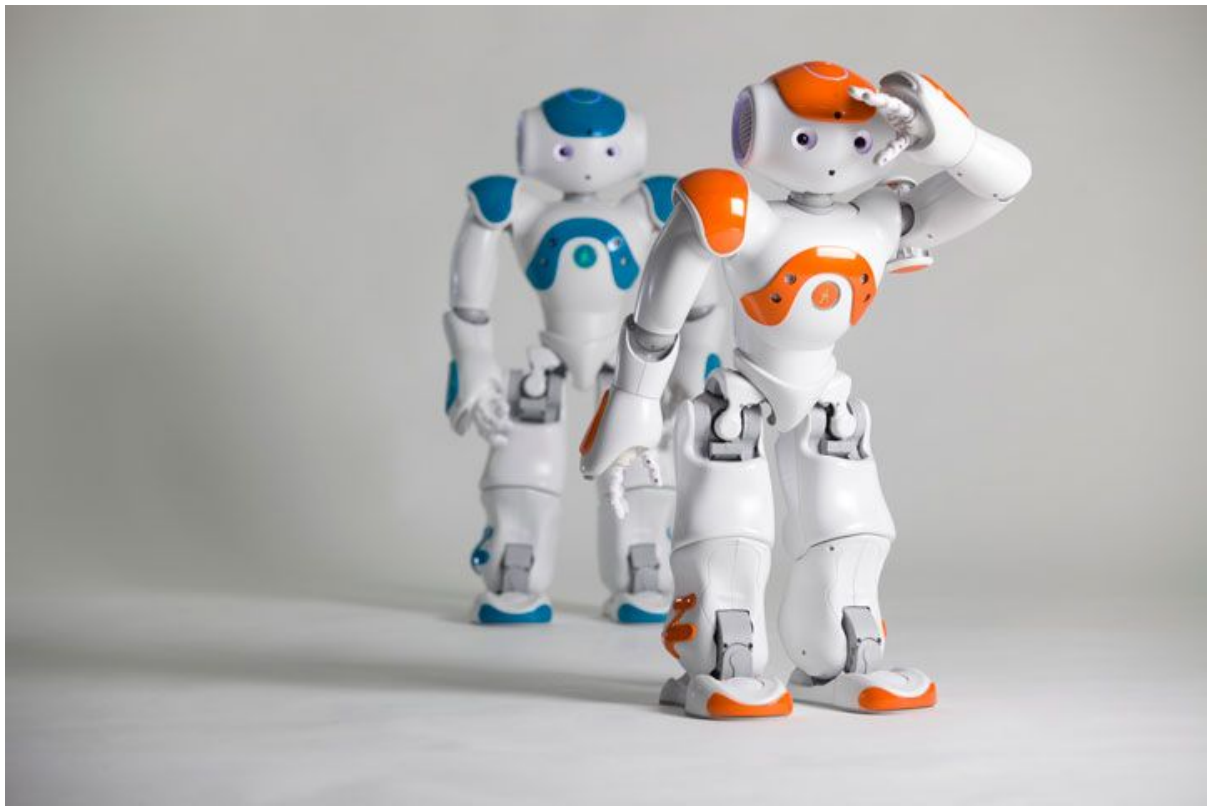


PROJET DE FIN D'ÉTUDE :
NAO, un traitement contre l’autisme infantile.

Rapport Final.

RASAMOELISON Lawrencio, Demailly Benjamin
2017
EISTI, ETIS

Remerciements

Nous tenons tout d'abord à remercier Ghiles MOSTAFAOUI, qui nous a permis de réaliser ce projet, pour sa confiance, son encadrement et pour nous avoir guidé tout au long de ce projet.

Un grand merci à Eva ANSERMIN, pour son aide précieuse et ses conseils tout au long de ce projet. Merci pour son accompagnement, son volontariat et sa disponibilité.

Un merci à Arnaud BLANCHARD, pour son apport technique sur ce projet, qui a pris le temps de nous aider.

Un merci également à l'ensemble de l'équipe de l'E.T.I.S. de l'Université de Cergy-Pontoise pour son chaleureux accueil.

Merci à Nga Nguyen de nous avoir soutenu durant ces six derniers mois.

Tables des matières

Introduction

Les troubles autistiques sont des troubles du développement humain qui limite les capacités sociales d'une personne. La personne est alors restreint dans sa communication, répétant les mêmes gestes. Les symptômes peuvent être détecté très tôt chez l'enfant. On parle aussi de spectre du trouble autistique, car possédant de multiples origines. Bien que la recherche avance, la compréhension de l'autisme n'est pas encore totalement aboutie. Et les chercheurs peinent à trouver des solutions pour sociabiliser les personnes atteintent du handicap. Plusieurs méthodes éducatives sont aujourd'hui proposé, mais le manque de recul et de moyen ne permettent pas de proposer de solution globale et efficace.

En août 2014, lors de la 23e édition du Symposium international sur la communication interactive entre le robot et l'Homme, les résultats d'une étude américaine menée par des chercheurs de la Viterbi School of Engineering suggèrent que le recours à des robots éducatifs pourraient améliorer la prise en charge des enfants atteints de troubles du spectre autistique. Ces résultats, très encourageants, sont en accord avec d'autres études similaires et incitent les professionels du domaine de la santé et de l'éducation à avoir recours à la thérapie robotique pour lutter contre l'autisme, en particulier chez les jeunes enfants.

Contexte

Présentation de l’E.T.I.S.

L’ E.T.I.S. (Equipes Traitement de l'Information et Systèmes) est une unité commune à l’Université de Cergy-Pontoise, le CNRS et l’ENSEA. ETIS est rattaché principalement à l'Institut des sciences informatiques et leurs interactions (INS2I). Elle compte environ 50 personnels permanents (enseignants-chercheurs et personnels support) et accueille une cinquantaine de doctorants et post-doctorants.

Principalement rattaché à l’Institut des Sciences Informatiques et leurs Interactions (INS2I), l’ETIS est structurée en quatre équipes de recherche 4 :

— Indexation Multimedia et Intégration de données (MIDI)
— Information, Communications, Imagerie (ICI)
— Architectures, Systèmes, Technologies pour les unités Reconfigurables Embarquées
(ASTRE)
— Neurocybernétique

C’est avec cette dernière équipe que nous avons travaillé dans le cadre de notre projet. Située dans les locaux de l’université de Cergy-Pontoise, c’est grâce à leur travaux sur la synchronie, ainsi que grâce aux différents projets réalisés les années précédentes que nous avons pu mener notre projet à bien.

Présentation association autisme ensemble 95

Autisme Ensemble 95 est une association basée à Pontoise qui regroupe des parents et des professionnels. Leur objectif est de créer un pôle actif en matière d'autisme et de Troubles du Spectre Autistique (TSA). Nous avons travaillé en collaboration avec cette association qui nous a donné un terrain favorable et un cadre pour l'expérimentation de nos travaux sur un enfant atteint de TSA.

Présentation des derniers projets

Notre projet s'inscrit à la suite d'un ensemble de projets, réalisé par d'ancien élèves de l'EISTI en collaboration avec l'ETIS, dont le thème principal porte sur le traitement de l'autisme infantile. L'approche utilisée ici se rapporte à la toute récente utilisation de robot comme thérapie de l'autisme. Plus précisément, on s'intéresse ici à l'effet que peut avoir une synchronisation entre un enfant autiste et un robot.

Le principe de cette synchronisation est plus simple à expliquer avec un exemple que l'on a tous déjà expérimenté : Imaginons que 2 personnes marchent l'un à côté de l'autre. Si ces deux personnes marchent à peu près à la même vitesse, le rythme de leur pas va irrémédiablement finir par être le même, ou en d'autre termes, se synchroniser. Cela s'applique pour toutes les personnes qu'importe leur taille, ou leur rythme de marche habituel. On parle alors de synchronisation non-intentionnelle. Si les rythmes de marche sont à la base trop éloignée, cette synchronisation n'a pas lieu, car demandant trop d'effort.

C'est cet effet là, que l'on tente d'exploiter dans cette thérapie, et donc dans ce projet. De plus, le choix du robot NAO n'est pas anodin. Contrairement à un humain, le robot NAO n'émet que peu de stimuli à caractères sociaux. Or, ces stimuli sont interprété différemment par les enfants autistes, ils sont compliqué à traiter pour eux. Aussi la forme humanoïde du robot est essentielle à la synchronie puisque, pour que celle-ci fonctionne, il est important que l'enfant puisse associer le robot, et ses différents membres, à une forme humaine.

Le premier projet était une première approche dont le but était d'étudier l'impact de la robotique sur les interactions sociales des enfants autistes. La problématique posée était alors : La robotique peut-elle avoir un aspect positif sur le spectre autistique?
Tentant de répondre à cette problématique, ce premier projet comprenait la recherche et le contact avec les acteurs (association autisme ensemble, aldebaran robotics), la planification des séances (quels gestes doit réaliser NAO etc.), la familiarisation avec Prométhée et les expérimentations. De plus une petite application a été réalisée permettant de réaliser des gestes simples à NAO.

