Stavanger, July 5, 2024

University of Stavanger

Faculty of Science and Technology

ELE610 Applied Robot Technology, H-2024

ABB robot assignment 4

This assignment was made and tested by Runar W. Skaget and Jacob A.M Reiersøl in their project for ELE630 spring 2023. Also, Ali H. Jabbour and Lars-Erik Nes Panengstuen worked on this problem autumn 2022 in ELE630, they mainly developed the software for the TATEM (Tool for Automatic Testing of Events and Motion) tool developed by ABB, to make it work with the ABB-robot Rudolf at UiS. After this assignment was first given, autumn 2023, a major revision was needed and this is included here.

In this assignment you will run a RAPID program to simulate a welding process using one of the ABB IRB140 robots. Rudolf is the preferred robot to use. The TATEM tool records data during the simulation process, and the data can be transferred to a PC using Python and Bluetooth Low Energy communication. Data is then stored on the PC as a JSON file, this file can be processed, visualized and interpreted to decide whether the simulation was successful or not.

In this assignment you may start by reading part 4.1 and check that the files in part 4.2 works on your computer. Then do part 4.3 in RobotStudio and part 4.4 on Rudolf in robot laboratory.

Approval of this assignment can be achieved by demonstrating the operation to the teacher, and then submitting a brief report as a pdf-file on *canvas*. The table (4.3.1.b), setup and running time (4.3.1.c), times from subsections 4.4.1 to 4.4.4, and perhaps some other results and parts of RAPID code should be included in the report.

The intention here is that you should make a solution as good as possible within the time limit for the assignment, which is twenty hours. If all tasks are not done a table showing time used, for each student of the group, should also be included in the report.

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

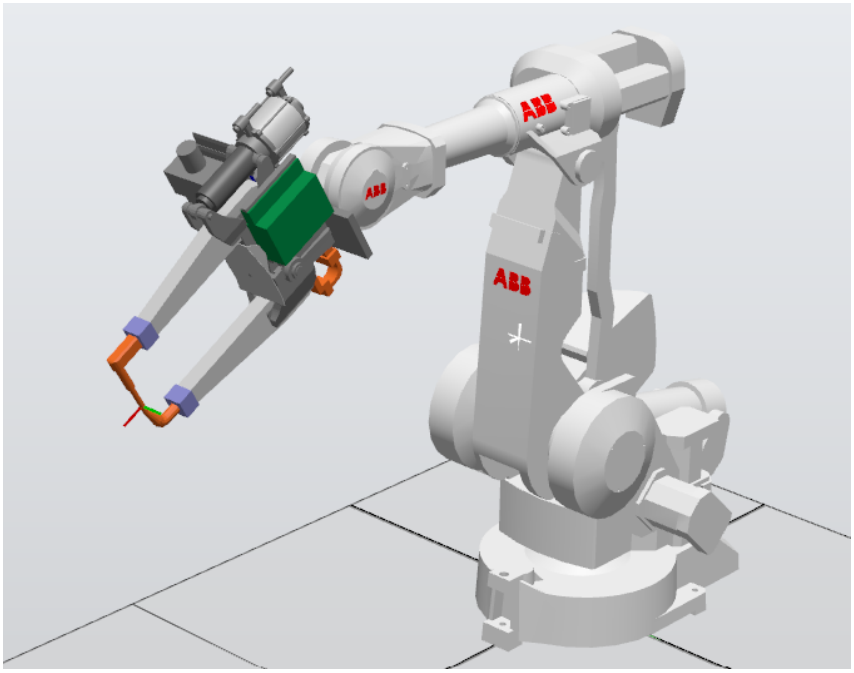

Figure 1: Robot equipped with welding tool. The electrode is the orange colored part of the tool.

4 Spot weld simulation

4.1 Actual spot welding

Welding is a common method in manufacturing used to join materials, usually metals, together. Spot welding is a method for welding where the tool is equipped with two electrodes on opposing sides of a clamp. Spot welding has applications in a number of industries, including automotive, aerospace, rail, white goods, metal furniture, electronics, medical building and construction. An example video of a manufacturing process for producing car components including spot welding can be seen on YouTube ↗. Figure 1 provides a visualization of an IRB4400 with a generic welding tool attached.

When welding, the clamp will first close over the two connection points, typically on each side of the two metal sheets to be joined and pressing them tightly together. Then, a large current will run through the electrodes for a short time (10-600 ms), heating mainly the metal between the connection points and joining the surfaces together. The tool must remain completely still throughout the process, then release the clamp, and move swiftly to the next point. The electrodes are non-consumable so no additional material is added to the metal.

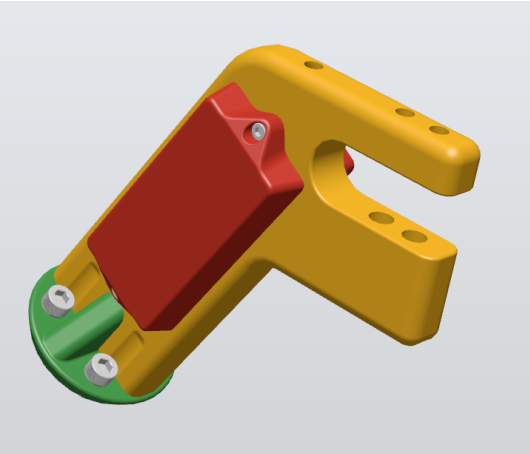

Figure 2: The TATEM tool used for spot welding simulation. The wide finger has two LEDs and the thin finger has two photo diode sensors built inside themselves. They are mounted opposite of each other so that when the tool is activated, both LEDs light and both photo diode sensors are activated unless something obstructs the gap. The outer pair is denoted as `L1` and `S1` and the inner pair as `L2` and `S2` in the ABB documentation. The tool coordinate system has origin in the middle of the gap between `L1` and `S1`, and the z-axis is pointing forward, out from the tool.

