Technical Note

Author : M. ANDREU Joan
Date :2015/09/04
Revision : V.1.0.0

Index

1 Oz Simulator

- **1.1** Generalities
- **1.2** Main Interface
- **1.3** Socket routing

2 Architecture diagrams

3 Communication Protocol

- **3 .1** Generalities
  - **3.1.1** Accelero Payload
  - **3.1.2** Actuator Payload
  - **3.1.3** Gps Payload
  - **3.1.4** Gyro Payload
  - **3.1.5** Lidar Payload
  - **3.1.6** Magneto Payload
  - **3.1.7** Motors Payload
  - **3.1.8** Odo Payload
  - **3.1.9** Remote Payload

4 Oz

- **4 .1** Generalities

5 Frequently Asked Questions

- **5 .1** Simulator
- **5 .2** Naio Protocol
- **5 .3** Oz
- **5.4** Misc

1 Oz Simulator

1.1 Generalities

The simulator enables IA of the core to run without any hardware device connected, then you could design and test your controlling program without uploading code into the robot. It can also create the playground, the test fields.

It simulates the physics on a simple way, there is no friction on wheels, no gps errors, no odometrics problems, centimetric positions everything runs like a charm.

Don't forget that, on the target plateform, the reality strike back in the strongest way possible :

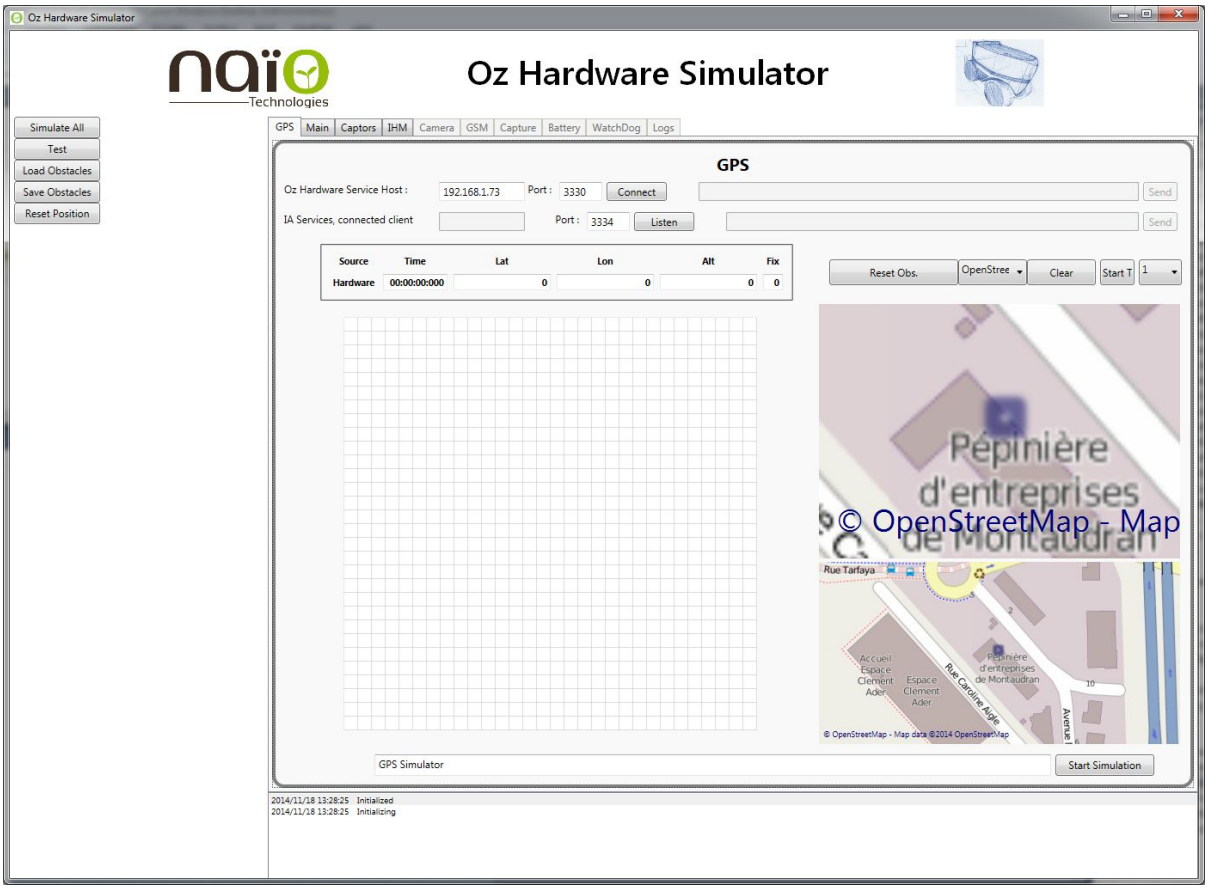
- Gps is sometimes under 2 meters of precision, when you have 3d position.
- Wheels can slide.
- Wheels can be blocked by mud, or rocks.
- IMU (gyro/accelero) are very noisy, the ground is not solid glass perfect, and needs to be calibrated by sofware (min/max values).
- Magnetometer (magnetic north bearing) needs calibration (min/max values), and electric motors or metallic infrastructures can alter the quality of signal.
- For the lidar herbs, lettuces or mud is detected in the same way, it's only an obstacle.
- Remote has a short communication range.
- When actuators plant the tool to deep in earth, the robot can be blocked by rocks, or can be slowed down a lot.
- The robot must be recharged back every 4 hours of work.
- Farmers are not robots themselves and line of vegetables can have holes or derive a bit.
- You should take in account personal safety and try to guess if something detected by the obstacle is something to be destroy or avoided.