Le second projet est en continuation. Il reprend les bases du projet précédent en cherchant à améliorer les expériences sur quatres points : pérenniser le programme sur un nouveau modèle de NAO (le modèle utilisé à l'époque était de l'ancienne génération des robots NAO), pouvoir utiliser une wiimote, réduire les distractions de l'enfant (essentiellement réduire le câblage et ajouter des interactions), et intégrer d'autre mouvements possibles.

Spécification de l'application

Comme nous l’avons vu, notre projet de fin d’étude s’inscrit à la suite d’un ensemble de projet.

Dans cette partie, nous détaillons les spécifications de notre projet.

Notre projet complète les projets précédents. Le point important était de reprendre les acquis des projets précédents, de synthétiser et d’enlever les points négatifs. Notre mission n’était donc pas seulement la refonte du système de synchronie déjà établis, mais une évolution/amélioration de celui-ci.
Tout en re-développant les fonctionnalités des précédents projets, et en respectant les mêmes contraintes (exemple : temps réel), nous devions en plus apporter ce qui manquait aux projets précédents : la portabilité, l’indépendance (vis à vis de Prométhée, ce dernier nécessitant une assistance experte) et la facilité d’utilisation.
En somme, nous devions créer une application pratique, intuitive, finie, et prête à l’emploi pour un/une praticien(ne).

Cahier des charges

Contexte

Voir page 6

Objectifs

1. L'application doit ré-implémenter, au minimum, les fonctionnalités développées dans les projets précédents, à savoir :
   - 1.1. Calcul de flux-optique, vertical (respectivement horizontal), à partir d'une caméra.
   - 1.2. Génération d'une oscillation pour un mouvement vertical (respectivement horizontal) grâce à un oscillateur.
   - 1.3. Modification de la fréquence et de l'amplitude des oscillations verticales (respectivement horizontales) :
     - 1.3.1. Influencé par le flux-optique vertical.
     - 1.3.2. Fréquence modifiable par l'utilisateur.
     - 1.3.3. Amplitude modifiable par l'utilisateur.
2. L'application doit satisfaire les contraintes respectées dans les projets précédents, à savoir : l'ensemble des fonctionnalités édictées précédemment doivent pouvoir être assurées en direct au cours d'une expérience et en temps-réel.
3. L'application doit être facile d'utilisation (IHM).
4. L'application doit être portable.

Périmètre

Pour la portabilité, on se limitera à l'ensemble des distributions GNU/Linux courantes, tels que les Debian et les Ubuntu. Le fonctionnement de l'application n'est pas garantit sur les autres distributions tels que Fedora, Red-hat, Arch-Linux, ou Gentoo, car cela n'a pas été testé, mais il est fort probable que cela marche, pour peu qu'on recompile adéquatement.

Description Fonctionnelle

1. **Générer un flux optique à partir d'une caméra.**
   - 1.1. Ré-implémenter l'algorithme du flux optique fourni (utilisé dans Prométhée), le modifier, l'optimiser.
   - 1.2. Pouvoir choisir la caméra (externe notamment).
   - 1.3. Générer un flux optique vertical et un flux optique horizontal.
   - 1.4. Optionnellement pouvoir gérer la sensibilité des flux optiques.

2. **Implémenter un oscillateur afin de générer une oscillation**
   - 2.1. Ré-implémenter un(des) oscillateur(s) comme celui dans Promethée.
   - 2.2. Générer une oscillation.
   - 2.3. Pouvoir modifier les paramètres de(s) l'oscillateur(s), à savoir :
     - 2.3.1. Le facteur de couplage.
     - 2.3.2. La variable alpha.
     - 2.3.3. La variable beta (influent la fréquence).

3. **Permettre une synchronie.**
   - 3.1. Connecter la sortie du flux optique (de manière adéquate) à l'entrée de l'oscillateur correspondant (i.e vertical ou horizontal) afin d'influer celui-ci.
   - 3.2. Pouvoir connecter la sortie de l'oscillateur vertical (respectivement horizontal) à NAO.
   - 3.3. Pouvoir choisir le mouvement à effectuer.

4. **Temps réel.**

   Parmi les fonctionnalités doivent pouvoir être générer et gérer temps réels:
   - 4.1.1. Concernant les flux optiques les fonctionnalités : 1.3 et 1.4.
   - 4.1.2. Concernant les oscillations les fonctionnalités : 2.2 et 2.3 (en entier).
   - 4.1.3. Concernant la synchronie connexion: 3 en entier.
   - 4.1.4. Concernant les mouvements du robot: 6.1.
   - 4.1.5. Concernant les données à enregistrer: 7.1.1, 7.1.2 et optionnellement 7.1.3.

5. **Connexion avec le robot NAO.**
   - 5.1. Permettre la connexion avec le robot NAO afin de permettre les fonctionnalités 3.x.

5.2. Doit être assez facile d'utilisation.

6. Mouvement du robot en fonction de la synchronie.

6.1. Le robot doit effectuer un mouvement en fonction de la sortie de l'oscillateur.

7. Enregistrement de données.

7.1. L'application doit permettre d'enregistrer des données sur les expériences :

7.1.1. Données sur les flux optiques.

7.1.2. Données sur les oscillateurs.

7.1.3. Vidéos (optionnel).

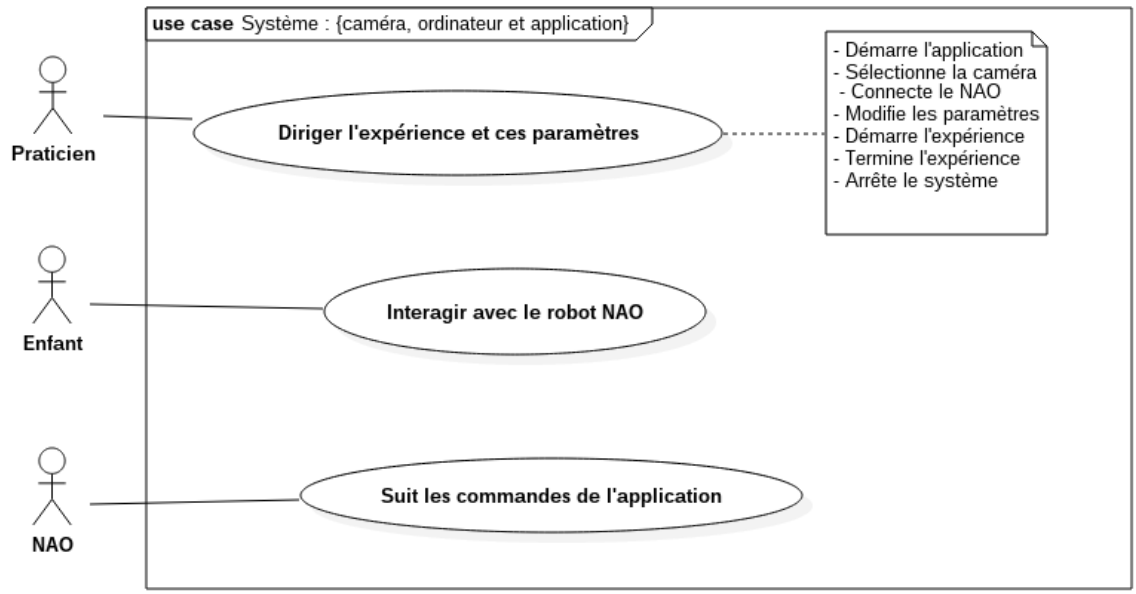
8. IHM.

