Institut Supérieur
d'Informatique, de
Modélisation et de
leurs Applications

Campus des Cézeaux
1 Rue de la Chebarde,
63178 Aubière CEDEX

Easymov Robotics

59, boulevard Léon
Jouhaux
CS 90706 - 63050
Clermont-Ferrand Cedex 2

Rapport d'élève ingénieur
Stage de 2ème année (6 mois)

Université Clermont Auvergne
Institut Supérieur d'Informatique, de Modélisation et de leurs Applications

Développement d'outils et modules à la plateforme web OROS

Présenté par : ***Boris LEGROS***

Tuteur de stage : Noël MARTIGNONI
Référant ISIMA : Romuald AUFRERE

Soutenance le 25 Août
Du 03 Avril au 01 Septembre 2017

Remerciements

Je remercie l'entreprise Easymov Robotics d'avoir accepté de m'accueillir pour ce stage, et tout particulièrement Noel Martignoni en tant que tuteur pour avoir été à l'écoute. Je remercie également Monsieur Romuald AUFRERE en sa qualité de référent de stage à l'ISIMA.

Table des illustrations

Résumé

Ce stage avait pour but d'ajouter divers outils à la plateforme **web Oros**. Oros est une plateforme permettant de connecter des robots programmés sous **ROS** afin de diriger les agissements de ceux-ci ou d'observer l'activité de leurs différents capteurs. L'organisation du stage a été divisée en trois projets consécutifs : l'ajout d'outils de **test** et de **documentation** (un mois), ensuite l'ajout d'un composant de **visualisation de données** (un mois) et enfin un outil d'**édition de carte** (quatre mois).

Les principaux langages utilisés pour mener à bien ces différents projets ont été le **Javascript** et le **HTML** qui sont tous les deux langages dédiés à la programmation web. La bibliothèque **ReactJS** a été l'outil majeur de programmation pour la réalisation des différents ajouts à la plateforme. Pour l'ensemble de ces projets, le développement a été réalisé avec l'éditeur de texte **Atom**.

A ce jour les trois projets ont été réalisés entièrement. L'application de test et la génération de documentation est automatique et standardisée, deux composants de visualisation de données (pour les valeurs numérique et la vidéo) sont aujourd'hui actifs. Pour le troisième projet seules la gestion de la sauvegarde et la recherche de carte existante n'ont pas pu être réalisées.

Mots-clés : web, Oros, Ros, test, documentation, visualisation de donnée, édition de carte, Javascript, HTML, ReactJS, Atom.

Abstract

The course was designed to add various tools to the **Oros web** platform. Oros is a platform for connecting robots programmed under **ROS** to direct the actions or observe the activity of its various sensors. The organization of the course was divided into three consecutive projects: the addition of **test** and **documentation** tools (one month), the addition of a **data visualization** component (one month) and a **Mapping** tool (four months).

The main languages used to carry out this project were **Javascript** and **HTML**, which are both languages dedicated to web programming. The **ReactJS** library has been the main programming tool for the realization of the various additions to the platform. The development of these tools was carried out with the **Atom** text editor. For all of these projects, the development was carried out with the Atom text editor.

To date, all three projects have been completed. The test application and documentation generation is automatic and standardized, two data visualization components (for digital values and video) are now active. For the third project only the management of the backup and the search of existing map could not be realized.

Keywords: web, Oros, Ros, test, documentation, data visualization, mapping, Javascript, HTML, ReactJS, Atom.

Table des matières

Lexique

Balise HTML : Élément de base du codage HTML permettant de décrire une page web. Elle suit généralement ce schéma :

<TypeDeBalise [option1=valeur1 option2=valeur2 ...] />.

Conteneur logiciel : Environnement spécifique à une application permettant de modifier les paramètres système de manière locale.

CSS : Acronyme de Cascading Style Sheet, permettant de styliser du code HTML.

DOM : Acronyme de Document Object Model, est une interface de programmation permettant à des scripts de lire et de modifier une page web.

Framework : Ensemble d'outils et de facilités regroupés et utilisés en tant que squelette d'un programme informatique.

Front-End : terme employé pour regrouper tous les aspects visuels d'une page web ainsi que le code exécuté par le navigateur. En opposition avec le terme Back-End définissant la programmation d'un site côté serveur.

Git : Utilitaire de gestion de version du code source.

HTTP : Acronyme de Hypertext Transfer Protocol permettant la communication entre un client (exemple: navigateur web) et un serveur internet sur le World Wide Web (WWW).

HTTPS : Protocole HTTP effectué de manière sécurisée.

Incubateur : Ici, incubateur d'entreprise, structure d'accompagnement de projets de création d'entreprise.

Issue : Terme anglais signifiant "question", utilisé dans le jargon pour référencer un souci ou un bug dans un projet informatique.

ORM : Acronyme de Object Relational Mapping, technique de programmation permettant d'utiliser une base de données relationnelle comme une base de données objet.

Pipeline : Suite d'actions effectuées de manière systématique en réponse à une action de départ. Ici une demande de modification de l'application entraîne une suite de vérifications.

Plateforme web : site web permettant de créer et d'utiliser des programmes informatiques.

Serveur NGINX : Serveur informatique utilisant le protocole NGINX pour traiter des connexions multiples en simultané. Il peut aussi être utilisé pour rediriger des URL.

Socket : Interface de communication entre deux appareils.

Socket web : Interface de communication utilisant un protocole web.

SVG : Acronyme de Scalable Vector Graphics, est un langage XML permettant de décrire des graphiques en deux dimensions.

Template : Modèle de style graphique automatique, ici donne une cohérence générale aux documents.

Test : Les tests informatiques permettent de vérifier l'intégrité d'une application au fur et à mesure de son développement.

Transpiler : Convertir du code d'un langage à un autre.

UI : Acronyme de User Interface. Terme employé pour décrire une interface utilisateur et plus largement l'expérience de l'utilisateur par rapport à celle-ci.

Introduction

Ce stage d'une durée de six mois a été réalisé dans le cadre de la deuxième année du cycle ingénieur à l'école ISIMA. L'offre de stage a été transmise aux élèves de l'ISIMA par l'entreprise Easymov Robotics. Jeune startup, son domaine d'expertise est le développement de logiciels pour la robotique mobile

Durant ce stage je serai amené à participer au développement de composants pour une plateforme web dédiée à la robotique. Cette plateforme est réalisée par Easymov Robotics et doit permettre la connexion d'un robot afin de pouvoir observer son activité et celle de ses capteurs ainsi que de le programmer simplement.

Après avoir répondu à l'annonce et passé un entretien, ma demande a été acceptée. L'intérêt de ce stage porte sur les technologies utilisées (web), le cadre de l'entreprise (startup) et le domaine de compétence de celle-ci (robotique). En effet, au vue de l'importance croissante du web aujourd'hui, avoir une expérience dans ce domaine semble être une nécessité. Mais le fait que le domaine d'application soit la robotique donne une dimension supplémentaire très intéressante à ce stage. Enfin le mode de travail dans une startup n'est pas le même que dans une grosse entreprise (type Michelin) et m'intéresse tout particulièrement.

De par mes différentes missions, je serai amené à manipuler et ajouter un framework de test et de documentation à l'application, développer de nouveaux composants pour améliorer des fonctionnalités existantes et pour le développement d'une nouvelle fonctionnalité.

Afin d'expliciter les démarches du travail réalisé, il sera présenté dans un premier temps le contexte du stage ainsi que l'environnement et les outils de développement utilisés, ensuite seront détaillées les différentes étapes de la réalisation des différents projets et pour finir une observation des résultats obtenus ainsi que des pistes de poursuite du projet.

1.Présentation du stage

1.1.Présentation de l'entreprise

Easymov Robotics est une startup fondé en janvier 2017 par deux anciens élèves de l'ISIMA : Noel MARTIGNONI et Gérald LELONG. Après avoir obtenu le statut étudiant-entrepreneur et avoir porté le projet de création d'entreprise au sein de l'incubateur* BUSI, ils se lancent dans la robotique mobile de haut niveau, c'est à dire la conception d'applications permettant de contrôler des robots pouvant se déplacer. Actuellement Easymov Robotics fait partie d'un accélérateur de startup clermontois LE BIVOUAC (Figure 1 : Lieu du stage).

Figure 1 : Lieu du stage

1.2.Présentation de l'existant

Pour bien comprendre le reste de ce rapport, il est important d'expliquer le fonctionnement des principaux outils et applications utilisés pour ce projet.

1.2.1.Introduction à ROS

Robot Operating System (ROS) est un ensemble d'outils open-source permettant le développement d'applications robotiques de tout niveau d'abstraction (du contrôle direct du capteur d'un robot jusqu'à sa simulation dans un environnement 3D). Initialement, ROS a été développé par Willow Garage, un laboratoire en robotique, en 2007. Depuis 2013 son développement est géré par l'Open Source Robotics Foundation (OSRF), une fondation supportant le développement et la distribution de logiciel open-source pour la robotique. Mais le développement de ROS vient principalement de sa communauté. En effet chacun peut librement partager le contenu de ses programmes. Chaque programme partagé ainsi devient open-source et agrandit l'ensemble d'outils disponibles. Chaque version officielle vient standardiser certains outils et ajouter de nouvelles fonctionnalités.

ROS fonctionne uniquement sur linux et plus particulièrement avec la distribution Ubuntu. Le rythme de sortie des versions de ROS suit de moitié celui des versions d'Ubuntu : ROS sort en mai, un mois après la première version annuelle d'Ubuntu. Une version de ROS est adaptée pour une version d'Ubuntu (rapport aux bibliothèques C++). La compatibilité interversion est possible mais n'est pas assurée par l'OSRF. Chaque version de ROS n'a pas le même cycle de vie. La durée d'un cycle correspond au nombre d'années qu'une version sera maintenue. Il existe deux cycles : le premier, dit cycle court, dure deux ans. Il permet aux entreprises de recherche ou souhaitant évoluer rapidement de pouvoir se maintenir aux nouveautés du jour. Le second, dit cycle long, dure cinq ans. Il permet aux entreprises souhaitant assurer un projet stable sur le long terme de ne pas avoir à changer de version trop rapidement.

Le Système ROS est décomposé en trois composantes principales : les nœuds, les topics et le master. Un nœud est un programme simple servant à réaliser une tâche précise. Un topic est un canal où transitent des messages. Ces messages sont publiés par un ou plusieurs nœuds (on parle de nœud *Publisher*) et peuvent être reçus par un ou plusieurs nœuds abonnés à ce topic (on parle alors de nœud *Subscriber*). Chaque fois qu'un message est publié sur un topic il est reçu par tous les nœuds qui y sont abonnés. Un nœud peut publier et écouter simultanément sur plusieurs topics. Le master est un programme système servant exclusivement à mettre en relation les bons nœuds avec les bons topics. Par la suite la communication s'effectue directement entre les nœuds (Figure 2 : Schéma de fonctionnement du master sous ROS).

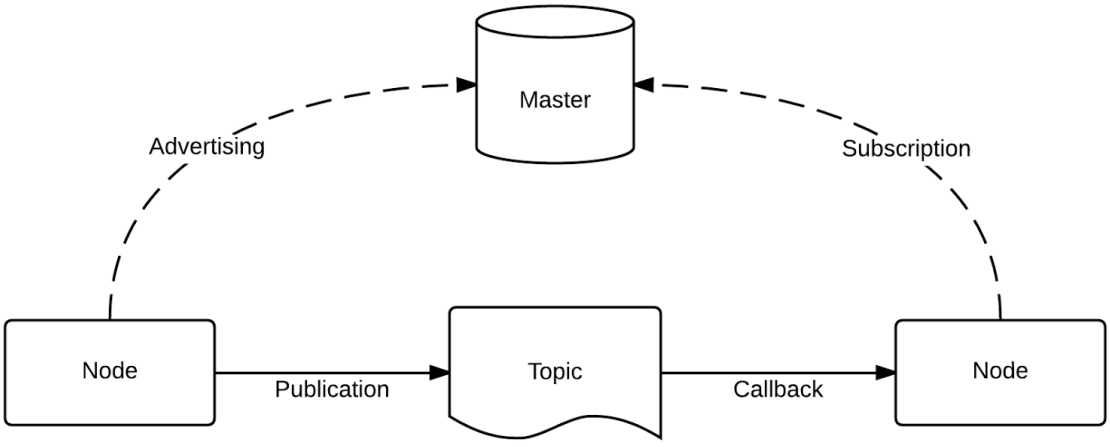


Figure 2 : Schéma de fonctionnement du master sous ROS

Concrètement un programmeur ROS ne développe que des nœuds et se contente de nommer les topics quand il en a besoin. Le master se charge de les créer et de connecter les nœuds (un topics étant en faites une socket* TCP entre deux nœuds). Il existe une autre composante : le service. Celui-ci est une alternative au topic et fonctionne avec une structure "client-serveur" : un nœud effectue une requête pour obtenir une donnée d'un service et celui-

ci la transmet seulement à ce moment-là. Elle est cependant beaucoup moins utilisée que les topics.

La grande force de ROS repose dans son architecture. Celle-ci facilite la création de package et leur partage. En effet il est intéressant pour le fabricant d'un robot de fournir un programme de démonstration utilisable et modifiable par tous. L'autre aspect très intéressant de l'architecture ROS est d'être nativement adaptée à une utilisation distribuée sur un réseau. Cela permet de faire communiquer très simplement plusieurs robots entre eux. Par contre le parti pris de ROS est d'être totalement ouvert. Cela signifie que l'aspect sécurité du réseau est à la charge de l'utilisateur.

1.2.2.Présentation d'OROS

Oros est une plateforme web* créée par Easymov Robotics permettant de connecter un ou plusieurs robots fonctionnant sous ROS. Le but de cette plateforme est principalement de pouvoir observer et administrer une flotte de robot dans le monde industriel.
Il est possible d'observer les différents topics et d'afficher les messages qu'ils contiennent (Figure 3 : Outil de visualisation du système ROS). Il est également possible de dessiner une carte pour paramétrer le déplacement des robots dans une certaine zone (Figure 4 : Outil de création de carte). Enfin il sera bientôt possible de définir des scénarios de tâches à accomplir par les robots en définissant des zones d'intérêt sur la carte (Figure 5 : Outil d'écriture de scénario). Le développement de ces deux dernières fonctionnalités a débuté durant ce stage.

