University of Stavanger

Faculty of Science and Technology

Stavanger, September 9, 2021

ELE610 Robot Technology, autumn 2021

ABB robot assignment 4

In this assignment you will run the program on one of the real ABB robots, both Norbert or Rudolf can be used. You should establish communication between MATLAB and RobotWare and from MATLAB tell the robot what we want it to do.

Approval of the assignment can be achieved by demonstrating the robot program to the teacher, and then submit a report including RAPID code on *canvas*. Only the RAPID code needs to be submitted, note that this should be a txt-file (not the mod-file that Windows will interpret as a video file). Possibly, you may submit a pdf-file and perhaps also include comments or questions. Make sure that the relevant RAPID code is clearly marked, for example by using Courier font. Nothing but RAPID code needs to be included in the report.

This assignment is demanding if you insist on making the solution perfect, especially the last part may be difficult. You should do as much as you can within the time limit of twenty hours.

4 Control robot from Matlab

You may start from the same pack-and-go file used earlier, it contains a station similar to the actual laboratory in E458. If not already done, download Pack-and-go-file, `UiS_E458_nov18.rspag`, and make sure that file extension is `rspag`.

4.0 Introduction

In this assignment there are several files you need, and some you don't actually need but that can still be useful and give some hint about how things can be done. Or maybe just increase the confusion? Note that many of these files,

Karl Skretting, Department of Electrical Engineering and Computer Science (IDE), University of Stavanger (UiS), N-4036 Stavanger. E-mail: karl.skretting@uis.no.

perhaps in other (older) versions, are available from different locations on local PCs in E458 as well.

List of files:

- Pack-and-go file that contains a station as before
  Pack-and-go-file, `UiS_E458_nov18.rspag`.
- `TriplePen.rslib`, a more advanced pen tool. You may also use the simple pen, or even solve the task without any particular tool attached to the robot, simply use the tool already mounted.
- `TriplePen.mod`, RAPID code for the triple pen tool.
- `abbCom.zip`, Collection of MATLAB functions for robot (PCSDK) communication. Note that PCSDK only works for the 32 bits MATLAB version.
- `MatlabCom.mod`, RAPID code that can be used to receive (and respond to) messages sent from MATLAB. Your main task in this assignment is to make the MATLAB code that sends these messages.
- `lab4_path.m`, a simple MATLAB m-file that set points on a path for the robot to follow.

The purpose of section 4.1 is to understand the `abb*.m` files, why they are made, what each one does, and how they are used. You should also assert that they actually works as intended, and that you know how to use them. The purpose of section 4.2 is much the same, but now you should be able to do the communication *while* a RAPID program is running. The RAPID program for this exercise should be as simple as possible, the `MatlabCom.mod` module is not needed. You don't need a MATLAB program, but can execute the needed commands (functions) directly from MATLAB command prompt.

The purpose of section 4.3 is to understand the `MatlabCom.mod` module, understand the different parts here, especially the different variables and the main loop, but also the other functions. Hopefully, you will understand why a module like this can be helpful for controlling (parts of) the robot program (robot path) from an external program. Here, the external program the MATLAB m-file is important to solve the task. Actually, when the `MatlabCom.mod` module is understood and used correctly, all the logic (programming) for solving the task should be done from the MATLAB m-file.

In section 4.4 the task is made a little bit more complicated by transmitting a short path describing what to draw, i.e. how the robot should move. You should learn how to write elements into a RAPID array and how the `MatlabCom.mod` module can be used to move the robot along an (infinite) long

path. The given problem (to draw a "flower") can be easily(?) solved by a sufficient large array in RAPID, but the real challenge here is to alternately use two (six) RAPID arrays, each of limited length (ex. 50), to transfer, and make to robot move along, an (infinite) long path. Note that the `MatlabCom.mod` module is made ready to do this.

4.1 Communication basics

PC SDK (Software Development Kit) is a ABB tool used to adapt robot solutions for customer demands, information and documentation available at ABB PC SDK web page. The idea for PCSDK is quite simple. PCSDK has full access to the robot controller, including the variables defined in the RAPID code. PCSDK can both read from and write to these, even when the robot controller is running the program. This way it is possible to crash the program. Actually, if communication is not well planned and designed or not correctly implemented, a crash is the very likely outcome.

