

# Guide for creating Gazebo worlds

This is a quick guide for creating Gazebo simulations with a georeferenced floor model and additional objects made up of cylinders, in this examples to build electrical towers and transmission lines as pylons and wires. Also, a brief explanation about how to animate models with Gazebo and include any 3D model from Blender is included. Some examples can be seen in the repositories “[grvc-ual](#)” or the ones in “[grvc-utils/grvc\\_gazebo\\_worlds](#)”.

## Outline

1. Simulation world with georeferenced world:.....	2
- <i>A model of the floor</i> :.....	2
- <i>A launch file</i> :.....	6
- <i>A world file</i> :.....	7
2. How to add pylons and wires with cylinders:.....	10
3. Animated models in Gazebo:.....	16
4. Model from photogrammetry:.....	19
5. Model editing with Blender oriented to Gazebo:.....	20

## 1. Simulation world with georeferenced world:

Needed files: (following the *grvc-utils* folder structure)

- **Model of the floor:** satellite image of the ground to have a visual reference of the scenario, placed in [/grvc-utils/grvc\\_gazebo\\_worlds/models/burquillos\\_power\\_lines/materials/textures/burquillos\\_power\\_lines.jpg](/grvc-utils/grvc_gazebo_worlds/models/burquillos_power_lines/materials/textures/burquillos_power_lines.jpg)
- **Model config file:** XML language file that describes the model and its author, placed in [/grvc-utils/grvc\\_gazebo\\_worlds/models/burquillos\\_power\\_lines/model.config](/grvc-utils/grvc_gazebo_worlds/models/burquillos_power_lines/model.config)
- **Model sdf file:** XML language file where the terrain model is defined, placed in [/grvc-utils/grvc\\_gazebo\\_worlds/models/burquillos\\_power\\_lines/model.sdf](/grvc-utils/grvc_gazebo_worlds/models/burquillos_power_lines/model.sdf)
- **Material file:** where the material properties are defined and the image of the floor specified, placed in [/grvc-utils/grvc\\_gazebo\\_worlds/models/burquillos\\_power\\_lines/materials/scripts/burquillos\\_power\\_lines.material](/grvc-utils/grvc_gazebo_worlds/models/burquillos_power_lines/materials/scripts/burquillos_power_lines.material)
- **Launch file:** XML language file where ROS nodes/scripts are indicated, needed for roslaunch command to set up and start the simulation, placed in [/grvc-utils/grvc\\_gazebo\\_worlds/launch/simulator\\_burquillos\\_power\\_lines.launch](/grvc-utils/grvc_gazebo_worlds/launch/simulator_burquillos_power_lines.launch)
- **World file:** XML language file where Gazebo simulation parameters and models are specified, placed in [/grvc-utils/grvc\\_gazebo\\_worlds/worlds/burquillos\\_power\\_lines.world](/grvc-utils/grvc_gazebo_worlds/worlds/burquillos_power_lines.world)

### - A model of the floor:

located in the “models” folder. This is usually a huge but thin 3D box, which can almost be treated like a 2D rectangle (0’1 m height).

For this you need to use Google Earth for saving the image of the floor (Figure 1), make sure that the image is 100% perpendicular to the ground and the North up, and maybe crop the image later with an image editor program.

Note: We suggest to save the floor image with and without placemarks in order to facilitate a further step and to save the kml file with the placemarks coordinates (see Section 2), that will be useful later to change the simulation origin in order to georeferenciate it.



Figure 1. Floor image with and without placemarks

Measure both distances (x and y axes) covered in the final image using the ruler of Google Earth (see Figures 2, 3, 4). ENU: x = East direction, y = North direction, and z positive = up.