You should take this limitation points in account into the design and the usage of any sensors you will integrate into your artificial intelligence program.

The team fixing or improving the simulator should gain time meanwhile improving their understanding of underlying technologies and concepts used in the main architecture.

Each feature of the robot is using a dedicated socket, in the interface you should be able to see the used port. You can change it for debugging purposes, but remember the target will used the pre-loaded values.

1.2 Main Interface

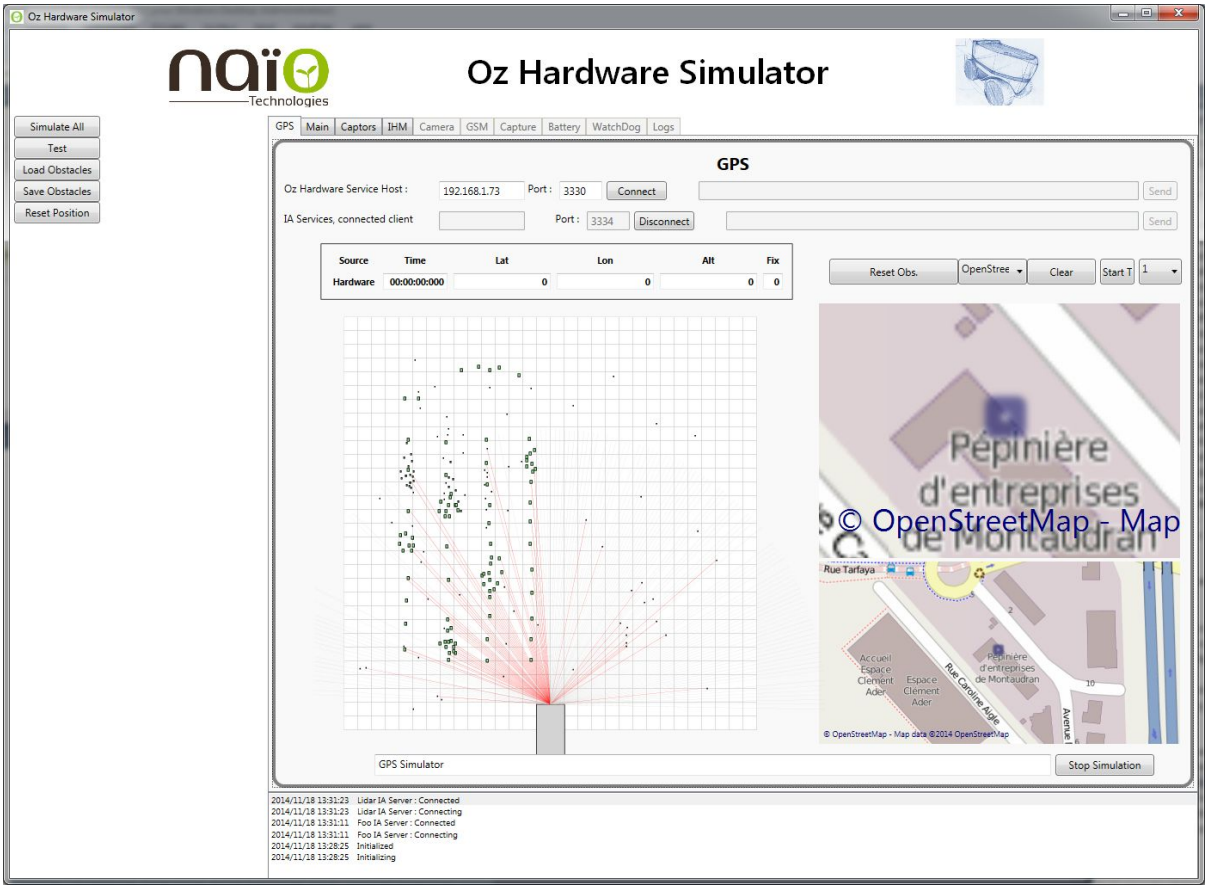


What to do :

-Click on Simulate On.

- Load Obstacle, and choose level1.obs map.

In the gps tab you should now see lidar rays, and be able to connect with sockets to the simulator. If you want to use the remote control, click on “start simulation” on main/Remote tab.



The list box, 1...128 concerns the size in cm of obstacles you will place in the grid map with a right mouse click. Start T enables Robot tracing, clear removes the points. Reset Obs, removes all obstacles in the map.

OzSimulator needs .Net Framework v4.5.

