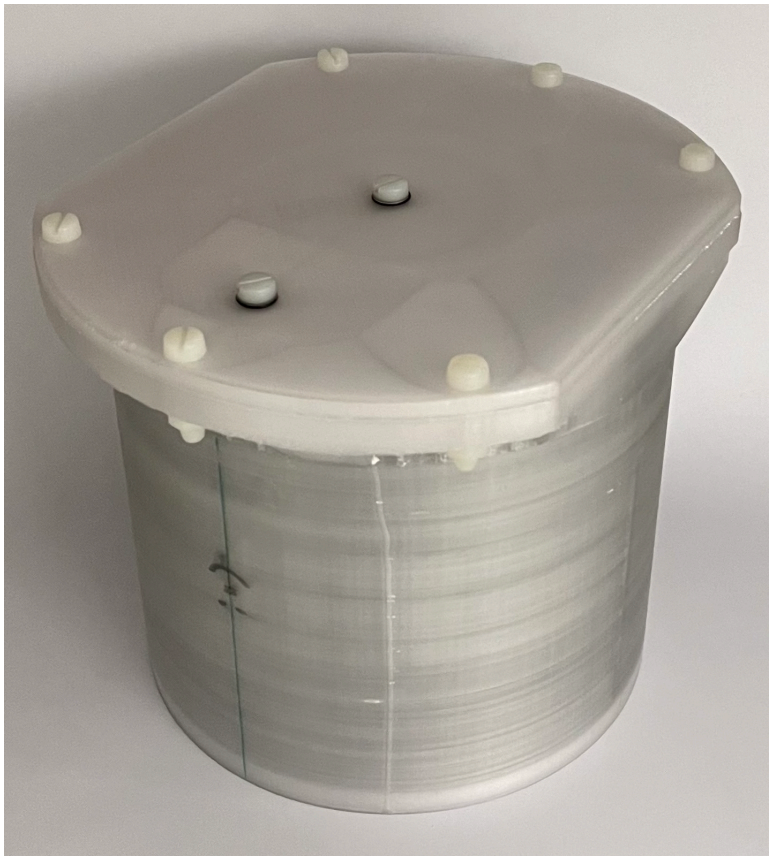
ORQA MANUAL



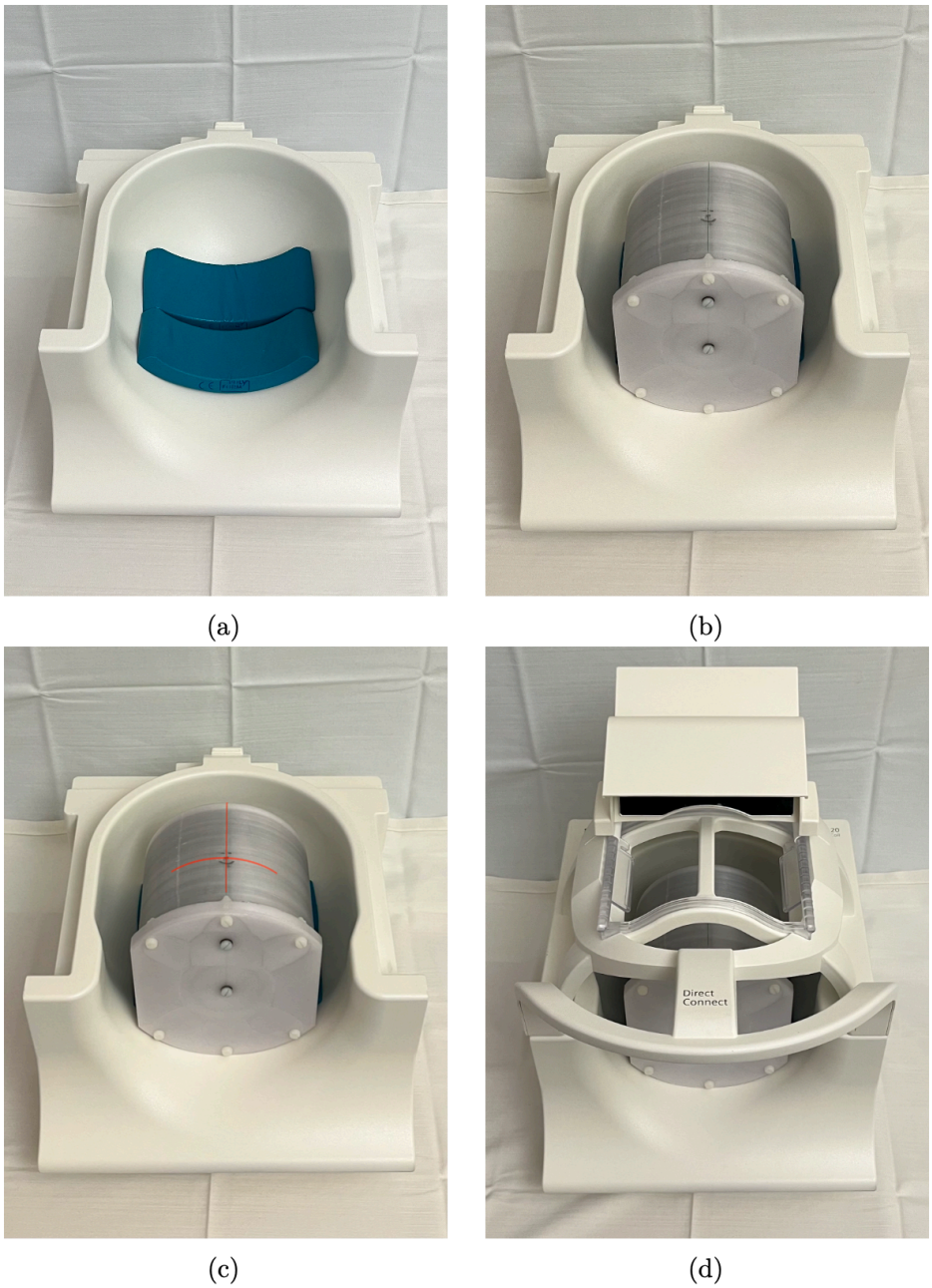
What it is

The Open regular Quality Assurance (OrQA) Protocol is inspired by the ACR Protocol and comes with a custom, 3D-printed phantom. Key considerations are a matching size for all scanners, ease of use for operators in scanning and evaluating the data and furthermore modularity. The last point plays a key role, because it means that improvements can be made fast and without compromising the whole phantom, which will be saving money and especially printing time allowing for rapid prototyping and easy parts replacement.

The OrQA Phantom consists of a cylindrical case which measures 160mm in outer diameter and 142mm in height, not including the lid. The inner part of the phantom is 148mm in diameter and 132mm in height. The inner part houses four rails at the side of the walls. This case houses the modules and is filled with saline solution which is MR-visible. Earlier versions were filled with pure water but had problems with signal outages at 7T caused by suboptimal electric conductivity. The used materials used for the phantom are polyethylene terephthalate glycol (PETG) and polylactide (PLA) which are MR-opaque and have a similar susceptibility to water.

