



University of
Stavanger

Faculty of Science
and Technology

Stavanger, September 12, 2023

ELE610 Applied Robot Technology, autumn 2023

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.

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 by another Python program to decide whether the simulation was successful or not.

Approval of this assignment can be achieved by demonstrating the operation to the teacher, and then submitting a report and RAPID code on *canvas*. Note that the RAPID code should be submitted preferably as a txt-file, or alternatively as a pdf-file where the code is clearly marked, for example by using courier font.

This assignment does not demand a perfect solution. However, it still expects you to attempt to optimize your solution within the recommended time of twenty hours.

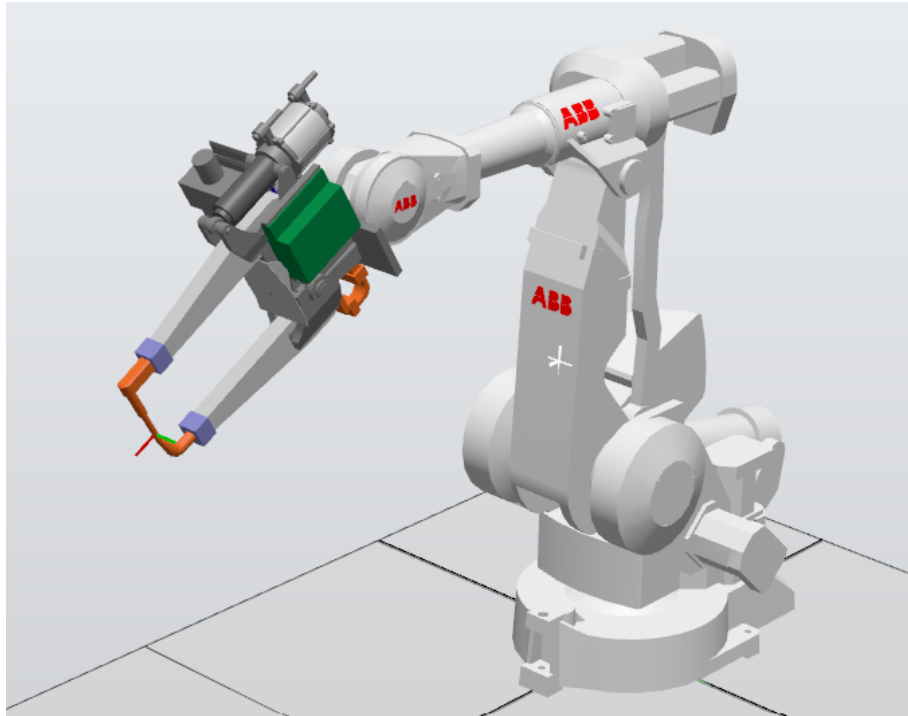


Figure 1: Robot equipped with welding tool. The electrode is the orange colored part of the tool.

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. 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. Spot welding is used in many industries, such as the automotive industries. An example video of a manufacturing process for producing car components including spot welding can be seen [here](#).

4.2 Python preliminaries

In this assignment you will use Python to transfer data from the TATEM tool to a PC. Python is also used to view the simulation results. For these two tasks the needed files are collected into a zip-file, [TATEM.zip ↗](#). This file should be copied into your PC and unpacked into a catalog `..\ELE610\TATEM\`, the unpacked catalog will contain 0.6 MB in 73 files in 4 catalogs.

It may be a good idea, but probably not needed, to create and use a (new) virtual environment for this assignment. It is recommended you use Python 3.9 or higher, perhaps also version 3.8 will do? You can create the virtual environment in your preferred interface or IDE (Integrated Development Environment), in the following the interface is assumed to be Anaconda.