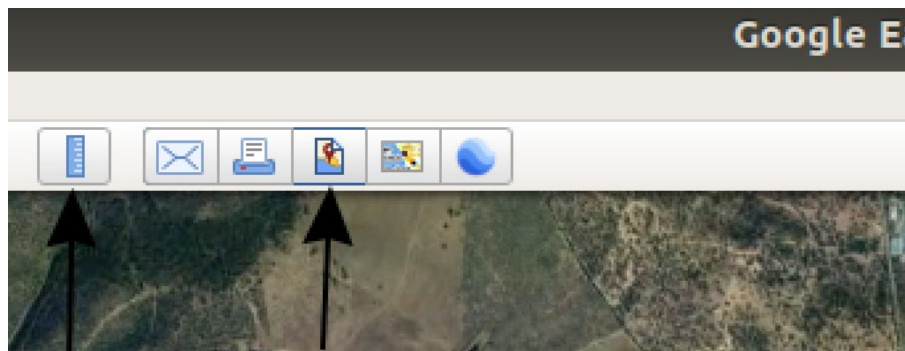


Figure 2: Google Earth Toolbar. “Ruler” and “Save image” tools needed

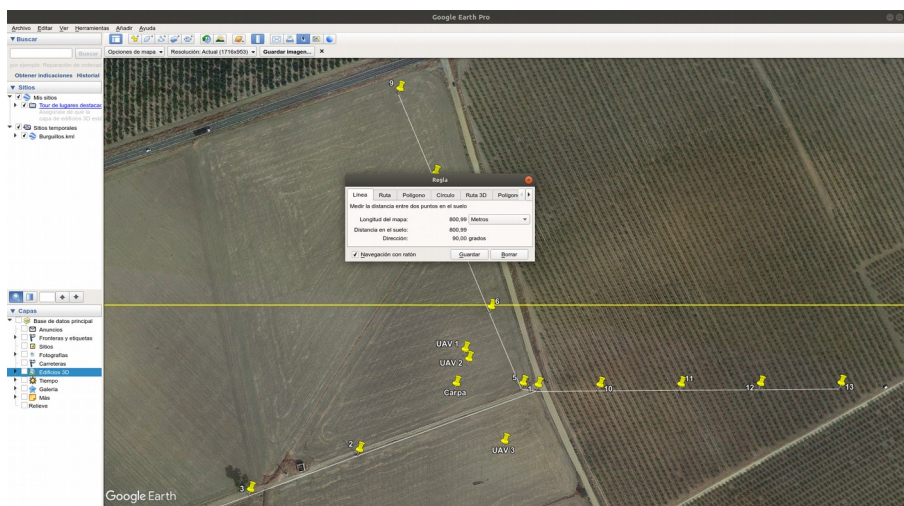


Figure 3. Measuring horizontal length

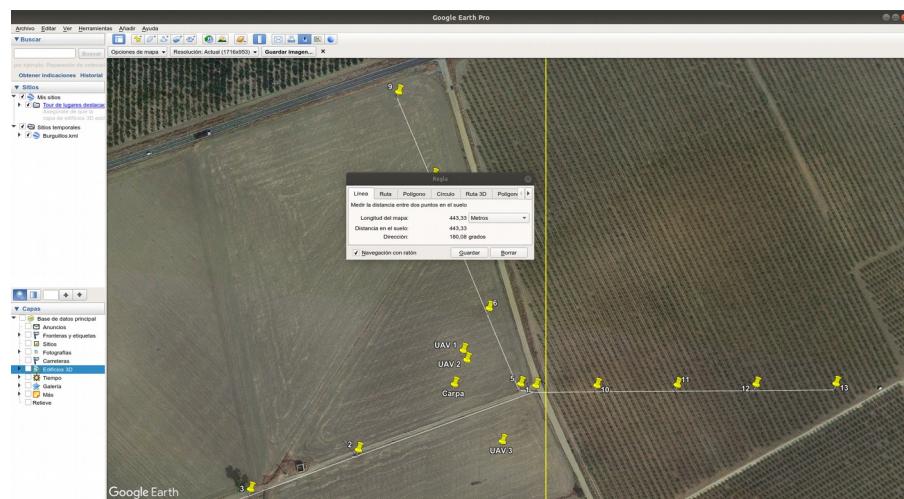


Figure 4. Measuring vertical length

Note: the simulation origin for this example will have the *Carpa* placemark coordinates (long, lat = -6.003676484441573, 37.564003804467). We need to save its coordinates in order to change the `sim_origin` parameter later in the launch file (see Code 1). Altitude value will be obtained from [opentopodata](http://opentopodata.org) database (alt = 65.36515808105469).

```
<Placemark>
  <name>Carpa</name>
  <LookAt>
    <longitude>-6.003382489503577</longitude>
    <latitude>37.56426456190803</latitude>
    <altitude>0</altitude>
    <heading>2.560816672533956</heading>
    <tilt>4.408122145555024</tilt>
    <range>318.2994133891621</range>
    <gx:altitudeMode>relativeToSeaFloor</gx:altitudeMode>
  </LookAt>
  <styleUrl>#m_ylw-pushpin200</styleUrl>
  <Point>
    <gx:drawOrder>1</gx:drawOrder>
    <coordinates>-6.003676484441573,37.564003804467,0</coordinates>
  </Point>
</Placemark>
```

*Code 1: Carpa coordinates from kml file (Point section ones)*

It is important that the name of the model is the same in: the folder name, name field in the `model.config` file, `model.sdf` name field, and `materials/scripts/burguillos_power_lines.material` material field.

In the `model.config`, a brief description of the model should be written. In the `model.sdf`, you define the visual box (the one rendered graphically in Gazebo) and the collision box (the one that defines where the dynamic objects can collide). The size field of those boxes defines the actual size of the floor, so it is here where you should put the measures that you carefully took before, e. g., Code 2:

```

<?xml version="1.0" ?>
<sdf version="1.5">
  <model name="burguillos_power_lines">
    <static>true</static>
    <link name="floor">
      <collision name="collision">
        <geometry>
          <box>
            <size>459.08 783.68 .1</size>
          </box>
        </geometry>
      </collision>
      <visual name="visual">
        <cast_shadows>false</cast_shadows>
        <geometry>
          <box>
            <size>459.08 783.68 .1</size>
          </box>
        </geometry>
        <material>
          <script>
            <uri>model://burguillos_power_lines/materials/scripts</uri>
            <uri>model://burguillos_power_lines/materials/textures</uri>
            <name>burguillos_power_lines</name>
          </script>
        </material>
      </visual>
    </link>
  </model>
</sdf>

```

*Code 2: Ground box defined in the model.sdf file*

Add the material script and texture (with the floor image saved from Google Earth) to the box with Y X Z dimensions as the following: 459'08 meters x 783'68 m x 0'1 m as measured before with the Google Earth Rule Tool. As Gazebo fills the floor's box faces with the texture, the upper face appears rotated -90° around Z-axis. For this reason, the box will be rotated later 90° around the vertical axis, and the dimensions of the box must be specified in the following order: Y X Z.

The floor image is added in the materials/textures folder, and it has to be later referred in the materials/scripts file (see Code 3).



```

material burguillos_power_lines
{
    technique
    {
        pass
        {
            ambient 0.5 0.5 0.5 1.0
            diffuse 0.5 0.5 0.5 1.0
            specular 0.2 0.2 0.2 1.0 12.5

            texture_unit
            {
                texture burguillos_power_lines.jpg
                filtering anisotropic
                max_anisotropy 16
                scale 1 1
            }
        }
    }
}

```

Code 3: material script file content

### - A launch file:

located in the “launch” folder.

IMPORTANT: we use the “launch\_gzworld.py”, located in [grvc-ual/px4\\_bringup/scripts](#) package, that is a Python script invoked in the launch file to add to Gazebo the “models” path of our package (Code 4). That script can be in any package, but you have to tell the launch in which package the script is located. Also, that script is what runs the world file with all its models, and sets the simulation origin (previously located) in the geographic coordinates where they should be. Note that the sim\_origin parameter is already placed in *Carpa* coordinates.

```

<launch>

  <!-- Launch Gazebo simulation -->
  <rosparam param="/use_sim_time">true</rosparam>
  <node pkg="px4_bringup" type="launch_gzworld.py" name="gazebo_world"
output="screen"
  args="-physics=ode -world=$(find grvc_gazebo_worlds)/worlds/burguillos_power_lines.world
      -add_model_path=$(find grvc_gazebo_worlds)/models
      -description_package=robots_description">
    <rosparam param="sim_origin">[37.564003, -6.003676484, 65.3651]</rosparam>
  <!-- [lat,lon,alt] -->
  </node>

</launch>

```

Code 4: Launching gzworld node from launch file. Georeference pose specified in rosparam sim\_origin

Note: the world file should exist before launching. We will use launch file to configure the world file, that is why in this document the launch file is explained before the other one.

### - A world file:

located in the “worlds” folder.

ATTENTION: we have to differentiate between where the Gazebo frame is located over the floor model picture, and to which global coordinates does the (0,0,0) of Gazebo corresponds (*Carpa* coordinates).

We may start having the floor model in Gazebo and the Gazebo frame in a place which is not the desired one (*Carpa*'s area of the floor image), and the purpose here is to make that frame coincide with a desired point that is on the floor. In order to fix this, we have to make a displacement to the floor box to make the desired position of the ground model coincide with the (0,0,0) of the Gazebo frame.