Scanning Protocol

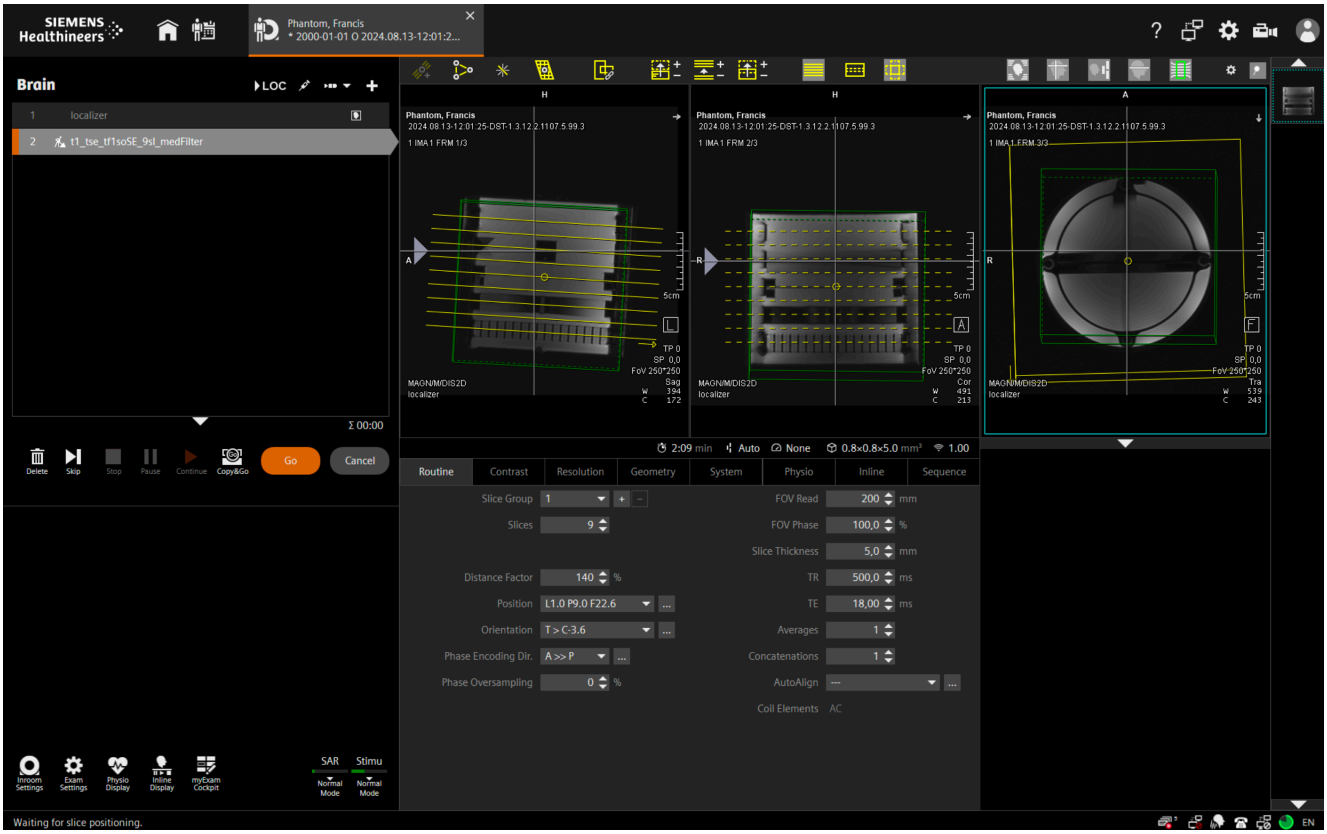
Placement in the scanner



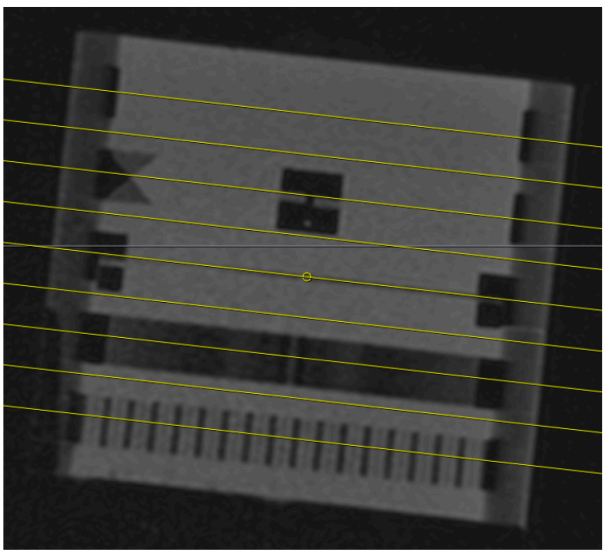
Demonstration of the OrQA Phantom's placement in a head coil. (a) placements of two small pillows in order to make the phantom sit more upright and to prevent the phantom of sliding in the head coil. (b) the placement of the phantom on top of the small pillows. The phantom shall be securely sitting in the coil and using the alignment laser the axial rotation shall be aligned with the vertical line on the phantom. (c) Same image as (b) but with demonstration of laser alignment on the phantom. Note that the lasers vertical line goes through the middle line and the horizontal through the "nose" of the phantom. (d) shows the aligned phantom in the fully assembled head coil.

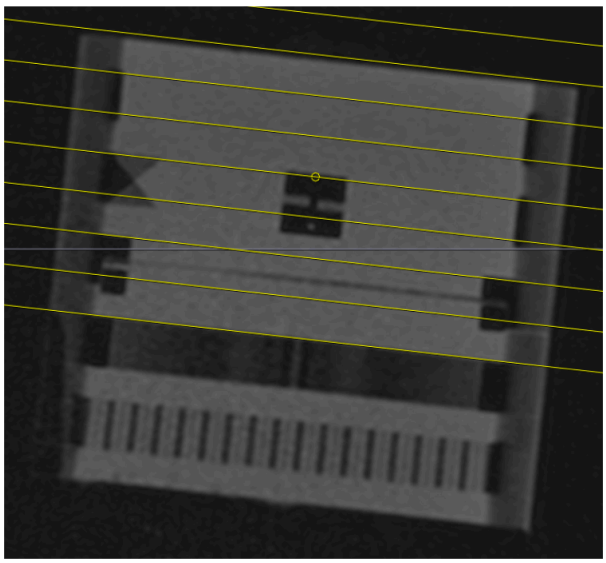
Positioning of Slices

By default should the positions already be fine, since the positioning of the phantom in the head coil is done the same way. If necessary, aim the slices parallel to modules (image on the left and middle image) and without rotation (right image).



Use modules as guide for parallel aiming





Scanning sequences

The corresponding sequences are either already saved on the scanner (MR1, MR2, MR13) under the name OrQA or Francis.

