Quantum state tomography on spatial modes with intense light: concept and implementation (NOTES)

Ermes Toninelli   Bienvenu Ndagano   Adam Vallés
Bereneice Sephton   Isaac Nape   Antonio Ambrosio
Federico Capasso   Miles J. Padgett   Andrew Forbes

August 31, 2018

In these notes we describe the software required to automate our DIY roto-flip stages: an Arduino firmware loaded to the microcontroller of each stage, and an accompanying LabVIEW automation programme, which allows to sequentially control the connected stages and the CCD measuring detector. Finally, we also discuss an optimiser programme, which allows to drastically reduce the time required to align the optical channel, by iteratively scanning the best position of the central pixel of the CCD camera, from which the light intensity is measured in each projection measurement.

1 Firmware for the Arduino microcontroller

At the heart of an Arduino Nano is the *ATmega328P* microchip, which can be programmed using the freely available Arduino integrated development environment (IDE) software. Here is where code can be written that links input signals from either the analogue or digital physical pins on the board to the output pins, allowing to process an input (from a linked component, such as a switch, a proximity sensor, or a message from the serial port, etc.) and to produce a certain output (to a linked component, such as a light-bulb, a speaker, or a motor, etc.). In this experiment the input that the microcontrollers of each of the four roto-flip stages need to process is a message sent from LabVIEW via the serial port, and the required output is either rotation of the polarisation optics (i.e. the actuation of the stepper motor), or the pivoting of the stage to move it out of the optical path (i.e. the actuation of the servo motor). The flow diagram of the firmware loaded onto the roto-flip stages is shown in Fig. 1. Accordingly, four states are defined for the operation of the roto-flip stage:

1. *Initialisation.* This state is comprised of code that is run only once, as soon as the stage is powered on, allowing the microcontroller to drive the servo and to start a bidirectional communication session via serial (over the mini-USB port), as well as to define whether each of the required physical pins on the board are to be used send (i.e. an output) or receive (i.e. an input) a voltage signal;

2. *Handshake with LabVIEW.* This state is used to sync the stage to the commands received from the LabVIEW automation programme, ensuring that LabVIEW waits for the stage to have completed its motion before sending another command, and when the stage receives a command from LabVIEW via the serial port it parses it and if it is an enquiry about its readiness it answers back that it is ready, however, if the received message is different then the stage parses it by looking at the first letter of the string;

3. *Rotation.* This state is triggered if the first character of the string received from LabVIEW is either an m or a p (which indicate a positive or negative number of steps, i.e. clockwise or counter-clockwise rotation), and the *turns*-function drives current to the coils of the stepper motor satisfying the required number of steps;

4. *Pivoting.* This state is triggered if the first character of the string received from LabVIEW is am s, causing the microcontroller to attach the servo and drive its motor until it has turned by the desired amount (i.e. until the resistance of the potentiometer fitted inside the servo motor matches a desired value).

