ReadSco: An Open-Source Web-Based Optical Music Recognition Tool

Antonio Ríos-Vila, Jorge Calvo-Zaragoza, David Rizo, Jose M. Iñesta
Department of Software and Computing Systems
University of Alicante, Alicante, Spain
{arios,jcalvo,drizo,inesta}@dlsi.ua.es

***Abstract*—We introduce READSCO, an open-source web-based community tool for Optical Music Recognition. READSCO aims to serve as a connection between research results and practical use. We describe the design decisions considered to both favors a rapid integration of new advances from the research field and facilitate the community's participation in its development. The project is still in its planning phase, so this work is a good opportunity to present the main idea and get direct feedback from other researchers.**

***Index Terms*—Optical Music Recognition, Open-Source Software, Web-based Application**

I. Introduction

Optical Music Recognition (OMR) [1] is still considered an open problem, especially in the most complex contexts. However, recent efforts, supported by modern machine learning techniques, have moved the research field from dealing with sub-problems (such as staff-line removal [2] or symbol classification [3]), to a state in which complete results are attainable [4]–[6].

Despite the above, many issues hinder taking OMR results from research to practice, such as the file management system, the development of appropriate visual interfaces, or exporting the recognition results to standard formats. In this work we introduce READSCO project, which aims to fill that gap from a community and scientific perspective. READSCO is a web-based open-source cross-platform OMR system, which covers the engineering processes required, while also seeking to be considered as a tool on which to integrate recent results of OMR research.

II. Background

Since the rise of general-purpose computing, there have been efforts to make computers capable of reading music. It is obvious that the industry is aware of this, and there exist commercial applications that attempt to provide this service as SmartScore [7] or PhotoScore [8]. However, these applications are proprietary and therefore do not pursue the philosophy of scientific and community development.

There also exists Audiveris [9], an open-source desktop-based system for OMR. This system is the one that most closely resembles our intention; however, it has not established itself as a community item. We aim to build an open-source tool that can be shared by the research community and enables a rapid integration of recent developments. At the same time, we want to provide a useful tool for practitioners.

We take inspiration from two recent web-based applications: Rodan [10] and MuRET [11]. Rodan is a rather generic system, where users can specify their own OMR workflow from pre-defined steps. Our application architecture and OMR pipeline conformation are inspired by how it defines a "process" ("jobs" in Rodan). MuRET is a tool specifically designed for scholars, which covers all transcription phases from the manuscript source to the encoded digital content. It is designed as a technology-focused research tool, allowing different approaches to be used, as well as producing both the transcribed contents in standard formats and data for the study of the transcription process itself. In this case, we took its technology as a reference for developing the project, as well as an example of a user interface for this kind of applications.

III. ReadSco

In this section, we describe the technologies that build up READSCO. More details are provided in its public repository, available at https://github.com/OMR-Research/ReadSco.

A. System Architecture

From an architectural point of view, all technical requirements of the application must be considered:

- Offer both fast analyses without user intervention and step by step user-guided analyses
- Easy scalability
- Easy code maintenance and fast error detection
- Low coupling among developed services and independence among the technologies used to develop the application's functionalities
- Warrant the easy and secure community contribution and support

A microservices architecture seems to be the most suitable architecture to achieve those requirements. This architecture splits the application's logic in a set of programs, known as microservices, which focus on a specific task [12]. They are completely independent of one another. However, combining them allows the development and fulfillment of complex functionalities in a distributed way. One practical example of what microservices would be in a common application would be database access, file storage, or, in our case, a specific task or process which takes part in the OMR workflow. This architecture simplifies considerably the application scalability.

In the case of READSCO, the microservice architecture allows the fulfillment of our primary objectives. It makes it possible to separate OMR tasks in detached programs. Splitting up OMR into microservices by its function and coordinate them with an event management system allows us to easily control and manage them with a user interface. It also enables us to handle the substitution and exchange of them, as it is supposed in a collaborative model. That is, microservices architecture supports the pursued community contribution model. Contributions will be made to a particular set of microservices, and not to the whole application. Further complex errors and code conflicts in contributions will also be easier to handle in this architecture, which will be an enormous advantage in the future.

Once has been determined the convenience of this architectural model, the first blueprint of the application structure was designed, as represented in Figure 1.