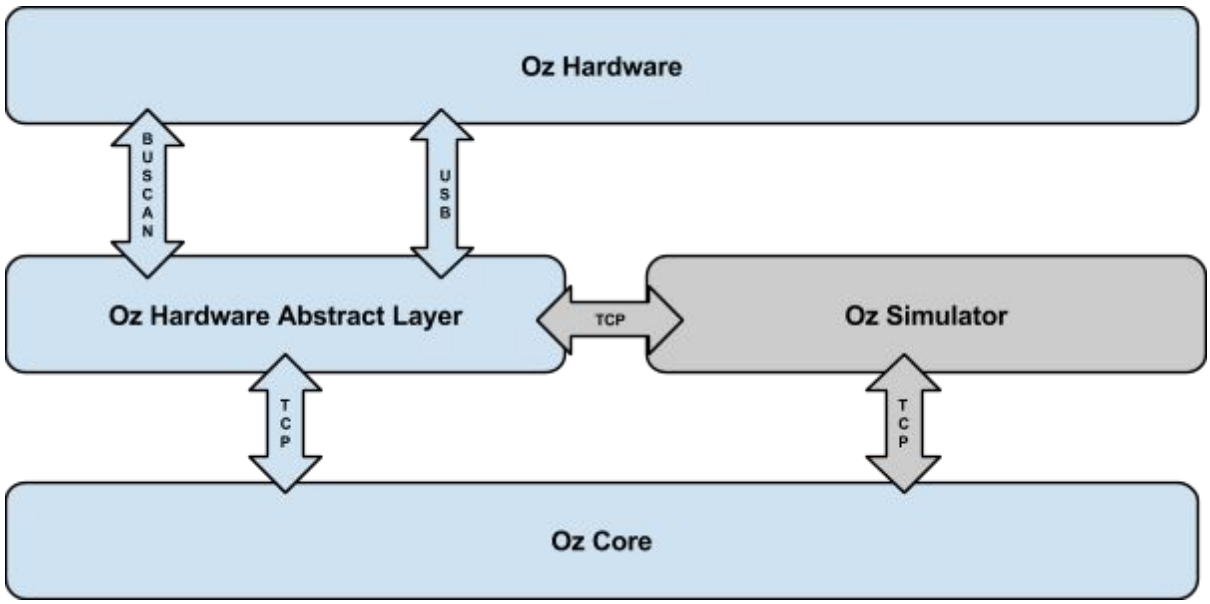
1.3 Socket routing

On tabs, you can activate socket routing, clicking on “connect”, that will connect to specified adress and port, and bridge data to the connected IA, your program.

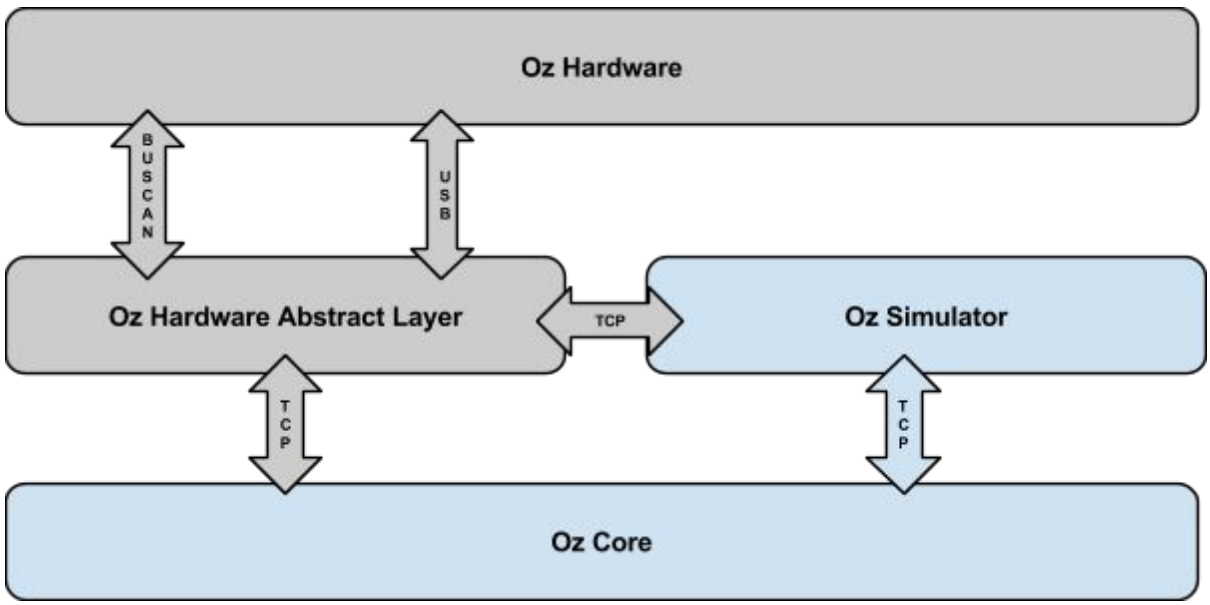
Use this only for testing, or bypassing simulated data with your own simulator.