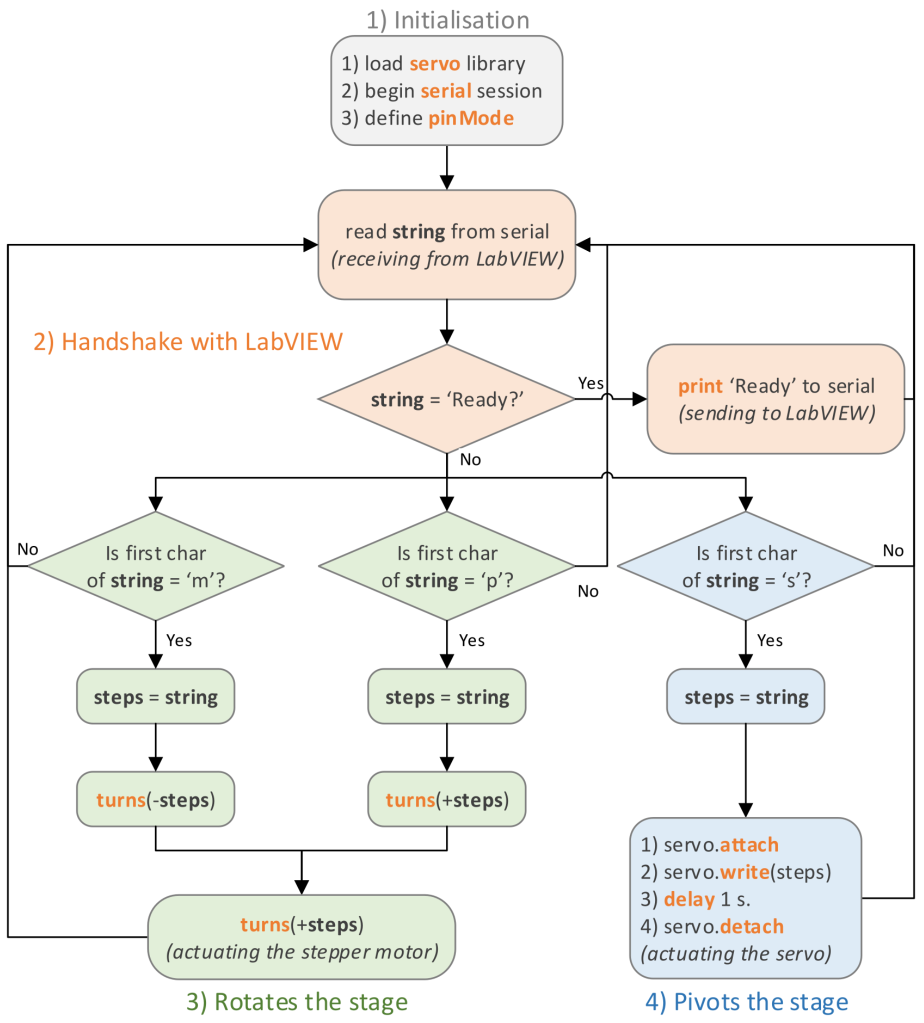


Figure 1: Flow-diagram of the firmware for the Arduino microcontroller. The firmware programme described by pseudocode in the flow-diagram is used to control each of the four roto-flip stages used in our experiment. Accordingly, the operation of a roto-flip stage is defined by four states: 1) initialisation; 2) handshake with LabVIEW; 3) rotation; and 4) pivoting. For additional convenience, the bold orange and black words match the names of variables and functions used in the Arduino code.

2 Automation of the experiment via LabVIEW

LabVIEW was chosen as the programming language for the automation of the experiment. However, any programming language able to communicate with the serial port and a CCD detector (or a photo-diode) may be used instead. The LabVIEW automation programme was coded according to the state-machine software architecture, allowing to separately handle initialisation, rotation, pivoting, and synchronisation of the four connected roto-flip mounts, as well as data acquisition and saving. Specifically, the COM ports corresponding to the attached microcontrollers of the stages can be selected by the user, as shown in the initialisation pane in Fig. 2.