8.1. L'application doit avoir une Interface Graphique simple d'utilisation et intuitive.

Budget et délais

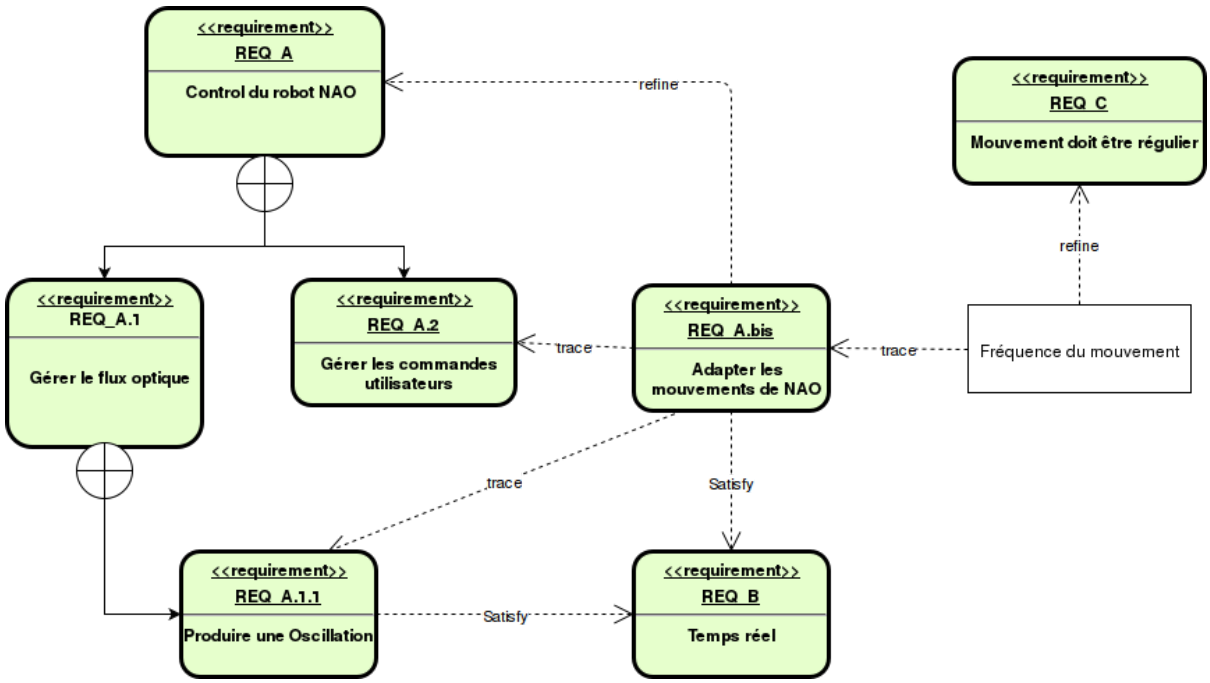
1. Délai de 3 mois (de début Janvier au 4 Avril).
2. Accès à un robot NAO en salle INEM (EISTI).
3. Accès à un robot NAO à l'ETIS.
4. Locaux de la salle INEM.
5. Locaux de l'ETIS.

Diagramme cas d'utilisation



Ce diagramme représente l'ensemble des acteurs en communication avec le système. Le robot est contrôlé par l'application et suit les commandes qui lui est transmis. L'enfant interagit avec le robot en l'imitant, il est filmé par la caméra externe relié à l'ordinateur.
Le praticien dispose du contrôle de l'application : c'est à lui que revient la configuration du robot NAO et des paramètres de l'oscillateur, ainsi que le choix de la caméra externe, il lui revient aussi de choisir le moment de connecter le robot NAO ainsi que d'enregistrer les données. A savoir que la modification des paramètres de l'oscillateur et de l'enregistrement peut intervenir à tout moment.

Diagramme d'exigence

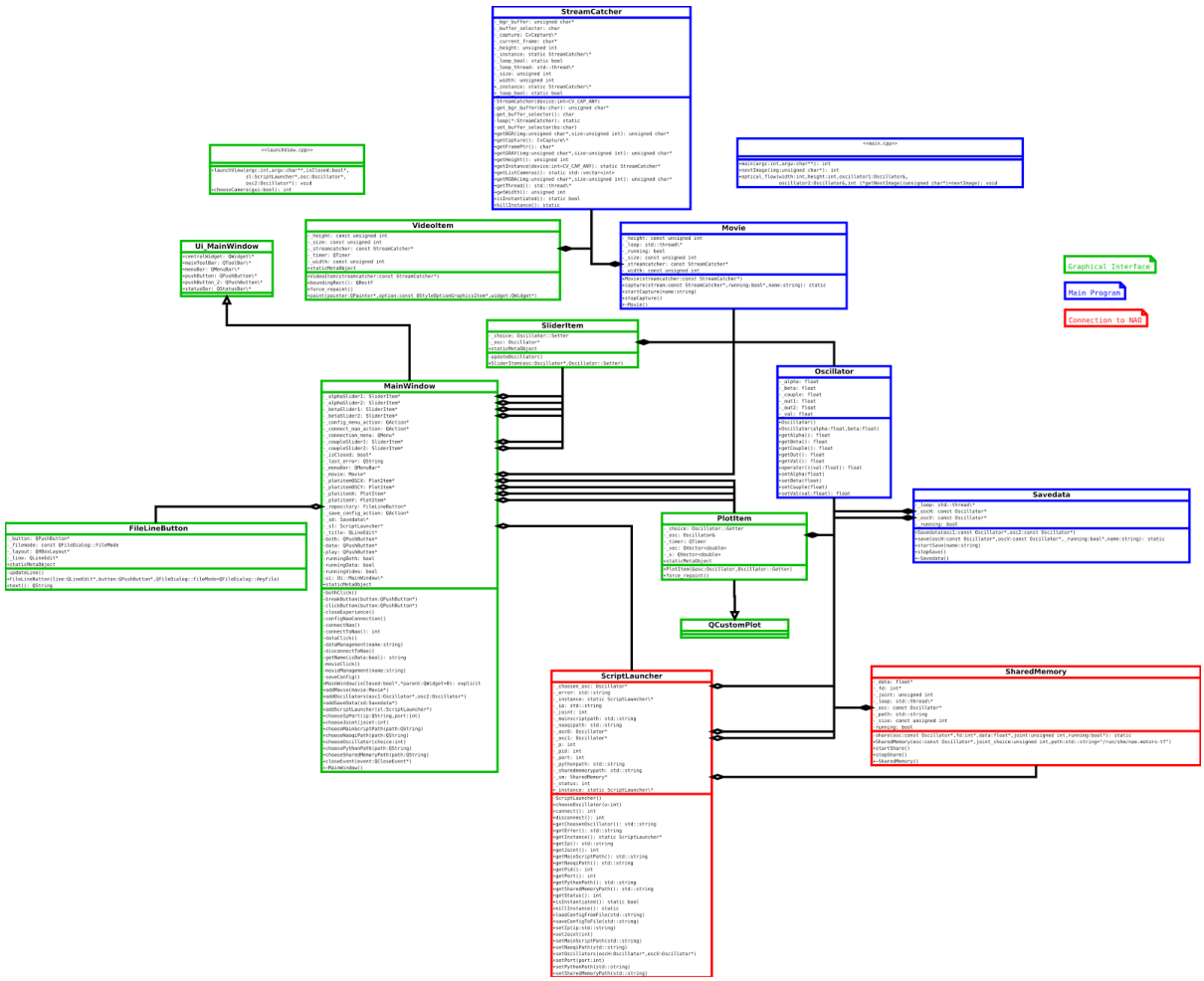


Ce diagramme permet d'énumérer l'ensemble des contraintes qui découlent des spécifications. Il permet d'illustrer les connexions entre les différentes parties de l'application. Voici donc ces exigences :

