30 September 2021
anni.kauniskangas@hotmail.com

A Python tool for visual activity planning at the Proton Synchrotron

Author: Anni Kauniskangas
Supervisors: Fernando Baltasar Dos Santos Pedrosa, Estrella Vergara Fernandez

Keywords: ACE, planning, Proton Synchrotron, Python, Bokeh, Visualisation, PowerPoint, engineering, Summer Student Programme, Python automation, data visualisation

Summary

This report describes the work done as a part of the CERN Summer Student Programme of 2021. The project was aimed at developing a tool for the automated creation of visual representations of activity planning data on the Proton Synchrotron. The end goal was to obtain a program that could read planning data in an Excel format as an input, and produce PowerPoint slides with the data visualisations for each day as an output. The project was implemented in Python using the Bokeh visualisation library. The final code was able to meet the goals, automatically producing slides with the relevant visual activity representations as well as generating an interactive version of the map that allowed the user to view the activities by selecting a day from a calendar interface.

Contents

1 Introduction

The Accelerator Coordination and Engineering (ACE) Group of the Engineering Department at CERN is responsible for the coordination of maintenance, testing and development activities at the CERN accelerator complex during planned stops. Considering the size and complexity of machines such as the LHC, it is easy to see why efficient and accurate planning plays a key role in the safe operation and maintenance of particle accelerators. The focus of the project described in this report was in improving the tools, practices and communication for activity planning in one of the LHC's injectors.

2 Project goals

The aim of this summer student project was to develop a Python program that would help automate the creation of visual representations from the ACE Group's activity planning data. Previously, images that display the locations of activities as coloured regions on the machines' maps have been used in the group's coordination meetings to help visualise the extent of the planned works. These spatial representations have the benefit of providing a lot of information on a single figure and giving an intuitive idea of the situation at a given time. The downside of the visualisations has so far been the rather laborious process of manually creating them for each day by colouring the correct machine regions in some image manipulation software.

The idea of this project arose from the need to reduce the manual work required in the generation of the visual representations of planning data. The end goal was to have a tool that could read in activity planning data from an Excel worksheet, create the visualisations for each day in a given period, and generate PowerPoint slides with the created images as an output. The work was limited to the planning for the Proton Synchrotron (PS), although the general idea was applicable to all of the machines.

3 Method

3.1 Why Python?

While this project could have been implemented in many different ways, there were a few reasons that made Python the natural choice for it. Firstly, the project involved several different steps from reading and manipulating data to generating graphics and creating PowerPoint slides. Integrating all the different functionalities into a single program required a very flexible and high-level programming language, something that Python is known for.

In addition, the vast selection of open-source packages and libraries available for Python, especially in data-analysis and visualisation, reduced the workload of the project and made it easy to implement high-level functions such as reading Excel files and exporting images. This was a crucial benefit because of the short timeline of the summer project, which would not have allowed enough time for crafting these functions from scratch. Furthermore, many of the relevant Python libraries, such as Pandas which is used handling large multi-column datasets,

are highly optimised as a result of years of development, and offer superior performance to most "self-made" solutions.

Finally, the easy readability, open-source access and large documentation database made Python a sensible choice for a project that would most likely be picked up and developed further by others after the summer.

3.2 Project plan

With a larger programming project like this, it made sense to divide the work into smaller tasks. The plan was to progress in approximately the following steps:

1. Select a suitable Python visualisation library
2. Work out how to read in data from Excel spreadsheets
3. Learn to plot pictures and/or shapes with the selected library
4. Develop a method of creating plots based on the given activity location data
5. Generate plots from data, for user-defined time-periods
6. Find out how to access PowerPoint files from Python
7. Automatically generate PowerPoint slides with the relevant visualisation plots added to them

Some tasks were completed quickly while others took more work. The highest priority of the project was to establish a method of drawing the visual representations automatically from data containing the activities, times and locations. The generation of PowerPoint slides was an important aspect of the project, but could only be done once the other features were implemented, so it was left as the last thing to do if the time allowed.