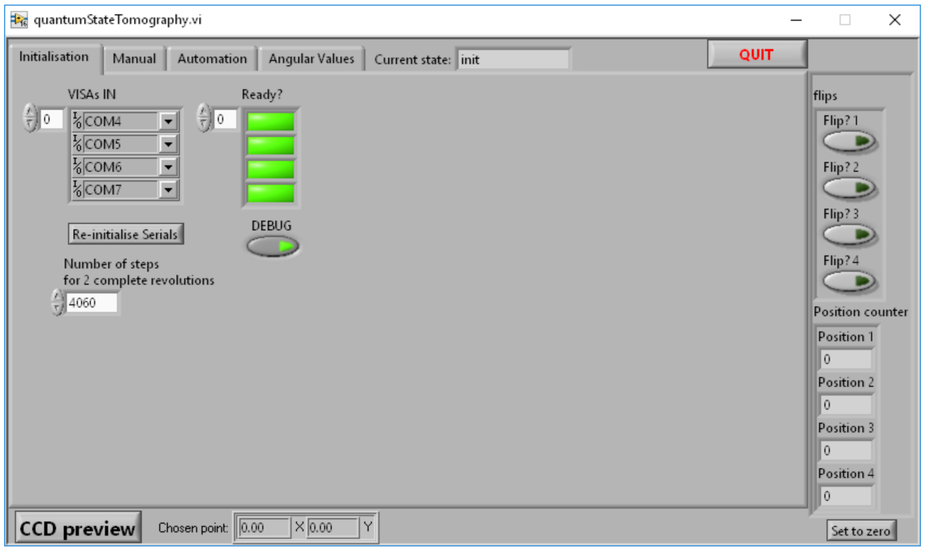
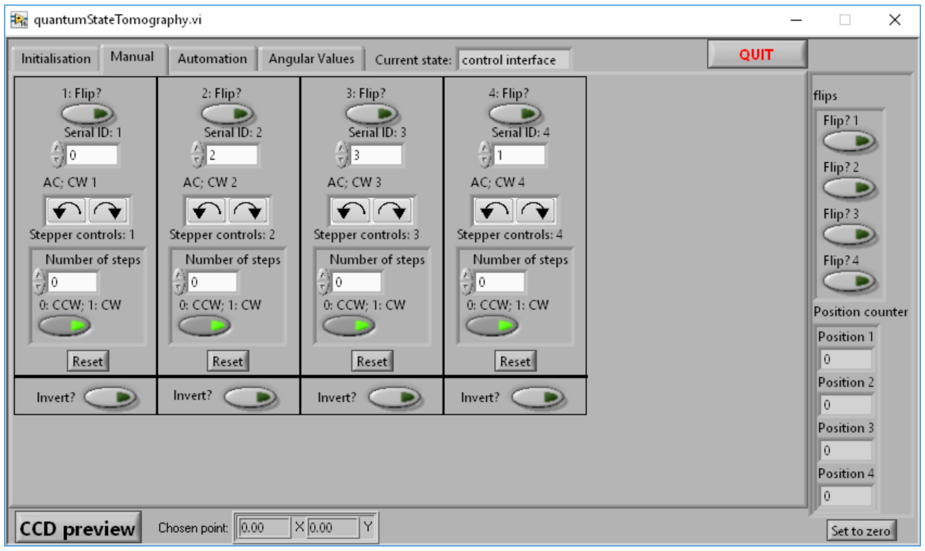


Figure 2: Screen-shot of the *Initialisation* tab of the LabVIEW programme.

Past initialisation, the control programme allows for both manual operation of the stages (used during alignment and testing and shown in Fig. 3), and an auto-mode (shown in Fig. 4, in which the full state-tomography of a particular input mode is performed and frames from the CCD camera are stored to disks.

In auto-mode the user is required to input the desired number of angular arrangements of the polarisation optics, in the form of an array of values representing the angular positions of each motorised roto-flip stage, as shown in Fig. 5.

The other required input from the user is the selection of the central pixel from a preview video-stream acquired with the CCD camera, by clicking on the bottom-left button labelled *'CCD preview'* in the front-panel of the LabVIEW programme. The automated measurement can be started by clicking on the *'Automate'* button of the automation tab, visible in Fig. 4. An automated tomography measurement can be observed in video that accompanies this pub-



