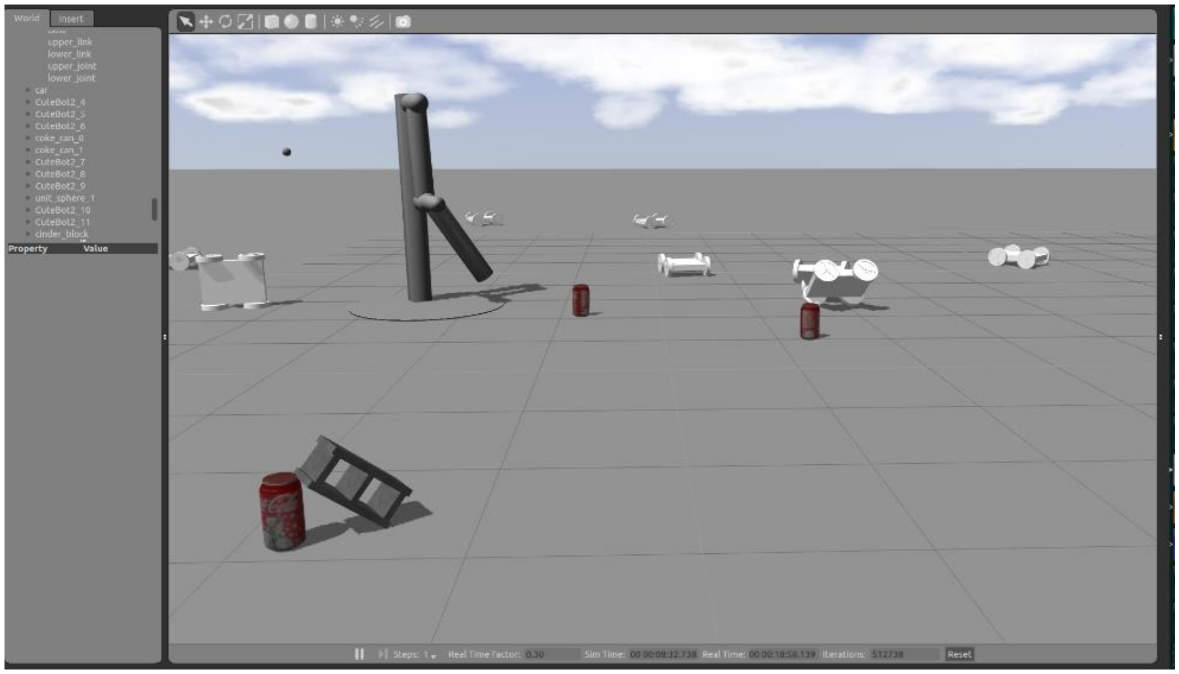
Modeling for Gazebo

A Design Guide for Proper Exporting from Solidworks for Gazebo Simulation



Contents

Table of Figures

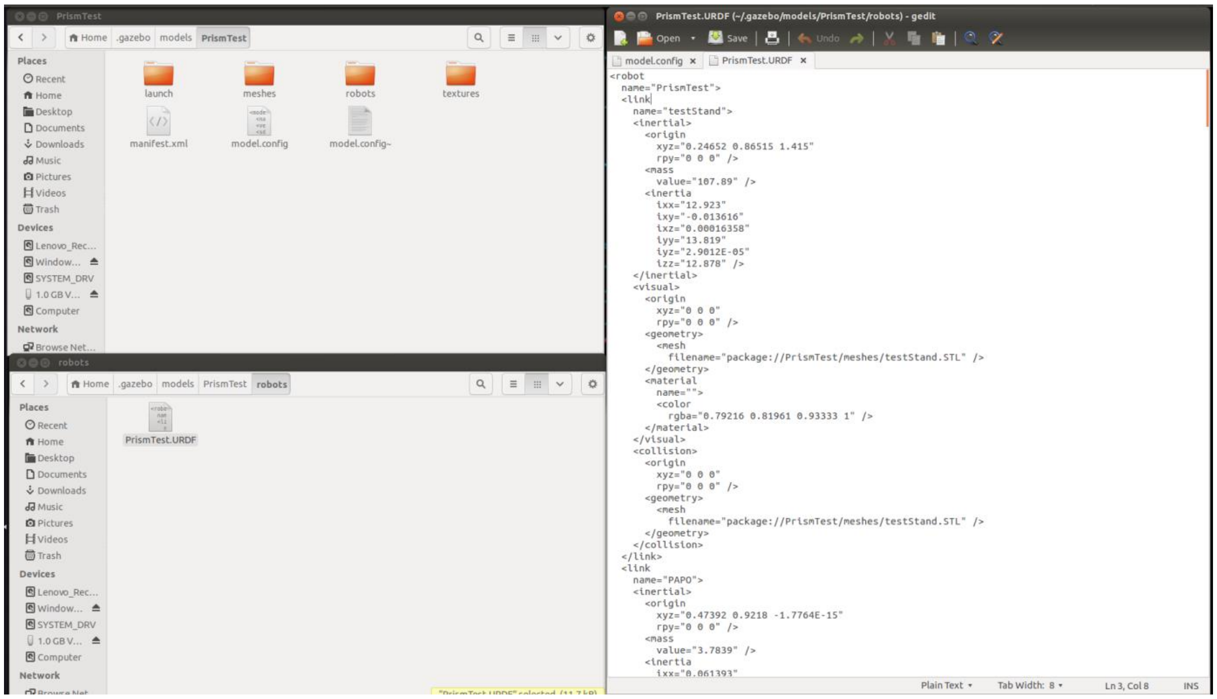
Read me

The purpose of this document is to guide the user through the set-up, export from Solidworks, post processing, and import into Gazebo of a *feasible* robot model. Feasible models are ones that only contain revolute, continuous, prismatic, free, and fixed joints, and do not contain N-bar linkages where a series of links connect back to a link previously in the chain. Topics covered include: An introduction to Gazebo and the special file formats it uses, types of joints, top-down and bottom-up design practices, and procedures for bringing a Solidworks assembly through the entire process to gazebo

What is Gazebo?

Gazebo is a simulation environment that allows for testing of complex systems – robotic or otherwise. It has many uses, including testing the dynamics of a control system before the system has been actualized. You can also use it to test new code or physical systems that might be too dangerous or expensive to test in the real world.

Our goal is to highlight the key processes involved in taking a CAD model in Solidworks and bringing it through the exportation process into Gazebo for testing. There are many minute details that must be attended to if you are to come out with a system that looks or works anything like the one you intended.



```
<robot
  name="PrismTest">
  <link
    name="testStand">
    <inertial>
      <origin
        xyz="0.24652 0.86515 1.415"
        rpy="0 0 0" />
      <mass
        value="107.89" />
      <inertia
        ixx="12.923"
        ixy="-0.013616"
        ixz="0.00016358"
        iyy="13.819"
        iyz="2.9012E-05"
        izz="12.878" />
    </inertial>
    <visual>
      <origin
        xyz="0 0 0"
        rpy="0 0 0" />
      <geometry>
        <mesh
          filename="package://PrismTest/meshes/testStand.STL" />
      </geometry>
      <material
        name="">
        <color
          rgba="0.79216 0.81961 0.93333 1" />
      </material>
    </visual>
    <collision>
      <origin
        xyz="0 0 0"
        rpy="0 0 0" />
      <geometry>
        <mesh
          filename="package://PrismTest/meshes/testStand.STL" />
      </geometry>
    </collision>
  </link>
  <link
    name="PAPO">
    <inertial>
      <origin
        xyz="0.47392 0.9218 -1.7764E-15"
        rpy="0 0 0" />
      <mass
        value="3.7839" />
      <inertia
        ixx="0.061393"
```

Figure 1: the folder with all the robot data and the contents of a URDF.

File Types: URDF and SDF

The original file for performing simulation is the Universal Robot Description Format, or URDF. This xml-like file type is used heavily in ROS for simulation and testing; it is a supported file type for rviz and other ROS tools. The file is a tree structure of child links (like wheels and arms) connected to parent links (like the chassis) by a series of joints. This file is the description of how your system moves internally, and this will determine how it can interact with the environment. The capabilities of the file however are limited in comparison to the new Simulator Description Format, or SDF. SDF files are what Gazebo uses when performing simulation, and in fact before importing one of your URDF models, it will convert it into an SDF. This file format includes more details like friction, damping, and environment properties like heightmaps and lights that are excluded from the URDF file. That being said, we will start with the URDF exporting process because of the useful SW2URDF plugin available for Solidworks users. This allows you to manually piece together a URDF file for use in simulation, and we will be looking at that process shortly.

Tree Limitations:

Because of the tree structure, there are limitations involving recursion that can make some systems difficult-or impossible- to model. For example, think about how you would model a four-bar linkage using a tree structure. The base link would connect to the crank, which connects to the floating link, which connects to the rocker. However, the base link is the parent, so it cannot be the child of the rocker. To overcome this dilemma, you can use special Gazebo tags to enable more of a map-like structure, creating an invisible link between the rocker and base.

Solidworks Concepts

Coordinate Frames / Origins

Coordinate Frames are a key concept for standard system control problems, and are just reference points which you can use to keep track of a body's 6-DOF position and orientation. In the realm of URDF models, there are three types of origins that you need to be familiar with. An origin that is built into the child link in a URDF tree is known as a Child Origin. You can see on wheel D in Figure 2 that the origin moves with the child link (the wheel) and is rotated. An origin built into the parent body is a Parent Origin, and can be seen on the main body on the robot at the right. The last type of origin is the Global Origin. This is an origin built at the assembly level, as can be seen below the robot. This is an important origin because it is what relates the initial position of the robot in space to its spawn location in Gazebo, in addition to whatever link relations you apply during the export process (for example creating a prismatic joint which references the global origin).

