Technical Note

Author : M. ANDREU Joan
Date :2016/11/28
Revision : V.1.5.1

Contents

1. Simulatoz

1.1 General information

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

The simulated robot does not behave like the real one in every way : the friction of the wheels is not exact, the noise on the sensors does not exactly match the real noise and the playground is not like a real field with mud and dirt.

Do not forget that, on the target platform, the reality strikes back in the strongest way possible :

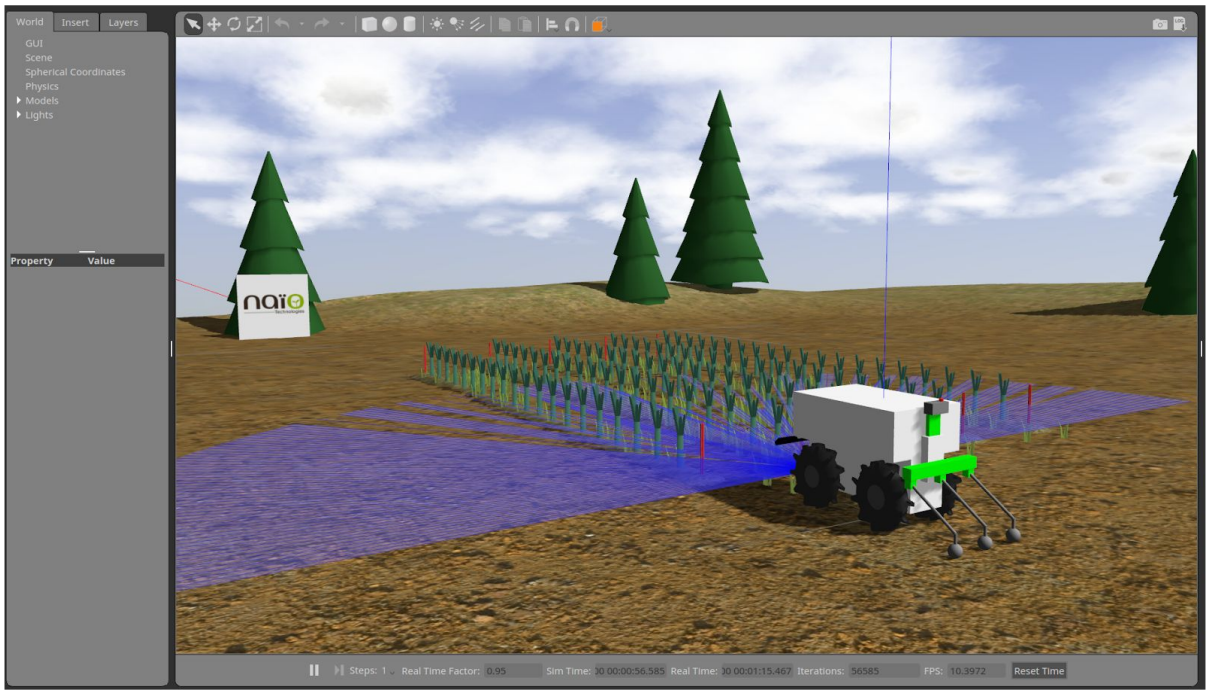
- Gps is sometimes under 2 meters of precision, when you have 3d position.
- Wheels can slide.
- Wheels can be blocked by mud, or rocks.
- IMU (gyro/accelero) are very noisy, the ground is not solid glass perfect, and needs to be calibrated by sofware (min/max values).
- For the lidar, grass, lettuces or mud are detected in the same way, it's only an obstacle.
- Remote has a short communication range.
- When actuators stick the tool too deep in the earth, the robot can be blocked by rocks, or slowed down a lot.
- The robot's batteries must be charged every 4 hours of work.
- Farmers are not robots themselves and lines of vegetables are not necessarily straight and can have holes.
- You should take into account personal safety and try to guess if an obstacle detected is something to destroy or avoid.

You should take these limitation points into account for your design and when you use a sensor into your artificial intelligence program.