Current state: control interface

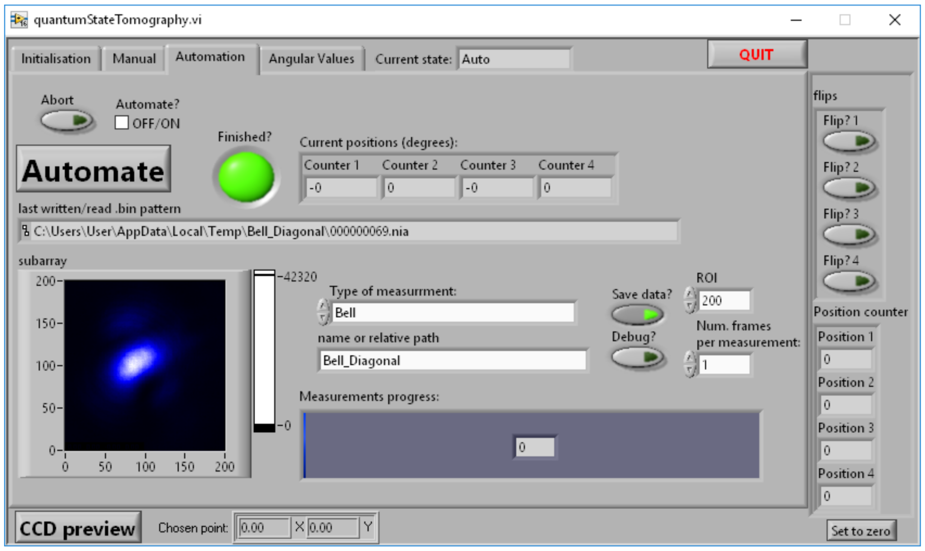
Serial ID: 1 — 0
Serial ID: 2 — 2
Serial ID: 3 — 3
Serial ID: 4 — 1

Number of steps: 0 (0: CCW; 1: CW) — all four controls

Position 1: 0
Position 2: 0
Position 3: 0
Position 4: 0

Chosen point: 0.00 X 0.00 Y

Figure 3: Screen-shot of the *Manual* tab of the LabVIEW programme.



Current state: Auto

Automate? OFF/ON

Current positions (degrees):

Counter 1	Counter 2	Counter 3	Counter 4
-0	0	-0	0

last written/read .bin pattern: C:\Users\User\AppData\Local\Temp\Bell_Diagonal\000000069.nia

Type of measurment: Bell

name or relative path: Bell_Diagonal

ROI: 200

Num. frames per measurement: 1

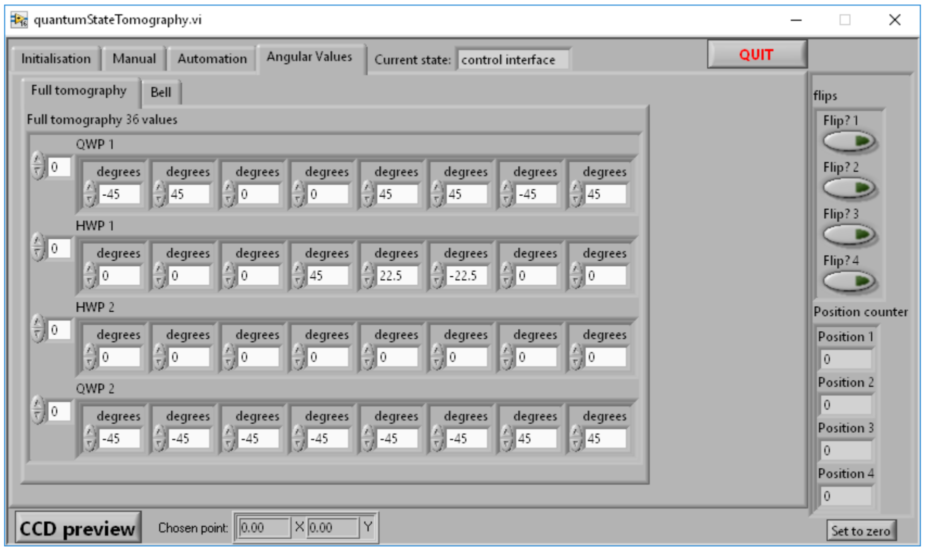
Measurements progress: 0

Position 1: 0
Position 2: 0
Position 3: 0
Position 4: 0

Chosen point: 0.00 X 0.00 Y

Figure 4: Screen-shot of the *Automation* tab of the LabVIEW programme.

lication, accessible at the following link[1]. Tomography results can be obtained by processing the acquired data. Once the results have been retrieved it is possible to set another input state (using the polarisation optics responsible for the state generation) and run the automated tomography measurement again.



