|  |
| --- |
|  |
| Projet Robot Ambiant |
|  |

|  |
| --- |
| Zheng, Raphaël Bresson, Charles Roger  19/02/2016 |

Table des matières

**Introduction1**

1. **Cahier des charges et analyse fonctionnelle :4**

Tapez le titre du chapitre (niveau 2)5

Tapez le titre du chapitre (niveau 3)6

1. **Designs possibles :4**
2. Design ROS + microcontrôleur5
3. Design FPGA5
4. Comparaison des designs 5
5. **Cahier de recettes :4**
6. **Milestones :4**

**Conclusion1**

Introduction

Les besoins du client

Notre client nous demande de concevoir un robot capable de prendre en compte son environnement. Le robot doit pouvoir être programmé grâce à une API java.

L’interface de programmation doit être suffisamment haut niveau pour que l’utilisateur puisse programmer le robot en JAVA en utilisant cette API sans pour autant maitriser la technologie du matériel embarqué.

Environnement de notre projet

Notre robot sera testé en laboratoire sur sol plat et sans pente, il doit donc être capable d’évoluer dans un environnement intérieur. Il n’y a pas de contrainte d’étanchéité.

Nous nous intéresserons tout d’abord à l’analyse fonctionnelle. Puis nous comparerons les différents designs possibles afin de choisir le plus adapté. Enfin nous détaillerons le cahier de recettes et les milestones du projet.

1. Cahier des charges et analyse fonctionnelle

# Cahier des charges du système

## I) Spécifications

1 – Robot dont le comportement dépend de l'environnement

→ Capteurs (caméra, lidar, …) → détection d'obstacles

2 – Peut attraper des objets

→ Bras articulé + pince

3 – Peut se déplacer

→ roues + moteurs

4 – Constitué de deux cartes :

- Carte HIGH LEVEL (API, calculs, centralisation des données des capteurs + acquisition pour la caméra [et le lidar])

- Carte LOW LEVEL (acquisition des capteurs secondaires + contrôle des actionneurs)

## II) Contraintes

1 – La carte HIGH LEVEL doit pouvoir exécuter JAVA

2 – L'algorithme de la carte LOW LEVEL doit être concurrent

## III) Choix à faire

1 – Protocole de communication entre les cartes LOW LEVEL ou HIGH LEVEL (UART ou ETHERNET)

2 – Choix de la carte LOW LEVEL : Microcontrôleur ou FPGA.

# Diagramme Pieuvre

**FP2**

**FP1**

**FP4**

**FP3**

**FC7**

**FC9**

**FC6**

# Diagramme FAST

Déplacer le système et attraper des objets

Diriger le robot

FP1

Choisir Chemin

FC1

Générer carte des obstacles

FC2

Récupérer les valeurs des capteurs

FC6

Faire avancer le robot

FP2

Commander les moteurs

FC3

Prendre un objet

FP3

FC4

Bouger bras

Commander le système

FP4

Interfacer L’API et la couche bas niveau

FC5

Envoyer la commande

FC7

Caméra/Lidar/capteurs de présence

Hacheur/carte commande de moteur

Servomoteur/ bras mécanique

Carte Firefly(linux embarqué)

Microcontrôleur/ FPGA

1. Designs possibles
2. Design de l’API JAVA

Classes de gestion de capteurs reliés directement à la carte HIGH LEVEL (caméra, lidar)

Ordres

Centre de décision

Classes de gestion de communication vers la carte LOW LEVEL

(ETHERNET ou UART)

Données brutes des capteurs internes

Données brutes des capteurs externes

Classes de génération de la carte des obstacles

(OPENCV)

Carte des obstacles

Code IA

Figure 1 : Design de l’API JAVA

1. Design ROS + microcontrôleur

Wifi

HIGH LEVEL (Carte Firefly avec ROS)

Rôle :

* Décision en fonction des acquisitions de la caméra, du lidar et des capteurs du LOW LEVEL
* Envoi des ordres de haut niveau au LOW LEVEL

LOW LEVEL (MBED ARCH PRO)

Rôle :

* Transformation des ordres de haut niveau en un ordre de bas niveau compréhensible pour les actionneurs.
* Envoi des données des capteurs au HIGH LEVEL

Valeurs brutes des capteurs

Ethernet

Ordres de haut niveau

Ordres de bas niveau

Valeurs brutes

Données brutes

Images

Lidar

Caméra

Actionneurs

Capteurs

Figure 2 : Design haut niveau design ROS + microcontrôleur

On utilise ici deux cartes distinctes :

* La première, appelée HIGH LEVEL, est une carte Firefly exécutant les instructions de haut niveau (décision).
* La deuxième, appelée LOW LEVEL, est une carte Seeeduino Arch Pro (compatible MBED) qui exécutera les instructions de bas niveau (relever les valeurs des capteurs et contrôler les actionneurs).