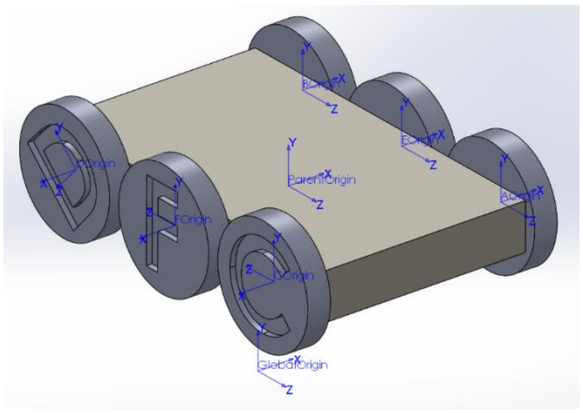


Figure 2: Various coordinate systems in a robot

Axes

An Axis is a line oriented in space, used as a reference for motion. Revolute and Continuous joints will rotate around axes, and prismatic joints will translate along axes. Just like Origins, Axes can be in the child, parent, or global frame.

Supported Links

The four kinds of links supported by the SW2URDF exporter are Revolute, Prismatic, Continuous, and Fixed. Each requires special attention to how they are exported to make sure they work as intended (as you can see below, where a series of 9 joints was exported each with a different set of parent coordinates, resulting in only three viable prismatic joints).

Revolute

A revolute joint is a 1-degree-of freedom joint that allows rotation about a single axis. It can be used to describe an arm or hinge joint. It is capable of multiple rotations, but allows for joint limitations so that you could restrict the motion to a 90 degree arc, for example. To properly export a Revolute joint, the origin must be on the child link. In other words, when selecting your origin during the export process, the origin must be on the link that will be rotating (for example, a wheel). The actual axis of rotation can be in the parent link, the child link, or in the global frame and the joint will still operate properly. It is worthwhile to note here that if you do not enter limits for your joint, then it will not move.

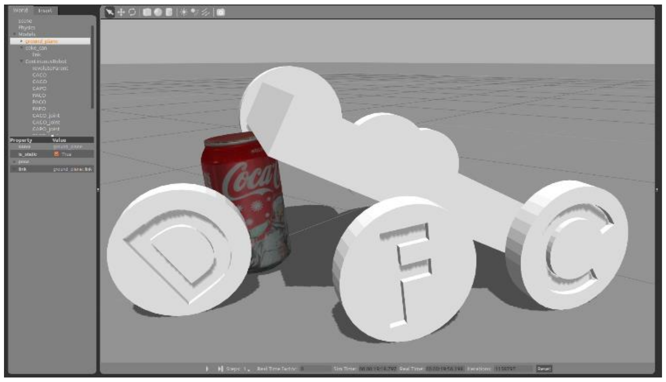
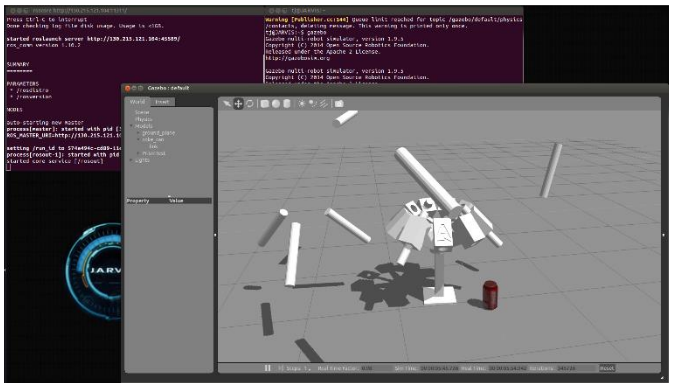


Figure 3: Revolute joint import test wheels D and F failed while C works

If you do not specify a lower and upper angular bound in addition to a force and velocity limit, then these will go in as zeroes and the link can only move zero radians at zero radians per second! (that is to say not at all)

Prismatic

A prismatic joint is a 1-degree of freedom linear translating joint. It allows for translation along an axis, but not rotation around the axis. There is an SDF joint called Piston that allows for that kind of movement, but that is not avail bile through the current exporter. To properly export a prismatic joint, the origin must be from the global frame. In Solidworks, that means the origin at the Assembly level. The prismatic axis (the axis along which the part will translate) can be in the child link, the parent link, or in the global frame and the link will still operate properly. It is worthwhile to note here that if you do not enter limits for your joint, then it will not move. If you do not specify a lower and upper bounds as well as a force and velocity limit, then these will go in as zeroes and the link can only move zero meters at zero meters per second!



```
press ctrl-c to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://130.215.121.104:45589/
ros_comm version 1.10.2


SUMMARY
========

PARAMETERS
 * /rosdistro
 * /rosversion

NODES

auto-starting new master
process[master]: started with pid [
ROS_MASTER_URI=http://130.215.121.1

setting /run_id to 574e494c-cd89-11
process[rosout-1]: started with pid
started core service [/rosout]
```

```
Warning [Publisher.cc:144] Queue limit reached for topic /gazebo/default/physics
/contacts, deleting message. This warning is printed only once.
tj@JARVIS:~$ gazebo
Gazebo multi-robot simulator, version 1.9.5
Copyright (C) 2014 Open Source Robotics Foundation.
Released under the Apache 2 License.
http://gazebosim.org

Gazebo multi-robot simulator, version 1.9.5
Copyright (C) 2014 Open Source Robotics Foundation.
```

Figure 4: Prismatic joint import test

Continuous

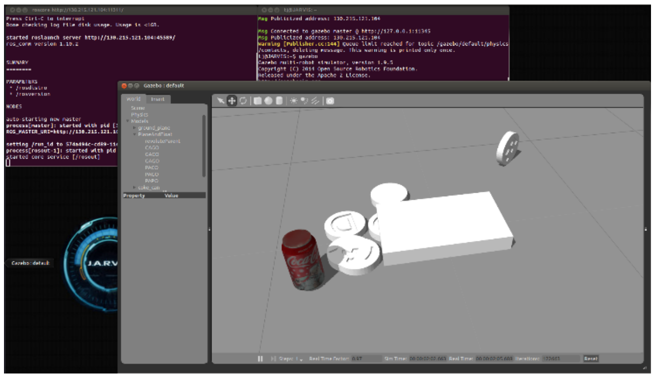
A continuous joint is best represented by a wheel, which can rotate continuously. Similar to revolute joints, a proper continuous joint will have the origin on the child link. The axis can be in the global, child, or parent frame. This joint does not require velocity and effort limits but it isn't a bad idea to include them just in case

Fixed

A fixed joint is used when you want a child link to be incapable of movement with respect to the parent link. In other words, the link is fixed. A proper fixed link makes use of a global origin, otherwise it will export in an erroneous location. This joint does not require limits of any type.

Planar and Floating

If you decide after hitting the *Preview and Export* button that you want to change the link type, you will want to change it in the Joint Editing page. If you click the pull down menu on this page, you might notice that you have two more options than before: Planar and Floating joints. Currently these both do the same thing, where the link generated is free from the parent body, disconnected and unrelated to it. You can use these kinds of links if you want to import multiple free bodies into the simulator. Remember though, that if you want to be able to manipulate one of these free bodies with the main robot model, you would need to enable internal collision detection because the body is considered part of the robot even though it does not maintain a physical relationship with the parent body.



```
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://130.215.121.104:45589/
ros_comm version 1.10.2


SUMMARY
========

PARAMETERS
 * /rosdistro
 * /rosversion

NODES

auto-starting new master
process[master]: started with pid [
ROS_MASTER_URI=http://130.215.121.1

setting /run_id to 574a494c-cd89-11
process[rosout-1]: started with pid
started core service [/rosout]
```

```
Msg Publicized address: 130.215.121.104

Msg Connected to gazebo master @ http://127.0.0.1:11345
Msg Publicized address: 130.215.121.104
Warning [Publisher.cc:144] Queue limit reached for topic /gazebo/default/physics
/contacts, deleting message. This warning is printed only once.
tj@JARVIS:~$ gazebo
Gazebo multi-robot simulator, version 1.9.5
Copyright (C) 2014 Open Source Robotics Foundation.
Released under the Apache 2 License.
```

Figure 5: Free and Planar joint import test

Collision vs. Visual Models

Shown below is the AndyMark 4" wheel used in the standard FRC Kitbot. This wheel may not seem that complicated, but if you tried to use if as-is for simulation, you would quickly realize that it takes a lot of effort to check the collision models of complex parts! A collision model is the model that Gazebo references when the robot comes in contact with anything in the simulator. The simulator has to make sure that when your robot goes to grab something, it doesn't just pass right through it. To do this, it checks the faces of your collision models to see if they are intersecting anything. If you tried to use the raw wheels from CAD, you would likely get a drastically reduced real-time factor; I've gotten down to 0.00% real-time when trying to simulate a robot straight from export. The key is in how complex the part is. This wheel as shown on the far-right has over 18,000 faces! That's just too much for the simulator to handle when it is checking for collisions with your model.

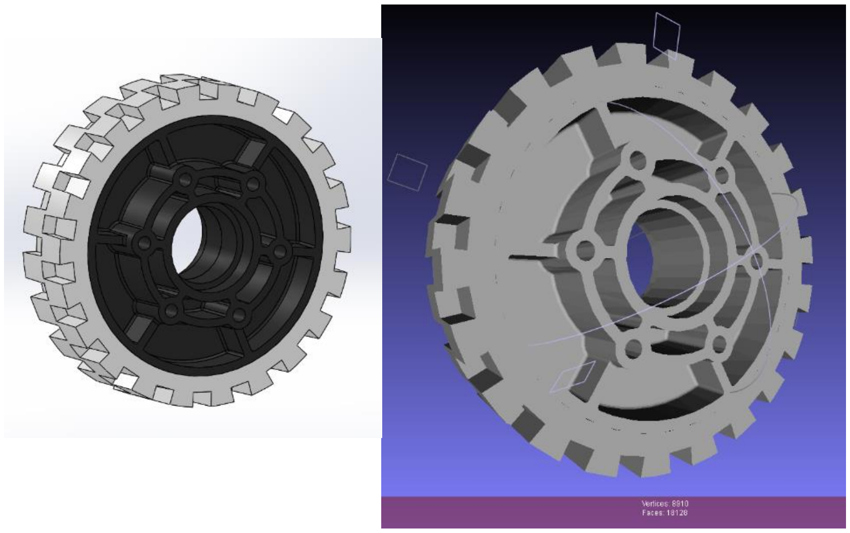


Figure 6: This wheel has a lot of detail that is not required to simulation

The secret to creating a useable model for simulation lies in having different files for you Visual and Collision models. A visual model only has to be rendered on the screen, so a complex shape like the one above is fine. The collision for the AndyMark wheel only needs to be a rough cylinder for the model to work as intended. As you can see below, if you can render the robot you wish to model using a very simple series of geometric shapes, then the simulator will have a much easier time and you will see faster real-time factors. Deciding what you want to use for Collision models is no easy task. You want the simplest form that will mimic how the system is supposed to operate, but will not overburden the simulator's processing. The simplest shape to render is a tetrahedron, because it can be rendered with only four faces. But this kind of shape would be all but useless as a wheel, because it wouldn't roll! It will take some practice and maybe even some testing to determine what the best option is. There will always be a tradeoff between collision accuracy and model simplicity. If complex geometry is required for proper dynamics, be ready to sacrifice simulation speed.