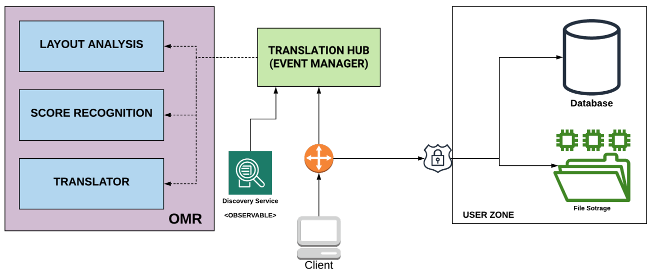


Fig. 1. Application structure blueprint

This blueprint is an ideal representation of the minimum services required for the application. There are two main areas: the first one is the OMR pipeline, which contains all the system's detection and recognition layers, as well as an encoding unit, called translator, between our internal format and common music representation formats, such as MEI, MusicXML or `**kern`. Each of these layers is represented by a microservice. These, in turn, are connected to an event manager. It is responsible for sending instructions to all of the OMR layers to do their tasks and send back the results for the user. To access this area there is an intermediate service, called Translation Hub, which receives all user's requests and processes them to prevent OMR layers from taking responsibilities they should not handle. It also allows asynchronous task handling without losing connection with the user.

The second zone corresponds to the user's data management. In this current state, there will be only a simple user account management, with its corresponding database, and a file system to store rendered scores and its corresponding OMR result. This zone may suffer several changes, as it hasn't been a priority during development and needs thorough research on document indexation and storage.

Generally speaking, access to all back-end services there is within an external application. Namely with a web browser, a mobile application or a command-line client that will be developed for advanced functionalities and uses. Access to microservices will only depend on the user's requests. So, all workload will be distributed in the application. Users will only use the required microservices for their specific requests. This strategy will avoid further overload issues in our servers.

Once the system is reviewed, there will be only one final issue to tackle: how to organize the application to support different technologies and programming languages. As the intention is to be a community-supported project, it will be desirable to not require a specific language knowledge for our contributors to their further software development or services for the application. Our proposal to alleviate this issue is the use of Docker [1] and Ansible [2] as deployment technologies.

Docker is a virtualization software that allows packaging applications for easy deployment on the Internet. The most remarkable advantage to READSCO is the possibility of virtualizing microservices developed in different programming languages and execute them, on a deployment scenario as in a development one, without the necessity of making compiler and interpreter installations or additional configurations. In turn, it exists the possibility to share all of these virtual systems, called images, publicly. Hence, a community user who wants to contribute to a specific service or to develop its own does not need to make any software configurations to set the system on his other development environment. He or she just needs Docker to execute the public image or locally build our system with few Docker commands.

Unfortunately, to coordinate numerous containers with the Docker API may result uncomfortable and tedious in both software development and application deployment scenarios. As our objective is to make these tasks as easy as possible to engage our contributors, it is required to count with some automation in what is called "container orchestration". For this issue, we chose Ansible as our orchestrator. Ansible is an IT automation tool developed by RedHat which allows full automation of application configuring and running. In our case, it will take the responsibility of pulling updated Docker images, verify any changes and automatically kill and restart all affected microservices. It works with configuration files, called playbooks, that execute a user-defined set of tasks. It is compatible with all operative systems and does not require any specific server architecture to work, which makes it a powerful tool to use in development and deployment workflows. Community contribution will be even easier: the user only needs to download our default Ansible configuration files and update them with the specific Docker images he wants to work with. It is not needed to rebuild and run manually any container, the tool will do it for him or her. In any case, to provide a comfortable developing experience and avoid common network troubles, READSCO organization will leave detailed instructions on how it is recommended to work with its technologies in the official application documentation. Furthermore, it is not necessary to push code, even it is

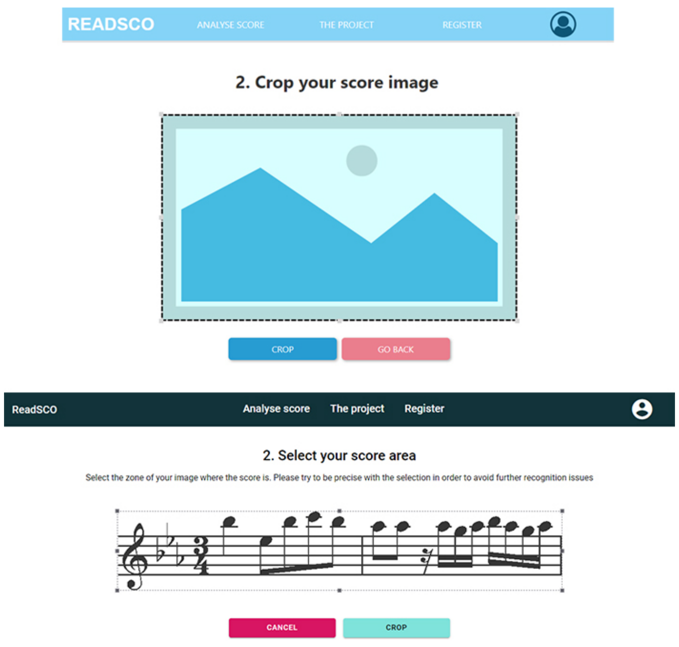
[1]https://www.docker.com
[2]https://www.ansible.com

suggested for error debugging and community updates. It will only be needed for an upgraded system or the new microservice Docker image to be pushed into the official READSCO Docker repositories.[3] Any contribution will be reviewed by the official READSCO organization, which will have the Ansible playbooks to easily include new updates and services into the official application servers.

