Design optimization of innovative electrical machines topologies based on Pyleecan open-source object-oriented software

P. Bonneel, J. Le Besnerais, E. Devillers, C. Marinel, R. Pile

***Abstract* – Breakthrough innovations in electrical machines may be limited by parametric overlays and templates provided in commercial electromagnetic simulation software. Disruptive design spaces must therefore be explored using more flexible open-source software solutions. However, a significant scripting effort is necessary to define some new parametric geometries suitable for design optimization based on open source multiphysics solvers. This article illustrates the use of Pyleecan open-source simulation software under Python to more efficiently model, evaluate and optimize disruptive topologies of 2D or 3D electrical machines. The current status of Pyleecan initiative is first presented. Then, the principle and the advantages of the object-oriented approach of electrical machines are detailed. Some examples of complex innovative topologies that can be generated with Pyleecan are then introduced (e.g. complex winding, uneven slot types, multiple rotor and stators), as well as the optimization possibilities. Finally, the development roadmap of Pyleecan project is given.**

***Index Terms*-- Design optimization, Electrical machines, Multiphysics, Open source, Simulation software**

I. Introduction

Pyleecan stands for PYthon Library for Electrical Engineering Computational ANalysis. This open-source project under Python and Apache license was first presented at ICEM 2018 [1]. The initial purpose of the project is to boost applied research and development in electric mobility and sustainable energies by providing an object-oriented development framework of electrical machines and drives modeling. The project is in Python, one of the most widely used scientific software language, and it includes IDE and graphical post-processing features which are as easy to use.

Fig. 1. Pyleecan logo (www.pyleecan.org)

One can indeed observe that very similar functionalities, such as coupling a Matlab / Scilab / Octave script to FEMM [2], Elmer [3] or GetDP [4] in order to draw a parametrized geometry of an electrical machine, are redundantly developed among electrical engineering laboratories and industrial R&D departments, without being shared to the whole scientific community. This is a major issue for reproducible science and research efficiency, and this observation has motivated the start of Pyleecan project. To efficiently share, version, correct, and drive the development of a complex scientific code, the use of an online platform with software development services is necessary. Today, Pyleecan project is hosted on GitHub at https://github.com/Eomys/pyleecan.

Currently, Pyleecan handles the geometrical modelling of main 2D radial flux machines such as:

- surface or interior permanent magnet machines (SPMSM, IPMSM),
- synchronous reluctance machines (SynRM),
- squirrel-cage induction machines and doubly-fed induction machines (SCIM, DFIM),
- would rotor synchronous machines and salient pole synchronous machines (WSRM),
- switched reluctance machines (SRM).

All topologies can be drawn as inner or outer rotor, with any winding types, slot shapes, pocket shapes and ventilation duct shapes. Fig. 2 illustrates some electrical machines topologies that can be modeled in Pyleecan.

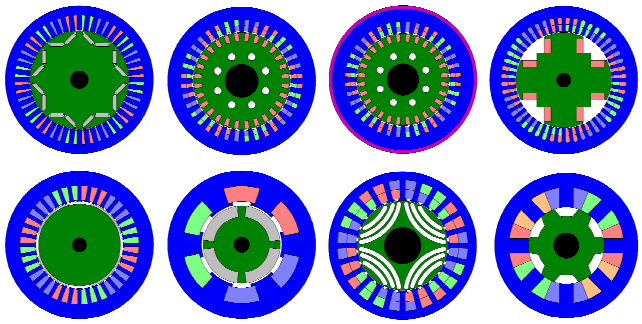

Fig. 2. Examples of topologies modelled with Pyleecan

Pyleecan is also fully coupled to the open-source electromagnetic finite-element software FEMM, including sliding band solver and symmetries. This means that current-driven non-linear magnetostatic simulations can be automatically carried out and post-processed to evaluate electromagnetic performances of these machines such as torque, torque ripple, inductances, flux linkages, back electromotive forces, and magnetic losses.

In January 2020, a Graphical User Interface (GUI) developed under PyQt was added to the project to graphically design all the available machine types. The GUI code was structured to automatically include new slot shapes, and to ease the addition of new topologies (see Fig. 3).

Pyleecan also includes a multiphysic material library for magnetic materials (e.g. magnets, laminations), active materials (e.g. copper) and structural materials. Material properties can be added and edited through the GUI.

P. Bonneel, J. Le Besnerais, E. Devillers, C. Marinel and R. Pile are with the company EOMYS ENGINEERING, Lille-Hellemmes, France (website: www.eomys.com, e-mail: contact@eomys.com).