To create the QA sequence from scratch, use the following parameters:

B_0	Sequence Name	Type of Sequence	TR	TE	Resolution	FOV [mm]	# Slices	Slice Thickness	Slice Gap
1.5T, 3T	QrQA T1	Spin Echo	500ms	20ms	256 x 256	256x256	9	5mm	7mm
7T	OrQA T1	Spin Echo	500ms	20ms	256 x 256	200x200	9	5mm	7mm

With enabled B1 filtering and disabled acceleration. The same goes for ACR sequences.

Use of Evaluation Software

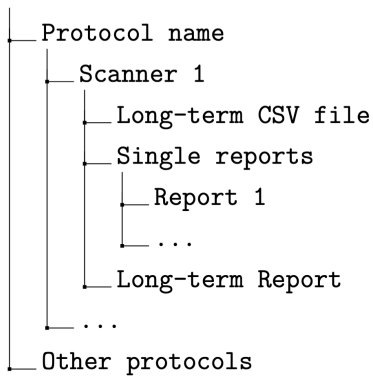
Equivalent for all protocols. Runs inside the provided Conda environment "Karo".
Note: The name "Francis" in the software refers to OrQA which has historical reasons
The base script looks like this:

```python
import mrphantomqa

path_to_dicom = ""
savedir = ""


dfs = mrphantomqa.dicomFolderScanner(path_to_dicom)
dfs.choose_scan_via_menu()
dfs.get_data()

Analyzer = mrphantomqa.francisAnalyzer(dfs, savedir)
Analyzer.runall()

```

path_to_dicom is the path the dicom data is stored at
savedir is the path where the reports will be saved

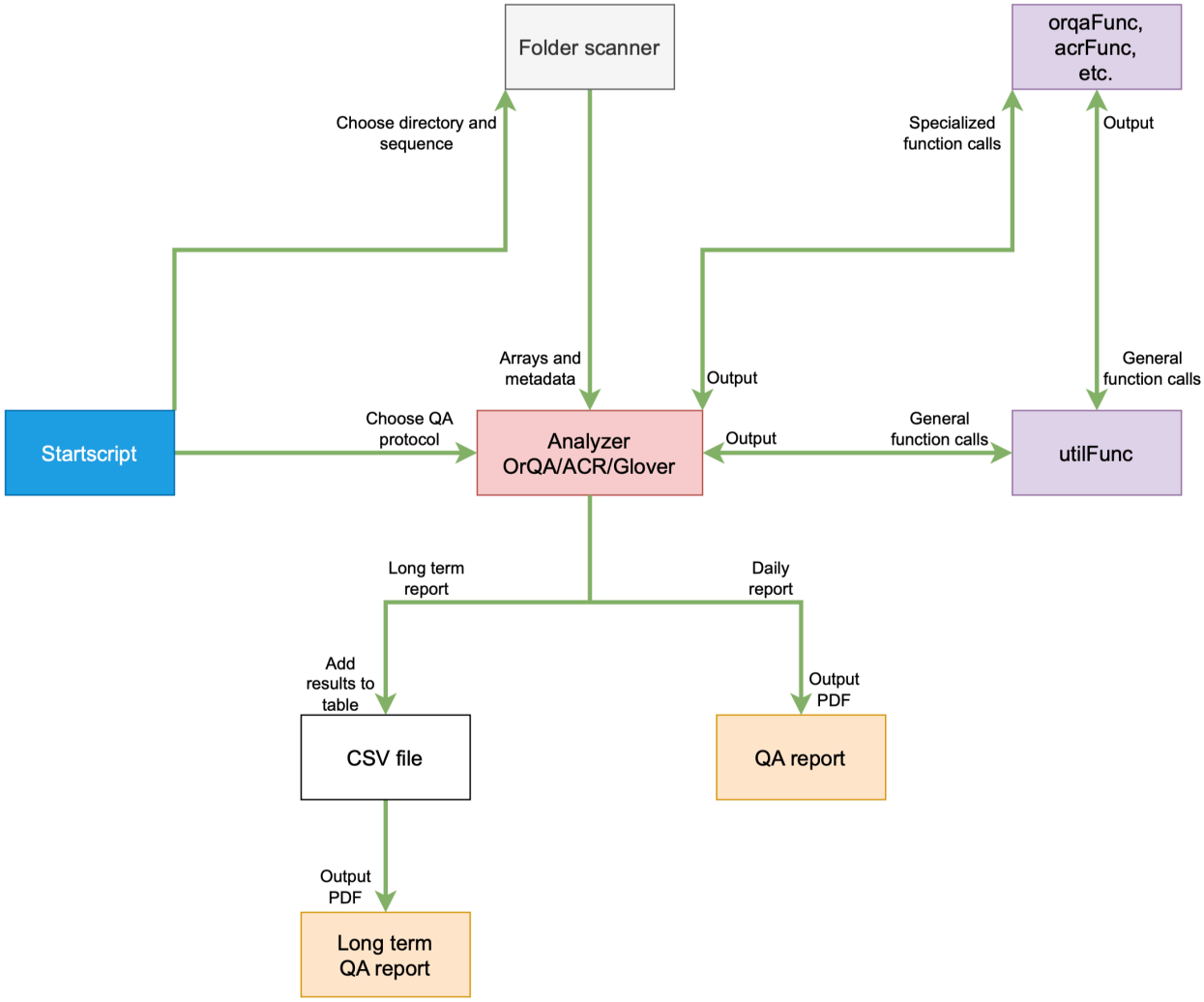
Choose_scan_via_menu() shows menu and operator can choose a scan. With the flag True, the lates scan in the folder will be chosen automatically
Alternatively, dfs.list_scans can be used to show a list of all scans inside a folder and a specific scan be used via dfs.choose_scan("SEQUENCE_NAME").
get_data() creates 4D array (time, slice, y, x) with metadata at dfs.metadata.

Analyzer is initialized by giving it the dfs object and the directory to save everything to. With runall() are all tests run automatically and reports generated using this file structure.



Modifying the software

How the software is written



The start point is marked by Startscript in blue which initializes the Folder Scanner class together with the corresponding Analyzer class. Here the user gives instructions what the program shall do. The Analyzer object performs the selected methods for evaluation of the given images given from the Folder Scanner by performing the different tests and saving both images and results. General image processing functions are stored in utilFunc while test specific functions for a given test are stored separately and called by using orqaFunc, acrFunc, etc. The results are stored as attributes in the Analyzer object and saved as PDF and CSV files on the users PC. One PDF is for the results of the days test in order to verify the programs conclusions while the other PDF is the long term report which stores and shows the test results over time.