Current state: control interface

Full tomography 36 values

		degrees	degrees	degrees	degrees	degrees	degrees	degrees	degrees
QWP 1	0	-45	45	0	0	45	45	-45	45
HWP 1	0	0	0	0	45	22.5	-22.5	0	0
HWP 2	0	0	0	0	0	0	0	0	0
QWP 2	0	-45	-45	-45	-45	-45	-45	45	45

Flip? 1
Flip? 2
Flip? 3
Flip? 4

Position counter

Position 1: 0
Position 2: 0
Position 3: 0
Position 4: 0

Chosen point: 0.00 X 0.00 Y

Figure 5: Screen-shot of the *Angular Values* tab of the LabVIEW programme.

The main functions of the LabVIEW automation programme are shown in Fig. 6. In the initialisation stage (labelled ***'init.'*** in Fig. 6), the programme connects to the roto-flip stages, by performing a hand-shake with the in-built Arduino microcontroller. The next stage of the programme is a user interface (labelled ***'Control interface'*** in Fig. 6) through which the roto-flip stages can be manually controlled and the CCD camera can be operated in preview mode (labelled ***'CCD Preview'*** in Fig. 6), from which the pixel-coordinates for the projection measurements can be chosen. From the user interface it is also possible to start the automated measurements of either a Bell or a state tomography experiment. By choosing an automated measurement the system iterates through user-defined angular arrangements of the polarisation optics, saving to disk the data acquired from the CCD camera at each step, to allow further off-line processing. Each angular configuration involves the positioning (and potentially the pivoting) of all four roto-flip stages. For this reason, during an automated run of the system, the programme iterates through four states corresponding to rotation (labelled ***'Roto (1-4)'*** in Fig. 6) and four states for pivoting (labelled ***'Flip 1-4'*** in Fig. 6). Once the optics has been positioned, the system acquires and saves to disk a user-defined number of frames from the CCD camera, via the corresponding state (labelled ***'Measurement'*** in Fig. 6).

A state-machine software architecture allows for more robust code, that can better deal with user-defined input parameters and interaction. Accordingly, the programme flow is redirected to the appropriate state, following the input from the user, as opposed to unidirectionally execute from start to finish. In this way, it is possible to split the programme and the code based on specific

