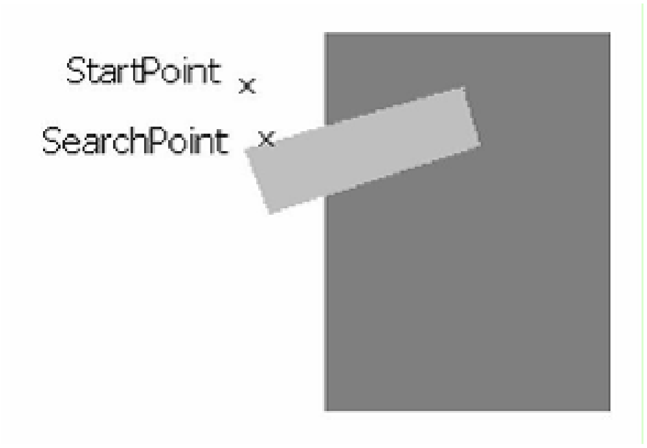
	Action

11	Move the cursor to the first line, `MoveJ`. Using the **Copy** and **Paste** buttons, copy the `MoveJ` and paste the copy below.
	We will add more movements and a search instruction between the two `MoveJ` instructions. 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.
	
	`  PERS pose peEX1 := [[0,0,0],[1,0,0,0]];`
	
	`PROC disp_ex1()`
	`  MoveJ *, v1000, z50, tWeldGun;`
	`  PDispSet peEX1;`
	`  MoveL *, v1000, fine, tWeldGun;`
	`  MoveL *, v1000, fine, tWeldGun;`
	`  MoveL *, v1000, fine, tWeldGun;`
	`  MoveL *, v1000, fine, tWeldGun;`
	`  MoveL *, v1000, fine, tWeldGun;`
	`  PDispOff;`
	`  MoveL *, v1000, fine, tWeldGun;`
	`ENDPROC`
	
	`PROC disp_ex2()`
	`  MoveJ *, v1000, z50, tWeldGun;`
	`  MoveJ *, v1000, z50, tWeldGun;`
	`  PDispSet peEX1;`
	`  MoveL *, v1000, fine, tWeldGun;`
	`  MoveL *, v1000, fine, tWeldGun;`
	`  MoveL *, v1000, fine, tWeldGun;`
	`  MoveL *, v1000, fine, tWeldGun;`
	 Messages
	Event log
	100%
	Axis 1-3
	T_ROB1
	ST_TEST
	disp_ex2
	Release
	Show Menu
	Home
	Code
	15:05
	xx2300000302

12	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.  xx1400001496

13	Insert the instruction, `Search_1D`, after the `MoveL`. Use the following data:
	`    Search_1D peEX1,*,*,v200,tWeldGun;`
	The routine should look like this:
	
	`  MoveL *, v1000, fine, tWeldGun;`
	`  MoveL *, v1000, fine, tWeldGun;`
	`  MoveL *, v1000, fine, tWeldGun;`
	`  MoveL *, v1000, fine, tWeldGun;`
	`  PDispOff;`
	`  MoveL *, v1000, fine, tWeldGun;`
	`ENDPROC`
	
	`PROC disp_ex2()`
	`  MoveJ *, v1000, z50, tWeldGun;`
	`  MoveL *, v1000, fine, tWeldGun;`
	`  Search_1D peEX1, *, *, v1000, tWeldGun;`
	`  MoveJ *, v1000, z50, tWeldGun;`
	`  PDispSet peEX1;`
	`  MoveL *, v1000, fine, tWeldGun;`
	`  MoveL *, v1000, fine, tWeldGun;`
	`  MoveL *, v1000, fine, tWeldGun;`
	`  MoveL *, v1000, fine, tWeldGun;`
	`  MoveL *, v1000, fine, tWeldGun;`
	`  PDispOff;`
	`  MoveL *, v1000, fine, tWeldGun;ENDPROC`
	`ENDMODULE`
	 ABB Robotics FlexPendant
	Messages
	Event log
	100%
	Axis 1-3
	T_ROB1
	ST_TEST
	disp_ex2
	Release
	Show Menu
	Home
	Code
	15:07
	xx2300000303

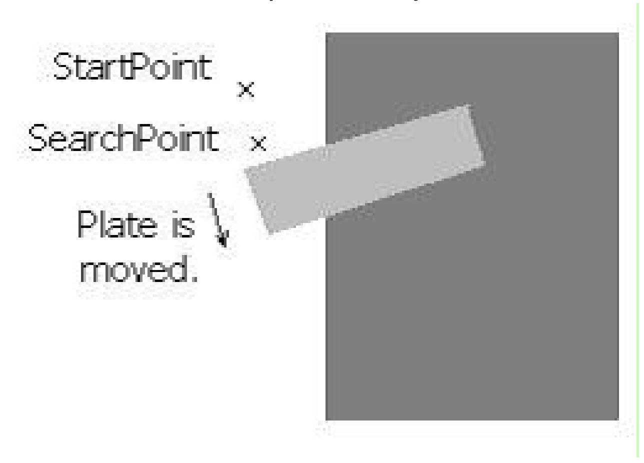
Continues on next page

	Action
14	Jog the robot to the `SearchPoint` and modify the position of the second robtarget in the `Search_1D` instruction. The gas cup should make contact with the plate without deflecting the torch.
15	Jog the robot to the `StartPoint` and modify the position of the first `robtarget` in the `Search_1D` instruction.  xx1400001498  xx1400001499
16	Toggle the Program Window to the test mode and move the program pointer to the `Search_1D` instruction.
17	Press the three-position enabling device and press the forward button. The robot should move to the `StartPoint`, then search towards the `SearchPoint`, until the plate is detected.

Continues on next page

	Action

18	Toggle the Program Window to instruction mode and using the **Copy** and **Paste** buttons, copy the `MoveL` before 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`

19	Run the routine from the beginning. The torch should search the plate and then trace out the plate.
20	Move the plate about 10 mm away from the `SearchPoint` and try running the routine (see the figure). If the plate was moved in the direction of the search, without any rotation, the torch should still trace out the plate correctly.  xx1400001500

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

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

Continues on next page

2 What errors occur when the plate is moved too far away? Experiment with the error recovery options to see what they do. See *Exercise 5: using SmarTac with work object manipulation on page 57* for details on error handing.

4.3.3 Programming tips

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 button. When the robot stops motion with the torch touching the part, move the cursor to the `SearchPoint` and modify the position of the robtarget. This ensures that a search on a perfect part will return a displacement that is very close to zero.

4.4 Exercise 3: using SmarTac for multi-dimensional searching

4.4.1 Introduction

About multi-dimensional search

As seen in exercise 2 (*Exercise 2: one-dimensional search on page 40*), 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.

Limitations for single and multi-dimensional searching

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. Exercise 3 demonstrates this for a two-dimensional search.

4.4.2 Exercise 3: two-dimensional search

About the exercise

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

The instruction `Search_1D` is described in *Search_1D - One-dimensional search on page 79*.

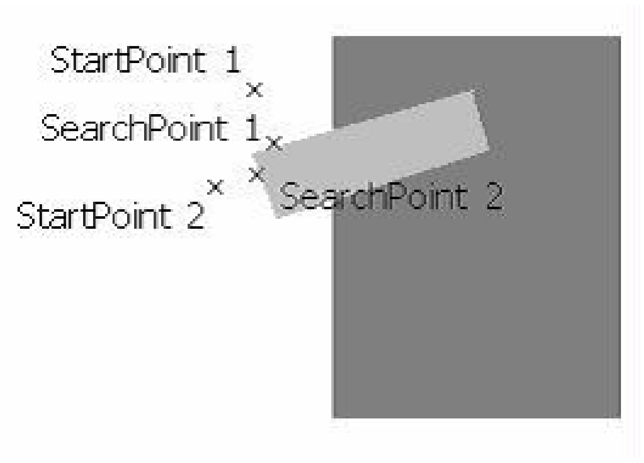
Instruction

	Action
1	View the routines in the module `ST_TEST`.
2	Select `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 instruction `PDispOff` near the end of the routine.
6	Press the three-position enabling device and press the step forward button once to execute the 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	Press the three-position enabling device and press the step forward button 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. Press the three-position enabling device and step forward until the search is complete and the robot stops at the `MoveL` after the instruction `Search_1D`.
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 new `MoveL`.
12	Copy the new `MoveL` and insert it after the last `Search_1D`.