4.2 Python preliminaries

In this assignment you will use Python, version 3.8 or higher, to transfer data from the TATEM tool to a PC and to view the downloaded reports. Several Python files are needed, they should all be included in `ELE610py3files.zip` ↗ and after unpacking be in your `..\ELE610\py3\` catalog. The files needed here are:

File or catalog	Comment
`datasets`	catalog with example file(s)
`AppTATEM`	catalog with some additional file(s)
`appDashTest.py`	test of dash web-server
`appTATEM.py`	the dash web-server application
`showJSON.py`	show summary of TATEM report
`TatemArduinoIf.py`	interface to Arduino in TATEM tool
`TatemRapidIf.py`	interface to ABB robot
`tatemCom.py`	the cmd2 communication program

In addition several Python packages are needed, see `littPy3x.pdf` ↗. Some details of how to use these programs are in section 4.5.5, section 4.5.6 and section 4.5.7.

`TatemRapidIf.py` is used only to transfer values, typically times, from RAPID program to TATEM tool from program `appTATEM.py`.

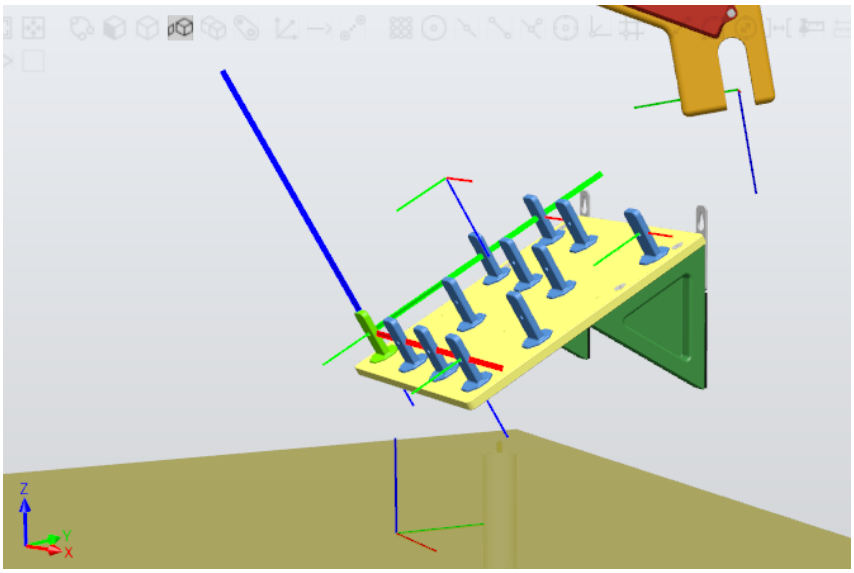

Figure 3: The test board for simulation in RobotStudio. The test board is here hovering above the table, this makes it the same height above the table as when the actual test board is placed on the table beside Rudolf in UiS laboratory. The work object `wobjTestBoard` is here clearly indicated and the peg in origin is shown in green.

4.3 Simulated spot welding, RobotStudio

A model of the TATEM tool is made and can be used in RobotStudio to prepare for spot welding simulation. To begin you will need to load the Pack and Go file, `UiS_RS4_16nov2023.rspag` ↗ into RobotStudio. Examine this example carefully, you may read the explanation of some used instructions in section 4.5.1 and of the RAPID code functions in section 4.5.2. The many time constants used may also be difficult to fully understand, see some explanation in section 4.5.3 and section 4.5.4. Run some simulations in RobotStudion, and finally do some changes to RAPID code and make sure that you understand what happens.

In particular you should be aware of how the signal that turns on and off the LEDs on the TATEM tool is used, even though this does not work in simulation. It needs the actual robot Rudolf with the TATEM tool connected to work. The signal is confusingly denoted as `AirValve`. A *trigger* can be used to turn this signal on, and an *interrupt* is used to turn this signal off.

4.3.1 RobotStudio simulation

a. First, do simulation of the welding path given by the RAPID code included in the pack-and-go file. Make sure you understand what this simulation does and that it runs as intended on your computer. You will not be able to retrieve data about success or failure in welding yet, you only have visual control of whether the tool enclose the peg or not and if this last for the wanted amount of time. Run both `testPeg00()` and `test4corners()` and check that they work as intended. You can try several combinations of `doSlow`, `useFlexPendant` and `useBoardOnTable`, but at least use all these three as `TRUE`.

b. Secondly, change the `testPeg00()` function such that it use the FlexPendant to ask for a peg to test, a number 1 to 16 for (dx,dy) $\in$ {(0,0), (0,60), (0,120), (0,180), (1,0), .., (180,180)}. It may be an idea to put this in a loop, quit loop if illegal number is given, to allow for another target point after the current one is done. Check that this works, then add another option that asks for rotation in steps of 45 °between -180 °and 180 °. You should note that not all these combinations are reachable. Perhaps if additional instructions are added in some cases more target points are reachable. Find out which combinations are reachable for work object `wobjTestBoardOnTable`, a table as below should be filled out and included in the report.