To get the quantity of the displacement in each of the axes, we just open a simulation with any displacement, e.g., (X, Y, Z, Rx, Ry, Rz) = (0, 0, -0.05, 0, 0, 1.57079), and move the model of the floor until the Gazebo frame is in the desired point. Then, we just note the pose values that appear in the lateral column of Gazebo selecting the ground model (X, Y) and write them in the world file (Code 7).

In order to obtain the floor box's displacement amounts in x and y axis, we suggest you to follow the next steps:

1. Create a world file that includes the floor box (see Code 5) without any displacement and change the texture .jpg file for the one that has the placemarks on it (see Code 3). Note that the box is displaced -0.05 meters (-box\_height/2) in Z-axis and it is also rotated 90° as previously explained (see Code 6).

```
<!-- The ground plane -->
<include>
  <uri>model://burguillos_power_lines</uri>
</include>
```

Code 5: Including ground from model file into world file

```
<!-- World state: rotate (ENU) and center the origin of the simulator -->
<state world_name='burguillos_power_lines_world'>
  <model name='burguillos_power_lines'>
    <pose frame=''>0 0 -0.05 0 0 1.570796327</pose>
  </model>
</state>
```

Code 6: World file state to be able to modify ground's pose in simulation NOT DISPLACED

2. Launch the world and infer the displacement by moving it manually in Gazebo until the Gazebo frame is over the desired point, in this case, *Carpa*, that can be easily located thanks to the placemarks. See Figure 5, 6, 7 and, in each one, look at the pose of the floor box that is at the left bar.