Pour la communication, nous utiliserons le protocole de transfert TCP/IP via ETHERNET.

Lidar

Caméra

IMU

Proximité

I2C

USB

Ethernet

MBED ARCH PRO

Multiplexeur

Firefly

Ethernet

Appli IA

DirG

PwmG

7 pins GPIO

DirD

PwmD

Alimentation Firefly et Mbed

H-Bridge

Bras articulé

Alimentation actionneurs

Moteurs

roues

Bras articulé

Lidar

*Actionneur Capteur Ordre Données des capteurs*

Figure 3 : Design détaillé pour ROS + microcontrôleur

Données des capteurs

Ordres High Level

Communication

Décision

Données des capteurs

Ordres Low Level

Thread 1 : Aller à gauche

Thread 2 : Accélérer

Thread 3 : Bouger le bras de n degrés

…

…

Figure 4 : design du programme C++ pour le LOW LEVEL

Le bloc communication est constitué d’un ensemble de classes C++ constituant le client TCP ainsi que de deux buffers (mémoire partagée) contenant les valeurs des capteurs et les ordres pour les actionneurs. A noter : deux threads distincts s’exécutent : l’un pour la réception, l’autre pour l’emission.

Le bloc décision est un ensemble de classes qui permettent la conversion des ordres de haut niveau en ordres de bas niveau.

Aucun traitement n’est effectué sur les valeurs des capteurs qui sont envoyées telles qu’elles au HIGH LEVEL.

1. Design FPGA

Wifi

HIGH LEVEL (Processeur JAVA embarqué)

Rôle :

* Décision en fonction des acquisitions de la caméra, du lidar et des capteurs du LOW LEVEL
* Envoi des ordres de haut niveau au LOW LEVEL

LOW LEVEL (Circuits dédiés)

Rôle :

* Transformation des ordres de haut niveau en un ordre de bas niveau compréhensible pour les actionneurs.
* Envoi des données des capteurs au HIGH LEVEL

Valeurs brutes des capteurs

Axi Bus

Ordres de haut niveau

MOJO V3 (SPARTAN 6)

Valeurs brutes

Ordres de bas niveau

Images

Données brutes

Capteurs

Actionneurs

Caméra

Lidar

Figure 5 : design haut niveau pour FPGA

Ici tout est monté sur le FPGA. Pour cela, il est possible de récupérer une IP Xilinx pour des processeurs JAVA embarqués (par exemple les JOP : Java Optimized Processor : <http://www.jopdesign.com/> ).

Pour la gestion des capteurs et des actionneurs nous pourrons utiliser des circuits dédiés écrits en VHDL qui aurons l’avantage de pouvoir fonctionner en parallèle. Pour certains capteurs et actionneurs nous pouvons nous aider d’un « shield » spécifique :

* Le « servo shield » : Pour la gestion des servomoteurs du bras articulé
* Le « caméra shield » : Pour la gestion de la caméra

Ces modules peuvent être trouvés sur le site : <https://embeddedmicro.com/products.html>.

Pour la communication interne nous pourrons utiliser un bus AXI.

Il faudra aussi prévoir deux alimentations distinctes : une pour le FPGA, l’autre pour les actionneurs.

Proximité

Lidar

Caméra

Shield

IMU

I2C

JAVA EMBEDDED PROCESSOR

AXI

CIRCUITS DEDIES

Multiplexeur

AXI

Appli IA

DirD et PwmD

DirG et PwmG

FPGA

Alimentation FPGA

H-Bridge

Servo Shield

Bras articulé

Alimentation actionneurs

Moteurs

roues

Bras articulé

Lidar

*Actionneur Capteur Ordre Données des capteurs*

Figure 6 : Design détaillé pour FPGA

Données des capteurs

Ordres High Level

Communication

+

Buffers

Décision (gestion de la table de vérité)

Données des capteurs

Ordres Low Level

Circuit 1 : Aller à gauche

Circuit 2 : Accélérer

Circuit 3 : Bouger le bras de n degrés

…

Figure 7 : Design du bloc des circuits dédiés …

L’entité de communication contient des entités générées par ISE pour la communication AXI ainsi que des buffers permettant de stocker les valeurs des capteurs et des actionneurs.