No.	dx,dy	-180 °	-135 °	-90 °	-45 °	0 °	45 °	90 °	135 °	180 °
1	(0,0)									
2	(0,60)									
...										

c. Finally, for the third option of the menu in the RAPID `main`-function call a new function `test7pegs()` similar to function `test4corners()`. Use pegs as the board layout shown in figure 5 if all these positions are reachable, if not change rotation of pegs, towards 0 °, until the position is reachable. You should not try to change the position and orientation of the pegs on the test board in the RobotStudio station but keep the layout as it is, figure 3, and thus allow collisions during simulation. However, you should see that the robot movement is as expected. Include the used rotations for each peg, and the total time as reported on the FlexPendant in the report.

d. Optionally, for the fourth option of the menu in the RAPID `main`-function call a new function `options()` that let the operator select which work object to use, i.e. above table or above conveyor belt, and select value of `doSlow` parameter.

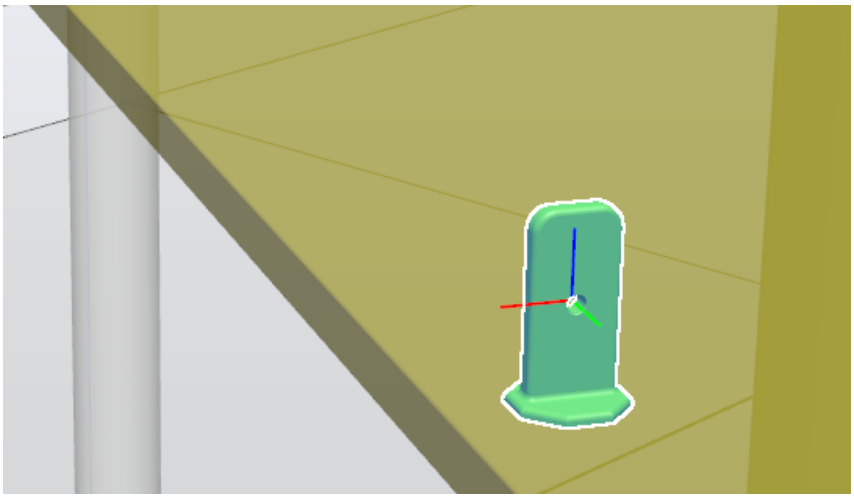

Figure 4: One free peg on the table. The pegs are simple flat surfaces with a small hole in the middle, the origin of the free peg is in the center of this hole, but the pegs on the simulated test board have origin far away somewhere. The actual pegs has a rod on the bottom so it can be placed onto the test board. The pegs are designed to be the weak point in any collision with the tool and relatively expendable. The actual pegs can be placed in different patterns on the actual test board, but to change the setup in the visualization in the RobotStudio station is more difficult.

4.4 Using TATEM tool on Rudolf in UiS lab.

Now, you should do the spot weld simulation using the actual TATEM tool on the ABB robot Rudolf in UiS laboratory. The test board should be placed within working distance from the robot, either on the table or on the conveyor belt, and some pegs should be placed on the test board, at least in each of the corners. First, let the orientation of each peg be 0 °. A free peg is placed on the table in the RobotStudio station, figure 4.

4.4.1 Make welding simulation work

Now, you should run the RAPID program from the previous section on the actual robot Rudolf. Follow the procedure given in the RS2 assignment. First run the program `test4corners()` and check that the robot moves work as intended without the actual test board on the table. To make the work object for the test board and the actual test board match there is two options, either move the actual test board to the used work object or define a new work object to the actual position of the test board on the table. I recommend to use the first option. Use the program and place the robot tool where the peg in origin is supposed to be, when the tool is activated, i.e. the LEDs turn on, release the motors on switch on the FlexPendant and the robot stops. After the robot has

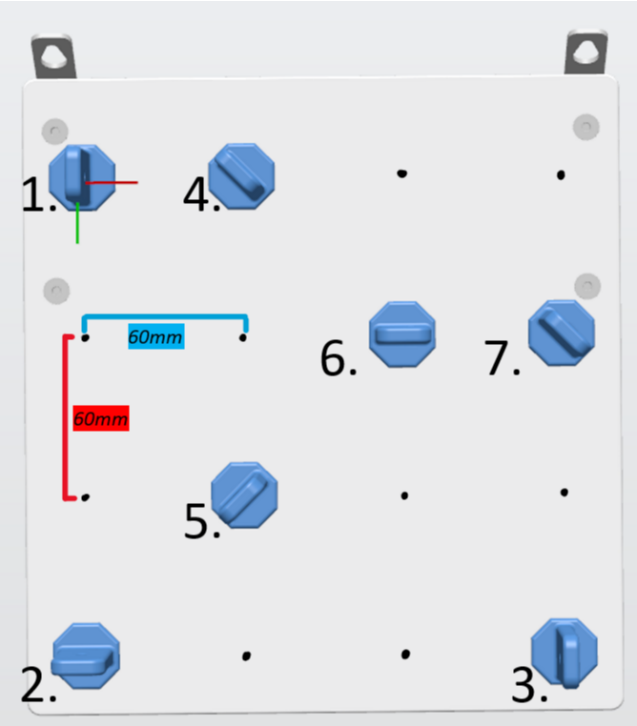

Figure 5: The test board where the numbers indicate the welding order. Here, the peg numbered 2 is the one in origin, x-axis of work object `wobjTestBoard` is toward the peg numbered 3 and y-axis is toward the peg numbered 1. The length between any two grid points horizontally or vertically is 60 mm.
For the actual test boards the pegs may have different orientations, preferable in 45 °increments, making in total 4 possible rotations. A ruler can be used to align the edges of each peg base. You can easily and accurately set the rotations of the pegs by placing a ruler horizontally on each row so that the flat side of the octagon base of the pegs are all aligned with the ruler.