3.3 Choice of visualisation tool

The first task of the project was to choose a suitable Python visualisation tool for the application. This needed to be done before any programming work could be undertaken. While the selection of the tool might seem like an unimportant part of the project, the sheer number of different libraries available made the choice non-trivial.

After gauging the options available, the following graphics and data visualisation libraries were selected for consideration:

1. Matplotlib
2. Seaborn
3. Bokeh
4. PyGame

5. Plotly
6. Pyecharts

The choice was then made based on the following criteria:

- Ability to display and use png/jpeg/svg images
- Ability to draw arbitrary geometric shapes
- Interactive features
- Export formats
- General complexity and ease of use
- Author's previous experience
- Dependencies and integration with other libraries like NumPy, Pandas
- Amount and quality of documentation available

With these considerations, the tool best suited for the purpose was found to be Bokeh. Bokeh is a fairly mature data visualisation library for Python that is mainly focused on creating interactive, web-based plots from data. It provides an easy way to create typical data visualisations such as pie charts, histograms and scatter plots, but also has a surprising amount of flexibility for other graphics. Bokeh supports images in png, jpeg and svg formats, and provides methods for creating arbitrary polygon shapes from coordinates. Plots are displayed in html format in a browser, but can also easily be exported to png or svg form. Bokeh integrates well with Pandas' *DataFrame* datatype, and provides some easy pre-built interactive elements such as sliders and hovers for data inspection. Perhaps most importantly, the Bokeh webpage provides extensive and clear documentation as well as code examples for getting started. Having the documentation information, along with examples from the wide userbase made it easy to learn the basics and helped significantly in debugging and troubleshooting.

3.4 Creating the images

One of the main challenges in creating the visualisations was to find a way to accurately mark any of the possible regions on the Proton Synchrotron map. After some testing with Bokeh shapes, it became evident that doing this by manually defining the polygon or circle-arc coordinates was very difficult and did not yield good results.

The alternative idea was to utilise the information on an existing svg image of the PS to draw and colour the relevant regions. This approach seemed ideal, as it eliminated the need to try and manually define coordinates or areas inside the code. The caveat of this method was that Bokeh did not provide a method for directly drawing shapes based on the path objects used by svg format. After some investigation into similar examples for Python and Bokeh, a workaround was found: the information on the svg image could be used for drawing

shapes in Bokeh, as long as the svg paths were first parsed into generic x-y coordinates. This method was applied in the code to draw the arbitrary shapes of the different PS locations, and was found to work well, providing also good forwards compatibility for applying the code for other machine layouts.

3.5 Summary of libraries and packages used

1. **Bokeh**
   The most central part of this project was the visual representation of the location data. As described in the previous section, this was implemented with an existing, high-level visualisation library of choice. Bokeh, the one chosen, is an extensive, open-source visualisation library for Python. It is built specifically with interactive data visualisations in mind, and has a variety of built-in methods available for creating charts, plots, histograms and shapes, as well as interactive tools such as sliders and cursors.

   For this project, the main functionalities of Bokeh used were the *Figure*-class and the *figure.patches*-method. *Figure* is subclass of *Plot*, and it creates a blank figure canvas with default axes and tools. The *figure*-instances have methods for adding Bokeh *glyphs* on the canvas, that can be for example, lines, bars or polygons. The glyphs used for this project were *Patches* objects, which represent arbitrary polygon shapes with in-built attributes like colour and opacity. They were needed to draw the outlines of the PS map, as well as the coloured sections and sectors on the figure canvas from a pre-defined set of coordinates.

   For the interactive functions of the visualisation, the project code made use of Bokeh's *HoverTool*-tool and the *DatePicker*-widget. The *HoverTool* was configured to inspect and display information related to the data entry when hovering the mouse cursor on top of the corresponding *Patches*-glyph. In practice, this meant that for each displayed event on the map, the user could inspect the name and additional information of the event by hovering the cursor over its location. The *DatePicker*-widget was added to filter the amount of data displayed on the visualisation by date. This allowed the user to pick a date on a given interval from the *DatePicker*'s visual calendar interface. The selection would then trigger a callback that executed the necessary code to filter the dataset, choosing the correct events for the selected date and redrawing the figure with the correct glyphs.