The current catalog on your computer should probably be `..\ELE610\TATEM>` instead of `C:\>` as shown in the lines below. The Anaconda commands to create a new environment (here `py39`), list all available environments, activate the new environment, list all packages installed in the active environment, and install the wanted packages using `pip` are:

```
(base) C:\> conda create --name py39 python=3.9
(base) C:\> conda env list
(base) C:\> [conda] activate py39    (conda is optional)
(py39) C:\> conda list
(py39) C:\> pip install package
```

You will need to load the packages listed below into your virtual environment. Note that, with the exception of `rwsuis`, using the exact version listed is a minor concern. Using older versions would likely not pose a problem, and using newer versions is also unlikely to do so. Below the version used by RWS and JAMR is listed first followed by the version selected by Anaconda for Python 3.9 in august 2023.

- `cmd2` version 2.4.3
This is a python package for building powerful command-line interpreter (CLI) programs. See [cmd2 documentation on web ↗](#).
- `pandas` version 1.5.3 or 2.0.3
This is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language. See [pandas user guide on web ↗](#).
- `plotly` version 5.13.0 or 5.16
The `plotly` Python library is an interactive, open-source plotting library that supports over 40 unique chart types covering a wide range of statistical, financial, geographic, scientific, and 3-dimensional use-cases. See

[plotly on web ↗](#). It is recommended to use `plotly` together with `dash`.

- `dash` version 2.8.1 or 2.12.1
Dash is the original low-code framework for rapidly building data apps in Python. See [dash on web ↗](#). It is recommended to also install `pandas`.
- `bleak` version 0.19.5 or 0.20.2
Bleak is an acronym for Bluetooth Low Energy platform Agnostic Klient. It is a GATT client software, capable of connecting to BLE devices acting as GATT servers. It is designed to provide a asynchronous, cross-platform Python API to connect and communicate with e.g. sensors. See [bleak for Python ↗](#).
[GATT ↗](#) is an acronym for the **Generic ATtribute** Profile, and it defines the way that two Bluetooth Low Energy devices transfer data back and forth using concepts called Services and Characteristics,
- `numpy` version 1.21.2 or 1.25.2
[numpy ↗](#) is the fundamental package for scientific computing with Python.
- `matplotlib` version 3.5.0 or 3.7.2
[Matplotlib ↗](#) is a comprehensive library for creating static, animated, and interactive visualizations in Python.
- `pytest` version 6.2.5 or 7.4.0
The `pytest` framework makes it easy to write small tests, yet scales to support complex functional testing for applications and libraries. Can be used to do [unittest-based tests ↗](#). See [pytest for Python ↗](#).
- `rwsuis` version 0.2
This is the RWS package made for UiS (Universitetet i Stavanger).
- `alive-progress` version 3.0.1 or 3.1.4
A new kind of Progress Bar, with real-time throughput, ETA, and very cool animations according to [the installation web-page ↗](#).

4.2.1 Harvest result data using Python

The TATEM tool attached to Rudolf records data during the simulation process. This process needs to be initialized, and how this is done is describe already now so you can be prepared for what to do on the laboratory when you do the work there later on.

With the virtual environment (`py39`) ready, you need to get within Bluetooth range of the welding tool. Then run the `tatemCom.test.py` program and write the connect command after the TATEM prompt in the terminal, the result should look something like in figure 2, and finally clear previous data by `resetArduino`.

```
Welcome to the TATEM application!
tatem>connectArduino A1:F8:14:CE:BA:AC
Connecting to the Arduino...
Connecting to the Arduino...
Connecting to the Arduino...
Connecting to the Arduino...
Connecting to the Arduino...
Connecting to the Arduino...
Connecting to the Arduino...
Connecting to the Arduino...
Connecting to the Arduino...
Connection successful
Connected to Arduino with IP: A1:F8:14:CE:BA:AC
```