I have made some MATLAB-files intended to simplify communication from MATLAB to ABB robots via PCSDK (.NET DLL). Files are packed into `abbCom.zip`, and should work quite well, but they are not extensively tested yet.

```
abbCom          Make a struct to use for communication with ABB robot
abbEdit         Edit ABB RAPID variable in struct, no communication to robot.
abbMain         Script to test abb*.m functions and RAPID MatlabCom module
abbMaster       Asks for master access to a robot
abbNew          Make a struct with fields as an ABB RAPID variable
abbRead         Read from ABB RAPID variable into a struct
abbString       Make a string of ABB RAPID struct (like .StringValue)
abbText         Make a text from ABB RAPID struct
abbWrite        Write to ABB RAPID variable from a struct
abbWriteArray   Write to ABB RAPID array nMatlabComArray* in MatlabCom
```

These files should be installed on `..\Documents\MATLAB\common` catalog on the two PCs (closest to the windows) in E458, Molaugståvå. Note that earlier versions of these files may exist in other catalogs on these computers, to check which version is active you may type `help abbCom` or another function, if help text is in Norwegian it is surely an older version (which probably don't work now). Include module `MatlabCom.mod` into the RAPID program and try (the first part of) `abbMain` to test communication. Change one of the RAPID variables from MATLAB, write it to the robot, then use the FlexPendant to check that the variable actually is changed.

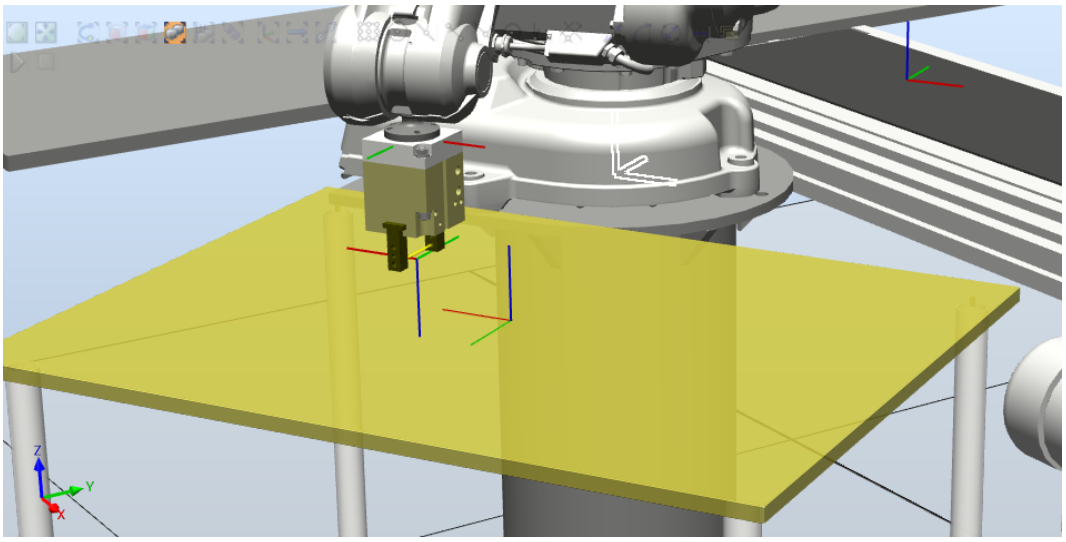

Figure 1: The table beside Norbert as used in exercise 4.2. Note that the work object 'wobjTable' is shown in the middle of the table, also axis for the tool 'tGripper' is shown. For the tool to be aligned with a target point that uses 'wobjTable' we see that the target point should be rotated 180 degrees around the x-axis.

4.2 Change direction

You should now define a rectangular path a safe height above the table beside one of the robots by using an array of 8 points. Write a program that makes the robot move along this path either clockwise or counterclockwise depending on the value of a boolean variable 'clockWise'. The instructions should be within an infinite loop, so that the robot continue to move until it is stopped, or when the safety button on the FlexPendant is released.

When this simple program works as intended you are ready to move to the actual robot. Do this an check that the program works here too. Now, you can start MATLAB, 32 bits version, on a PC connected to the robot, the same local network as the actual robot controller. Use the functions in 'abbCom.zip' to read some variables in the robot controller. You can also set the value of RAPID variable 'clockWise' from MATLAB using `abbMaster(..)` and `abbWrite(..)`. Check that this is working and try to understand what happens. Run the program on the robot, then while the program is running you can again set the value of RAPID variable 'clockWise' from MATLAB. You should now be able to turn the direction of the robot movements while the robot is running from another program, i.e. MATLAB.