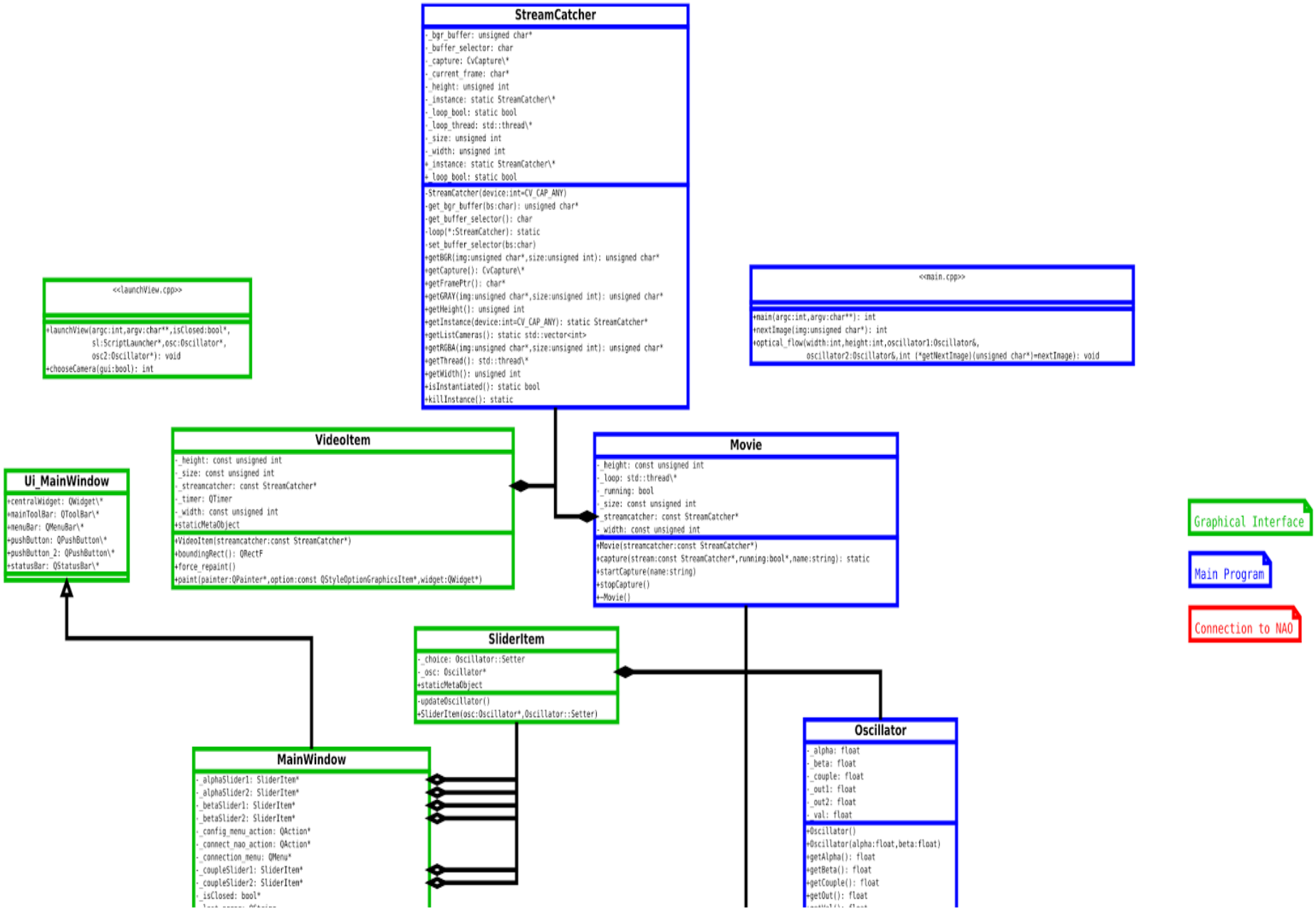
- Contrôle du robot NAO : le robot doit être entièrement contrôlé par le robot, sachant que le robot doit provoquer la réaction de l'enfant par le mouvement répété d'un geste simple, l'expérience ne doit pas être interrompu par une action non souhaitée.
- Gérer le flux optique : pour cela l'application doit gérer la capture d'image par image et créer un flux optique en direct (durant l'expérience).
- Produire une oscillation : le traitement du flux optique permet l'obtention d'une synchronie qui permet d'en déduire l'importance du mouvement (sa fréquence et son amplitude).
- Gérer les commandes utilisateurs : l'application a pour objectif de traiter les expériences avec les enfants autistes, et d'analyser la réaction de ces derniers. Permettre les modifications et influencer le mouvement du robot NAO est alors indispensable.
- Adapter les mouvements de NAO : il reste alors à appliquer ces modifications au robot NAO en direct et observer la réaction de l'enfant.
- Temps réel : pour imiter le mouvement de l'enfant, le robot NAO doit faire preuve de réactivité, il est alors indispensable que l'application soit temps réel.

- Le mouvement doit être régulier : pour que l'enfant s'identifie au robot, et ne pas perturber l'expérience, le mouvement de ce dernier doit être fluide et régulier.

Diagramme de classe



Ce diagramme de classe montre l'organisation de notre application d'un point de vue orienté objet. En rouge, sont représentés les scripts pour le mouvement du robot NAO, en vert les outils nécessaires pour l'interface graphique et en bleu le programme principal de l'application. En haut à droite, vous localisez le main.cpp qui lance l'application alors qu'à gauche le launchview.cpp lance la vue.


StreamCatcher
- _bgr_buffer: unsigned char*
- _buffer_selector: char
- _capture: CvCapture*
- _current_frame: char*
- _height: unsigned int
- _instance: static StreamCatcher*
- _loop_bool: static bool
- _loop_thread: std::thread*
- _size: unsigned int
- _width: unsigned int
+ _instance: static StreamCatcher*
+ _loop_bool: static bool
-StreamCatcher(device:int=CV_CAP_ANY)
-get_bgr_buffer(bs:char): unsigned char*
-get_buffer_selector(): char
-loop(*:StreamCatcher): static
-set_buffer_selector(bs:char)
+getBGR(img:unsigned char*,size:unsigned int): unsigned char*
+getCapture(): CvCapture*
+getFramePtr(): char*
+getGRAY(img:unsigned char*,size:unsigned int): unsigned char*
+getHeight(): unsigned int
+getInstance(device:int=CV_CAP_ANY): static StreamCatcher*
+getListCameras(): static std::vector<int>
+getRGBA(img:unsigned char*,size:unsigned int): unsigned char*
+getThread(): std::thread*
+getWidth(): unsigned int
+isInstantiated(): static bool
+killInstance(): static
<<launchView.cpp>>
+launchView(argc:int,argv:char**,isClosed:bool*,
sl:ScriptLauncher*,osc:Oscillator*,
osc2:Oscillator*): void
+chooseCamera(gui:bool): int
<<main.cpp>>
+main(argc:int,argv:char**): int
+nextImage(img:unsigned char*): int
+optical_flow(width:int,height:int,oscillator1:Oscillator&,
oscillator2:Oscillator&,int (*getNextImage)(unsigned char*)=nextImage): void
VideoItem
- _height: const unsigned int
- _size: const unsigned int
- _streamcatcher: const StreamCatcher*
- _timer: QTimer
- _width: const unsigned int
+staticMetaObject
+VideoItem(streamcatcher:const StreamCatcher*)
+boundingRect(): QRectF
+force_repaint()
+paint(painter:QPainter*,option:const QStyleOptionGraphicsItem*,widget:QWidget*)
Movie
- _height: const unsigned int
- _loop: std::thread*
- _running: bool
- _size: const unsigned int
- _streamcatcher: const StreamCatcher*
- _width: const unsigned int
+Movie(streamcatcher:const StreamCatcher*)
+capture(stream:const StreamCatcher*,running:bool*,name:string): static
+startCapture(name:string)
+stopCapture()
+~Movie()
Ui_MainWindow
+centralWidget: QWidget*
+mainToolBar: QToolBar*
+menuBar: QMenuBar*
+pushButton: QPushButton*
+pushButton_2: QPushButton*
+statusBar: QStatusBar*
SliderItem
- _choice: Oscillator::Setter
- _osc: Oscillator*
+staticMetaObject
-updateOscillator()
+SliderItem(osc:Oscillator*,Oscillator::Setter)
MainWindow
- _alphaSlider1: SliderItem*
- _alphaSlider2: SliderItem*
- _betaSlider1: SliderItem*
- _betaSlider2: SliderItem*
- _config_menu_action: QAction*
- _connect_nao_action: QAction*
- _connection_menu: QMenu*
- _coupleSlider1: SliderItem*
- _coupleSlider2: SliderItem*
- _isClosed: bool*
Oscillator
- _alpha: float
- _beta: float
- _couple: float
- _out1: float
- _out2: float
- _val: float
+Oscillator()
+Oscillator(alpha:float,beta:float)
+getAlpha(): float
+getBeta(): float
+getCouple(): float
+getOut(): float
Graphical Interface
Main Program
Connection to NAO