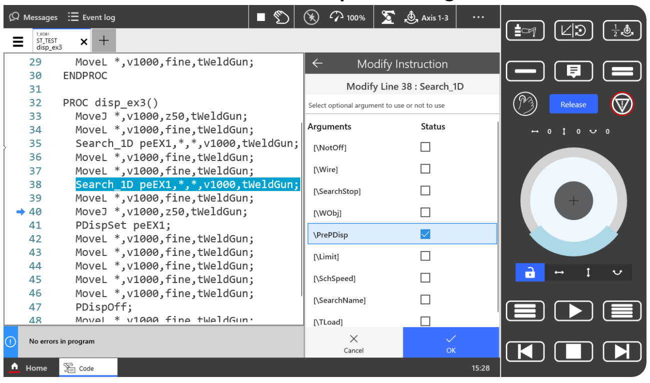
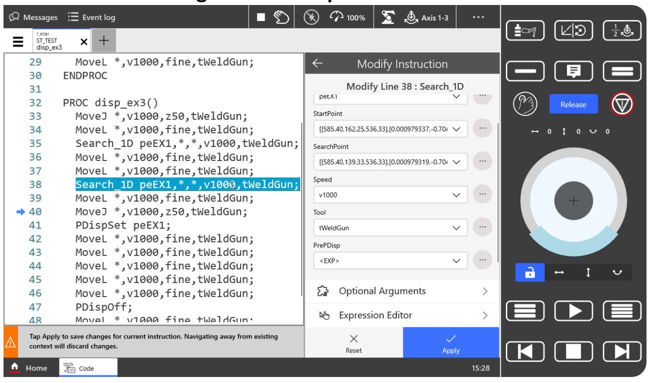
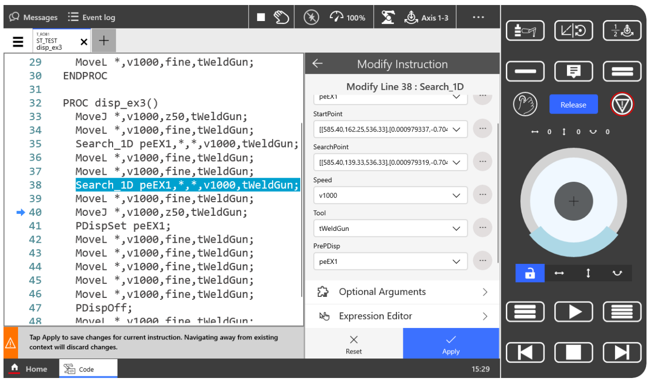
Continues on next page

	Action

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`
	`  Search_1D peEX1,*,*,v200,tWeldGun; ! New Search_1D`
	`  MoveL *,v200,fine,tWeldGun; ! Copy of MoveL`
	`  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`.  StartPoint 1 SearchPoint 1 StartPoint 2 SearchPoint 2 xx1400001501
15	Highlight the second `Search_1D` instruction and tap Enter.
16	Tap **OptArg** to look at the optional arguments.

Continues on next page

	Action
17	Move the cursor down to the optional argument named `PrePDisp` and tap **Use**.  xx2300000304
18	Tap **OK**.
19	Select the new argument to open the new window.  xx2300000305  xx2300000306

Screenshot xx2300000304:

Messages | Event log | 100% | Axis 1-3

T_ROB1 ST_TEST disp_ex3

```
  MoveL *,v1000,fine,tWeldGun;
ENDPROC

PROC disp_ex3()
  MoveJ *,v1000,z50,tWeldGun;
  MoveL *,v1000,fine,tWeldGun;
  Search_1D peEX1,*,*,v1000,tWeldGun;
  MoveL *,v1000,fine,tWeldGun;
  MoveL *,v1000,fine,tWeldGun;
  Search_1D peEX1,*,*,v1000,tWeldGun;
  MoveL *,v1000,fine,tWeldGun;
  MoveJ *,v1000,z50,tWeldGun;
  PDispSet peEX1;
  MoveL *,v1000,fine,tWeldGun;
  MoveL *,v1000,fine,tWeldGun;
  MoveL *,v1000,fine,tWeldGun;
  MoveL *,v1000,fine,tWeldGun;
  MoveL *,v1000,fine,tWeldGun;
  PDispOff;
  MoveL *,v1000,fine,tWeldGun;
```

Modify Instruction

Modify Line 38 : Search_1D

Select optional argument to use or not to use

Arguments	Status
[\NotOff]	☐
[\Wire]	☐
[\SearchStop]	☐
[\WObj]	☐
\PrePDisp	✓
[\Limit]	☐
[\SchSpeed]	☐
[\SearchName]	☐
[\TLoad]	☐

Cancel | OK

No errors in program

Home | Code | 15:28

Screenshot xx2300000305:

Modify Instruction

Modify Line 38 : Search_1D

peEX1

StartPoint: [[585.40,162.25,536.33],[0.000979337,-0.704

SearchPoint: [[585.40,139.33,536.33],[0.000979319,-0.704

Speed: v1000

Tool: tWeldGun

PrePDisp: <EXP>

Optional Arguments

Expression Editor

Reset | Apply

Tap Apply to save changes for current instruction. Navigating away from existing context will discard changes.

Home | Code | 15:28

Screenshot xx2300000306:

Modify Instruction

Modify Line 38 : Search_1D

peEX1

StartPoint: [[585.40,162.25,536.33],[0.000979337,-0.704

SearchPoint: [[585.40,139.33,536.33],[0.000979319,-0.704

Speed: v1000

Tool: tWeldGun

PrePDisp: peEX1

Optional Arguments

Expression Editor

Reset | Apply

Tap Apply to save changes for current instruction. Navigating away from existing context will discard changes.

Home | Code | 15:29

Continues on next page

	Action

20	From the list of available pose data select `peEX1`. Tap **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 10 mm in any direction and re-execute the routine. The torch should trace out the plate.

Questions

1. Why is it necessary that the `PrePDisp` is 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 directions that are perpendicular to one another. Why?

Advanced

1. What happens when the plate is rotated slightly? Why?
2. Add the optional argument `NotOff` to the first search instruction and execute the program. What does this do?

Tip

See *Search_1D - One-dimensional search on page 79*.

Continues on next page

3 What would happen if the argument `NotOff` was added to the second search and the next section of the routine had a welding instruction?

Tip

See *Search_1D - One-dimensional search on page 79*.

4 Version 7.0 only: Why must there always be at least one Move instruction between two searches?

Tip

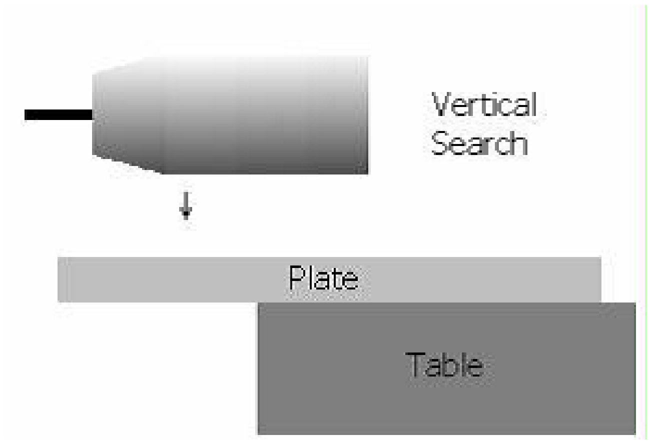
What happens when SmarTac is activated while the torch is touching the part?