stopped, place the actual test board on the table so the corner peg is enclosed by the tool. As the LEDs are on you will find this helpful to get best placement here. Perhaps the height of the work object needs a minor correction to match the actual height of the peg hole. Then, use the program and move the robot tool to the peg in the opposite corner and repeat the procedure done for the first peg. Adjust the position of the test board, and perhaps the z-value of the work object, until they match just perfect. Let variable `doSlow := TRUE;` and run the program `test4corners()` and assert that the outer LED shines through each of the holes in the four pegs.

Finally you should also check the report from the TATEM tool, see subsection 4.5.5. You should get a report like the in figure 6. Include the total time as reported on the FlexPendant in the report and the time between start of first peg and start of last peg in TATEM tool report, in figure 6 this is 8.5129 s.

```
(py38) C:\Users\karl\Dropbox\ELE610\py3>python showJSON.py
Try to parse file C:\TFS\TATEM\datasets\2023-11-16_09-41-14.json
File C:\TFS\TATEM\datasets\2023-11-16_09-41-14.json contains a list of 4 elements.
Event   0 is Success    tA=  30.8   tO= 399.4  tR= 236.2 [ms]  (time since first event is   0.0000 [s])
  More times          tA2= 100.4 DOon= 499.8 tR2= 236.2 L1on= 736.1 [ms]
Event   1 is Success    tA=  21.4   tO= 400.0  tR= 247.0 [ms]  (time since first event is   2.7871 [s])
  More times          tA2= 100.2 DOon= 500.2 tR2= 247.0 L1on= 747.1 [ms]
Event   2 is Success    tA=  34.5   tO= 400.0  tR= 237.9 [ms]  (time since first event is   5.9370 [s])
  More times          tA2= 100.2 DOon= 500.1 tR2= 237.9 L1on= 738.0 [ms]
Event   3 is Success    tA=  40.0   tO= 400.2  tR= 237.6 [ms]  (time since first event is   8.5129 [s])
  More times          tA2= 100.0 DOon= 500.2 tR2= 237.6 L1on= 737.8 [ms]
Summary: 4 Success, 0 TaError, 0 NotDefined, 0 Other.
```

Figure 6: The report summary from `showJSON.py` after running program `test4corners()`

4.4.2 Make welding simulation successful

Now, let variable `doSlow := FALSE;` and run the program `test4corners()`. Check the report from the TATEM tool again and hopefully it is 4 success again, see figure 6. If not, you have to adjust some of the time variables in the program, see subsection 4.5.3. Include the total time as reported on the FlexPendant in the report and the time between start of first peg and start of last peg in TATEM tool report.

4.4.3 Make welding simulation on 7 pegs

Now, let variable `doSlow := FALSE;` and run the program `test7pegs()`. Check the report from the TATEM tool again and hopefully it is 7 success. You may have to activate the tool a shorter time before the tool reach its target point, or let it be activated a little bit longer. It may also be that the time the tool stand still at the target point should be increased. Include the three lines (instructions) from your RAPID code where these three times are used. When it works well also include the total time as reported on the FlexPendant in the report and the time between start of first peg and start of last peg in TATEM tool report.

4.4.4 Make welding simulation faster

The main task in this assignment will be to improve the implementation of the welding operation done in RAPID function `test7pegs()` where variable `doSlow = FALSE;`. It may be possible to increase speed from the previous subsection without getting any errors in the TATEM tool report. You may also try to reduce the three times used in timing instructions, you may try to increase path speed or path movements.

When it works fast and well include the total time as reported on the FlexPendant in the report and the time between start of first peg and start of last peg in TATEM tool report.

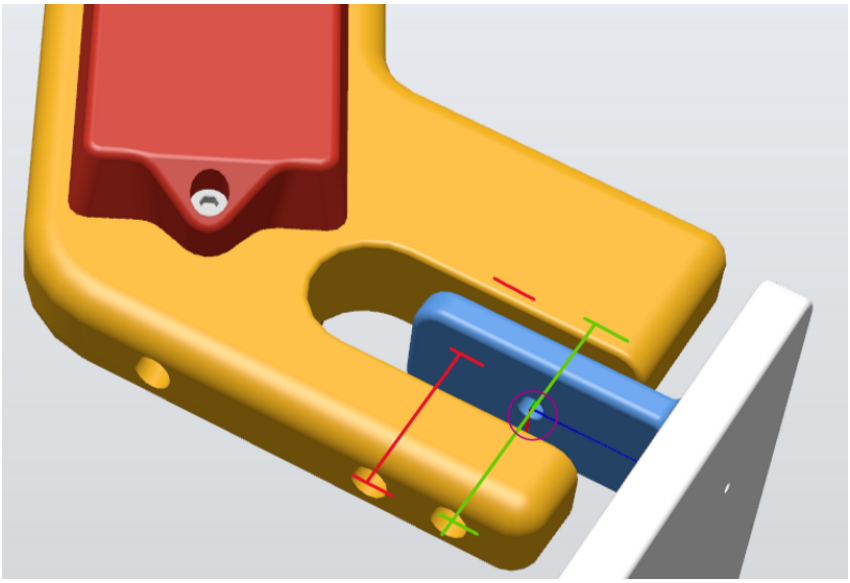

Figure 7: The TATEM tool is here positioned over a peg where welding is simulated.