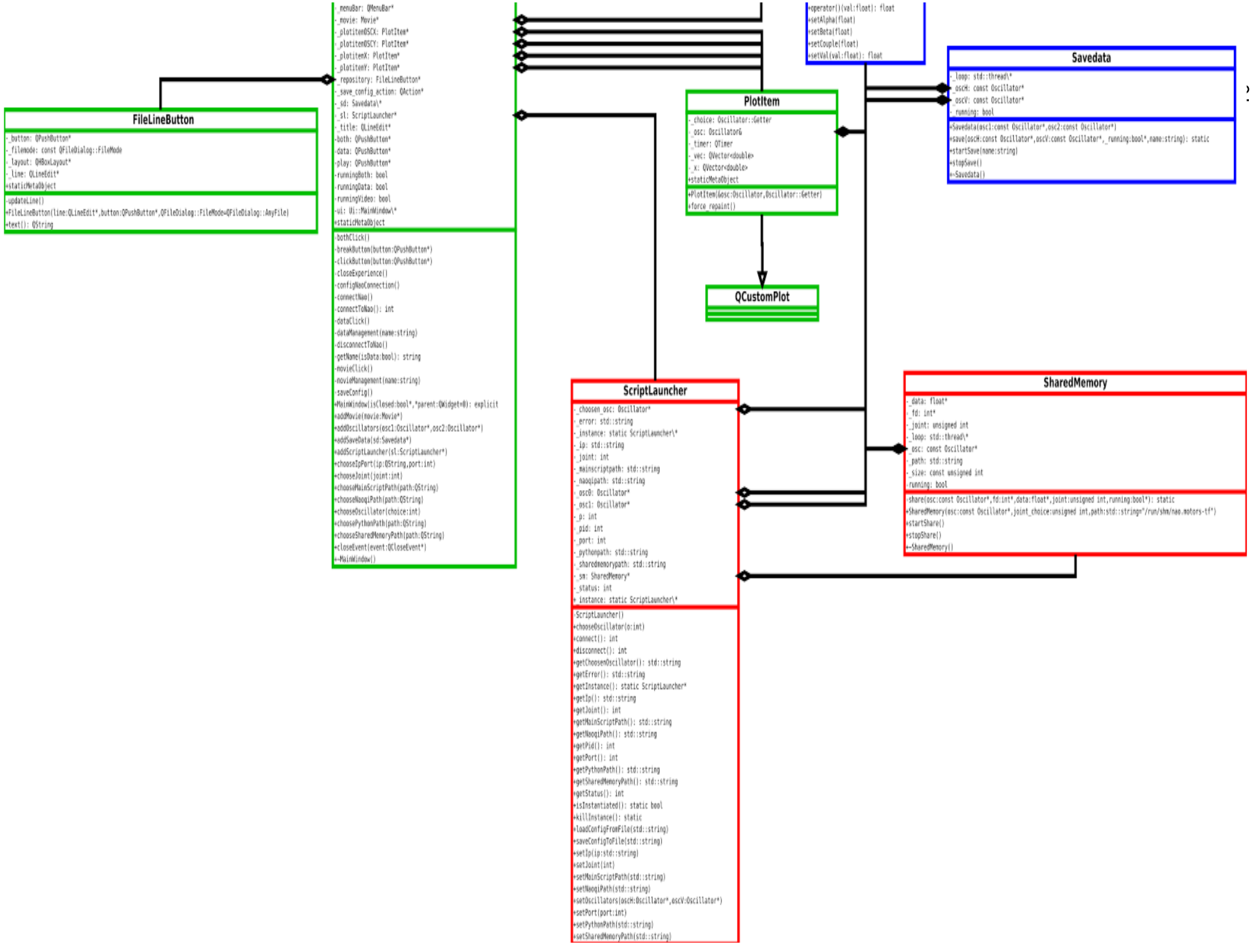
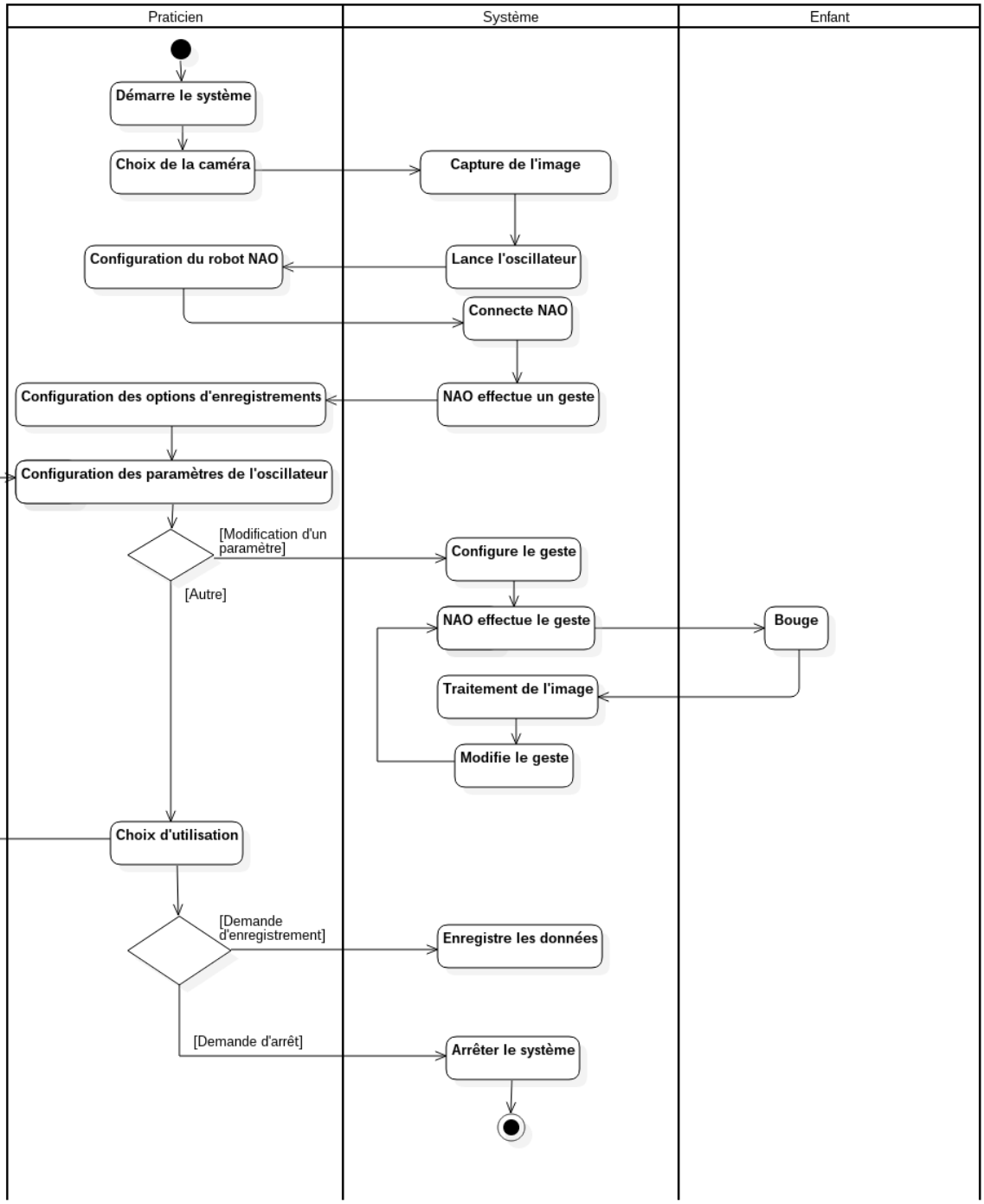

FileLineButton
- _button: QPushButton*
- _filemode: const QFileDialog::FileMode
- _layout: QHBoxLayout*
- _line: QLineEdit*
+staticMetaObject
-updateLine()
+FileLineButton(line:QLineEdit*,button:QPushButton*,QFileDialog::FileMode=QFileDialog::AnyFile)
+text(): QString
- _menuBar: QMenuBar*
- _movie: Movie*
- _plotitemOSCX: PlotItem*
- _plotitemOSCY: PlotItem*
- _plotitemX: PlotItem*
- _plotitemY: PlotItem*
- _repository: FileLineButton*
- _save_config_action: QAction*
- _sd: Savedata*
- _sl: ScriptLauncher*
- _title: QLineEdit*
-both: QPushButton*
-data: QPushButton*
-play: QPushButton*
-runningBoth: bool
-runningData: bool
-runningVideo: bool
-ui: Ui::MainWindow*
+staticMetaObject
-bothClick()
-breakButton(button:QPushButton*)
-clickButton(button:QPushButton*)
-closeExperience()
-configNaoConnection()
-connectNao()
-connectToNao(): int
-dataClick()
-dataManagement(name:string)
-disconnectToNao()
-getName(isData:bool): string
-movieClick()
-movieManagement(name:string)
-saveConfig()
+MainWindow(isClosed:bool*,*parent:QWidget=0): explicit
+addMovie(movie:Movie*)
+addOscillators(osc1:Oscillator*,osc2:Oscillator*)
+addSaveData(sd:Savedata*)
+addScriptLauncher(sl:ScriptLauncher*)
+chooseIpPort(ip:QString,port:int)
+chooseJoint(joint:int)
+chooseMainScriptPath(path:QString)
+chooseNaoqiPath(path:QString)
+chooseOscillator(choice:int)
+choosePythonPath(path:QString)
+chooseSharedMemoryPath(path:QString)
+closeEvent(event:QCloseEvent*)
+~MainWindow()
+operator()(val:float): float
+setAlpha(float)
+setBeta(float)
+setCouple(float)
+setVal(val:float): float
PlotItem
- _choice: Oscillator::Getter
- _osc: Oscillator&
- _timer: QTimer
- _vec: QVector<double>
- _x: QVector<double>
+staticMetaObject
+PlotItem(&osc:Oscillator,Oscillator::Getter)
+force_repaint()
QCustomPlot
Savedata
- _loop: std::thread*
- _oscH: const Oscillator*
- _oscV: const Oscillator*
- _running: bool
+Savedata(osc1:const Oscillator*,osc2:const Oscillator*)
+save(oscH:const Oscillator*,oscV:const Oscillator*,_running:bool*,name:string): static
+startSave(name:string)
+stopSave()
+~Savedata()
ScriptLauncher
- _choosen_osc: Oscillator*
- _error: std::string
- _instance: static ScriptLauncher*
- _ip: std::string
- _joint: int
- _mainscriptpath: std::string
- _naoqipath: std::string
- _osc0: Oscillator*
- _osc1: Oscillator*
- _p: int
- _pid: int
- _port: int
- _pythonpath: std::string
- _sharedmemorypath: std::string
- _sm: SharedMemory*
- _status: int
+ instance: static ScriptLauncher*
-ScriptLauncher()
+chooseOscillator(o:int)
+connect(): int
+disconnect(): int
+getChoosenOscillator(): std::string
+getError(): std::string
+getInstance(): static ScriptLauncher*
+getIp(): std::string
+getJoint(): int
+getMainScriptPath(): std::string
+getNaoqiPath(): std::string
+getPid(): int
+getPort(): int
+getPythonPath(): std::string
+getSharedMemoryPath(): std::string
+getStatus(): int
+isInstantiated(): static bool
+killInstance(): static
+loadConfigFromFile(std::string)
+saveConfigToFile(std::string)
+setIp(ip:std::string)
+setJoint(int)
+setMainScriptPath(std::string)
+setNaoqiPath(std::string)
+setOscillators(oscH:Oscillator*,oscV:Oscillator*)
+setPort(port:int)
+setPythonPath(std::string)
+setSharedMemoryPath(std::string)
SharedMemory
- _data: float*
- _fd: int*
- _joint: unsigned int
- _loop: std::thread*
- _osc: const Oscillator*
- _path: std::string
- _size: const unsigned int
- _running: bool
-share(osc:const Oscillator*,fd:int*,data:float*,joint:unsigned int,running:bool*): static
+SharedMemory(osc:const Oscillator*,joint_choice:unsigned int,path:std::string="/run/shm/nao.motors-tf")
+startShare()
+stopShare()
+~SharedMemory()