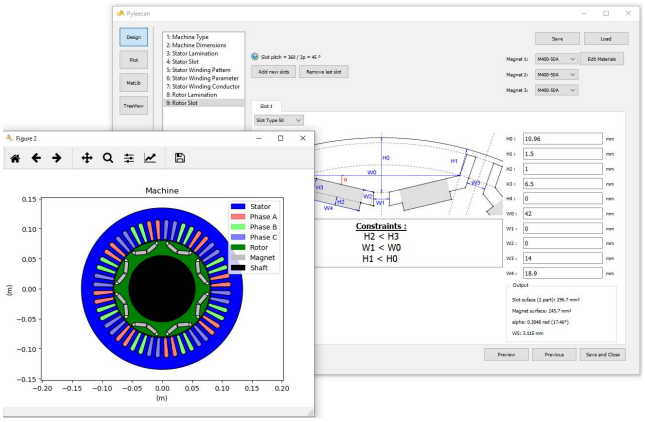

Fig. 3. GUI of Pyleecan

II. Geometric Modeler of Pyleecan

A. *Principle of OOP*

As explained in the publication that introduced Pyleecan [1], Object-Oriented Programming (OOP) is a programming paradigm based on the concept of "objects". These objects, programmed as "classes", represent an abstraction of real objects that are found in electrical machines such as laminations, magnets, winding, etc. These entities are defined by their attributes and their methods.

Using object orientation is particularly useful during design exploration. As an example, once a lamination with a new AC winding type is defined, it can be easily applied to all AC machines (e.g. PMSM, SCIM, DFIM).

B. *Pyleecan Classes Organization*

The geometric modeler of Pyleecan is organized around the *Machine*, *Lamination* and *Slot* classes (and their respective daughters). The *Machine* classes detail and gather all the parts of the machine (stator, rotor, shaft, frame…). Fig. 4 shows how the different machine types are organized.

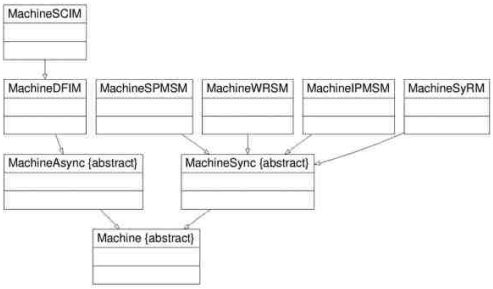

Fig. 4. Machine classes organization

This graph shows the relation between classes. The arrow indicates that a class "inherits" from another. For instance, *MachineIPMSM* and *MachineWRSM* (respectively the classes for Interior Permanent Magnet Synchronous Machine and Wound Rotor Synchronous Machines) inherit from *MachineSync* (abstract class for Synchronous machine). In this case, *MachineIPMSM* is said to be a "daughter" of the *MachineSync* class, or that *MachineIPMSM* is a particular case of *MachineSync*.

OOP enables to define "interfaces", to define parts of the code as black boxes with predefined input and output formats. Thus, different objects can be used to model parts of the software provided that they follow the interface standards. In this case, *MachineSync* can be seen as an interface defining what can be done with a synchronous machine (e.g. calculating D-axis and Q-axis position). Then, each daughter provides its own way of implementing the interface. In particular, the *MachineSync* says that a synchronous machine has a rotor and a stator. Both *MachineIPMSM* and *MachineWRSM* have a stator with a winding but they have different rotors (with interior magnets or winding around poles). The code defines how to interact with a *MachineSync* rotor whatever its actual kind. This way, a simple command line can be used to draw any electrical machine, which naturally calls the respective drawing command lines of rotor and stator laminations, each drawing method being adapted to each lamination type.

The *Machine* interface enables to define electrical machines with different types of laminations. Fig. 5 shows how *Lamination* classes are organized.

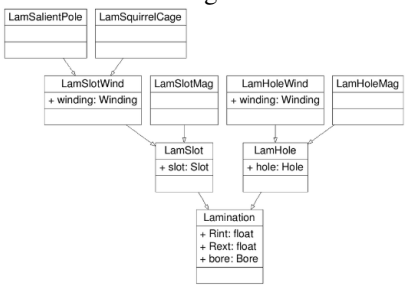

Fig. 5. Lamination classes organization

The *Lamination* class corresponds to a plain cylinder lamination, while *LamHole* and *LamSlot* correspond to *Laminations* with empty holes inside the lamination or empty slots along the lamination bore radius. Fig. 6 shows how Holes and Slots are defined.

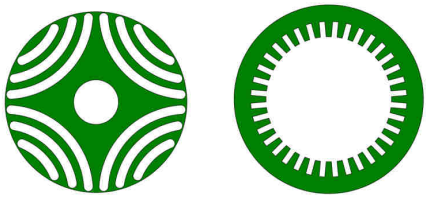

Fig. 6. LamHole on the left, LamSlot on the right

Holes are by definition "carved" into the cylinder, while *Slots* are "grooved" along bore diameter. Then, *LamSlotWind* and *LamHoleMag* correspond respectively to laminations with slots containing windings and to laminations with holes containing magnets. Fig. 7 shows how the Slots and Holes interfaces are organized.

