TERIIHOANIA JOAN HEIMANU

GUBSTEP

SCRIPT DE TRADUCTION TEXTUELLE PAR EXTRACTION SYNTAXIQUE

MOTS CLES :

PYTHON,
GUBSTEP
GOPIGO,
CHOREGRAPHIE,
DEXTER INDUSTRIES

IUT MONTPELLIER
2019

SOMMAIRE

RESUME

L'objectif de ce document sera de développer la réflexion autour du projet *Gubstep*[1]. En se basant sur le langage chorégraphique développé pour le robot GoPiGo[2] de Dexter Industries[3], nous verrons le cheminement de pensée afin de résoudre un problème simple : « *Faciliter la compréhension d'instructions chorégraphiques à l'aide d'un ou de plusieurs supports complémentaires* ». Pour cela, nous utiliserons l'aide du langage python[4] afin de générer les dits supports. Dans notre démarche, nous opterons pour un support textuel et imagé.

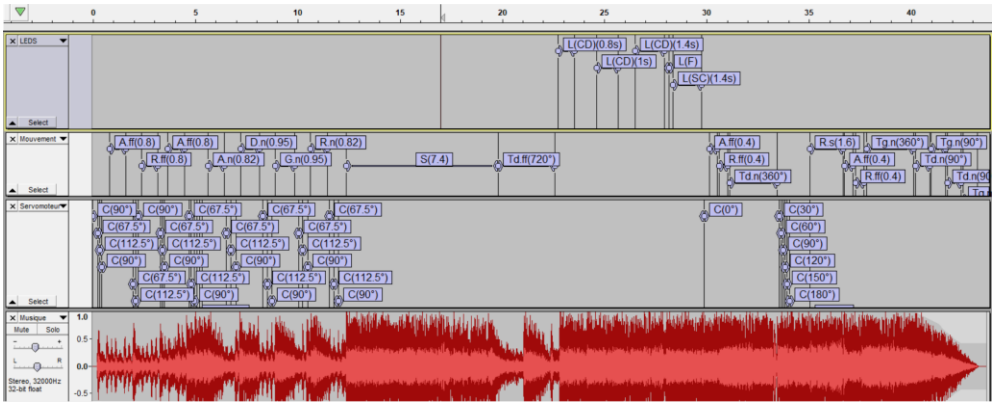
Nous utiliserons également la librairie Colorama[5] ainsi que les modules intégrés à l'installation Python. Nous disposerons comme source, l'exportation CSV de marqueurs extraits du logiciel Audacity[6] (via sa fonctionnalité de *Scripting*[7]) qui correspondent à des instructions chorégraphiques réparties sur une période donnée sur une piste musicale déterminée.

PREAMBULE

PROPOS D'INTRODUCTION LIMINAIRE

Le script de génération textuelle par séquençage, fragmentation ou traduction syntaxique est un algorithme permettant de traduire des instructions chorégraphiques. Ces instructions répondent à la syntaxe du langage chorégraphique décrit par le document de spécification des besoins du projet *Gubstep*.

L'idée est de faciliter et aider la compréhension de ces instructions en y apportant un support complémentaire. En effet, en observant la représentation suivante :



On peut déjà conclure qu'elle est relativement difficile à comprendre bien qu'elle soit structurée et ordonnée temporellement. Ainsi, il nous faut réfléchir à un moyen nous permettant de faciliter ou aider à la compréhension des instructions précisées sans pour autant perdre leur sens.

Nous devrons également réfléchir aux spécificités techniques, au matériel et objets requis pour la réalisation de cet algorithme et de ces supports, leur forme et leur contenu. Il nous faudra également prescrire un guide d'utilisation puis rédiger une documentation précise si nécessaire.

REFLEXION

METHODOLOGIE, CONTENU, CARACTERISTIQUES

LE LANGAGE

Afin de développer un algorithme, il est nécessaire de déterminer un langage de programmation préalable dans lequel le programme sera écrit. Dans un souci d'efficacité et de simplicité, l'utilisation de *Python* devient rapidement une évidence. La syntaxe et le fonctionnement de celui-ci étant relativement simples, son utilisation en devient d'autant plus pertinente.

Pourtant, sa simplicité n'est qu'une des raisons pour laquelle choisir ce langage. La multiplicité de librairie qu'elle détient, sa richesse en matière de communauté lui procure également une base de connaissance et de support indéniable.

