# NOTE interface to FANUC robot

* This note contains a LITTLE and NOT UPTODATE documentation related to the interface between the FANUC robot and an external control.

Date: 29/01/2014

Ole Madsen (om@m-tech.aau.dk)

Contents

[NOTE interface to FANUC robot 1](#_Toc378786867)

[1. Robot-server: FANUC 2](#_Toc378786868)

[2. Command (from extern computer to FANUC) 3](#_Toc378786869)

[2.1 Commands to the FANUC server 4](#_Toc378786870)

[3. Use 6](#_Toc378786871)

[3.1. Start-up: 6](#_Toc378786872)

[3.2. Test 6](#_Toc378786873)

[3.3. MATLAB 6](#_Toc378786874)

[Appendix A: The SERVER (In KAREL) 7](#_Toc378786875)

[APPENDIX B: RobotConnector.m (MAtlab) 16](#_Toc378786876)

[Appendix C: Translating and transferring a KAREL program 22](#_Toc378786877)

# Robot-server: FANUC

This note contains a LITTLE and NOT UPTODATE documentation related to the interface between the FANUC robot and an external control.

The FANUC robot acts as server in this control..

FANUC server

Ip = 192.168.1.25

port = 59002

Extern compurer

# Command (from extern computer to FANUC)

A number of messages have been defined between the FANUC robot and the extern computer.

The general syntax of a message is:

<CommandType>;<Data>;

where

**Commandtype** is a string characterizing the command send to the robot server (e.g MOVE, GETPOS ..)

**Data** are data related to the message (e.g positional data related to a move command. Note that in some cases there are not data

The robot gives the following replies to the commands:

<commandType>;started; (immediately after receiving the command)

<commandType>;ended; (after successful execution of the commend)

<commandType>;ERROR,<description of error>

In case of an unknown command the server will return:

ERROR;unknown command; or ERROR;wrong syntax;

## 2.1 Commands to the FANUC server

|  |  |
| --- | --- |
| **test;** | |
| **Data:** | None |
| **Description:** | Executes a test routine |
| **Example:** | test; |

|  |  |
| --- | --- |
| **GRABON;** | |
| **Data:** | None |
| **Description:** | Opens the gripper |
| **Example:** | GRABON; |

|  |  |
| --- | --- |
| **GRABOFF;** | |
| **Data:** | None |
| **Description:** | Closes the gripper |
| **Example:** | GRABOFF; |

|  |  |
| --- | --- |
| **MOVEJ;[<double>,<double>,<double>,<double>,<double>,<double>,<double>,<double>];** | |
| **data:** | X,Y,Z,RX,RY,RZ,Velocity (RX,RY,RZ = orientation of robot, using FANUC definitions)  Location of TOOL0 relatively to WORLD coordinates |
| **Description:** | Moves robot end-effector (TOOL0) to the desired location using J-move (PTP)) |
| **Example:** | MOVEJ;[0,0,100,90,0,0,100]; |

|  |  |
| --- | --- |
| **MOVEL;[<double>,<double>,<double>,<double>,<double>,<double>,<double>,<double>];** | |
| **data:** | X,Y,Z,RX,RY,RZ,Velocity (RX,RY,RZ = orientation of robot, using FANUC definitions)  Location of TOOL0 relatively to WORLD coordinates |
| **Description:** | Moves robot end-effector (TOOL0) to the desired location using L-move (Liniear)) |
| **Example:** | MOVEL;[0,0,100,90,0,0,100]; |

|  |  |
| --- | --- |
| **MOVELR;[<double>,<double>,<double>,<double>,<double>,<double>,<double>,<double>];** | |
| **data:** | dX,dY,dZ,dRX,dRY,dRZ,Velocity (RX,RY,RZ = orientation of robot, using FANUC definitions)  Location of TOOL0 relatively to WORLD coordinates |
| **Description:** | Relative linear move. The robot end-effector (TOOL0) is translated the dX,dY,dZ and rotation is changed: dRX,dRY,dRZ. |
| **Example:** | MOVEL;[0,0,100,90,0,0,100]; |