As for the *Lamination* class, the *Slot* and *Hole* classes are organized depending on whether they are intended to contain a winding, a magnet or nothing. For instance, *SlotW10* and *HoleM10* represent a specific parameterized geometry of the *Slot* and *Hole* classes. All different *Slot* and *Hole* parameterized geometries are available in the GUI.

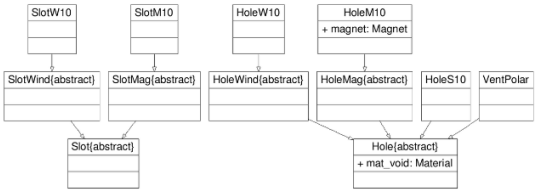

Fig. 7. Slot classes organization

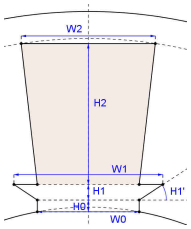

Fig. 8. SlotW10 class schematics

C. Geometry Modeler

This part details how Pyleecan takes advantage of the OOP to simplify the definition of complex topologies. The method used in Pyleecan to draw a *Machine* with two *LamSlot* objects for rotor and stator is hereafter described. Each daughter of *Lamination* has its own method to be drawn - some of them are presented in part IV - but they all follow the logic of *LamSlot*.

The geometry modeler is organized around the *build_geometry* method that returns a "list of *Surface* objects needed to draw the object". *Surface* objects are defined by a label, a reference point (in complex coordinates) inside the surface, and a list of *Line* objects corresponding to surface edges. The *Line* object defines several ways to go from one point to another (arc or segment). Each line can also have its own label. As a convention, the line list of a *Surface* object is defined in order to describe the edges of a closed surface.

The *build_geometry* method is implemented in the *Machine*, *Lamination* and *Slot* objects. To gain in abstraction, the machine *build_geometry* method calls the lamination *build_geometry* method, which itself calls the slot one.

The *Slot build_geometry* method returns a list of *Lines* defining the contour of a single slot centered along x- axis. Lines are ordered along trigonometric direction. Each slot type has its own *build_geometry* method returning a different set of lines according to its parametric geometry. The list of available slot types is available in Pyleecan GUI and on Github.

Then, in the *LamSlot build_geometry* method, the *Slot comp_angle_opening* method is called to get the angular position of the starting and ending points of the *Slot* along the bore radius. *LamSlot build_geometry* method then copies / rotates the first slot contour *Line* and connects it with bore radius lines (most of the time an arc, but it can be easily changed to alter the bore shape or include notches). As the slot is manipulated through its beginning and ending points' angular positions, this method which generates the *LamSlot* bore contour works whatever the slot type used. Thanks to OOP, a new slot type can be easily introduced into Pyleecan by defining a new *build_geometry* method for this *Slot*. Moreover, the *LamSlot build_geometry* method includes a parameter for symmetry. To draw only half of the machine, Pyleecan copies/rotates only half the slot, and only half of the first and last bore radius lines are added. In the other *Lamination* objects, this behavior is adapted to also consider surface objects for winding, holes and magnets if needed. This means that any 2D geometry can be easily cut to include any type of symmetry depending on physics (e.g. magnetic, thermal, structural).

Finally, the *Machine build_geometry* method is the simplest one. To get all the *Surfaces* needed to draw the machine, all the surfaces of each part are simply concatenated by calling the corresponding *build_geometry* local method. In our example, since both rotor and stator laminations are of type *LamSlot*, the same code is used for both.

III. 2D/3D Modeling and Meshing

A. Plot Machine

As seen in part II. , *build_geometry* method enables to get a "list of Surface objects needed to draw the object". The first application of this method is to draw the machine (*plot* method of *Machine*). In this global method, the local *plot* method of each machine part (rotor, shaft, frame…) is called, using the corresponding *build_geometry* methods. Then, each surface is converted to a matplotlib "patch" by the *Surface* method *get_patch*, that mostly returns its Polygon equivalent. As each surface has its own label, it can be identified and set to the correct color for the rotor, the magnets, winding, etc. The legend is then adapted according to the surfaces that are currently plotted.

B. 2D Mesh Generation - FEMM coupling example

Pyleecan currently includes a coupling with FEMM [2] to automatically define a 2D geometry and mesh it, define a non-linear magnetostatic problem (e.g. physics of magnetic materials, current sources), solve it and post process it to obtain the magnetic flux density distribution inside the whole electrical machine. The coupling with FEMM is strongly built around the geometry modeler. The logic of the coupling with FEMM can be summarized with the following pseudo-code:

```
for surface in machine.build_geometry()
   for line in surface.get_lines():
      line.draw_FEMM()
   assign_surface(surface.ref_point, surface.label)
```

As the *build_geometry* method enables to get only a symmetric part of the lamination (half of the lamination for instance), Fig. 9 shows that symmetries are natively available in the FEMM coupling by calling *build_geometry* with the right input parameters.