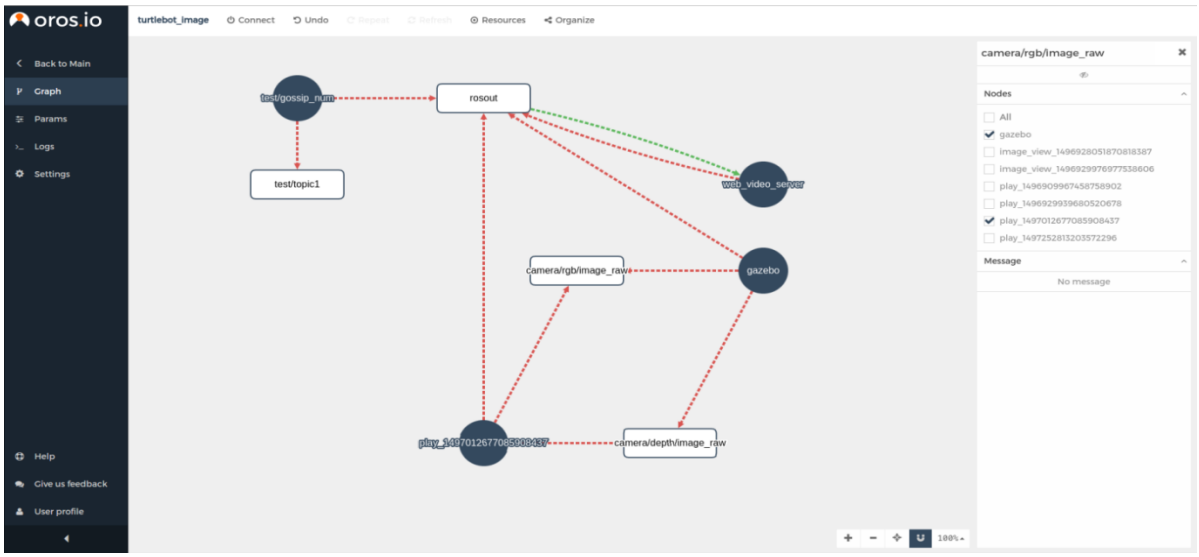


Figure 3 : Outil de visualisation du système ROS

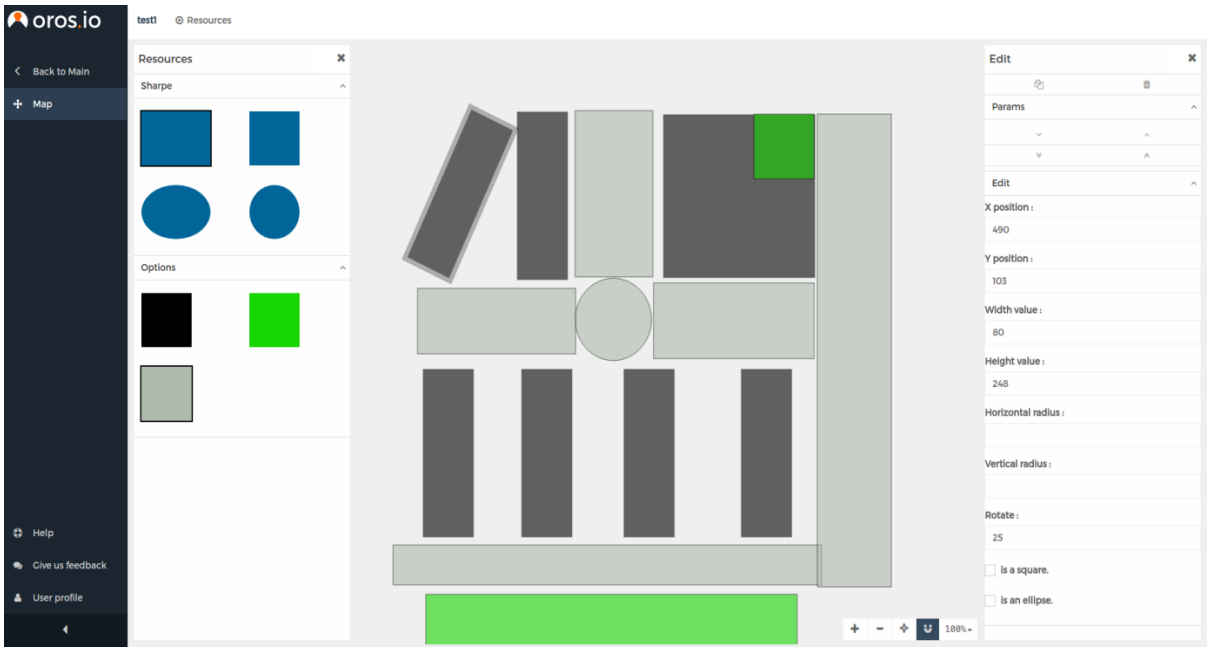


Figure 4 : Outil de création de carte

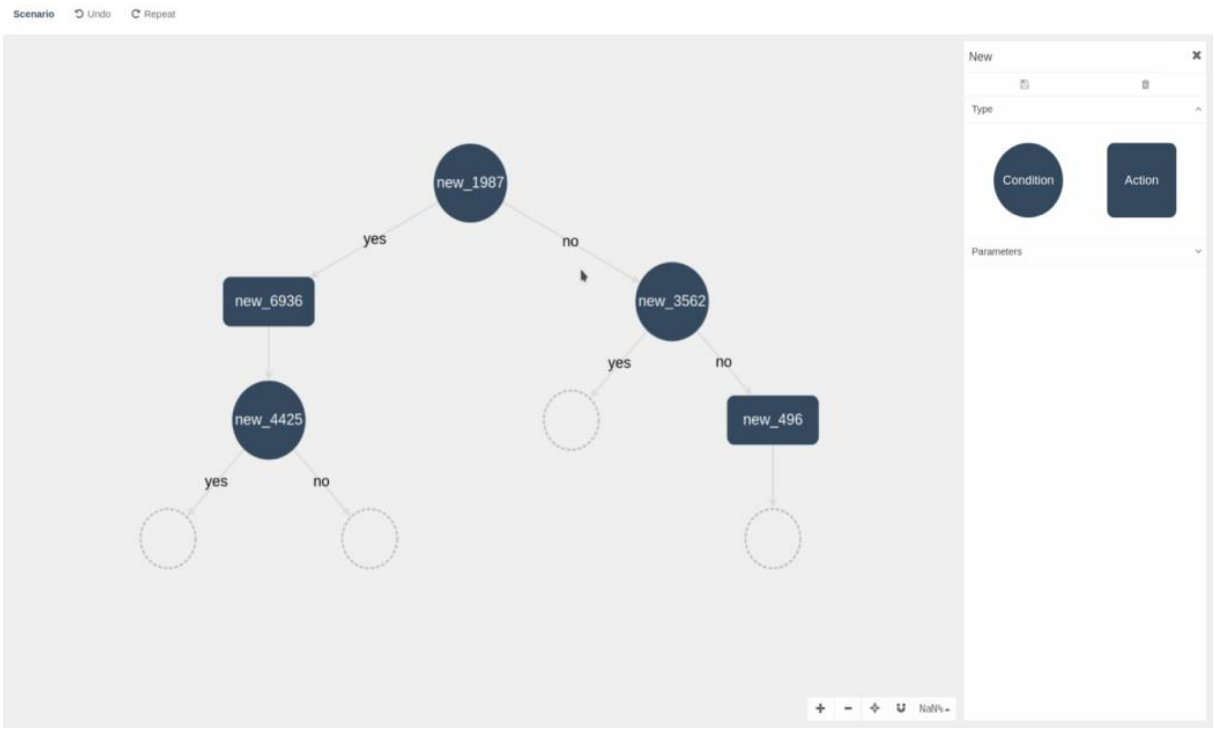


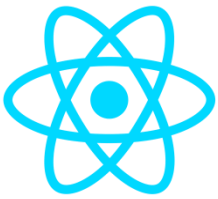
Figure 5 : Outil d'écriture de scénario

1.2.3.Différents outils de développement

Cette partie va faire la liste des principaux outils informatiques utilisés dans ce projet. Il n'y sera décrit que leur fonctionnement général et leur utilité. En complément des informations suivantes, le développement s'est fait sur un ordinateur portable ASUS Rog.

Outils de développement JavaScript :

JavaScript[1] est un langage de programmation permettant, à l'origine, de réaliser des scripts web pour animer une page web. Aujourd'hui le JavaScript peut être utilisé pour réaliser entièrement un site web, on parle aussi d'application web (comme c'est le cas pour Oros). Dans ce projet la version JavaScript utilisée suit la norme ECMAScript6, la dernière en vigueur. Cette norme rend l'écriture du JavaScript plus facile surtout au niveau de la création d'un modèle objet. Seulement cette norme d'écriture n'est pas lisible par tous les navigateurs, il est donc obligatoire de transpiler* le code vers une écriture plus classique du JavaScript avant qu'il ne soit interprété.

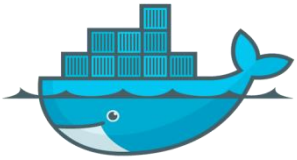


ReactJS[2] est une bibliothèque JavaScript spécialisée dans le front-end* développée par Facebook. Elle permet de créer des composants graphiques réutilisables. L'aspect graphique est défini grâce à des balises html*. Un composant peut contenir un ou plusieurs composants, ceux-ci peuvent recevoir des paramètres au moment de leur création pour adapter leur utilisation. ReactJS a aussi un intérêt pour les performances. En effet là où le JavaScript rafraîchit entièrement une page web à chaque fois que le DOM* est modifié, ReactJS utilise un DOM virtuel qui ne modifie que les éléments qui en ont besoin (par le biais des paramètres de création). Cette technique permet de diminuer le temps de recharge d'une page, ainsi que la quantité de ressource système utilisée.

Django[3] est un framework* python servant au développement de site web. Il permet de créer un site rapidement et de bonne qualité. Ici il est utilisé pour charger entièrement le site Oros lors de son premier chargement (et ainsi éviter de charger l'application à chaque changement de page). Enfin il sert d'ORM* pour gérer la base de données.

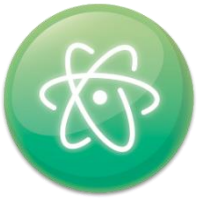
[1] https://www.javascript.com/
[2] https://facebook.github.io/react/docs/hello-world.html
[3] https://www.djangoproject.com/

Outils de gestion de projet :



Docker[4] est un logiciel libre qui automatise le déploiement d'applications dans des conteneurs logiciels*. Cet outil permet de construire des environnements spécifiques à chaque application afin de la tester en environnement réel même si on ne possède pas la machine finale où l'application sera utilisée. Ici cela permet de faire tourner le côté serveur de l'application sur une machine classique afin de simuler les requêtes au serveur effectuées par l'application lors de la phase de développement.

Gitlab[5] est un outil pour gérer les dépôts Git*. C'est aussi une plateforme pouvant servir à la gestion de projet. Ici la découverte d'un bug doit être signalée en créant une issue*. L'autre fonctionnalité utilisée est la mise en place de pipeline* de vérification au moment d'enregistrer des modifications dans la branche principale du projet. Ce pipeline permet d'ajouter des modifications à l'application en s'assurant de son bon fonctionnement. Il est décomposé en quatre étapes : premièrement la création d'un docker sur un serveur privé, deuxièmement la construction de l'application entière et l'exécution des tests dans ce docker, troisièmement rendre la documentation publique et quatrièmement la publication de ce docker sur un serveur public.



Atom[6] est un éditeur de texte réalisé par Github. Il permet, de manière très simple, de gérer des packages à la volée afin d'ajouter de nombreuses fonctionnalités en fonction du type de projet en cours : style et coloration syntaxique en fonction du langage, raccourci d'écriture, utilitaire Git et bien d'autres.

1.3.Organisation du stage

Le sujet de ce stage est le développement et l'ajout d'outils à la plateforme Oros. Il est divisé en trois parties distinctes qui seront détaillées par la suite. La plateforme Oros étant développée en JavaScript, c'est ce langage qui a été le plus utilisé durant ce stage.

[4] https://www.docker.com/
[5] https://about.gitlab.com/
[6] https://atom.io/

1.3.1.Les différentes phases du projet

Ce stage était divisé en trois missions principales. La première fut d'ajouter un outil de gestion des tests* (unitaire, validation, non régression…) ainsi qu'un outil de création de documentation pour l'application et pour le code source. Pour cette mission il fallait avant tout rechercher des outils et sélectionner les plus intéressants (autant pour le gestionnaire de test que pour l'outil de création de documentation).
Pour la deuxième mission il fallait concevoir un composant graphique changeant en fonction du type de topics observé à cet instant. Ce composant général devait faire appel à d'autres composants spécialisés de manière automatique.
Enfin la troisième mission a été de penser et développer un éditeur de carte permettant de représenter l'environnement de travail et les zones de déplacements des robots.

1.3.2.Organisation du stage

Afin de mieux comprendre l'organisation de ce stage, un diagramme de Gantt final a été réalisé (Annexe 1 : Diagramme de Gantt). Les différentes tâches et objectifs de ce stage s'enchaînent selon une certaine logique : leurs difficultés et les connaissances du système nécessaires pour leur réalisation vont de manière croissante. En effet la première mission était d'ajouter des éléments externes à l'application (connaissance des outils sans avoir à connaître l'architecture de l'application), la deuxième d'améliorer une fonctionnalité existante en ajoutant des composants (connaissance des composants et de la manière dont ils interagissent entre eux) et la troisième de créer entièrement une nouvelle fonctionnalité (connaissance de l'organisation complète de l'architecture de l'application).

2.Réalisation et conception

Dans cette partie, il sera présenté les différentes missions effectuées de manière détaillée, une par sous-partie.