2 Architecture diagrams

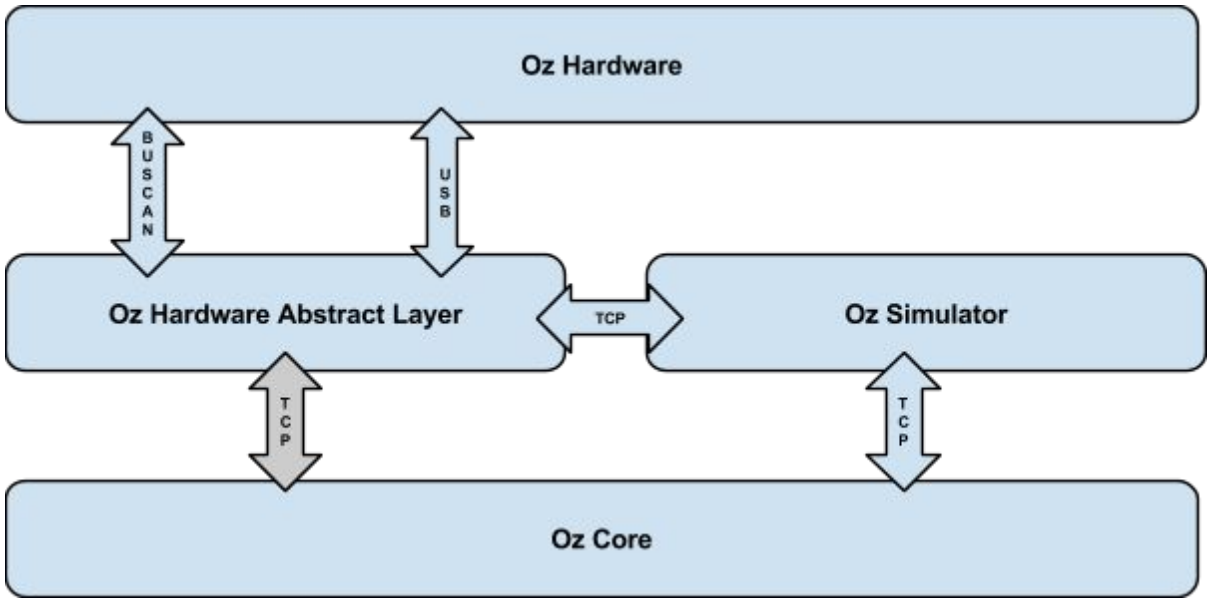
Robot mode :



Fully simulated mode :



Mixed Mode :


Oz Hardware
BUSCAN
USB
Oz Hardware Abstract Layer
TCP
Oz Simulator
TCP
TCP
Oz Core

3 Communication Protocol

3 .1 Generalities

Naïo Protocol Base Packet:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	n-3	n-2	n-1	n
N	A	I	O	0	1	19	0	0	0	5	x	x	x	x	x	crc	crc	crc	crc

- 6 bytes : protocol header the string 'NAIO01'
- 1 byte : packet id
- 4 bytes : packet content size, there we have five bytes data : 0 0 0 5
- X bytes : payload data
- 4 bytes : CRC32 : Present but not checked in this version, and not to check.

Caution : In payload data Strongiest bits first, lesser last.

3.1.1 Accelero Payload

id : 0x09

- 2 bytes : X, integer16 value, mG.
- 2 bytes : Y, integer16 value, mG.
- 2 bytes : Z, integer16 value, mG.

In simulator only Y axis is used, Z is fixed to 1000mG (1 G / 9.81m/s^{22}).

3.1.2 Actuator Payload

id : 0x0F

- 1 byte : Value, Byte value :
  // raise tool : 00000001
  // lower tool : 00000010
  // freeze tool : 00000011

Simulator listen to this socket, but value are not used.

3.1.3 Gps Payload

id : 0x04

- 8 bytes : time, Double value, gps time. ms since epoch.
- 8 bytes : Lat, Double value, gps latitude.
- 8 bytes : Lon, Double value, gps longitude.
- 8 bytes : Alt, Double value, gps altitude.
- 1 byte : Unit, byte value, gps unit used (meters there).
- 1 byte : NumberOfSat, byte value, number of satelite used by gps.
- 1 byte : Quality, byte value, 0 : no fix, 1 fixing, 2 : fix 2D, 3 : fix 3D, 6 super fix 3D.
- 8 bytes : GroundSpeed, double value, gps speed in km/h.

3.1.4 Gyro Payload

id : 0x0A

- 2 bytes : X, integer16 value, mDegree / s.
- 2 bytes : Y, integer16 value, mDegree / s.
- 2 bytes : Z, integer16 value, mDegree / s.

In simulator only Z.

The gain is 30.5 :

Simul :

Double degreeByMs = (degreeDiff * 1000.0 * secFactor) / 30.5;

Core :

```
gyr_raw_[0] = gyrData->x() * (30.5 / 1000);
gyr_raw_[1] = gyrData->y() * (30.5 / 1000);
gyr_raw_[2] = gyrData->z() * (30.5 / 1000);
```

3.1.5 Lidar Payload

id : 0x07

- 2 * 271 bytes : distance[271], uinteger16[271] value, lenght in mm of lidar ray with degree n[0-270];
- 1 * 271 bytes : Albedo, byte value[271], albedo of obstacle.

You could ignore albedo.