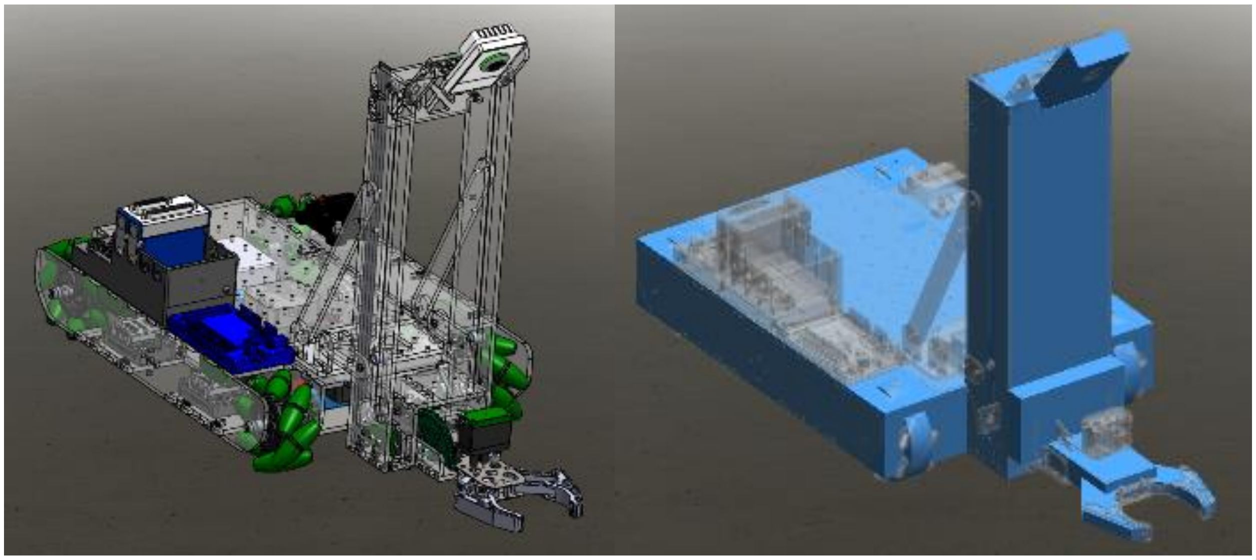


Figure 7: Gears bot Visual model (left) and collisional model (right)

Every Collision model will have different needs to fulfill but there are some good guidelines to follow.

1. Group Rigid bodies together

Any group of parts that does not move relative to each other is considered a rigid body. Base your collisional models around these and not the individual parts inside of them. For example the motor on an arm or claw does not need a separate collisional model it would be part of the arm it was attached to

2. Use only what you need.

Remember that only outside surfaces of your model are important to collisions so unless there something inside that you need to worry about the inside geometry can be ignored. For instance a closed cardboard box is hollow inside but the collisional model can be solid to reduce the complexity of the geometry.

3. Basic shapes work best.

If the intent of you collisional model can be captured with a box or a cylinder, use those! A cylinder is 1000x smaller (file size wise) than the wheel above.

4. It can be ok for your collisional models to collide

If you are not planning on using internal collisions in your model it is perfectly fine to have collisional models intersect as shown by the wheels and body of Gearsbot, above.

With the current exporter, there is no way to export collision and visual models simultaneously. The best method we have come up with so far is to export two models, one which is all visual and one which is collisional. The best way to do this is to save a copy of the assembly with collisional parts in it and use the second model to export the collisional model. This will save the trouble of having to constantly change the export configuration. If you place the collisional models directly over the visual model as shown above, Then creating the combined robot is as “easy” as combining the URDF files. The files contain positions for different types of models, and so when you export the visual model the section under visual and also the section under collision will both be the visual model. It’s the same case for the collision model, so by taking the collision model link elements and replacing the one if the visual file with them, you create a combined file. Note that you will have to update the file paths for the STL files associated with the links. You can choose to manage different folder locations but it is perhaps easier to copy all of the block collisional parts to the *meshes* folder of the visual model. Just make sure the file names are different! Once you do this you should be able to open your robot in Gazebo and see if your links are in the right place. If you see that your collision models are not in the same place as the visual models, there may be an error with your origins in the Solidworks model. To make your life easier, you should try to place the origins of the visual and the collision model in the same place, as this will guarantee that they come in in the right spot.

Design Practices:

When it comes to design methodologies, there are two basic types. The most common one is Bottom-Up design, which is where you build subsystems and then piece them together to create a more functional system. If you were going to build a robot to pick up a ball using the Bottom-up method, you would probably break into subteams and one group would build a drive train and another would build a gripper and arm mechanism. Then the teams would get together and integrate their designs. A different method of design is called the Top-Down strategy. A Top-Down strategy looks at the challenge as a whole, and then breaks it down into subcomponents, often using "black-boxes" and other fabrications that eliminate the need for focusing in on fine details. If you were to build the same robot using this strategy, there would be more planning involved and more communication between the groups before and during build, but then integration would go much faster because there was an overall blueprint from the start that was flushed out afterwards. Each of these methods lends itself to the exportation process differently, and so we will address them differently as well.

Bottom-Up Design:

This style of design assumes that you have a model of your robot complete, or that you have acquired a robot model from another source. The first step in this design process is to clean up the robot model. To make the export process as easy as possible, you should generate your own Origins according to the joint rules mentioned above, in addition to axes. The usefulness of this step is that the exporter will pull the names of these axes from your Solidworks document and you can select them from the dropdown instead of having to select them by hand. After that initial setup it is just a matter of going through the export process, meaning selecting your link bodies, giving them names and origins, and then setting their parameters like limits. Another thing to remember is to group solidworks parts into subassemblies based on what parts move together and form *rigid bodies*. This will help in the export process because selecting link elements will consist of only 1 or 2 clicks instead of having to select every part individually!

After you have successfully exported your model, you need to bring it into Gazebo. This involves booting into Ubuntu and configuring your files. You must first create a model.config file to be placed in the root of the URDF package. A sample configuration file is shown below. For the full file, we recommend copying one of the configuration files from the existing models in you .gazebo folder (This is a folder in home, if you cannot see it press ctrl+h to show hidden folders) and modifying it accordingly.

```
<model>
    <name>[put your robot's name here]</name>
    <version>1.0</version>
    <sdf>[put the relative path to the URDF file from the model's main directory here, e.g. robots/therobot.urdf]</sdf>
    <description>[optional description, or blank]</description>
</model>
```

Once you have the file created, you need to place your URDF package (the entire folder) in the models folder. After that, it's just a matter of running Gazebo and selecting your model from the list of available models.

Top-Down Design:

Top-down design is when parts are defined in the context of the assembly. It is a method of defining a skeleton of the model and making it more detailed from there. The most common way of doing this in Solidworks is using blocks. Blocks are sketch elements that move as rigid bodies. They can be edited, saved, resized, assigned mass properties and made into parts.

Known issues and Drawbacks

Blocks can be immensely helpful in prototyping a design there are however some drawbacks to using them in this context: In order to be made into part the blocks must be made in what Solidworks calls a *layout sketch.* Layout sketches can only be made on the front plane of a part or assembly there is no known setting to change this. All part and assemblies are limited to one layout sketch. If you insert a layout driven subassembly in a larger assembly the subassembly cannot be made flexible. What this means for robot modeling is that blocks can only be used to model your robot for all motion that can be captured in a single plane so if you have a claw with a gripper at the end or an arm that moves in 3 dimensions you cannot use blocks to design that. This severely limits the power of blocks for complex robot applications. They can still be used for prototyping sub systems independently or it can be used as a jumping off point for complex robots.

FAQ

Parts made from Blocks jump to incorrect positions

Rebuild the document. *Edit* >> *Rebuild***.** Or press Ctrl+B

Rebuild does not fix displaced parts

The cause of this is currently unknown. The apparent issue is that the blocks move and update the part-block Relation locking them in the new, incorrect position but the 3D solids themselves try to stay in the same place creating issue like orbiting wheels and other offset geometry. To fix it you must suppress the part made from the block and move the driving block in the layout sketch back to the correct position

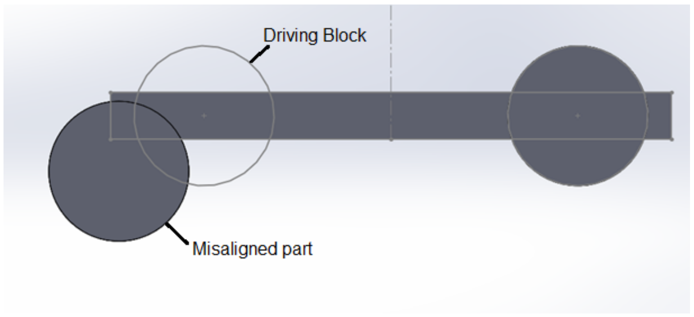


Figure 8: A Block driven wheel that has shifted positions

Another instance:

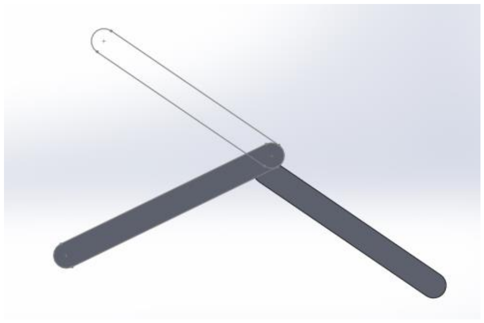


Figure 9: A block driven arm that has shifted positions

Model disappears when rotating view

Fit the model to view.
This icon:

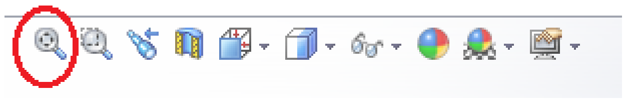


Figure 10: image showing the toolbar where Fit View can be found

Or press the f key

Getting Started

This is a step by step procedure to create a block driven tricycle robot

You may want to start by showing the *Blocks* toolbar; this will help you manipulate the blocks you make.