B. Interface

READSCO is also an application allocated for practical user interaction. Therefore, we also focused our attention on bringing a comfortable User Experience. After analyzing the different user profiles which may interact with our application, we decided to guide its design on usability. The application is devised to be simple, direct and easy to use. Another feature taken into account in this process is the responsive layout, as it will be displayed on both computer screens and mobile devices. It should be emphasized that READSCO is intended to be deployed as a web software and as a native Android and iOS application. Therefore we are currently developing it with Ionic 4 and Angular 2 [13], which can fulfill this requirement.

In the current state of the application, users upload score images, select the area in which they are interested to be analyzed and receive the score in a standard coding format that can be rendered with Verovio [14].



Select the zone of your image where the score is. Please try to be precise with the selection in order to avoid further recognition issues

Fig. 2. READSCOHigh-Fidelity prototype for image selection functionality (above) and final interface implementation (below)

We are aware that OMR technologies do not provide perfect results, and they probably never will. However, in the current state of our project, we do not find it interesting to develop a correction interface but we rather follow an "I'm Feeling Lucky" approach. The user can download the recognition result in a standard format that can be opened by any score edition software, to correct the remaining errors. In the future, we will consider the possibility that these corrections can be done online, although a thorough investigation about the best interface to do this is still required.

[3]https://hub.docker.com/orgs/readsco/repositories

C. Optical Music Recognition

Our optical processing does not treat the document globally but instead divides it into bounded regions. In our case, the main recognition core works at a single-staff level. This is analogous to what happens in text recognition algorithms, where single-line level recognition is assumed [15].

Thus, we use the optical recognition model proposed by Alfaro-Contreras et al. [16]. This work is based on the use of deep neural networks that process a single-staff image and yields a sequence of tokens as a hypothesis. These tokens include both the recognized musical symbols and some information about the geometric relationship among them. This allows recognizing non-sequential structures such as chords or ligatures. In addition to its excellent performance, this approach is *holistic*, that is, the recognition is performed in a single step, thus facilitating its integration into the system. In our case, during the training of the neural network, we consider a strong data augmentation process to provide robustness against possible alterations of the input image [17].

To complete the recognition, we simply add two algorithms that operate before and after the staff-level recognition:

- *Staff segmentation*: there are several algorithms to divide a score into isolated staves, given the excellent reference that the staff lines represent. Specifically, we currently use the idea proposed by [18].
- *Notation Reconstruction*: the result provided by the optical recognition needs to be parsed to retrieve the actual music semantics. We convert the sequence of tokens to a sequence-based semantic representation (similar to `**kern`). This process is currently performed by a rule-based algorithm. We are however willing to explore other approaches such as those used in machine translation.
- *Encoding*: once the music content is captured, there is still the need for encoding it into a standard format. For this purpose, we built a hand-crafted parser that processes semantic sequences and exports them into MEI, MusicXML, and MIDI. The former is automatically retrieved for the recognition to be rendered by Verovio (see the previous section) and the rest of them are provided when requested.

At present, we leave the most complex levels of music notation structure (such as pianoform or polyphony) out of our targeted domain since it is not clear how to deal with it with state-of-the-art end-to-end technologies. However, the system is constructed to allow for an easy replacement of the underlying OMR operation. Therefore, any advance in the research field should be easy to integrate into READSCO.

As a summary, our current OMR pipeline is depicted in Figure 3. It should be noted that this is the pipeline that we initially assume in READSCO. As mentioned above, we intend that the tool allows integrating different approaches to OMR through a standard communication protocol among the different agents involved within the system.