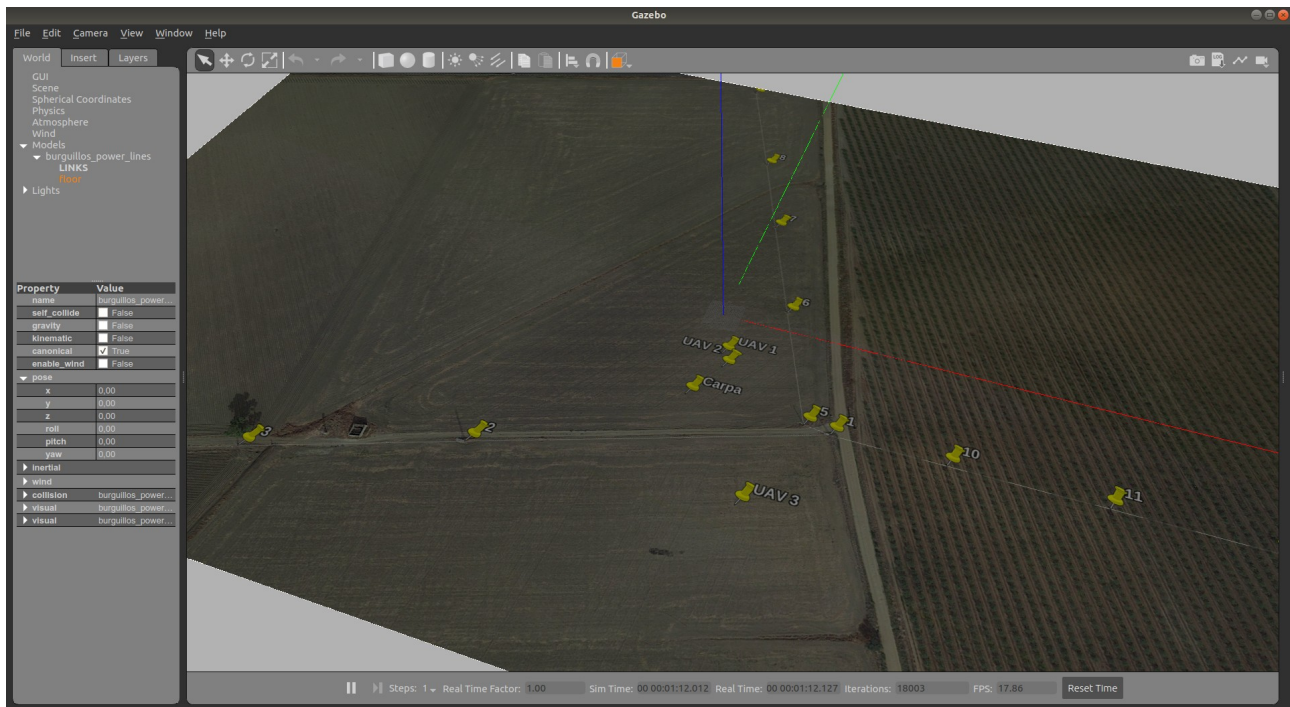


Figure 5. Floor box with placemarks. The floor box has not been displaced yet

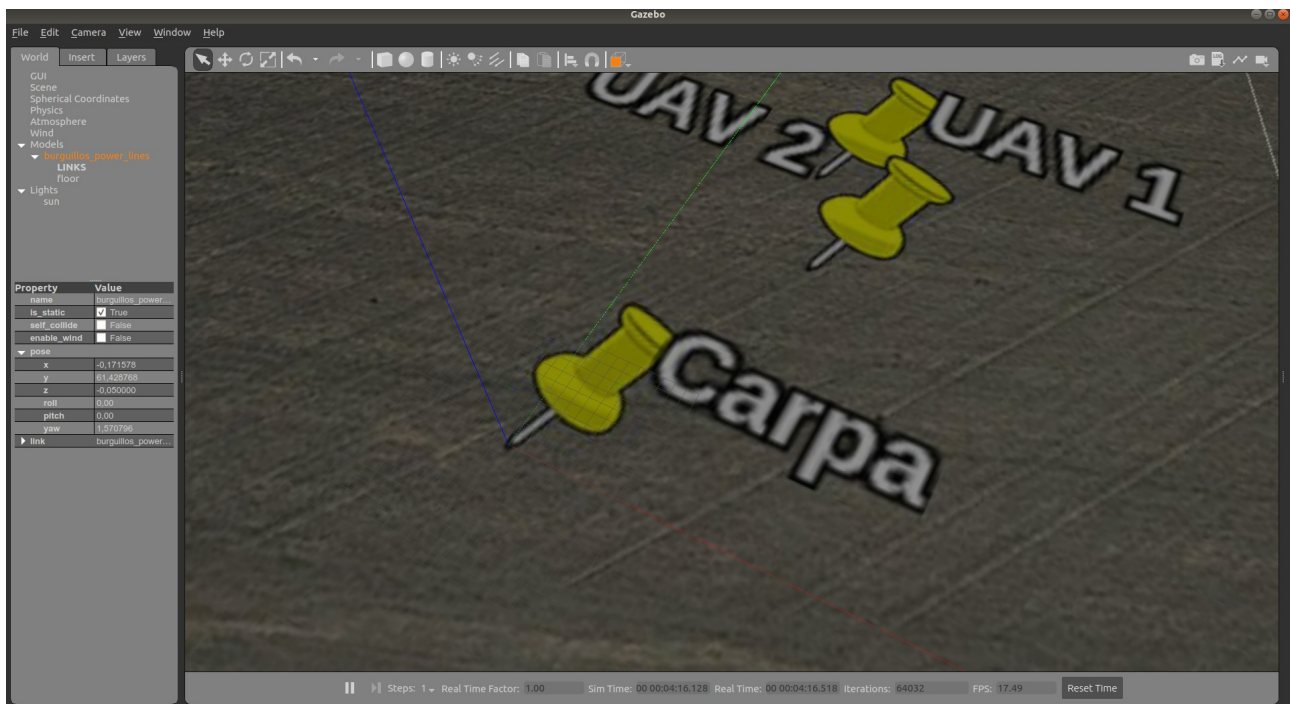


Figure 6. Floor box with placemarks. Moving the floor box until the Gazebo frame coincides with *Carpa*'s placemark position



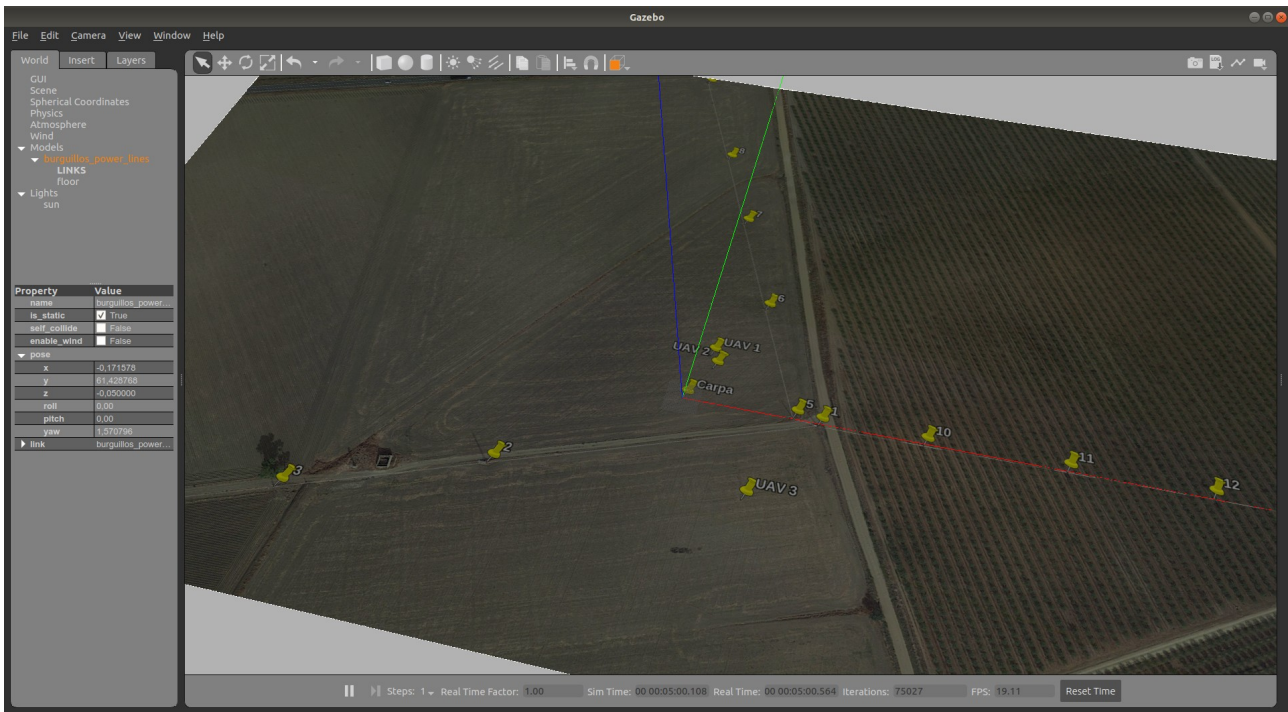


Figure 7. Floor box with placemarks. The floor box has been displaced

3. Take the floor box image back to the original one, without placemarks (see Code 3), and make the displacement done in Gazebo (-0.171578, 61.428768) in the world file, see it in Code 7.

```
<!-- World state: rotate (ENU) and center the origin of the simulator -->
<state world_name='burguillos_power_lines_world'>
  <model name='burguillos_power_lines'>
    <pose frame=''>-0.171578 61.428768 -0.05 0 0 1.570796327</pose>
  </model>
</state>
```

Code 7: World file state to be able to modify ground's pose in simulation DISPLACED

## 2. How to add pylons and wires with cylinders:

In this section, we are going to follow the next steps:

- Locate the pylons on Google Earth
- Estimate the pylons' height
- Transform to Cartesians
- Execute `wires_and_pylons_for_simulation.m` to get the data adapted for Gazebo

The idea is to create a file as `/grvc-utils/grvc_gazebo_worlds/worlds/burguillos_power_lines.world` that includes simple models of electric towers (pylons) and wires as primitive cylinders.

First, find the place and, using the satellite view and zooming in, locate placemarks in the origin of the shadows of pylons (Figure 8, Figure 9). You can also select points to locate the UAVs at the beginning of the missions, or the origin of Cartesian coordinates.



*Figure 8: Google Earth - Top View. Pylons' placemarks and wires lines.*

The line paths are just for visual help, and represent the wires between pylons (Figure 9).



Figure 9: Google Earth – Detail of Figure 8.

Arrange the placemarks in a folder, and select “Save Place As” after finishing with all the pins. Select export to .kml. There you will find the geographic coordinates of the points.

In case that you do not know the height of the pylons, we propose a method to estimate them. Knowing the height of one tower that, in first instance, we can estimate it, you can do a rule of thirds to roughly estimate the rest of heights measuring the length of the shadows. During the first visit to the place we can measure the real heights of the towers and correct the towers’ height.

Note: the real ground where we are working may not be flat, so the tower’s height in Gazebo probably will not all be the same. To make a more accurate approach, we will vary the tower’s height in simulation by knowing the altitude of each point thanks to [opentopodata](https://api.opentopodata.org/v1/eudem25m?locations=37.56399945115107,-6.00282095886654|37.56399945115107,-6.00282095886654) (will see later). We also propose to collect all the information that concerns to the pylons and its connections with the others in a file as the [pregraph.txt](#).