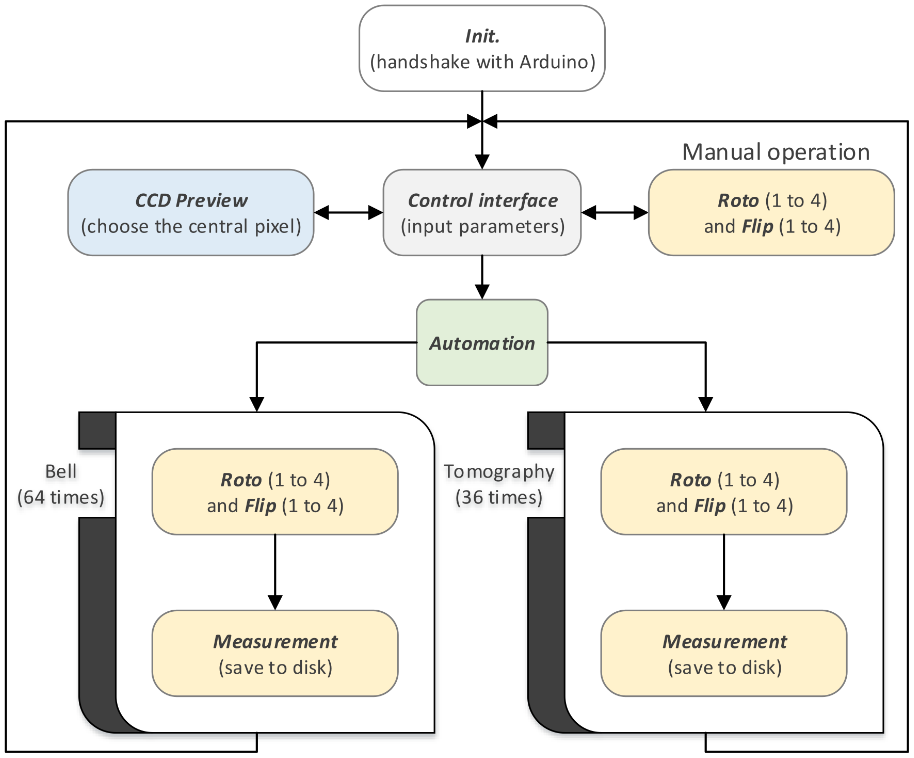


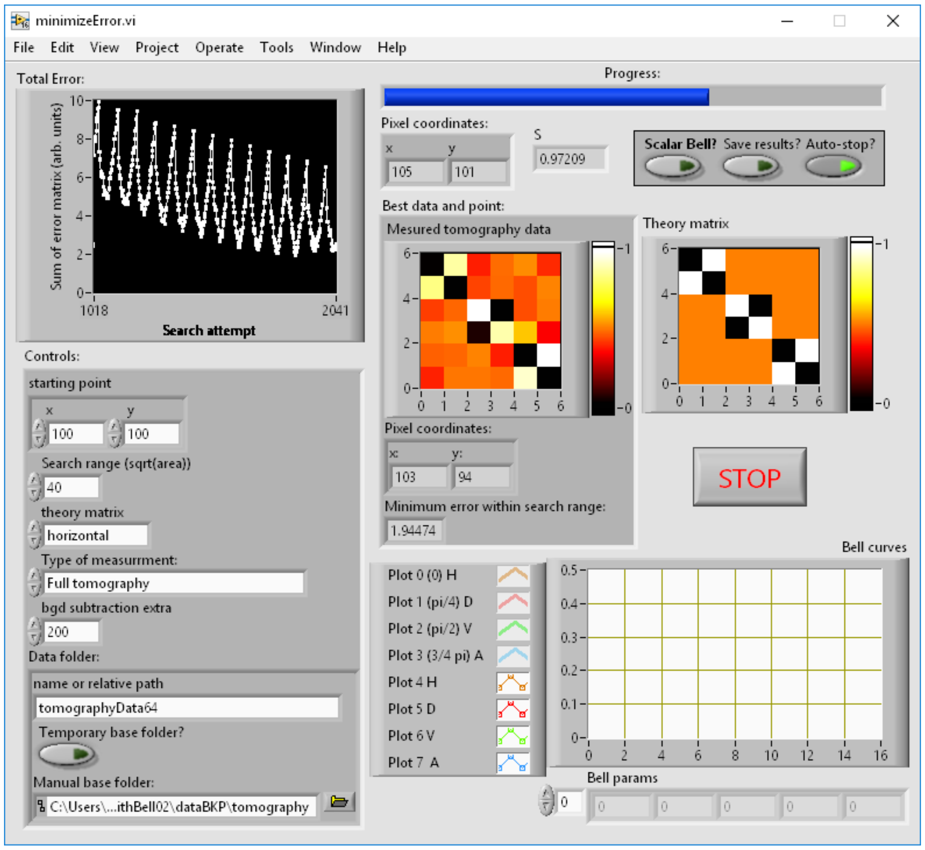
Figure 6: Flow-diagram of the LabVIEW control programme.

functionality (i.e. the states of the state-machine), simplifying potential issues about hardware resources and synchronisation. For example, the if the roto-flip stages are being controlled manually by the user (i.e. by the 'Manual operation' state) there is no danger of inadvertently starting an automated measurement -which would also request the use of the roto-flip stages- as the two states are separate by design: each state that relies either on the CCD or on the roto-flip stages hardware resources can only be operated once the state machine is waiting idle at the 'User interface' state.

3 Optimiser code with a known input state

Having a spatially resolved detector to measure the transmitted light intensity for each projection measurement can provide extra flexibility in the alignment of the optical channel. In the case of a spatially unresolved bucket-detector it is necessary to scan the detection plane by moving the detector one position at the time for each full set of measurements. However, if a CCD camera is available, it is instead possible to acquire a set of frames for a certain alignment of the optics and find the best position for the projection measurement via software, by scanning the choice of the central pixel coordinates from which the intensity measurements are retrieved. This approach requires the previous knowledge of the what the input state is, so that an *error-matrix* and a subsequent total-error metric may be computed by calculating the absolute value of the difference between the measured tomography-data matrix and the theory-matrix of the known input-state. The pixel-coordinates that return the minimum total-error correspond to the best position of the detector for the projection measurement. Once this position has been determined by running the optimiser-programme it is possible to change the input-state to a different one and run a state-tomography measurement using the optimised central pixel coordinate, as found by the previous run. It should be mentioned that this approach is only possible when using intense classical beams of light, which are detectable with enough precision by a spatially-resolved camera detector.