5 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
```

Continues on next page

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



xx1400001504

6 In Example 1, what happens if you search the same edge twice using `PrePDisp` to add the second search result to the first?

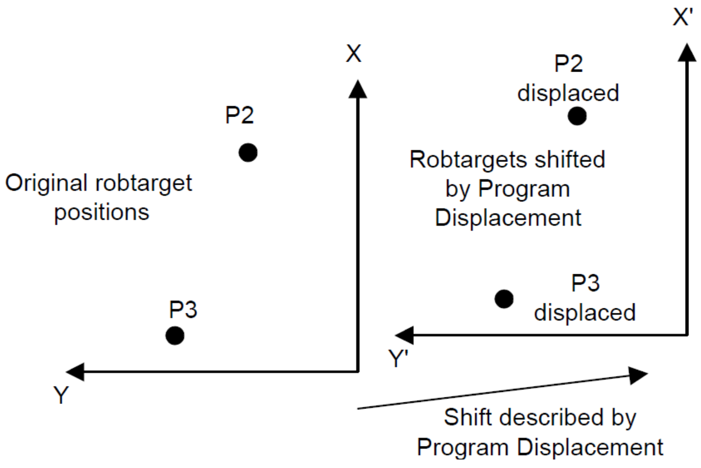
4.5 Exercise 4: using SmarTac to determine simple rotational changes

4.5.1 Introduction

Translation and rotation

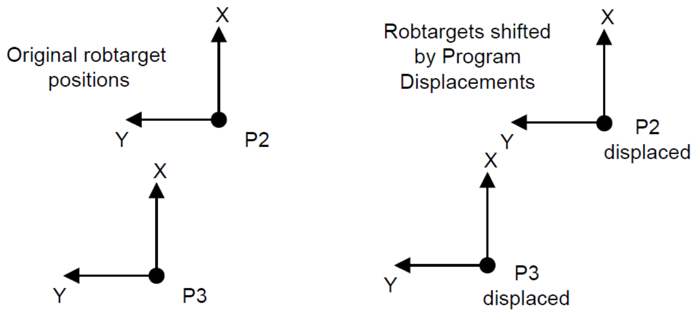
Up to this point basic one dimensional searches have been used to accurately locate part features that have moved only in translation, not rotation.

Using SmarTac to determine simple rotational changes



xx1400001505

Using this same basic concept, it is possible to search a weld seam that moves both in translation and rotation. Imagine that the robtargets `P2` and `P3` in the illustration, 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 rotational as well.



xx1400001506

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. See *Exercise 4: part feature with simple rotation on page 55*.

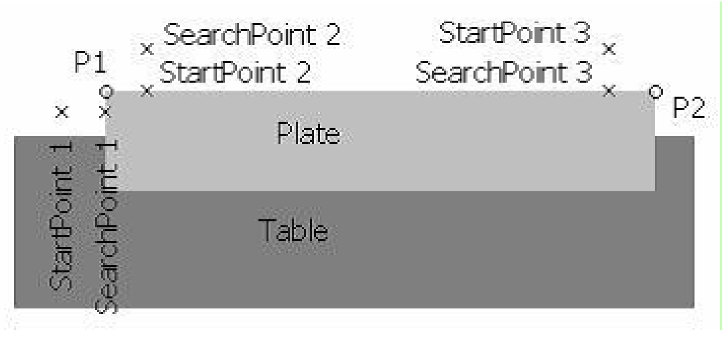
4.5.2 Exercise 4: part feature with simple rotation

About the exercise

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

Instructions

	Action

1	Create a new routine called `disp_ex4` that looks like this:
	`PROC disp_ex4()`
	`  MoveJ *, v200, fine, tWeldGun;`
	`  MoveL *, v200, fine, tWeldGun;`
	`  ! Search 1`
	`  Search_1D peEX1, *, *, v200, tWeldGun;`
	`  MoveL *,v200,fine,tWeldGun;`
	`  MoveL *,v200,fine,tWeldGun;`
	`  ! Search 2`
	`  Search_1D peEX2, *, *, v200, tWeldGun\PrePDisp:=peEX1;`
	`  MoveL *, v200, fine, tWeldGun;`
	`  MoveL *, v200, fine, tWeldGun;`
	`  ! Search 3`
	`  Search_1D peEX3, *, *, v200, tWeldGun\PrePDisp:=peEX1;`
	`  MoveL *, v200, fine, tWeldGun;`
	`  MoveL *, v200, fine, tWeldGun;`
	`  PDispSet peEX2;`
	`  ! Corner 1`
	`  MoveL P1, v200, fine, tWeldGun;`
	`  PDispSet peEX3;`
	`  ! Corner 2`
	`  MoveL P2, v20, fine, tWeldGun;`
	`  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 pre-displacement is called for each search instruction.
	The searches should look similar to those used in exercise 2 & 3 (see *on page 43*), but positioned around the plate like this:
	 SearchPoint 2
	StartPoint 2
	StartPoint 3
	SearchPoint 3
	P1
	P2
	Plate
	Table
	StartPoint 1
	SearchPoint 1
	xx1400001507

Continues on next page

	Action
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 movements to clear the plate.
4	Step through the routine to test the positions. If `P1` and `P2` are out of position, you can jog them into position and modify the positions with the program displacement turned on.
5	Execute the program from the beginning. Watch the robot trace the edge of the plate.
6	Move the plate in various directions, including rotation, 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 *Exercise 3: two-dimensional search on page 48*?
2. Look at the values in each of the program displacements `peEX1`, `peEX2`, and `peEX3`. What are the values for the rotation portions, q1-q4?

Advanced

1. 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?
2. Why would it be difficult to shift an intermittent stitch weld in this way?

4.6 Exercise 5: using SmarTac with work object manipulation

4.6.1 Introduction

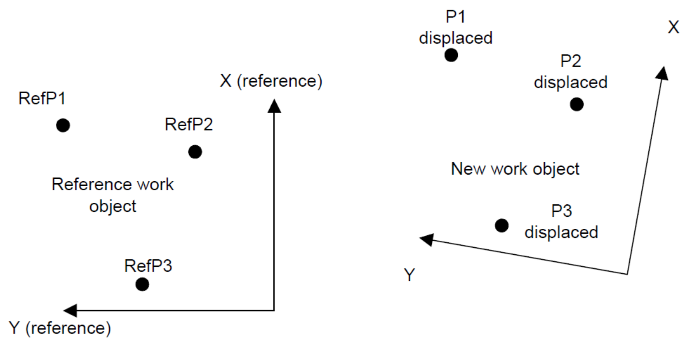
Weld paths

Sometimes using multiple displacement frames can not provide an easy way of determining a weld seam's location. In exercise 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, that is a stitch weld, this would not work. There is no displacement information about the intermediate weld points.

Work object

In some cases 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.

Example 1



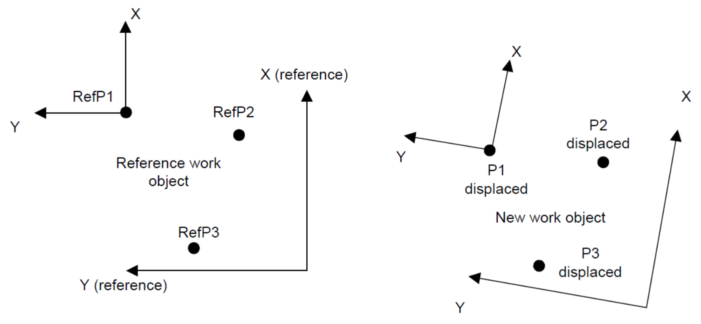
xx1400001508

Note

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.

Continues on next page

Example 2



xx1400001509

In this example the 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.

4.6.2 SmarTac functions

Mathematical functions

Two global mathematical functions are provided in the SmarTac module.

- *PoseAdd - Adds the translation portions of pose data on page 99*
- *OFrameChange - Create a new shifted object frame on page 100*

Exercise 5 will illustrate the usage of these mathematical tools.

PoseAdd

`PoseAdd` is a simple function used to add the transport 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 `Search_1D`.

OFrameChange

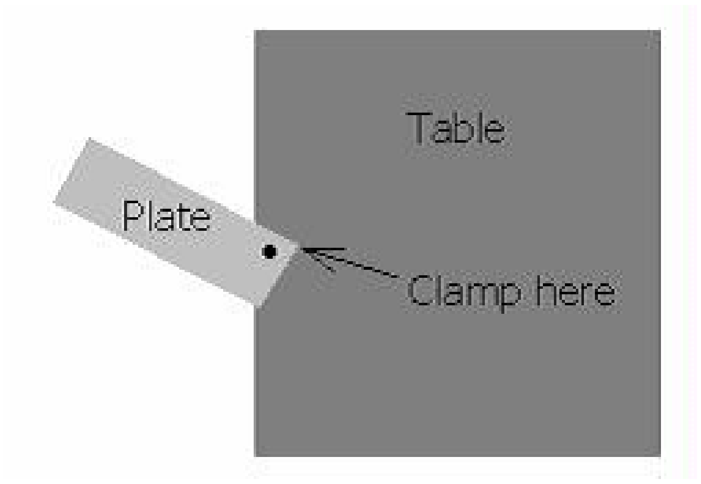
`OFrameChange` uses seven arguments; a reference work object, three reference points, and three displacement frames. The function returns `wobjdata`. In use it looks like this:

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

4.6.3 Exercise 5: object frame manipulation

About the exercise

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.

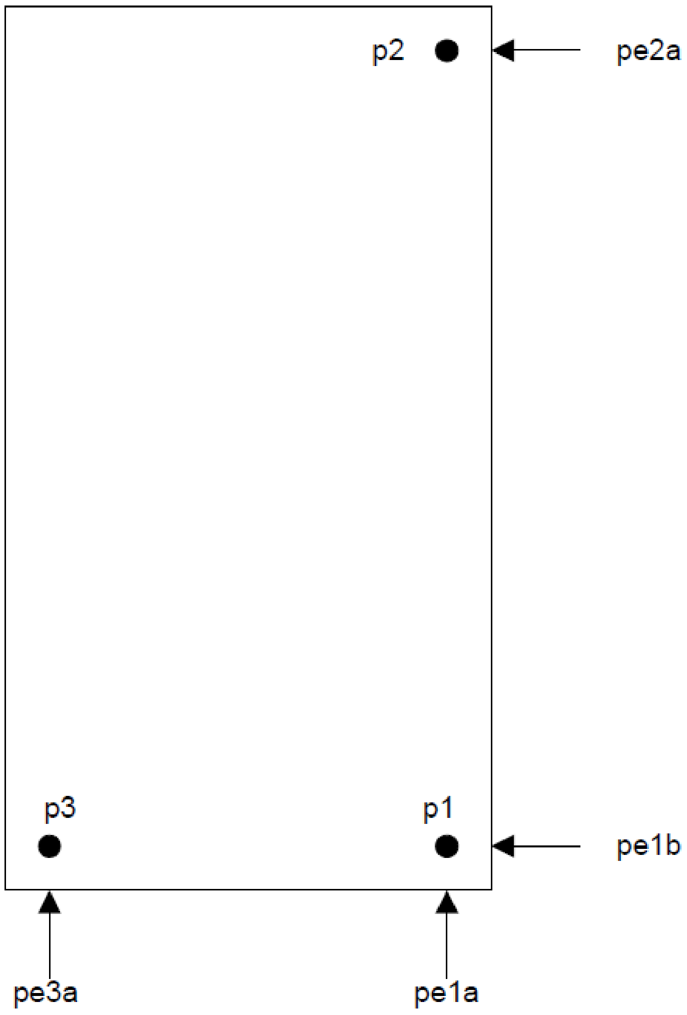


xx1400001510

Continues on next page

Reference sketch

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



xx1400001511

Code for OFrame