|  |  |
| --- | --- |
| **MOVEJR;[<double>,<double>,<double>,<double>,<double>,<double>,<double>,<double>];** | |
| **data:** | dX,dY,dZ,dRX,dRY,dRZ,Velocity (RX,RY,RZ = orientation of robot, using FANUC definitions)  Location of TOOL0 relatively to WORLD coordinates |
| **Description:** | Relative joint (PTP) move. The robot end-effector (TOOL0) is translated the dX,dY,dZ and rotation is changed: dRX,dRY,dRZ. |
| **Example:** | MOVEL;[0,0,100,90,0,0,100]; |

|  |  |
| --- | --- |
| **GETPOS;** | |
| **data:** | None |
| **Description:** | Returns the cartesian coordinates of the robot endeffctor in WORLD coordinates |
| **Eksempel:** | GETPOS; |

|  |  |
| --- | --- |
| **EXIT;** | |
| **data:** | None |
| **Description:** | Disconnecter serveren. |
| **Eksempel:** | EXIT |

# Use

## Start-up:

Start the program: ROBSERV on the FANUC robot.

See Appendix A for program. See Appendix C for a description on how to change the ROBSERV program.

## Test

Test the connection to the server using an HyperTerminal

## MATLAB

A matlab Class: RobotConnector has been made (Appendix B). Put this into the MatLab path

A simple MATLAB program could look as:

r = RobotConnector;

r.openGrapper

r.moveRelativeLinear(0,0,-100,0,0,0,100);

r.closeGrapper;

r.moveJoint(100,100,100,90,0,0,100)

## Appendix A: The SERVER (In KAREL)

PROGRAM ROBSERV

%ENVIRONMENT REGOPE

%comment= 'jan 2014 - ROBOT server: ip: 192.168.1.25; port = 59002'

%alphabetize --Specifies that variables will be created in alphabetical order when p-code is loaded.

----------------------------------------------------------------------------------

%RWACCESS

%STACKSIZE = 4000

--%NOLOCKGROUP

%NOPAUSE=ERROR+COMMAND+TPENABLE

%ENVIRONMENT uif

%ENVIRONMENT sysdef

%ENVIRONMENT memo

%ENVIRONMENT kclop

%ENVIRONMENT bynam

%ENVIRONMENT fdev

%ENVIRONMENT flbt

%ENVIRONMENT STRNG

%ENVIRONMENT MULTI

%ENVIRONMENT TPE

%INCLUDE klevccdf

%INCLUDE klevkeys

%INCLUDE klevkmsk

------------------------------------------------------------------------------

CONST

BUFFER\_SIZE=20

VAR

socket : FILE

entry,robotReady: integer

running : boolean

wrongData : boolean;

------------------------------------------------------------------------------

--------------------

-- INITIALIZATION --

--------------------

ROUTINE initialize

VAR

status : integer

tool : XYZWPR

user : XYZWPR

BEGIN

tool.x = 0

tool.y = 0

tool.z = 0

tool.w = 0

tool.p = 0

tool.r = 0

user.x = 0

user.y = 0

user.z = 0

user.w = 0

user.p = 0

user.r = 0

WRITE(CHR(128))

running = TRUE

SET\_VAR(entry, '\*SYSTEM\*','$HOSTS\_CFG[3].$SERVER\_PORT',59002,status)

$UTOOL = tool -- Defines tool frame

$UFRAME = user -- Sets user frame at current position

--TP\_CLS -- Forces user menu on TP to be cleared and shown

END initialize

----------------

-- NETWORKING --

----------------

ROUTINE waitClient

VAR

status : integer

BEGIN

WRITE('Ver: Jan 2014: Robot Server waiting for connection...',cr)