The optimiser programme requires a number of inputs from the user, as shown in the left-pane of the graphical-user-interface reported in Fig. 7. Below



Pixel coordinates: x 105 y 101

S 0.97209

Scalar Bell? Save results? Auto-stop?

Controls:

starting point: x 100 y 100

Search range (sqrt(area)) 40

theory matrix horizontal

Type of measurment: Full tomography

bgd subtraction extra 200

Data folder:

name or relative path tomographyData64

Temporary base folder?

Manual base folder: C:\Users\...ithBell02\dataBKP\tomography

Pixel coordinates: x: 103 y: 94

Minimum error within search range: 1.94474

Bell params 0 | 0 0 0 0 0

Figure 7: Screen-shot of the optimiser LabVIEW programme.

we describe three perhaps not self-explanatory parameters user-defined parameters of the optimiser programme:

1. *Theory matrix.* This is a user-defined two-dimensional array of values that represent what the result of the state-tomography measurement should look like based on a known input-state. This ground-truth will be used in the optimisation problem of finding the pair of pixel-coordinates that generate a set of tomography results which differ the least from the known set (i.e. from the user-defined theory matrix).

2. *Starting point.* This is a pair of pixel coordinates which specify the centre of a square region-of-interest (ROI), within which tomography results will be computed for each point. As mentioned, the ability to fine-tune the position of the detector after the measurement, allows for a quicker set-up of the experiment, while ensuring that the point chosen for the projection measurement (i.e. the output pixel-coordinates found by the optimiser programme) is *the* best point, within experimental error.

3. *Search range.* This specifies the length of the side of the square ROI within which the best set of coordinates for the projection measurement is searched. If for example during the alignment of the system, the centroid of a Gaussian-shaped projection was estimated to be roughly at the centre of the field of view of the camera, the user can then input a search-range of a few tens of pixels.

A full run of the optimiser programme produces a number of outputs, as visible in Fig. 7. Below we describe a few that are particularly useful in interpreting the results:

1. *'Total Error' chart.* This chart is updated point-by-point for every tested pixel of the user-defined ROI, as the programme computes the absolute value of the difference between the theory matrix and the matrix of tomography data, as produced by the current choice of pixel. For a well-aligned system, the total error can be < 1.

2. *'Best data and point' cluster.* This is the main output of the optimiser programme, containing the pixel-coordinates and a representation of the associated tomography data that best resembles the theory matrix. The minimum error is also provided (which is equal to the absolute value of the difference between the theory matrix and the measured tomography data for a certain choice of pixel coordinates). Using this last metric, it is possible to make adjustments to the optical setup and monitoring in an informed manner how this changes affect the fidelity of the results.

3. *'Theory matrix' intensity graph.* This intensity graph simply displays back to the user the theory matrix.

4. *'Bell Curves' graph and 'Bell params' array.* If the programme is used to analyse the data of a Bell measurement (as selectable in the 'Controls:'

input cluster and in the top-right boolean switch in the case of a Bell measurement for a scalar input-state) the programme will produce a set of Bell curves and associated Bell parameters.

The flow-chart of our optimiser programme is shown in Fig. 8. In the first and second highlighted blocks the user-defined input parameters determine set-up the optimisation run. The user selects whether the data to be retrieved from disk represents a tomography or a Bell measurement and the frames are loaded to RAM for fast iterative processing. The third block of the diagram represents a for-loop, according to which data is analysed for every pixel within the user-defined ROI. The fourth block represents the extraction of the pixel intensities from the current search pixel-coordinates, which are then used to compute the total error metric. Finally, once the programme has computed the experimental tomography data for every point within the search range, the optimal pixel coordinates that correspond to the best tomography data results are returned to the user.