```
! 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]];
```

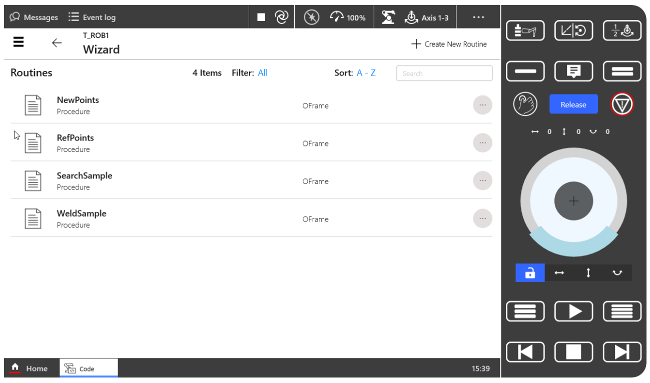
Continues on next page

```
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;
```

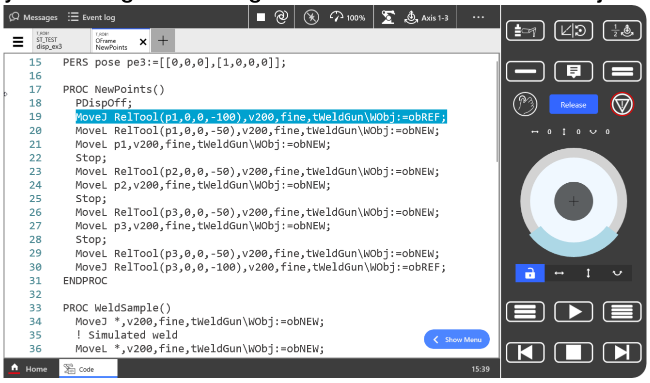
Continues on next page

```
            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
```

Instruction

	Action
1	Load the program module `OFrame`. **Tip** The module is included in the add-in. It is also shown in *Code for OFrame on page 61*.
2	View the routines in the module. You should see the following routines:  T_ROB1 Wizard Routines 4 Items Filter: All Sort: A - Z NewPoints Procedure OFrame RefPoints Procedure OFrame SearchSample Procedure OFrame WeldSample Procedure OFrame Messages Event log 100% Axis 1-3 Create New Routine Search Release Home Code 15:39 xx2300000307
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 *Reference sketch on page 61* or step *14*.
4	Move the program pointer to the procedure `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 button. (This ensures that no displacements are active.)
6	Jog the robot so that the torch is situated about 150 mm above the plate surface and pointing down at the plate.

Continues on next page

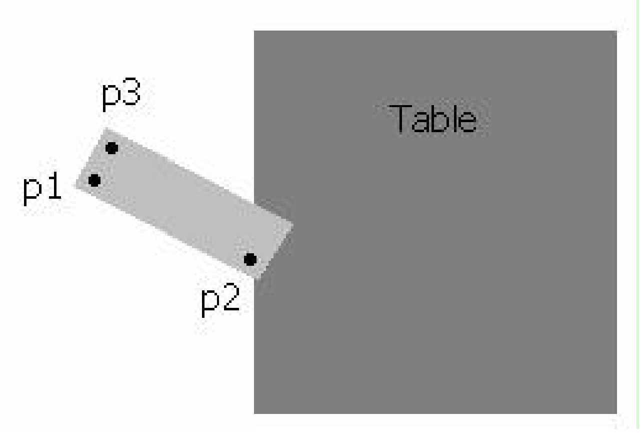
	Action
7	Move the cursor to the first `MoveJ` and modify the position. You may have to change your settings in the Jog Window to reflect the work object change. 

Messages | Event log | 100% | Axis 1-3

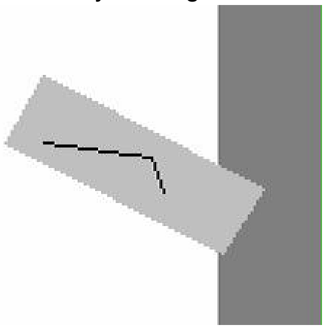
T_ROB1 ST_TEST disp_ex3 | T_ROB1 OFrame NewPoints

```
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;
```

xx2300000308

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 150 mm 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 `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`.  xx1400001514
15	Jog the robot down so that the torch gas cup is in a position to search the edge of the plate. See *Reference sketch on page 61*, and modify the points the first `Search_1D`. The search direction is indicated in the sketch for the displacement frame `pe1a` (the first search result). Remember that the search direction should be perpendicular to the edge of the part.
16	Modify all the rest of the moves and searches as shown in *Reference sketch on page 61*.
17	Test run the `SearchSample` procedure.

Continues on next page

	Action
18	Move the program pointer to the routine called `WeldSample`. `WeldSample` does not have any `ArcL` instructions so RobotWare Arc does not need to be present to load this module. It has only `MoveL` instructions with slow speeds to simulate welding.
19	Draw a simulated weld on the surface of the plate using a straight edge and marker. `WeldSample` has only two segments. Add more if desired.  xx1400001515
20	Modify the first point to be above the plate at least 100 mm.
21	Modify the second point to be the start of the simulated weld.
22	Modify the third and fourth points to be the middle and end of the weld.
23	Modify the last point to be above the plate at least 100 mm.
24	Run `WeldSample` to be sure it follows the line correctly.
25	Run `SearchSample`.
26	Run `NewPoints`. The points `p1`, `p2`, and `p3` should be pointed out correctly. If not, there is a mistake somewhere. Check your program.
27	Run `WeldSample` again to be sure everything is ok. If the path is not followed, check the program again.
28	Leaving the plate clamped at the corner, move the plate about 10 mm at the end.
29	Run `SearchSample`.
30	Run `WeldSample`. The path should follow correctly.
31	Run `NewPoints`. The points should be pointed out correctly.

Questions

1. What work object is used in `RefPoints` and `SearchSample`?
2. What work objects 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?

Continues on next page

9 Would this be a concern for most real-world fixtures?

Advanced

1 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.

2 Select `obREF` as the work object and `WObj` for the coordinate system in the Jogging 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.

3 Go though Example 5 again with the new work object definition. How might this help when programming?

4.7 Exercise 6: Search_Part

4.7.1 Introduction

About 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`.

See *Search_Part - Search for feature presence on page 93*.

Example

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
```