3.1.6 Magneto Payload

id : 0x0B

- 2 bytes : X, integer16 value, Degree.
- 2 bytes : Y, integer16 value, Degree.
- 2 bytes : Z, integer16 value, Degree.

In simulator :

```
Double rad = Math.PI * this.Bearing / 180.0;

    Double x = Math.Cos(rad) * 100;
    Double y = Math.Sin(rad) * 100;
    Double z = -23;
```

3.1.7 Motors Payload

id : 0x01

- 1 byte : LCM, sbyte value, left power [-127;127]
- 1 byte : RCM, sbyte value, right power [-127;127]

powers used are :

- 0 : stop
- 127 : full forward speed
- -127 : full backward speed

3.1.8 Odo Payload

id : 0x05

- 1 byte : FrontRight, byte value, 0 or 1
- 1 byte : RearRight, byte value, 0 or 1
- 1 byte : RearLeft, byte value, 0 or 1
- 1 byte : FrontLeft, byte value, 0 or 1

Each 6.465 cm run by a wheel, the tick status changes, 0 to 1, or 1 to 0, you cannot detect direction, you had to handle this with last motors command send, or noisy accelero/gyro.

3.1.9 Remote Payload

id : 0x08

- 1 byte : LeftAnalogX, byte value, left stick.
- 1 byte : LeftAnalogY, byte value, left stick.
- 1 byte : RightAnalogX, byte value, right stick.
- 1 byte : RightAnalogY, byte value, right stick.
- 1 byte : buttons, byte value, buttons.
- 1 byte : l1l3r1r3, byte value, other buttons.
- 1 byte : L2, byte value, L2 button.
- 1 byte : R2, byte value, R2 button.
- 1 byte : AccelX, byte value, accel of remote.
- 1 byte : AccelY, byte value, accel of remote.
- 1 byte : AccelZ, byte value, accel of remote.
- 1 byte : GyroX, byte value, gyro of remote.
- 1 byte : GyroY, byte value, gyro of remote.
- 1 byte : GyroZ, byte value, gyro of remote.

buttons bit positions (0/1 unpressed/pressed, Left Pad arrows)

Triangle	Circle	Cross	Square	Up	Left	Down	Right
7	6	5	4	3	2	1	0

other buttons bit positions (0/1 unpressed/pressed) :

Usb	Ps	Option	Share	R3	R1	L3	L1
7	6	5	4	3	2	1	0

Note : PS3 pad is used, and must be connected in pc running simulator.

Same Remote is used in the robot.

4 Oz

4.1 Generalities

Acceding the last phases, you should be able to run your program on the Robot, Oz.

Is brain is a multi-processor **Ubuntu** plateform, so you can develop with C, C++, Java, Python or every language supported by the OS.

Just remember the real Core is developped in C++11.

Investigating on internet should give you hints for evaluating the general behavior of the robot, several videos and article were published .

5 Frequently Asked Questions

5.1 Simulator

Question >

Bonjour,

Avant d'aller plus moins dans notre programmation du serveur/client du simulateur, nous avons oublié de vous demander si il y a t'il un langage de programmation imposé ou vivement recommandé ?

Cordialement l'équipe Ozira

Answer >

Non pas de langage imposés, le simulateur (ainsi que la plate-forme du robot réel) communique via sockets tcp, l'intelligence et la prise de décisions sont donc découplées fortement des couches basses.

Les couches basses du robot sont écrites en C++11 (ainsi que son IA, qui correspond à la partie qui vous est demandée par jeu) le Simulateur en C#4.5/WPF.

L'intérêt est de pouvoir exécuter le code sur le robot qui lui tourne sous Ubuntu 14, donc, plutôt exit ce quine pourrait pas tourner sous wine par exemple, mais la encore, on peut poser un pc windows sur le capot.

Si vous pensez à Java, Python, C, C++, ou même votre propre langage, avec de bonnes raison de le choisir, c'est que c'est probablement le bon choix !

- Si vous avez l'esprit aventurier, regardez du côté de "D", langage de haut niveau style java et C#, entièrement compilable sous un environnement C++, probablement une future et superbe évolution de ce langage.

- Si vous êtes frileux, restez en C++.
- Si vous voulez plutôt vous occupez des concepts que des erreurs de compil, allez vers C# (Mono par ex) ou Java (Attention en java, les bits de poids forts sont pas du même côté :p).

Question >

Serait il possible de connaitre les différentes conditions et objectifs que nous devons atteindre afin de respecter les consignes du projet ? De plus, existe d'autre informations que nous devons connaitre avant de débuter le projet ? Enfin, avons nous des documents à vous transmettre pendant ou au final de ce projet ?

En vous remerciant pour ces informations,

Answer >

Bonjour,

vous avez déjà l'exécutable, et le document technique à votre disposition, et je suis présent pour répondre à vos questions ou lever des doutes qui pourraient intervenir durant votre développement.

Le but, est bien sûr de faire déplacer le robot virtuel dans les rangées de la map level1, et des autres cartes à venir, de façon automatique. Commencez par le faire se déplacer grâce à une manette, pour mieux intégrer les différents aspects de la communication via le protocole mis en place.