You can find it with the other toolbars in View > Toolbars > Blocks

1. Open a new Assembly .
2. Close the Insert *Components Command* property window.
3. Navigate to the *Layout* tab and select *Create layout.*

   Note: The layout is always created on the front plane. There is no way to change this.
4. Create a *Center Rectangle* centered at the origin and Dimension the Rectangle as shown:

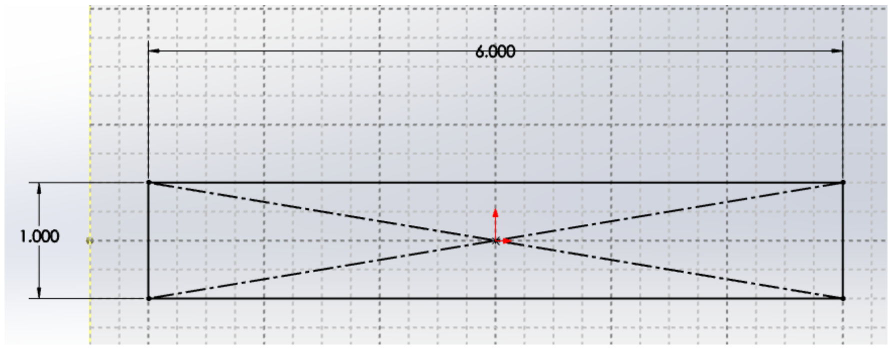


Figure 11: A center rectangle: 6" by 1" to be used as the robot chassis

5. Create 2 circles Horizontal with the origin, 1” away from the outside edges of the rectangle, and 3” in diameter, shown below with relations.

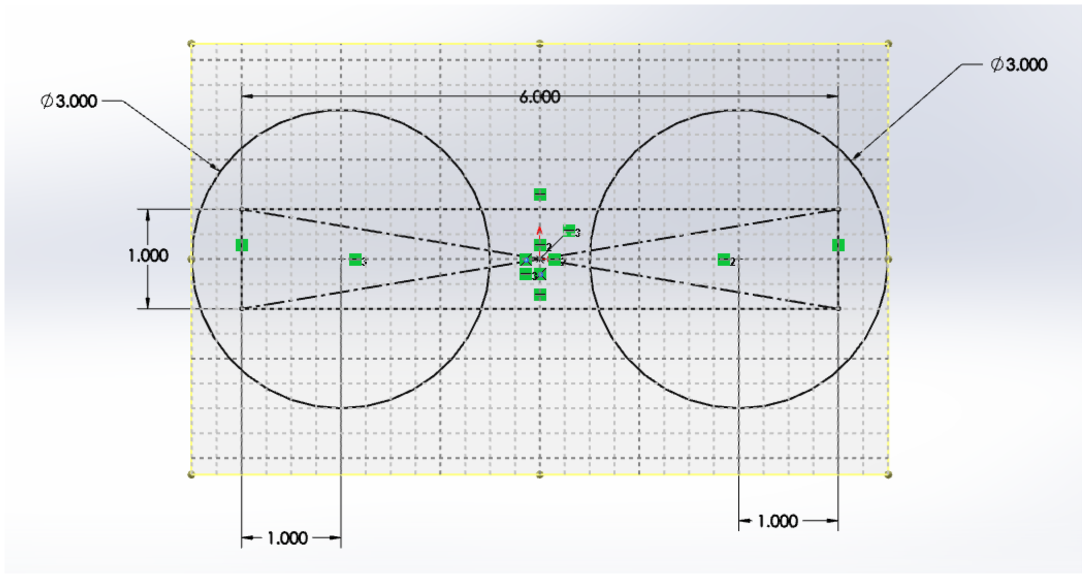


Figure 12: 2 3" Diameter circles added to the sketch, these will be the wheels

6. Select the line segments of the rectangle by right-clicking on one of the segments and clicking *Select Chain.*
7. Click *Make Block* from either the *Block* toolbar or *Layout* pane.
8. Click the *Green Check Mark.*
9. The rectangle should now appear light gray, and the dimensions associated with it will be unsolvable or conflicted.

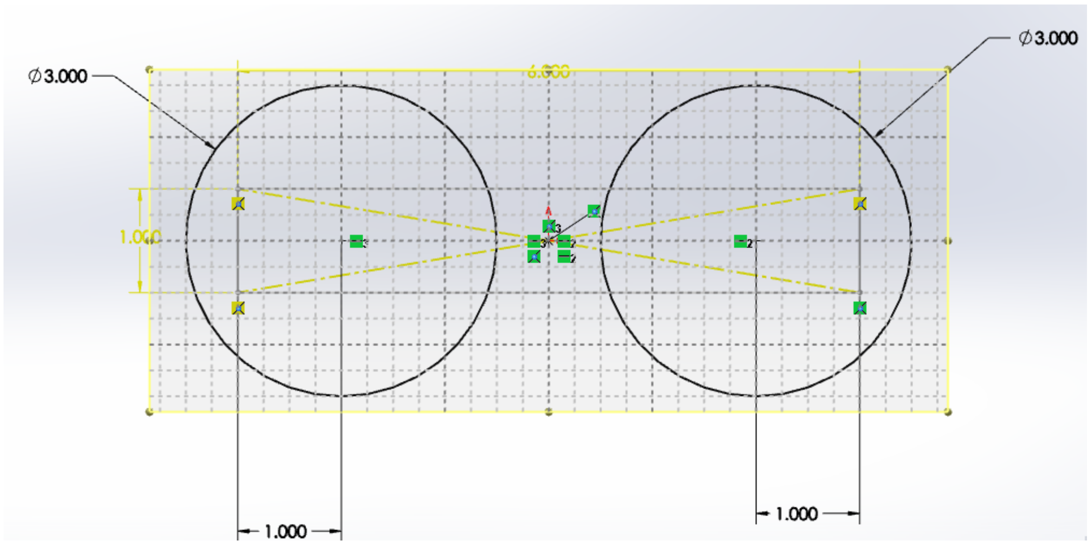


Figure 13: A figure showing the redundant dimmnsions

10. Delete the dimensions in question.

Note: If all dimensions appear as normal exit the layout sketch and enter again by using the layout button on the *layout* Pane

11. Add a Horizontal Constraint to the bottom edge of the rectangle.
12. Make a block out of each circle following steps 9-12 for both of them.
13. The resultant sketch should have 3 blocks and look like this:

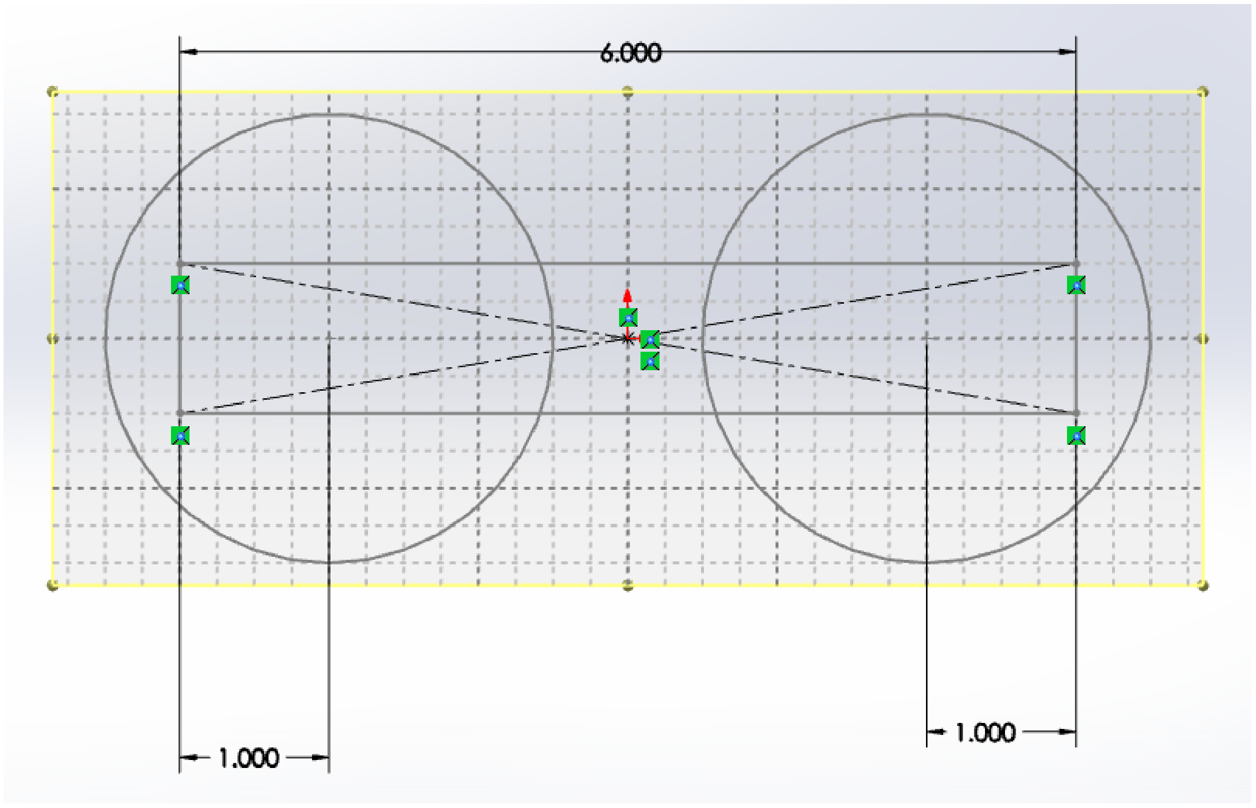


Figure 14: A figure showing the 3 blocks

Note: the 6" dimension is for the centerlines they can be deleted as they add nothing to the sketch.