4.7.2 Exercise 6: using Search_Part

Instruction

	Action
1	Create a new procedure in the module `ST_TEST`, named `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

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

4.8 Exercise 7: wire searching

4.8.1 Introduction

About wire searching

Sometimes it is necessary to search with the welding wire, rather than the gas cup. In some systems with the necessary optional hardware installed, this is possible. The SmarTac instructions are designed to handle this. `Search_1D` and `Search_Part` each have an optional argument, `Wire`, that will switch the signal to the wire if selected. The instruction `Search_Groove` assumes that wire searching capabilities are present.

4.8.2 Exercise 7: wire searching

Note

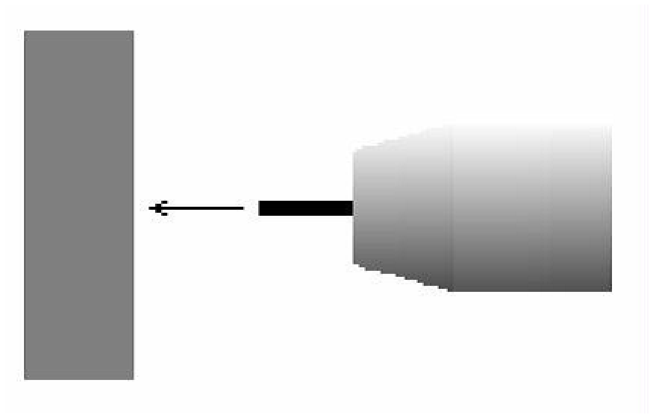
This exercise can only be done on systems that have wire searching capability.

Instruction

	Action
1	Add the optional argument `Wire` to the `Search_Part` instruction used in exercise 6.
2	Move the search points so that the wire will touch the part instead of the gas cup.

Questions

1. Did the system work correctly?
2. What problems could arise from searching with the side of the wire?
3. What problems could arise when searching with the tip of the wire in the direction of the wire? See the following figure.



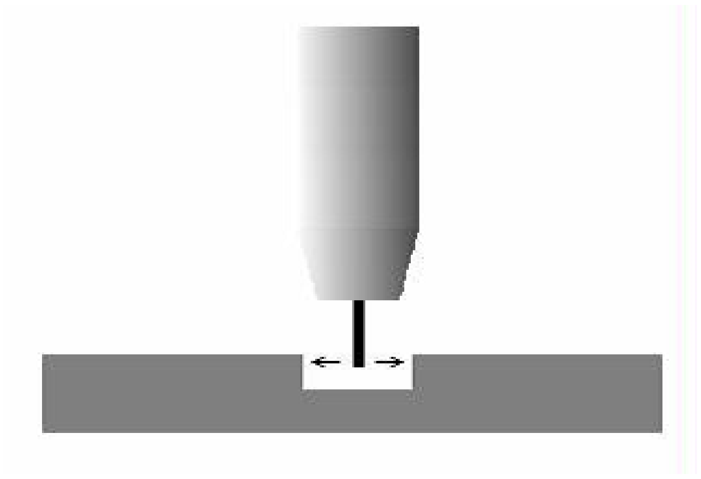
xx1400001516

4.9 Exercise 8: searching for a groove

4.9.1 Introduction

Example of weld seam

In heavy welding applications it is common to have *groove* type weld seams. The simplest example is the square groove with a backing.



xx1400001517

Searching

In heavy welding tolerances are usually large so searching is critical in determining the location and width of seams like the one above. The instruction `Search_Groove` has been provided to satisfy this need.

Note

See *Search_Groove - Find groove width and location on page 86*.

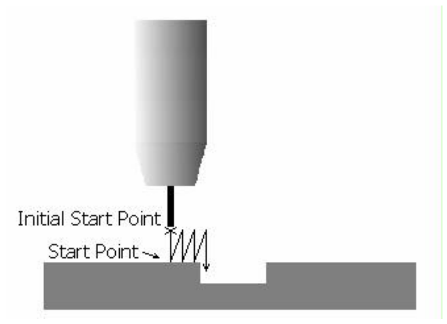
Requirements for Search_Groove

`Search_Groove` performs a series of searches when executed. It requires two robtargets. One is programmed outside the groove, and the other in the center of the groove. It requires a displacement frame that will be returned as the seam offset. It requires a number that will be returned as the actual width of the seam. It also requires a nominal width, a speed, and a tool.

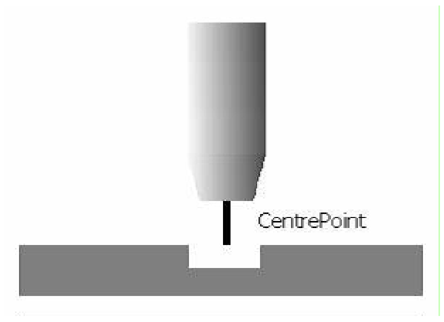
Continues on next page

Illustration

In use it looks like this:



xx1400001518



xx1400001519

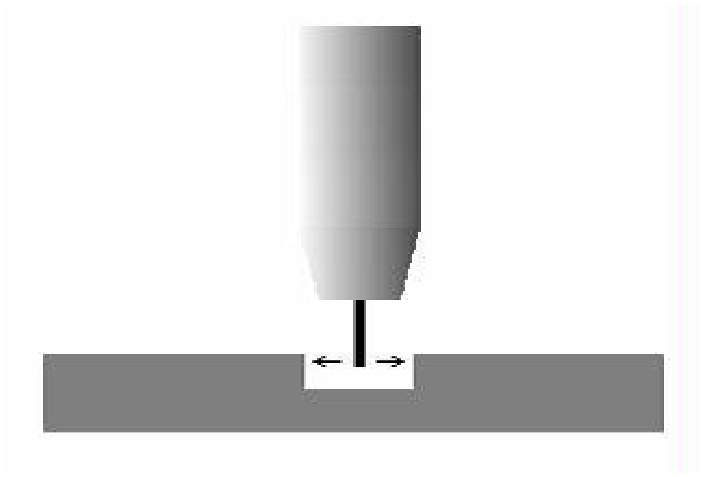
```
Search_Groove peOffset, nWidth, p1, p2, 10, v200, tWeldGun;
```

Facts

The groove search is used to find the location and width of the groove to be welded.

- The groove has a 10 mm nominal width.
- The program displacement is stored in `peOffset`.
- The actual width of the groove is stored in `nWidth`.
- The `StartPoint` is `p1` and the `CentrePoint` is `p2`.
- The `Initial Start Point` is 15 mm above the `StartPoint` by default.

Searching for groove width and groove location



xx1400001520

4.9.2 Exercise 8: Searching for a groove weld

Instruction

	Action
1	Build a seam something like the one shown in *Searching for groove width and groove location on page 72*.
2	Try programming a `Search_Groove` instruction using the information in *Search_Groove - Find groove width and location on page 86*. Remember: Only the wire can be used on a groove search.

Questions

Use *Search_Groove - Find groove width and location on page 86* as a reference in answering these questions.

1. Where should the first robtarget, `StartPoint`, be programmed?
2. Where should the second robtarget, `CentrePoint`, be programmed?
3. What effect does changing the nominal groove width, `NomWidth`, have on the search pattern?
4. What effect does it have on the results (displacement & actual width)?
5. Which moves are effected by changes in the `speeddata`?