What the Analyzer object looks like

Imports

```
1   from .utils.methods import functions as utilfunc
2   from .francis.methods import functions as francisfunc
3
```

Line 1: General image processing functions (thresholding, draw lines, upscaling, etc.)
Line 2: Image processing functions specifically designed for the tests of the given QA protocol (in this case OrQA)

Init

```
18      def __init__(self, data, workdir) -> None:
19          self.imagedata          = data.imagedata[0] if hasattr(data, 'imagedata') else None
20          self.metadata           = data.metadata if hasattr(data, 'metadata') else None
21
```

Load image and metadata form Folder scanner object

```
21
22          self.spacing            = [1,1]
23          # Get correct pixel spacing
24          if self.metadata.get(0x52009230) is not None: # Case Enhanced save
25              self.spacing        = self.metadata[0x52009230][0][0x00289110][0][0x00280030].value
26          if self.metadata.get(0x00280030) is not None: # Case INteroperability
27              self.spacing        = [float(i) for i in self.metadata[0x00280030].value]
```

Get pixel size (multiple ways of accessing it are tried, defaults to 1 if nothing is found)

All test's results are saved here

```
29          self.res_RES                = None  # Resolution
30          self.res_RES_SD             = None  # Resolution SD
31          self.res_GA                 = None  # Geometric lenghts
32          self.res_GA_SD              = None  # Geometric lengths SD
33          self.res_LCOD               = None  # Low contrast
34          self.res_IIU                = None  # Image uniformity
35          self.res_STA                = None  # Slice thickness
36          self.res_SPA                = None  # Slice position
37          self.res_Grid_size          = None  # Grid size
38          self.res_Grid_angle         = None  # Grid angle
39          self.res_Grid_lines_hori    = None  # Detected grid lines horizontal
40          self.res_Grid_lines_vert    = None  # Detected grid lines vertical
41          self.res_Ghosting           = None  # Percent Ghosting Ratio
42
```

```
43          # # Evaluator Part - put in own class at some point
44          self.scannername        = self.metadata[0x00080080].value
45          self.creationdate       = self.metadata[0x00080012].value
46
47          self.workdir            = workdir
48          while True:
49              if os.path.exists(self.workdir):
50                  break
51              else:
52                  print(f"Path to given working directory does not exist ({self.workdir}). Try anew.")
53                  self.workdir = str(input("Type the path of the desired working directory: \n"))
54
55          self._data_organized    = None  # All important data and metadata is stored here to create the corresponding reports.
56
57          self.longtermdata       = {}    # Data from CSV files is stored here.
58
59          # Make folders
60          self.dirs               = {
61              "png"   : os.path.join(workdir, "francis_reports", self.scannername, "imgs", ""),
62              "csv"   : os.path.join(workdir, "francis_reports", self.scannername, ""),
63              "srp"   : os.path.join(workdir, "francis_reports", self.scannername, "single_reports", ""),
64              "lrp"   : os.path.join(workdir, "francis_reports", self.scannername, "")
65          }
66
67          for filetype, dir_to_save_to in self.dirs.items():
68              if not os.path.exists(dir_to_save_to):
69                  os.makedirs(dir_to_save_to, exist_ok=True)
70
```

Information such as scanner name, and dates to create necessary directories (images, csv, short term report, long term report) are stored in self.dirs and created in line 67ff.

self._data_organized stores all metadata needed to create reports to have everything in one big dictionary instead of all over the codebase.

```
368             self._data_organized = {
369                 "Resolution": {
370                     "result": self.res_RES,
371                     "deviation": self.res_RES_SD,
372                     "criteria": {"min": np.round(self.spacing[0]*0.6,1), "max": np.round(self.spacing[0]*1.4,1)},
373                     "unit": "mm",
374                     "image": self.dirs["png"]+"francis_res.png",
375                     "display_range": [0.3,2],
376                     "description": "The module consists of 12 triangles arranged around a center point. The triangles work as
377                 },
378                 "Diameter": {
379                     "result": self.res_GA,
380                     "deviation": self.res_GA_SD,
381                     "criteria": {"min": 144, "max": 152}
```

Every test needs *result, criteria(with "min" and "max"), unit, display range (for long term report)* and *description*. Optional is *deviation* if the result has a deviation (e.g. standard deviation).

Individual evaluation algorithms

In Analyzer object should the algorithm to come to a conclusion be written so that major steps are visible. Smaller steps are done using functions from the helper function objects from the imports. The first boolean flag lets the user decide to show the plot in a window (debugging) while the second saves the figure to the image directory defined by self.dirs. Results are saved into the class attribute (here self.res_STA)

```
237     def thickness(self, showplot=False, savefig=False):
238         img = self.imagedata[6]
239         rect_img = francisfunc.sta.cutoutRect(img)
240 
241         length, coords, border = francisfunc.sta.measureLength(rect_img, self.spacing[1])
242         self.res_STA = np.round(length,1)
243 
244         if showplot or savefig:
245             y_half = rect_img.shape[0]/2
246             plt.vlines(coords[0],ymin=0,ymax=y_half)
247             plt.vlines(coords[1],ymin=0,ymax=y_half)
248             plt.vlines(coords[2],ymin=y_half,ymax=y_half*2, color="red")
249             plt.vlines(coords[3],ymin=y_half,ymax=y_half*2, color="red")
250             plt.imshow(rect_img, cmap="bone")
251             plt.xlabel("Left to right")
252             plt.ylabel("Posterior to anterior")
253             if showplot:
254                 plt.show()
255             if savefig:
256                 plt.savefig(self.dirs["png"]+"francis_thickness.png")
257                 plt.close()
```

Create PDF report of daily QA run

```
494     def create_report(self):
495         combined_results_mapping = self.data_organized
496 
```

This method creates a table with the results of all tests on the first page
This method goes through the keys of self._data_organized and adds an individual page to the report for every test which has a key "image".

```
544         for key, value in combined_results_mapping.items():
545             if value.get("image") is None:
546                 continue
547             image_filename = value["image"]
548             metric_name = key
549             metric_value = value["result"]
550             metric_unit = value["unit"]
551             metric_desc = value["description"]
552 
553             metric_deviation = f'+-{value["deviation"]}' if value.get("deviation") else ""
554 
```

Save and read results using CSVs

All results are saved using add2csv. The CSV is sorted after date and duplicates get automatically removed using Pandas.

```
453     def add2csv(self):
454         # csv header and data
455         savedata = {
456             "Date of measurement":  f"{self.metadata[0x00080020].value}",
457             "Time of measurement":  f"{self.metadata[0x00080031].value}",
458             "Time of evaluation":   f"{datetime.now()}"
459         }
460         for testname, prop in self.data_organized.items():
461             savedata[testname] = prop["result"]
462 
463         csv_filename = self.dirs["csv"] + f'{self.scannername}_francis.csv'
464         write_header = not os.path.isfile(csv_filename)
```

The method create_longterm_report reads the CSV file automatically in and plots the results of each test over time

```
589     def create_longterm_report(self):
590         self._readcsv()
```

The method runall runs all methods needed for a QA run. These methods can also be called seperately in the startscript if preferred. All tests must be executed to create a report otherwise is a test's "result" key in self.data_organized *None* which results in an error when calling the PDF report methods

```
640     def runall(self):
641         self.resolution(False, True)
642         self.low_contrast(False, True)
643         self.uniformity(False, True)
644         self.size(False, True)
645         self.grid(False, True)
646         self.thickness(False, True)
647         self.position(False, True)
648         self.ghosting(False, True)
649 
650         self.add2csv()
651         self.create_report()
652         self.create_longterm_report()
```

4.3.1. Algorithm for the Location of Centerpoints

One common technique used in every test is the creation of a threshold image in order to find the center of the phantom in the images. This is done by first using the function *createThresholdImage* which takes an image array (see for example Figure 21a) and a threshold as arguments. The returned image has binary pixel values which correspond to 0 if the original pixel value was under the threshold and 1 if it was equal or above as shown in Figure 21b. The threshold value is calculated, by using Otsu's method [34]. It calculates the inter class variance for all possible thresholds and returns the maximum inter class variance as the threshold value.