2. **Pandas**
   Pandas is an industry standard when it comes to handling multi-column data in Python. It was a natural choice for this project, where the input data came in .xlsx format and had multiple columns of variable type data. The main usage for Pandas in this project was in reading in the data from the supplied files, but the Pandas *DataFrame* methods came also useful in filtering, merging and retrieving entries from the data in different parts of the code.

3. **Svgpathtools**
   For the visualisations, the best way of colouring in the relevant regions of the PS map

was to use the information contained in the existing svg file. However, the problem with this approach was that the svg file contained the area shapes as path objects, which were not directly supported by Bokeh. The method to solving this problem was to parse the svg paths into plain x-y coordinates that could then be inputted to Bokeh *Patches*.

The tools for this coordinate parsing were provided by the package svgpathtools [1, 2]. The package, created by Andrew Port, is a collection of tools for reading, writing and manipulating svg files and related Bézier curves. In this project, it was used for reading in the svg file information, and handling the path object and their continuous subpaths. The code used for extracting the x-y coordinates from the paths was borrowed from the repository SVG2coords [3] by Mike Woodward that used the method *svg2paths2* from svgpathtools.

4. **Python-pptx**
   The interactive visualisation was the first main goal of the project. The second one was to automate the generation of PowerPoint slides for the ACE group's coordination meetings. This required being able to generate and manipulate .pptx files programmatically, from a Python script. Building this functionality from scratch would have been far from trivial and taken more time than was available for the whole project, but fortunately, there was already a Python library created for the purpose: python-pptx.

   Python-pptx [4], created by Steve Canny, is a library that allows the generation and modification of Microsoft PowerPoint files directly from Python. It is still a project in development, but already provides methods for creating new PowerPoints, opening existing ones, and adding slides with text, pictures or charts on them. In this project, python-pptx was used to do all the heavy lifting on the PowerPoint generation, so that slides could be generated from the main script with the relevant images and text added to them.

5. **Other dependencies**
   In addition to the packages and libraries mentioned above, the code had some dependencies on other packages that were used for minor things.

   The package **re**, which provides regular expressions for Python, was used for transforming the names of locations, such as "Sector 05" into the corresponding codes e.g "PR-SECTOR-05". This was done by using re's functionalities for searching and replacing parts of strings.

   The Python module **datetime** was required for transforming strings representing dates into *datetime objects*. This was essential for comparing dates with each other when filtering the activity data based on selected time periods.

   Another dependency in the code was the webdriver required for running interactive visualisations on the Bokeh server. As stated in the Bokeh documentation [5], either **ChromeDriver** or **GeckoDriver** is suitable for the purpose.

4 Results

4.1 Program overview

An overview of the structure of the finished project code is provided in the diagrams below. Fig.1 displays a top-down modular diagram of code, representing all the different functions and sub-tasks performed during a run. The main steps of the program are all executed in smaller parts, some of which also consist of their own sub-tasks. For example, the generation of PowerPoint slides requires the program to first filter out the correct activities by date, then to plot the corresponding locations on the map, after which the figure can be exported as an image and added to a slide. The creation of the slide on its own also requires several steps, such as adding the title, text and a correctly scaled image. In general, the overall level of complexity in the code is reflected in the number of sub-tasks included in each step of the diagram.

For a more intuitive idea on how the project code works, Fig.2 presents a mostly linear flowchart of the program. The diagram highlights the main steps of the code, as well as points of decision and user input. The colour codes in both Fig.1 and Fig.2 indicate the organisation of the functions into the different python modules based on functionality. The code was divided into four separate modules to help simplify and improve the readability of the main program file.