After treating the data of the coordinates inside the KML, you can extract the latitude and longitude coordinates of each tower.

```
37.56399232488992, -6.002762736955408
37.56342474674445, -6.00475133058687
37.56307650946646, -6.005956632441254
...
```

To extract the elevation of the ground (altitude), we use *Open Topo Data*. You can either use it online or install it on your machine and use it locally. Example of the online use:

<https://api.opentopodata.org/v1/eudem25m?locations=37.56399945115107,-6.00282095886654|37.56399945115107,-6.00282095886654>

You get a JSON response of the server with the elevations (floor altitude over the sea).

In a .txt file, we suggest you to arrange the coordinates like this:

```
37.56399232488992, -6.002762736955408, 66.28861999511719
37.56342474674445, -6.00475133058687, 60.09061813354492
37.56307650946646, -6.005956632441254, 60.859031677246094
```

To transform geographic to Cartesian coordinates, a transformation to UTM coordinates is needed: [https://en.wikipedia.org/wiki/Universal\\_Transverse\\_Mercator\\_coordinate\\_system](https://en.wikipedia.org/wiki/Universal_Transverse_Mercator_coordinate_system). This coordinate system converts latitude and longitude (degrees) to easting and northing (meters) from given lines of a grid. There are online tools and software libraries for this conversion, and the Cartesian coordinates would be the result of the difference between the UTM coordinates of the points minus the origin of coordinates in UTM. We created a header-only library to do all this (*geographic\_to\_cartesian.h*) that can be found in the [UAL repository](#) or in [grvc-utils](#).

There also exist a source code, [geographic\\_to\\_cartesian.cpp](#), that lets you to do the transformation from geograhic coordinates to XYZ coordinates. To use it, it is necessary to have a .txt file where the geographic coordinate list of points are written in the following format:

<u>Latitude</u>	<u>Longitude</u>	<u>Altitude</u>
37.56399232488992	-6.002762736955408	66.69452667236328
37.56342474674445	-6.00475133058687	59.94318389892578
37.56307650946646	-6.005956632441254	59.35587692260742
...		
37.564003804467	-6.003676484441573	65.36515808105469

where the last one has to be the origin of coordinates, *Carpa's* coordinates in this case, e.g., [input\\_data.txt](#). To execute it, you have to put in terminal:

```
roslaunch mission_lib geographic_to_cartesian path_to_input_file path_to_output_file
```

Once it is executed, you get the list of X Y Z points in the output file.

Afterwards, the idea is to give all that information the format needed for Gazebo, to define the cylinders of pylons and wires. For this, the [low res wires and pylons for simulation.m](#) script can be used to get the raw data of the position, length and rotation of the cylinders. The input of this script are the pylons' positions in Cartesians (obtained by the *geographic\_to\_cartesian.h* library) and the connections that each pylon has with the others. The output of the script is the (X, Y, Z, Rx, Ry, Rz) of each wire and pylon and is printed in the command window of MATLAB/Octave. We have to copy all of these data and it is necessary to adapt them to the .world file.

Using *VS Code* multicursors can make the adaptation of data task easier, as we have to instantiate many models that have the same structure. We are going to explain below the commands.

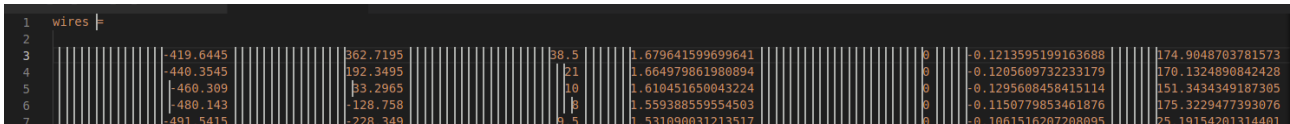
```
1 wires =
2
3
4
5
6
7
8
9
```

1	1	-419.6445	362.7195	38.5	1.679641599699641	0	-0.1213595199163688	174.9848703781573
2	1	-440.3545	192.3495	21	1.664979861980894	0	-0.1205609732233179	170.1324890842428
3	1	-460.309	33.2965	10	1.610451650043224	0	-0.1295608458415114	151.3434349187305
4	1	-480.143	-128.758	8	1.559388559554503	0	-0.1150779853461876	175.3229477393076
5	1	-491.5415	-228.349	9.5	1.531090831213517	0	-0.1061516207208095	25.19154201314401

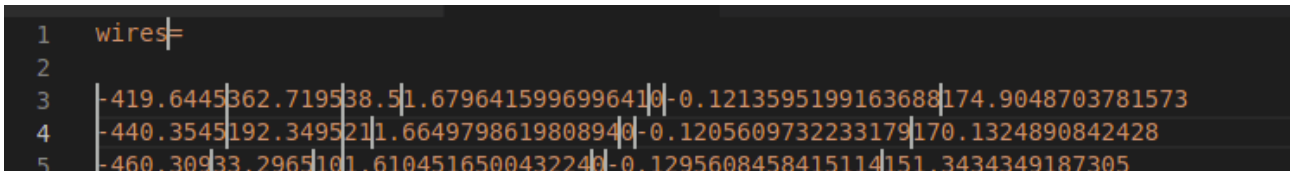
Figure 10: Initial code shape. Need to be adjusted