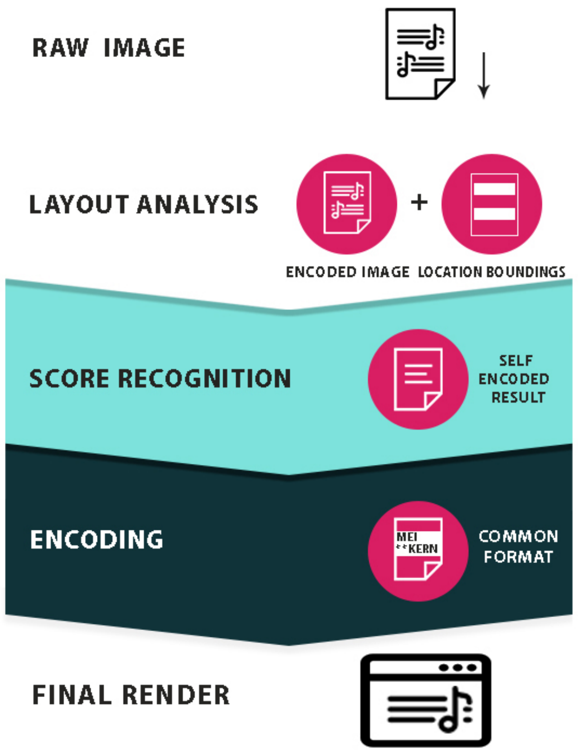


Fig. 3. ReadSco recognition pipeline from image upload to end-user visualization. Results produced by one system come as input data to the next one.

IV. Conclusions

The latest results in the OMR field make us optimistic about developing effective technology to solve the problem of automatically encoding the content of a music score image into a structured representation.

READSCO intends to establish itself as an open-source tool for which the community can be involved with the least possible effort. The main objective of our initiative is to provide a link between research and practice, as well as identifying new challenges and opportunities in this research field.

Acknowledgment

This work is supported by the Spanish Ministry HISPAMUS project TIN2017-86576-R, partially funded by the EU.

References

[1] Jorge Calvo-Zaragoza, Jan Hajic Jr., and Alexander Pacha. Understanding optical music recognition. *Computer Research Repository*, abs/1908.03608, 2019.

[2] Antonio-Javier Gallego and Jorge Calvo-Zaragoza. Staff-line removal with selectional auto-encoders. *Expert Systems with Applications*, 89:138–148, 2017.

[3] Alexander Pacha and Horst Eidenberger. Towards a universal music symbol classifier. In *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, volume 2, pages 35–36. IEEE, 2017.

[4] Jorge Calvo-Zaragoza and David Rizo. End-to-end neural optical music recognition of monophonic scores. *Applied Sciences*, 8(4), 2018.

[5] Arnau Baró, Pau Riba, Jorge Calvo-Zaragoza, and Alicia Fornés. From optical music recognition to handwritten music recognition: A baseline. *Pattern Recognition Letters*, 123:1–8, 2019.

[6] Alexander Pacha, Jorge Calvo-Zaragoza, and Jan Hajič jr. Learning notation graph construction for full-pipeline optical music recognition. In *20th International Society for Music Information Retrieval Conference (in press)*, 2019.

[7] Musitek. Smartscore x2. http://www.musitek.com/smartscore-pro.html, 2017.

[8] Neuratron. Photoscore 2018. http://www.neuratron.com/photoscore.htm, 2018.

[9] Hervé Bitteur. Audiveris. https://github.com/audiveris, 2004.

[10] Andrew Hankinson. Optical music recognition infrastructure for large-scale music document analysis. McGill University, 2015.

[11] David Rizo, Jorge Calvo-Zaragoza, and José M. Iñesta. Muret: A music recognition, encoding, and transcription tool. In *5th International Conference on Digital Libraries for Musicology*, pages 52–56, Paris, France, 2018. ACM.

[12] Nicola Dragoni, Ivan Lanese, Stephan Thordal Larsen, Manuel Mazzara, Ruslan Mustafin, and Larisa Safina. Microservices: How to make your application scale. In Alexander K. Petrenko and Andrei Voronkov, editors, *Perspectives of System Informatics*, pages 95–104, Cham, 2018. Springer International Publishing.

[13] Ionic framework official web. https://ionicframework.com/.

[14] Laurent Pugin, Rodolfo Zitellini, and Perry Roland. Verovio: A library for engraving mei music notation into svg. In *ISMIR*, pages 107–112, 2014.

[15] Baoguang Shi, Xiang Bai, and Cong Yao. An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition. *IEEE transactions on pattern analysis and machine intelligence*, 39(11):2298–2304, 2016.

[16] Mara Alfaro-Contreras, Jorge Calvo-Zaragoza, and Jose M. Iñesta. Approaching end-to-end optical music recognition for homophonic scores. In *9th Iberian Conference on Pattern Recognition and Image Analysis*, 2019.

[17] Jorge Calvo-Zaragoza and David Rizo. Camera-PrIMuS: Neural End-to-End Optical Music Recognition on Realistic Monophonic Scores. In *Proceedings of the 19th International Society for Music Information Retrieval Conference, ISMIR 2018, Paris, France, September 23-27*, pages 248–255, 2018.

[18] Lorenzo Quirós, Alejandro Toselli, and Enrique Vidal. Multi-task layout analysis of handwritten musical scores. In *9th Iberian Conference on Pattern Recognition and Image Analysis*, 2019.