Holes are filled in the threshold image by going row and column wise through the image and filling every gap between the most outer pixels with value one which can be seen in Figure 21c. With the filled threshold image available, the center is determined by calculating the "center of mass" of the image. The resulting center point is shown in Figure 21d

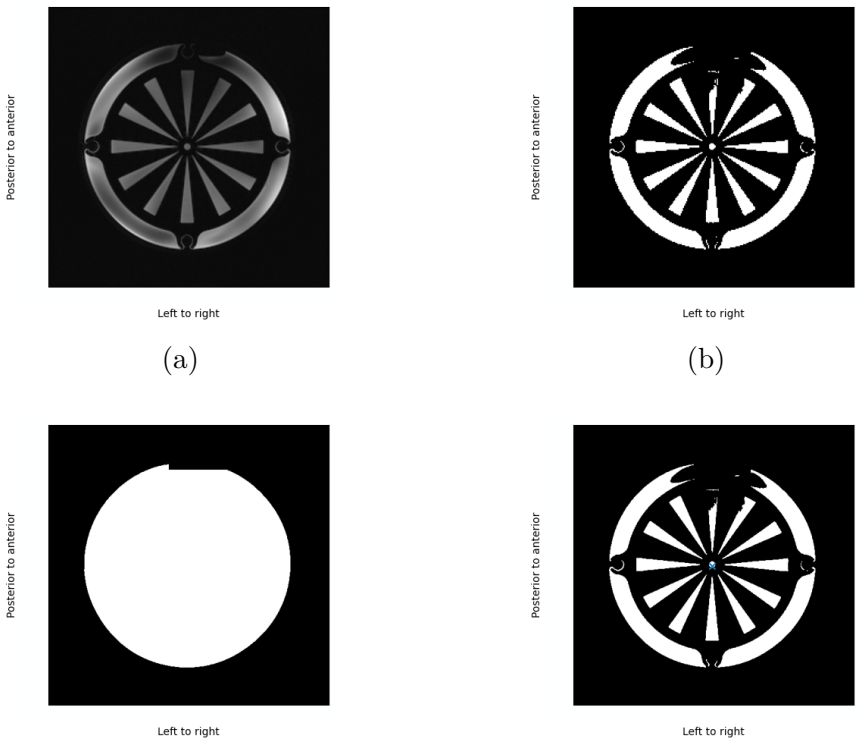


Figure 21: (a) image of an example slice. (b) Binary image of same slice using Otsu's method to get a threshold. (c) binary image taken for Center of Mass calculation with its holes filled. (d) the determined centerpoint/center of mass is shown as a blue cross in the middle of the phantom.

4.3.2. Diameter Algorithm (OrQA, ACR)

The slice containing the Resolution Module (See the module at Figure 11a) is chosen and a centerpoint determined. Because the slice has a high contrast point in the center (see Figure 11b) a small circular mask is drawn around the first centerpoint which only includes the bright spot and the dark area around it. The calculation described in Chapter 4.3.1 is repeated to have an even more exact center point determined.

Two lines at 45° and 135° are drawn from the centerpoint and put over the threshold image. The length is then determined by taking coordinates of the first and the last pixel with value 1 on the line and calculating the length by using Pythagorean's Theorem taking the conversion factors from pixel to mm in both x and y direction into account.

The mean length between the two calculated lengths is returned and Figure 22b included in the report.

(a) (b)

Figure 22: Demonstration for the diameter measuring algorithm. The centerpoint is determined and by using image (a) as a basis two lines are drawn in 45° and 135° . The most outer points on the lines which are still on the phantom are taken the distance between them calculated.

4.3.3. Slice Position Algorithm (OrQA, ACR)

The slice containing the Slice Position/Thickness Module (see Figure 7a) is chosen and the center determined. The threshold image created in the initial step is taken and the radius of the phantom measured as described in Chapter 4.3.2 . Using the measured diameter and centerpoint as a measure of the phantoms position and size relative to the image, a cutout of the upper high part of the phantom is performed so that only the ramps and background is visible as seen in Figure 23. This cutout gets binarized using Otsu's method. The length of the two ramps is measured by dividing the cutout in a left and right half. For each column, the sum of pixel values is calculated and the median result for both the left and right half is taken as the result of the measured length. The difference is calculated with the equation given in the ACR manual [1] which gives a measure of the positional offset:

$$Position[\text{mm}] = Length_{\text{left}}[\text{mm}] - Length_{\text{right}}[\text{mm}] \tag{12}$$

The actual slice position offset is received by dividing the result from Equation 12 by 2. The result is then returned and Figure 23 included in the report.

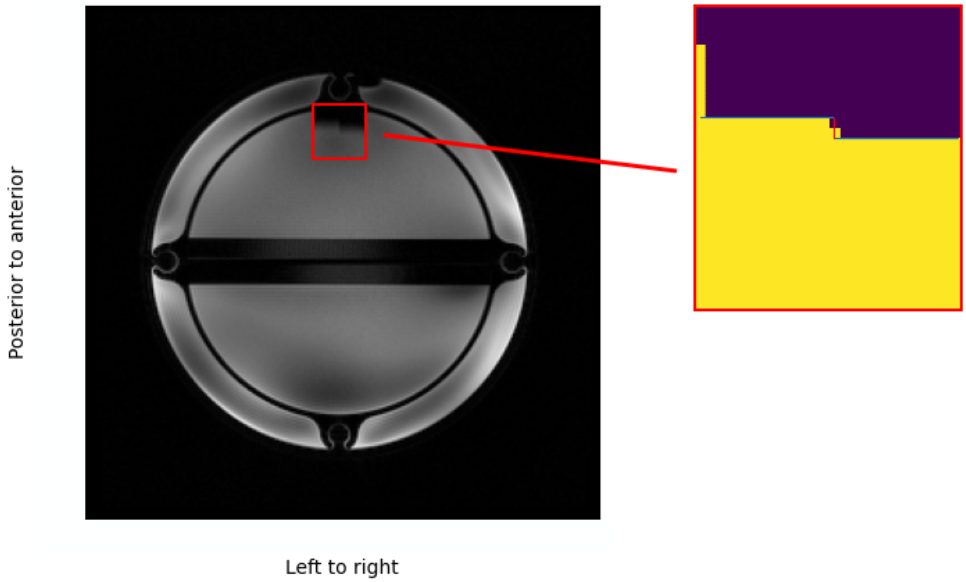


Figure 23: Demonstration of the Slice Position evaluation. A cutout on the shown rectangular region of interest is made based on the center position and diameter of the slice. The cutout image is thresholded using Otsu's method. The difference in length is calculated by computing the pixel value sum along the columns and taking the median value of the right side and the median value of the left side as the measured length. The resulting length of both the left and right side is shown the small blue lines in the right-hand image at the borders.