For the boundary conditions on the yoke edges, the *build_geometry* method of the corresponding *Lamination* class sets the label of the corresponding lines and then, when drawing the lines in FEMM, Pyleecan checks if the line has a label that requires to set a boundary condition.

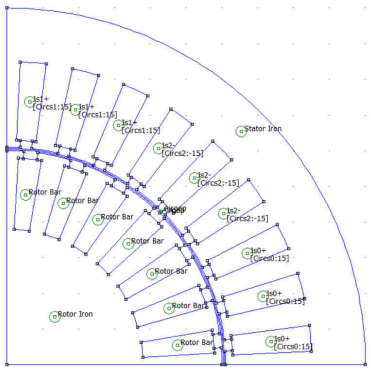

Fig. 9. FEMM model with symmetry obtained with Pyleecan

C. 3D Mesh Generation - Gmsh coupling example

Pyleecan also currently includes a coupling with Gmsh [4], an open source mesher, to generate a 3D mesh of a lamination with empty slots. This coupling was developed to prepare open-source 3D Finite Elements Analysis (FEA) magnetic calculations with GetDP/OneLab, but also to ease calls to other free multiphysic FEA solvers such as Elmer [3] and Agros2D [5]. Gmsh coupling uses *build_geometry* to get a surface that defines a symmetric part of the Lamination (most of the time a single tooth). This surface is drawn in Gmsh, copied/rotated to get the complete 2D lamination, then extruded and meshed to get the 3D lamination stack.

The geometry modeler creates each line of the surface with its own label. They can therefore be identified and selected to set particular properties (e.g. apply an equivalent

mass or set the number of elements on the line). Fig. 10 shows a *LamSlot* with a *SlotW10* meshed using Pyleecan, where the number of elements is enforced for each tooth line to get smaller elements in the tooth compared to yoke ones.

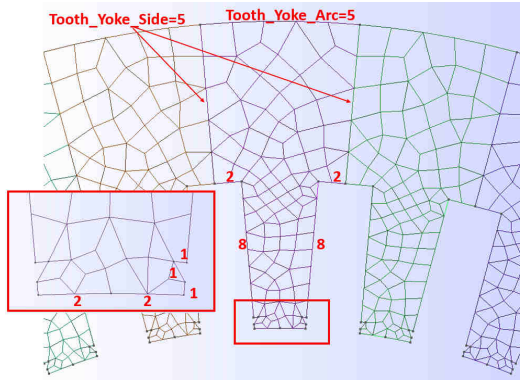

Fig. 10. Top view of a 3D mesh with enforced elements number obtained with Pyleecan

In general, all features and complex topologies that are available through *build_geometry* methods can be reused directly in all coupling methods without further work (including notches, new slot or magnet shapes). Fig. 11 shows the 3D mesh of a *LamSlotMulti* object that is introduced in part IV. In this case, the original surface is not a tooth but is generated with the *build_geometry* method using a user-defined order 4 symmetry.

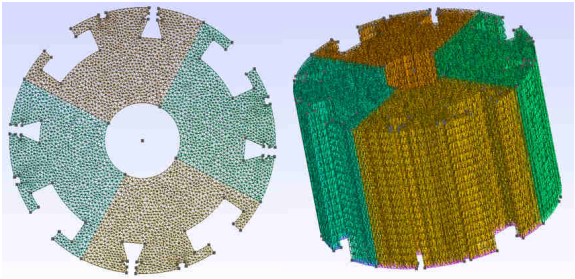

Fig. 11. 3D mesh of a Lamination with two slot kinds and notches obtained with Pyleecan

D. *New Coupling Capabilities*

Both coupling with FEMM and Gmsh are centered around *build_geometry* method that encapsulates all the geometric complexity. These coupling workflows are generic enough to be adapted and reused for other 2D/3D software coupling or to import/export the resulting meshes. Some software now includes Python-friendly scripting capabilities that can also ease Pyleecan coupling (e.g. Altair Flux using pyflux library [6]).

When developing a new coupling feature, all the electrical machine topologies available in Pyleecan will be automatically available, which is a significant gain in development time. Conversely, when developing new topologies in the geometric modeler of Pyleecan, it would also be directly available in all the software coupling features based on the geometry modeler.

When benchmarking two different software (e.g. Gmsh [4] Vs Salome-Meca [7] meshing algorithm, Ansys Maxwell Vs Altair Flux magnetic solvers), Pyleecan geometry modeler is also interesting, because the geometry is defined according to exactly the same surface objects, lines, and point coordinates in both software, which is a gain in reproducibility.

Finally, the geometry modeler benefits from the open source approach. Researchers working on different electrical machine types or even different scientific fields (heat transfer, structural mechanics, acoustics) can use the same objects and share their common work. If one contributor optimizes a topology or introduces a new feature in the *build_geometry* method, all Pyleecan community can directly use it without any extra work.