1.2 Using the Simulator

The simulator has been developed using the ROS and Gazebo technologies. Thus it requires the proper version of those softwares to work properly. This is why we decided to use a Docker container to release it. The container uses a Ubuntu 16.04 kernel, and has ROS, Gazebo and the simulator installed. In order to use it you need to have a Linux operating system and to follow the instructions in the Simulatoz Linux file.

Once you have installed the container, you can launch the simulator with the roslaunch command (see in the Simulatoz Linux file). Two different modes are available : with or without graphics. If your computer is not very powerful we strongly advise you to use the no graphics mode.



Graphical interface of the Simulator

Once the simulator is launched, it will communicate as described in the next pages. An example of how to communicate with the simulator is given by the test viewer ApiClient (see in the Simulatoz Linux file). This program gets and prints all the sensors data from the simulator. It also acquires data from the keyboard and sends motor and actuators orders that will be used by the simulator to move the simulated robot.

You will be able to create different test fields with various parameters (see in the Simulatoz Linux file) :

- Texture of the ground
- Leek or cabbage
- Mounds under the rows of vegetables
- Number, length and width of the rows
- Rocks in the ground to disrupt the progression of the robot
- Grass to add noise to the lidar data

The simulator communicates via the NAIO01 Protocol codecs, you can download a c++ implementation on :

https://github.com/NaioTechnologies/ApiCodec

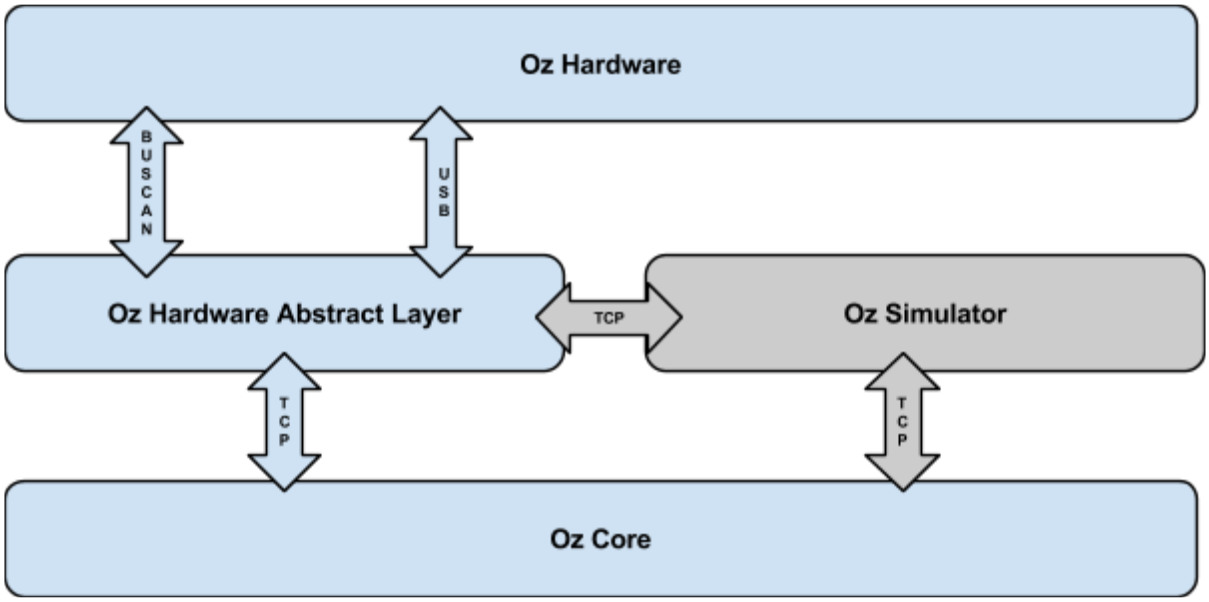
You can find a c++ sample program, using the last codec and the simulator on :