MSG\_CONNECT('S3:',status)

IF status = 0 THEN

OPEN FILE socket ('rw','S3:')

status = io\_status(socket)

IF status = 0 THEN

WRITE('Server Connected :-)',cr)

ENDIF

ENDIF

END waitClient

ROUTINE closeSocket

VAR

status : integer

BEGIN

CLOSE FILE socket

WRITE('Disconnecting...',cr)

MSG\_DISCO('S3:',status)

END closeSocket

----------------------

-- STRING CONVERTER --

----------------------

-- Reading a stringf of the structure <double>,<double>,<double>

ROUTINE getValues(message: string) : ARRAY OF REAL

VAR

values: ARRAY[7] OF REAL

i,index,length : integer

str : string[128]

realval : REAL

BEGIN

WRITE('Parsing string: ',message,cr)

index=1

str=''

length=STR\_LEN(message)

IF (SUB\_STR(message,1,1)<>'[') THEN

wrongData = TRUE

RETURN(values)

ENDIF

IF (SUB\_STR(message,length-1,2)<>'];') THEN

wrongData = TRUE

RETURN(values)

ENDIF

FOR i=2 TO length-1 DO

IF ((SUB\_STR(message,i,1)=',') OR (i=length-1)) THEN

CNV\_STR\_REAL(str,realval)

WRITE(str,cr)

WRITE(realval,cr)

values[index]=realval

index= index+1

str=''

ELSE

str=str+SUB\_STR(message,i,1)

ENDIF

ENDFOR

wrongData = FALSE;

RETURN(values)

END getValues

----------------------

-- COMMAND FEEDBACK --

----------------------

ROUTINE commandStart(commando:string)

BEGIN

WRITE socket(commando,';started;',cr)

END commandStart

ROUTINE commandEnd(commando:string)

BEGIN

WRITE socket(commando,';ended;',cr)

END commandEnd

ROUTINE cPosInv

BEGIN

WRITE ('Position invalid',cr)

WRITE socket('ERROR;Position invalid',cr)

END cPosInv

---------------------

-- CUSTOM COMMANDS --

---------------------

ROUTINE myGrabOFF

BEGIN

commandStart('GRABOFF')

WRITE('Grab OFF',cr)

RDO[1]=TRUE

RDO[2]=FALSE

commandEnd('GRABOFF');

END myGrabOFF

ROUTINE myGrabON

BEGIN

commandStart('GRABON')

WRITE('Grab ON',cr)

RDO[1]=FALSE

RDO[2]=TRUE

commandEnd('GRABON');

END myGrabON

-- MOVE JOINT IN WORLD FRAME --

ROUTINE myMoveJ(data:string)

VAR

XYZwprs : ARRAY[7] OF REAL -- Containing position, orientation and speed

vXYZ : XYZWPR

status : integer

BEGIN

commandStart('MOVEJ')

XYZwprs = getValues(data) -- Values parsed from main loop

vXYZ = CURPOS(0,0) -- Hack, to avoid configure something

IF wrongData THEN

write SOCKET('MOVE;ERROR, wrong data format');

ELSE

vXYZ.x = XYZwprs[1] -- x position

vXYZ.y = XYZwprs[2] -- y position

vXYZ.z = XYZwprs[3] -- z position

vXYZ.w = XYZwprs[4] -- x orientation

vXYZ.p = XYZwprs[5] -- y orientation

vXYZ.r = XYZwprs[6] -- z orientation

$SPEED = XYZwprs[7] -- velocity

$MOTYPE = JOINT

MOVE TO vXYZ

CHECK\_EPOS((vXYZ), $UFRAME, $UTOOL, status)

IF (status <> 0) THEN

cPosInv

ELSE

MOVE TO vXYZ

ENDIF

commandEnd('MOVEJ')

ENDIF

END myMoveJ

-- MOVE LINEAR IN WORLD FRAME --

ROUTINE myMoveL(data:string)