4.3.4. Slice Thickness Algorithm (OrQA, ACR)

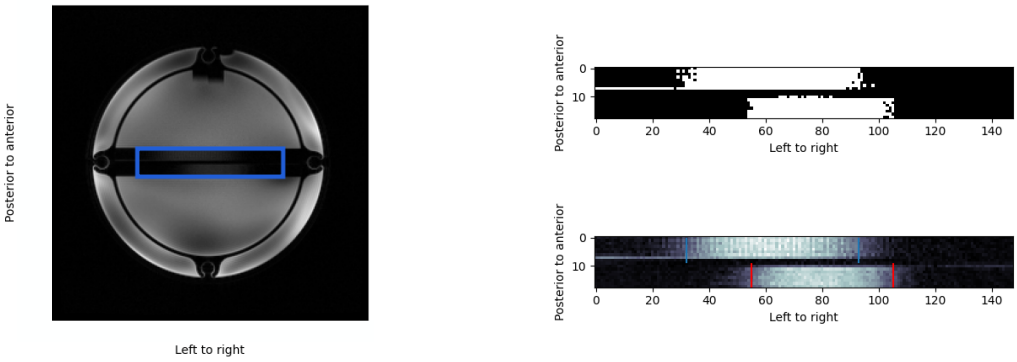
The slice containing the Slice Position/Thickness Module (see Figure 7a) is chosen and the center determined. Using the center coordinates, the phantom's diameter is measured as mentioned in Chapter 4.3.2. With the centerpoint and diameter, relative coordinates are used to create a cutout of the center rectangle as seen in Figure 24a. A separate threshold image is created from the cutout (See the top image at Figure 24b). Next an array of the sum along the rows is calculated and convolved with the array [1,-1] to find the point of the highest change to separate the upper from the lower ramp. Once done, the length of a ramp is measured by determining the length of the line until its width is lower than half of its maximum width as seen in Figure 24b. The measured length is converted into the corresponding slice thickness by use of the following equation for the OrQA Protocol:

$$\text{Thickness(length)[mm]} = 0.125 \cdot \text{length[mm]} - 1\text{[mm]} \tag{13}$$

See Chapter A.5 in the appendix for an explanation of this equation. For the ACR Protocol, the equation from the manual [1] is used

$$\text{Thickness[mm]} = 0.2 \cdot \frac{\text{top[mm]} \cdot \text{bottom[mm]}}{\text{top[mm]} + \text{bottom[mm]}} \tag{14}$$

The result is returned and Figure 24b is included in the report.



(a) (b)

Figure 24: Demonstration of the Slice Thickness evaluation. (a) shows the region of interest which is cut out into a separate array. The processing of this array is demonstrated in (b) where the array gets thresholded to separate foreground from background and the lengths of both lines separately measured and the result of the measurement shown in the blue and red lines.

4.3.5. Image Uniformity Algorithm (OrQA, ACR)

The slice containing the Uniformity/Ghosting Module (See Figure 9a) is chosen and the center determined. The whole image gets convolved with a rectangle filter with a square-shaped kernel of size $1cm^2$ which shall emulate the criteria of finding $1cm^2$ big areas of high and low signal required by the ACR Protocol [1]. The exact kernel size in pixels is determined by dividing one centimeter by the pixel spacing to retrieve the amount of pixels. A circular image mask is created whose size is 70% of the phantom's radius. Next the highest and lowest point of the convolved image is determined for the calculation of the Image Intensity Uniformity via the equation from the ACR manual [1]

$$\text{IIU} = 100 \cdot (1 - (\frac{\text{high - low}}{\text{low + high}})) \tag{15}$$

The result is returned and Figure 25 shows the output image which is included in the report.

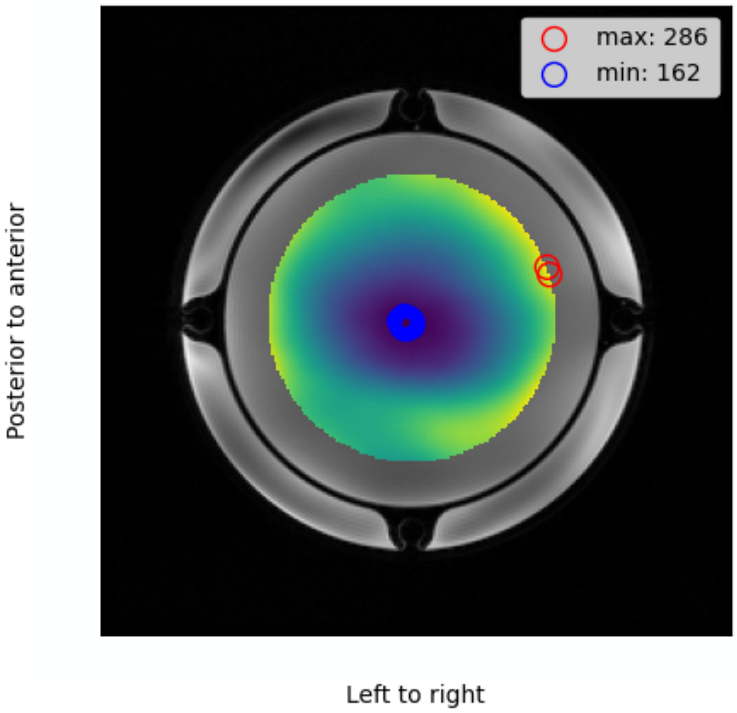


Figure 25: Demonstration of the Uniformity evaluation algorithm. The original image of the slice is overlaid with the convolved image of the slice in the relevant ROI. On this ROI, high- and lowpoints are marked with circles and the exact values given in the legend.

4.3.6. Ghosting Algorithm (OrQA, ACR)

The slice containing the Image Intensity Uniformity/Ghosting Module (see Figure 9a) is chosen and the center determined. The same area as in the Image Intensity Uniformity Test (See Chapter 4.3.5) is masked out in addition to four ellipses top, bottom, left and right of the phantom as shown in Figure 26. The size of the ellipses is determined by measuring the distance from the phantom to the four edges of the image to prevent overdraw.