Les exécutables et documents mis à jours vous seront transmis régulièrement selon l'optimisation ou la correction de problèmes, ou l'amélioration de la FAQ du document avec vos questions, y compris cette dernière.

Lors des phases qualificatives, les codes seront exécutés sur le simulateur, mais aussi sur les robots, dans un environnement proche de celui simulé, prévoyez des gains sur les puissances moteurs, modifiables rapidement, les environnements simulés et réels étant probablement différents :).

Je vous invite à m'envoyer régulièrement un mini compte rendu de projet, et quelques copies d'écrans, je vous dirais ce que j'en pense, et pourrais vous donner quelques astuces.

Question >

Dois-je lancer tous les appareils simulés avant l'envoi des trames ? (ex : GPS, motors, Odometrics, ... sur listen)

Answer >

Pour voir bouger le robot, il faut effectivement activer la simulation sur l'onglet gps, pour avoir le retour odométrique, il faut activer Odo etc... seul motors n'est pas à simuler, puisque c'est vous qui envoyez les commandes.

Pour voir le robot avancer sur une distance suffisante il faut envoyer la commande moteur très souvent, toutes les 20ms par exemple.

Answer >

Format du protocole différent / erroné par rapport au document technique.

S'agit il d'une erreur d'interprétation de notre part au bien une erreur lié au protocole du robot ? (En sachant que nous arrivons à envoyé des commandes au moteurs avec le format définie dans le document)

De plus, est il normal d'obtenir une valeur dans le CRC ?

Par ailleurs, nous nous demandons comment fonctionne "le capteur" Remote du robots Oz. En effet, malgré qu'on lui transmette des informations, aucune réaction n'est à observé. De ce fait, nous nous demandons aussi à quoi sert cette fonction (S'agit il bien d'une fonction permettant de controler le robot) ?

5 .2 Naio Protocol

Question >

Lorsque que j'envoi une trame pour commander les moteurs, rien ne se passe, je ne reçois pas de trame d'erreur et le robot simulé ne bouge pas.

Voici la trame envoyée pour les moteurs :

```
byte[] ba = new byte[20];
ba[0] = 0x4E; // N
ba[1] = 0x41; // A
ba[2] = 0x49; // I
ba[3] = 0x4F; // O
ba[4] = 0x00; // 0
ba[5] = 0x01; // 1
ba[6] = 0x01; // ID = 01
ba[7] = 0x00; // SIZE
ba[8] = 0x00; //SIZE
ba[9] = 0x00; //SIZE
ba[10] = 0x05; //SIZE
ba[11] = 0x7F; // byte : LCM, sbyte value, left power [-127?127]
ba[12] = 0X7F; // byte : RCM, sbyte value, right power [-127?127]
ba[13] = 0x00; //
ba[14] = 0x00; //
ba[15] = 0x00; //
```

```
ba[16] = 0x00; // CRC
ba[17] = 0x00; // CRC
ba[18] = 0x00; // CRC
ba[19] = 0x00; // CRC
```

Ma trame est-elle correcte ?

Answer >

En ce qui concerne la trame, la size est sur 4 bytes, ok, mais il n'y a que 2 bytes de payload, (left et right), le buffer aux index 13, 14 et 15 semblent donc superflus, le crc termine bien la trame avec 4 bytes.

En bonus un exemple de trame moteur valide :

```
[0x00000000]  0x4e  byte // header
[0x00000001]  0x41  byte // header
[0x00000002]  0x49  byte // header
[0x00000003]  0x4f  byte // header
[0x00000004]  0x30  byte // header
[0x00000005]  0x31  byte // header
[0x00000006]  0x01  byte // packet id : motors
[0x00000007]  0x00  byte // size
[0x00000008]  0x00  byte // size
[0x00000009]  0x00  byte // size
[0x0000000a]  0x02  byte // size
[0x0000000b]  0x7f  byte // LCM
[0x0000000c]  0x7f  byte // RCM
[0x0000000d]  0x00  byte // fake crc
[0x0000000e]  0x00  byte // fake crc
[0x0000000f]  0x00  byte // fake crc
[0x00000010]  0x00  byte // fake crc
```

Question >

Comment fonctionne le lidar ?

Answer >

- Les données lidar sont codées sur 2 octets.
- Le lidar remonte 271 points / degrés.
- Le point au centre est donc à 271 / 2.
- La visibilité du lidar est de 180° frontal, donc, [0 - 45] inutile, [45 , 225] lidar (centre 135), [225, 271] inutile.
- La suite est l'albédo, qui n'est pas utilisé.
- Si lidar ne voit rien il peut remonter sa valeur max soit 4000mm.

5.3 Oz

Be careful in the real robot : timeout are not the same !

You should send motors data each 50 ms to avoid motor timeouts. The simulator can handle this timing, so it's a good idea to start with.