2.1.Mise en place d'outils

2.1.1.Les tests

Les tests dans un programme informatique servent à vérifier le bon fonctionnement du programme de manière générale jusqu'à l'exactitude des résultats obtenus. Il existe plusieurs niveaux de test :

- unitaire (pour vérifier le résultat d'une fonction)
- intégration (vérifier que l'ajout d'un composant du programme ne perturbe pas le reste du système déjà en place)
- système (permet de tester un scénario d'utilisation)
- acceptation (ou test utilisateur)

De plus les tests d'un même niveau peuvent se différencier par leur nature :

- fonctionnel (comportement du logiciel)
- compatibilité (installation, droit et autorisation)
- robustesse (utilisation de l'application dans des conditions extrêmes)
- performance (regard sur les performances réelles)
- montée en charge (utilisation massive sur le réseau)
- ergonomie (utilisation classique et apparence)
- interface graphique (détection des régressions)

Les outils de test ajoutés durant cette mission sont des outils permettant de réaliser tout type de test. Il faut savoir qu'en JavaScript et plus généralement dans le développement web, les outils de test ne se sont démocratisés que récemment et il n'en existe donc pas beaucoup. De plus la bibliothèque React étant très jeune, il y a peu d'outils.

Pour tester ce projet, après avoir essayé de nombreuse méthodes et l'association de plusieurs outils plutôt qu'un seul, le choix s'est porté vers le framework de test **Jest**[7]. Celui-ci est développé par Facebook (qui est aussi à l'origine de React). Il permet de combiner très simplement plusieurs types de test car il est fait exclusivement pour la bibliothèque React. Il se combine aussi avec des frameworks de test plus dédiés comme Enzyme ou Jasmine. Ils permettent d'effectuer :

[7] https://facebook.github.io/jest/

- des tests unitaires et la réaction des composants face à différents événements (clic de souris, modification de variable …)

```
describe('Product', () => {
 it('Should render product with name, price and stock', () => {
   const testedComponent = renderIntoDocument(
     <Product
       product={{}}
       withStock={false}
     />
   );
   const productTitle = findRenderedDOMComponentWithTag(testedComponent, 'h3');
   expect(productTitle.textContent).toEqual('');
 });
});
```

Exemple de fonction de test unitaire

- du shallow rendering, c'est à dire tester un composant sans prendre en compte ses composants enfants.

```
import { shallow } from 'enzyme';

describe('<MyComponent />', () => {
 it('should render three <Foo /> components', () => {
   const wrapper = shallow(<MyComponent />);
   expect(wrapper.find(Foo)).to.have.length(3);
 });
 it('simulates click events', () => {
   const onButtonClick = sinon.spy();
   const wrapper = shallow(<Foo onButtonClick={onButtonClick} />);
   wrapper.find('button').simulate('click');
   expect(onButtonClick.calledOnce).to.equal(true);
 });
});
```

Exemple de fonction de shallow rendering

- des snapshot tests pour comparer l'affichage graphique d'une page et de repérer les différences non voulues (régression).

```
test('Link changes the class when hovered', () =>
{
   const component = renderer.create(
     <Link page="http://www.facebook.com">Facebook</Link>
   );
   let tree = component.toJSON();
   expect(tree).toMatchSnapshot();

   // manually trigger the callback
   tree.props.onMouseEnter();
   // re-rendering
   tree = component.toJSON();
   expect(tree).toMatchSnapshot();

   // manually trigger the callback
   tree.props.onMouseLeave();
   // re-rendering
   tree = component.toJSON();
   expect(tree).toMatchSnapshot();
});
```

Exemple de fonction de snapshot test

- des mocks (simulation d'un objet pour tester le résultat de fonctions).

```
const myMock = jest.fn();
console.log(myMock());
// > undefined

myMock.mockReturnValueOnce(10)
.mockReturnValueOnce('x')
.mockReturnValue(true);

console.log(myMock(), myMock(), myMock(), myMock());
// > 10, 'x', true, true
const filterTestFn = jest.fn();

// Make the mock return `true` for the first call,
// and `false` for the second call
filterTestFn
 .mockReturnValueOnce(true)
 .mockReturnValueOnce(false);

const result = [11, 12].filter(filterTestFn);
```

Exemple de fonction de mock

Il est possible d'effectuer ces différents tests avec un framework dédié pour chaque, mais Jest à l'avantage de tout réunir en un seul. De plus il s'installe très facilement sur un projet React et c'est pourquoi il a été choisi pour ce projet. Enfin, lors d'une exécution, les retours console sont clairs (Figure 6 : Capture d'écran d'un retour lors de l'exécution automatique de tous les tests (réussite et échec)).

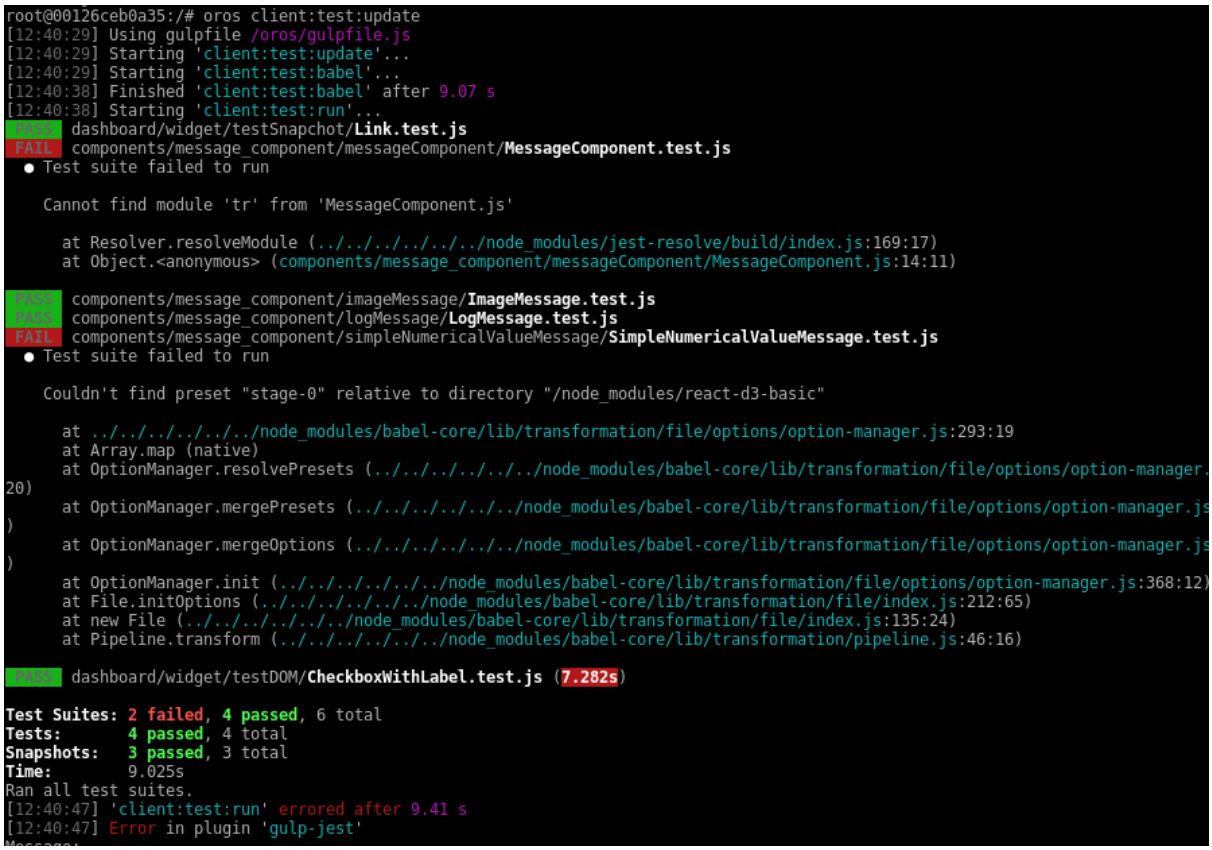


Figure 6 : Capture d'écran d'un retour lors de l'exécution automatique de tous les tests (réussite et échec)

2.1.2.La documentation

La création d'une plateforme et plus généralement d'un projet destiné à être utilisé par d'autres doit être muni d'une documentation efficace et précise. Ici il a été mis en place deux documentations différentes : une pour documenter le code source et décrire les fonctions et une pour décrire le fonctionnement de l'application. Une description de ces deux documentations sera faite ci-après. Il est important pour ce projet d'avoir deux sources de documentation distinctes car elles ne sont pas destinées aux mêmes personnes. En effet la documentation concernant le code source est pour les programmeurs qui continueront le développement et la documentation concernant l'application elle-même est à destination des futurs utilisateurs.

Pour la documentation de l'application en elle-même, l'outil **Sphinx**[8] a été utilisé. Il permet de réaliser simplement des documents en les écrivant au format *Restructured Text* (langage de balisage léger* proche du Markdown). De plus il permet facilement de lier des documents entre eux par des liens. Le format final du fichier est le HTML ce qui permet un partage et une lecture par tous.

[8] http://www.sphinx-doc.org/en/stable/

```
Introduction
============

.. toctree::
  :maxdepth: 2

  Content : <messageComponentIntroduction>


Description
-----------
This react component is use to display topic's informations one your workspace.
It need a list of all messages you want display and a topic's type. In function of this type the
messageComponent render an appropriate component for informations.
Often just last informations are display.

Props
-----
When you would build a messageComponent, you need to inform the different props :
 * list_message : List of the receive messages. Contains all messages.
 * count : Number of messages.
 * topic : Topic's name.
 * type : Topic's type.


Render
------
In this part we will explain when, which and how the messageComponent renders
```

Exemple d'écriture au format ReST

JSDoc est l'outil de documentation d'un code JavaScript le plus répandu. A l'aide de quelques tags spécifique dans un commentaire, il génère une documentation complète. De la même manière que Sphinx, le format final choisi des fichiers est le HTML pour les même raisons.

```
/**
 * Generate SVG element for rendering.
 * @param  {Array} elt - element description
 * @return {svg}     SVG element
 */
```

Exemple de commentaire avec JSDoc

Afin de conserver une certaine unité de ce projet, le même template* visuel a été utilisé pour la génération de la documentation : **Read The Doc** (RTD[9]). Il a l'avantage d'être clair, simple et assez répandu (ainsi les utilisateurs ne sont pas perdus lors des premières lectures (Figure 7 : Capture d'écran d'une page de documentation de l'application avec le framework RTD).

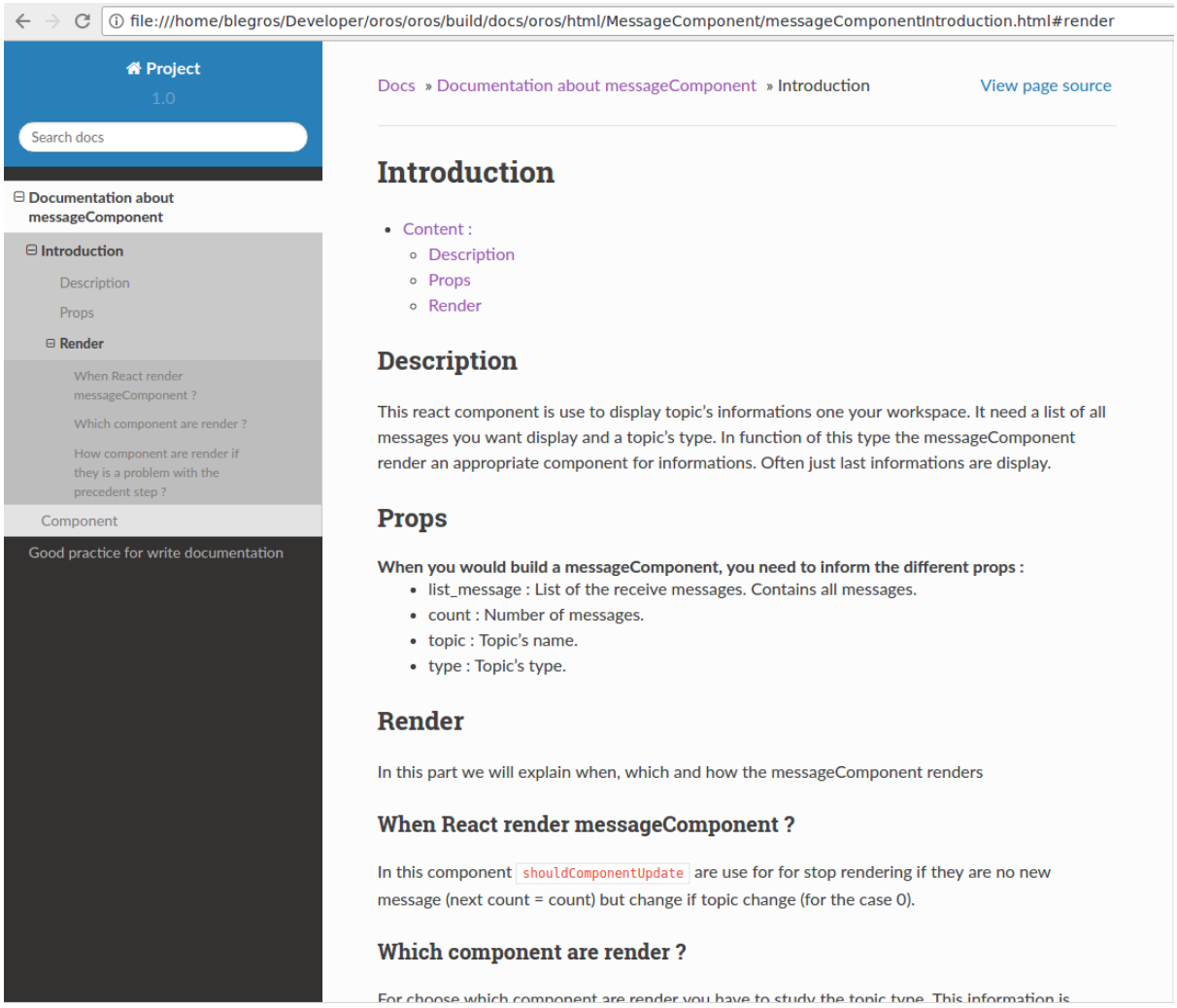


Figure 7 : Capture d'écran d'une page de documentation de l'application avec le framework RTD

2.1.3.L'automatisation

L'utilisation de ces outils est très importante pour le bon déroulement d'un projet et l'expansion d'une application. En effet les tests permettent de prévenir des potentiels bugs créés involontairement à l'ajout d'une fonctionnalité. Cela arrive très fréquemment. Ensuite la

[9] https://readthedocs.org/

documentation permet de gagner du temps à la modification d'un code existant ou lorsqu'un projet n'est pas réalisé entièrement par une seule personne. Pour ces mêmes raisons il est important que tous ces éléments gardent un même style tout au long du projet afin qu'une vieille documentation puisse autant servir qu'une nouvelle par exemple. Cependant réaliser des tests et écrire une documentation sur la durée n'est pas une chose facile et cela est même très contraignant pour les programmeurs. De plus, sur de longs projets, il est parfois indispensable de mettre à jour la forme de la documentation afin d'obtenir quelque chose de plus contemporain et plus utile. Mais ce genre d'activité est fastidieuse et souvent très dure sans perte d'information.