From all areas is the average value taken in order to calculate the Percent Ghosting Ratio according to the equation from the ACR manual [1]

$$\mathrm{PGR} = \frac{|(\mathrm{meanTop} + \mathrm{meanBot}) - (\mathrm{meanLeft} + \mathrm{meanRight})|}{\mathrm{meanCenter}} \tag{16}$$

The value is then returned and Figure 26 included in the report.

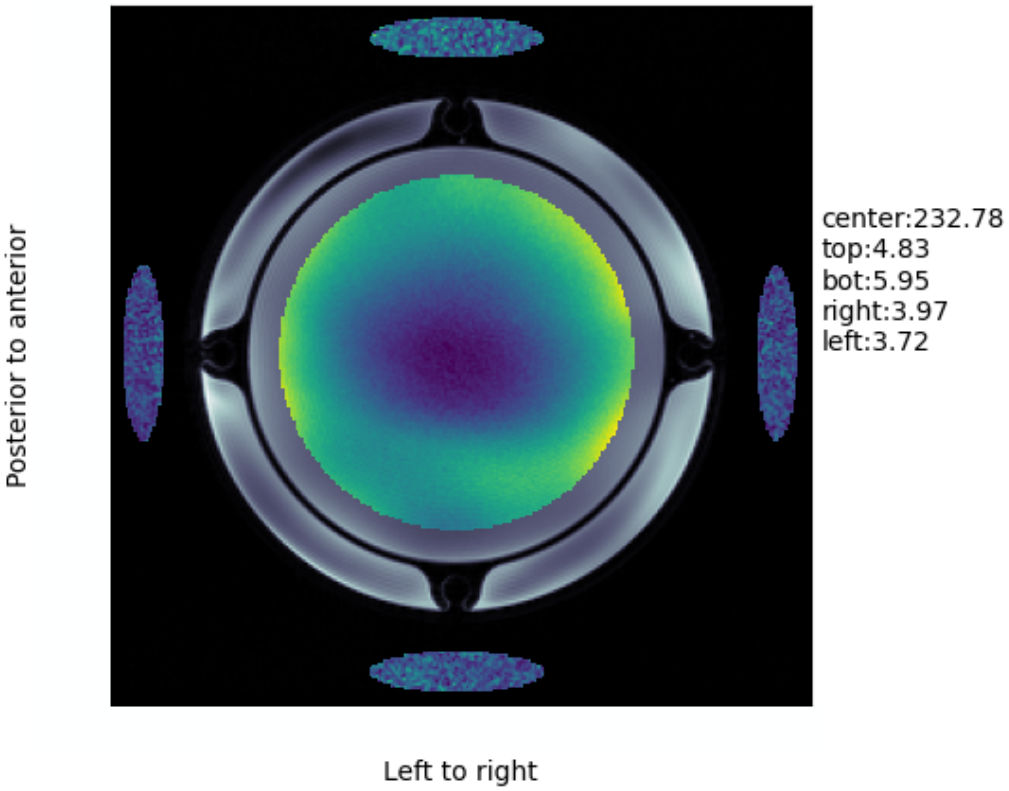
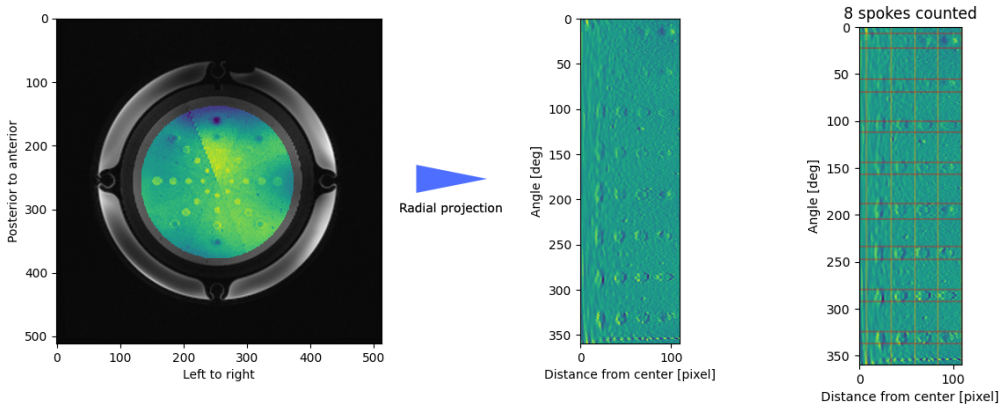


Figure 26: Demonstration of the Ghosting evaluation algorithm. The necessary regions of interest are drawn and shown as an overlay onto the original grayscale image. The average values are shown on the right-hand side.

4.3.7. Low Contrast Object Detectability Algorithm (OrQA, ACR)

While the test takes place on four different slices on ACR the same test taken with OrQA takes only one slice, but the algorithm is the same. First the image resolution is doubled by upscaling to ensure better results in the following steps. The middle of the center circle is determined and a transformation of the image from Cartesian in polar coordinates is performed with the distance of a point from the origin on the x-axis and the angle on the y-axis. This procedure is shown in Figure 27a. The resulting array is then convolved along the r-axis with an edge detecting kernel to filter out low frequency components in the signal.

With *a priori* knowledge of the angles and distances where each circle is placed at, four zones are defined for each spoke's circles which is seen in the rectangles enclosed by the red and orange lines in figure 27b. The spoke detection then relies on defining a threshold with the mean value over all pixel values plus a multiple of the standard deviation. A spoke counts as detected if at a given spoke-angle-range, all four zones have an image value surpassing this threshold. This method counts on the edges having a value larger than the background noise. The total amount of counted spokes is returned with Figure 27a and 27b included in the report.

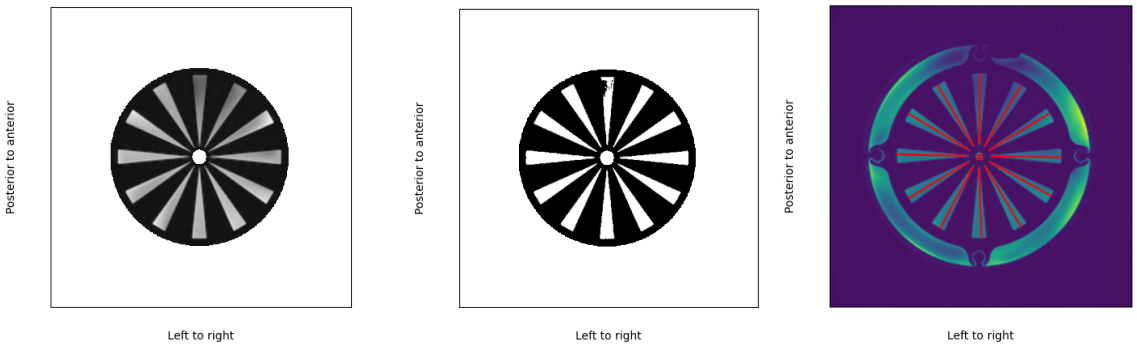


(a) (b)

Figure 27: Demonstration of the Low Contrast evaluation. (a) the left-hand side shows the slide in grayscale for reference with a circular region of interest shown on top. The right-hand side shows a radial projection of the region of interest. The X axis represents the distance from the center whilst the Y axis shows the angle starting left from the spoke at 12 o'clock. The array gets then convolved with an edge filter along the radius direction, resulting in the final array. (b) shows the detection step in which the detection of spokes is done by detecting values over and under a certain threshold for each spoke. The orange lines mark the radii where each circle is and the red lines at angle of the spokes. This results in four zones for each spoke in which each circle is enclosed by the red and orange lines. The red lines act also as indicators for identified spokes.