Advanced

	Action
1	Add the optional arguments, `InitSchL` and `NomDepth`.
2	Set the `InitSchL` equal to 15.
3	Set the `NomDepth` equal to 3.

1. What happens when `InitSchL` is changed?
2. What happens when `NomDepth` is changed?

4.10 Conclusions

About the overview

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 *Instructions on page 79*. There is also an instruction called `PDispAdd` which is used with the same effect as the function `PoseAdd`.

Work objects

Work objects are described in *Operating manual - OmniCore*. 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.

5 Troubleshooting

5.1 False-positive torch contact

Description

An error message appears on the screen stating that the torch has made contact with the part, before searching has begun, but the torch is clearly not touching the part. Or an activation error message appears on the screen.

5.2 No detection of the part

Description

The robot never detects the part when searching. A crash results.

Possible causes

1. Check that the sensing surface is free of dirt, soot, etc. that would otherwise prevent good electrical contact. Clean the cup at regular intervals. Grind any non-conductive coatings from the part that is to be searched.
2. Activate the welders touch sense functionality by setting the *Sensor On Output*. Check that there is the correct VDC measured between the torch sensing surface and the part. (Voltage level depends on the used power source.) Ideally there should be about 50-70 VDC present. A reading lower than the voltage specified by the used power source indicates that there is a significant loss that will make search results inaccurate.

5.3 Inaccurate results

Description

Search results are inaccurate.

Possible causes

1. Search result data is being used improperly. Verify that the RAPID search instructions are being used correctly. See *User's guide on page 33*.
2. The sensing voltage is too low. Activate the welders touch sense functionality by setting the *Sensor On Output*. Check that there is the correct VDC measured between the torch sensing surface and the part. (Voltage level depends on the used power source.) Ideally there should be about 50-70 VDC present. A reading lower than the voltage specified by the used power source indicates that there is a significant loss that will make search results inaccurate. Low voltage can result from contaminated coolant, defective torches, and grounded welding wire. See *False-positive torch contact on page 75* for more details.
3. The TCP is moving. Check to see that the torch is securely mounted to the robot and that no movement is detected when searching.

6 RAPID reference

6.1 Instructions

6.1.1 Search_1D - One-dimensional search

Usage

`Search_1D` is an instruction used for tactile searching 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.

Basic examples

```
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] [\Wire] Result [\SearchStop] StartPoint
     SearchPoint Speed Tool [\WObj ] [\PrePDisp] [\Limit]
     [\SearchName] [\TLoad]
```

`[\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.

`[\Wire]`

Data type: `switch`

If selected the output `doWIRE_SEL` will be set high when the SmarTac activation occurs. The SmarTac sensor will be switched from the gas cup to the wire when selected.

`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`

Continues on next page

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` (in mm).

`[\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 event log.

`[\TLoad]`

Data type: `loaddata`

The `\TLoad` argument describes the total load used in the movement. The total load is the tool load together with the payload that the tool is carrying. If the `\TLoad` argument is used, then the `loaddata` in the current `tooldata` is not considered.

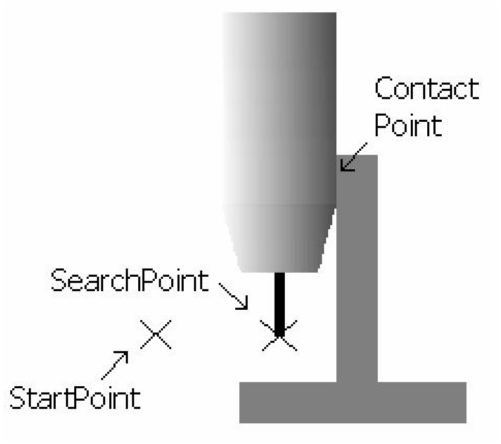
If the `\TLoad` argument is set to `load0`, then the `\TLoad` argument is not considered and the `loaddata` in the current `tooldata` is used instead. For a complete description of the `TLoad` argument, see `MoveL` in *Technical reference manual - RAPID Instructions, Functions and Data types*.

Continues on next page

Program execution

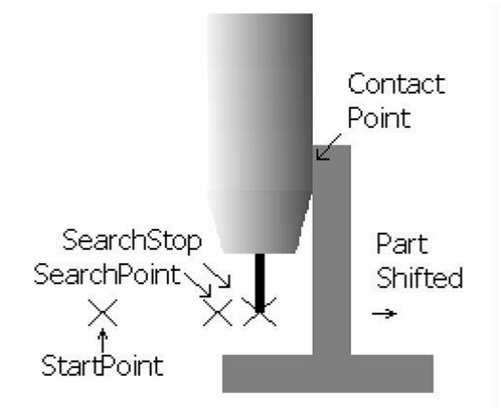
When executed, the robot makes a linear movement to the start point, `StartPoint`. The SmarTac board is activated and motion starts towards the search point, `SearchPoint`. The robot will continue past the search point for a total search distance described by twice the distance between `StartPoint` and `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`.

Normally the gas cup is used for searching, however, on some systems the wire can be used for searching. When the switch, `Wire`, is selected, the digital output, `doWIRE_SEL`, is set high. This switches the SmarTac signal from the gas cup to the wire.



xx1400001490

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.



xx1400001491

Continues on next page

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.

Error handling

Fault	Menu message
Fault 1	Activation of the SmarTac failed
Fault 2	Search failed
Fault 3	`GasCup` or `Wire` touching part

Fault 1

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 preoffset included in the search instruction. A message will be logged in the event log.

Fault 2

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

Search 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 preoffset included in the search instruction. A message will be logged in the event log.

Continues on next page

Fault 3

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

GasCup or Wire touching part

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 preoffset included in the search instruction. A message will be logged in the event 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 event log.

More 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;
PDispSet pose1;
ArcL\On,*, vmax, sm1, wd1, wv1, z1, tWeldGun\Wobj:= wobj2;
ArcL\Off,*, vmax, sm1, wd1, wv1, z1, tWeldGun\Wobj:= wobj2;
```

Continues on next page

```
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;
```

Note

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 wire active and the maximum limit set at 4 mm. If the magnitude of the transport of peOffset is greater than 4 mm an error is flagged.

```
Search_1D\Wire, 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 event log. See *Error handling on page 82*.

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

Syntax

```
Search_1D
  ['\ ' NotOff ',']
  ['\ ' Wire ',']
  [ 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 > ]
  [ '\' TLoad ':=' ] < persistent (PERS) of loaddata > ] ';'
```

Related information

	Described in
Search_Groove	*Search_Groove - Find groove width and location on page 86*
Search_Part	*Search_Part - Search for feature presence on page 93*
Data type pose	*Technical reference manual - RAPID Instructions, Functions and Data types*

Continues on next page

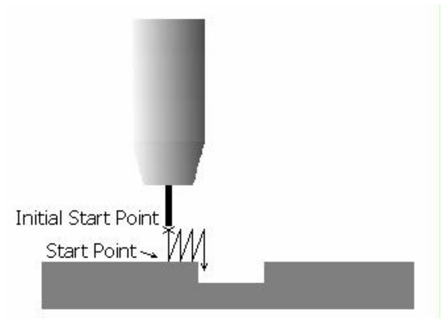
	Described in
Data type `wobjdata`	*Technical reference manual - RAPID Instructions, Functions and Data types*
Data type `robtarget`	*Technical reference manual - RAPID Instructions, Functions and Data types*
`MoveL`	*Technical reference manual - RAPID Instructions, Functions and Data types*
Definition of `loaddata`	*Technical reference manual - RAPID Instructions, Functions and Data types*

6.1.2 Search_Groove - Find groove width and location

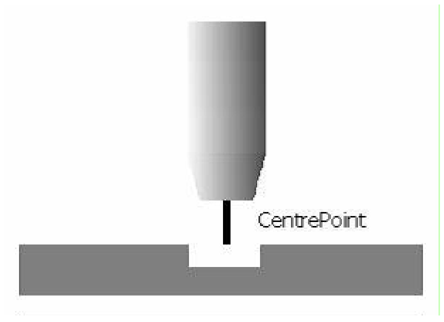
Usage

`Search_Groove` is an instruction used for tactile searching of a "groove" with SmarTac. Searching is done with the wire. A series of searches are preformed to find the groove and determine its width. The `StartPoint` is programmed outside the groove at a point touching the part. The `CentrePoint` is programmed level with the `StartPoint`, but in the center of the groove. The search result is stored as pose data in the required argument Result. All SmarTac board activation and deactivation is automatically handled.