Figure 2: Bruk av `tatemCom_test.py`

After `textttresetArduino` you run the RAPID weld simulation program on Rudolf to do the wanted test. When the test is finished, you need to write `getReport` in the terminal. This should display several bytearrays and save a JSON (JavaScript Object Notation) file on your computer at `C:\TFS\TATEM\datasets`. At this point you no longer need to be connected to the Arduino inside of the TATEM tool. End the `tatemCom_test.py` program by `exit`. A summary of commands is below:

```
(py39) C:\ELE610\TATEM\> python tatemCom_test.py
tatem>connectArduino A1:F8:14:CE:BA:AC
tatem>resetArduino .. run weld test on Rudolf .. tatem>getReport
tatem>exit
```

4.2.2 View result data using Python

You may look at the JSON data file using either a simple text editor or simple Python commands or more specialized Python commands. When you unpack the TATEM zip-file you will also get a set of example JSON files, and here we start looking at one of them.

You start (Anaconda) Python and may also use the file `startup.py` load some useful functions, see [littPy3x.pdf](#) ↗ section 2.5. `startup.py` is assumed to be in a neighboring catalog `py3`, but you may have put it elsewhere if you have copied it from ELE610 web-page. This startup-file import the useful `whos()` function which is used below to display Python variables in a more readable way.



You may now get the variables loaded into the workspace, and view workspace variables by `vars()` function or show the resulting dict element in a more readable output by using `whos()`. You may even use the `whos()` function to view a particular *list* of *dict* variable. The summary of commands is below:

```
(py39) ..\ELE610\TATEM> python -i ..\py3\startup.py
>>> import json
>>> fn = 'datasets\\2022-09-07_09-20-31.json' (or another file name)
>>> with open(fn) as user_file: parsed_json = json.load(user_file)
>>> whos(vars())
>>> whos(parsed_json)
>>> whos(parsed_json[0]) (output below)
>>> whos(parsed_json[0]['details'])
>>> whos(parsed_json[0]['details'][0]) (output below)
```

The variable `parsed_json` should be a list, and each element should be similar to the first element shown below.

```
details          list : 5 elements
doOff            int  : 0
eventEnd         int  : -1
eventResult      str, len=10 : NotDefined
eventStart       int  : 161047841
no               int  : 0
operationStart   int  : 0
tA               int  : 0
t0               int  : -1
tR               int  : -1

DO               int  : 1
ERROR            int  : 0
L1               int  : 1
L2               int  : 1
S1               int  : 0
S2               int  : 0
STATE            str, len=19 : ToolActivationState
no               int  : 1
time             int  : 161047841
```

As you see it is not simple to find and display the relevant information from the JSON file. A Python program based on [dash](#)  has been made for this purpose, and is described in next subsection. However, I did not make this file work so I made a much more simple Python program `showJSON.py` that should be in the [TATEM.zip-file](#) .

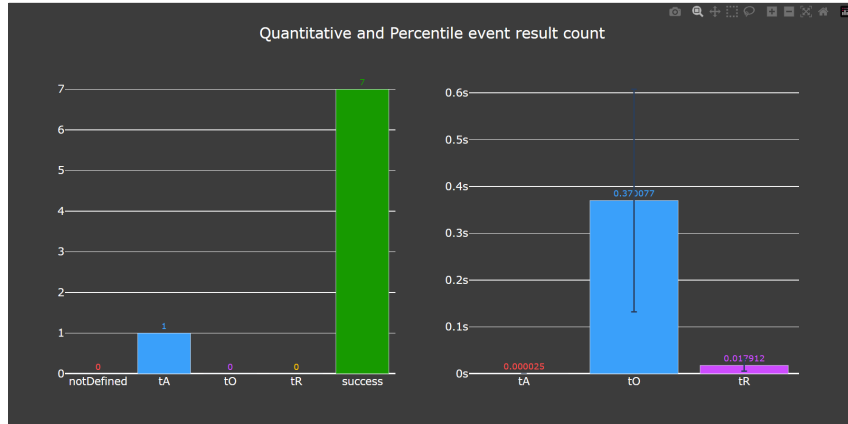


Figure 3: Example for Python-based visualization of results, averages

#	eventStart	eventEnd	eventDuration	operationStart	doOff	operationDt	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 4: Example for Python-based visualization of results, individual welds

4.2.3 Weld test result page

To find and display the relevant information from the JSON file you only need to run the file `app.py` that after unpacking [TATEM.zip](#) should be copied into your PC into catalog `..\ELE610\TATEM\`.