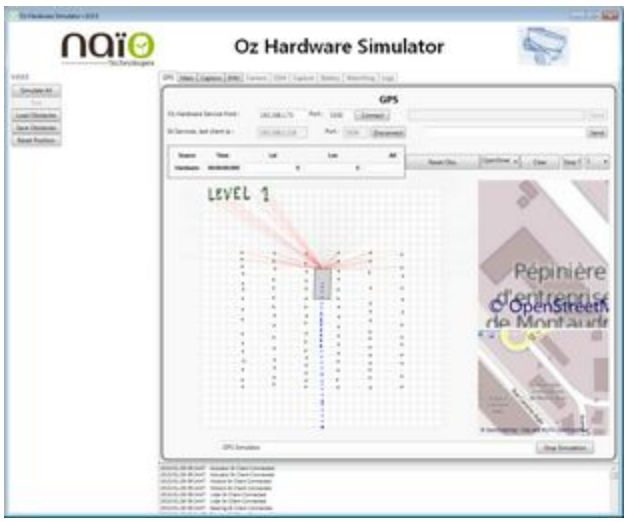
Don't try to control the robot with "time only", this will be ok with the simulator in its actual version, but we will integrate ground slippering, blocked wheels, stones that make the robot jumps off 10cm easyly and so on !

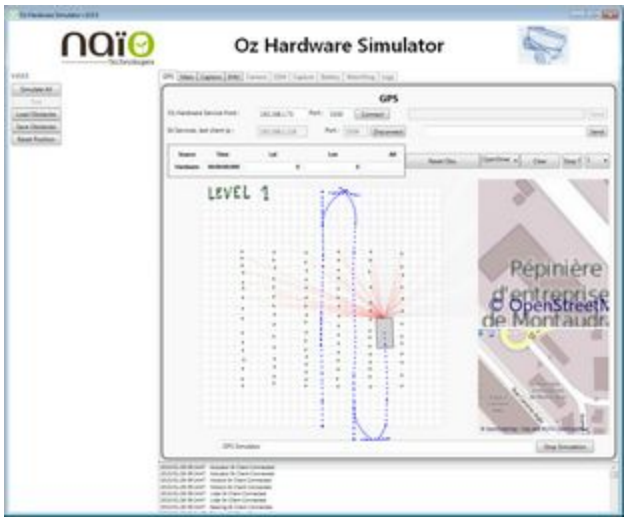
Don't try to use magneto meter, the motors themselves are enough to make a lot of interferences, and it needs to be calibrated.

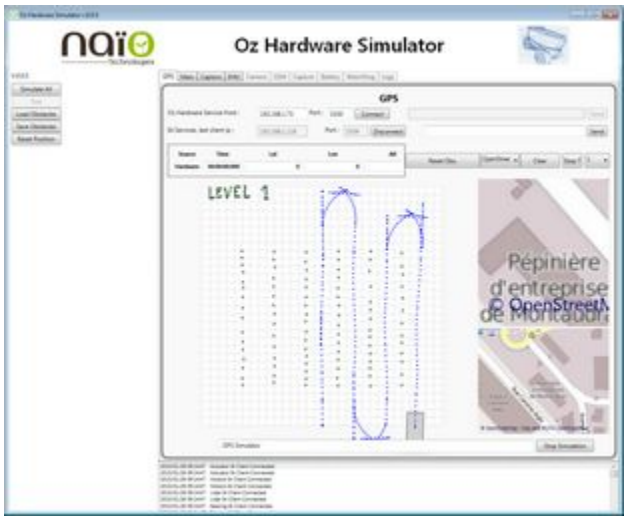
Go with IMU for heading, go for lidar for line extractions ! This are precious hints !

5.4 Misc

Some picture of a journey between vegetables :