You may also set the robot in automatic mode, which allows it to move faster than in manual mode, and include the total time as reported on the FlexPendant in the report and the time between start of first peg and start of last peg in TATEM tool report now.
Finally you should show the final Rudolf solution and its results to the teacher.

4.5 Rapid and Python details

4.5.1 RAPID instructions

Some special RAPID functions and instructions are used in the example in the Pack and Go file, `UiS_RS4_16nov2023.rspag` ↗. You should see RAPID documentation for complete explanation, below a brief explanation is given for these.

- `RelTool` is a function to add a displacement and/or a rotation to a robot position. The arguments are used relative to the tool coordinate system, while the `Offs`-function use displacement but no rotation relative to the work object coordinate system.
- `TriggIO` define a fixed position or time I/O event near a stop point. It is used in `PROC initTatemTool()` as:
  `TriggIO PGunOn, tAdelay\Time \DOp:=AirValve, 1;`
  This instruction connects the `triggdata` variable `PGunOn` to the DO signal `AirValve` and set the startup time related to when the tool is

tAdelay seconds before reaching the target point. The last argument, 1, is the value to assign to the DO signal when triggered. Interrupt is used to set the DO signal AirValve off (value 0) some time after it is set.

- IDelete Interrupt Delete is used to cancel (delete) an interrupt subscription. It is used in PROC initTatemTool() as:
  IDelete igun_on;
  to delete the interrupt named igun_on before it is created in next line

- CONNECT is used to find the identity of an interrupt and connect it to a trap routine. The trap routine in our program is ResetSignal, and in this routine ISleep is used to turn interrupt temporarily off while the routine reset (= set value to 0) the digital out (DO) signal AirValve after a given time. In the end IWatch is used to turn interrupt on again. It is used in PROC initTatemTool() as:
  CONNECT igun_on WITH resetSignal;
  to set an interrupt named igun_on and connect it to function TRAP resetSignal.

- ISignalDO Interrupt Signal Digital Out is used to order and enable interrupts from a digital output signal. It is used in PROC initTatemTool() as:
  ISignalDO AirValve, 1, igun_on;
  to let the IO-signal AirValve raise an interrupt igun_on when it is set on.

- ISleep and IWatch are used in TRAP resetSignal to temporarily turn a named interrupt off and on.

- SetDO set an digital output to a given value. It is used in TRAP resetSignal as:
  SetDO \SDelay:=tA+tOp+tOpdelay, AirValve, 0;
  to set IO-signal AirValve off (to 0) after a given delay time (in seconds). This is done without interfering with the rest of the program. As this interrupt is raised exactly when the IO-signal AirValve is turned on, the time here is exactly the time the IO-signal is on.

- TriggL activate the trigger synchronized to the robot linear movement, as specified in the trigger definition above. It is used in PROC doPeg00(..) when doSlow = FALSE as:
  TriggL rt, vSlow, PGunOn, fine, TatemTool1\WObj:=wobjTestBoard;
  where rt is a robtarget as returned by the RelTool function.

4.5.2 `MainModule` functions

This give some extra explanation for the `MainModule` in the Pack and Go file, `UiS_RS4_16nov2023.rspag` ↗. The file start by defining some (global) variables: The tool `TatemTool1`, some work objects, `wobjTestBoard` is used in program and set to either `wobjTestBoardOnTable`or `wobjTestBoardOnBelt`, and some `speeddata`-constants. Then some global time variables, these are not easy to fully understand but I try to do this in next subsection. The three most important are `tA`, `tOp` and `tR`, which should not be changed as they are also used by the TATEM tool (see documentation from ABB available on *canvas*). Also some target points are defined as `robtarget`, not for all pegs but actually only one peg is used in the following code, or `jointtarget` which are useful for setting initial absolute position for the robot axis as these should be dependent on where the test board is placed. Finally the trigger and the interrupt is defined and three logical variables for selecting what to do, or more precisely who to do it.

The functions in `MainModule` in the Pack and Go file, `UiS_RS4_16nov2023.rspag` ↗ are listed below.

- `PROC main()` simply does some initialization and selects which task to run.
- `PROC testPeg00()` is a simple setup for how a task can be defined. Here it goes to peg at origin, i.e. the lower left peg on the test board as seen from the front of the test board, and simulate a welding operation on this peg.
- `PROC test4corners()` is is similar, but here the four corners on the test board is done.
- `PROC doPeg00(num dx, num dy, num rotz)` do weld simulation relative to peg at origin. Two different ways are tried, one slow and one fast. Both could probably be done faster and still get success from the TATEM tool report. The `doSlow` case is quite simple as it uses either trigger or interrupt.

  The `NOT doSlow` case is more involved. A trigger is used to activate the TATEM tool, i.e. turn I/O-signal `AirValve` on, a short time before reaching the target point. An interrupt is raised when the I/O-signal `AirValve` is turned on, and this interrupt function then turn of the I/O-signal after a given time and deactivate the tool.
- `PROC initTatemTool()` will initialize the trigger and set up the interrupt.
- `TRAP resetSignal` is the interrupt trap function that runs when the interrupt happens. Here it is just a short time before the tool reach the

target point. This function turn off (value 0) the DO signal `AirValve` a delayed time after the function is called.