4.3 Jog robot from Matlab

In this part you include the module MatlabCom.mod in your program. Let us start by looking into 'MatlabCom.mod' and see what it does, below we only display the most important lines here

```
MODULE MatlabCom
    ! Module MatlabCom: Karl S, March 2014 --> March 2015 --> Jan 2016  --> Feb 2019

    PERS wobjdata wobjMatlabCom;
    PERS tooldata tMatlabCom;
    VAR speeddata vMatlabCom := v100;
    VAR zonedata zMatlabCom := z1;

    ! the most important variables
    VAR num nMatlabComWRD := 0;   ! What Robot Does
    VAR num nMatlabComWMW := 0;   ! What Matlab Wants robot to do next

    ! more variables
    VAR robtarget pMatlabCom0 := [[0,0,0],[0,1,0,0],[0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]];
    VAR robtarget pMatlabCom1 := [[0,0,0],[0,1,0,0],[0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]];
    VAR robtarget pMatlabCom2 := [[0,0,0],[0,1,0,0],[0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]];
    ! current (or last) position
    VAR robtarget pMatlabComCur := [[0,0,0],[0,1,0,0],[0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]];
    VAR num nMatlabComNum1 := -1;

    PROC MatlabComInit()
        wobjMatlabCom := wobj0;
        tMatlabCom := tool0;
    ENDPROC

    PROC MatlabComMain()
        TPWrite "MatlabComMain starts.";
        WHILE TRUE DO
            nMatlabComWRD := 0;              ! robot waits
            IF (nMatlabComWMW = 0) THEN
                TPWrite "Robot waits until Matlab set WMW.";
            ENDIF
            WaitUntil (nMatlabComWMW <> 0);  ! Matlab wants robot to do something
            nMatlabComWRD := nMatlabComWMW;  ! and robot does this
            nMatlabComWMW := 0;              ! and is ready for receiving next wish
            TEST nMatlabComWRD
            CASE -1 : TPWrite "MatlabComMain quit."; RETURN;
            CASE  0 : TPWrite "MatlabComWRD = 0 is strange here."; WaitTime 0.1;
```

```
39              CASE  1 : MatlabCom01;     ! path 1
40              CASE  2 : MatlabCom02;     ! path 2
41              CASE  3 : MatlabCom03;     ! moves to jMatlabCom
42              CASE  4 : MatlabCom04;     ! changes speeddata
    vMatlabCom
43              CASE  5 : MatlabCom05;     ! changes zonedata
    zMatlabCom
44              CASE  6 : MatlabCom06;     ! moves to pMatlabCom0
45              CASE  7 : MatlabCom07;     ! moves to pMatlabCom1
46              CASE  8 : MatlabCom08;     ! moves to pMatlabCom2
47              CASE  9 : MatlabCom09;     ! store current (position
     and) angles
48              CASE 10 : MatlabCom10;     ! changes (TriplePen)
    tool
49              ENDTEST
50          ENDWHILE
51      ENDPROC
52
53      PROC MatlabCom06()
54          MoveL pMatlabCom0, vMatlabCom, zMatlabCom, tMatlabCom \
    WObj:=wobjMatlabCom;
55          pMatlabComCur := pMatlabCom0;
56      ENDPROC
```

The intention with this code from `MatlabCom.mod` is to make it possible to change variables in RAPID from an external program MATLAB without interfering with what robot does and avoid to change variables that are used by RAPID. This way communication can be done while the robot is moving. Below the one possible flow of actions is described.

In RAPID the function 'main()', or another function, calls 'MatlabComInit()' then calls 'MatlabComMain()' and enters an (almost) infinite loop. The 'MatlabComInit()' function should be changed to set the correct work object and the correct tool, and possible some other initiation as wanted by this program. As the program execution starts the main loop of 'MatlabComMain()' it comes to the `WaitUntil (nMatlabComWMW <> 0);` statement where it waits until the variable 'nMatlabComWMW' is changed from outside, typically from MATLAB. Before MATLAB changes this it may have set new values to the point 'pMatlabCom0' and when 'nMatlabComWMW' is changed to 6 the robot continue its execution by starting to do what MATLAB wants. After setting 'nMatlabComWRD' to 6 and 'nMatlabComWMW' to 0, the robot starts function 'MatlabCom06'.

Since 'nMatlabComWMW' is 0, MATLAB is allowed to give more wishes. The external program can at any time read, but never write, the 'nMatlabComWRD' variable and thus know what function RAPID is running, or if RAPID wait (`nMatlabComWRD = 0`). As long as it is 6 the RAPID variable 'pMatlabCom0', among others, is used and should not be changed, and as it is not 7 the RAPID variable 'pMatlabCom1' can be changed. From MATLAB the next value to move to is written into 'pMatlabCom1' and the 7 is written

into ‘nMatlabComWMW’. When function ‘MatlabCom06’ is finished RAPID will not wait but continue to do function ‘MatlabCom07’ after setting ‘nMatlabComWRD’ to 7 and
‘nMatlabComWMW’ to 0. This way MATLAB can, point by point, tell the robot where to move.