VAR

XYZwprs : ARRAY[7] OF REAL -- Containing position, orientation and speed

vXYZ : XYZWPR

status : integer

BEGIN

commandStart('MOVEL')

XYZwprs = getValues(data) -- Values parsed from main loop

vXYZ = CURPOS(0,0) -- Hack, to avoid configure something

IF wrongData THEN

write SOCKET('MOVE;ERROR, wrong data format');

ELSE

vXYZ.x = XYZwprs[1] -- x position

vXYZ.y = XYZwprs[2] -- y position

vXYZ.z = XYZwprs[3] -- z position

vXYZ.w = XYZwprs[4] -- x orientation

vXYZ.p = XYZwprs[5] -- y orientation

vXYZ.r = XYZwprs[6] -- z orientation

$SPEED = XYZwprs[7] -- velocity

$MOTYPE = LINEAR

CHECK\_EPOS((vXYZ), $UFRAME, $UTOOL, status)

IF (status <> 0) THEN

cPosInv

ELSE

MOVE TO vXYZ

ENDIF

commandEnd('MOVEL')

ENDIF

END myMoveL

-- MOVE JOINT RELATIVE --

ROUTINE myMoveJR(data:string)

VAR

XYZwprs : ARRAY[7] OF REAL -- Containing position, orientation and speed

vXYZ : XYZWPR

BEGIN

commandStart('MOVEJR')

XYZwprs = getValues(data) -- Values parsed from main loop

--vXYZ = CURPOS(0,0) -- Hack, to avoid configure something

IF wrongData THEN

write SOCKET('MOVE;ERROR, wrong data format');

ELSE

vXYZ.x = XYZwprs[1] -- x position

vXYZ.y = XYZwprs[2] -- y position

vXYZ.z = XYZwprs[3] -- z position

vXYZ.w = XYZwprs[4] -- x orientation

vXYZ.p = XYZwprs[5] -- y orientation

vXYZ.r = XYZwprs[6] -- z orientation

$SPEED = XYZwprs[7] -- velocity

$MOTYPE = JOINT

MOVE RELATIVE vXYZ

commandEnd('MOVEJR')

ENDIF

END myMoveJR

-- MOVE LINEAR RELATIVE --

ROUTINE myMoveLR(data:string)

VAR

XYZwprs : ARRAY[7] OF REAL -- Containing position, orientation and speed

vXYZ : XYZWPR

BEGIN

commandStart('MOVELR')

XYZwprs = getValues(data) -- Values parsed from main loop

--vXYZ = CURPOS(0,0) -- Hack, to avoid configure something

IF wrongData THEN

write SOCKET('MOVE;ERROR, wrong data format');

ELSE

vXYZ.x = XYZwprs[1] -- x position

vXYZ.y = XYZwprs[2] -- y position

vXYZ.z = XYZwprs[3] -- z position

vXYZ.w = XYZwprs[4] -- x orientation

vXYZ.p = XYZwprs[5] -- y orientation

vXYZ.r = XYZwprs[6] -- z orientation

$SPEED = XYZwprs[7] -- velocity

$MOTYPE = LINEAR

MOVE RELATIVE vXYZ

commandEnd('MOVELR')

ENDIF

END myMoveLR

ROUTINE myPosition

VAR

cPos : XYZWPR

BEGIN

cPos = CURPOS(0,0)

WRITE socket('[',cPos.x,cPos.y,cPos.z,cPos.w,cPos.p,cPos.r,']',cr)

END myPosition

ROUTINE myTest

VAR

XYZwprs : ARRAY[7] OF REAL -- Containing position, orientation and speed

vXYZ : VECTOR

BEGIN

commandStart('test')

vXYZ.x = 0 -- x position

vXYZ.y = 0 -- y position

vXYZ.z = 100 -- z position

$SPEED = 100 -- velocity

$MOTYPE = LINEAR

MOVE RELATIVE vXYZ

commandEnd('test')