Néanmoins, l'argument principal est bien entendu le *Scripting* qu'offre le logiciel Audacity en commun avec le langage Python. Etant donné que le projet audio et les instructions chorégraphiques sont stockés via ce logiciel, il est important de le prendre en compte. Par fortune, Audacity permet d'extraire, consulter et manipuler des données au sein d'un fichier Audio via un script, algorithme ou programme depuis un script Python ou Perl.

Par manque de temps et par simplicité, nous ignorerons Perl et nous concentrerons sur Python et notamment la commande *Export Labels*. Cette commande permet d'exporter, comme son nom l'indique, des marqueurs (*labels* en anglais). Idéal afin de collecter les marqueurs (et instructions chorégraphiques) en vue de les utiliser, les modifier et les extraire.

QUELS SUPPORTS ?

A présent que le langage a été sélectionné, ne reste plus qu'à déterminer les supports qui seront générés par l'algorithme afin de remplir son objectif. En observant les instructions chorégraphiques, il est facile de noter une chose : *On ne peut les comprendre au premier coup d'œil*. En effet, leurs formes et leurs notations sont très difficiles à la compréhension sans documentation préalable.

Afin de faciliter cette compréhension, le support qui peut être à considérer est bien entendu un support écrit dans un langage facilement compréhensible. L'objectif premier du langage chorégraphique développé pour le projet *Gubstep* était de pouvoir spécifier une suite d'instructions rapidement et simplement sur l'interface du logiciel Audacity. Il n'a donc pas pour objectif principal une rapide compréhension. On entend facilement par *langage facilement compréhensible* un langage connu et utilisé par la grande majorité. L'un des langages les plus utilisés est bien entendu le *Français*.

Ainsi, nous avons notre premier support : **Un support écrit en Français**.

Néanmoins, ce support n'est pas suffisant. En effet, malgré qu'il soit facilement compréhensible, clair et précis, ce support manque sévèrement de synthétisation. Les phrases sont nombreuses et limitent une compréhension simple et rapide des instructions chorégraphiques. Il nous faut donc un second support.

Afin de représenter une suite d'instructions sur une frise temporelle de manière claire tout en favorisant une compréhension rapide, le support le mieux adapté pour cela est une *illustration*. Une image, un dessin ou encore un graphe. Cette représentation de suites d'instructions est un *Diagramme d'activité* qui image l'enchaînement d'étapes prédéterminées. Nous utiliserons donc ce diagramme d'activité pour imager l'enchaînement chorégraphique du projet en se basant sur la définition d'UML[8].

Nous avons enfin notre second support : **Un diagramme d'activité**.

ECRIT

EXTRACTION

Afin de pouvoir convertir la syntaxe des instructions, il nous faut la connaître et interpréter une logique syntaxique. Pour cela, nous devrons découper les instructions chorégraphiques en plusieurs parties contenant chacune une valeur, chacune décrivant une information sur le mouvement chorégraphique correspondant. Il nous faut donc passer l'instruction au travers d'une fonction de désassemblage selon une logique précise. Cette logique est dictée par la notation et la syntaxe des instructions.

Aussi, à partir des valeurs extraites, nous pourrons construire une phrase bien construite selon la syntaxe grammaticale *Française*. Nous prendrons en exemple de la syntaxe des instructions d'avance, de recul et de tour dont la notation est la suivante :

$$Action.Vitesse(\boldsymbol{Durée\ en\ seconde});$$

Note : *Les instructions sont fournies avec des marqueurs temporels (*temps à laquelle l'instruction débute et se fini*).*

On entend par *Action* la lettre correspondance à l'action du robot. Soit *A* pour avancer, *R* pour reculer et *G/D* pour tourner, respectivement à gauche et à droite selon la syntaxe spécifiée par le projet *Gubstep*. Depuis cette observation, on peut ainsi extraire notre première valeur : **L'action effectuée** par simple découpage de l'instruction.

Techniquement parlant, nous utiliserons un *array* et *spliterons* l'instruction en se basant sur la position du caractère « . ». Ainsi, nous obtenons deux éléments :

- $Action$ que nous pouvons ainsi stocker dans une variable,
- $Vitesse(\boldsymbol{Durée\ en\ seconde})$; que nous pouvons utiliser afin de poursuivre l'analyse de l'instruction.

