Docker and gzweb

This file contains 4 sections.

Section 1. `Virtual Field Robot Environment we made` contains documentation on how the launch file for the virtual field robot environment works, an how the world files and the robot files are made. This section also contains documentation on how ROS and gazebo handle some files.

Section 2. ` Example models in gzweb with docker` Contains my journey to implement an example robot in the docker file, and problems and solutions I encountered.

Section 3. `Custom models in gzweb and docker` Is about how to implement custom robot models in the a\_container.

Section 4 `B container` Is about how to implement the B container.

# Virtual Field Robot Environment we made

### Launch file

Currently, in the simulation environment <https://github.com/FieldRobotEvent/Virtual_Field_Robot_Event> models for the maize plants and for the `jackal` robot and its sensors are available. The field and the robot are spawned in gazebo using the launch script: `[Virtual\_Field\_Robot\_Event](https://github.com/FieldRobotEvent/Virtual_Field_Robot_Event)/[virtual\_maize\_field](https://github.com/FieldRobotEvent/Virtual_Field_Robot_Event/tree/master/virtual_maize_field)/[launch](https://github.com/FieldRobotEvent/Virtual_Field_Robot_Event/tree/master/virtual_maize_field/launch)/simulation.launch`. A launch is just an XML file for ROS to know which nodes(scripts or binaries) to run with wich parameters. You can launch this script by running: `roslaunch virtual\_maize\_field simulation.launch`.

`virtual\_maize\_field` is the name of the package. After catkin\_make the [Virtual\_Field\_Robot\_Event](https://github.com/FieldRobotEvent/Virtual_Field_Robot_Event) and sourcing the setup.bash (as described in the README.md), ROS knows the location of the package. `simulation.launch` is then just the launch file. The launch file broken down:

The first part consists of a series of argument definitions. These are the default parameters and can be overruled by using the terminal. E.g. by running `roslaunch virtual\_maize\_field simulation.launch **world\_name:=simple\_row\_level\_1.world**` the parameter `world\_name` is overwritten by this new value `simple\_row\_level\_1.world`.

<arg name="use\_sim\_time" default="true" />

<arg name="gui" default="true" />

<arg name="headless" default="false" />

<arg name="world\_path" default="$(find virtual\_maize\_field)/worlds/"/>

<arg name="world\_name" default="generated.world"/>

<!-- Short-term hack to support the original front\_laser:=true argument for spawning

the simulator. This will be removed in favour of using the config:=x arg instead. -->

<arg name="front\_laser" default="true" />

<arg name="default\_config" value="front\_laser" if="$(arg front\_laser)" />

<arg name="default\_config" value="base" unless="$(arg front\_laser)" />

<!-- end of hack -->

<!-- Configuration of Jackal which you would like to simulate.

See jackal\_description for details. -->

<arg name="config" default="$(arg default\_config)" />

<!-- Optionally enable teleop for the simulation -->

<arg name="joystick" default="false" />

`

The second part uses the `empty\_world.launch` from the `gazebo\_ros` package. This starts gazebo and creates an empty world. The arg world\_name tells gazebo which world to load. More about the world files later.

<!-- Launch Gazebo with the specified world -->

<include file="$(find gazebo\_ros)/launch/empty\_world.launch">

<arg name="debug" value="0" />

<arg name="gui" value="$(arg gui)" />

<arg name="use\_sim\_time" value="$(arg use\_sim\_time)" />

<arg name="headless" value="$(arg headless)" />

<arg name="world\_name" value="$(arg world\_path)$(arg world\_name)" />

</include>

The third part, `spawn\_jackal.launch` from the `jackal\_gazebo\_fre` is used to spawn the jackal robot in the world. This package is located in `[Virtual\_Field\_Robot\_Event](https://github.com/FieldRobotEvent/Virtual_Field_Robot_Event)/jackal\_fre/jackal\_gazebo\_fre`. More about this package later.