This should give you a showcase of the results of your weld test. But doing this on my laptop nothing happen, the program started but then just waited for Ctrl+C to quit. I had to look at the JSON file using Python as described above, or even more simple just look at the JSON file in a simple text editor. In both cases it is more difficult to find relevant information.

If the file `app.py` runs as intended the results of an example test visualized by `app.py` can be seen in 3 and 4.

The left half of the figure 3 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 3 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 4 shows these times for each individual weld attempt.

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 in the Pack and Go file, [UiS_RS4_sep23.rspag ↗](#), into RobotStudio, then read the explanation below, look at the example in RobotStudio, then run some simulations in RobotStudio, and finally do some changes to RAPID code and assert that you understand what happens.

4.3.1 Explanation of prepared example

Some special RAPID instructions are used in this example. You should see RAPID documentation for complete explanation, below a brief explanation is given for some instructions.

- **TriggIO** define a fixed position or time I/O event near a stop point. It is used as:
`TriggIO PGunOn,tA_delay\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 `tA_delay` 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.
- **TriggL** activate the trigger synchronized to the robot linear movement, as specified in the trigger definition above.
- **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.
- **ISignalDO** Interrupt Signal Digital Out is used to order and enable interrupts from a digital output signal.
- **IDelete** Interrupt Delete is used to cancel (delete) an interrupt subscription.

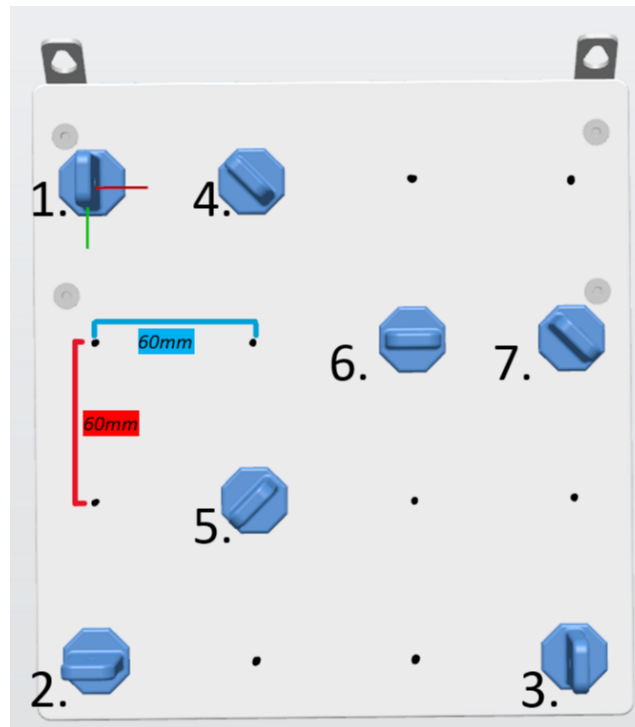


Figure 5: The test board where the numbers indicate the welding order

Some of the functions in `MainModule` in the Pack and Go file, [UiS_RS4_sep23.rspag](#) ↗ are listed below.