IV. Examples Of Topology Variation

This part introduces how to use Pyleecan to define some complex topologies. Most the figures from this publication are done with Pyleecan. Most of these cases are simple variations or a combination of standard objects. All the corresponding code is available on Pyleecan Github [8] for a better understanding on how these topologies are defined and to take inspiration to create new topologies. The code for each figure is gathered in pyleecan/Tests/Plot/test_ICEM_2020.py

A. *Multiple Stators and Rotor*

The *build_geometry* method used for plots and coupling with third party software such as FEMM can be extended to several rotors and stators. This is done defining a list of laminations (instead of just one rotor and one stator) and calling the *build_geometry* of each lamination to get the complete description of the machine. Fig. 12 shows a double stator Flux Switching Permanent Magnet machine defined with two stators and two rotors [9]:

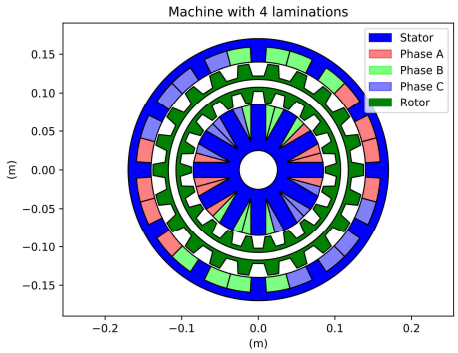

Fig. 12. FSPM with two rotors and two stators obtained with Pyleecan

B. *Uneven Slot / Tooth / Notches*

The *build_geometry* method of the *LamSlot* object can be easily adapted to a lamination with several kinds of *Slots*. The *LamSlotMulti* object contains a list of *Slot* objects and a list of the angular position of each slot center. The *build_geometry* method of this object is nearly the same as the one for *LamSlot*: instead of copying the contour lines of the slot, the *build_geometry* of each slot is called, the proper rotation is applied according to requested angular position, and the lamination bore lines are defined accordingly.

In Fig. 13, two different types of slots are unevenly distributed and combined with evenly distributed rectangular notches. Two slots are also modified to highlight the possibility to change any slot. In Pyleecan, notches are modeled using the Slot object as well, so that every time a new slot is added to Pyleecan, it can automatically be used for notching as well.

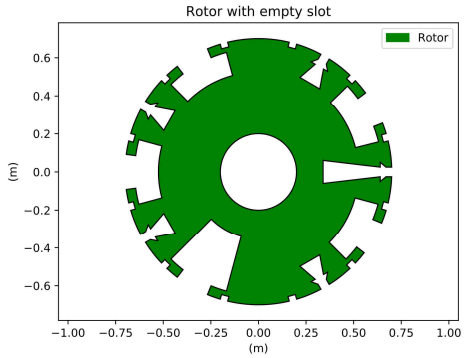

Fig. 13. Lamination with uneven slot and notches obtained with Pyleecan

C. *User-defined slot shapes*

The *Slot* interface requires any slot class to have the following methods: *build_geometry*, *comp_angle_opening*, *comp_height*, *comp_surface*. The three last methods compute the opening angle, the height and the surface of the slot. All three of them can be computed numerically according to the result of *build_geometry*, which makes *build_geometry* the only method required to define a new slot (the others can be defined to provide an analytical way of computing the quantities). This means that the only piece of code needed to add a new slot in Pyleecan is a method to characterize its contour geometry.

Starting from this observation, the user-defined slot class *SlotUD* is defined. This class has two properties: *point_list,* a list of complex coordinates, and *is_sym* to duplicate the coordinates by symmetry (and *Zs* inherited from the Slot class). This slot simply defines several points that are automatically connected with a *Segment* object in a generic *build_geometry* method. The only constraint is that the first (and last point if *is_sym* is False) must be on the bore radius. Fig. 14 shows an example of *SlotUD* (with *Zs*=6)

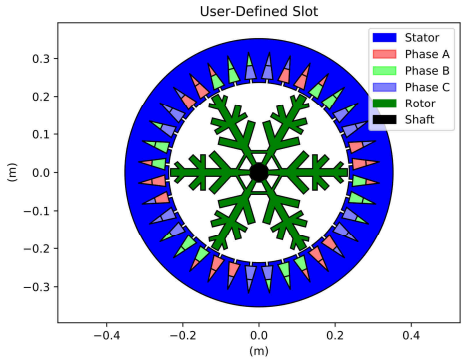

Fig. 14. Lamination with user-defined slot obtained with Pyleecan

D. *User-defined winding*

For the *SlotWind* interface (slot containing winding), three new methods are needed: *build_geometry_wind*, *comp_height_wind* and *comp_surface_wind*. The first one defines the *Surface* objects of winding active materials, the other two compute the winding "height" and surface. As for the *Slot* object, *comp_height_wind* and *comp_surface_wind* can be computed numerically by using the return of *build_geometry_wind* method.

build_geometry_wind takes two parameters as argument: Nrad and Ntan (number of winding layers in radial or tangential direction). When Nrad>1 and/or Ntan >1, instead of returning a single surface, *build_geometry_wind* "cuts" the original active surface to define the correct number of winding layers as shown in Fig. 15.