https://github.com/NaioTechnologies/ApiClient

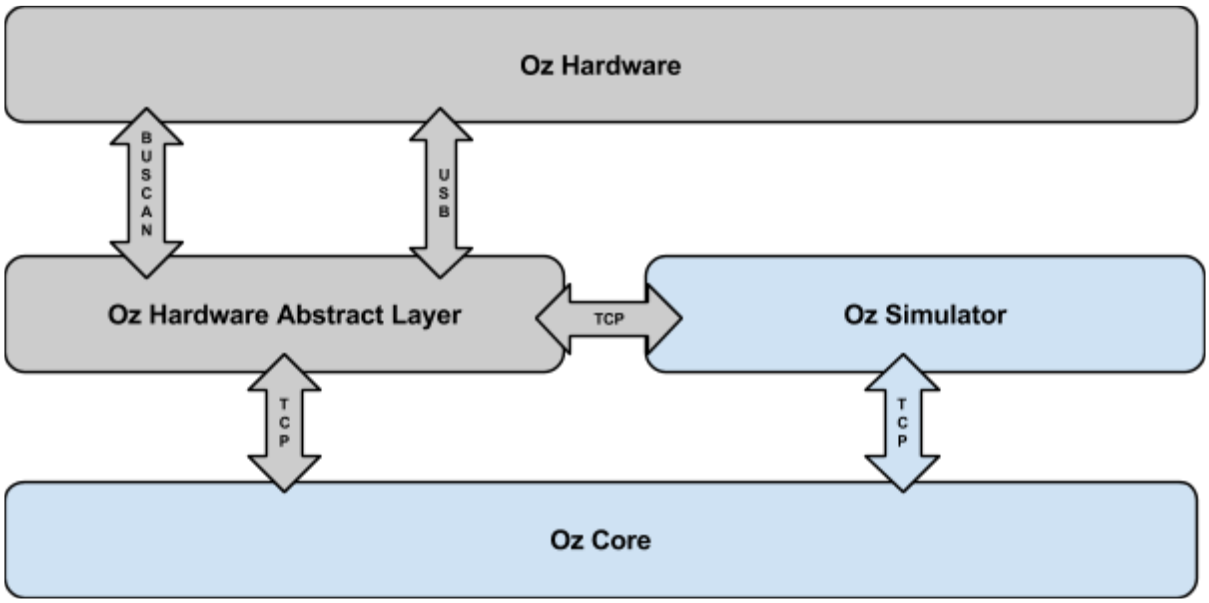
Please read the README.md file bound in the projetc.

2. Architecture diagrams

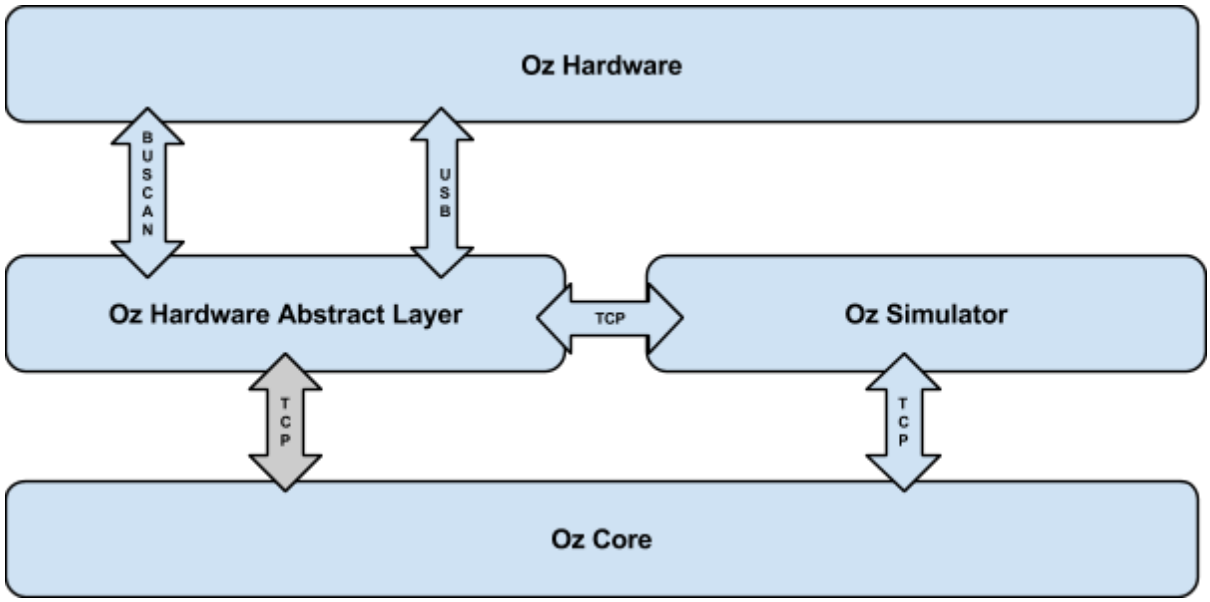
Robot mode :