<!-- Spawn Jackal -->

<include file="$(find jackal\_gazebo\_fre)/launch/spawn\_jackal.launch">

<arg name="x" value="0" />

<arg name="y" value="0" />

<arg name="z" value="0.1" />

<arg name="yaw" value="0" />

<arg name="config" value="$(arg config)" />

<arg name="joystick" value="$(arg joystick)" />

</include>

### World file

In this case, the world files are beforehand generated by another script. The world files available for the participants are located in `[Virtual\_Field\_Robot\_Event](https://github.com/FieldRobotEvent/Virtual_Field_Robot_Event)/[virtual\_maize\_field](https://github.com/FieldRobotEvent/Virtual_Field_Robot_Event/tree/master/virtual_maize_field)/worlds/`. These world files are just a very large XML file containing information about where to place which models in the world. `\*level\_0.world` contains a very simple world with cylinders (an object model available in gazebo by default). `\*level\_[1-6].world` contain a more complex world with maize plant made form 3D meshes. The maize models are located in `[Virtual\_Field\_Robot\_Event](https://github.com/FieldRobotEvent/Virtual_Field_Robot_Event)/[virtual\_maize\_field](https://github.com/FieldRobotEvent/Virtual_Field_Robot_Event/tree/master/virtual_maize_field)/models/`. The `\*.world` file refers to the meshes of these models in e.g. `<uri>model://maize\_02/meshes/maize\_02.dae</uri>`

However, by default, gazebo cannot find this model. We first must tell gazebo where these models are located. This is done (as described in the README.md) by extending the $GAZEBO\_MODEL\_PATH variable with the model directory (`export GAZEBO\_MODEL\_PATH=<YOUR WS ROOT DIR>/src/Virtual\_Field\_Robot\_Event/virtual\_maize\_field/models/:$GAZEBO\_MODEL\_PATH`

### Jackal robot and sensor model

In the folder `[Virtual\_Field\_Robot\_Event](https://github.com/FieldRobotEvent/Virtual_Field_Robot_Event)/jackal\_fre/` multiple packages for the simulation of the jackal robot are located. I copied those packages from the original jackal repository and edited them a bit to better suit our needs.

As discussed earlier, `virtual\_maize\_field simulation.launch` includes `[Virtual\_Field\_Robot\_Event](https://github.com/FieldRobotEvent/Virtual_Field_Robot_Event)/[jackal\_fre](https://github.com/FieldRobotEvent/Virtual_Field_Robot_Event/tree/master/jackal_fre)/[jackal\_gazebo\_fre](https://github.com/FieldRobotEvent/Virtual_Field_Robot_Event/tree/master/jackal_fre/jackal_gazebo_fre)/[launch](https://github.com/FieldRobotEvent/Virtual_Field_Robot_Event/tree/master/jackal_fre/jackal_gazebo_fre/launch)/spawn\_jackal.launch`. This launch file does very little. It launched scripts needed to control the simulated wheel motors, and it launches the visual part of the robot by including the script `[Virtual\_Field\_Robot\_Event](https://github.com/FieldRobotEvent/Virtual_Field_Robot_Event)/[jackal\_fre](https://github.com/FieldRobotEvent/Virtual_Field_Robot_Event/tree/master/jackal_fre)/[jackal\_description\_fre](https://github.com/FieldRobotEvent/Virtual_Field_Robot_Event/tree/master/jackal_fre/jackal_description_fre)/[launch](https://github.com/FieldRobotEvent/Virtual_Field_Robot_Event/tree/master/jackal_fre/jackal_description_fre/launch)/description.launch`. This script launches the `$(find jackal\_description\_fre)/urdf/jackal.urdf.xacro`.

Now this is another story. Gazebo uses \*.SDF files (Simulation Description Format), for describing robots and other objects in the environment. ROS uses \*.URDF files (Universal Robot Description Format). Roslaunch can spawn URDF files in gazebo. However, things cannot be too simple. URDF files are sometimes very long and repetitive, so they invented xacro. This is a macros style language, that can handle parameters and scripts to reduce repetitive stuff. \*.xacro files can be parsed into a URDF file, which can then be launched into gazebo.