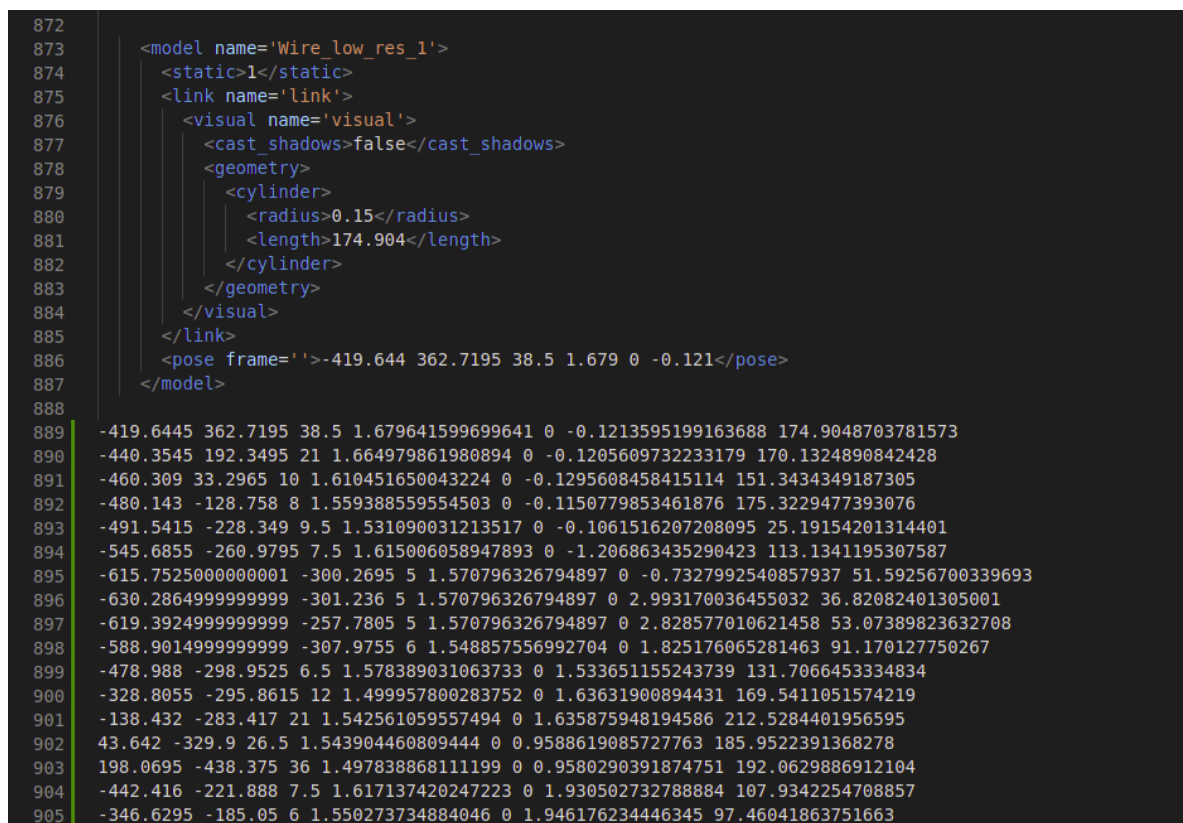
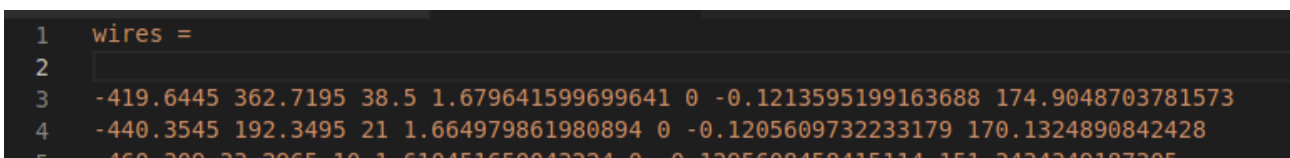
With multicursors, we need to reshape the data (Figure 10) for the .word: **Ctrl + Shift + L** (space highlighted, see Figure 11) => put a cursor in all the search occurrences.



Delete them and put one space to separate them (see Figure 12), repeat if necessary to get this:



Paste the shaped data (Figure 13) into the .world (Figure 14):





The idea is, for each line we copied, give them the shape of a Gazebo cylinder model (just like the one above). The six numbers of the `<pose frame="">...` are the first 6 numbers that we copied. The `<length>` field is the seventh number we copied. The radius will be the same within the same models. Using multicursors this is quite straightforward. Copy the string corresponding to the first block information (Figure 15), and paste it in all the following cursors:

```

873   ...<model name='Wire_low_res_1'>
874   ...<static>1</static>
875   ...<link name='link'>
876   ...<visual name='visual'>
877   ...<cast_shadows>false</cast_shadows>
878   ...<geometry>
879   ...<cylinder>
880   ...<radius>0.15</radius>
881   ...<length>174.904</length>
882   ...</cylinder>
883   ...</geometry>
884   ...</visual>
885   ...</link>
886   ...<pose frame=''>-419.644 362.7195 38.5 1.679 0 -0.121</pose>
887   ...</model>
888

```

Figure 15: First block information

Ctrl + Shift + Down Arrow => inserting multicursors down until all the new lines pasted have their own cursor (Figure 16).

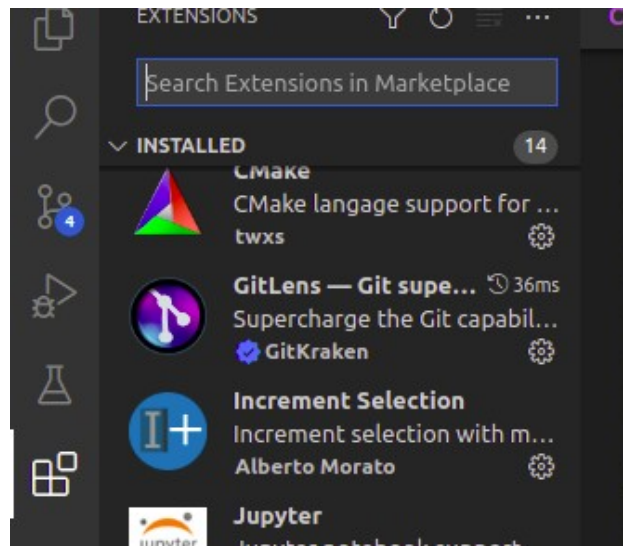
```

888
889 -419.6445 362.7195 38.5 1.679641599699641 0 -0.1213595199163688 174.9048703781573
890 -440.3545 192.3495 21 1.664979861980894 0 -0.1205609732233179 170.1324890842428
891 -460.309 33.2965 10 1.610451650043224 0 -0.1295608458415114 151.3434349187305
892 -480.143 -128.758 8 1.559388559554503 0 -0.1150779853461876 175.3229477393076
893 -491.5415 -228.349 9.5 1.531090031213517 0 -0.1061516207208095 25.19154201314401
894 -545.6855 -260.9795 7.5 1.615006058947893 0 -1.206863435290423 113.1341195307587
895 -615.7525000000001 -300.2695 5 1.570796326794897 0 -0.7327992540857937 51.5925670033969
896 -630.2864999999999 -301.236 5 1.570796326794897 0 2.993170036455032 36.82082401305001
897 -619.3924999999999 -257.7805 5 1.570796326794897 0 2.828577010621458 53.07389823632708
898 -588.9014999999999 -307.9755 6 1.548857556992704 0 1.825176065281463 91.170127750267
899 -478.988 -298.9525 6.5 1.578389031063733 0 1.533651155243739 131.7066453334834
900 -328.8055 -295.8615 12 1.499957800283752 0 1.63631900894431 169.5411051574219
901 -138.432 -283.417 21 1.542561059557494 0 1.635875948194586 212.5284401956595

```

Figure 16: Using multicursors to select the data

You will end up having many models with the same name. Install the following extension in VS Code: Increment selection (Figure 17).