4.3.8. Resolution Algorithm (OrQA)

The slice containing the Resolution Module (See Figure 11a) is chosen and a centerpoint determined. Because the slice has a high contrast point in the center (see Figure 11b) a small circular mask is drawn around the first centerpoint which only includes the bright spot and the dark area around it. The centerpoint detection described in Chapter 4.3 is repeated to find an even more exact centerpoint. By measuring the diameter as described in Chapter 4.3.2, two circular masks are created to cut out the outer part of the phantom and the middle section (see Figure 28a) to find another threshold with just this part of the image and to create the corresponding threshold image as seen in Figure 28b.



(a) (b) (c)

Figure 28: Demonstration of Resolution evaluation. (a) shows the given slice with a donut shaped mask which cuts out the middle and outer part of the module in order to perform thresholding only with the data in the region of interest. (b) shows the masked array which got thresholded. (c) shows the resulting lines for the Resolution Test. It shows the slice image with a red cross in the middle marking the centerpoint with red lines in every triangle marking the line which was measured as longest according to the test protocol. To make the reading easier was the colormap *viridis* used.

From the center point, the angle range of every triangle is considered *a priori* knowledge. For each triangle, several lines in the angle range are drawn. The coordinates of the first and last point which contains a 1 are taken to measure the length of the triangle via Pythagoras' Theorem. The longest distance l taken as the length of the triangle. The median length of all triangles' height is taken as the result. The resolution which described the minimum width of the triangles walls before no usable contrast is detected anymore is calculated by

$$\text{Resolution}(l)[mm] = (5[mm] - \frac{l[mm]}{10}) * 2 \tag{17}$$

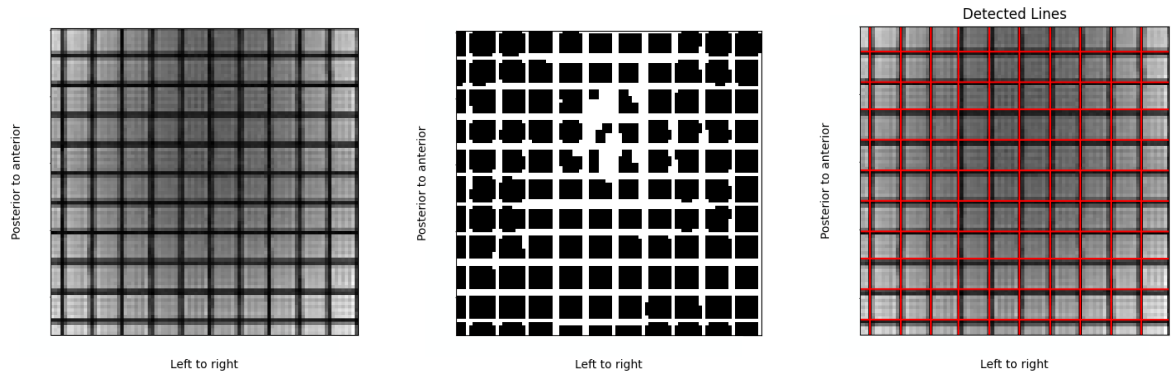
See the full explanation for Equation 17 in the appendix at Chapter A.6 . All lines taken for measuring the triangles' height are shown in Figure 28c. The result of Equation 17 is then returned and Figure 28c included in the report.

4.3.9. Grid Algorithm (OrQA)

The slice containing the Grid Module (see Figure 6a) is chosen and the center determined. A square cutout in the middle is performed to get an image with just grid lines which is shown in Figure 29a. In order to determine the grid lines the image is first binarized using Otsu's Method which is shown in Figure 29b. The lines get smoothed using one pass of dilatation and erosion using a 3x3 kernel.

The grid lines themselves are located using Hough Transformation for which a rudimentary explanation is provided in the appendix in Chapter A.2 and an even more detailed one at [35]. The results are categorized into horizontal and vertical lines by taking all lines at an angle of -20 and 20 degrees as the range for horizontal lines and angles of 70 to 110 degrees for vertical lines. The angle group of lines with the most detections is taken as the detection for the horizontal/vertical lines and gets visualized as seen in Figure 29c.

The angle between them is calculated to test perpendicularity and the median distance between the horizontal and vertical lines taken to measure the grid size. The amount of lines detected is also saved to detect potential large differences over time and Figure 29c included in the report.



(a) (b) (c)

Figure 29: Demonstration of Grid evaluation. (a) shows the region of interest used from the slice. (b) shows the given ROI which is then thresholded with Otsu's method, inverted to have the grid lines set as 1 and processed by performing one step of closing which removes rough edges. (c) detected lines found by the Hough transformation are drawn in red.

4.3.10. Pass Criteria for OrQA Protocol

The criteria to mark the proposed tests as passed or failed are self-determined or related to the ACR Protocol (see the manual at [1]). The criteria for passing the individual tests is shown and explained in table 1 .

Test	Pass criteria	Origin of criteria
Diameter	148mm ± 4mm	ACR with larger margin
Resolution	Theoretical resolution ± 40%	Self-derived
Low Contrast	Min 6 spokes	Self-derived
Image Uniformity	Min 70%	ACR with larger margin
Slice Thickness	5mm ± 1mm	From ACR
Slice Position	0mm ± 3mm	ACR with smaller margin
Grid Size	36mm2 ± 8mm2	Self-derived
Grid line count	10 ± 3	Self-derived
Ghosting	Max 5%	From ACR

Table 1: Table with all criteria used to decide of given test is passed or failed. The ACR criteria can be found at [1].

4.3.11. Creation of Reports

The program outputs after each run of the program two PDF reports for the daily inspection and one for the long term view.

The first report opens with a table of all performed tests, the criteria, the measured result and a pass/fail field to allow for a quick check of each test. To ensure the proper functionality of the program and to check for anomalies during the evaluation steps, the images shown in the previous chapters are included on the following pages together with an explanation of the QA test in question. The long-term report shows the multiple graphs with the test results over time to allow for a long term view of the system.

See further information about the folder structure in the appendix at A.4 .