4.5.3 RAPID time constants

The RAPID program uses some time constants for different times needed(?) for the TATEM welding tool to execute a proper weld, shown in the list below.

- `tA` is the time it takes to activate the tool, closing the clamp
- `tA_delay` is the amount of time before entering the point that the DO signal will be turned on and activate the laser. It should be less than `tA`
- `tA_diff` is the amount of time the robot should stand still in its position before executing the spot welding.
  It should (perhaps ?) be equal to `tA - tA_delay`.
- `tOp` is The execution time of the task
- `tOp_delay` is the extra time to make sure the task is finished before retracting the tool
- `tR` is the time it takes to retract the tool, opening the clamp
- `tR_delay` is the extra time to make sure that the tool has been reopened before exiting the point
- `tWait` the time that the tool is not moving, used in `doPeg00()`.

These values can be changed, but for this lab assignment at least `tA=0.1`, `tOp=0.3` and `tR=0.1` should be fixed. How these times are used in RAPID should match the TATEM tool, section 4.5.4,

4.5.4 The TATEM tool

The TATEM tool features two lasers and two opposing sensors, where each laser-sensor pair can individually determine if the path between them is obstructed. The figure 7 shows the tool when it has arrived at a welding spot which in simulation is given by a peg on the test board. The tool is ready to start the welding process, and should be in this position for a predefined time. In this position the laser marked in green passes through the hole in the peg, while the laser marked in red is blocked. The software on the tool measure the time the tool position and orientation is kept, and gives an error signal if it is shorter than the predefined time. The TATEM tool should be mounted on

the robot Rudolf. If the pen tool is mounted on Rudolf you should unmount it and mount the TATEM tool, remember to also attach the cable.

The welding operation is characterized by the tool moving between the various pegs on the board. Each peg represents a desired welding target, and the tool needs to move to each target one by one, and place itself at the point (peg) in the specific angle and position, to correctly perform the operation. Once the target is reached, the tool should be activated for a short period to 'weld' it. When the TATEM tool is activated, the Arduino program in the tool will turn on the lasers and start a clock. If the outer laser, the green one in figure 7, goes through the hole and the inner laser, the red one in figure 7, is blocked the tool is assumed to be in the correct position and orientation. If the correct position is kept for the entire activation (the predefined welding time), the weld is successful. Otherwise, the weld has failed and an error signal will be transmitted from the tool. This will be logged by the Arduino program.

The TATEM tool documentation from ABB should be available on *canvas*, in Modules on the ELE610 page. The state machine, shown in figure 8, shows the different states of the tool. Short names are used for the states: Starting from Init State and going clockwise we have IS, TAS, TOS, EOS, TRS, ERS, and finally RS for Reset State and ES for Error State. The two photodiodes are denoted as S1 (the outer one) and S2 (the inner one), when the tool is in correct position over a peg S1 should be on and S2 should be off `(S1 AND NOT S2)`. As I understand it:

a. The tool leaves IS and enters TAS when the tool is (very) close to a new peg, the tool gets a signal (the signal is turned on) from the robot. Both LEDs are turned on. This signal is confusingly denoted as `AirValve` in RAPID and I/O in figure 8. The tool is in TAS for `tA` s. The tool may move, actually it should do the final movements to the peg. If the signal is turned off in TAS an error happens giving NotDefined-error. The photodiodes are not checked in TAS.

b. The tool enters the TOS where it should stay for `tO` s, it may stay longer. If the tool moves in TOS, actually `NOT (S1 AND NOT S2)`, an error happens giving tA-error. A tA-error happens when the tool is activated more than tA s before entering the test point. If the signal is turned off in TOS an error happens giving tO-error. A tO-error happens when the tool is deactivated less than `tO` s after the operation has started.

c. The tool enters the EOS where it stays until the signal (I/O) is turned off. This time should be short. The tool should not move in this state, if it does an error happens giving NotDefined-error.

d. The tool enters the TRS where it stays for `tR` s. If the tool moves in TRS, actually `NOT (S1 AND NOT S2)`, an error happens giving tR-error. A tR-error happens when the robot stands still for less than `tR`

s after the tool has been deactivated. This would mean that the robot would start moving before the tool has finished retracting. If the tool is (re)activated, the I/O signal is turned on, in TRS an error happens giving NotDefined-error.

e. The tool enters the ERS where it stays until it moves from the peg, actually `NOT (S1 AND NOT S2)`, normally S1 goes off. Both LEDs are now turned off. During this state the I/O signal should still be off, if not an error happens giving NotDefined-error. The time in this state should be short.

f. Finally, the loop is completed and the tool is in IS again. Here, it may move to a new peg before the cycle is repeated.

Thus the I/O signal should be on for at least `tA+tO` microseconds, in Tool Activation State, Tool Operation State and Extended Operation State. After the I/O signal is turned off it should stay off for at least `tR` microseconds, in Tool Retraction State and Extended Retraction State. The tool should stand still over the peg for at least `tO+tR` microseconds, in Tool Operation State, Extended Operation State, Tool Retraction State and Extended Retraction State. The LEDs should be on for at least these four states, but may be on also in Tool Activation State even though any of the two photodiodes changes value. Both the extended states EOS and ERS should be short.