*Figure 17: VSCode Plugin – Increment Selection*

Select the first repeated name model, `Ctrl + Shift + L` to select all, and then with the multicursors select only the number part. `Ctrl + Alt + I` => increment the numbers with the extension “Increment selection”.

### 3. Animated models in Gazebo:

The objective of this section is to explain how to do a simple cyclic animation of a model with Gazebo. Example file:

[grvc-utils/grvc\\_gazebo\\_worlds/worlds/ATLAS\\_renewable\\_power\\_plant.world](#)

With its corresponding roslaunch:

[roslaunch grvc\\_gazebo\\_worlds simulator ATLAS\\_renewable\\_power\\_plant.launch](#)

The result expected is to have spinning blades (Figure 18):



*Figure 18: Gazebo world with animated models*

The wind turbine has two different parts: one static and another one that moves, so we really need two different models. The model of the base is included in the world file as a regular static one (Figure 19):

```

58     </gui>
59
60
61     <!-- Include wind turbine (static base) -->
62     <include>
63       <uri>model://spinning_wind_turbine_base</uri>
64       <pose>-20 85 0 0 0 0</pose>
65       <name>spinning_wind_turbine_base_1</name>
66     </include>
67
68     <include>
69       <uri>model://spinning_wind_turbine_base</uri>
70       <pose>-40 105 0 0 0 0</pose>
71       <name>spinning_wind_turbine_base_2</name>
72     </include>

```

Figure 19: Including a model on code

But the blades are inside an “actor” block of code, in which the model is included, and a script is attached with the trajectory of waypoints to follow. Each waypoint has its time stamp (in seconds), and what Gazebo does is to interpolate in time the motion between waypoints (Figure 20). In this case, the x, y, z position of the waypoints are the same, and what changes is the rotation around the y axis.

```

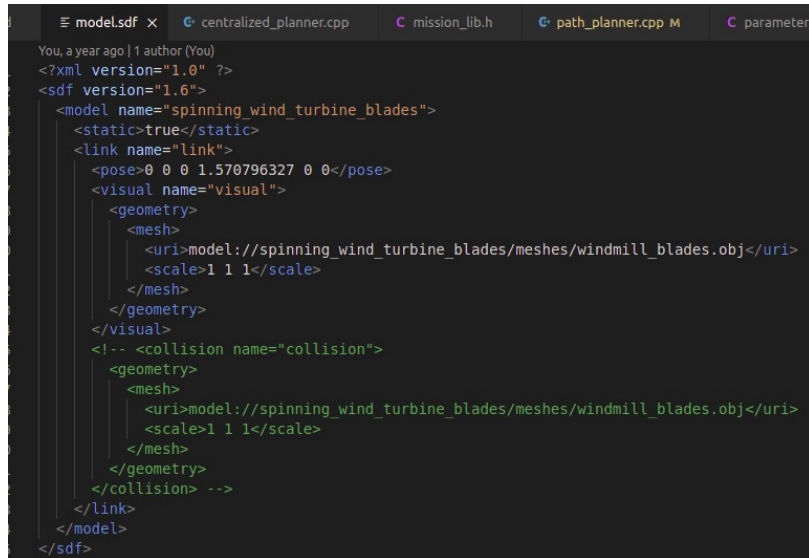
87     <!-- Include wind turbine (spinning blades) -->
88     <actor name="spinning_wind_turbine_blades_1">
89       <include>
90         <uri>model://spinning_wind_turbine_blades</uri>
91         <name>spinning_wind_turbine_blades_1</name>
92       </include>
93       <script>
94         <loop>true</loop>
95         <delay_start>0.000000</delay_start>
96         <auto_start>true</auto_start>
97         <trajectory id="0" type="spin">
98           <waypoint>
99             <time>0.0</time>
100            <pose>-20 85 21.6127 0 0 0</pose>
101          </waypoint>
102          <waypoint>
103            <time>1.0</time>
104            <pose>-20 85 21.6127 0 -1.570796327 0</pose>
105          </waypoint>
106          <waypoint>
107            <time>2.0</time>
108            <pose>-20 85 21.6127 0 -3.141592654 0</pose>
109          </waypoint>
110          <waypoint>
111            <time>3.0</time>
112            <pose>-20 85 21.6127 0 -4.71238898 0</pose>
113          </waypoint>
114          <waypoint>
115            <time>4.0</time>
116            <pose>-20 85 21.6127 0 -6.2832 0</pose>
117          </waypoint>
118        </trajectory>
119      </script>
120    </actor>

```

Figure 20: Waypoints given to the actor to make the rotation of the blades

Maybe the most difficult part for this is the previous preparation of the 3D models in Blender, as you need to set the origin of coordinates of the blades' model in the center of the rotation before exporting the model. Another important thing is to reduce the number of triangles of the mesh as much as possible to reduce the computation load of the simulation. This can be done with the "decimate" modifier of Blender. Both things will be discussed in the next and last section, but there are a lot of tutorials of Blender online.

It is important to consider the collision geometry if needed. You can uncomment the collision field of the model.sdf (Figure 21) to use the same 3D mesh of the object, but it will probably slow down the simulation. For efficient simulation, you could consider replacing the <mesh> field with simpler geometries such as spinning cylinders or boxes.



```
models.sdf x centralized_planner.cpp mission_lib.h path_planner.cpp M parameter_
You, a year ago | 1 author (You)
<?xml version="1.0" ?>
<sdf version="1.6">
  <model name="spinning_wind_turbine_blades">
    <static>true</static>
    <link name="link">
      <pose>0 0 0 1.570796327 0 0</pose>
      <visual name="visual">
        <geometry>
          <mesh>
            <uri>model://spinning_wind_turbine_blades/meshes/windmill_blades.obj</uri>
            <scale>1 1 1</scale>
          </mesh>
        </geometry>
      </visual>
      <!-- <collision name="collision">
        <geometry>
          <mesh>
            <uri>model://spinning_wind_turbine_blades/meshes/windmill_blades.obj</uri>
            <scale>1 1 1</scale>
          </mesh>
        </geometry>
      </collision> -->
    </link>
  </model>
</sdf>
```

Figure 21: Collisions commented



## 4. Model from photogrammetry:

An interesting feature of drones is the possibility of scanning a 3D environment through a photogrammetry tool, such as Pix4D or Autodesk ReCap Pro. The output is usually 3 files: obj (triangles mesh), jpg (texture), and mtl (material file). To put this into Gazebo, you can open it with MeshLab and export it to .dae.

You will need to change the ambient brightness, as the model will appear too dark In Gazebo by default. Modify the dae as a text file, changing the <ambient> from 0 0 0 0.9 to 1 1 1 1.