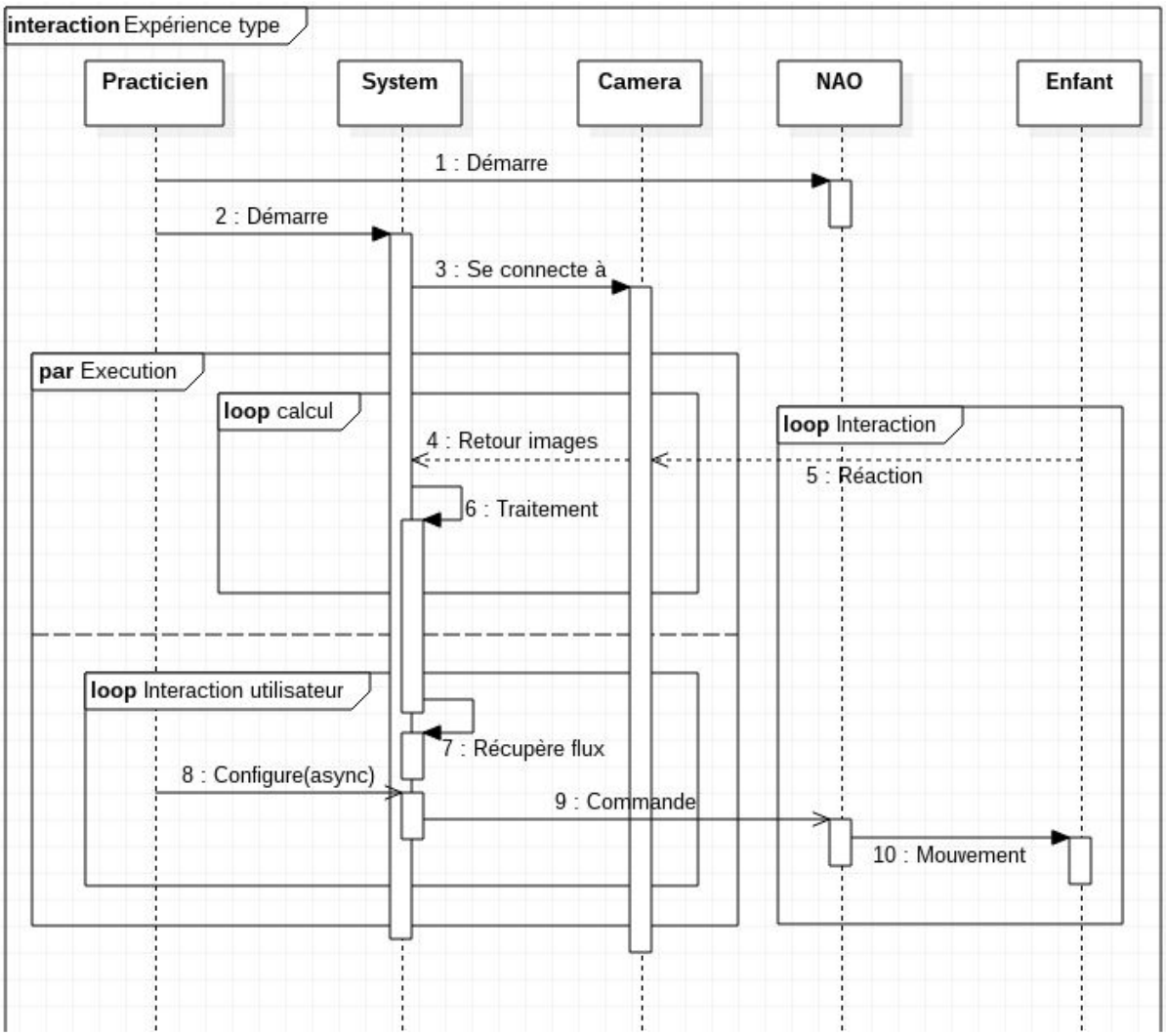
Diagramme d’activité



Ce diagramme montre le scénario “type” d’une expérience. Le praticien doit lancer l’application, on lui demande alors de choisir une caméra. Cette caméra va alors capturer les images les unes après les autres pour implémenter un oscillateur. Afin de permettre la transmission de donnée au robot NAO, le praticien doit configurer les informations qui permettent de contrôler le robot, ce qui lui permet de faire et répéter

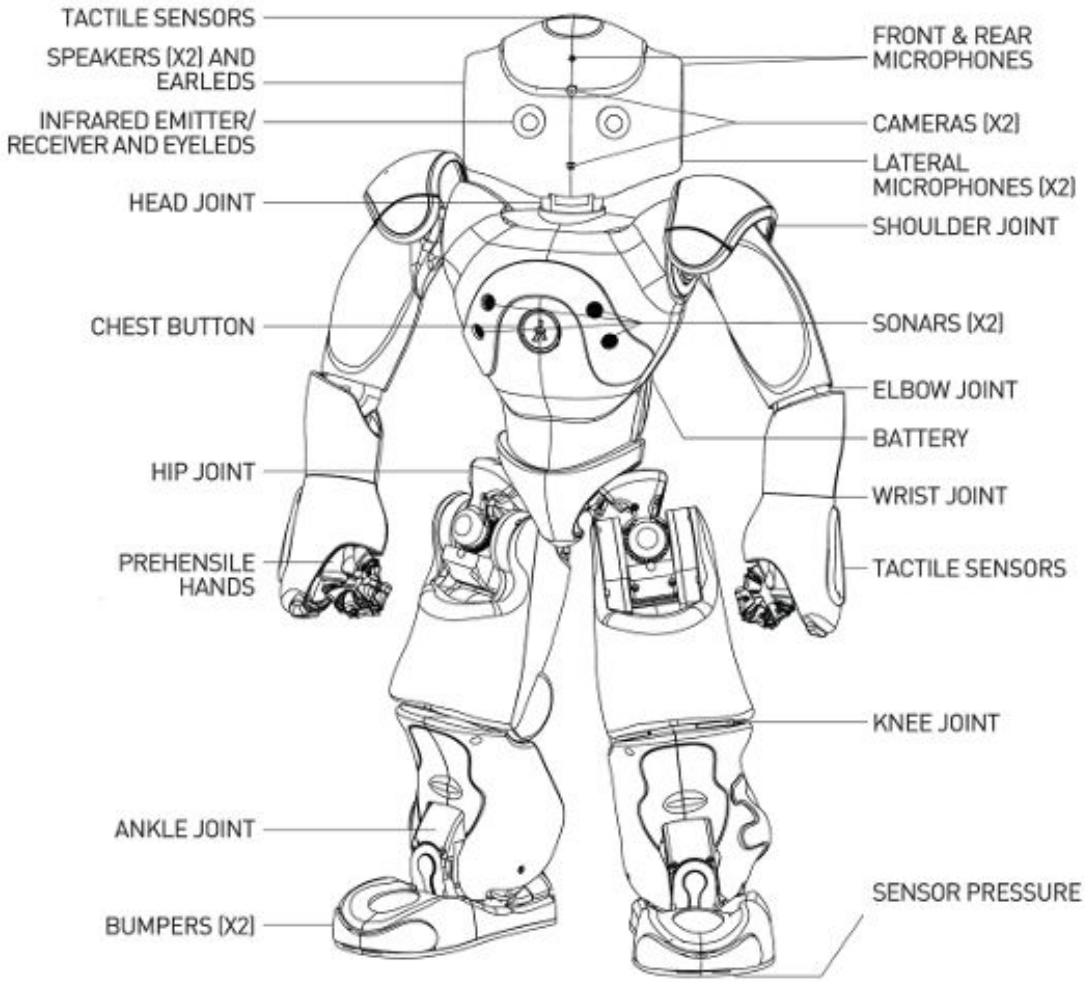
un geste. Le praticien a alors le libre choix de faire varier les paramètres de l'oscillateur ou d'enregistrer des données (vidéo ou résultat oscillateur). Sachant qu'à partir de ce moment là, une réaction de l'enfant est attendu (un mouvement), ce mouvement est alors traité et le mouvement du robot NAO (amplitude/fréquence) est alors ajusté. Le praticien peut choisir l'emplacement de son enregistrement. Le système fonctionne jusqu'à la demande de l'arrêt du système.