L’entité de décision contient la table de vérité des actionneurs et assure la conversion ordres de haut niveau vers ordres de bas niveau compréhensible pour les actionneurs.

Les buffers des capteurs sont actualisés sur front montant pour actualiser le buffer qui sera accessible en lecture pour le HIGH LEVEL.

Les buffers des actionneurs sont actualisés sur front descendant et l’action sur front montant.

1. Comparaison des design FPGA/ROS + microcontrôleur

|  |  |  |
| --- | --- | --- |
| Critère de comparaison | FPGA | ROS + µContrôleur |
| Performances | ++ | -- |
| Temps de développement | ++ | -- |
| Complexité du développement | ++ | -- |
| Programmation dynamique | OUI | NON |
| Disponibilité immédiate du matériel | NON | OUI |

Tableau 1 : Comparaison des designs FPGA / ROS + µContrôleur

Nous avons choisi le design microcontrôleur + ROS en raison des contraintes de temps et de disponibilité du matériel qui a déjà été acheté. La solution FPGA peut par contre être un axe d’amélioration possible.

1. Cahier de recettes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Fiche de test unitaires | | | | |
| Numéro de test | Composants | Description | Scénario | Résultat |
| 1 | Lidar + HIGH LEVEL | Test de l'acquisition du Lidar | Mettre un objet à une distance connue du Lidar et vérifier à l'aide du Debug de Microvision si la valeur est cohérente |  |
| 2 | Caméra + HIGH LEVEL | Test de l'acquisition de la Caméra | Afficher la sortie de la caméra à l'écran |  |
| 3 | Lidar + Caméra + HIGH LEVEL | Génération de la carte des obstacles | Générer la carte des obstacles à l'aide des valeurs des capteurs (caméra + lidar) |  |
| 4 | LOW LEVEL + HIGH LEVEL | Communication ETHERNET ou UART | Envoyer une trame de données et vérifier la cohérence de ce qu'on reçoit via affichage( LOW LEVEL vers HIGH LEVEL) ou via debug de Microvision (HIGH LEVEL vers LOW LEVEL) |  |
| 5 | LOW LEVEL + Moteurs | Commande des moteurs | Vérifier la bonne mise en marche et extinction des moteurs en parallèle |  |
| 6 | LOW LEVEL + Bras articulé | Commande du bras articulé | Vérifier la bon fonctionnement en parallèle des servo-moteurs du bras articulé |  |
| 7 | LOW LEVEL + HIGH LEVEL + Moteurs + Bras Articulé | Commande des moteurs et du bras à haut niveau | Vérifier que les moteurs et le bras sont bien commandés |  |
| 8 | LOW LEVEL + HIGH LEVEL + Moteurs + Bras Articulé | Test de saisie d'objet | Saisir un objet donné |  |
| 9 | LOW LEVEL + HIGH LEVEL + Moteurs + Bras Articulé | Recherche de chemin | Se déplacer vers un point sans heurter d'obstacle |  |
| Tableau 2 : Fiche de tests unitaires | | | | |
|  |  |  |  |  |
| Interactions avec l'environnement | | | | |
| Numéro | Capteurs utilisés | élements externes utilisés | Scénario | Résultat |
| 1 | Caméra + Lidar | QRCodes ou Aruco | Le robot doit saisir l'objet ayant un code |  |
| 2 | Caméra | Aruco ou QRCode | Le robot doit suivre l'utilisateur qui porte un code |  |
| 3 | Tous les capteurs | Aucun | Le robot doit trouver le plus court chemin pour arriver à une destination décidée à l'avance |  |
| 4 | Caméra | éventuellement codes Aruco | Le robot doit sortir d'une pièce tapisée de codes aruco tout seul |  |

Tableau 3 : Fiche de tests d’interaction avec l’environnement

1. Milestones

Conclusion