This file refers to the 3D meshes by e.g. `<mesh filename="package://jackal\_description\_fre/meshes/jackal-wheel.stl"/>`. This is a ROS way of referring to a the mesh file, and this spawning is handled by ROS, and therefore the location of these meshes does **NOT** have to be added to the gazebo model path to be spawned in gazebo.

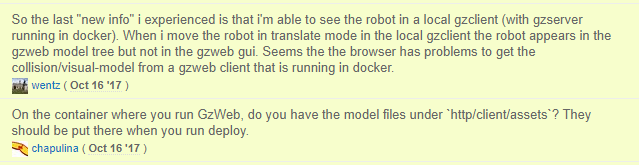
# Example models in gzweb with docker

This is just about the current stuff available in the docker file.

* Gzweb is working, and I can place the simple shapes, a sphere, cube and cylinder in the world using the gzweb web client.
* When starting gazbeo\_ros by placing `CMD roslaunch gazebo\_ros empty\_world.launch && npm start` in the docker file I can also use gzweb to interfere with gazebo\_ros.
* I tried to add a standard robot form an apt repository. I added the turtle bot. This is maybe the best documented robot and is used in various tutorials. I added the installation stuff for the turtlebot (I just install all turtlebot stuff, it’s maybe a bit much, but just for testing purposes) and ended the docker file with launching the turtlebot. The turtlebot is spawned in gazebo and using the gazbeo web client I can find the robot model in the overview(see screenshot). When showing collision, I can also see the hitbox of the robot(consisting of simple shapes) however, I cannot see the visual mesh (see screenshot).

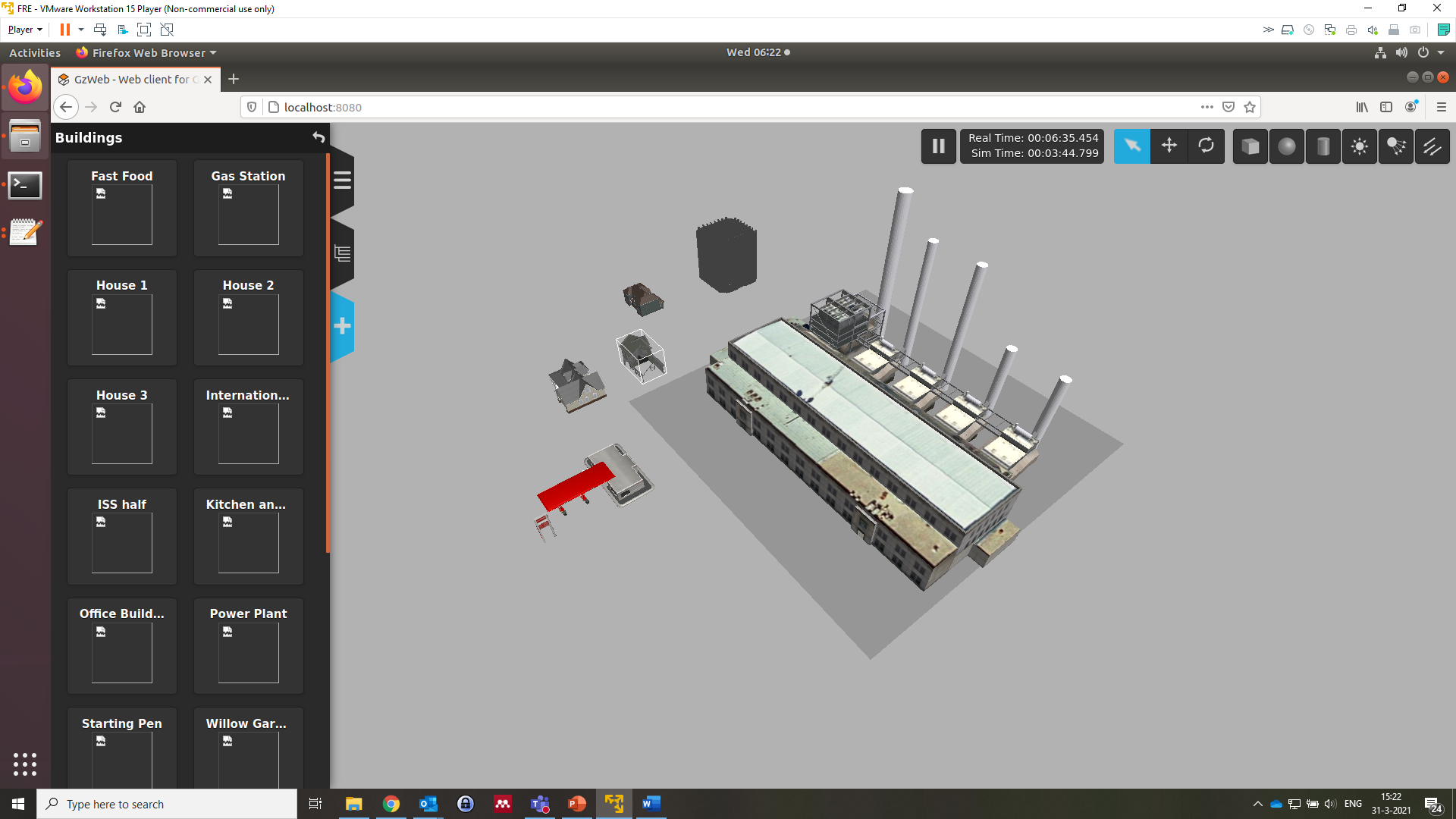
|  |  |
| --- | --- |
|  |  |
| Turtlebot model tree, but no visual mesh of the turtlebot. | Collision model of the turtlebot correctly loaded in gzweb. |