4.5.5 Get report from TATEM tool

The TATEM tool attached to Rudolf records data during the simulation process. This data file can be downloaded to a computer and viewed. The `cmd2`-based program `tatemCom.py` is used for communication with the TATEM tool. It also uses some other python files, all of the needed files should be included in `ELE610py3files.zip` ↗ and after unpacking be in your `..\ELE610\py3\` catalog.

When the `tatemCom.py` program is started you get an adapted `cmd2` CLI (Command Line Interface). Several commands are available as can be seen with the `help` command, see figure 9. If you are within Bluetooth range of the welding tool you can connect to the TATEM tool, more specifically the Arduino processor inside the tool. The command `getReport` should display several bytearrays while the report is downloaded from the TATEM tool, then the report is saved as a JSON (JavaScript Object Notation) file on your computer most likely in catalog `C:\TFS\TATEM\datasets`, finally a summary of the report is shown. A typical session may include commands as below.

```
(py39) ..\ELE610\py3M\> python tatemCom.py
tatem>discover    (see available Bluetooth addresses)
tatem>connectArduino    (uses default MAC-address)
tatem>setTvalues    (tA, tO and tR)
tatem>resetArduino    (delete old results)
.. run weld test on Rudolf ..
tatem>getReport    (download report from TATEM tool)
tatem>load fn    (display summary of file fn)
tatem>exit
```

4.5.6 Optional way to view result data

The brief report summary which can be shown from the `tatemCom.py` program using the `load` command, perhaps including an optional argument for the file name, should be sufficient for viewing the TATEM report. Actually, this summary can also by shown by running a much more simple Python program `showJSON.py` that should also be in the ELE610py3files.zip ↗.

```
(py39) ..\ELE610\py3\> python showJSON.py    (use newest file)
(py39) ..\ELE610\py3\> python showJSON.py filename    (use given file)
```

However, the `tatemCom.py` program must still be used to get the TATEM report. You may also look at the downloaded JSON data file using a simple text editor.

4.5.7 Graphical view of TATEM report file

It is also possible to view the TATEM log-files, i.e. the JSON files or reports downloaded from the TATEM tool to your computer, as a graphically visualization on a (locally generated) web-page. The web-page is generated by the Python `dash`-package used (imported) in `appTATEM.py`. The `appTATEM.py` file and the `AppTATEM`-catalog should all be included in ELE610py3files.zip ↗ and after unpacking be in your `..\ELE610\py3\` catalog. Note that to run this file several Python packages should be instilled in your (conda) environment, in particular `pandas`, `dash` and `plotly`. To create the web-page simply run the program

```
(py39) ..\ELE610\py3\> python appTATEM.py
```

The program display some information while the web-page is created. View the web-page in any web-browser. Here you should be able to select which file to show, as long as it is in the catalog where the files are assumed to be. The results may look something like in figure 10 and figure 11. When you are finished, you can stop the web-server by Ctrl+C.

The left half of the figure 10 shows the number of successes and failures of different kinds. A tA failure is when the tool is activated more than tA seconds before reaching the peg, a tO failure is when the tool is deactivated less than tO seconds after the operation has started. In a future version a tR failure should happen when the tool moves before tR seconds have passed after tO, but this isn't currently applicable.

The right half of figure 10 shows the average time taken for the various phases of the weld. Here, the tA phase is duration between tool activation and the robot stopping movement, the tO phase is the duration between the robot stopping movement and the tool deactivating, and tR is the duration between the tool deactivating and the robot resuming movement. This could be useful to further optimize your program. In addition, the right side of figure 11 shows these times for each individual weld attempt.