- `doRelPeg` do weld simulation relative to peg in upper left corner of test board. Two different ways are tried, one slow and one fast. Note how movement speed here is adjusted using variables of the RAPID data type `speeddata`. See IRC5 Documentation and “RAPID instructions, functions and data types” for explanation of the four values that can be given when a `speeddata` variable is defined.
- `test_10` defines the path where 5 spots are 'welded', i.e. 5 pegs are visited. This is the function used in `main` now, but as you go forward in this assignment other paths should be defined and called from `main`.
- `initTatemTool` will initialize the tool.
- `testPosition` will activate the TATEM tool laser and perform the weld. It set, and turn off, the DO-signal without using the trigger.
- `turnOffIO` will deactivate the tool after the task is finished. Notably, it is strictly necessary to disable the tool before receiving any report data from the TATEM tool.
- `TRAP_ResetSignal` is the interrupt trap function that turn off (value 0) the DO signal `AirValve` a delayed time after the function is called.

There are also some physical constants regarding the time needed for the (actual) 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 must 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

Since the TATEM tool is not a real welding tool, it is not actually constrained by these values, but rather artificially emulates them. These values can be changed, but for this lab assignment they should remain as $tA = 0.1$, $tA_{delay} = 0.06$, $tA_{diff} = 0.04$, $tO = 0.3$, $tO_{delay} = 0.1$, $tR = 0.1$ and $tR_{delay} = 0.05$.

Important to note is that the tool only needs to stand still during the preparatory stages tA_{diff} and tA_{delay} and the 'welding' stages tO and tOp_{delay} . This information as well as the `TriggL`-command in RAPID may prove useful later in this lab assignment. In a realistic welding situation, the tool should also be required to stand still at the beginning of the clamps opening tR but the current TATEM tool will not give an error for this, hence it is not a requirement.

4.3.2 Try prepared example

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 for this RobotStudio simulation, you only have visual control of whether the tool enclose the peg or not and if this last for the wanted amount of time.

Secondly, run some of the other paths (examples) prepared in the example. Try to make the simulation run faster, but keep the tool in the 'weld'-position the defined time, i.e. set `doSlow` to false.

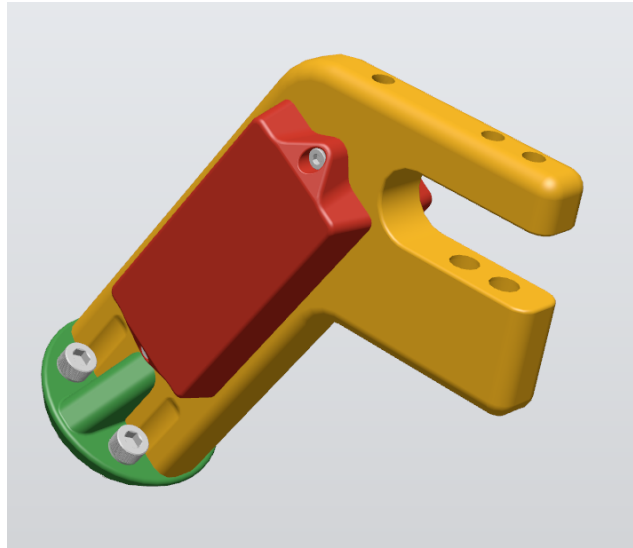


Figure 6: The TATEM tool used for spot welding simulation.

Finally, Make a new path as if the board layout is as in figure 5. You may use the not finished function `do7pegs` for this path. There is also a function `do7pegs3times` that wrap around this function and measure the time by using a `clock`. The time is communicated to the user on the FlexPendant. Include in your report the time used for doing 7 pegs 3 times.

It is not easy to change the position and orientation of pegs on the board in the RobotStudio simulation so I suggest that you make the robot do the wanted path 60 mm above the (now wrongly placed) pegs on the simulation board. This can be done in several ways. Anyway, you should do it so that it is easy to change this height to whatever we want, ex 50 mm or 0 mm to run the same path on Rudolf in the laboratory.

The path in function `do7pegs` uses `doRelPeg(...)`, but an optimal solution where the speed from first to last peg is minimized while still keeping the 'welding'-time for each peg above its minimum time, would more likely move the tool around each peg in a better designed path.

4.4 Simulated spot welding, Rudolf in UiS lab.

The last part in this assignment is to do the weld simulation on the ABB robot Rudolf in UiS laboratory.