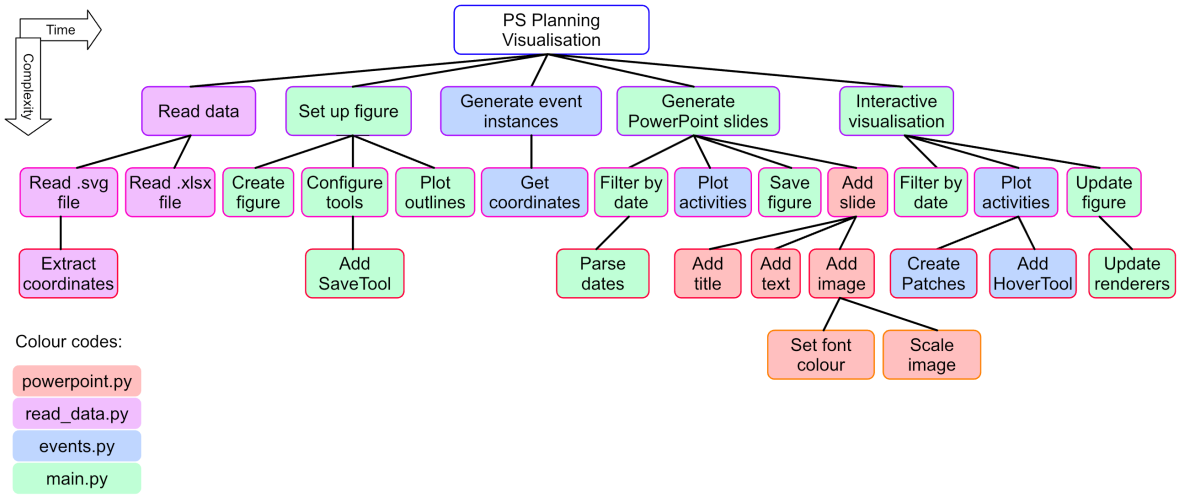


Figure 1: A top-down representation of the project code. Tasks are performed from left to right, and the complexity increases from top to bottom.

4.2 Functionality

At its current stage, the visualisation tool has two main functionalities: the automated generation of PowerPoint slides, and the interactive map of activities. In principle, the program takes an Excel spreadsheet of planning data as an input, and returns PowerPoint

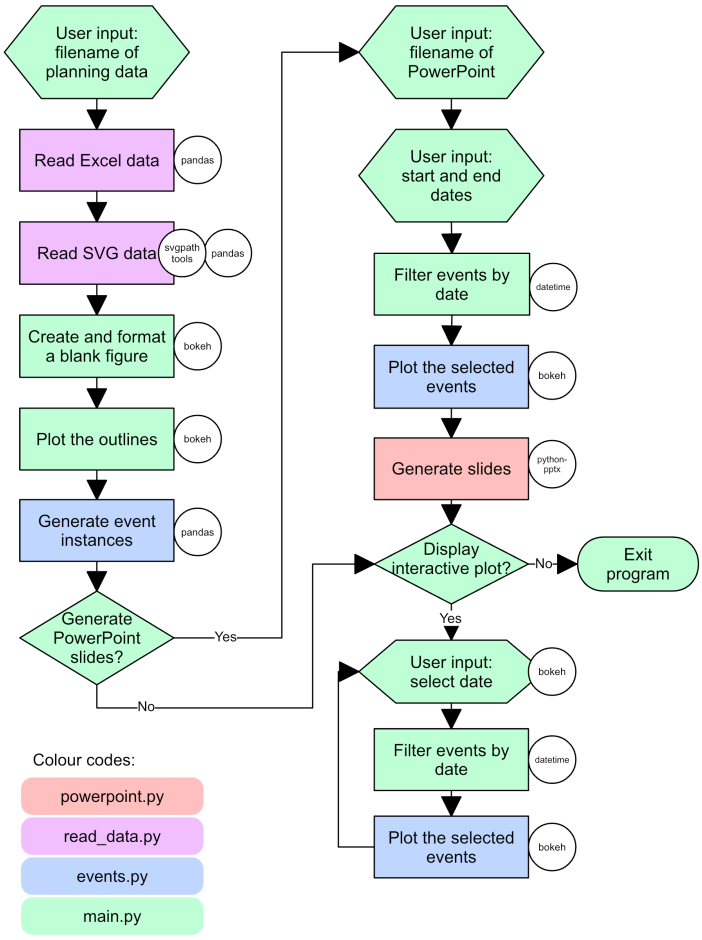


Figure 2: A flowchart of the visualisation program. The colour codes indentify the module in which each process in contained in.

slides and an interactive map as an output. However as seen from Fig.1-2, the process from the input to the output is complex and involves several different steps.