14. Right-Click Block1-1 from the *Assembly Tree,* Choose 'properties' and change the name to "Chassis".
15. Click *OK.*
16. Change the name of the left wheel to "Rear Wheel" and the right wheel to "Front Wheel".
17. Select the Chassis Block and click *Make Part from Block* from the *Layout* pane.
18. Make sure *On Block* is selected under *Block to Part Constraint.*
19. Click the *Green Check Mark.*
    Notice a new part has been added to the assembly.
20. Edit the newly made part in the assembly
21. Expand the part and select Sketch1.
22. Click *Extrude Boss/Base* from the *Features* panel.
23. Set the *End Condition* to *Mid plane* and the *Depth* to 4.

24. Click the *Green Check Mark.*
25. On the top plane of the chassis solid, sketch the following rectangle.

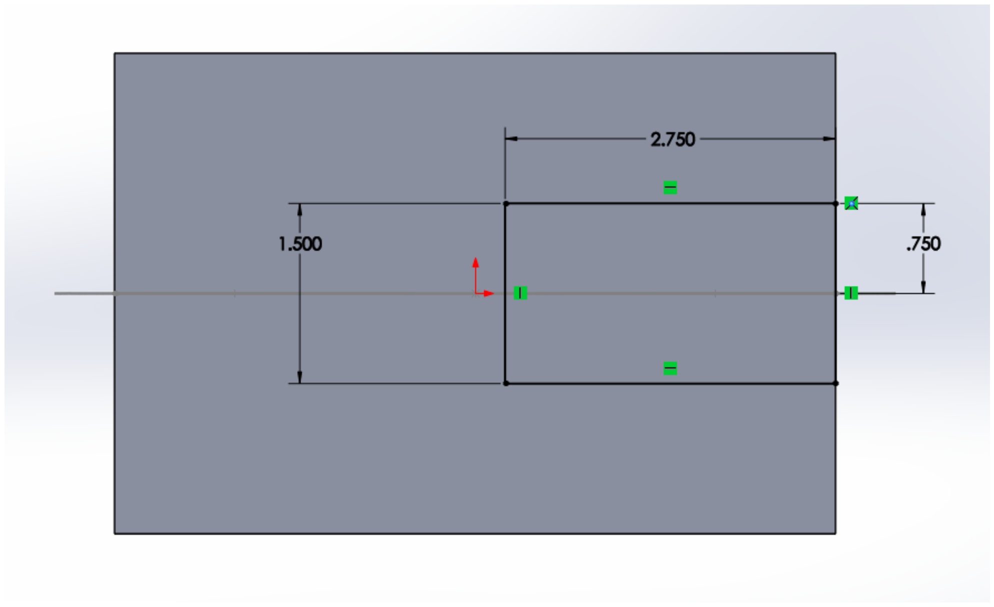


Figure 15: Top view of chassis with new sketch added

26. Select the Rear Wheel Block and click *Make Part from Block* from the *Layout* pane.
27. Select *Project* as *Block to Part Constraint.*
28. Click the *Green Check Mark.*
29. Edit the new part
30. Expand the part and select Sketch1.
31. Click *Extrude Boss/Base* from the *Features* panel.
32. Set the *Depth* to 1.
33. Click the *Green Check Mark.*
34. *Mate* the back face of the wheel to the front face of the chassis.
35. Hold Ctrl, Click and Drag the wheel part to make a copy of the wheel.
36. *Mate* the front face of the wheel to the back face of the chassis.
37. *Mate* the cylindrical face concentric with the cylindrical face of the first wheel.
38. Select the Rear Wheel Block and click *Make Part from Block* from the *Layout* pane.
39. Select *On Block* as *Block to Part Constraint.*
40. Click the *Green Check Mark.*
41. Edit the new part
42. Expand the part and select Sketch1.
43. Click *Extrude Boss/Base* from the *Features* panel.
44. Set the *Depth* to 1.
45. Click the *Green Check Mark.*

46. The finished model should look like this:

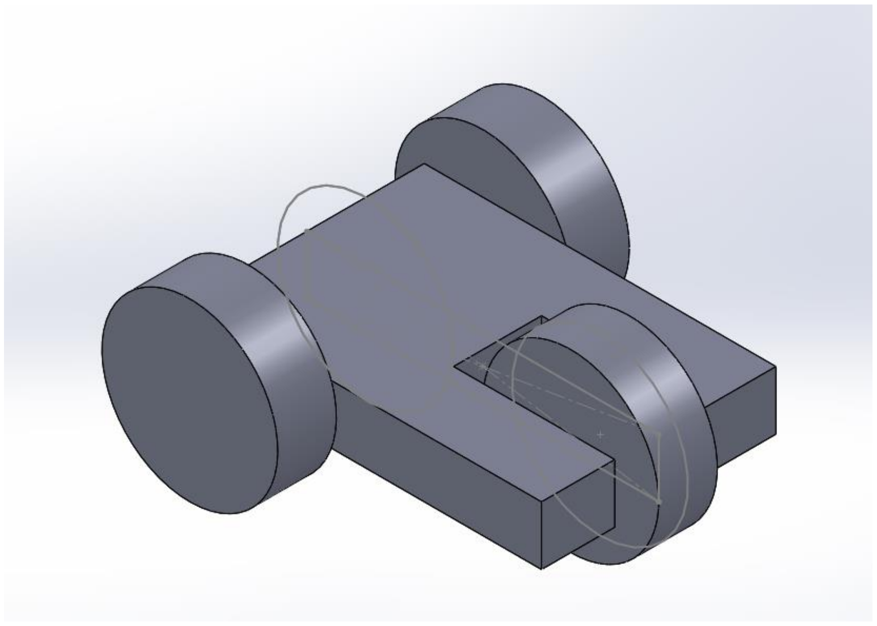


Figure 16: Finished model

47. To change the dimension of the blocks. You must enter the layout sketch, right click on the block in question and choose edit.
48. Edit the sketch of the front wheel and change the diameter to 2.75”
49. Edit the height of the chassis to be .75” and the width to be 8”
50. The model should now look like this:

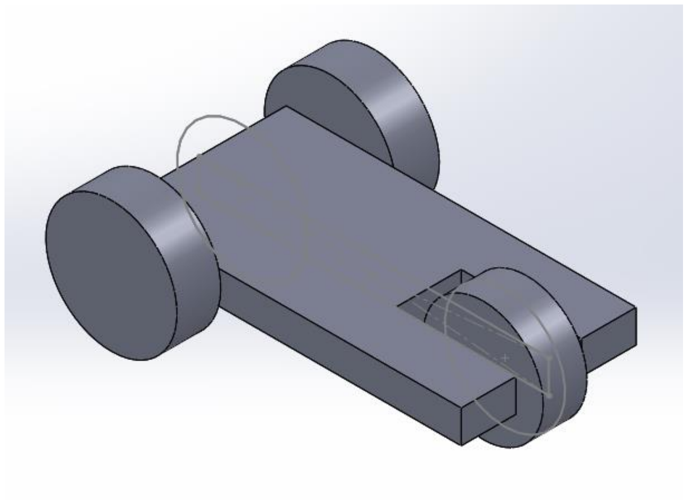
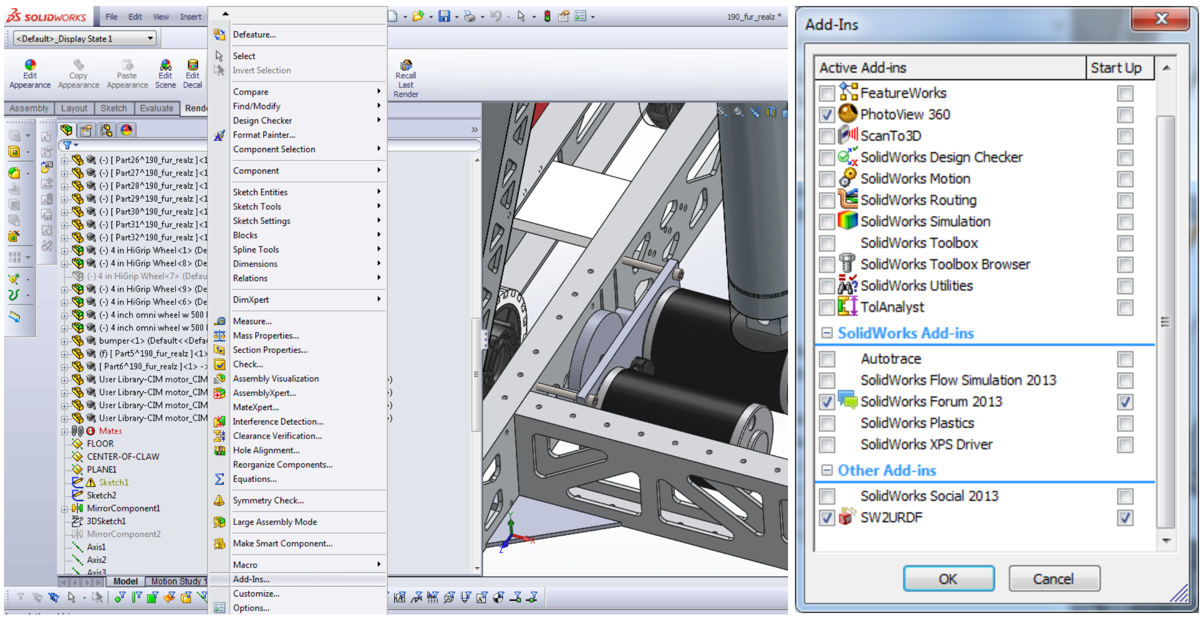


Figure 17: Model with some dimensions changed

This robot would not need a separate collisional model as these parts are very simple. However the point of blocks is to use them to prototype the skeleton of the robot and fill in the details later.

