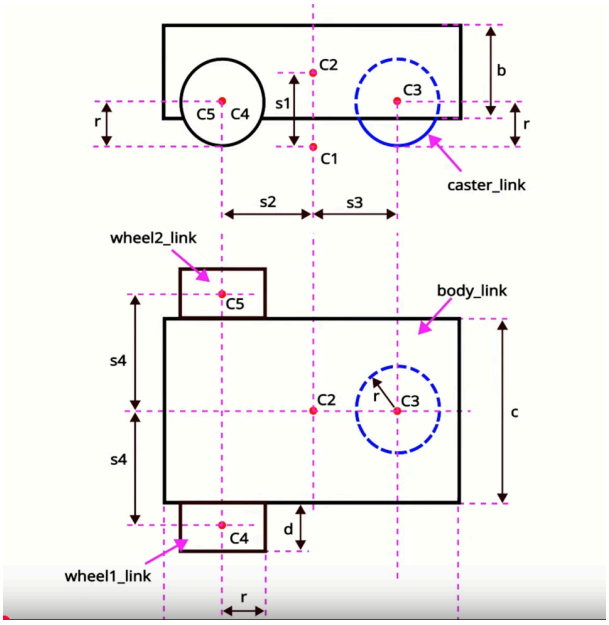
NOTES BASED ON THE https://www.youtube.com/watch?v=9sjTrpxtBaE
till 1:30:00, but you can watch last 15min by yourself, as it may be interesting or the file would be updated (also later would add code for that tutorial)
this guy's github - we can get a lot of inspiration from that, specially about integration of openCV and camera in ROS https://github.com/AleksandarHaber?tab=repositories

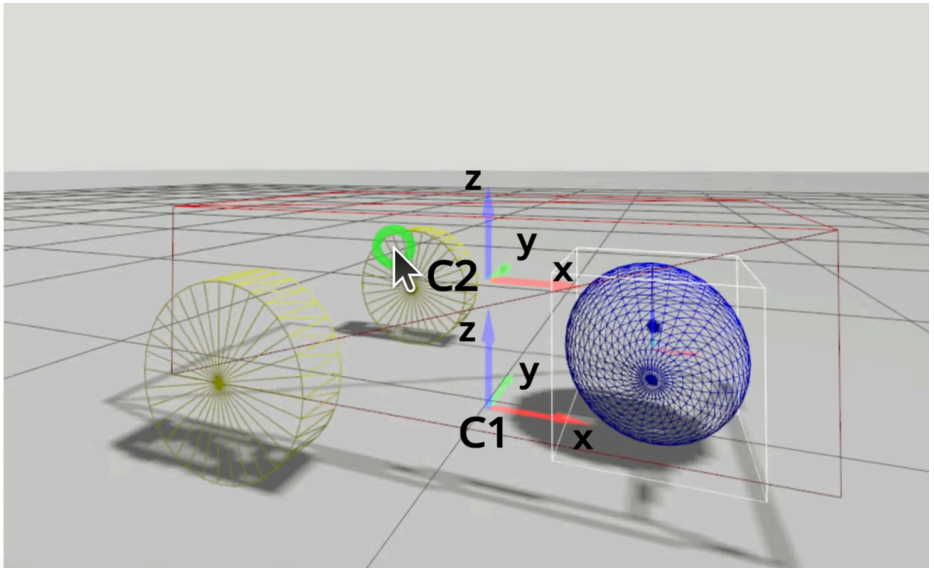
ros is using SI - kg, m, s
everything is described like links
after defining basic parameters you can define the rest related to them using formulas
<xacro:property name="a" value="1">
<xacro:property name="b" value="0.3">
<xacro:property name="c" value="0.6">
<xacro:property name="V1" value=$"a*b*c">

what must be defined
volume, mass, inertia in every axis (xyz)

in xacro you cannot use r^2, you should write r*r

to properly define every link you need to add <inertial> which defines every element mass center in relation to general mass center