Diagramme de séquence



Implémentation et outil

NAO



NAO est un petit robot humanoïde de petite taille (58 cm pour 5Kg). Muni de nombreux capteurs, d'articulations et d'une caméra, ce robot est largement utilisé à des fins académiques et universitaires. Son évolution lui permet d'effectuer de plus en plus de tâches, ce qui le rend de plus en plus intéressant. Son design et son comportement sont pensés pour être le plus proche de celui de l'être humain, pour ainsi être accepté par l'homme et le côtoyer. Sa principale caractéristique est qu'il est entièrement programmable : tout mouvement, compréhension orale (dans une langue choisie) ainsi que son expression peut-être contrôlée.

NAO est aujourd'hui le leader dans le paysage de la robotique mondial grâce à son interaction homme-machine, sa motricité, sa vision et son langage. NAO trouve de plus en plus d'application, et tente de reproduire le comportement humain dans des situations de la vie réelle et constitue un sujet des plus concrets pour l'implémentation d'IA.

L’application

L’application a entièrement été implémenté en C/C++, seul un script Python est utilisé pour la transmission du mouvement au robot. L’application doit permettre à NAO d’imiter un enfant autiste dans l’amplitude et la fréquence d’un mouvement facilement reproduisible. Pour cela l’application doit avoir accès à une caméra, choisie par l’utilisateur, pour capturer une image, générer un flux optique qui implémente un oscillateur, une synchronie est ainsi obtenu dès l’ouverture de l’application. Par la suite, l’utilisateur peut configurer des paramètres pour se connecter avec le robot NAO choisi. La connexion obtenue permet de transmettre la synchronie au robot, le robot NAO peut alors exécuter un mouvement dans l’amplitude et la fréquence calculé. A tout moment, l’utilisateur peut enregistrer la vidéo et/ou les données de l’expérience.

Aucune des features cités au dessus n’arrête, ni ne ralentit l’application, le temps réel étant l’une des principales contraintes au projet. Pour cela l’application est largement multi threadée (6 threads au total) et le script python est lancé dans un fork de processus.

Capture d'image

La capture de l’image représente l’étape initiale de l’application (ainsi que son fonctionnement). La stabilité du robot NAO durant une expérience ainsi que la qualité de sa caméra nous a poussé à choisir une caméra externe pour la capture de l’image. En effet, le robot obtiendrai une image instable dû au mouvement exécuté par une partie de son corps ainsi qu’une faible qualité mettaient en difficulté les résultats de l’expérience. Aussi, la connexion de l’application avec la caméra du robot NAO dépend d’une connexion internet plus que incertaine. C’est pourquoi nous avons opté pour la capture de l’image à partir d’une caméra externe (type webcam) dont le support sera fixé pour les besoins de l’expérience et ainsi des résultats correct.

La capture de l’image se déroule image par image, et permet l’obtention d’un vecteur composé de pixel (format RGB). Pour cela nous utilisons la librairie OpenCV qui permet la capture et l’enregistrement d’image et de vidéo. Cette librairie se distingue par ses fonctionnalités et son efficacité, son implémentation en C/C++ lui permet une très bonne réactivitée et facilite le travail temps réel de l’application.

Flux optique

Suite à la capture de l'image, notre application calcule les flux optiques verticaux et horizontaux.

Le flux (ou flot) optique est défini comme le champ de vecteur vitesse décrivant le mouvement apparent des motifs d'intensité d'une image.
Généralement, le calcul du flux optique se base sur l'hypothèse que la couleur (l'intensité perçue) d'un point se mouvant, est indépendante du temps.
Concrètement, cela signifie que l'on considère que les variations d'intensité ayant lieu à un point de coordonnées fixes dans une image sont obligatoirement dû à un mouvement (de ce qui est "filmé") et non dû à une variation de la luminosité ambiante, ni à une variation de la saturation, ni à aucun autre facteur.

On pointe alors sur une des faiblesses principale de cette méthode de calcul du flux optique : elle est inopérante dans un milieu où l'intensité lumineuse change beaucoup.
Cependant, cette faiblesse peut être ignorée dans notre cas, puisque les expériences se dérouleront le plus souvent dans un environnement ayant une intensité lumineuse fixe (en intérieur, en salle).

L'intensité lumineuse "I" en un point étant indépendante du temps on a donc l'équation suivante :

$$\frac{\partial I_x}{\partial x}u + \frac{\partial I_y}{\partial y}v + \frac{\partial I_t}{\partial t} = 0 \ (1)$$

$$\text{Avec } u = \frac{\partial x}{\partial t} \text{ et } v = \frac{\partial y}{\partial t}$$

Dans cette équation indiquant le mouvement dans l'image, on a deux inconnues U et V. On ne peut donc pas résoudre directement cette équation. Cette sous-détermination est désignée par Horn et Schunck [1981] sous le nom de "problème d'ouverture" (aperture problem).
Le problème d'ouverture est simple à expliquer : si l'on regarde une image un mouvement à travers un trou minuscule, il est alors difficile de savoir dans quel direction l'image se déplace. Or, c'est exactement ce que l'on fait là en ne considérant qu'un seul pixel.

Les différentes méthodes de résolutions de l'équation (1) la résolve en ajoutant d'autres contraintes, principalement en prenant en compte les pixels voisins et en considérant que dans ce voisinage la variation du flux optique est moindre (nulle).
La méthode de Horn et Schunck considère une régularité globale du flux optique (et donc pas seulement le voisinage).
Le problème s'écrit alors comme une minimisation de la fonctionnelle :

$$\iint \left[(I_x u + I_y v + I_t)^2 + \alpha^2(\|\nabla u\|^2 + \|\nabla v\|^2)\right] \mathrm{d}x\mathrm{d}y$$

ou α est un paramètre de régularisation. Un α plus grand donne un flux plus fluide.

La minimisation de cette fonctionnelle est donnée dans la méthode Horn et Schunck. Elle nous donne les équations solutions itératives :

$$u^{k+1} = \overline{u}^k - \frac{I_x(I_x\overline{u}^k + I_y\overline{v}^k + I_t)}{\alpha^2 + I_x^2 + I_y^2} \quad (2.1)$$

$$v^{k+1} = \overline{v}^k - \frac{I_y(I_x\overline{u}^k + I_y\overline{v}^k + I_t)}{\alpha^2 + I_x^2 + I_y^2} \quad (2.2)$$

Algorithmiquement, une dérivée partielle de l'intensité dans le temps, se traduit par une soustraction entre la valeur d'une variable représentant l'intensité à une itération t+1 et la même variable à l'itération t.
De même, une dérivée partielle de l'intensité dans l'espace se traduit par une soustraction entre la valeur de l'intensité en un point, à la valeur de l'intensité du point immédiatement à sa gauche ou droite (pour U, le flux optique horizontal) ou en haut ou en dessous (pour V, le flux optique vertical).