END myTest

----------

-- ETC. --

----------

ROUTINE myExit

BEGIN

commandStart('Exiting')

WRITE('Exiting Program!',cr)

commandEnd('Exiting');

END myExit

------------------

-- MAIN ROUTINE --

------------------

ROUTINE mainLoop

VAR

message,commando,data : string[128]

status,i,J : integer

connectionOk,commandFound : boolean

str: string[5]

BEGIN

connectionOK = TRUE

WHILE connectionOk DO

READ socket(message)

status = io\_status(socket)

IF (status = 0) THEN

--Search for command

WRITE('Received string: ',message,cr)

commandFound = FALSE

i = STR\_LEN(message)

j = 1

WHILE ((NOT commandFound) AND (j<=i)) DO

IF(SUB\_STR(message,j,1)=';') THEN

commando = SUB\_STR(message,1,j-1)

data = SUB\_STR(message,j+1,i-j+1)

commandFound = TRUE;

ELSE

--WRITE(SUB\_STR(message,j,1))

j = j+ 1

ENDIF

ENDWHILE

IF (NOT commandFound) THEN

write SOCKET('ERROR;wrong syntax;')

ELSE

WRITE('Recieved command: ',commando,cr)

commandFound = FALSE

IF(commando='GRABON') THEN

myGrabON

commandFound = TRUE

ENDIF

IF(commando='test') THEN

myTest

commandFound = TRUE

ENDIF

IF(commando='GRABOFF') THEN

myGrabOFF

commandFound = TRUE

ENDIF

IF(commando='MOVEJ') THEN

myMoveJ(data)

commandFound = TRUE

ENDIF

IF(commando='MOVEL') THEN

myMoveL(data)

commandFound = TRUE

ENDIF

IF(commando='MOVELR') THEN

myMoveLR(data)

commandFound = TRUE

ENDIF

IF(commando='MOVEJR') THEN

myMoveJR(data)

commandFound = TRUE

ENDIF

IF(commando='GETPOS') THEN

myPosition

commandFound = TRUE

ENDIF

IF(commando='EXIT') THEN

connectionOK=FALSE

running = FALSE;

commandFound = TRUE

myExit

ENDIF

IF (NOT commandFound) THEN

write SOCKET('ERROR;unknown command;')

ENDIF

DELAY 10

ENDIF

ELSE

connectionOk=FALSE

ENDIF

ENDWHILE

END mainLoop

----------

-- MAIN --

----------

BEGIN

initialize

WHILE running DO

waitClient

mainLoop

closeSocket

ENDWHILE

END ROBSERV

## APPENDIX B: RobotConnector.m (MAtlab)

classdef RobotConnector

%UNTITLED Summary of this class goes here

% Detailed explanation goes here

properties

tcpipSocket;

hostAddress = '192.168.1.25';

hostPort = 59002;

debug = false;

end

methods

% Constructor

function obj = RobotConnector(hostAddress\_,hostPort\_)

% See if connector is started with an argument

if nargin > 0

obj.hostAddress = hostAddress\_;

obj.hostPort = hostPort\_;

end

% Connect to the server

obj.tcpipSocket = tcpip(obj.hostAddress,obj.hostPort);

try

fopen(obj.tcpipSocket);

catch err

disp(['COULD NOT CONNECT TO SERVER: ' err.message]);