I have been looking into this problem, and found something: <https://answers.gazebosim.org//question/17367/simulated-robot-model-randomly-appears-in-gzweb-gui/>. According to some of the reactions, all the meshes must be in the `http/client/assets` directory.



In the installation instructions of gzweb, <http://gazebosim.org/#install-collapse-1> a deploy script is mentioned (`npm run deploy --- -m`). This script should place al the models in the right location. However, we use this script in our dockerfile and still we do not get the visual model of the turtlebot in gzweb.

* I changed the useage of the deploy script to include the gazebo model library, and now I can also ad models form the model library, including a powerplant and the iss. These models also use a texture, so the deploy script is probably doing what is should do. See the screenshot below.



According t the installation instructions the deploy script places all the models in the gazebo model path in the right directory.

|  |
| --- |
|  |

So probably we need to add the ROS models files to the $GAZEBO\_MODEL\_PATH before the deploy script. I thought this was done by sourcing the setup.sh however, this is probably not the case. Anyway, after writing this document I have the idea that we can search in a very specific direction.

We are not using the turtlebot during the event, but I think we should know how to make gazebo deal with the visual meshes of these models before we continue with our own models? Or we can try to add the maize field, which I think is very straight forward to implement, and see if that works?

# Custom models in gzweb and docker

We add our git (<https://github.com/FieldRobotEvent/Virtual_Field_Robot_Event>) to the docker file. We just provide the sensors that are in there. If people want to add their own sensor they have to make a pull request to our git repo. We will check of no illegal methods are used to create a sensor and accept the pull request. The sensor is then automatically available for all other teams as well.

The a\_container should then just accept an external urdf, or xacro file to spawn a new robot. This external file is provided by the competing team.

# B container

I think we are clear on the b\_container. We should provide a file that connects to the a container and lets the jackal drive forward.

The Jackal motors are controlled by the topic `/jackal\_velocity\_controller/cmd\_vel` with the message type `geometry\_msgs/Twist`. When running the following command in the terminal (btw I messed up the enters), the robot starts to drive 1m/second forward:

`

rostopic pub -r 10 /jackal\_velocity\_controller/cmd\_vel geometry\_msgs/Twist "linear:

x: 1.0

y: 0.0

z: 0.0

angular:

x: 0.0

y: 0.0

z: 0.0"

`