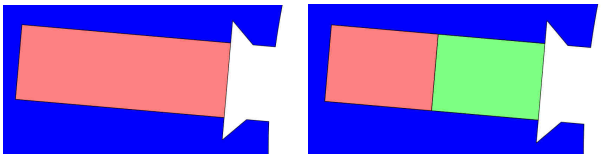

Fig. 15. Left: Nrad=1 and Ntan=1, right: Nrad=2 and Ntan=1

The *build_geometry* method of *LamSlotWind* then copies/rotates all the surfaces to generate all the layers of all the slots. Each surface has a unique label to identify or select it. Pyleecan then defines several *Winding* classes that correspond to different winding patterns. Every winding class has a method *comp_connection_mat* that returns a winding connection matrix of size (Nrad, Ntan, Zs, qs), defining the number of turns in each layer of each slot for each phase. Fig. 16 shows a *LamSlotWind* wound rotor with a *WindingUD* user-defined winding.

Fig. 17 shows that it is also possible to adapt the *build_geometry_wind* of any slot to define uneven winding surface for any layer.

Note that this feature is not available in Pyleecan at the moment of article writing. However, the corresponding code is commented in the *SlotW11 build_geometry_wind* code as a proof of concept.

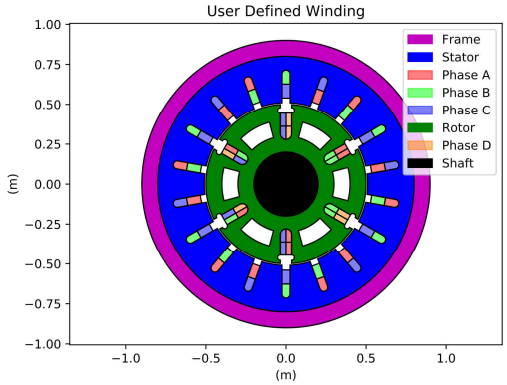

Fig. 16. Illustration of a user-defined winding obtained with Pyleecan

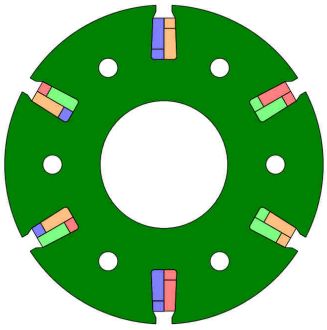

Fig. 17. Uneven winding layers

E. *Uneven bore radius*

When calling *build_geometry*, slot lines are duplicated and connected to lamination bore radius lines. By default, these "bore lines" are simply a single arc between two adjacent slots, but they can be changed to something else. For now, Pyleecan only enables to define an uneven bore radius for *LamHole* (Lamination for Hole, used for rotor of IPMSM machines). The *build_geometry* method of the *LamHole* defines the lamination surface with two circles, when a *Bore* object is set, the bore circle is replaced by the shape defined by the *Bore* object. Fig. 18 shows a *LamHole* with an uneven bore shape to model flux concentration machines:

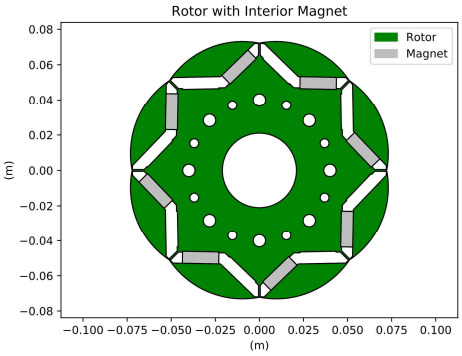

Fig. 18. *LamHole* with uneven bore obtained with Pyleecan

On a side note, the *HoleMag* object contains a *Magnet* object for each magnet in the Hole. Which enables to define different material or magnetization type for each magnet and to completely remove one magnet as illustrated in Fig. 18. When removing a magnet, the corresponding surface is not returned by *build_geometry*.