The program begins by asking the user to specify a path to the file containing the activity planning data. The file will then be read, saving the data onto a Pandas *DataFrame* object. In the next step, the location data from the svg image is read, and x-y coordinates are extracted from the svg paths.

After all the data has been collected and formatted into *DataFrames*, the program initiates a blank Bokeh figure and plots the outlines of the Proton Synchrotron map on it. Then, the activity planning data is iterated through, and an instance of an *event* object is created for each activity to be represented on the map. The *Event* class is a class created specifically for this project. The instances of the class keep all the relevant information of the activity as attributes of the object to provide easy access to them in other parts of the code. The class also provides a *draw* method to render the activity location patch, along with the relevant *HoverTool*, on a given Bokeh figure.

Once all the preliminary steps of reading and manipulating data and setting up a figure

are done, the code proceeds to the main functionalities. The user can choose to generate PowerPoint slides for a selected range of dates, or to move on to the interactive visualisation. If slides are to be generated, the code goes on to iterate over each day in the selected period, filtering out the relevant events for each day and plotting them. A slide per day, such as the one shown in Fig.3, is added to the given PowerPoint presentation using the functions in the module powerpoint.py. Then, the user can select to either end the program or to launch the interactive visualisation.

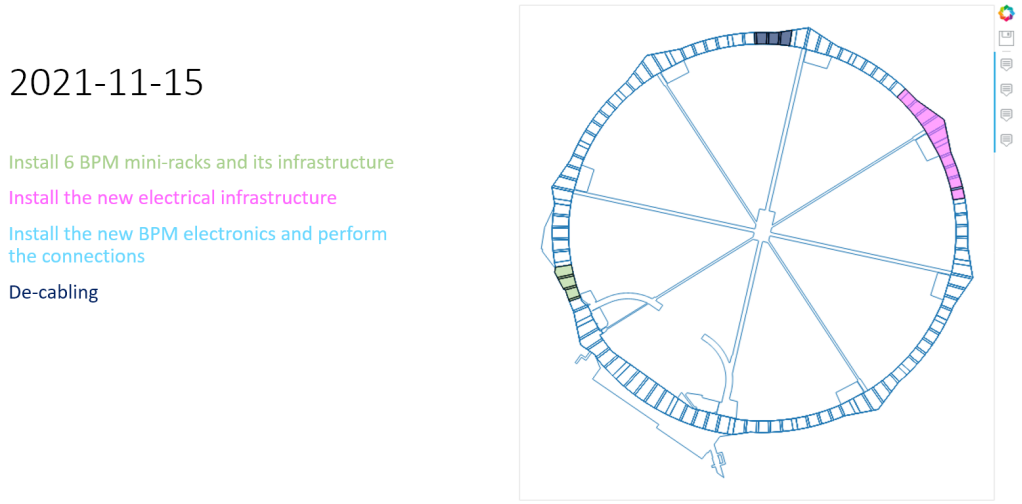
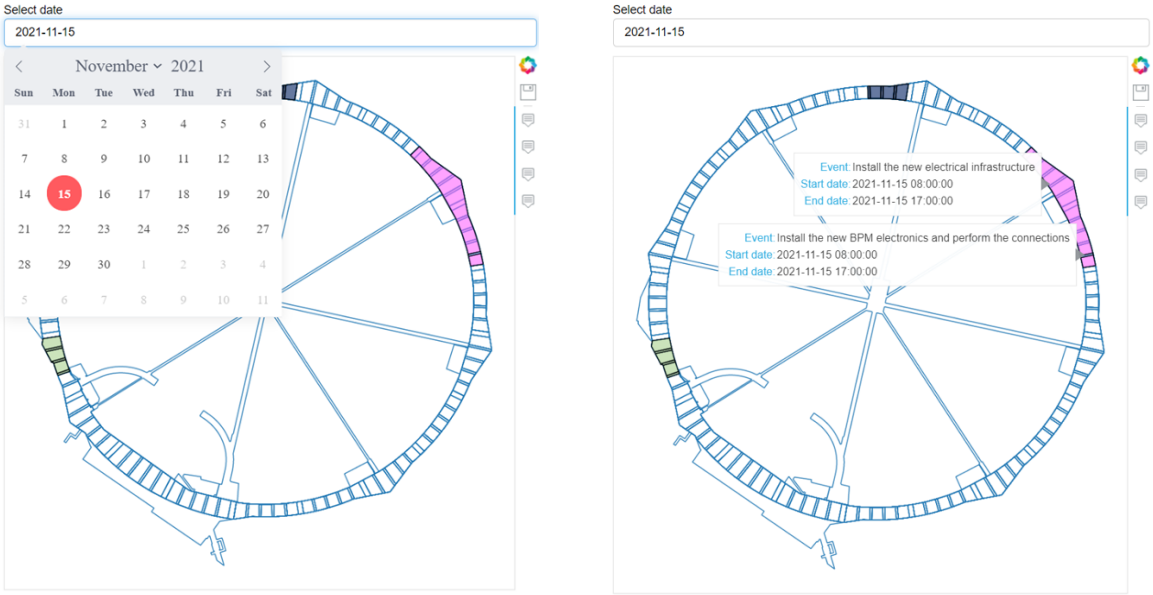