Basic examples



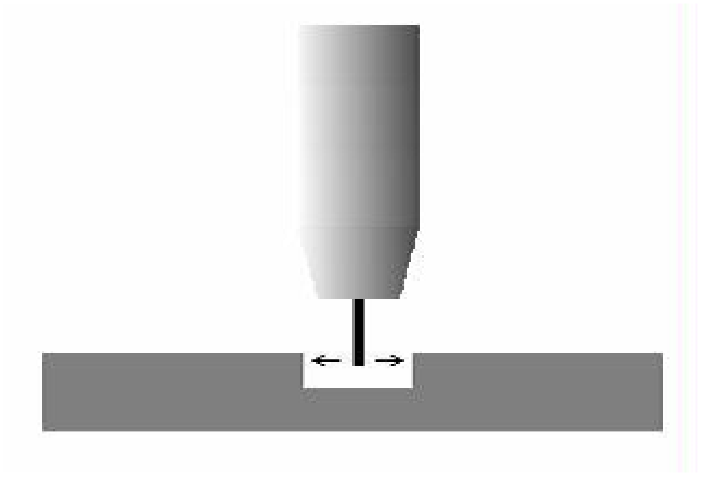
xx1400001521



xx1400001522

```
Search_Groove peOffset,nWidth,p1,p2,10,v200,tWeldGun;
```

The groove search is used to find the location and width of the groove to be welded. The groove has a 10 mm nominal width. The program displacement is stored in `peOffset`. The actual width of the groove is stored in `nWidth`. The `StartPoint` is `p1` and the `CentrePoint` is `p2`. The `Initial Start Point` is 15 mm above the `StartPoint` by default.



xx1400001520

Continues on next page

Arguments

```
Search_Groove [\NotOff] Result GrooveWidth [\SearchStop] StartPoint
     CentrePoint NomWidth [\NomDepth] [\InitSchL] Speed Tool [\WObj
     ] [\PrePDisp] [\SearchName] [\TLoad]
```

`[\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

`GrooveWidth`

Data type: `num`

The calculated groove width (in mm) determined by the search.

`[\SearchStop]`

Data type: `robtarget`

If selected, this robtarget will be updated as the point where the center of the groove is.

`StartPoint`

Data type: `robtarget`

The start point of the search sequence. This point should be programmed outside the groove, touching the part surface with the wire's tip. See the initial start point in *Basic examples on page 86*.

`CentrePoint`

Data type: `robtarget`

The point where the groove should be. This robtarget should be programmed so that the wire's tip is above the center of the groove, level with the adjacent part surface. See the center point in *Basic examples on page 86*.

`NomWidth`

Data type: `num`

The expected groove width (in mm). This number will effect the dimensions of the search sequence.

`[\NomDepth]`

Data type: `num`

The expected groove depth (in mm). If selected, this number will effect the dimensions of the search sequence. The default is 2.5 mm.

`[\InitSchL]`

Data type: `num`

Continues on next page

The length of the first search. If selected, this changes the `Initial Start Point`. The default is 15 mm. See the initial start point in *Basic examples on page 86*.

`speed`

Data type: `speeddata`

The speed data used when moving to the `Initial Start Point`. 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`.

`[\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 event log.

`[\TLoad]`

Data type: `loaddata`

The `\TLoad` argument describes the total load used in the movement. The total load is the tool load together with the payload that the tool is carrying. If the `\TLoad` argument is used, then the `loaddata` in the current `tooldata` is not considered.

If the `\TLoad` argument is set to `load0`, then the `\TLoad` argument is not considered and the `loaddata` in the current `tooldata` is used instead. For a complete description of the `TLoad` argument, see `MoveL` in *Technical reference manual - RAPID Instructions, Functions and Data types*.

Program execution

When executed, the robot makes a linear movement to a point above the start point, the `Initial Start Point`. The height of the `Initial Start Point` above the `StartPoint` can be changed by the optional parameter, `InitSchL`. The SmarTac board is activated and motion starts towards the `StartPoint` (see initial start point in *Basic examples on page 86*).

The robot will continue past the `StartPoint` for a total search distance described by twice the distance between the `Initial Start Point` and the `StartPoint`.

Continues on next page

When the surface of the plate is found, more searches occur, each one closer to the edge of the groove.

When the groove is found, two searches are made inside the groove to determine the location and width (see *Searching for groove width and groove location on page 72*).

The start of both searches is beneath the `CentrePoint` (see the center point in *Basic examples on page 86*, and *Searching for groove width and groove location on page 72*). The optional parameter, `NomDepth`, will control how far into the groove the width and location searches will be. The displacement data is stored in Result.

This program displacement can later be used to shift programmed points using the RAPID instruction `PDispSet`. The width of the groove is stored in `GrooveWidth`.

Limitations

To use `Search_Groove`, the system must have wire-searching capability.

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.

Error handling

Fault	Menu message:
Fault 1	Activation of the SmarTac failed
Fault 2	Search failed
Fault 3	GasCup or Wire touching part
Fault 4	Groove not found
Fault 5	Groove search failed

Fault 1

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 preoffset included in the search instruction. A message will be logged in the User Error Log.

Continues on next page

Fault 2

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

Search 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 preoffset included in the search instruction. A message will be logged in the User Error Log.

Fault 3

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

GasCup or Wire touching part

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 preoffset included in the search instruction. A message will be logged in the User Error Log.

Fault 4

If the groove walls are not found when searching for the groove width and location, the following message appears:

Groove not found

RETURN	Continues the program with default search result.
RAISE	Sends error to calling routine.

When RETURN is selected a default search result is used which will include any preoffset included in the search instruction. A message will be logged in the User Error Log.

Continues on next page

Fault 5

If an error occurs when searching for the groove width and location, the following message appears:

Groove search 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 robot tries the search again.

When RETURN is selected a default search result is used which will include any preoffset included in the search instruction. A message will be logged in the User Error Log.

More examples

The groove search is used to find the location and width of the groove to be welded. The program displacement is stored in `peOffset` and the width of the groove is stored in `nWidth`. The weave width in this example is set to `nWidth`.

```
MoveJ *, vmax,fine, tWeldGun;
Search_Groove peOffset,nWidth,p1,p2,10,v200,tWeldGun;
WvAdapt.weave_width:=nWidth;
PDispSet peOffset;
ArcL\On,*,vmax,sm1,wd1,wvAdapt,fine,tWeldGun;
ArcL\Off,*, vmax,sm1,wd1,wvAdapt,fine,tWeldGun;
PDispOff;
```

Groove search with optional returned robtarget

The robtarget `p3` is updated with the actual groove centerline:

```
Search_Groove\SearchStop:= p3, pose1, nWidth, p1, p2, 10, v200,
     tWeldGun;
```

Groove search that is "named"

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 event log. See *Error handling on page 89*.

```
Search_Groove peOffset, nWidth, p1, p2, 15, v200,
     tWeldGun\SearchName:= "First";
```

Groove search that has a 30 mm first-search instead of the default 15 mm

```
Search_Groove peOffset, nWidth\InitSchL:= 30, p1, p2, 7, v200,
     tWeldGun;
```

Syntax

```
Search_Groove
  ['\ ' NotOff ',']
  [ Result ':=' ] < expression (INOUT) of pose > ','
  [ GrooveWidth ':=' ] < expression (INOUT) of num >
  [ '\' SearchStop ':=' < expression (INOUT) of robtarget >','
  ]
  [ StartPoint ':=' ] < expression (IN) of robtarget > ','
  [ CentrePoint ':=' ] < expression (IN) of robtarget > ','
```

Continues on next page

```
[ NomWidth ':=' ] < expression (IN) of num >
[ '\' NomDepth ':=' < expression (IN) of num > ]
[ '\' InitSchL ':=' < expression (IN) of num > ] ','
[ Speed ':=' ] < expression (IN) of speeddata > ','
[ Tool ':=' ] < persistent (PERS) of tooldata >
[ '\' WObj ':=' < persistent (PERS) of wobjdata > ]
[ '\' PrePDisp ':=' < expression (IN) of pose > ]
[ '\' SearchName ':=' < expression (IN) of string > ]
[ '\' TLoad ':=' ] < persistent (PERS) of loaddata > ] ';'
```

Related information

	Described in:
`Search_1D`	*Search_1D - One-dimensional search on page 79*
`Search_Part`	*Search_Part - Search for feature presence on page 93*
Data type `pose`	*Technical reference manual - RAPID Instructions, Functions and Data types*
Data type `wobjdata`	*Technical reference manual - RAPID Instructions, Functions and Data types*
Data type `robtarget`	*Technical reference manual - RAPID Instructions, Functions and Data types*

6.1.3 Search_Part - Search for feature presence

Usage

`Search_Part` is an instruction used for tactile searching 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.