The Exporter

Once the exporter is installed make sure that it is enable by going to *tools* >> *add-ins*

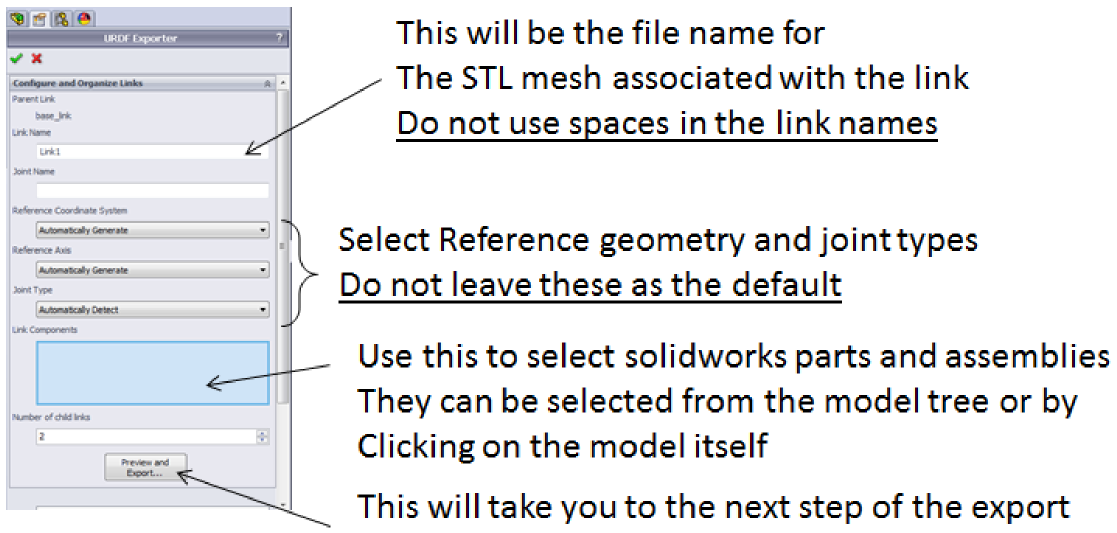


Add-Ins

Active Add-ins		Start Up
☐	FeatureWorks	☐
✓	PhotoView 360	☐
☐	ScanTo3D	☐
☐	SolidWorks Design Checker	☐
☐	SolidWorks Motion	☐
☐	SolidWorks Routing	☐
☐	SolidWorks Simulation	☐
☐	SolidWorks Toolbox	☐
☐	SolidWorks Toolbox Browser	☐
☐	SolidWorks Utilities	☐
☐	TolAnalyst	☐
SolidWorks Add-ins		
☐	Autotrace	☐
☐	SolidWorks Flow Simulation 2013	☐
✓	SolidWorks Forum 2013	✓
☐	SolidWorks Plastics	☐
☐	SolidWorks XPS Driver	☐
Other Add-ins		
☐	SolidWorks Social 2013	☐
✓	SW2URDF	✓

Figure 18: Insure the exporter is enabled

Export as URDF is available from the file menu. There are three phases of the exporter the first part appears as a Property Manager window on the left side of the screen.



Parent Link: base_link
Link Name: Link1
Joint Name:
Reference Coordinate System: Automatically Generate
Reference Axis: Automatically Generate
Joint Type: Automatically Detect
Link Components:
Number of child links: 2

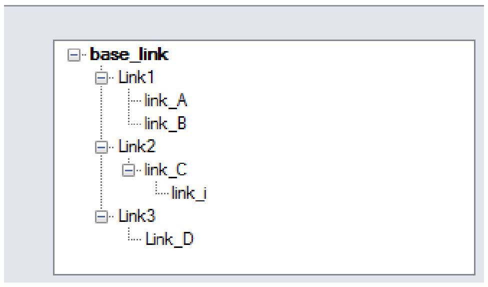
This will be the file name for
The STL mesh associated with the link
<u>Do not use spaces in the link names</u>

Select Reference geometry and joint types
<u>Do not leave these as the default</u>

Use this to select solidworks parts and assemblies
They can be selected from the model tree or by
Clicking on the model itself

This will take you to the next step of the export

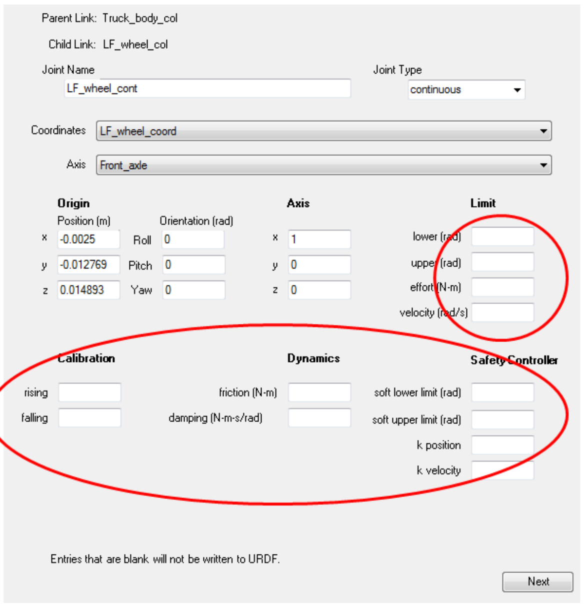
Figure 19: the features of the top half of the exporter



```
base_link
├─ Link1
│   ├─ link_A
│   └─ link_B
├─ Link2
│   └─ link_C
│       └─ link_i
└─ Link3
    └─ Link_D
```

Figure 20: The bottom part of the exporter shows the tree structure of the model

The second page of the exporter is the joint properties window where coordinate reference settings for the joints can be changed and joint limits applied.



Parent Link: Truck_body_col

Child Link: LF_wheel_col

Joint Name: LF_wheel_cont

Joint Type: continuous

Coordinates: LF_wheel_coord

Axis: Front_axle

Origin

	Position (m)		Orientation (rad)
x	-0.0025	Roll	0
y	-0.012769	Pitch	0
z	0.014893	Yaw	0

Axis

x	1
y	0
z	0

Limit

lower (rad)	
upper (rad)	
effort (N-m)	
velocity (rad/s)	

Calibration

rising	
falling	

Dynamics

friction (N-m)	
damping (N-m-s/rad)	

Safety Controller

soft lower limit (rad)	
soft upper limit (rad)	
k position	
k velocity	

Entries that are blank will not be written to URDF.

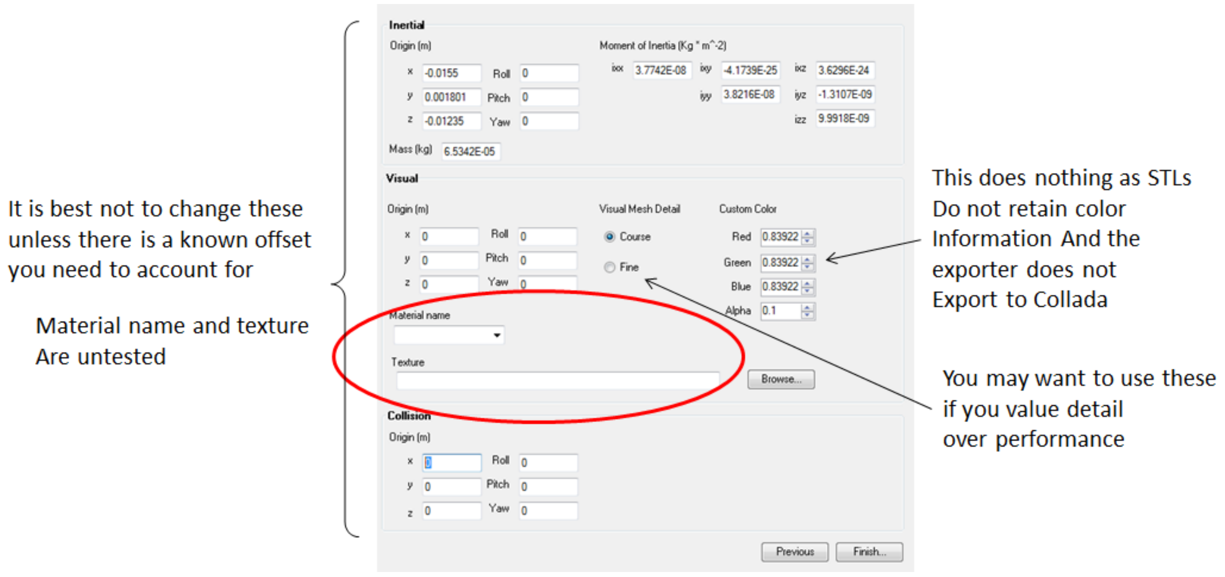
It is best not to change these unless there is a known offset you need to account for

Change the joint limits here

These are untested

Figure 21: The joint properties window

The third step of the exporter is the link properties tab. The most important feature of this is to check to see if the link has a proper inertia tensor in the top right-hand corner. If they are all 0 then you will have to quit and re-export your model. You may want to change the mesh from course to fine but this often results in a much more complex model for very little (if any) visual difference.



It is best not to change these unless there is a known offset you need to account for

Material name and texture Are untested

This does nothing as STLs Do not retain color Information And the exporter does not Export to Collada

You may want to use these if you value detail over performance

Inertial

Origin (m)

x -0.0155 Roll 0
y 0.001801 Pitch 0
z -0.01235 Yaw 0

Mass (kg) 6.5342E-05

Moment of Inertia (Kg * m^-2)

ixx 3.7742E-08 ixy -4.1739E-25 ixz 3.6296E-24
iyy 3.8216E-08 iyz -1.3107E-09
izz 9.9918E-09

Visual

Origin (m)

x 0 Roll 0
y 0 Pitch 0
z 0 Yaw 0

Visual Mesh Detail: [✓ Course] Fine

Custom Color

Red 0.83922
Green 0.83922
Blue 0.83922
Alpha 0.1

Material name

Texture

Collision

Origin (m)

x 0 Roll 0
y 0 Pitch 0
z 0 Yaw 0

Figure 22: The link Properties window

Sample export procedure

This is a sample procedure for exporting a robot, in this case a LEGO truck

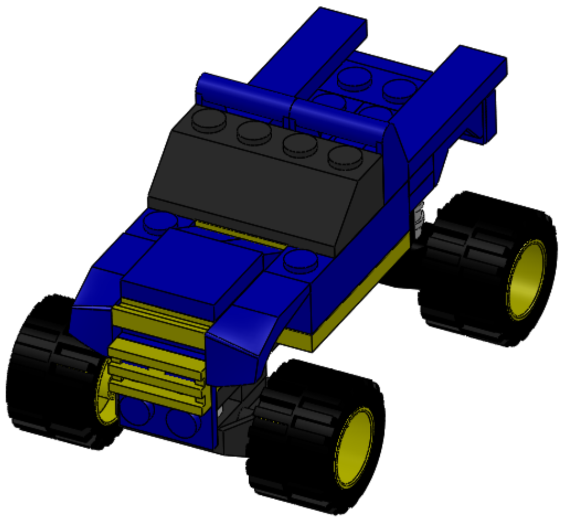


Figure 23: LEGO Car model

The collisional models for the wheels are cylinders. You can choose to make external parts to use as collisional models or define them in the assembly. Here we have elected to do it in the assembly.