La solution ici est d'utiliser **Gulp**[10]**,** c'est un automatiseur de tâche, c'est à dire un outil permettant de lancer plusieurs scripts ou fonction dans un certain ordre, à la suite d'une commande shell ou à l'exécution d'une certaine action (exemple : à l'enregistrement d'une page de code). Grâce à l'ajout de plugin, il est possible de réaliser toutes sortes de tâches allant de l'application automatique d'une page de style, à la création d'un projet entier, en passant par la génération automatique de documentation.

Gulp est utilisé par le projet général et permet de nombreuses actions comme la compilation automatique du code. Pour ce projet particulier il va permettre de rendre automatique l'exécution de tests, la validation de ceux-ci, le regroupement de la documentation et son formatage. En effet l'écriture de la documentation par le programmeur n'est pas du tout mise en forme. C'est l'application d'un template qui rend la documentation lisible. L'intérêt d'utiliser un template est de pouvoir en changer facilement et d'adapter le style du projet au goût du jour. Il faut seulement garder à l'esprit que Gulp permet de finaliser et exécuter les traitements mais il n'écrira ni les tests ni la documentation de manière automatique.

2.2.Développement de composants

2.2.1.Conception générale

La plateforme OROS permet d'obtenir certaines informations sur les nœuds (Figure 8 : Affichage d'un nœud) et topics (Figure 9 : Affichage d'un topic) d'un système ROS. Pour ce faire elle utilise un ensemble d'outils nommés oros_tools, mis en place en interne. Oros_tools est un nœud ROS utilisant un serveur web servant de pont entre le topics ou nœuds voulant être observé et OROS, ce grâce à une socket web*. Pour récupérer des informations, oros_tools fait appel à un autre nœud : rosapi. Ce nœud permet d'obtenir diverse données sur le système ROS en cours d'utilisation, par exemple la liste des nœuds, topics et services actifs.

[10] https://gulpjs.com/

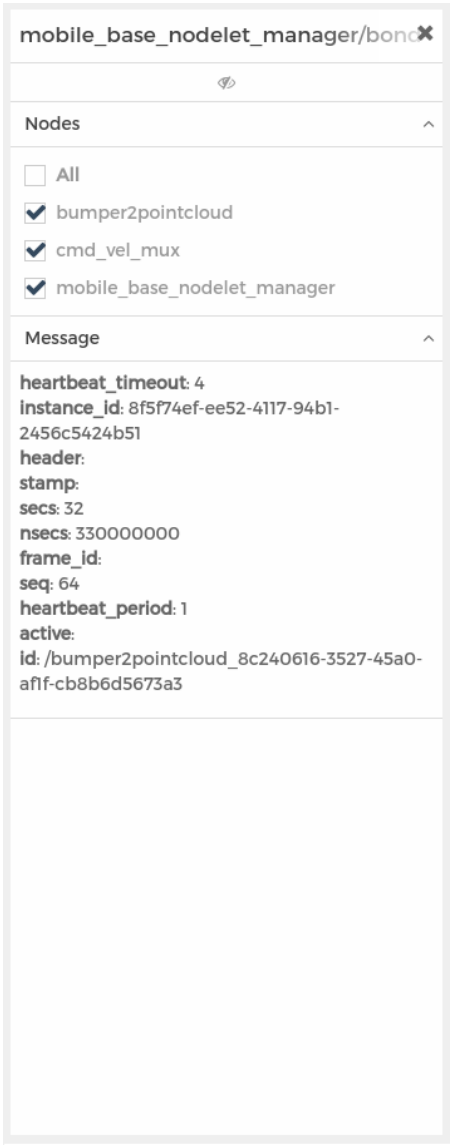


Figure 8 : Affichage d'un nœud

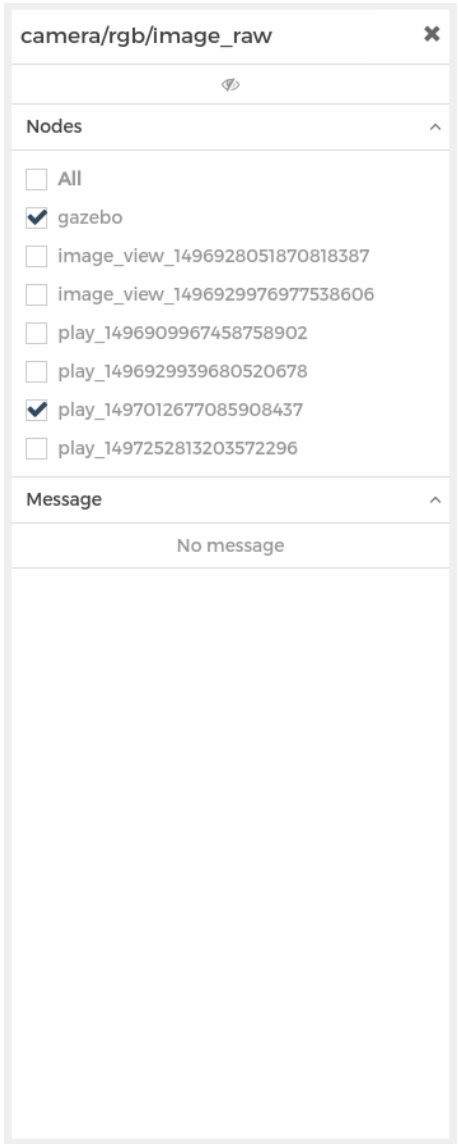


Figure 9 : Affichage d'un topic

2.2.2.Cycle de vie d'un composant React

Un composant React existe à partir du moment où il est visible. Au moment de sa création il est dit qu'un composant est "monté". Il est alors possible de donner des paramètres au composant par le composant parent appelé "props". Le montage d'un composant appelle les fonctions React suivantes :

- constructor() : constructeur de la classe objet représentant le composant React.
- componentWillMount() : permet d'appliquer des modifications aux paramètres avant l'affichage.
- render() : affichage du composant.
- componentDidMount : permet d'appliquer des modifications aux paramètres après l'affichage. Ne modifie pas celui-ci.

La fonction render () est celle qui va définir le composant à l'aide de balises JSX et les retourner pour qu'elles soient affichées par le navigateur. Cette étape est appelée le render d'un composant.

Pour mettre à jour un composant il existe deux moyens : premièrement le composant parent est render et donc de nouveaux props sont envoyés au composant enfant. C'est ici la vraie force de ReactJS, si les props sont identiques le composant n'est pas mis à jour. Deuxièmement il existe un autre type de paramètre, interne au composant (contrairement aux props qui ont une origine externe) appelé "state". Une modification d'un state entraîne le render du composant.
La mise à jour d'un composant React appelle les fonctions suivantes :

- componentWillReceiveProps () : permet d'effectuer des actions avant de recevoir de nouvelles props.
- shouldComponentUpdate() : retourne un booléen. S'il est faux, le render du composant est interrompu.
- componentWilUpdate() : permet d'appliquer des modifications aux nouveaux paramètres avant l'affichage.
- render () : met à jour le composant.
- componentDidUpdate () : permet d'appliquer des modifications aux paramètres après l'affichae. Ne modifie pas celui-ci.

Un composant est "démonté" et détruit à partir du moment où il n'est plus visible à l'écran, c'est à dire que le composant parent n'existe plus ou ne l'a pas remonté lors d'un render. la destruction d'un composant React entraîne l'appel de la fonction suivante :

- componentWillUnmount () : permet de libérer les différents objets créés et de fermer les connexions réseaux.

Un composant est décrit par un classe objet JavaScript héritant de la classe React.Component. Les fonctions décrites ci-dessus viennent de cet héritage mais ne sont pas obligatoirement surchargées.

2.2.3.Description des composants

Le composant réalisé durant ce projet, nommé *MessageComponent*, permet d'afficher les messages circulant sur un topic de différentes manières suivant le type de donnée du topic. En effet les données transmises entre deux nœuds sont typées et il est facile au travers des informations transmises par oros_tools et rosapi de connaître ce type. Ce composant n'est actif que lorsque l'utilisateur veut visualiser un topic. A chaque nouveau message passant sur ce topic, le composant parent du *MessageComponent* transmet les messages reçus par ce topic depuis qu'il est observé, le nom et le type de donnée du topic. *MessageComponent* analyse ce type et se charge de transmettre les messages au bon composant enfant pour qu'il les affiche.
Le type est reçu sous une chaîne de caractères de cette forme : *bibliothèque/type*. Une *bibliothèque* regroupe plusieurs *types* en fonction d'un thème (exemples : la bibliothèque *std_msgs* regroupe les messages les plus standards comme les nombres (entiers et décimaux), les booléens et les tableaux. La bibliothèque *sensor_msgs* regroupe tous les messages pouvant

être envoyés par des capteurs physiques tels que la pression d'un fluide, les données d'un laser ou le niveau de batterie restant). Ces deux informations sont extraites de la chaîne puis une première recherche est effectuée sur la bibliothèque, suivie de la recherche finale du composant à utiliser sur le type. Le premier tri sert en cas de plusieurs types ayant le même nom dans des bibliothèques différentes mais ne devant pas être représentés de la même façon.

La difficulté ici était de savoir comment et quand enregistrer les messages afin de pouvoir donner un maximum d'informations à l'utilisateur. En effet pour pouvoir lire les messages sur un topic, Oros doit s'abonner à celui-ci. Mais il n'est pas possible de s'abonner à tous les topics en même temps, cela utiliserait trop de ressource sur le réseau. Donc seul le topic ciblé par l'utilisateur est écouté à un instant T. La partie du programme permettant la connexion avait déjà été réalisé et le travail présenté ici s'appuie dessus.
A l'origine, le composant principal se contentait de recevoir un message et de le transmettre directement. La première version du *MessageComponent* stockait les messages dans un tableau tant que l'utilisateur observait un topic (observer = cliquer sur un topic pour afficher la fenêtre d'observation et la partie masquable permettant de visualiser ses messages). Seulement si l'utilisateur fermait la fenêtre d'observation et la rouvrait ou masquait le composant et le réaffichait, celui-ci était d'abord détruit puis reconstruit ce qui entraînait la perte des données précédentes. Hors ce comportement n'était pas souhaité. Il fallait , tant que l'utilisateur ne changeait pas de topic, que les données soient utilisables. La solution a donc été de modifier le composant principal afin qu'il enregistre lui-même le tableau de messages et le transmette entièrement au *MessageComponent* à chaque nouveau message arrivant.

Ci-après la liste des composants développés dans ce projet en plus du *MessageComponent*. A titre d'exemple, le code des différents composants présentés ici est disponible en annexe (Annexe 2, Annexe 3, Annexe 4).

SimpleNumericalValueComponent :
Ce composant (Figure 10 : Exemple d'affichage du composant : SimpleNumericalValueComponent) permet d'afficher sous forme d'un graphique les 10 dernières valeurs reçues sur le topic. La valeur 10 a été choisie car, en terme d'UI*, au-delà de cette valeur il y a une perte de lisibilité par rapport au nombre de données visibles. Les types de données pouvant être lues ici sont les suivant : nombre décimal, nombre entier, booléen, bite, durée. Elles sont exprimées en fonction de leur ordre de réception par Oros.
Pour réaliser le graphique une bibliothèque externe a été utilisée : React-d3. D3 est une bibliothèque JavaScript permettant de faire toutes sortes de dessins, diagramme et autre schéma en SVG*. React-d3 est une adaptation de cette bibliothèque pour pouvoir être facilement utilisée par ReactJS. Mais cette bibliothèque est assez lourde, ce choix n'a donc pas été le premier. Seulement les autres bibliothèques testées présentaient des défaillances ou le rendu final n'était pas satisfaisant.

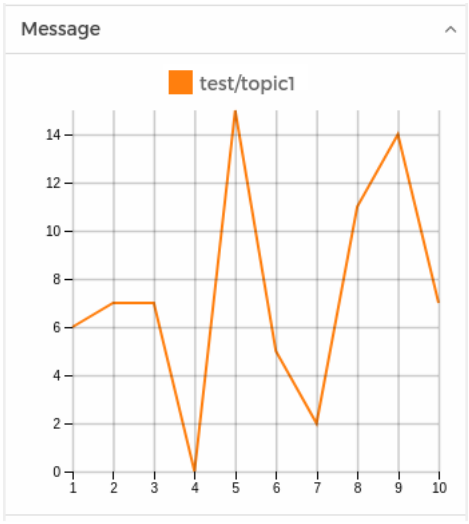


Figure 10 : Exemple d'affichage du composant : SimpleNumericalValueComponent

ImageComponent :

Ce composant (Figure 11 : Exemple d'affichage du composant : ImageComponent) permet d'afficher une image ou une vidéo transférée par une caméra 2D. Pour cela le composant recherche un flux vidéo dispensé par un nœud ros : *web_video_server*. Ce nœud permet de transmettre le flux d'images d'une caméra, une vidéo n'étant, ici, qu'une suite rapide d'images. Ce flux est transmis par le protocole HTTP*. Or la plateforme Oros utilise le protocole HTTPS* qui n'accepte pas les connexions HTTP. Pour résoudre ce problème, un serveur NGINX* a été mis en place, configuré de manière à écouter le port d'arrivée de la vidéo en HTTP (n° 4041) et de le rediriger en HTTPS sur le port n°4040, port écouté par le noeud *web_video_server* pour afficher le flux vidéo.