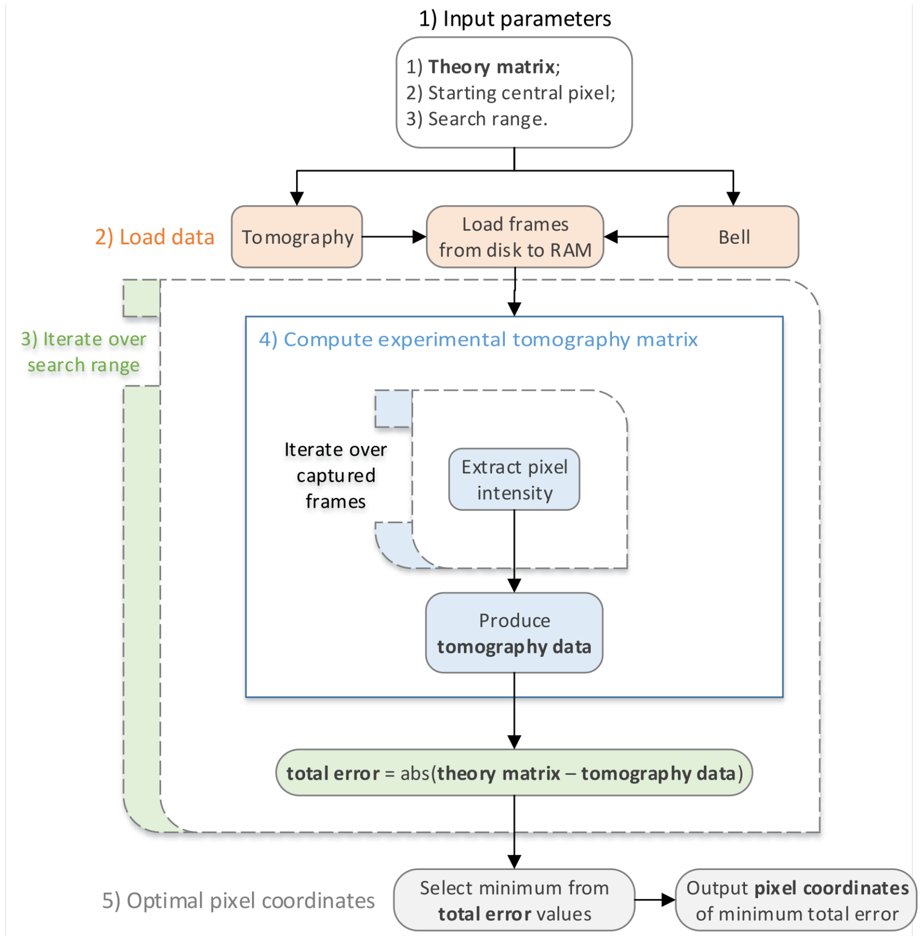

1) Input parameters
1) Theory matrix;
2) Starting central pixel;
3) Search range.
2) Load data
Tomography
Load frames from disk to RAM
Bell
3) Iterate over search range
4) Compute experimental tomography matrix
Iterate over captured frames
Extract pixel intensity
Produce tomography data
total error = abs(theory matrix – tomography data)
5) Optimal pixel coordinates
Select minimum from total error values
Output pixel coordinates of minimum total error

Figure 8: Flow-diagram of the optimiser programme.

4 Conclusions

In these notes we have provided information about the custom software that is used to control the roto-flip stages, allowing to perform a state tomography measurement and analyse the corresponding data. All of the shared resources can be found at the following link [1].

References

[1] Supplemental material to this publication including the video of the system in action, 3D-designs of the roto-flip stages, the Arduino firmware, and the LabVIEW programmes, can be found at the following links: http://dx.doi.org/10.5525/gla.researchdata.559 and https://github.com/ErmesT/Quantum-state-tomography-on-spatial-modes.