F. Eccentricity and pole displacement

Each surface is defined as a list of lines and a reference point, it includes rotate and translate methods. Therefore, the surfaces of the machine can be easily altered to simulate manufacturing tolerances or faults. For instance, in Pyleecan coupling with FEMM, a "transform_list" can be defined to apply rotation or translation on some surfaces selected according to their label. Fig. 19 shows the effect of the following "transform_list" on a FEMM model:
transform_list = [{"type": "rotate", "value": 0.08, "label": "MagnetRotorRadial_N_R0_T0_S3"},
*{"type": "translate", "value": gap * 0.75, "label": "Rotor"})]*

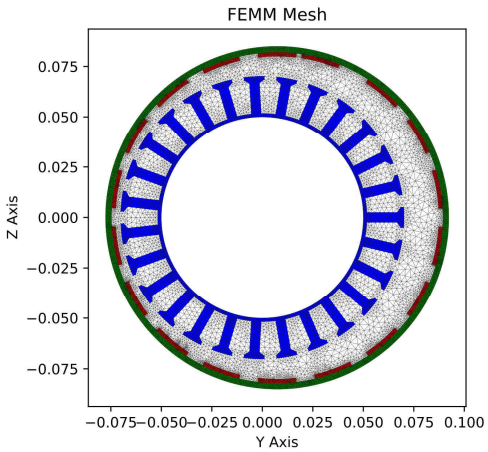

Fig. 19. FEMM model with transformation obtained with Pyleecan

As expected, the third magnet is rotated of 0.08 radians and the whole rotor is translated of 75% airgap wdith. For futur development, Pyleecan should include new objects/options to directly generate the modified surfaces with *build_geometry*.

G. Topology optimization

To take full advantage of the geometric modeler, Pyleecan embeds a coupling with the Python optimization library DEAP [10]. This coupling enables to solve global optimization problem using the NSGA-II algorithm [11], which is often used in electrical machine design optimization for its robustness and possibility to handle mixed variables under constraints. This optimization module of Pyleecan also uses OOP to be flexible and to ease the addition of more optimization algorithms in the future. This organization enables to define every kind of design variables, constraints and objective functions based on Pyleecan objects such as: winding, material properties, slot, magnet, etc.

In the following example, Pyleecan is used to maximize the fundamental torque while minimizing first torque harmonic of a machine. The machine is a 12-slot/8-pole three-phase PM motor with concentrated winding. Design variables are rotor magnet width between 0.3927 and 0.7775 radians, and stator slot opening width between 0.0628 and 0.3142 radians. No constraint is imposed for this example. For the time being, Pyleecan can only handle minimization problems, so that the maximization is treated as a minimization of the opposite of average torque.

The solver takes 20 hours to solve the optimization problem using NSGA-II with 100 generations of 20 individuals on a single 2.5GHz core. Each simulation has 32 timesteps and a mesh containing 5500 elements and 3000 nodes and is computed with FEMM. The following graph shows the fitness values for each individual. Fig. 20 shows that the algorithm converges to a Pareto front in the bottom left-hand corner.

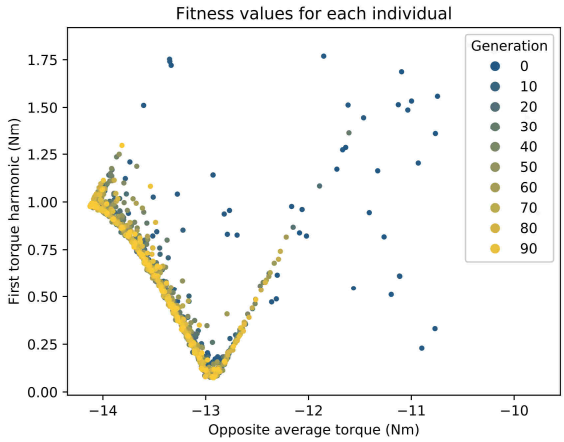

Fig. 20. Individuals in the fitness space

Fig. 21 shows the original topology, the one that maximizes the average torque and the one that minimizes the first torque harmonic:

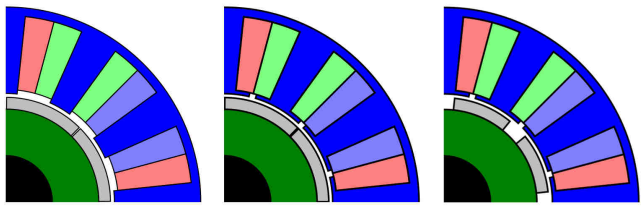

Fig. 21. Initial machine (left), best topologies for first objective (middle) and for second objective (right)

H. New electrical machine topologies

Pyleecan was initially created to focus on radial flux rotating machines, but OOP formalism can be used to extend it to axial flux machines, as well as linear machines.

In terms of radial flux machines, other topologies that could be modelled with Pyleecan include line start permanent magnet synchronous machines, spoke-type PMSM, tooth winding induction machine, brushless doubly-fed induction machines, as well as flux switching machines. Pyleecan could also be used for the design optimization of magnetic bearings and magnetically geared electrical machines.

V. PYLEECAN ROADMAP

The development of Pyleecan new features are discussed on Github issue pages. Here are few interesting directions to investigate:

- GUI extension to more complex machines, especially axial flux machines and linear machines
- Loss calculation models of magnets and laminations, based on magnetic field obtained on FEMM mesh
- Electrical Equivalent Circuit modeling for voltage driven simulation
- Coupling to Elmer and GetDP to perform magneto-harmonic analysis of induction machines, and 3D magnetostatic simulation of 3D machines (e.g. axial flux, claw pole alternators)
- Import/Export with third party commercial software such as Ansys Maxwell, Altair Flux, Jsol Jmag
- Electromagnetic design application modules (e.g. flux linkage maps, MTPA/MTPV, torque/slip curve)
- Online documentation with Jupyter notebook tutorials and videos.