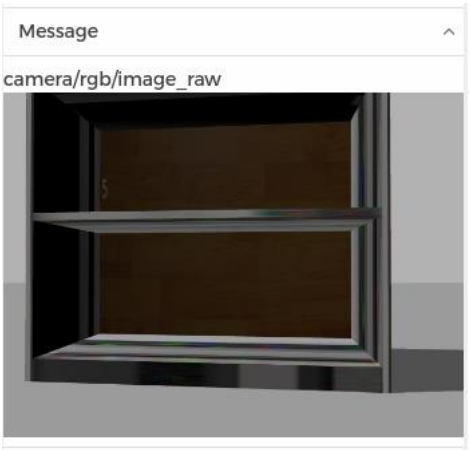


Figure 11 : Exemple d'affichage du composant : ImageComponent

2.3.Nouvelle fonctionnalité : l'éditeur de carte

2.3.1.Description

Le but de ce projet est de pouvoir représenter simplement l'environnement de travail des robots, afin de délimiter leurs différentes zones d'activités. Cette carte va permettre au robot de se situer de manière autonome dans l'espace. Elle sera couplée avec une carte faite directement par le robot en explorant l'espace.
Ce projet s'inscrit dans une suite de fonctionnalité dont le but est de rendre le robot totalement autonome. Il est principalement lié à celui permettant de définir des scénarios d'actions et leur enchaînement automatique.
Pour revenir sur la carte, il existe principalement trois types de zone : les couloirs de circulation, les obstacles et les objectifs. Un couloir délimite un espace où le robot peut circuler. Celui-ci est contraint par une direction de circulation et d'une vitesse maximum. Un obstacle est une zone où le robot ne peut pas aller. Enfin un robot doit se rendre d'un objectif à un autre pour valider un scénario.
Il ne sera abordé ici que la construction de la carte et pas le traitement des données par le robot.

Le code source de l'application est organisé selon un système de vue, dit View. Une vue est composée d'un écran unique permettant d'utiliser une fonctionnalité de l'application auquel un menu est ajouté. Les vues d'une même fonctionnalité sont regroupées sous forme de state et partagent un même menu (exemple non contractuel d'un ensemble de vues pouvant former une state : zone d'édition, paramètres généraux et exportation). Ce menu permet de naviguer dans les différentes vues d'une state mais aussi de changer de state. Les states sont liées entre elles sous forme d'arbre, c'est à dire que depuis une state, il n'est possible d'accéder qu'à sa state parente ou à ses states enfants. La racine de cet arbre est l'écran principal de l'application, celui visible à son ouverture (Figure 12 : Explication du fonctionnement du système de vue sur Oros). Il sera décrit par la suite la state "MapState" ne comportant à ce jour qu'une seul vue : l'éditeur de carte. Cette gestion des différents écrans de l'application est propre à Easymov. La première partie de ce projet a donc été de mettre en place la state de l'éditeur de carte d'environnement.

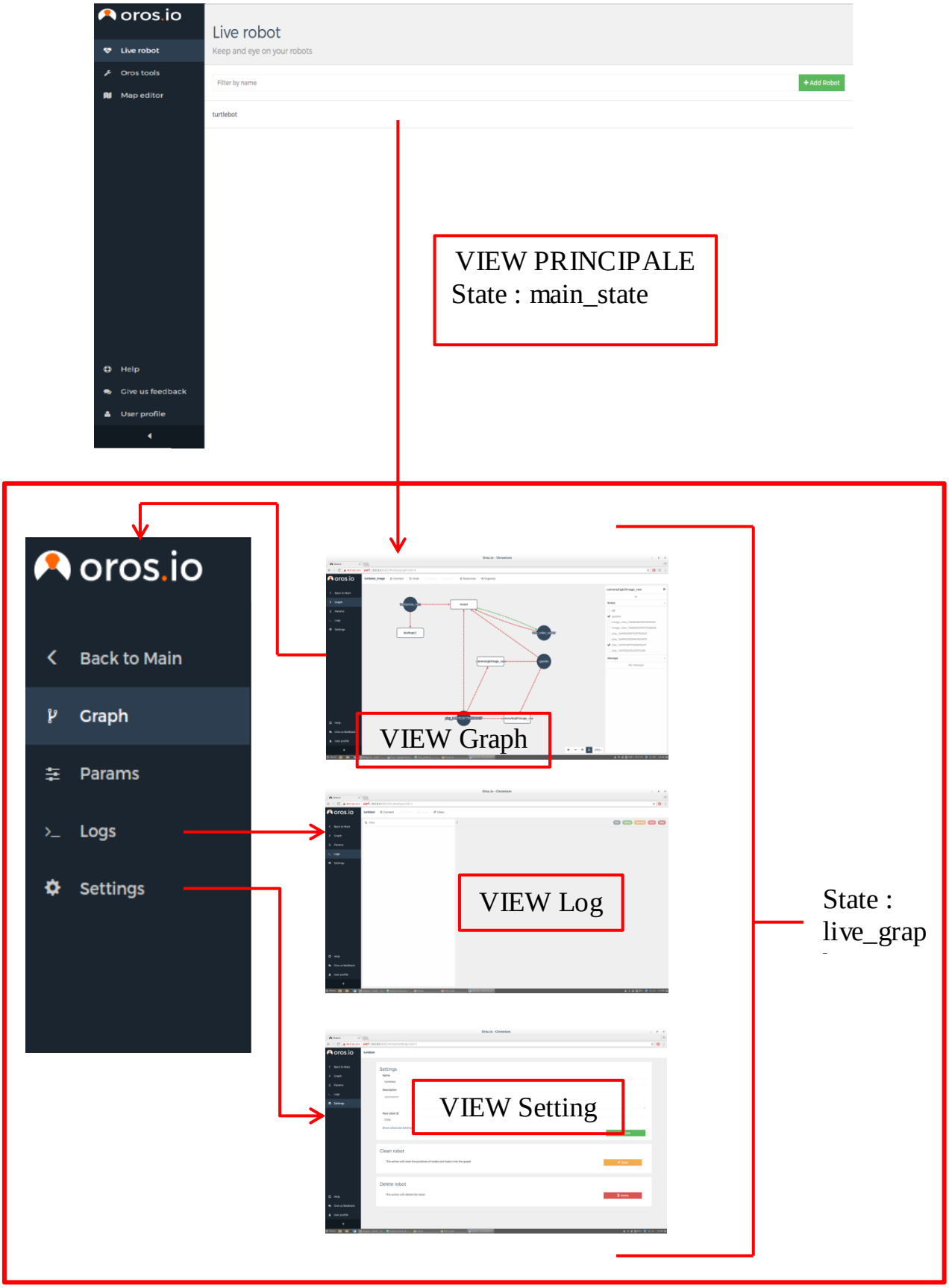


Figure 12 : Explication du fonctionnement du système de vue sur Oros

Afin de dessiner les différentes zones et représenter la carte, il a fallu choisir une technologie qui permettait de réaliser un tel affichage. En HTML5 il existe deux solutions : le dessin vectoriel en Scalable Vector Graphics (SVG) ou le dessin par pixel classique en utilisant Canvas. Ces deux solutions disposent chacune d'avantages et d'inconvénients dans le dessin (Voir Figure 13 : Confrontation entre rendu SVG et Canvas en HTML).

	SVG	Canvas
Type de dessin	Le SVG permet de représenter des formes géométriques simples ainsi que des polygones possédant des arêtes courbes.	Les dessins Canvas permettent de représenter n'importe quelle sorte de forme. La description est souvent faite par un script JavaScript.
Type de stockage	Chaque forme est un objet indépendant.	Tout le canvas est stocké telle une seul image.
Stratégie de modification	Utilisation du DOM JavaScript, la modification est unitaire. Peut être lent.	Chaque modification entraîne un réaffichage total du canvas. Cela en utilisant directement les capacités de l'ordinateur : rapide.
Avantage général	Permet de zoomer et dézoomer sur une forme à l'infini sans perte. Chaque élément est indépendant.	Permet de réaliser des animations fluides.
Inconvénient général	Utilise le DOM ce qui ralentit l'application si trop de modifications simultanées	Les éléments sont seulement dessinés sur le Canvas et ne permettent pas d'être modifiés indépendamment les uns des autres.

Figure 13 : Confrontation entre rendu SVG et Canvas en HTML

Aux vues de ces différents aspects c'est le dessin SVG qui fut choisi. En effet celui-ci permet plus facilement de modifier chaque objet individuellement. De plus l'utilisation de React et de son DOM particulier permet de rendre plus rapide la mise à jour d'un élément SVG.

Seulement ici, l'utilisation du SVG permet juste le rendu final et rend l'expérience utilisateur plus simple. En pratique, un élément SVG est décrit par un objet JavaScript reprenant les différentes caractéristiques d'un élément SVG classique plus certains ajouts (comme le type de zone et la possibilité de sauvegarde par exemple). Cet objet est simplement nommé Forme (Annexe 5). Une Forme permet de représenter un rectangle. Les valeurs de longueurs et de largeurs sont libres mais peuvent être contraintes à être égales afin d'obtenir un carré. Il est aussi possible d'appliquer un arrondi aux coins du rectangle et même de les contraindre afin d'en faire une ellipse et, dans le cas d'un carré cela donne un cercle. Ces

transformations sont rendues très simples par l'utilisation du SVG. Une fonction de cet objet permet de retourner un élément SVG <rect /> qui sera affiché et manipulable.

2.3.2.Interface principal

Ci-après une image montrant l'éditeur de carte légendée et l'ensemble de ses fenêtres, afin de mieux comprendre les prochaines explications (Figure 14 : Capture d'écran de l'outil d'édition de carte légendée).

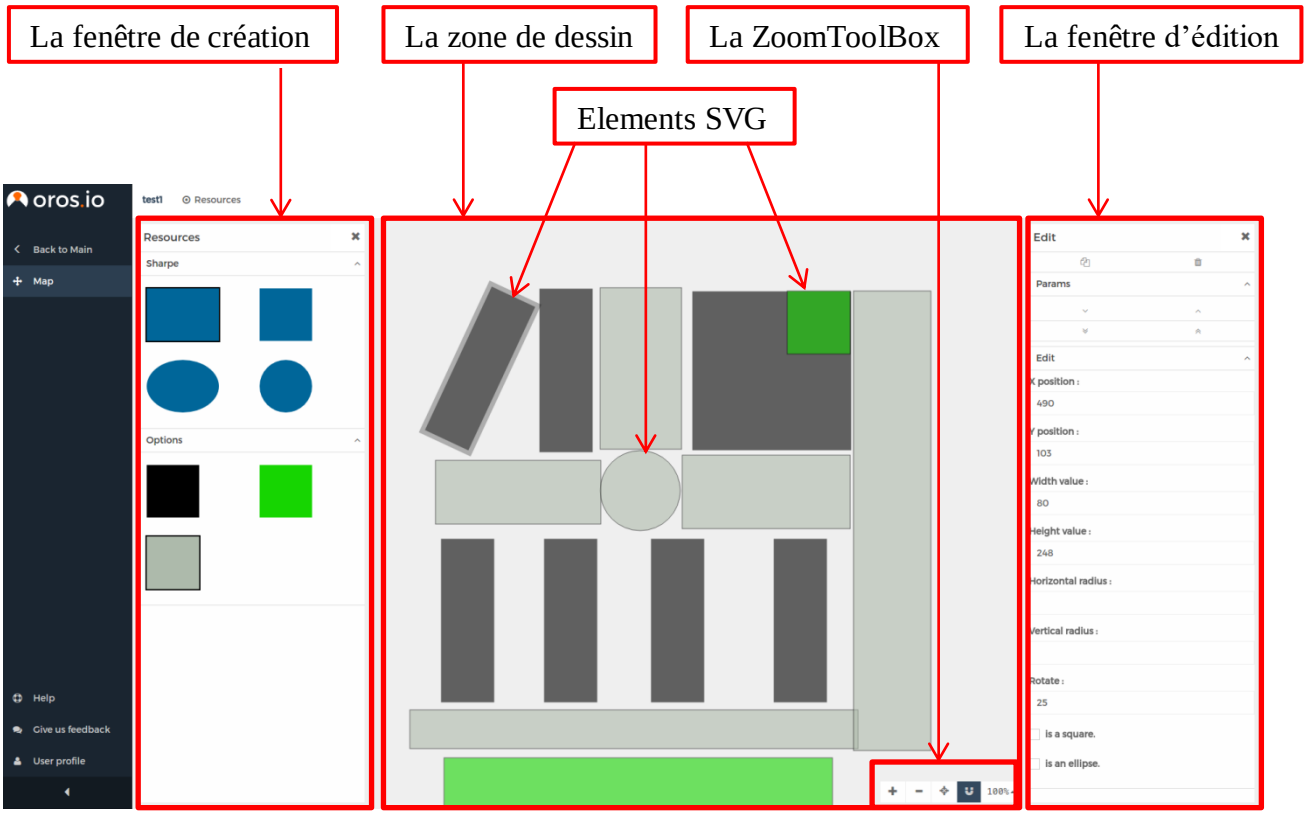


Figure 14 : Capture d'écran de l'outil d'édition de carte légendée

La zone de dessin :

C'est la zone principale de cette vue. Cette zone est de type <svg />. C'est à l'intérieur de cette balise que seront ajoutés les différents éléments SVG construits à partir des Formes. Ces Formes sont stockées dans un tableau JavaScript transmis par le composant parent. Ce tableau est vide dans le cas d'une nouvelle carte, sinon il doit contenir les différentes Formes composant déjà la carte.

Un clic sur un élément SVG permet d'ouvrir la fenêtre de modification. Un appui et glissement de la souris sur un élément permet de le déplacer. Cette même action sur la zone en elle-même entraîne le déplacement de tous les éléments.

La fenêtre de création :

Cette fenêtre permet de déclencher la création d'une nouvelle zone. Elle est composée de deux panneaux. La partie supérieure permet de déterminer la forme de la nouvelle zone. La partie inférieure permet, grâce à un code couleur, de déterminer le type de la nouvelle zone : noir pour les obstacles, vert pour les objectifs et gris pour les couloirs.