%rethrow(err);

end

end

% Destructor

function delete(obj)

fclose(obj.tcpipSocket);

delete(obj.tcpipSocket);

end

%\*\*\*\*\*\*\*\* Move functions

% Move Joint

function moveJoint(obj,varargin) % moveJoint(x,y,z,w,p,r,speed) or moveJoint(A,speed) where A = [ x y z w p r ];

if nargin == 3

A = varargin{1};

if (length(A) == 6 )

x = A(1); y = A(2); z = A(3); w = A(4); p = A(5); r = A(6);

else

disp(['Position matrix has the wrong dimension, expected 6, but got: ' num2str(length(A))]);

return;

end

speed=varargin{2};

elseif nargin == 8

x = varargin{1}; y = varargin{2}; z = varargin{3}; w = varargin{4}; p = varargin{5}; r = varargin{6};

speed = varargin{7};

else

disp(['Incorrect number of argumets, expected 2 or 7, but found: ' num2str(nargin) ]);

return;

end

str = [ 'MOVEJ;[' num2str(x) ',' num2str(y) ',' num2str(z) ',' num2str(w) ',' num2str(p) ',' num2str(r) ',' num2str(speed) '];' 10 ];

robotSend(obj,str);

robotWait(obj,['MOVEJ;started;' 10]); % Wait for commands

robotWait(obj,['MOVEJ;ended;' 10]);

end

% Move Linear

function moveLinear(obj,varargin) % moveLinear(x,y,z,w,p,r,speed) or moveLinear(A,speed) where A = [ x y z w p r ];

if nargin == 3

A = varargin{1};

if (length(A) == 6 )

x = A(1); y = A(2); z = A(3); w = A(4); p = A(5); r = A(6);

else

disp(['Position matrix has the wrong dimension, expected 6, but got: ' num2str(length(A))]);

return;

end

speed=varargin{2};

elseif nargin == 8

x = varargin{1}; y = varargin{2}; z = varargin{3}; w = varargin{4}; p = varargin{5}; r = varargin{6};

speed = varargin{7};

else

disp(['Incorrect number of argumets, expected 2 or 7, but found: ' num2str(nargin) ]);

return;

end

str = [ 'MOVEL;[' num2str(x) ',' num2str(y) ',' num2str(z) ',' num2str(w) ',' num2str(p) ',' num2str(r) ',' num2str(speed) '];' 10 ];

robotSend(obj,str);

robotWait(obj,['MOVEL;started;' 10]); % Wait for commands

robotWait(obj,['MOVEL;ended;' 10]);

end

% Move Relative Joint

function moveRelativeJoint(obj,varargin) % moveRelativeJoint(x,y,z,w,p,r,speed) or moveRelativeJoint(A,speed) where A = [ x y z w p r ];

if nargin == 3

A = varargin{1};

if (length(A) == 6 )

x = A(1); y = A(2); z = A(3); w = A(4); p = A(5); r = A(6);

else

disp(['Position matrix has the wrong dimension, expected 6, but got: ' num2str(length(A))]);

return;

end

speed=varargin{2};

elseif nargin == 8

x = varargin{1}; y = varargin{2}; z = varargin{3}; w = varargin{4}; p = varargin{5}; r = varargin{6};

speed = varargin{7};

else

disp(['Incorrect number of argumets, expected 2 or 7, but found: ' num2str(nargin) ]);

return;

end

str = [ 'MOVEJR;[' num2str(x) ',' num2str(y) ',' num2str(z) ',' num2str(w) ',' num2str(p) ',' num2str(r) ',' num2str(speed) '];' 10 ];

robotSend(obj,str);

robotWait(obj,['MOVEJR;started;' 10]); % Wait for commands

robotWait(obj,['MOVEJR;ended;' 10]);

end

% Move Relative Linear

function moveRelativeLinear(obj,varargin) % moveRelaviteLinear(x,y,z,w,p,r,speed) or moveRelativeLinear(A,speed) where A = [ x y z w p r ];

if nargin == 3

A = varargin{1};

if (length(A) == 6 )

x = A(1); y = A(2); z = A(3); w = A(4); p = A(5); r = A(6);

else

disp(['Position matrix has the wrong dimension, expected 6, but got: ' num2str(length(A))]);

return;

end

speed=varargin{2};

elseif nargin == 8

x = varargin{1}; y = varargin{2}; z = varargin{3}; w = varargin{4}; p = varargin{5}; r = varargin{6};