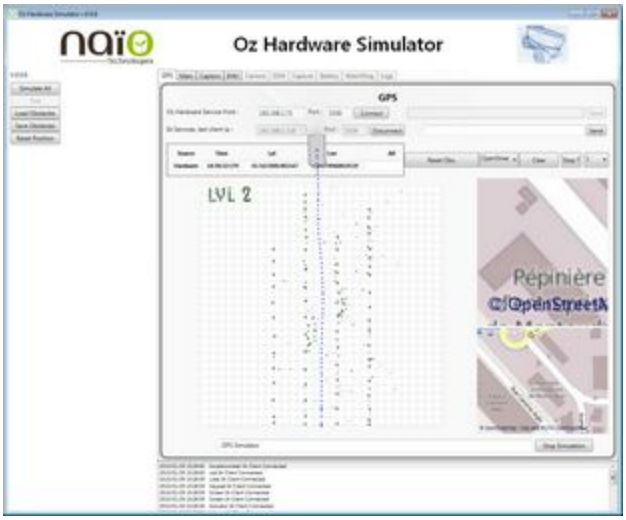


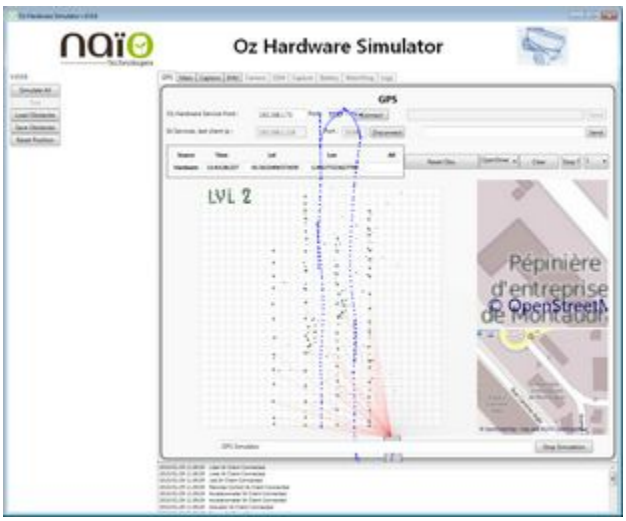


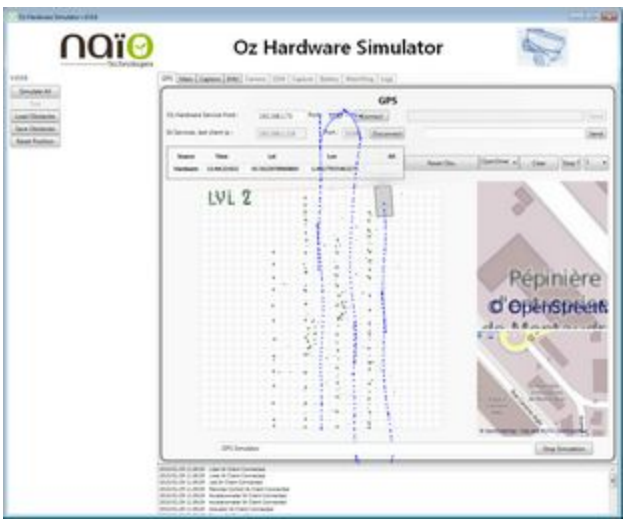


Two lines,two passages, 75cm large, 10cm of edge, turning left !

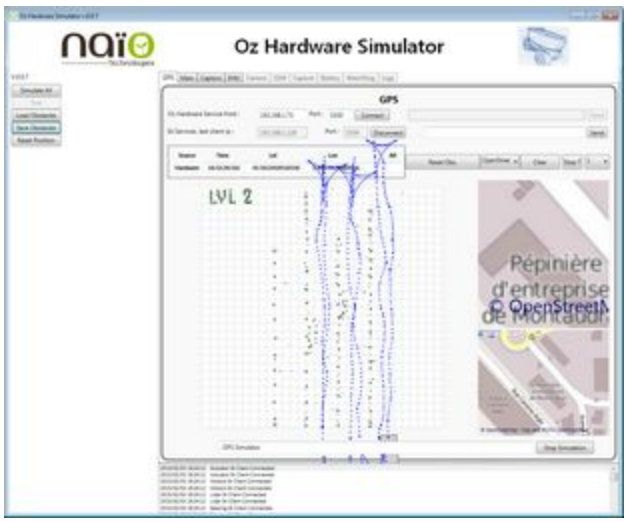
Smooth and effective run between vegetables :



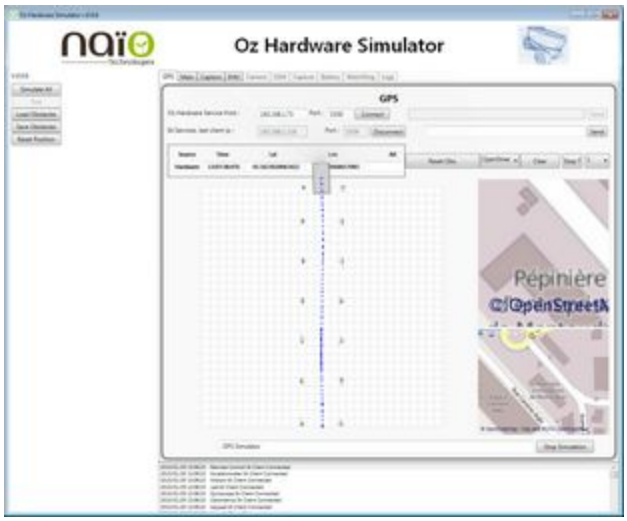




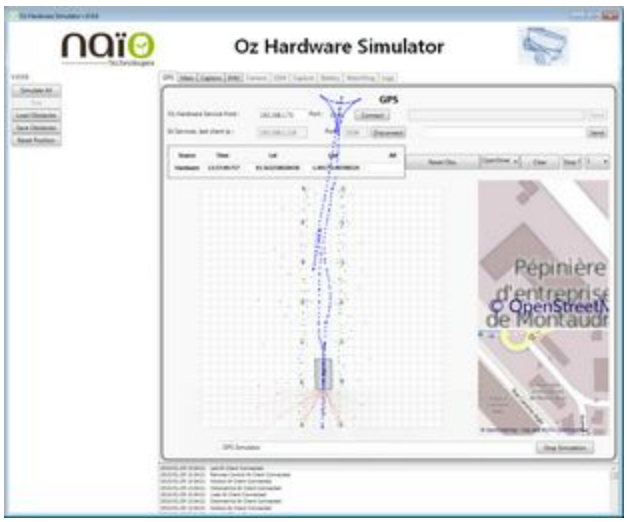
2 Runs by line :


Oz Hardware Simulator
GPS
LVL 2
Pépinière
d'entreprise
© OpenStreetMap

A trek in a world of grappes :



A travel in a world of noisy grappes, two passages :



Oz should detect when the line is tricky :