<origin rpy=”0 0 0” xyz=”0 0 ${s1}”>
rpy - roll, pitch, yaw

inertia_wheel - has two coordinates frames, one rotated relating to another

```
robot.xacro
<!-- This macro defines the complete inertial section of the wheel -->
<!-- It is used later in the code -->
<xacro:macro name="inertia_wheel">
    <inertial>
    <origin rpy="1.570795 0 0" xyz="0 0 0"/>
    <mass value="${m2}"/>
    <inertia ixx="${I_wheel}" ixy="0.0" ixz="0.0" iyy="${I_wheel}" iyz="0" izz="${Iz_wheel}" />
    </inertial>
</xacro:macro>

<!-- This macro defines the complete inertial section of the caster -->
<!-- It is used later in the code -->
<xacro:macro name="inertia_caster">
    <inertial>
    <origin rpy="0 0 0" xyz="0 0 0"/>
    <mass value="${m3}"/>
    <inertia ixx="${I_caster}" ixy="0.0" ixz="0.0" iyy="${I_caster}" iyz="0" izz="${I_caster}" />
    </inertial>
</xacro:macro>
```

base_footprint - fictional links, that are not parts of the robot, needs to be defined to show how robot is fixed to the ground

```
<!-- ##########################################################    -->
<!-- START: Fictious link of the robot                             -->
<!-- ##########################################################    -->

<!-- We need to have this link otherwise Gazebo will complain -->
<link name="base_footprint"> </link>
<joint name="body_link_joint" type="fixed">
    <parent link="base_footprint"/>
    <child link="body_link"/>
</joint>

<!-- ##########################################################    -->
<!-- END: Fictious link of the robot                               -->
<!-- ##########################################################    -->
```

between 2 links must be a joint, it allows to rotate one object around another

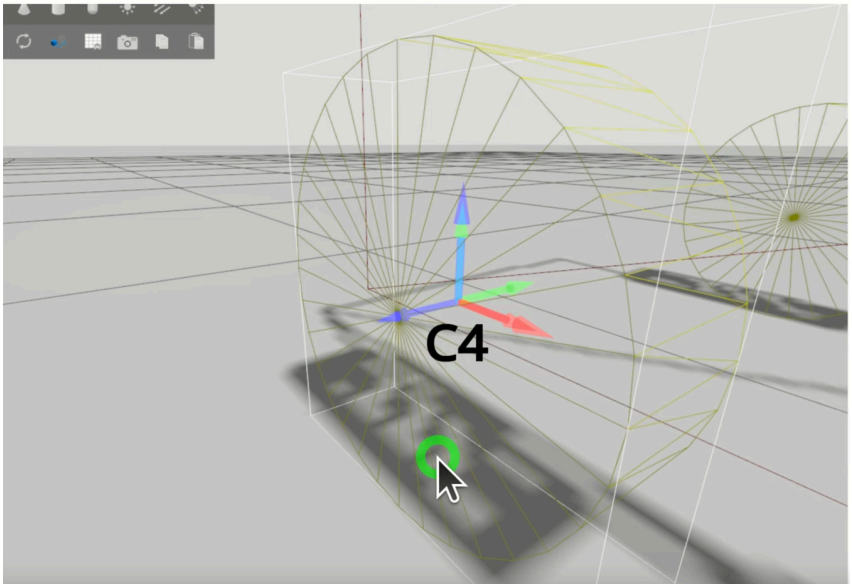
```
135
136 <!-- ######################################################    -->
137 <!-- START: Body link of the robot                            -->
138 <!-- ######################################################    -->
139 <link name="body_link">
140     <visual>
141         <geometry>
142             <box size="${a} ${c} ${b}" />
143         </geometry>
144     <!-- Displacement and orientation of the geometric center of the link with respec to the link f
145     <origin rpy="0 0 0" xyz="0 0 ${s1}"/>
146     </visual>
147
148     <collision>
149         <geometry>
150             <box size="${a} ${c} ${b}" />
151         </geometry>
152         <origin rpy="0 0 0" xyz="0 0 ${s1}"/>
153     </collision>
154
155     <xacro:inertia_body />
156 </link>
157
158 <!-- ###################################################### -->
159 <!-- END: Body link of the robot -->
160 <!-- ###################################################### -->
161
162
```