We've decided that I need a good level of detail in the collisional model but not as much as the visual model is; a rectangular block will not do. Instead I have chosen to do a rough outline of the truck with particular detail to the front back and underside of the truck because we are concerned with ground clearance for this particular model. We then simply extruded the outline to make the collisional model. We usually set the collisional model to air as a material just as a fast way to make it transparent.

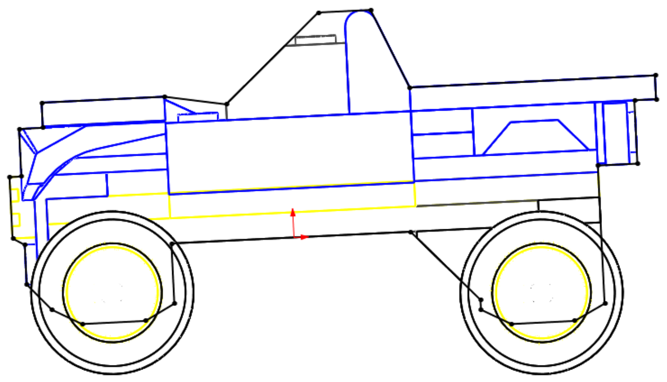


Figure 24: Side view showing the sketch of the collisional model

Next up is coordinate systems and axes

We placed a coordinate system for each wheel at the assembly level using the origin of the wheel assembly which I designed to be in the correct place for this. The exact placement does not matter as long at the coordinate system is on the axis that you will be using for the joint. It can be anywhere along the axis. If you are not fortunate to have an existing point or origin where you want a coordinate system you can always sketch a point using the center point of the arc of the wheel. When placing any coordinate system you may have to use other geometry to the axis pointing in the right direction. This is less important for axisymmetric joints like a wheel but for other rotating objects make sure the coordinate systems for the visual and collisional model are aligned. Or better yet use the same coordinate system for both.

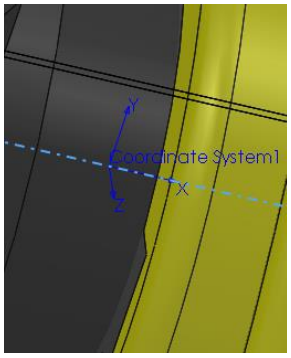
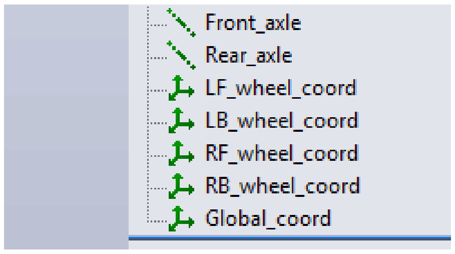


Figure 25: Coordinate system is along axis

It is a good plan to label your reference geometry


Front_axle
Rear_axle
LF_wheel_coord
LB_wheel_coord
RF_wheel_coord
RB_wheel_coord
Global_coord

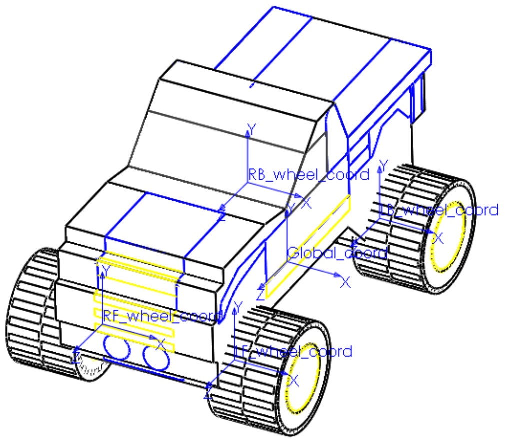
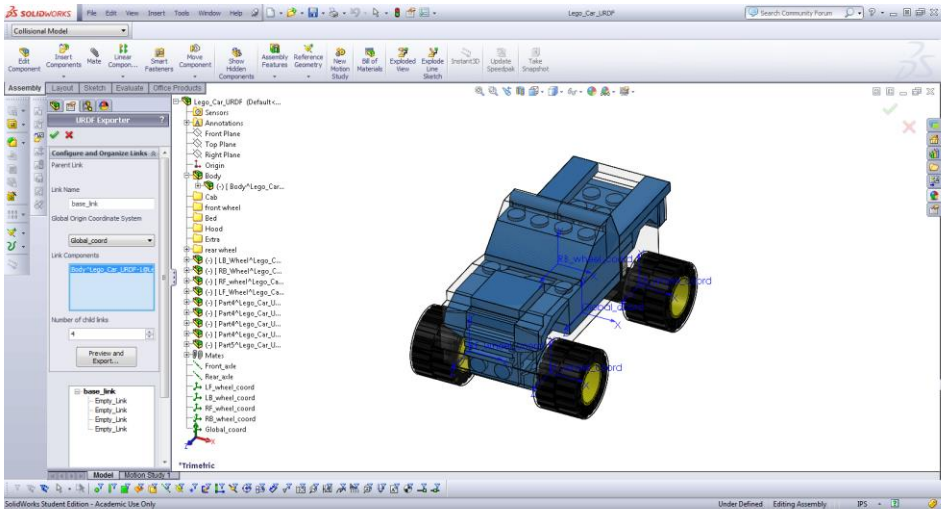


Figure 26: all coordinates are labeled

Rename a feature by triple clicking it in the model tree or right-clicking and selecting feature properties. If you rename a feature, and it does not update in the model window press Ctrl+B to rebuild the model and they will update.

Save your assembly

Open the exporter and select the truck body from the design tree under *Link Components*


URDF Exporter

Configure and Organize Links

Parent Link

Link Name: base_link

Global Origin Coordinate System: Global_coord

Link Components: Body^Lego_Car_URDF-1@Le

Number of child links: 4

Preview and Export...

base_link
- Empty_Link
- Empty_Link
- Empty_Link
- Empty_Link

SolidWorks Student Edition - Academic Use Only

Figure 27: Exporter and Truck. The collisional model is the transparent body around the model

Rename the link to “Truck_body”

Select the global coordinate system under *Global Origin Coordinate System*

Change the number of child links to 4

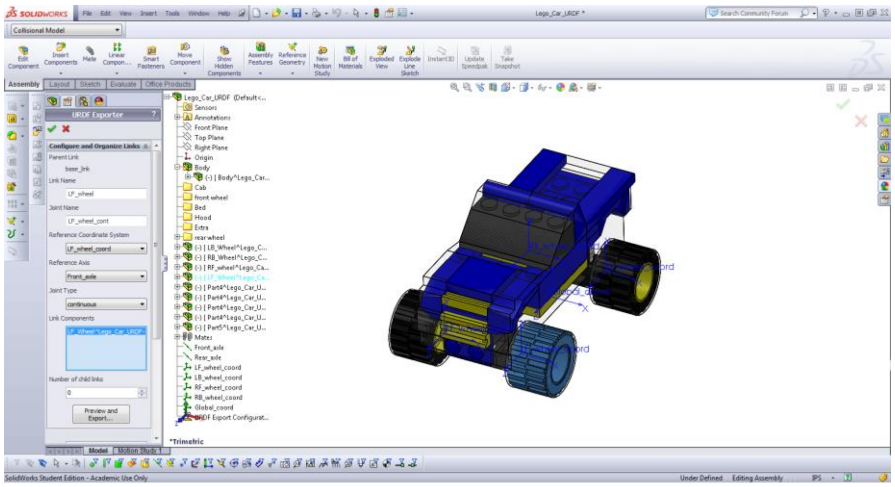
Select the first empty link and name it “LR_wheel”

Type “LR_wheel_cont” as the *Joint Name*

Set the *Reference Coordinate System* as “LF_wheel_coord”

Set the *Reference Axis* as “Front Axle”

Set the joint type to *continuous*


URDF Exporter

Configure and Organize Links

Parent Link: base_link

Link Name: LF_wheel

Joint Name: LF_wheel_cont

Reference Coordinate System: LF_wheel_coord

Reference Axis: Front_axle

Joint Type: continuous

Link Components: LF_Wheel^Lego_Car_URDF-

Number of child links: 0

Preview and Export...

SolidWorks Student Edition - Academic Use Only

Figure 28: The exporter with all parameters for the wheel set

Repeat these steps for the other 3 wheels

Click *Preview and Export*

Because all of our joints are continuous nothing needs to be specified under *Limit* but it might be an idea to put some reasonable values in there so your robot doesn't attempt to move at Mach 1. 100rad/s is about 1000RPM. I don't see this LEGO car needing to go that fast.

Click *Next*

Click through all the links to make sure they have non-zero numbers in the boxes under *Moment of Inertia.* A few zeros are fine but they should not all be zero. If they are you must close the export form and click *Preview and Export* again

Click Finish; select a file name and directory for your Model; Click Save then OK

Browse to the file directory and check to make sure the STL files have been written properly. If an STL file is 1MB it is likely empty. The one exception is that single feature rectangles can be 1MB also. You can double check by bringing the STL file into Solidworks. File > open or simply dragging it in work.