Figure 3: And example of a slide generated by the project code. The layout consists of simply the date as the title, the image of the relevant visualisation plot, and the names of the activities with colours matching those of the plotted *Patches*.

The interactive visualisation map runs on a Bokeh server, and is automatically launched in a browser window. The tool provides a calendar widget, shown in Fig.4 a), that allows the user to select a date from a calendar-like interface. The plot is updated at each user interaction to display the events that are active on the selected date. The interactive map also incorporates Bokeh *HoverTools* for inspecting the data; the user can, by hovering the mouse cursor over a location, see the name and dates of the relevant activity as demonstrated in Fig.4 b). The Bokeh *SaveTool* was also added to the right-hand side menu of the plot to give the user an option to download the plot as a png image by simply clicking the "save" icon.

4.3 Limitations

There are a few limitations to the project code's functionality. Firstly, python-pptx does not currently support adding slides into the middle of an existing presentation. Due to this,



(a) Calendar widget for date selection

(b) HoverTool for inspecting the activity information

Figure 4: Screencaptures of the interactive visualisation tool. The calendar widget, shown in a) allows the user to select a day for which they want to generate a visual planning representation for. The *HoverTool* demonstrated in b) can inspect the activity information by hovering the mouse cursor over the relevant location. In addition to these, the interactive tool includes a Bokeh *SaveTool* on the right-hand menu, that gives the user the option to download the current plot as a png image.

all slides generated by the program are added to the end of the given presentation, and any reordering has to be done manually.

Secondly, there are some problems with the interactive visualisation when it comes to representing simultaneous activities at the same location. If multiple activities occur on the same day at the same location, the coloured patches marking the location get rendered on top of each other. The problem arises from the Bokeh *HoverTools* that are used to inspect the name and dates of the activities by hovering over the location. Instead of displaying the information for all overlapped activities, the *HoverTool* textboxes get rendered on top of each other, and only the information of one of the activities gets displayed. Unfortunately, no easy fix or workaround for this problem was found, as the textbox behaviour is built into Bokeh and is not user configurable. The issue can be mitigated by allowing the user to toggle the *HoverTools* on an off, and inspecting the overlapping textboxes one at a time. A way to avoid the issue altogether would be to find an alternative way to display the activity information on the interactive plot without using *HoverTools*.

Finally, the program functionality is currently still limited to a very specific set of user inputs, and cannot handle data in any formats that deviate from the standard used in the program development. The activity planning data must be provided in .xlsx format and needs to have the correct names for columns and locations. Due to the way strings are treated by Python, it is not easy to implement code that would allow for highly flexible

input formatting. However, the problem could be alleviated from the user's point of view by adding "safety checks" to the code to ensure the correctness of the input data format.

5 Future Development

The work done during this summer project served as a proof of concept for visual activity planning, and is the first step towards a more general-purpose tool. The obvious goal for the future is to extend the functionality of the program to the other CERN machines that are being managed by the EN-ACE group. With slight modifications, the existing code could be applied to all the different machines to create similar visualisations for them. The end product should include a user interface for selecting a machine as well as the dates for the visual representations.