Basis examples

```
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] [\Wire] bDetect StartPoint SearchPoint Speed
     Tool [\WObj] [\TLoad]
```

`[\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.

`[\Wire]`

Data type: `switch`

If selected, the output, `doWIRE_SEL`, will be set high when the SmarTac activation occurs. The SmarTac sensor will be switched from the gas cup to the wire when selected.

`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`

Continues on next page

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.

[\TLoad]

Data type: loaddata

The \TLoad argument describes the total load used in the movement. The total load is the tool load together with the payload that the tool is carrying. If the \TLoad argument is used, then the loaddata in the current tooldata is not considered.

If the \TLoad argument is set to load0, then the \TLoad argument is not considered and the loaddata in the current tooldata is used instead. For a complete description of the TLoad argument, see MoveL in *Technical reference manual - RAPID Instructions, Functions and Data types*.

Program execution

When executed, the robot makes a linear movement 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.

Error handling

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

RETRY	Tries to search again with start point moved 50%
DETECT	Continues the program with detection TRUE
REJECT	Continues the program with detection FALSE
RAISE	Sends error to calling routine.

Continues on next page

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 event 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 ',']
  ['\ ' Wire ',']
  [ 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 > ]
  [ '\' TLoad':=' ] < persistent (PERS) of loaddata > ] ';'
```

Related information

	Described in:
`Search_1D`	*Search_1D - One-dimensional search on page 79*
Data type `bool`	*Technical reference manual - RAPID Instructions, Functions and Data types*
`MoveL`	*Technical reference manual - RAPID Instructions, Functions and Data types*
Definition of `loaddata`	*Technical reference manual - RAPID Instructions, Functions and Data types*

6.1.4 PDispAdd - Add program displacements

Usage

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

Basic examples

```
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`	*Search_1D - One-dimensional search on page 79*
`PoseAdd`	*PoseAdd - Adds the translation portions of pose data on page 99*
Data type `pose`	*Technical reference manual - RAPID Instructions, Functions and Data types*

6.1.5 SwitchSmarTacSettings - Switch SmarTac signals and search speed

Usage

`SwitchSmarTacSettings` is an instruction used to switch the configuration of *SmarTac - Standard Signals* and *Smartac Speeds* to be used for searching with SmarTac. This can for example be used to change between searching with the wire or the cup.

Basic examples

The following example illustrates the instruction `SwitchSmarTacSettings`.

Example 1

```
SwitchSmarTacSettings "Wire_R1", "WireSpeed_R1";
```

The signals specified in the *SmarTac - Standard Signals* configuration `Wire_R1` and *Smartac Speeds* configuration `WireSpeed_R1` are activated.

Arguments

```
SwitchSmarTacSettings ( sSmarTacSignals sSmarTacSpeeds [\WaitInpos]
     )
```

`sSmarTacSignals`

Data type: `string`

This argument specifies the *SmarTac - Standard Signals* configuration instance that will be activated.

`sSmarTacSpeeds`

Data type: `string`

This argument specifies the *SmarTac Speeds* configuration instance that will be activated.

`\WaitInpos`

Data type: `num`

If this argument is used, RAPID execution will wait the specified number of seconds for robot and external axes to come to a standstill.

More examples

More examples of the instruction `SwitchSmarTacSettings` are illustrated below.

Example 1

We assume that the *SmarTac - Standard Signals* for searching with the wire is `Wire_R1` and for searching with the gas cup is `GasCup_R1`. We also assume that the *SmarTac Speeds* for searching with the wire is `WireSpeed_R1` and for searching with the gas cup is `GasCupSpeed_R1`.

Search with the wire.

```
SwitchSmarTacSettings "Wire_R1", "WireSpeed_R1";
```

Search with the gas cup and ensure that the robot is standing still, by using the optional argument `\WaitInpos`.

```
SwitchSmarTacSettings "GasCup_R1", "GasCupSpeed_R1"\WaitInpos:=2;
```

Continues on next page

Syntax

```
SwitchSmarTacSettings
  [sSmarTacSignals ':='] <expression (IN) of string>
  [sSmarTacSpeeds ':='] <expression (IN) of string>
  ['\' WaitInpos ':='] <expression (IN) of num>';'
```

6.2 Functions

6.2.1 PoseAdd - Adds the translation portions of pose data

Usage

`PoseAdd` is a function that requires two or three `pose` data arguments and returns the sum of the translation portions in `pose` form.

The returned `pose` data will have the quaternions set to [1,0,0,0].

Basic examples

```
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

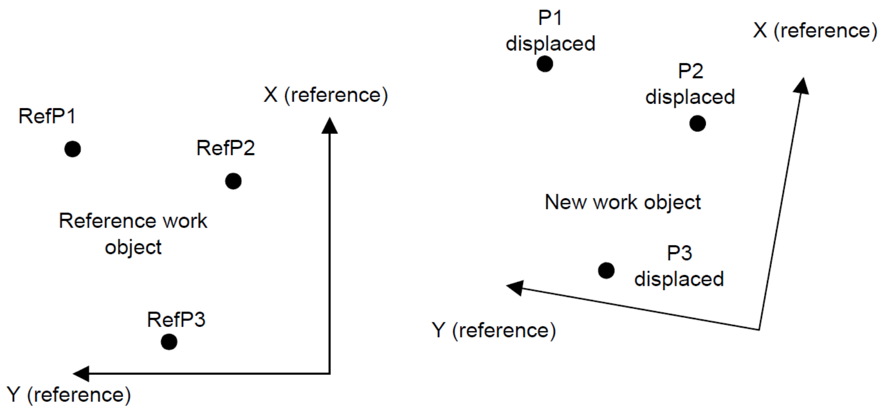
	Described in:
`PDispAdd`	*PDispAdd - Add program displacements on page 96*
Data type `pose`	*Technical reference manual - RAPID Instructions, Functions and Data types*

6.2.2 OFrameChange - Create a new shifted object frame

Usage

`OFrameChange` is a function that returns a work object based on a required reference work object, three reference points, and three corresponding displacements, described within the reference work object.

Basic examples



xx1400001523

```
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 reference 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`

Continues on next page

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

`DispP1`

Data type: `pose`

Displacement frame affecting reference point `RefP1`.

`DispP2`

Data type: `pose`

Displacement frame affecting reference point `RefP2`.

`DispP3`

Data type: `pose`

Displacement frame affecting reference point `RefP3`.

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`	*PDispAdd - Add program displacements on page 96*
`PoseAdd`	*PoseAdd - Adds the translation portions of pose data on page 99*
Data type `pose`	*Technical reference manual - RAPID Instructions, Functions and Data types*
Data type `wobjdata`	*Technical reference manual - RAPID Instructions, Functions and Data types*
Data type `robtarget`	*Technical reference manual - RAPID Instructions, Functions and Data types*

This page is intentionally left blank

Index

ABB AB
Robotics & Discrete Automation
S-721 68 VÄSTERÅS, Sweden
Telephone +46 10-732 50 00

ABB AS
Robotics & Discrete Automation
Nordlysvegen 7, N-4340 BRYNE, Norway
Box 265, N-4349 BRYNE, Norway
Telephone: +47 22 87 2000

ABB Engineering (Shanghai) Ltd.
Robotics & Discrete Automation
No. 4528 Kangxin Highway
PuDong New District
SHANGHAI 201319, China
Telephone: +86 21 6105 6666

ABB Inc.
Robotics & Discrete Automation
1250 Brown Road
Auburn Hills, MI 48326
USA
Telephone: +1 248 391 9000

abb.com/robotics

3HAC087219-001, Rev A, en