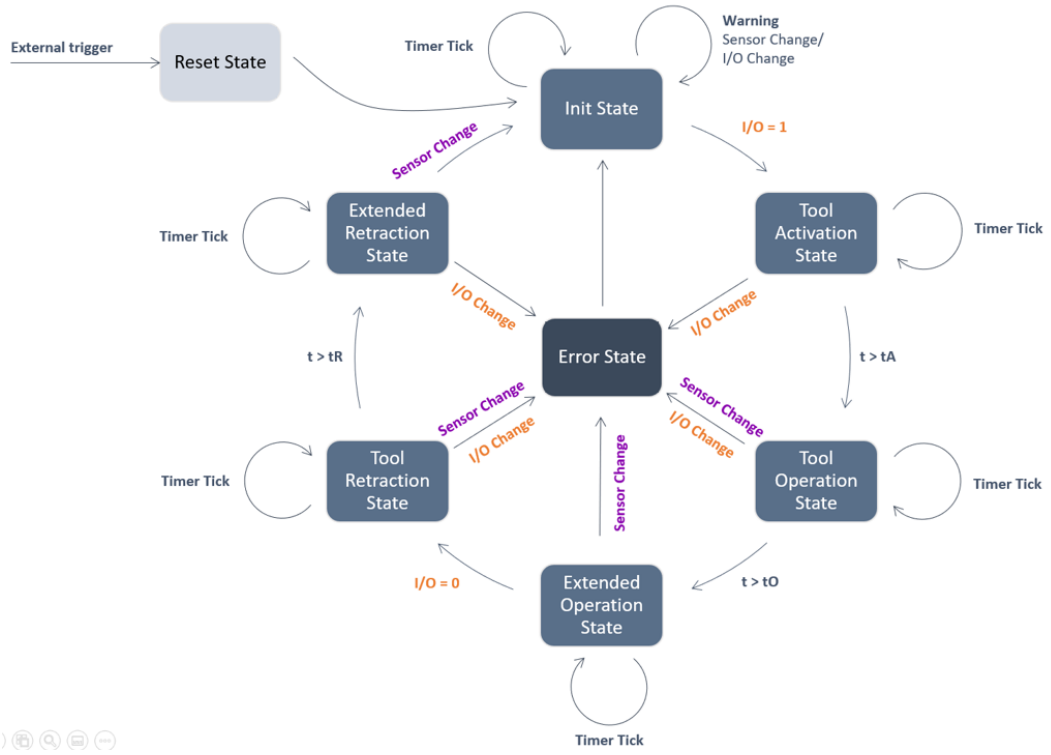

Figure 8: The state machine for the TATEM tool. This is from the ABB documentation available on *canvas*. In text the following short names are often used for the states: Starting from Init State and going clockwise we have IS, TAS, TOS, EOS, TRS, ERS, and finally RS for Reset State and ES for Error State. Sensor change for a transition means that the robot moves when it should not. When the tool is in correct position over a peg S1 should be on and S2 should be off `(S1 AND NOT S2)`, and robot movement is first detected by S1 going off. I/O change is when the signal for tool activation is turned off too soon (after it was turned on) for TAS and TOS, and if it is turned on too soon (after it was turned off) for TRS and ERS.
The transaction from TOS to ES raises a tA-error if the robot moves and a tO-error if the activation signal is turned off.
The transaction from EOS to ES raises a tO-error or tR-error, I'm not sure.
The transaction from TRS to ES raises a tR-error if the robot moves.

```
(py11) C:\Users\2900780\Dropbox\ELE610\py3>python tatemCom.py
Welcome to the TATEM application for ELE610, ver.23.11.08
tatem>help

Documented commands (use 'help -v' for verbose/'help <topic>' for details):
================================================================================
alias                   disconnectRobot  help     resetArduino  set_tR
checkArduinoConnection  discover         history  run_pyscript  setTvalues
checkRobotConnection    edit             info     run_script    shell
connectArduino          exit             load     set           shortcuts
connectRobot            getReport        macro    set_tA
disconnectArduino       getTvalues       quit     set_tO

tatem>
```

Figure 9: Use of `tatemCom.py`

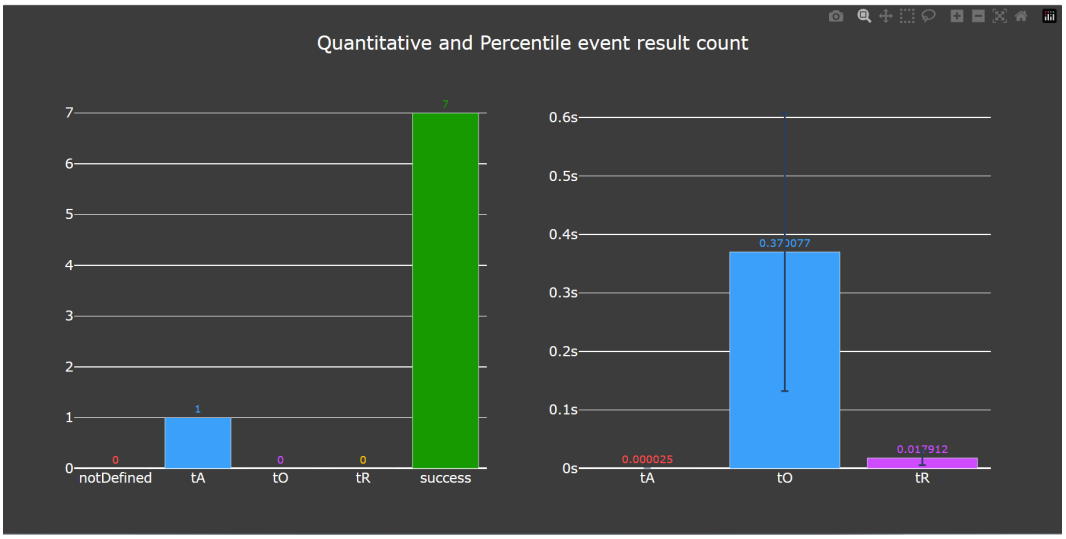

Figure 10: Example for visualization of results, averages

	n	eventStart	eventEnd	eventDuration	operationStart	doOff	operationDu	tA	tO	tR	eventResult
☐	0	-1456691803	-1456157241	534562	-1456689533	-1456191896	497637	32	497637	34655	Success
☐	1	-1450167909	-1449644125	523784	-1450165443	-1449666943	498500	32	498500	22818	Success
☐	2	-1444832669	-1444311605	521064	-1444830344	-1444332220	498124	55	498124	20615	Success
☐	3	-1441832932	-1441299305	533627	-1441830462	-1441332981	497481	32	497481	33676	Success
☐	4	-1435007444	-1434486256	521188	-1435005114	-1434507483	497631	32	497631	21227	Success
☐	5	-1431253041	-1430729008	524033	-1431250728	-1430753616	497112	32	497112	24608	Success
☐	6	-1427398160	-1426877483	520677	-1427396031	-1426897701	498330	31	498330	20218	Success
☐	7	-1423402377	-1	1423402376	0	0	0	0	-1	-1	TaError

Figure 11: Example for visualization of results, individual welds