Pour commencer la création d'une forme il suffit de cliquer sur une forme et de choisir un type de zone. Cela permettra au système de savoir quoi dessiner. La phase de création à proprement parler sera décrite dans la partie suivante.

Les différentes formes dessinées ici sont réalisées directement grâce à du css* ce qui les rend plus polyvalentes et adaptables à la taille de l'écran. Elles sont séparées en deux blocs (les formes et les types) contenus dans un composant React. Ces deux composants sont eux même placés dans un composant délimitant la fenêtre (Figure 15 : Détail des composants de la fenêtre de création). La visibilité de cette fenêtre est gérée par le composant servant de vue principale. Ceci est une utilisation classique de ReactJS et de l'intérêt de ses composants réutilisables.

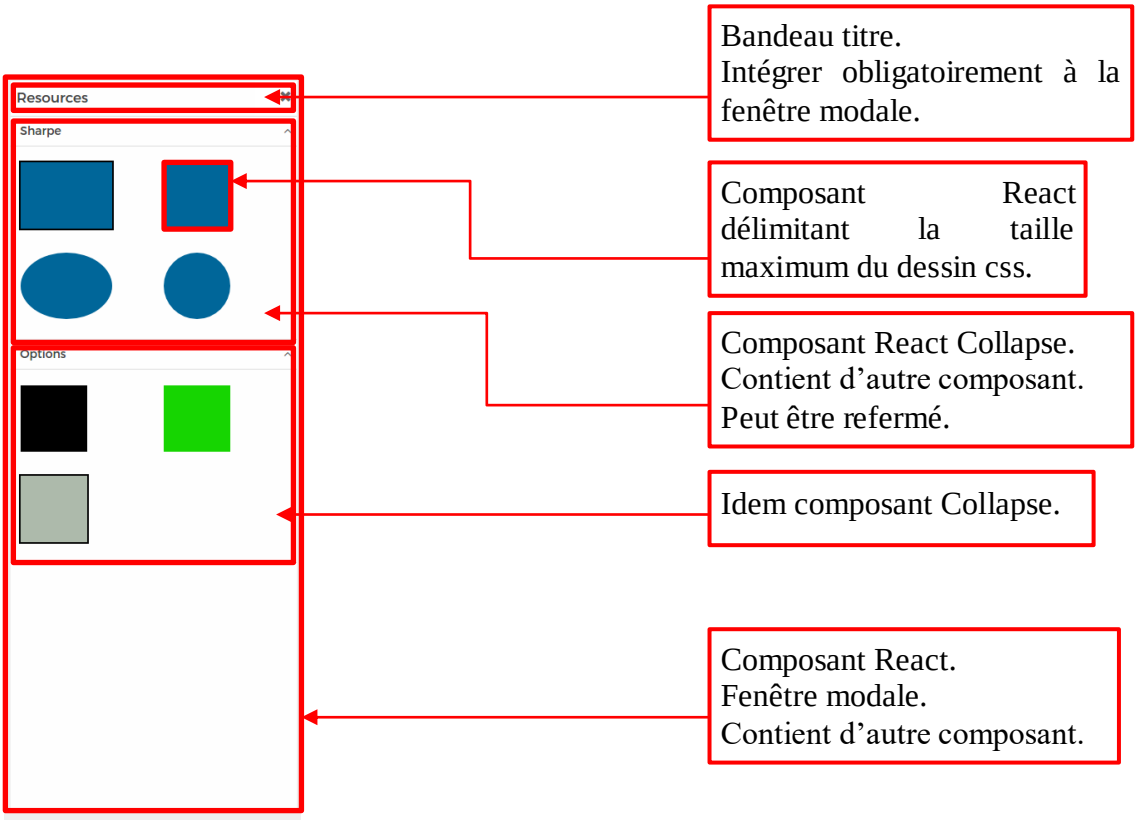


Figure 15 : Détail des composants de la fenêtre de création

La fenêtre d'édition :

Cette fenêtre permet de modifier un élément SVG au travers de l'objet Forme qu'il représente. Les différentes caractéristiques modifiables sont les suivantes : position en x, position en y, longueur, largeur, déformation des angles sur les arêtes horizontales, déformation des angles sur les arêtes verticales, angle de rotation de la forme, les contraintes

carré et ellipse ainsi que l'affichage de l'élément par rapport aux autres dans l'espace SVG (valeur en z).
Cette dernière n'est pas seulement la modification d'une valeur numérique car en SVG les éléments sont empilés les uns sur les autres au moment de la création (le dernier élément créé est celui le plus "au-dessus", le plus visible). Il existe quatre possibilités de déplacement d'un élément : juste au-dessus, juste en dessous, tout en dessous ou tout au-dessus. Ils sont chacun effectués par des fonctions JavaScript permettant de modifier un tableau pour d'ordonner correctement les éléments afin d'obtenir l'empilement attendu.
Grâce aux deux boutons présents en haut de cette fenêtre, il est possible de copier ou supprimer un élément SVG.

Ces trois parties (modification générale, visibilité et copie/suppression) sont placées dans des composants React différents déjà existants. Les champs de modification des caractéristiques sont des composants reprenant un <input /> HTML mais en adaptant leur style (Annexe 6).

La gestion du zoom :
Ce composant permet de gérer le zoom ainsi que le recentrage de l'écran grâce aux différents boutons présents sur cette toolbox (Figure 16 : Détail de la ZoomToolBox). Il agit sur un attribut de la balise <svg /> nommé ViewBox. Celle-ci permet de modifier la fenêtre d'affichage de la zone SVG. Pour bien comprendre son fonctionnement il faut savoir que la taille de la zone SVG est définie à sa création. Cela inclut la création d'un repère dont l'origine (le point de coordonnée (0,0)) est le coin supérieur gauche de cette zone Mais ceci ne représente que la partie visible car l'espace de dessin SVG est infini en deux dimensions. C'est grâce à la ViewBox qu'il est possible de modifier la fenêtre d'observation de cet espace. Elle est composée de quatre attributs : x, y, width et height. Les deux premiers attributs servent à redéfinir l'origine de la zone (coordonnée du coin supérieur gauche), les deux autres servent à modifier la taille d'affichage. Par défaut les attributs de la ViewBox ont pour valeurs : (0, 0, longueur réelle, largeur réelle). Ce qui nous amène la possibilité de deux nouvelles fonctionnalités :

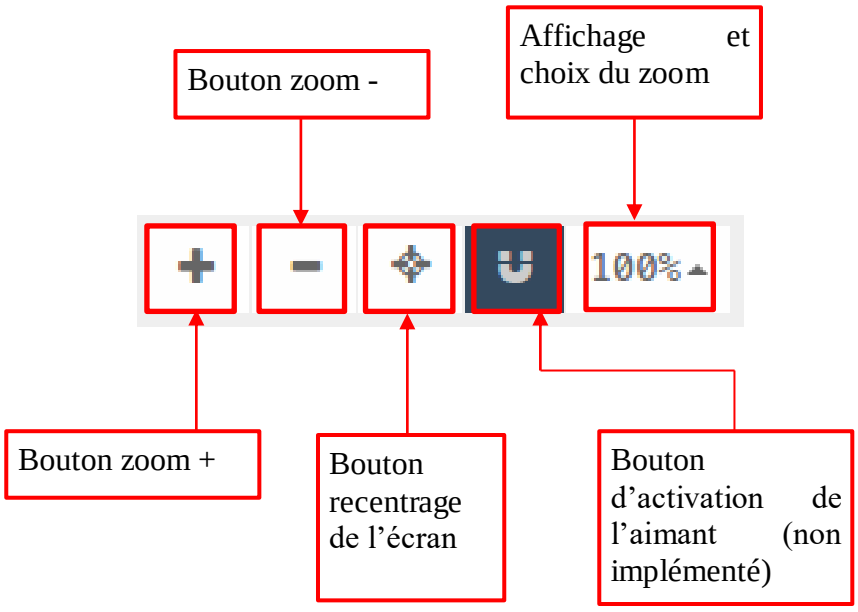


Figure 16 : Détail de la ZoomToolBox

- le déplacement général : simplement en modifiant les valeurs x et y de la ViewBox. Les coordonnées des éléments présents étant fixés par rapport au repère SVG général, ils se déplacent simplement.
- le zoom : les attributs width et height permettent de modifier la taille de la fenêtre d'affichage. Mais attention cela ne modifie pas la taille de la zone SVG qui est réellement affichée dans le navigateur. Prenons un exemple, pour une zone SVG dont les attributs width et height de base valent 100px chacun, par défaut les attributs width et height de la ViewBox valent 100px également (Figure 17 : Exemple d'affichage non zoomé). Maintenant en donnant la valeur 50 aux attributs de taille de la ViewBox, la zone SVG va devoir afficher une fenêtre d'observation de 50px par 50px dans une zone de 100px par 100px. Le système va donc redéfinir le rapport entre le nombre de pixel affiché à l'écran et la taille (en pixel) d'un élément en appliquant cette formule :

$$\text{nombre de pixel affiché pour un pixel de taille} = \frac{\text{taille de la zone d'origine}}{\text{taille de la ViewBox}}$$

Ce pour chaque dimension. Ici le nombre de pixel affiché sera de 100/50 = 2px. Ce qui va entraîner le doublement de la taille de tous les éléments affichés et donc un effet de zoom (Figure 18 : Exemple d'affichage zoomé (120%)). Mais la taille réelle des composants ne change pas, seulement leur affichage. Donner des valeurs à ces deux attributs de la ViewBox plus grandes que les valeurs d'origine entraîne un effet de dézoome (Figure 19 : Exemple d'affichage dézoomé (80%)).

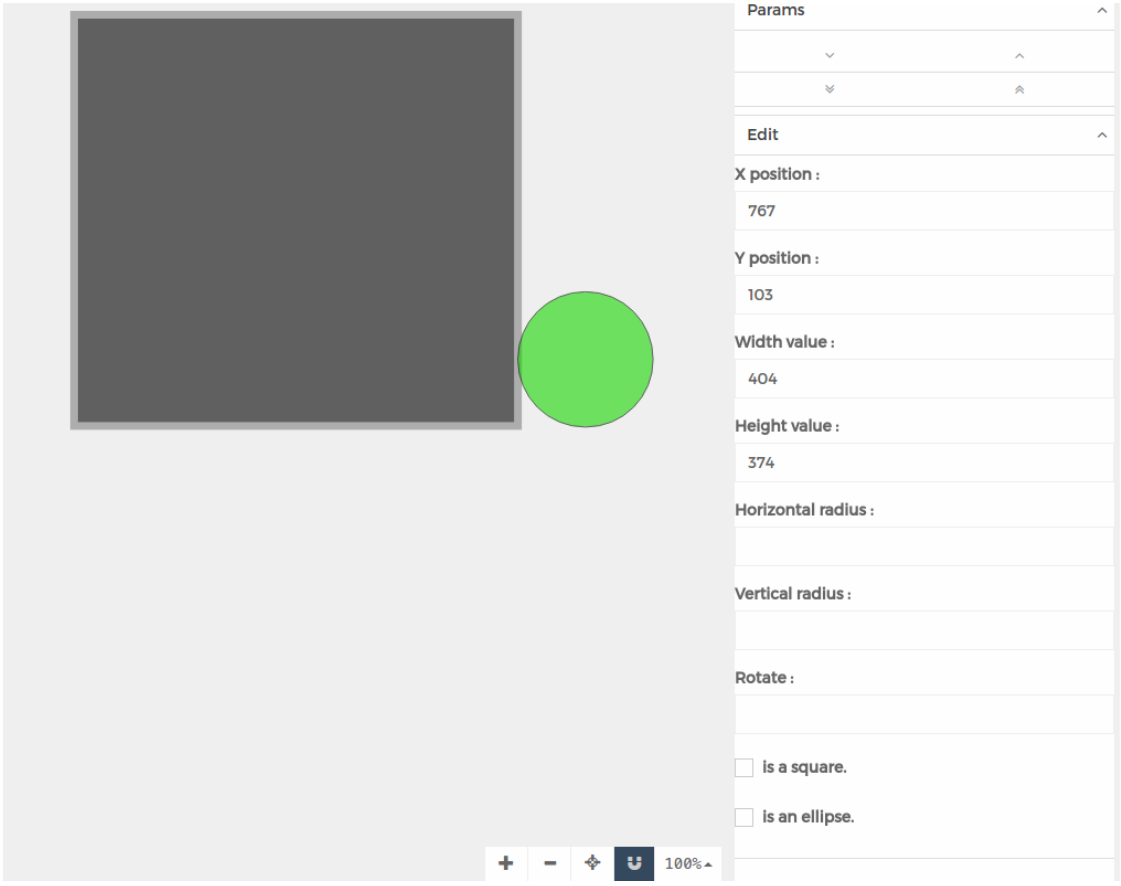


Figure 17 : Exemple d'affichage non zoomé

Taille de la zone de dessin : 100, 100 - Valeurs de la ViewBox : 0, 0, 100, 100

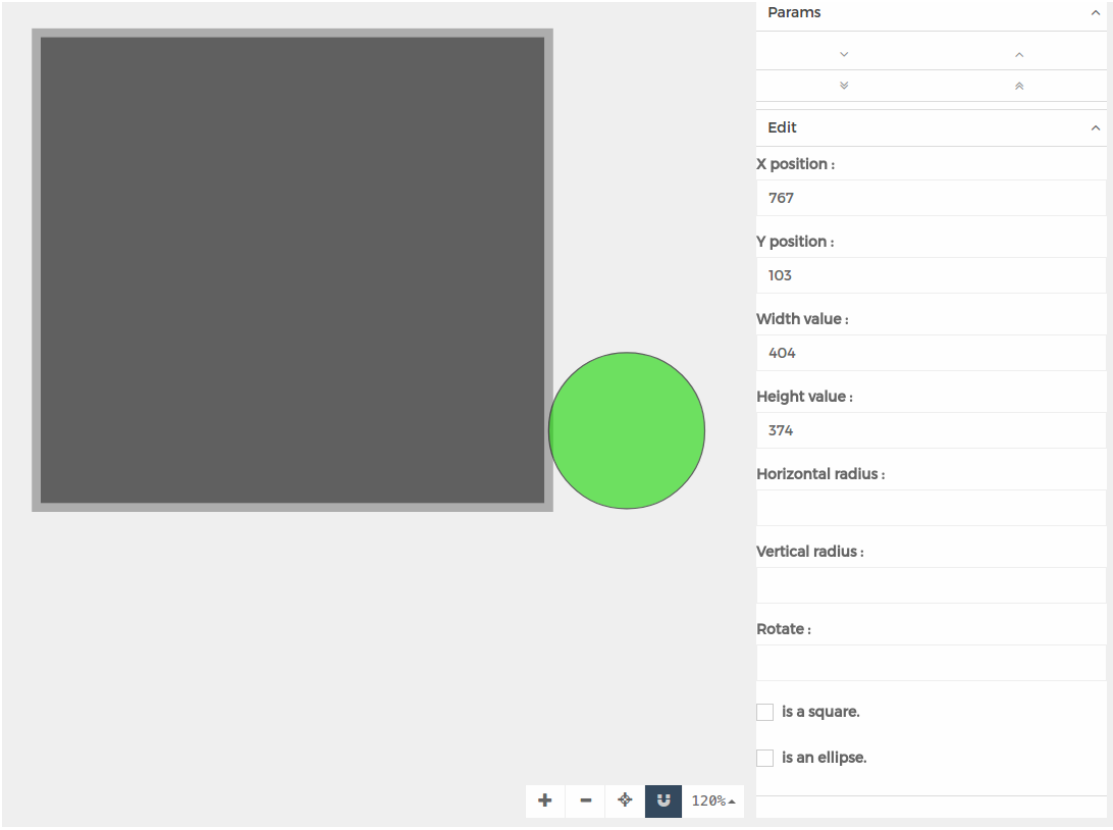


Figure 18 : Exemple d'affichage zoomé (120%)