Note : *Les instructions C, L et S n'ont pas de valeur vitesse. Leur notation est donc uniquement telles que Action(Valeurs). Il est donc important de séparer leur analyse des autres instructions.*

Nous obtenons donc l'instruction $Vitesse(\boldsymbol{Durée\ en\ seconde})$; que nous devons encore analyser pour en extraire les données. Nous pouvons observer ici rapidement. La *Vitesse* et la *Durée en seconde* sont séparées par un caractère « (». Il ne nous reste plus qu'à répéter l'opération faite plus haut afin d'obtenir :

- $Vitesse$que nous pouvons ainsi stocker dans une variable,
- $\boldsymbol{(Durée\ en\ seconde)}$; que nous pouvons utiliser afin de poursuivre l'analyse de l'instruction.

Idéalement, l'analyse de l'instruction s'arrêterait ici. Néanmoins, en tenant compte des autres notations telle que l'instruction de mouvement du servomoteur : $C(\boldsymbol{Angle°})(\boldsymbol{Durée\ en\ seconde})$, il apparaît qu'arrêter l'analyse ici serait contre-productif, étant donné qu'il est possible que la notation se poursuit.

On notera bien évidemment que les seules valeurs stockées sont uniquement des valeurs numériques (*pouvant être des entiers ou des nombres décimaux*).

Afin d'extraire efficacement les données, il nous faut trouver un paterne logique. Ici, nous observons bien deux valeurs chacune entre parenthèses l'une à côté de l'autre. Un paterne logique est trouvée. La solution serait alors de passer les instructions restantes, c'est-à-dire entre parenthèses, dans une fonction d'analyse spécifique prenant en paramètre les données telles que $(Valeur\ 1)\ ...\ (Valeur\ n)$ et qui retourne en sortie un *tableau* de valeurs numériques.

Ainsi en divisant l'instruction par caractère «)(», nous obtenons :

- $(Valeur\ 1$
- …
- $Valeur\ n)$

Il ne nous reste plus qu'à supprimer les caractères « (» au début de la première valeur et «) » à la fin de la dernière valeur pour convertir les données en valeurs numériques et les stocker au sein du tableau qui sera retourné à la fonction chargée de désassembler pour convertir l'instruction.

Cette procédure de conversion, en tenant compte des spécificités syntaxiques de chaque notation, peut ainsi être faite sur chaque type d'instruction depuis la liste fournie. Ne reste plus qu'à envoyer les données extraites (*Type d'action, vitesse* s'il y en a une, *durée* et *angle* s'il y en a un) à une fonction de traduction.

TRADUCTION

Afin d'établir une syntaxe compréhensible en Français, nous devons établir une phrase grammaticalement correcte et compatible avec la notation chorégraphique. En synthétisant cette dernière, on peut extraire, depuis la fonction d'analyse, les données suivantes :

- Type d'action (*essentielle*),
- Vitesse de déroulement de l'action (*pouvant être ignorée selon l'action*),
- Durée de l'action (*essentielle*),
- Angle d'orientation (*pouvant être ignorée selon l'action*).

En tenant compte des actions essentielles, nous pouvons ainsi former une structure principale telle que : *Action Durée* avec une structure comprenant des éléments pouvant être absents mais qui ne changeront pas le sens de la phrase d'un point de vue syntaxique telle que :

[***Sujet***][***Action***] *pendant* [***Durée***] $seconde(s)$[(***Vitesse***)] [à (***Angle d'orientation***) $degré(s)$].

Note : *Nous pouvons extraire le Sujet depuis le type d'action étant donné qu'une action peut être faite par, soit le robot, soit le servomoteur ou la LED.*

Note : *Les éléments avec* [texte (valeur) texte] *sont des éléments qui n'afficheront* texte *que lorsque* valeur *est présente. Sinon,* texte *ne sera pas affiché.*

Nous pouvons ainsi traduire chacun des éléments :

- Type d'action :
  [Le robot/Le servomoteur/La Led] [avance, recule, tourne à gauche/tourne à droite],
- Durée : *[valeur] seconde(s)*,
- Vitesse de déroulement de l'action :
  [très rapidement/rapidement/normalement/lentement/très lentement],
- Angle d'orientation : *à [valeur] degré(s)*.