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 General information

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, the most significant bit is first and the least significant bit is last.*

You can download all packet codecs in c++ on https://github.com/NaioTechnologies/ApiCodec.git.

3 .2 Payload description

3.2.1 Accelero Payload

id : 0x09

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

3.2.2 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.

3.2.3 Gyro Payload

id : 0x04

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

The gain is 30.5 :

Simulator : 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);

3.2.4 Lidar Payload

id : 0x07

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

You should/may/must ignore albedo. (The simulator returns 0 for the albedo).

3.2.5 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

3.2.6 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 the direction of the wheel, you have to use the last motor command or the accelero/gyro data to know if it is going forward or backward.

3.2.7 Raw Stereo Camera Payload

id : 0xA2

- 1 byte : **0x01 raw Images**, 0x02 unrectified colorized images, 0x03 rectified colorized images, 0x04 raw Images zlib compressed, 0x05 unrectified colorized images zlib compressed, 0x06 rectified colorized images zlib compressed
- 4 byte : buffer Size
- X byte : buffer data :

Only Raw images are supported actually by the simulator. They are in Bayer format in 752 * 480 * 1 (Bayer GBRG). (https://en.wikipedia.org/wiki/Bayer_filter)

The size of each **raw image** is 752 * 480 (width * height) in grey scale (Bayer GBRG), so the buffer for one image has 752 * 480 * 1 bytes. The total buffer size is : 752 * 480 * 1 * 2 = 721920 bytes.

- Images are send in a buffer starting from pixel (0;0) top left to (751;479) bottom right.
- For instance : the byte at the position 752 (752*1) in the buffer is the pixel of the left image (0;1) at column 0 and line 1.
- The left image is sent first in the buffer, so the right image starts at index 360960.

3.2.8 Actuator Payload

id : 0xA9

- 1 byte : uint8 position in % of the back tool actuator (100% : tool is down)

4. Oz

4 .1 General information

Accessing to the last stages, you should be able to run your program on the Robot, Oz.

Its brain is a multi-processor **Ubuntu** platform, so you can develop with C, C++, Java, Python or every language supported by the OS. Just remember the real Core is developed in C++11.

Investigating on the internet should give you hints to evaluate the general behavior of the robot, since several videos and articles where you can see how Oz moves were published .

4 .2 Advice

Be careful : on the real robot, timeouts are not the same !

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

Don't try to control the robot with 'time only', this might work with the simulator, but we will add ground sliding, blocked wheels, stones and other disrupting features that will change the robot's behaviour !

Use with IMU for heading and lidar for line extraction ! These are precious hints !

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++.

L'intérêt est de pouvoir exécuter le code sur le robot qui lui tourne sous Ubuntu 14, donc, plutôt exit ce qui ne 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 connaître 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 au clavier, 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.

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]    0x000x00,          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 qui correspondent aux 271 degrés (sens trigonométrique).
- 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 soit 0 soit sa valeur max soit 4000mm.