VI. CONCLUSION

It has been shown that the geometry modeler of Pyleecan enables to create efficient coupling with 2D and 3D mesher software thanks to its OOP structure. The current state of Pyleecan already enables to simulate several complex topologies (e.g. complex winding, uneven slot types, multiple rotor and stators) and to solve global optimization problems using the NSGA-II algorithm. Moreover, the open source Apache license enables all PhD students and R&D engineers in electrical engineering to use it, even commercially, to investigate and optimize new topologies. Finally, the work on any new topology or coupling is automatically capitalized for all the community which is a significant gain in reproducible science and research efficiency.

VII. REFERENCES

[1] P. Bonneel, J. Le Besnerais, R. Pile and E. Devillers, "Pyleecan: An Open-Source Python Object-Oriented Software for the Multiphysic Design Optimization of Electrical Machines," *2018 XIII International Conference on Electrical Machines (ICEM)*, Alexandroupoli, 2018, pp. 948-954.

[2] FEMM (Finite Element Method Magnetics) by D. Meeker. [Online]. Available: http://www.femm.info

[3] J. Keränen *et al*., "Efficient Parallel 3-D Computation of Electrical Machines With Elmer," in *IEEE Trans. Magn.*, vol. 51, no. 3, pp. 1-4, 2015.

[4] OneLab, Open Numerical Engineering LABoratory [Online]. Available: http://onelab.info/

[5] KARBAN, Pavel, MACH, František, KŮS, Pavel, *et al.* Numerical solution of coupled problems using code Agros2D. *Computing*, 2013, vol. 95, no 1, p. 381-408.

[6] TAYLOR, Ross. Pyflux: An open source time series library for Python. 2016. https://pyflux.readthedocs.io/en/latest/

[7] SALOME," The open source integration platform for numerical simulation". https://www.salome-platform.org/

[8] P. Bonneel, et al. (2020, February 28). Eomys/pyleecan: ICEM2020 (Version v0.1-ICEM20). Zenodo. http://doi.org/10.5281/zenodo.3691866

[9] MO, Lihong, QUAN, Li, ZHU, Xiaoyong, *et al.* Comparison and analysis of flux-switching permanent-magnet double-rotor machine with 4QT used for HEV. *IEEE Transactions on Magnetics*, 2014, vol. 50, no 11, p. 1-4.

[10] F.-M. De Rainville *et al.*, "DEAP: A Python Framework for Evolutionary Algorithms" in *Companion Proceedings of the Genetic and Evolutionary Computation Conference, 2012.*

[11] K. Deb, *et al.*, "A fast and elitist multiobjective genetic algorithm: NSGA-II," in IEEE Transactions on Evolutionary Computation, vol. 6, no. 2, pp. 182-197, April 2002.

VIII. BIOGRAPHIES

P. Bonneel graduated in 2014 from the "Ecole Nationale Supérieure des Sciences Appliquées et de Technologie" of Lannion (ENSSAT) in signal analysis, computing science and electronics. After a first experience in software development in speech synthesis at Voxygen, he currently works in EOMYS ENGINEERING as a software development engineer.

J. Le Besnerais, following a M.Sc. specialized in Applied Mathematics (Ecole Centrale Paris, France) in 2005, made an industrial PhD thesis in Electrical Engineering at the L2EP laboratory of the Ecole Centrale de Lille, North of France, on the reduction of electromagnetic noise and vibrations in traction induction machines with ALSTOM Transport. In 2013, he founded EOMYS ENGINEERING, where he currently works as an R&D engineer on eNVH analysis and reduction.

Emile Devillers has done an industrial PhD thesis at EOMYS ENGINEERING (Lille, France) and L2EP laboratory of the Ecole Centrale de Lille, North of France and graduated in 2018. He is now researcher and software developer at EOMYS, working on eNVH reduction.

C. Marinel has finished a M.Sc. degree in High Performance Computing and Simulation (University of Lille, France) in 2019. He is now software developer for MANATEE and Pyleecan at EOMYS ENGINEERING.

Raphaël Pile received a M.Sc. degree in Aerospace Engineering from ISAE-Supaéro, Toulouse, France and a M.Sc. degree in Fundamental and Applied Mathematics from Université Paul Sabatier, Toulouse, France, both in 2017. He is now working on an industrial PhD thesis at the L2EP laboratory (Univ. Lille, France) and LSEE laboratory (Univ. Artois, France) with the company EOMYS ENGINEERING. His PhD thesis focuses on the numerical methods to perform magneto-mechanical coupling for noise and vibration study of electrical machines.