speed = varargin{7};

else

disp(['Incorrect number of argumets, expected 2 or 7, but found: ' num2str(nargin) ]);

return;

end

str = [ 'MOVELR;[' num2str(x) ',' num2str(y) ',' num2str(z) ',' num2str(w) ',' num2str(p) ',' num2str(r) ',' num2str(speed) '];' 10 ];

robotSend(obj,str);

robotWait(obj,['MOVELR;started;' 10]); % Wait for commands

robotWait(obj,['MOVELR;ended;' 10]);

end

% Open and close grabber

function openGrapper(obj)

str = [ 'GRABOFF;' 10 ];

robotSend(obj,str);

%robotWait(obj,['GRABON;started;' 10]); % Wait for commands

%robotWait(obj,['GRABON;ended;' 10]);

end

function closeGrapper(obj)

% str = [ 'MOVEL;[-3 12 -41 9 -36 0 20];'];

str = [ 'GRABON;' 10 ];

robotSend(obj,str);

%robotWait(obj,['GRABOFF;started;' 10]); % Wait for commands

%robotWait(obj,['GRABOFF;ended;' 10]);

end

%\*\*\*\*\*\*\*\* Get position function

function A = getPosition(obj) % Retuenvalue A = [x y z w p r]

str = [ 'GETPOS;' 10 ];

robotSend(obj,str);

% Wait for the data to arraivem and extract the returned position

while (obj.tcpipSocket.BytesAvailable == 0)

end

% Read all the data available

data = fread(obj.tcpipSocket,obj.tcpipSocket.BytesAvailable);

data = char(data');

if obj.debug == true

disp(['Data Received: ' data ]);

end

% Get csv data inside inside square paranthesis

data = regexpi(data,'[[]]','split');

data = data{2};

% Extract the values from the csv data

vals = regexpi(data,' ','split');

mask = zeros(length(vals),1);

for i = 1:length(mask)

if length(vals{i}) > 0

mask(i) = 1;

end

end

mask = boolean(mask);

vals = vals(mask);

% Test if the correct number of cells has been received

if ( length(vals) ~= 6 )

disp(['Too few values in string, expected 6, but got: ' num2str(length(vals)) ]);

return;

end

% Form the resulting matrix

A = [str2num(vals{1}) str2num(vals{2}) str2num(vals{3}) str2num(vals{4}) str2num(vals{5}) str2num(vals{6})];

end

end

end

function robotSend(obj,str)

% Clear the incoming buffer

robotClearBuf(obj);

if obj.debug == true

disp(['Data sent: ' str(1:end-1)]);

end

fwrite(obj.tcpipSocket,str);

end

function robotWait(obj,str)

% Wait for data to be ready

while (obj.tcpipSocket.BytesAvailable == 0)

end

% Read all the data available

data = fread(obj.tcpipSocket,obj.tcpipSocket.BytesAvailable);

data = char(data');

if obj.debug == true

disp(['Data Received: ' data ]);

end

if strcmp(data,str) == 0

disp('DATA NOT AS EXPECTED!');

end

end

function robotClearBuf(obj)

if obj.tcpipSocket.BytesAvailable > 0

% Read all the data available

data = fread(obj.tcpipSocket,obj.tcpipSocket.BytesAvailable);

data = char(data');

if obj.debug == true

disp(['Data Received (clear buf): ' data ]);

end

end

end

## Appendix C: Translating and transferring a KAREL program

|  |  |
| --- | --- |
| 1. Start WinOLPC |  |
| 1. Select: Robot->Generic->V6.31-1 |  |
| 1. Press: KAREL translater |  |
| 1. Select kl file |  |
| 1. Successful ?? Look at screen |  |
| 1. Select: FANUC Robotics File Service->Select Robot -> FANUC |  |
| 1. In filters on right hand side select: ALL 2. Find libery with KAREL file and copy <name>.pc to the robot |  |