Ce qui nous donne, par exemple, pour l'instruction $G.f(2)$:

- Type d'action : (G) *Le robot tourne à gauche*
- Durée : *2*
- Vitesse de déroulement de l'action : *rapidement.*

Cette procédure peut ainsi être utilisée à toutes les instructions en général en traitant les cas particuliers et en respectant la syntaxe de chaque action.

OPTIMISATION, AMELIORATION

Jusqu'à maintenant, nous disposons donc de deux fonctions :

- Analyse *chargée de l'extraction des valeurs*,
- Traduction *chargée de la conversion des valeurs en phrase*.

Comme annoté plus haut, nous savons que la notation des instructions peut varier selon le type d'action effectué. Il est opportun de différencier leur traitement d'analyse.

Une solution possible serait d'utiliser deux autres fonctions appelées dans la fonction d'analyse qui se chargeraient de traiter, en fonction du type d'action. Nous savons grâce au document qu'une action n'est représentée que par une ou deux lettres, donc, il nous suffit de vérifier si le second ou le troisième caractère (donc après la notation du type d'action) est un « . » ou autre chose. Cela nous permet ainsi de différencier leur traitement et extraire les données en fonction.

Il est également opportun de fournir une certaine modularité au niveau de la construction du texte en français. Il serait gréé de permettre à l'utiliser de modifier le paterne de construction de la fonction de traduction en stockant le paterne dans un fichier texte (de configuration par exemple) ou en demandant à l'utilisateur d'entrée un paterne modifier celui existant :

[***Sujet***][***Action***] *pendant* [***Durée***] *seconde*(*s*) [(***Vitesse***)] [à (***Angle d'orientation***) *degré*(*s*)].

Ainsi, la fonction de traduction ne ferait qu'un *remplacement* de chaque section (*Sujet, Action, etc.*) par sa valeur correspondante (*Le robot, avance, etc.*) et si l'utilisateur le souhaite, il pourrait ajouter changer le paterne. Par exemple :

[***Sujet***] [***Action***] durant [***Durée***] *seconde*(*s*) [en allant (***Vitesse***)] [à (***Angle d'orientation***) *degré*(*s*)].

Egalement, il peut être opportun de fournir à l'utilisateur une langue de conversion et ainsi, selon la langue, utilisé un paterne différent (*pouvant être fournis par l'utilisateur*) et créer un fichier de configuration dans lequel le script irait piocher toutes les données de substitution. Ce fichier (*translate.json*) d'un format JSON pourrait ainsi contenir des données stockées comme, par exemple :

```
{
    'English' :
    {
        'default_pattern':
            '[Sujet][Action] during [Durée]
            [(Vitesse)  second(s)]
            [at (Angle d'orientation)  degree(s)].',
        'Sujet' :
            {
                'Le robot': 'The robot',
                'La LED': 'The LED',
                …
            },
        'Action' :
            {
                'avance': 'Move forward',
                'recule': 'Move backward'
                …
            },
        …
    }
}
```

La fonction irait ainsi sélectionner les valeurs de traduction correspondantes en fonction de la langue choisie, permettant une grande modularité au programme.

DIAGRAMME D'ACTIVITE

EXTRACTION DE DONNEES

Afin de faciliter notre démarche, il nous est possible d'utiliser les phrases générées par la fonction de traduction textuelle que nous avons décrite plus haut. Celle-ci nous donne ainsi un échantillon des instructions chorégraphiques textuelles qui seront précisées au sein de notre diagramme d'activité.

Nous pouvons ainsi considérer que nous avons en entrée, *les phrases générées en Français* au préalable à la sortie de la fonction de traduction. Nous disposons également des marqueurs temporels des phrases ainsi générées pour chaque instruction.

STRUCTURE

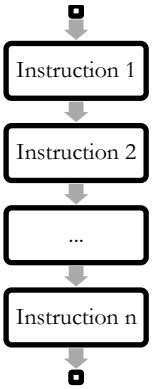
Nous nous baserons sur la structure d'un diagramme d'activité par l'ICOO[9][1]. Dans leur définition, un diagramme d'activité (*page 3 - Introduction*) comprend :

- Des activités *représentées par des encadrés contenant un texte description d'une étape d'exécution, d'une action ou d'un état-activité*,
- Des transitions *représentées par des flèches unidirectionnelles indiquant l'enchaînement d'exécution des activités.*

A partir de cette définition, des éléments que le diagramme contiendra ainsi que des données extraites que nous disposons, nous pouvons conclure que le diagramme sera une succession d'étape. Chacune de ces étapes seront des instructions chorégraphiques, des pas, mouvements ou actions, reliées par des transitions les unes à la suite des autres de manière ordonnée en fonction de leur marqueur temporel.