Here we denote the module containing function ‘main()’ as ‘main_4_3.mod’, in this case this module may be quite simple, even more simple than ‘main_4_2.mod’. Also the module ‘gripper.mod’ should be included in program.

```
MODULE main_4_3N
    ! Module for ELE610 exercise 4.3, Karl Skretting February 2019
    ! Use this module and MatlabCom.mod for exercise 4.3

    ! wobjTableX (middle of table) is defined relative to wobj0 (robot base)
    ! [1,0,0,0] : for wbjTableR means it is not rotated (relative to wobj0)
    ! PERS wobjdata wobjTableR := [FALSE,TRUE,"",[[150,500,8],[1,0,0,0]],[[0,0,0],[1,0,0,0]]]; ! Rudolf
    ! [0,0,0,1] : for wbjTableN means it is rotated 180 degrees around z-axis (relative to wobj0)
    PERS wobjdata wobjTableN:=[FALSE,TRUE,"",[[150,-500,8],[0,0,0,1]],[[0,0,0],[1,0,0,0]]];  ! Norbert
    VAR robtarget Target_10:=[[0,0,170],[0,1,0,0],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

    PROC main()
        MoveL Target_10,v100,z1,tGripper\WObj:=wobjTableN;
        MatlabComInit;
        MatlabComMain;
    ENDPROC
ENDMODULE
```

The task for you here is to write the MATLAB program that makes it easy for you to control, here jog, the robot from MATLAB. You should be able to move the robot along each axis a given distance, `stepLength`, and you should be able to change the step length. Perhaps you also can change robot speed and zone data. This, of course, can be designed in many ways. This work can be done anywhere, you only need to be at the robot laboratory to test it and perhaps make the final adjustments. A simple interface can be made by the functions `menu` and `input` as below, or even simpler by using only `input` to tell what to do.

```
    while true
        choice = menu('Select what to do : ',...
                    'x++', ...      % 1
                    'x--', ...      % 2
                    'y++', ...      % 3
                    'y--', ...      % 4
                    'z++', ...      % 5
                    'z--', ...      % 6
                    'Change step length', ...    % 7
                    'Quit' );           % 8
        % ... and then do it
        if choice == 1
            % ...
        end
        % ...
        if choice == 7
            stepLength = input(' Give step length [mm] : ');
        end
        if choice == 8
            break;
        end
    end
```

4.4 Give path to follow from Matlab

Now you should improve on the rather awkward way of sending a path "point-by-point" that was used in the previous section. Here, many points on the path should be transmitted at once, and while the robot follows this section of the path, the next section of the path can be transmitted, and the robot can, without a break in its movements, continue to follow the next section of the path. This way a long path can be followed. In 'MatlabCom.mod' it is prepared to do this in a quite simple way: The path to follow is just given by using the RAPID `Offs(..)` function relative to the point 'pMatlabCom0' where the offset values for `dx`, `dy` and `dz` are just given as values in three arrays (of num). Since we want to write into one set of arrays while one set of arrays is used for the robot motion, a total of 6 arrays are needed, these are defined in 'MatlabCom.mod'.

The main job in this exercise is to write the MATLAB function that controls the robot. The path the robot should follow can be the flower from `lab4_path.m`. The program on the robot can be the same as in previous exercise, 'main_4_3.mod', but it may be a good idea to try to make the program more robust, and informative, useful RAPID functions may be `ConfJ`, `ConfL`, `TPWrite`, `CPos`, `CRobT`, `CJointT`, and `CalcJointT`. Some changes can also be done in the 'MatlabCom.mod', like setting 'pMatlabCom0' in 'MatlabComInit()'.

To test and run this on the actual robot, either Norbert or Rudolf may be

used. The center point 'pMatlabCom0' should be in a safe distance above the table center. It may be an advantage to run the robot in automatic mode, this way you don't need to press the safety button on the FlexPendant all the time. Finally, for Rudolf and a working pen mounted, you should update the program such that it actually draw the flower on a sheet placed on the table. To make the flower fit onto an A4 sheet it should be scaled down by a factor 0.7 or smaller.

You may use the simple pen tool, or even better (more demanding) the TriplePen tool. The TriplePen tool has been out of ink for some time, and if that is not fixed yet the solution for this pen should as earlier just "draw" the flower by moving the tool, with the selected pen pointing straight downwards and a little bit (10 mm) above the sheet.