L'algorithme tourne donc en traitant deux images successives à chaque itération.On obtient alors l'algorithme suivant en pseudo-code (pour le voir en C, voir les sources) :

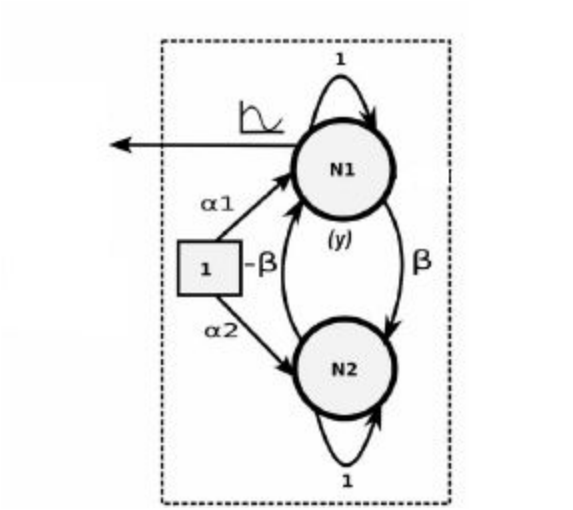
//la variable sensibility règle la sensibilité générale des oscillateurs vis-à-vis des flux //optiques. Cela permet, entre autre, d'adapter les réglages en fonction de la distance //du sujet par rapport à la caméra

```
img0 = getFirstImage()
img1 = getNextImage()
Ix[imagesSize()] //vecteur variation intensité verticale
Iy[imagesSize()] //vecteur variation intensité horizontale
It[imagesSize()] //vecteur variation intensité dans le temps

U[imagesSize()] //vecteur flux horizontal
V[imagesSize()] //vecteur flux vertical

while(1):
        for each point  p0, p1 in img0, img1: //calcul de : Ix, Iy et It dans l'équation (1)
                //(avec i un compteur)

                Iy[i] = (p0.bas_droite + p0.bas - p0 - p0.droite) +
                        (p1.bas_droite + p1.bas - p1 - p1.droite)
                Ix[i] = (p0.bas_droite + p0.droite - p0 - p0.bas) +
                        (p1.bas_droite + p1.droite - p1 - p1.bas)
                It[i] = (p1 + p1.droite + p1.bas + p1.bas_droite) -
                        (p0 + p0.droite + p1.bas + p1.bas_droite)
        //end for each

        for n iterations: //(avec i un compteur)
                for each point  p0, p1 in img0, img1:
                        mu = sommeDesFluxAdjacentACePoint(U) //somme des u
                associée a des poids selon la distance
                        mv = sommeDesFluxAdjacentACePoint(V)
                        vx = Ix[i]
                        vy = Iy[i]
                         vt = Iy[i]

                        //calcul des équations solutions (2.1) et (2.2)
                        U[i] = mu - vx * (vx * mu + vy * mv + vt) / ( α + vx² + vy²)
                        V[i] = mv - vy * (vx * mu + vy * mv + vt) / ( α + vx² + vy²)
                //end for each point (avec α = 1)
        //end for n iterations
img0 = img1
img1 = getNextImage()

//sensibility est une variable permettant de régler la sensibilité générale aux flux
sU = somme(U) * sensibility ; sV = somme(V) * sensibility  //sensibility ≃ 0.000001f
oscillateur_horizontal(sU); oscillateur_vertical(sV)
//end while
```

Oscillateur

La deuxième partie importante du système est l'oscillateur. Dans l'application, il y a deux oscillateur fonctionnant en même temps (un pour le flux horizontal, et un pour le flux vertical), mais nous parlerons ici que d'un seul. Considérez donc qu'il pourra s'agir autant de l'oscillateur vertical qu'horizontal, le principe étant le même.

L'oscillateur est basé sur un modèle simple à deux bascules NO (portes logiques). Plus précisément on le représente comme un système à deux neurones, s'inhibant mutuellement.



Concrètement, l'oscillateur peut être représenté par un système à deux suites :

$$N1_{(n+1)} = N1_{(n)} - \beta N2_{(n)} + \alpha$$

$$N2_{(n+1)} = N2_{(n)} + \beta N1_{(n)} + \alpha$$

N1 est la sortie de l'oscillateur.
Avec, précisons, N2 limité entre 0 et 1.
Cela nous donne directement le code suivant :

```
float Oscillator::setVal(float val)
{
   //Avec _out1, _out2, _val, _alpha, _couple des float
   _val = val;
   float tmp_out1 = _out1 - _beta * _out2 + _alpha + _couple * _val;
   _out2  = _out2 >  1.f ? 1.f : _out2;
   _out2  = _out2 <  0.f ? 0.f : _out2;
   _out2 = _out2 + _beta * _out1;
   _out1 = tmp_out1;

   return _out1;
}
```

La variable _couple est un facteur de couplage, permettant de régler un certain degré (minime) de sensibilité par oscillateur (contrairement à la variable sensibility dans la partie précédente, qui règle la sensibilité générale).

Les variables _alpha et _beta sont réglables en temps réel dans notre application.

Transmission

Pour la connexion avec NAO, nous avons repris le script Python écrit lors du dernier projet. Nous l'avons quelque peu modifié, afin de mieux correspondre aux attentes.

Notre application prends entièrement le contrôle du script en s'occupant du lancement et de la terminaison de celui-ci. Le chemin du script est spécifiable par configuration dans l'application, pour permettre plus de flexibilité si on veut tester de nouveau comportement de NAO. De plus les articulations de NAO, sont simplement représentée par un numéro (de 0 à 9) à l'intérieur de notre application, afin de permettre encore plus de flexibilité au niveau du choix de l'ordre des articulations dans le script.

Le script utilise la librairie naoqi (dont le lien est ci-dessous) :
http://doc.aldebaran.com/2-1/dev/python/install_guide.html
Cette librairie est nécessaire au fonctionnement du script. L'emplacement de cette librairie peut et doit être configuré au travers de l'application.

Enregistrements

Vis-à-vis des enregistrements nous devions pouvoir enregistrer les données des oscillateurs et des flux optiques et optionnellement le flux vidéo.
Pour les données des oscillateurs et des flux optiques nous enregistrons cela dans un fichier CSV, en 4 colonnes respectivement : flux horizontal, oscillateur horizontal, flux vertical, oscillateur vertical.
Enfin il est possible d'enregistrer en même temps (ou séparément) le flux vidéo, sous format AVI.
Tout cela n'interrompt pas, ni ne ralentit, le fonctionnement de l'application.

Librairie

Nous avons utilisé quelque libraire/framework pour la réalisation de notre projet.

Nom	Description	Utilisation
OpenCV http://opencv.org/	Librairie pour gestion et traitement d'image.	Utilisé uniquement dans notre application pour la gestion (capture, enregistrement) d'image, et non son traitement/calcul : • Choix de la caméra • Capture des images • Enregistrements vidéos
Qt(5) http://doc.qt.io/	Framework offrant des composants d'interface graphiques à l'origine, mais fournissant une librairie alternative à lib std	Utilisé uniquement pour l'interface utilisateur : • IHM
QCustomPlot http://www.qcustomplot.com/	Petite librairie dérivée de Qt, offrant la possibilité de tracer et visualiser des graphiques.	Utilisé uniquement pour le traçage de graphiques • Traçage des courbes des flux optiques • Traçages des courbes des oscillateurs
naoqi http://doc.aldebaran.com/2-1/dev/python/install_guide.html	Librairie python (et C++ mais ici version python) Pour le développement de script de contrôle du robot NAO	Utilisé (et nécessaire) ici par le script de contrôle de NAO (écrit durant le projet précédent)
librairie C et librairie C++ https://www.gnu.org/software/libc/documentation.html	Bibliothèques standards	Utilisés pour tout le reste.

http://www.cplusplus.com/		

Conclusion

Bien que la durée de ce PFE fût courte (3 mois), nous avons pu remplir les objectifs que nous nous étions fixés.

Nous avons beaucoup appris durant ce PFE. Au niveau technique certes, mais aussi sur le contexte, sur la robotique, et sur l'autisme infantile. Au début nous avions quelques inquiétudes, au niveau du délai principalement et de l'apport que cela aurait pour nous vis-à-vis de notre cursus en Informatique Embarquée. Il s'est avéré que le délai accordé était juste suffisant et qu'en fait les contraintes (particulièrement de temps réel) rencontrée dans ce projet sont pour beaucoup les mêmes qu'on puisse rencontré dans l'embarqué.

Enfin, et surtout, ce fut un projet concret, avec de vrais objectifs pour une vraie problématique. Cela change beaucoup de chose, en cela que nous somme fier d'avoir pu participer, ne serait ce qu'un petit peu, à la recherche pour le traitement de l'autisme infantile. Que notre projet va servir, et servir à une tâche dont la cause est honorable, est une idée qui nous a accompagné tout au long du projet et ainsi à a faire du mieux que l'on pouvait.

Nous espérons avoir apporté une maigre contribution à cette recherche.

Bibliographie

- Riadh Fezzani. Approche parallele pour l'estimation du ot optique par methode variationnelle. Traitement des images. Universite Pierre et Marie Curie - Paris VI, 2011. Francais. <tel-00713970> https://tel.archives-ouvertes.fr/tel-00713970/document
- https://en.wikipedia.org/wiki/Horn%E2%80%93Schunck_method
- http://www.cmap.polytechnique.fr/~bernard/FlotOptique/page1.html
- http://opencv.org/
- http://doc.qt.io/
- http://www.qcustomplot.com/
- http://doc.aldebaran.com/2-1/dev/python/install_guide.html
- https://www.gnu.org/software/libc/documentation.html
- http://www.cplusplus.com/
- http://doc.aldebaran.com/2-1/family/nao_h25/index_h25.html#nao-h25