Taille de la zone de dessin : 100, 100 - Valeurs de la ViewBox : 0, 0, 80, 80

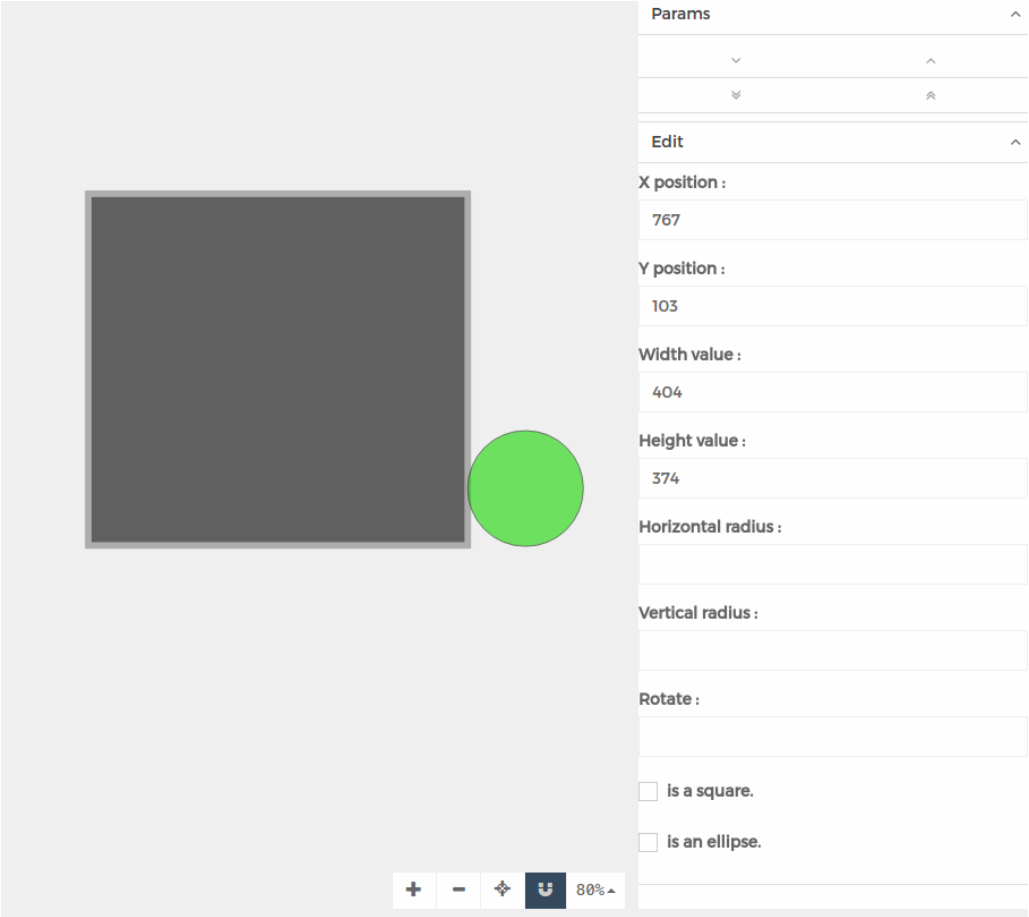


Figure 19 : Exemple d'affichage dézoomé (80%)
Taille de la zone de dessin : 100, 100 - Valeurs de la ViewBox : 0, 0, 120, 120

2.3.3.Fonction de CRUD

Le CRUD est un acronyme signifiant Create Read Update and Delete. Cela englobe toutes les fonctions de création, utilisation, modification et suppression d'un objet informatique. Dans cette partie il sera expliqué les différentes étapes de création, utilisation, modification et destruction d'une Forme.

La création :
Pour ajouter une zone à la carte il existe deux méthodes : la construction d'un nouvel élément et la construction par copie.
La construction d'un nouvel élément suit une séquence d'actions précise. Dans un premier temps il faut sélectionner une forme et un type de zone dans la fenêtre de création. Cela va entraîner le changement d'un paramètre dans l'espace SVG, le préparant à une création.

Pour créer une forme il suffit d'enfoncer le bouton gauche de la souris à l'emplacement du coin supérieur gauche souhaité sur la zone, de déplacer la souris pour obtenir la taille souhaitée puis de relâcher la souris. L'action d'enfoncer le bouton de la souris va, selon le paramètre reçu, créer un nouvel élément SVG et dessiner une petite forme qui va pouvoir être modifiée par les mouvements de la souris.

A chaque déplacement de la souris les distances entre la position d'enfoncement du bouton et la position actuelle du curseur en hauteur et largeur sont calculées. Ces valeurs sont attribuées au nouvel élément selon le type de forme choisi. Par exemple, pour un rectangle le coin supérieur gauche (origine de la forme) reste toujours à l'emplacement de l'appui et que c'est le coin inférieur droit qui suit le pointeur de la souris.
Au relâchement du bouton de la souris la Forme est enregistrée telle quelle et ajoutée au tableau de Forme. Le paramètre modifié initialement reprend alors sa valeur standard. La zone SVG est mise à jour pour afficher le nouvel élément.

La construction par copie revient à créer un objet Forme ayant les mêmes valeurs d'attributs que l'original. Une telle Forme est ajoutée automatiquement au tableau des Formes à afficher. Par soucis de lisibilité les attributs servant de coordonnées au nouvel élément SVG sont légèrement augmentés afin de ne pas le superposer entièrement à l'original.

L'ajout tardif des fonctionnalités de zoom et déplacement général a introduit quelques erreurs dans la construction d'un nouvel élément. Car si la ViewBox a ses valeurs par défaut alors un clic aura les mêmes coordonnées sur la zone SVG que dans l'espace SVG. Mais si la fenêtre d'observation de l'espace SVG est déformée ou déplacée, les coordonnées ne correspondront plus. Pour résoudre ce problème la formule suivante a été utilisée :

$$\text{Coordonnées dans l'espace SVG} = \text{clic}(x,y) \times \text{inverse du grossissement décalage}(x,y)$$

avec :
clic(x,y) les coordonnées du clic sur la fenêtre SVG,
grossissement en pourcentage (exemple : zoom 110% → inverse = 90%),
décalage(x,y) les coordonnées actuelles de la fenêtre d'observation.

Cela a aussi impacté le calcul de la taille d'un nouvel élément lors de sa création, problème résolu avec une formule similaire (sans tenir compte du décalage par exemple).

3.Résultat et discussion

3.1.Réalisation des projets

Dans cette sous-partie sera fait une rétrospective des différents projets, des principaux outils utilisés pour les réaliser ainsi que les résultats obtenus.

3.1.1.Ajout de test et de documentation

L'objectif de ce projet était d'ajouter des outils, d'une part, afin de pouvoir mettre en place des tests sur l'application Oros et d'autre part, permettant de standardiser la documentation de cette même application. Ce projet n'était pas orienté programmation mais demandait d'ajouter des outils au système global. Les principaux outils déjà intégrés au projet ayant servis ici sont Docker (responsable de l'environnement de développement) pour l'ajout de nouvelles variables globales pour les test et Gulp (automatisation de tâches) pour automatiser la création de documentation et l'exécution des tests.
Ce projet a été mené à bien et il est aujourd'hui possible d'effectuer ces différentes actions. Les tests sont effectués par le framework Jest qui assure tous les types de test sur la bibliothèque ReactJS. La documentation est générée par JSDoc et Sphinx avec un même template : ReadTheDoc.

3.1.2.Développement de composant graphique

Ce projet consistait à concevoir et développer de nouveaux composants React permettant de représenter des données transmises sur un topic différemment en fonction de leur type. Deux types ont été traités ici : les valeurs numériques (entier, flottant, booléen, durée …) et les images (photo et vidéo). Le système actuel possédait déjà un composant capable de recevoir les données d'un topic. Il a été modifié afin de pouvoir stocker les données du topic observé (plutôt que seulement la dernière donnée). Ces données sont ensuite transmises à un nouveau composant permettant de les traiter et, en analysant le type, de les transmettre au composant graphique capable de les afficher au mieux. Ici les valeurs numériques sont représentées sous forme de graphique avec la bibliothèque React-d3 (adaptation de la bibliothèque graphique JavaScript D3 pour une utilisation avec ReactJS) et les images sont affichées en temps réel (par un nœud ROS nommé *web_video_server)*.
Il reste de nombreux type de données à traiter mais ils sont affichés de manière générique sous forme de log.

3.1.3.Création d'un éditeur de carte

Ici l'objectif du projet était de mettre au point un éditeur de carte, avec des formes sommaires, permettant de dessiner l'environnement de travail d'un robot. Ceci dans le but de le rendre autonome quant à son déplacement dans l'espace. Il n'a été traité ici que la partie création de carte et non la transformation en données utiles pour un robot. La carte est donc dessinable avec des composants SVG, du dessin vectoriel directement utilisable en HTML5. Ici les éléments SVG ont été réalisés sans framework, de façon native. Chaque composant est modifiable après sa création au travers d'un objet JavaScript : Forme. Cet objet est donc modifiable et sauvegardable par le système. Cette fonctionnalité de sauvegarde n'est pas finie mais le projet ici était de réaliser l'application graphique permettant de dessiner la carte. Cette partie quant à elle est fonctionnelle.

3.2.Poursuite du travail réalisé

Le premier projet est achevé et ne nécessite pas plus d'amélioration.
Le deuxième projet pourrait être amélioré en ajoutant des composants graphiques pour d'autres types de message afin d'avoir une vision vraiment personnalisée en fonction de celui-ci. Il serait aussi intéressant de pouvoir stopper l'affichage de nouvelles données afin de pouvoir étudier le système à un moment précis.
Enfin le troisième projet n'est pas encore achevé et il manque aujourd'hui quelques fonctionnalités qui seront ajoutées par la suite. Il est possible de regrouper ces fonctionnalités manquantes en deux groupes : les graphiques et les systèmes.
Les fonctionnalités graphiques manquantes sont :

- la possibilité de modifier la taille d'un élément directement avec la souris sans avoir à passer par la fenêtre de modification,
- l'ajout d'un "aimant" entre les composants SVG permettant de les aligner les uns aux autres de manière automatique lorsque deux d'entre eux sont à proximité,
- enfin obtenir une meilleure distinction entre les différents types de zone en plus de la couleur (il serait intéressant de faire apparaître le sens de circulation sur une zone couloir par exemple).

Pour les fonctionnalités système manquantes, elles reposent surtout sur :

- l'ajout d'attribut à l'objet Forme afin de différencier leur rôle plus précisément qu'avec une simple chaine de caractère,
- l'aspect sauvegarde d'une carte qui n'a pas été abordé ici car les outils utilisés pour cela sur la plateforme Oros ont changé au cours du développement de ce projet,
- et enfin la mise en place de raccourcis clavier serait un plus.

Conclusion

Ce stage a été réalisé dans la startup Easymov Robotics, spécialisée dans le développement d'application pour la robotique mobile. Durant ces six mois j'ai été amené à travailler sur plusieurs projets, de manière autonome.

Ces différents projets, au nombre de trois, ont été menés à bien : l'ajout de tests et de documentation est aujourd'hui réalisable**,** la visualisation des données lors de l'observation d'un topic ROS est automatiquement adaptée en fonction du type de celle-ci et il est **possible** de dessiner une carte composée de formes simples afin de représenter l'environnement dans lequel les robots seront amenés à évoluer. Seul ce dernier sujet nécessite d'être poursuivi afin d'être utilisable. Cette suite sera assurée directement par les membres d'Easymov Robotics au cours d'une prochaine étape.

Les principales technologies utilisées ici furent le JavaScript et le HTML, deux langages où je n'avais que très peu d'expérience. Débuter sur une application complexe et comprenant déjà de nombreux fonctionnements internes n'est pas forcément simple et l'abondance d'exemples trouvables sur le web proposant des techniques différentes pour un même résultat peut rendre difficile l'assimilation des bonnes pratiques. De plus étant en totale autonomie la plupart du temps, mon rythme de travail était saccadé par des erreurs et incompréhensions souvent basiques, dues au manque de pratique.

Malgré cela, l'expérience d'être responsable d'un projet et de son avancement est très formatrice et appréciable. Cela exprime le niveau de confiance et de responsabilité donné en startup, niveau que l'on obtient plus facilement que dans une grosse entreprise.