**visual descriptor** will enable you to render a box/object in gazebo. to define visual representation you need to define geometry - box with parameters a,b,c
**collision box** - if there are collisions between objects you need to define it in simulation (how much place your object need)

```
167 <joint name="wheel1_joint" type="continuous" >
168     <parent link="body_link"/>
169     <child  link="wheel1_link" />
170     <origin xyz="${-s2} ${-s4} ${-s1}" rpy="0 0 0" />
171     <axis xyz="0 1 0"/>
172     <limit effort="50000" velocity="10"/>
173     <dynamics damping="1.0" friction="1.0"/>
174 </joint>
175
176 <link name="wheel1_link">
177     <visual>
178         <origin rpy="1.570795 0 0" xyz="0 0 0"/>
179         <geometry>
180             <cylinder length="${d}" radius="${r}"/>
181         </geometry>
182     </visual>
183
184     <collision>
185         <origin rpy="1.570795 0 0" xyz="0 0 0"/>
186         <geometry>
187             <cylinder length="${d}" radius="${r}"/>
188         </geometry>
189     </collision>
190
191     <xacro:inertia_wheel />
192 </link>
```

type=”continuous”  - as the wheel can continuously rotate, there more types of joints e.g. …. not ok would be to tell, that wheel rotates -180-180deg, cause then it wouldn’t be able to rotate like an actual wheel
origin must be described in respect to the coordinate frame of the parent link (not mass centre, but coordinate system)

WHEEL_LINK - origin should be described with respect to the wheel_joint, in that situation we have to coordinates frames attached in one place and rotated one relating another (pi/2 in radians)




```
<joint name="wheel2_joint" type="continuous" >
    <parent link="body_link"/>
    <child  link="wheel2_link" />
    <origin xyz="${-s2} ${s4} ${r}" rpy="0 0 0" />
    <axis xyz="0 1 0"/>
    <limit effort="50000" velocity="10"/>
    <dynamics damping="1.0" friction="1.0"/>
</joint>

<link name="wheel2_link">
    <visual>
        <origin rpy="1.570795 0 0" xyz="0 0 0"/>
        <geometry>
            <cylinder length="${d}" radius="${r}"/>
        </geometry>
    </visual>

    <collision>
        <origin rpy="1.570795 0 0" xyz="0 0 0"/>
        <geometry>
            <cylinder length="${d}" radius="${r}"/>
        </geometry>
    </collision>
    <xacro:inertia_wheel />
</link>
```

SECOND PART OF THE VIDEO

REMEMBER! every time you want to use ROS, you should source ros
gz sim

how to check what is running in background
ps aux
ps aux | grep gz
how to kill the process
kill -9 PID
where PID is a PID number of the process you want to kill

which packages to install
sudo apt-get install gedit
sudo apt-get install ros-jazzy-state-publisher
sudo apt-get install ros-jazzy-state-publisher-gui
sudo apt-get install ros-jazzy-xacro
sudo apt-get install ros-jazzy-teleop-twist-keyboard

print the list of all Gazebo related packages
ros2 pkg list | grep gz

you should see ros_gz and ros_gz_bridge - it helps to communicate between ros2 and gz

source /opt/ros/jazzy/setup.bash
cd ~
mkdir -p ~/ws_mobile/src
ls -la

cd ws_mobile
ls -la

colcon build - make sure u are inside base folder, otherwise the command wouldnt work
cd ~/ws_mobile/src

ros2 pkg create --build-type ament_cmake mobile_robot

mkdir launch model parameters - create in directory 3 folders
model - will contain xacro and udf files - geometry of robot and other properties, simplify model
parameters yaml file, to bridge ROS2 and Gz topics - new, introduced in Gz Harmonics
launch - python file, which defines what we need to launch

cd ~/ws_mobile/
colcon build

cd ~/ws_mobile/src/mobile_robot/model/

code .
file name robot.xarco - I would add code to that file on repo
+ create file robot.gazebo - also would add smwhr

mu - friction parameters
in robot.gazebo you should define all parameters like in robot.xacro, to reference them and so program knows how to connect them together, otherwise it wouldnt function -> names should be the same!!!

caster link has a incredibly low friction, so we just want to slide on the surface

plugin - controller

receive command velocity messages from the topic
(based on pressed keyboard keys)

now you dont need to define torque in robot.gazebo, as you should define it in robot.xacro as limit effort:

```
<joint name="wheel1_joint" type="continuous" >
    <parent link="body_link"/>
    <child  link="wheel1_link" />
    <origin xyz="${-s2} ${-s4} ${r}" rpy="0 0 0" />
    <axis xyz="0 1 0"/>
    <limit effort="50000" velocity="10"/>
    <dynamics damping="1.0" friction="1.0"/>
</joint>
```