MISE EN FORME

Comme précisé par l'ICOO dans son document[9], la mise en forme du diagramme d'activité du projet se fera telle que :



Considérant le premier ◘ comme le début de la chorégraphie et le dernier ◘ comme étant la fin de la chorégraphie.

[1] Initiation à la Conception Orientée Objet

INTEGRATION TECHNIQUE

A partir de notre mise en forme et des données que nous disposons, ne nous reste plus qu'à produire le support imagé. Pour cela, il existe de nombreuses librairies qui permettent. Nous utiliserons également la librairie *textwrap*[10] afin de gérer l'affichage multi-ligne du texte. Egalement *Pillow*[11] qui intègre les fonctions :

- Image *qui permet la gestion d'une image dans son ensemble (création ou altération des valeurs globales de l'image comme sa taille),*
- ImageDraw *qui permet de manipuler le contenu de l'image, ajouté du texte, des lignes, etc.*
- ImageFont *qui permet de modifier la police d'écriture utilisée par* ImageDraw.

Parmi ces fonctions sont incorporées des fonctionnalités permettant de mesurer la taille (en pixels) du texte, tracé précisément des formes géométriques simples. Considérant que nous générons l'image au cours de la fonction de traduction ou stockons les instructions dans un tableau qui sera parcouru de nouveau plus bas, avant de pouvoir débuter le tracé de l'image contenant le diagramme, il nous faut la créer.

Afin de la créer, il nous faut tout d'abord connaître sa taille (largeur et hauteur). Pour cela, il nous faut au préalable calculer la taille que prendra le texte. Comme précisé dans la mise en forme, on estime que la mise en page sera verticale. On peut considérer une largeur statique de 1170 pixels correspondant à la largeur d'une taille A4.

Afin de calculer la hauteur, il nous faut prendre en compte divers éléments (Note : *On rappelle que les encadrés sont des activités*) :

- Les marges de page (qui correspondent à l'espace séparant la bordure haute de la page au premier élément du diagramme et la bordure basse au dernier élément) *que nous définirons à 50 pixels*,
- Les marges d'encadré intérieures (qui correspondent à l'espace séparant les bordures de l'encadré au texte) *que nous définirons à 10 pixels*,
- Les mages d'encadré extérieures (qui correspondent à l'espace séparant un encadré d'un autre) *que nous définirons à la hauteur en pixels des flèches transitoires,*
- Les marges de texte (qui correspondent à l'espace séparant chaque ligne de texte avec celle du bas et du haut) *que nous définirons à 0 pixels étant donné que la police d'écriture l'inclus déjà*,
- La taille du texte *que nous définirons à 25 pixels*,
- La taille des encadrés *est relative au texte, la taille du texte et les marges d'encadré intérieures*,
- La taille des flèches transitoires *que nous définirons à 100 pixels*.

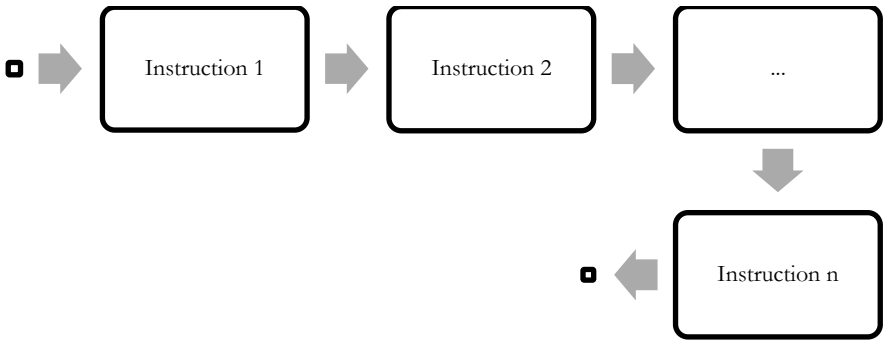
A partir de cela et grâce aux fonctions et librairies décrites plus haut, il ne nous manque plus que de tracer l'image et de l'afficher à l'utilisateur.