Les objectifs des différents projets étant globalement atteints, il reste cependant beaucoup à faire pour rendre la plateforme Oros utilisable dans son ensemble. L'éditeur de carte et le créateur de scénario sont des éléments très importants qui demandent à être améliorés par rapport à leurs versions actuelles. Durant la période de stage suivant la rédaction du rapport je serai amené à continuer le projet d'éditeur de carte pour mettre en place la sauvegarde et la réutilisation de donnée.

Bibliographie

ROS :

https://github.com/RobotWebTools/rosbridge_suite/blob/develop/rosapi/srv/TopicType.srv

http://wiki.ros.org/rosapi

http://wiki.ros.org/turtlebot_gazebo/Tutorials/indigo/Gazebo%20Bringup%20Guide

http://wiki.ros.org/rosbag

http://wiki.ros.org/image_view

http://wiki.ros.org/turtlebot_gazebo/Tutorials/indigo/Make%20a%20map%20and%20navigate%20with%20it

http://wiki.ros.org/web_video_server

http://wiki.ros.org/std_msgs

https://fr.wikipedia.org/wiki/Robot_Operating_System

Test et documentation :

http://blog.xebia.fr/2013/03/21/introduction-aux-tests-unitaires-en-javascript

https://softfluent.fr/blog/societe/les-meilleures-pratiques-du-test-logiciel

https://softfluent.fr/blog/expertise/2016/11/03/les-tests-unitaires-en-javascript

http://blog.lepine.pro/environnement-test-docker-behat

https://www.alsacreations.com/tuto/lire/1686-introduction-a-glup.html

http://www.sphinx-doc.org/en/stable/man/sphinx-build.html

Javascript et graphique :

https://www.w3schools.com/graphics/svg_intro.asp

http://www.petercollingridge.co.uk/interactive-svg-components/draggable-svg-element

http://fontawesome.io/icons/

http://debray-jerome.developpez.com/articles/geometrie-avec-css/

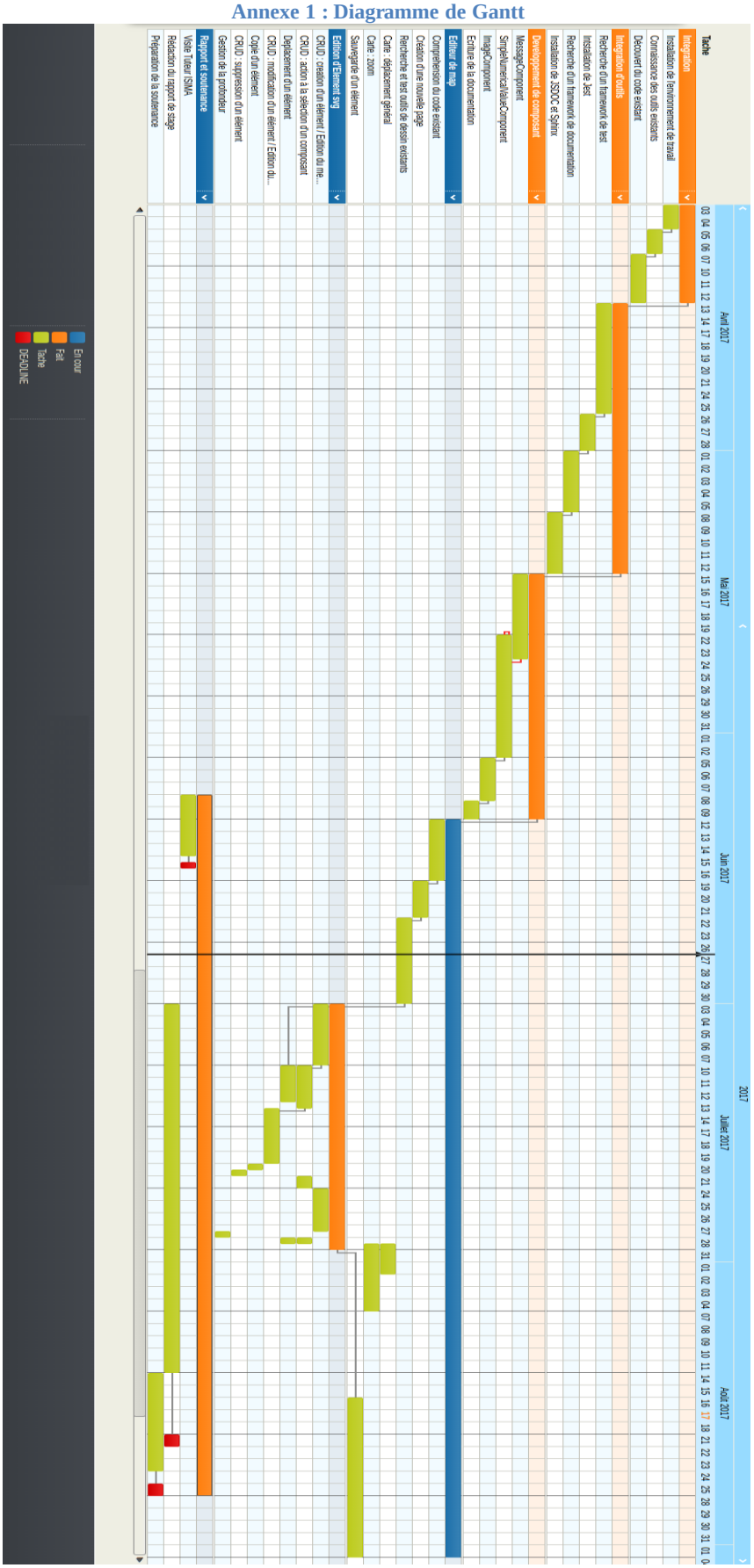
http://vanseodesign.com/web-design/svg-viewbox/

https://facebook.github.io/react/docs/react-component.html

http://www.reactd3.org/

Annexe

Annexe 1 : Diagramme de Gantt



Annexe 2 : Code source du composant React : MessageComponent

```
/**
* Created by blegros on 04/05/17.
*/

import React from 'react'

import tr from 'tr';
import{SimpleNumericalValueMessage} from
'../simpleNumericalValueMessage/SimpleNumericalValueMessage'
import {ImageMessage} from '../imageMessage/ImageMessage'
import {LogMessage} from '../logMessage/LogMessage'

export class MessageComponent extends React.Component
{
/**
 * Construct the messageComponent.
 * Props params :
 * @param {Array} list_message - List of the recive messages
 * @param {number} count - Number of messages
 * @param {string} topic - Topic's name
 * @param {string} type - Topic's type
 */
 constructor (props)
 {
   super(props)

   this.state = {
     howValues: 10,
   };
 }

 /**
  * Prevent the render if they are no new messages
  * @return {Boolean} retrun true if they are a new message or if the topic change, else
return false.
  */
 shouldComponentUpdate(nextProps, nextState)
 {
   return Boolean (this.props.count !== nextProps.count || this.props.topic !==
nextProps.topic)
 }

 /**
  * Render a new composant in terms of the topic
  */
 render()
 {
   let content = '';
   let bibliotheque = this.props.type.split('/')[0];
   let type = this.props.type.split('/')[1];

   // console.log(bibliotheque + " " + type);

   if (this.props.list_message != null && this.props.list_message.length != 0)
   {
     if (bibliotheque === "std_msgs")
     {
       if (type.indexOf("Array") == -1 && (type.indexOf("Float") != -1 |
                                           type.indexOf("Int") != -1 |
                                           type.indexOf("Bool") != -1 |
                                           type.indexOf("Byte") != -1 |
```

```
                                        type.indexOf("Duration") != -1 |
                                        type.indexOf("Time") != -1))
      {
        content =  (
          <SimpleNumericalValueMessage
            messages={this.props.list_message}
            title={this.props.topic}
            howValues={this.state.howValues}/>
        )
      }
    }
    else if (bibliotheque === "sensor_msgs")
    {
      if (type === "Image")
      {
        content = (
          <ImageMessage
            topic={this.props.topic}/>
        )
      }

    }
    else
    {
      content = (
        <LogMessage
          data={this.props.list_message[this.props.list_message.length-1]}/>
      )
    }
  }
  else
  {
    content =  (
      <span className='graph-data-window-content-no-message'>{tr('No message')}</span>
    )
  }

  return (
    <div id='component'>
      {content}
    </div>
  );
 }
}
```

Annexe 3 : Code source du composant React : SimpleNumericalValueMessage

```
import React from 'react';
import ReactDOM from 'react-dom';
var LineChart = require('react-d3-basic').LineChart;


export class SimpleNumericalValueMessage extends React.Component
{
 /**
  * Construct a simpleNumericalValueMessage.
  * Props params :
  * @param {Array} messages - list of all messages
  * @param {string} title - Title of this component
  * @param {int} howValues - number of values to display
  */
 constructor(props)
 {
   super(props);

   this.tabColor = ['#ff7f0e', '#7f0eff'];
 }

 /**
  * Formating informations for diplay
  * @return {Array} Array with format information
  */
 displayData ()
 {
   let data = [];
   let i =  1;

   // recuperer les howValues dernières valeurs
   let tabMsg = this.props.messages.slice(-this.props.howValues);
   for (let c of tabMsg)
   {
     // format les données pour l'affichage
     let x = {value: c.data, index : i};
     data.push(x);
     i++;
   }

   return data;
 }

 /**
  * Render a new SimpleNumericalValueMessage
  */
 render ()
 {
   // style graphique
   var chartSeries = [
     {
       field: 'value',
       name: this.props.title,
       color: '#ff7f0e',
       style: {
         "stroke-width": 2,
         "stroke-opacity": .2,
         "fill-opacity": .2
       }
     }
   ];
```

```
    let marges = {left: 50, right: 20, top: 10, bottom: 20};

    // fonction de retour
    let x = function(d)
    {
      return d.index;
    };

    // recuperation des data a afficher
    let data = this.displayData();

    // recherche du composant parent
    // pour recuperer sa taille
    let element = document.getElementById("component");
    let rect = element.getBoundingClientRect();

    return (
      <LineChart
        width={rect.width}
        height={300}
        data={data}
        chartSeries={chartSeries}
        margins={marges}
        x={x}/>
    )
  }
}
```

Annexe 4 : Code source du composant React : ImageMessage

```
import React from 'react'

export class ImageMessage extends React.Component
{

 /**
  * Construct the ImageMessage.
  * Props params :
  * @param {string} topic  : name of topic
  */
 constructor (props)
 {
   super(props);
 }

 /**
  * Render a new ImageMessage
  */
 render()
 {
   // construction de l'URL
   let src = "http://localhost:4041/stream?topic=";
   src += this.props.topic;

   // recherche du composant parent
   // pour recuperer sa taille
   let element = document.getElementById("component");
   let rect = element.getBoundingClientRect();


   return (
     <div>
       <span className='graph-data-window-item-title'>{this.props.topic}</span>
       <img src={src} width={rect.width}/>
     </div>
   );
 }
}
```

Annexe 5 : Code source de l'objet JavaScript : Forme

```
import React from 'react'

export class Forme
{
 constructor(x, y, z, rx, ry, w, h, c, o, t, g, s, e)
 {
   this.id = Math.random();
   this.x = x;
   this.y = y;
   this.z = z;
   this.rx = rx;
   this.ry = ry;
   this.width = w;
   this.height = h;
   this.color = c;
   this.opacity = o;
   this.rotate = 0;
   this.type = t;
   this.gradient = g;
   this.isSquare = s;
   this.isEllipse = e;
 }

 /**
  * Return a new Forme with exactly the same attributs
  * except x and y position.
  * @return {Forme} the new Forme
  */
 constructorByCopy ()
 {
   let nv = new Forme (
     this.x,
     this.y,
     this.z,
     this.rx,
     this.ry,
     this.width,
     this.height,
     this.color,
     this.opacity,
     this.type,
     this.gradient,
     this.isSquare,
     this.isEllipse
   )

   nv.x += 20;
   nv.y += 20;

   return(nv);
 }

/**
* Generate SVG element with all propertises.
* @return {SVG} SVG element ready for rendering
*/
 generate ()
 {
   return (<rect
     id={this.id}
     x={this.x}
```

```
    y={this.y}
    rx={this.rx}
    ry={this.ry}
    width={this.width}
    height={this.height}
    stroke={this.color}
    strokeWidth={1}
    fill={this.color}
    opacity={this.opacity}/>);
}

/**
 * Set the element width and height in respect of sharpe
 * @param {Integer} dx - new width size
 * @param {Integer} dy - new height size
 */
setSize (dx, dy)
{
  if (this.isSquare)
  {
    let size = parseInt(Math.sqrt((dx*dx) + (dy*dy)));
    this.width = size;
    this.height = size;
  }
  else
  {
    this.width = dx;
    this.height = dy;
  }

  if (this.isEllipse)
  {
    this.rx = this.width/2;
    this.ry = this.width/2;
  }
}

/**
 * Set the element position
 * @param {Integer} nx - new x position
 * @param {Integer} ny - new y position
 */
setPosition (nx, ny)
{
  this.x = nx;
  this.y = ny;
}

/**
 * Adapte the size in function of isSquare
 */
setSizeIsSquare ()
{
  if (this.isSquare)
  {
    this.height = this.width;
  }
}

/**
 * Adapte the rx and ry values in function of isEllipse
 */
setRadiusIsEllipse ()
```

```
 {
   if (this.isEllipse)
   {
     this.rx = this.width/2;
     this.ry = this.height/2;
   }
 }
}
```

Annexe 6 : Exemple de dessin en CSS

```
.rect
{
 width: $sizeMax;
 height: $sizeMin;
 background:$colorBackground;
}

.square
{
 width: $sizeMin;
 height: $sizeMin;
 background: $colorBackground;
}

.circle
{
 width: $sizeMin;
 height: $sizeMin;
 background: $colorBackground;

 -moz-border-radius: $sizeMin;
 -webkit-border-radius: $sizeMin;
 border-radius: $sizeMin;
}

.ellipse
{
 width: $sizeMax;
 height: $sizeMin;
 background: $colorBackground;

        border-top-left-radius: $sizeMax05 $sizeMin05;
        border-top-right-radius: $sizeMax05 $sizeMin05;
        border-bottom-right-radius: $sizeMax05 $sizeMin05;
        border-bottom-left-radius: $sizeMax05 $sizeMin05;
}
```