in robot.xacro we defined some fixious base_footprint, as it was needed for that part for ondometry

```
    <!-- topic, frames, and odometry -->
        <topic>cmd_vel</topic>
        <tf_topic>/tf</tf_topic>
        <odom_topic>odom</odom_topic>
        <frame_id>odom</frame_id>
        <child_frame_id>base_footprint </child_frame_id>
        <odom_publisher_frequency>30</odom_publisher_frequency>
```

close code, check if files are in the correct folder

```
flybozon@FlyBozon:~/ws_mobile/src/mobile_robot/model$ ls -la
total 20
drwxrwxr-x 2 flybozon flybozon 4096 mar  5 11:29 .
drwxrwxr-x 7 flybozon flybozon 4096 mar  5 10:52 ..
-rw-rw-r-- 1 flybozon flybozon 1454 mar  5 13:09 robot.gazebo
-rw-rw-r-- 1 flybozon flybozon 5156 mar  5 13:09 robot.xacro
```

NEXT
we need to create a yaml parameter file that will bridge ROS2 Jazzy and Gazebo. will allow us to communicate our keyboard keys as robot commands
cd ~/ws_mobile/src/mobile_robot/parameters
code .
create a bridge_parameters.yaml

the purpose of this file is to bridge topics of ros2 and gz. we cannot simply communicate between ros2 and gz using any topics, since ros2 and gz have there own topics and we need to pair them

DiffDrive plugin use few topics, so u should define bridges for all of them

**5. Create Python launch file**
cd ~/ws_mobile/src/mobile_robot/launch/
in that directory open
code .
create a gazebo_model.launch.py file

in that code function returns LaunchDescriptionObject which defined what file should be launched - will couple all gz files with all ros2 files

specify the robot xacro name
define the name of the pkg
!!!those names should match names you gave in previous files!!!
convert xacro model file to xml -> return xml file
line 47 - standard ros path, not one that we created!

ATTENTION! although we define to stop gz simulation after closing the window (line 55-56), gz sim can still run in background, so u may need manually kill it after

robot_description - xml file

```
# this is very important so we can control the robot from ROS2
bridge_params = os.path.join(
get_package_share_directory(namePackage),
'parameters',
'bridge_parameters.yaml'
)
```

store path to bridge_parameters.yaml

**6. Final adjustments and run the model**
cd ~/ws_mobile/src/mobile_robot/
gedit package.xml

add

```
<exec_depend>joint_state_publisher</exec_depend>
<exec_depend>robot_state_publisher</exec_depend>
<exec_depend>gazebo_ros</exec_depend>
<exec_depend>xacro</exec_depend>
<exec_depend>ros_gz_bridge</exec_depend>
```

adjust CmakeLists.txt file. we need to tell ros2 that important files are in the two new folders “lunch” and “model”
cd ~/ws_mobile/src/mobile_robot/
gedit CMakeLists.txt

add lines

```
install(
        DIRECTORY launch model parameters
        DESTINATION share/${PROJECT_NAME}
)
```

```
# find dependencies
find_package(ament_cmake REQUIRED)
# uncomment the following section in order to fill in
# further dependencies manually.
# find_package(<dependency> REQUIRED)

install(
        DIRECTORY launch model parameters
        DESTINATION share/${PROJECT_NAME}
)

if(BUILD_TESTING)
  find_package(ament_lint_auto REQUIRED)
  # the following line skips the linter which checks for copyrights
  # comment the line when a copyright and license is added to all source files
```

perform final build of the workspace and the pkges
cd ~/ws_mobile/
colcon build

**7. Test simulation**
source /opt/ros/jazzy/setup.bash
source ~/ws_mobile/install/setup.bash

run the model in gz
ros2 launch mobile_robot gazebo_model.launch.py

ATTENTION! you can face issue I had, that instead of running your new simulation you would open some previous. To prevent that you should kill gz sim server manually
ps aux
ps aux | grep gz
kill -9 PID
you must left with only

```
flybozon@FlyBozon:~/ws_mobile$ ps aux | grep gz
flybozon   29907  0.0  0.0  11804  2288 pts/0    S+   14:46   0:00 grep --color=auto gz
```

1:30:00
**8. Control the robot using keyboard keys**
source /opt/ros/jazzy/setup.bash
ros2 run teleop_twist_keyboard teleop_twist_keyboard

TO BE CONTINUE…