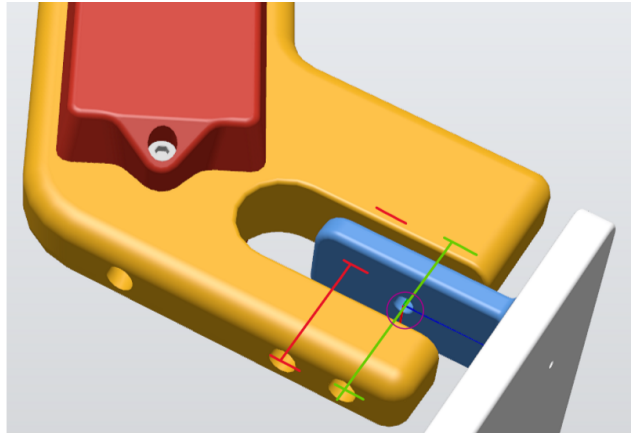


Figure 7: The TATEM tool is here positioned over a peg where welding is simulated.

4.4.1 The actual test board and 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 pegs are simple flat surfaces with a small hole in the middle, as well a rod on the bottom so it can be fitted onto the board. The pegs are designed to be the weak point in any collision with the tool and relatively expendable. The pegs will be placed in patterns on the test board. The test board has small holes in it, placed in a square grid where each grid point is a possible mounting point for the pegs. The length between any two grid points horizontally or vertically is 60 mm. 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.

The test board should be placed on the table beside Rudolf as shown in the RobotStudio Simulation. It need some precision to get this right. You should make a program in RAPID on RobotStudio that simply moves the tool to the upper left peg, and stops there. Transfer this program to Rudolf, and run it

on Rudolf over an empty table. After the robot has stopped place the test board on the table in appropriate position and orientation. The pegs on the test board should be placed such that no collisions occur when the robot moves the tool along the paths as given by the `testColumn` and `testRow` functions. Assert that the test board is correctly placed by running the `testColumn` and `testRow` functions at slow speed. Be ready to stop the robot, release the safety guard on the FlexPendant, just **before** a hard collision is about to happen. Adjust the position of the test board until it is just perfect. You may also have to do some minor changes on the work object `wobjTestBoard` in Rapid before position and orientation of the actual test board and the virtual test board in RobotStudio match perfectly.

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.

Similar to how a real welding process will be executed, the welds must be performed in a specific order, in specific areas and at specific angles as to successfully complete the weld. For testing in UiS laboratory, we will be working with the setup in figure 5.

4.4.2 Make welding simulation work

After proper preparation of Rudolf and correct placement of the test board the pegs on the test board should now be placed as shown in RobotStudio simulation, or even better as like the 5 pegs used in `test_10`. Be careful not to displace the board. To record data on the TATEM tool you should go through the Python steps as described in section 4.2.1. Then run the program. When the path is done, you should test if this trial was successful by inspecting the resulting JSON file. This can be done as described in section 4.2.3. It is likely that there was a failure somewhere, perhaps the tool was not perfectly aligned over a peg long enough. Once you run a test and confirm all the welds were successful, this task is complete.

4.4.3 Optimizing weld times

The main task in this assignment will be to create your own implementation of the welding operation. You should now use the same path as shown in figure 5. Your final solution should go through the path on the board 3 times similar to how it was done in RobotStudio simulation, example `do7pegs3times()`. You should measure the time and this time also record the results. After the path has been done test if this trial was successful by inspecting the resulting JSON file. Once you run a test and confirm all the welds were successful, this task is complete. The report should include the measured time for the path.

If there is still time left you can try to further improve the simulation time. There may be several ways to reduce the time, and you are encouraged to find these on your own and implement them in your solution. This task should take the remainder of the twenty allotted hours. The report should include the improved measured time for the path.

Finally you should show your results to Karl.