If there are any missing STLs re-export the same model as is and save if in a new folder. Do not bother specifying joint limits once this process is complete you will merely copy the STLs to the old folder. Expect that STLs that worked last time will break this time so only copy full ones to the folder. Repeat until all STLs are full. Expect to export once for every 5 STL files in this case there are 10 total (including the collisional model) so expect to export at least twice. (in fact as I was doing this I had to export twice just for the visual model)

Now, back in Solidworks save a copy of your model with "_col" or similar in the file name.

Open up the new assembly and click Export to URDF.

All the link information will be saved and since all the coordinate systems and axes are the same we only have to change the link components.

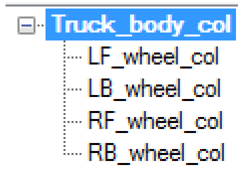
Clear the existing geometry from the list and select the corresponding collisional model. A good tool for selecting the collisional models is the *select other* tool which allows you to select from a list of all geometry 'under' your curser. This is found at the top of the right click menu.

As you do this add "_col" to all the link names. This will change the name of the STL so the collisional models and visual models can reside in the same folder.

Click *Preview and Export*

There is no need to specify the joint limits for this export we only care about the STL files which do not retain that information.

Your export tree should look like this:

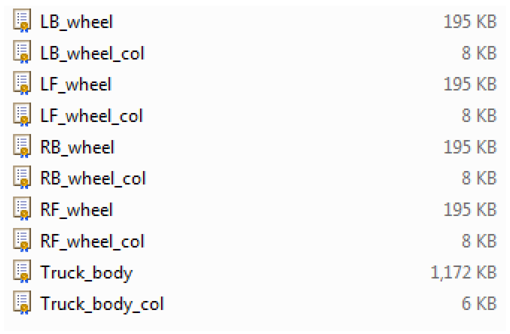


```
Truck_body_col
    LF_wheel_col
    LB_wheel_col
    RF_wheel_col
    RB_wheel_col
```

Figure 29: The tree structure of the assembly

Save the new files near your previous export under a new name

Copy the STLs out and into the *meshes* folder of the visual model

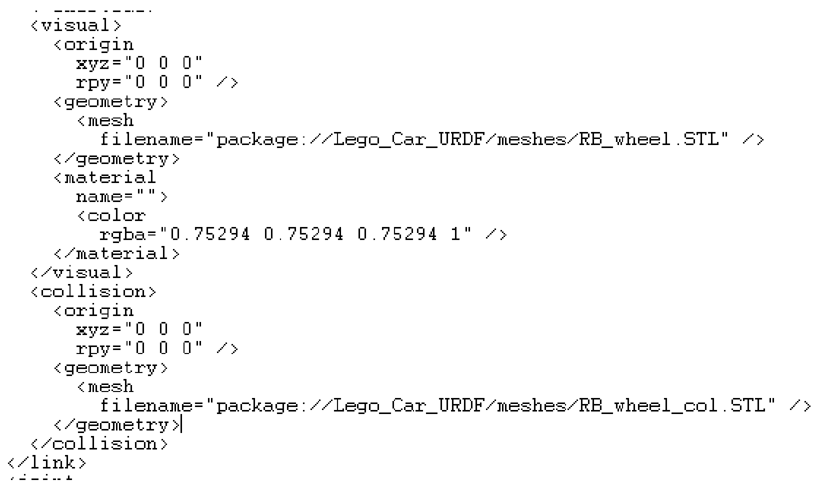


LB_wheel	195 KB
LB_wheel_col	8 KB
LF_wheel	195 KB
LF_wheel_col	8 KB
RB_wheel	195 KB
RB_wheel_col	8 KB
RF_wheel	195 KB
RF_wheel_col	8 KB
Truck_body	1,172 KB
Truck_body_col	6 KB

Figure 30: A picture showing the relative sizes of the collisional and visual STL files

Now navigate to the *robots* folder and open the text file inside

Add "_col" to the *mesh filename* tag under collision (always the second occurrence)



```
    <visual>
      <origin
        xyz="0 0 0"
        rpy="0 0 0" />
      <geometry>
        <mesh
          filename="package://Lego_Car_URDF/meshes/RB_wheel.STL" />
      </geometry>
      <material
        name="">
        <color
          rgba="0.75294 0.75294 0.75294 1" />
      </material>
    </visual>
    <collision>
      <origin
        xyz="0 0 0"
        rpy="0 0 0" />
      <geometry>
        <mesh
          filename="package://Lego_Car_URDF/meshes/RB_wheel_col.STL" />
      </geometry>
    </collision>
  </link>
```

To Linux!

Open the Manifest file and copy the text out of it.

Open up the first folder and create a new file called model.config (not a text file) open it in the text editor of your choice and type

```
<version>1.0</version>
<name>filename.URDF</name>
<description>your description here</description>
<!--
```

Then paste the text you coped from the manifest file and finish it off with

```
 -->
```

The finished text should look like this:

```
<version>1.0</version>
<name>filename.URDF</name>
<description>your description here</description>
<!-- <package>
      <description brief="filename">filename</description>
      <depend package="gazebo"/>
      <author>your_name</author>
      <license>BSD</license>
</package> -->
```

With filename replaced with the name of your .URDF file
Now your model is ready for Gazebo!

Move/Copy the folder to the robots folder in .Gazebo under user. (It is a hidden folder so you will need to hit Ctrl+H to find it.

Run gazebo.

Find your model name in the menu on the right side and click to select it. Click the pause button at the bottom of the screen then click in the environment to place it! (it will be sideways)

If you are interested in connecting your robot model to FIRST Robotics Competition WPILib Java code, you can see the more intricate details, including how to make a world file, in the following Youtube tutorial video: https://www.youtube.com/watch?v=DCNpT1ng9sA&feature=youtu.be

Current Exporter Tips and Tricks

Some helpful hint for using the current SW2URDF format

If you need to edit an assembly referenced by the exporter it is best to open the export configuration window and disconnect it from the link. Editing it while the two are linked has been known to cause an error that corrupts the configuration file (even if the exporter is closed).

Name the joints between your links something that tells you which link they are for and what type of joint they are. For example the prismatic joint for the shooter might be “shooter_pris” or “shooter_p”

Every so often you should exit the exporter configuration manager using the green checkmark the be sure the configuration is saved. An error (currently caused by doubling clicking the model window) will cause the exporter to close and not save changes

Make sure there are NEVER two exporter dialog boxes open at a time. There is one bug that causes the export to fail but the exporter merely moves behind Solidworks opening another dialog box can cause configuration data to be written to the wrong file, which is unrecoverable to our knowledge. Sometimes when this happens it will prevent you from ever begin able to export to URDF form that assembly again. file > save as will not fix this because the configuration stays with the new file.

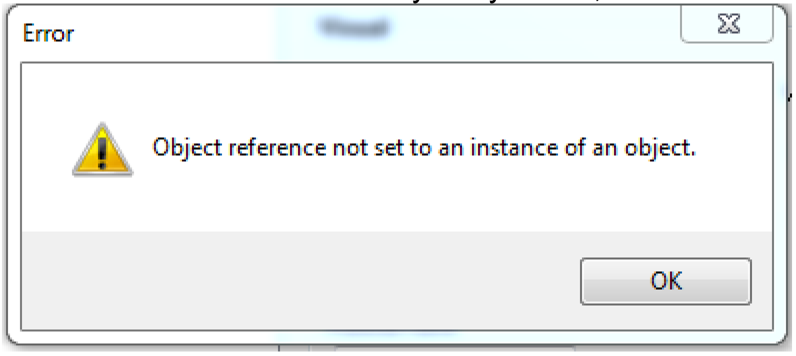
If a link appears in the second exporter dialog box with no inertia tensor (all zeroes in the upper right hand corner) exit the exporter and select the link in question before hitting the “preview and export” button

Some errors such as an empty link will prevent the exporter from closing using the cancel button. The x button still works.

To manage your collisional and visual models within the same assembly it is a good idea to turn the collisional parts transparent so you can see both them and the other components you are working on. If you have many parts you may want to create display states, one for the visual model and one for the collisional model.

Known Bugs (and possible solutions)

- Every once in a while during export, the plugin leaves the last link empty, which results in The exporter hanging up during the export process. The symptoms of this are that the link will have no mass information on the second page of the exporter form. If this happens the exporter will not let you cancel with the cancel button. you must click the 'x' when you try to exit, this error will appear


Error

Object reference not set to an instance of an object.

To fix this exit out of the exporter and open it back up again. Click on the link that failed last time and make sure it is selected before you click *preview and export*. This will decrease failure rate to 1%

- If *Automatically Generate* is ever chosen as the option for axes or coordinate systems the exporter will hang and will not generate any features.
    - The same happens for *none*
- Even though the fixed joint does not need a reference axis, something must be selected in *reference Axis* otherwise the exporter will hang. The exporter will delete whatever is selected so with every new export, this will have to be re-set.
    - sometimes if another error cause the plugin to crash this information will be retained. This is rare.
- Color function does not do anything. STL does not retain color information. in addition the Up/Down buttons don't modify the number in the Color boxes
- Double clicking on geometry in the model window causes the exporter command manager to close without saving. does not occur for double clicking things in the model tree
- Very often, and for unknown reasons there is an issue writing the stls. Expect to export once for every 4-5 STLs (both visual and collisional) (ie 30 STLs = 6-8 exports)
- Sometimes when old geometry is deselected and new components are added (ie when moving from visual to collisional models) to *Link components* the old geometry is exported instead. one was to fix this is to deselect and reselect the desired geometry and suppress all other geometry.
- Suppressed components in subassemblies are still exported... sometimes.
    - Selecting suppressed components outright has been known to cause the exporter to hang however
- Sometimes If a assembly referenced as a link is edited (has parts removed from it) then the exporter looses connection with the joint associated with it. clears The solidworks component from the exporter configuration and then assumes every other link is that link (usually the base link is first) Every time you click on a different link it will rename that link to the name of the broken one even if the exporter tree does not

show that this is happening. only known fix is to decrease the number of child links in the parent of the problem link until it deletes it from the tree.

- The exporter does not remember anything past the *preview and export* button the one exception being that any name changes of the joints and links will be saved to the command manager part of the plugin.