

# Arène pour Turtlebot - ROS

Ce projet a pour but d'améliorer l'arène pour turtlebot utilisée par les étudiants du Master Sciences de l'Ingénieur qui suivent l'unité d'enseignement de ROS. Cette arène consistera en un circuit urbain composé de différents obstacles tels que des feux de signalisation et des barrières automatiques ... Robin Lévêque Encadré par Sylvain Argentieri et Fabien Vérité

- [Simulation](#)
  - [Gazebo - Introduction](#)
  - [Gazebo - Méthode cinématique](#)
  - [Circuit sur TinkerCad](#)
  - [Gazebo - Méthode Actionneurs](#)
- [Realisation des pièces](#)
  - [Barrière automatique](#)
  - [Support pour feux de signalisation](#)
  - [Boitier contenant le circuit de l'arène](#)
  - [Support pour caméra raspicam rev2.2](#)

# Simulation

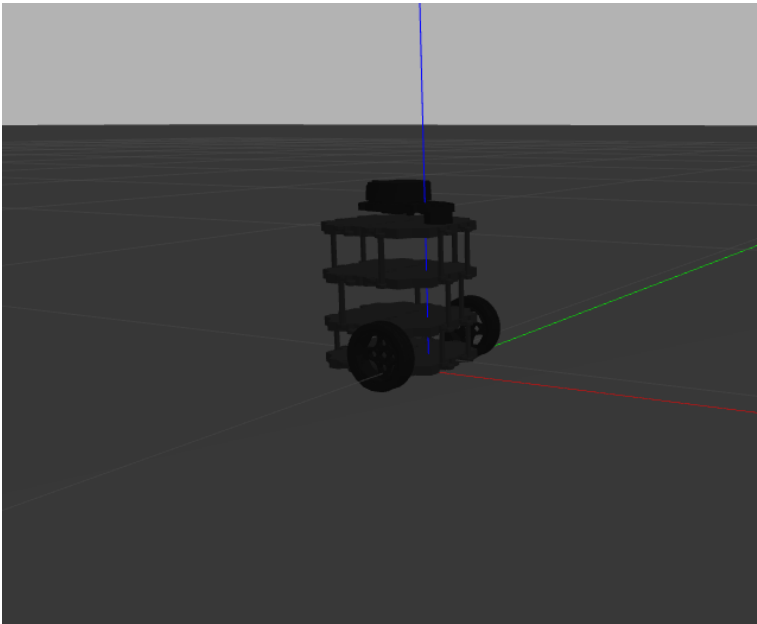
Avant chaque réalisation il faut pouvoir tester grâce à des simulations, pour ce projet on utilisera Gazebo pour la partie simulation 3D des turtlebots dans leur environnement, on utilisera aussi Tinkercad pour simuler le circuit électronique de l'arène.

# Gazebo - Introduction

Dans ce projet il sera question de nombreux outils de robotique notamment ROS et Gazebo, si vous souhaitez reproduire ce projet il est fortement conseillé de suivre les tutoriels ROS :

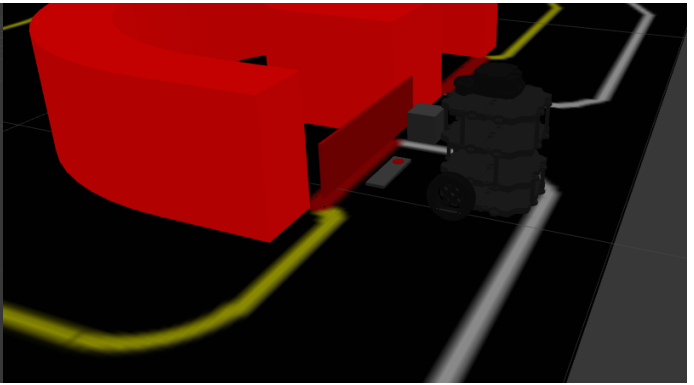
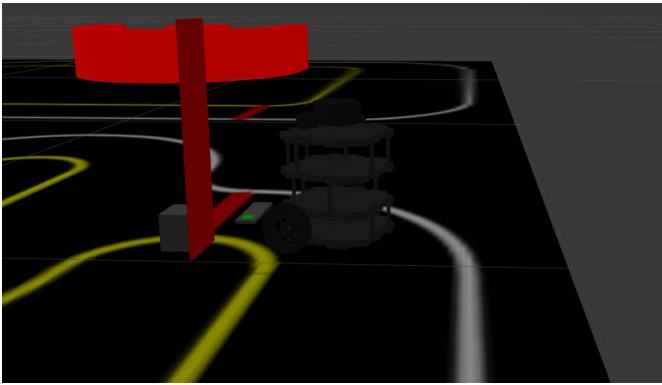
<http://wiki.ros.org/Documentation>