An example model in [grvc-utils/grvc\\_gazebo\\_worlds/models/karting\\_3D](#) (Figure 22) shows the result (it may take its time the first time it is loaded):

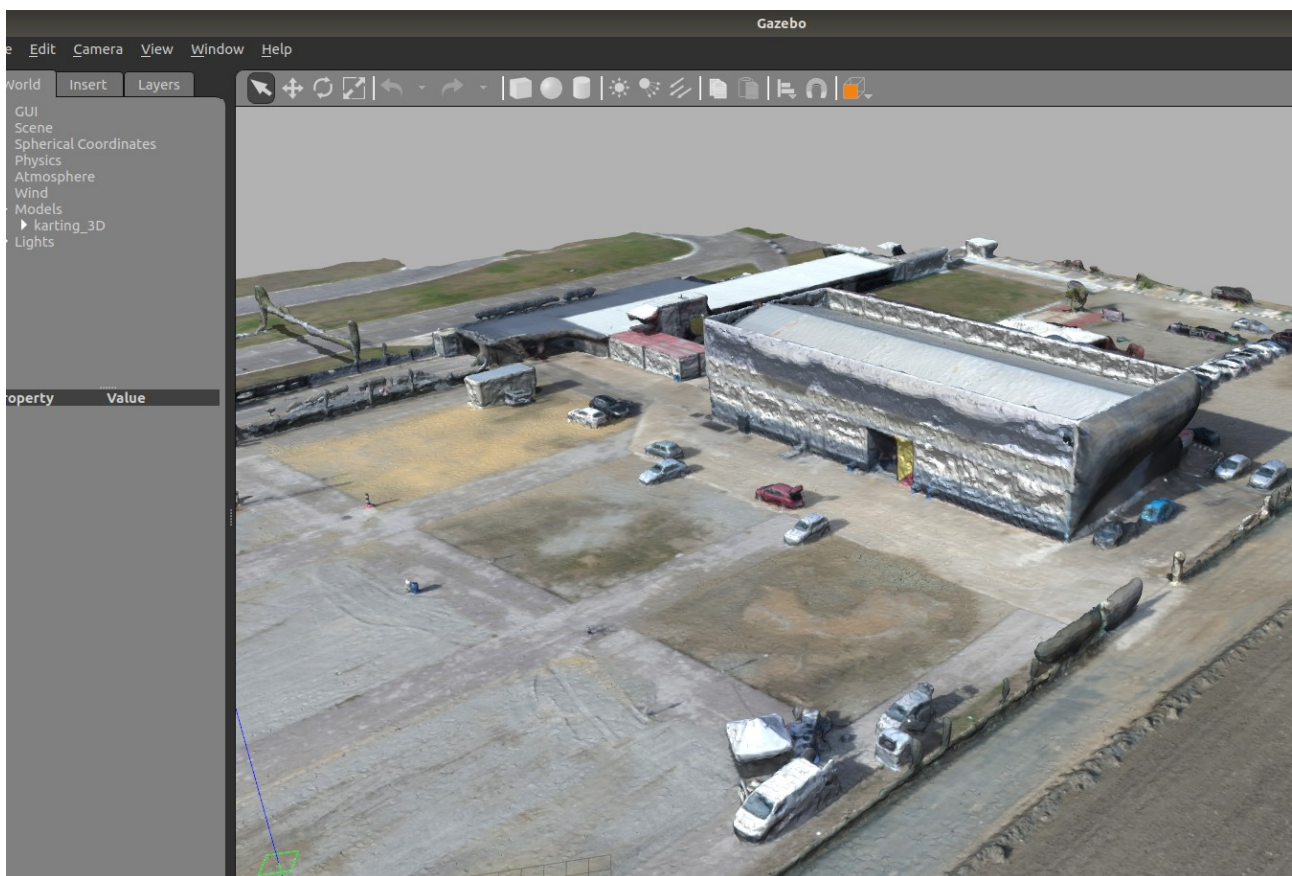


Figure 22: Gazebo world generated by photogrammetry

## 5. Model editing with Blender oriented to Gazebo:

Example of lowering resolution with Blender to a 3D model (human target in [grvc-utis/grvc\\_gazebo\\_worlds/models/target](https://github.com/grvc-utis/grvc_gazebo_worlds/models/target), originally created with Make Human), exporting it and importing it into Gazebo.

For 3D models, Gazebo supports .dae and .obj files. DAE is supposed to be a newer format that allows more functionalities (animations, deformation, etc.), and OBJ an older one that is only for static objects. For us, both will do the job, and OBJ files are usually lighter than DAE.

Import the DAE to Blender: it recognizes only where the textures are in the folder ../materials/textures/asdf.png, but the model looks dark/transparent. SOLUTION: select each component one by one and go to "shading" and join the "alpha" that comes out of the texture image with the BSDF. You should be able to see it (it can also be that the root of the textures is wrong). Reduce each component one by one with the "Decimate" modifier. Remove excess faces.

We painted parts of the model where the skinning of the clothes failed, specially the t-shirt. With texture paint, we painted in the color of the t-shirt the parts of the skin and jeans where the t-shirt went inside the model. We also corrected deformations by adding new faces (selecting and hitting F) and used "collapse" in the parts where we wanted a hodgepodge to join, filling in the way. We also put the GRVC logo on the t-shirt instead of the MakeHuman logo.

If you export it this way, that DAE is not recognized by Gazebo for some reason, but MeshLab does. It may be tempting to import it to MeshLab and export it later, but don't do it because the mesh loses the "smooth faces", leaving the mesh with a very polygonal face effect.

Better than that: still in Blender, select all components, TAB (edit mode), A (select all), Mesh, Shade => Smooth faces. This way you guarantee that the mesh has smooth faces and you don't see the faces blatantly. Now, export to DAE selecting again all components, checking "copy textures", "triangulate" and removing everything else you don't need (you can check "apply modifiers" if you don't want to apply them destructively).

This DAE is not yet recognized by Gazebo, so open it in text format and in the <library\_effects> tags, for each <effect id=...>, in each <technique sid="common">, remove everything inside (except the diffuse) and use technique blinn instead of the lambert, see Code 8.

```
<blinn>
  <ambient>
    <color sid="ambient">0.9 0.9 0.9 1</color>
  </ambient>
  <diffuse>
    <texture texture="eyebrow001_png-sampler" texcoord="v2_Eyebrow001-mesh-map"/>
  </diffuse>
</blinn>
```

*Code 8: Blinn technique*

Ambient is added because, if Gazebo is omitted, it sets a very low value and the model looks dark. In diffuse leave the line you had, in this case the one of the eyebrow texture.