OPTIMISATION, AMELIORATION

En observant la génération de nos images, une observation peut rapidement faire son apparition : La taille. En effet, lorsque l'on génère le diagramme de très grand nombre d'instructions, le diagramme apparaît en très petit.

Une optimisation et une amélioration pertinente serait de pouvoir optimiser la répartition de taille du diagramme sur la page. Deux options complémentaires se présent.

La première option serait de remettre en forme la structure du diagramme telle que :



Cela économiserait de la place mais ne résoudrais pas le problème en cas de suites d'instructions extrêmement longues.

La seconde option serait de découper le diagramme en section. Chaque section étant une page A4. L'algorithme calculerait au préalable la taille que ferait le document dans son ensemble, diviserait le tout en section de 827x1170 pixels (A4) en découpant le tableau d'instructions selon les calculs des hauteurs de chaque section.

ANNEXE

PROJET GUBSTEP

Tiré d'un article du site joan-teriihoania.fr

Le projet Gubstep s'est déroulé à l'IUT Montpellier sous la direction de Madalina Croitoru, professeure de l'université de Montpellier. Le projet avait pour objectif la réalisation d'une chorégraphie qui sera effectuée par un robot GoPiGo (comme spécifié plus bas). Ensuite, la chorégraphie ainsi conçue par une équipe, est incrémentée en code par une autre équipe qui reçoit le document de spécification des besoins fournis par la première équipe.

L'équipe de conception de la chorégraphie est composée de :

- Joan Heimanu Teriihoania
- Erwan Le Goff
- Pierre-Julien Fournie
- Thibault Odor

RESSOURCES

Afin de mener la conception et la réalisation de ce projet à terme, l'équipe a eu besoin de :

- Matériel robotique, le starter pack du robot GoPigo de Dexter Industries dont les références sont disponibles sur le document de spécification des besoins,
- Logiciel Audacity pour la représentation graphique et précise du déroulement des instructions chorégraphiques,
- Langage Python pour la réalisation d'un script de génération textuelle par séquençage syntaxique.

Grâce à ces ressources, ont été produites :

- Des données de mesure de vitesse de déplacement, de tour et de rotation du robot GoPiGo,
- Un langage chorégraphie adapté au logiciel Audacity,
- Un script python permettant la traduction du langage chorégraphique en français.

Qui sont disponibles au sein du document de spécification des besoins (*disponible dans l'article*).

BIBLIOGRAPHIE

DOCUMENTS, ARTICLES ET TEXTES DE REFERENCE

[1] "Gubstep – Joan Heimanu Teriihoania." .

[2] "GoPiGo3 is a Raspberry Pi Robot Car for Learning Coding." [Online]. Available: https://www.dexterindustries.com/gopigo3/. [Accessed: 16-Dec-2019].

[3] "Dexter Industries." [Online]. Available: https://www.dexterindustries.com/. [Accessed: 16-Dec-2019].

[4] "Welcome to Python.org." [Online]. Available: https://www.python.org/. [Accessed: 16-Dec-2019].

[5] "colorama · PyPI." [Online]. Available: https://pypi.org/project/colorama/. [Accessed: 16-Dec-2019].

[6] "Audacity.fr - Télécharger Audacity pour PC et Mac," *Audacity.fr - Télécharger Audacity pour PC et Mac*. [Online]. Available: https://audacity.fr/. [Accessed: 12-Nov-2019].

[7] "Scripting - Audacity Manual." [Online]. Available: https://manual.audacityteam.org/man/scripting.html. [Accessed: 16-Dec-2019].

[8] "Unified Modeling Language (UML) | Activity Diagrams - GeeksforGeeks." [Online]. Available: https://www.geeksforgeeks.org/unified-modeling-language-uml-activity-diagrams/. [Accessed: 16-Dec-2019].

[9] "Diagramme d'activités," p. 25.

[10] "textwrap --- Encapsulation et remplissage de texte — Documentation Python 3.8.1rc1." [Online]. Available: https://docs.python.org/fr/3/library/textwrap.html. [Accessed: 18-Dec-2019].

[11] "Manipulation d'images — The Hitchhiker's Guide to Python." [Online]. Available: https://python-guide-pt-br.readthedocs.io/fr/latest/scenarios/imaging.html. [Accessed: 18-Dec-2019].