While the code currently only works with the data and locations of the Proton Synchrotron, it should not be difficult to apply it to the other machines. The main thing required is the appropriate svg file of the machine outlines and all possible areas of interest. With the svg information, the locations for any machine can easily be plotted based on the data by inputting the correct location labels in the planning file. The only caveat is, that for some machines such as the LHC, the locations are organised into layers of larger areas containing smaller sections. The current code has not been tested on svg files containing multiple layers, and it is not clear whether the referencing of both the layers and the constituent areas will work properly.

In addition to the extension to other machines, the visualisation tool needs development in the usability and interface aspects. The product of the summer project is a collection of Python files that need to be run by typing a command into a terminal. The program works and is perfectly fine as a crude prototype, but it is by no means ideal from the user perspective. Future development of the project should focus on making the tool simple and easy to use for everyone, including those inexperienced in programming and Python.

One possible way of achieving an accessible interface would be to embed the interactive visualisation tool on a CERN webpage or server. That way, the tool would always be easily available and ready to use for those who need it. Implementing an online version of the interactive tool should, in theory, be somewhat straightforward, as Bokeh is designed with web-based visualisations in mind and the output is already in html format.

6 Conclusion

This summer student project was aimed at creating a Python program to automate the generation of visual representations of engineering activities at the Proton Synchrotron. Similar visualisations that showed the locations of activities on the machine map had previously been manually drawn and used in the coordination meetings for the Accelerator Coordination and Engineering group.

The development of the project took place in several steps. After a suitable Python visualisation tool had been found and selected, the work was focused on generating the images based on activity data. Since the selected library Bokeh supported interactive features, the focus was then moved to creating an interactive visualisation that allowed the user to

inspect the activities by selecting a date from a calendar-like interface. As the final step, the generated figures were added onto PowerPoint slides using the python-pptx package. The finished code gave the user an option to generate slides and to launch the browser-based interactive version of the visualisation.

The end results of this project demonstrated the capabilities of Python in the automation of tasks and visualisations, and showed its applicability to the activity planning of large-scale facilities. The functioning application for the Proton Synchrotron acted as a proof of concept, with the code being easily extendable to other machines by providing svg format maps of the layouts and locations. The future development of the project should focus on adding support for the other machines of the accelerator complex as well as on developing an easy to use interface to make the tool more accessible.

7 Acknowledgements

I would like to thank CERN for the opportunity to participate in the Summer Student Programme. The excellent lecture series has given me more insight into particle physics, accelerators and detectors, as well as an overview into CERN as a whole. The summer project has offered me chance to put my skills into practice and to contribute in a meaningful way to the work being done at CERN. It has also been an interesting journey, during which I have developed my skills and knowledge in programming and data visualisation, and seen some of the possibilities of applying them into useful real-world applications.

I would also like to give special thanks to my supervisors Fernando Baltasar Dos Santos Pedrosa and Estrella Vergara Fernandez for the great guidance, ideas, discussions and helpful suggestions they have given me during the course of this project. The work described here would not have been possible without them.

References

[1] A. Port, "mathandy/svgpathtools: A collection of tools for manipulating and analyzing svg path objects and bezier curves." date accessed 23/09/2021. [Online]. Available: https://github.com/mathandy/svgpathtools

[2] "Svgpathtools :: Anaconda.org," date accessed 23/09/2021. [Online]. Available: https://anaconda.org/conda-forge/svgpathtools

[3] M. Woodward, "Mikewoodward/svg2coords: Converts an svg file to coordinates, suitable for using with bokeh 'patch'." date accessed 23/09/2021. [Online]. Available: https://github.com/MikeWoodward/SVG2Coords

[4] S. Canny, "scanny/python-pptx: Create open xml powerpoint documents in python," date accessed 23/09/2021. [Online]. Available: https://github.com/scanny/python-pptx

[5] "Exporting plots — bokeh 2.3.3 documentation," date accessed 23/09/2021. [Online]. Available: https://docs.bokeh.org/en/latest/docs/user_guide/export.html