Nous allons simuler le robot turtlebot dans un environnement le plus réaliste possible. Pour cela, nous utiliserons Gazebo:



Dans cet environnement, notre turtlebot va rencontrer les obstacles auxquels il fera face ensuite hors de la simulation:

- Des lignes à suivre
- Un rond-point à contourner
- Une barrière et un feu pour contrôler le trafic
- Un corridor fermé par une barrière



# Gazebo - Méthode cinématique

Il nous faut une méthode pour contrôler le circuit et tous ces modules pour cela nous allons appliquer une vitesse sur l'articulation de la barrière et sur les articulations des feux.

Pour la barrière on utilise une liaison de type *revolute* avec des limites dans sa rotation, ainsi, elle ne peut tourner que de 90° (position haute et basse). On ajoute aussi un peu de friction pour qu'elle ne tombe pas ou ne rebondisse pas, ce qui est une limite de la simulation.

```
<joint name="base_to_barrier" type="revolute">
  <parent link="base_link"/>
  <child link="barrier"/>
  <origin xyz="-0.025 0.025 0.0" rpy="1.57 0 0"/>
  <axis xyz="0 0 1"/>
  <limit effort="1000" lower="-1.57" upper="0" velocity="1.0"/>
  <dynamics damping="0.0" friction="0.1"/>
</joint>
```

Pour ce qui est des feux on va avoir 3 disques Rouge Jaune et Vert qui utilisent une liaison de type *prismatic* avec le même genre de limites que pour la barrière, ainsi, chaque disque de couleur apparaîtra ou disparaîtra selon le contrôle en vitesse appliqué.

```
<joint name="base_to_light_red" type="prismatic">
  <parent link="base_link"/>
  <child link="light_red"/>
  <origin xyz="-0.0333 0.0 0.0" rpy="0 0 0"/>
  <axis xyz="0 0 1"/>
  <limit lower="0.0" upper="0.002" effort="1000.0" velocity="1000.0"/>
  <dynamics damping="0.0" friction="0.1"/>
</joint>
```

Ensuite, il faut contrôler chacune de ces liaisons pour cela il existe un *topic* généré par Gazebo, */gazebo/link\_states*. On va publier sur ce *topic* l'état de notre objet, dans ce cas ce sera une vitesse:

```
pubState = rospy.Publisher('/gazebo/set_link_state', LinkState, queue_size=1)
```

Il faut faire une petite manipulation avant de pouvoir publier sur la topic car si on ne publie que la vitesse, la position de l'objet sera réinitialisée alors il faut récupérer l'ancienne position de l'objet, ce sera comme déplacer l'objet par petits pas.

```
# Get the current state of the barrier
current_state = rospy.wait_for_message('/gazebo/link_states', LinkStates)

# Get the index of the barrier in the list of links
ind = current_state.name.index(self.link_name)

# Create the message
msg = LinkState()
msg.link_name = self.link_name
msg.pose = current_state.pose[ind]

# Change the position of the barrier
msg.twist.angular.x = 6 * (-1)**self.state
msg.twist.angular.y = 6 * (-1)**(self.state+1)

# Update the state of the barrier
self.state = not self.state

# Update the state of the traffic light
self.traffic_light.change_state()

# Publish the message
pubState.publish(msg)
```

Le problème avec cette méthode c'est que selon l'orientation de la barrière il faut adapter l'axe de la vitesse angulaire appliquée ( x ou y ).

Codes complets utilisés:

Description du feu de signalisation: [traffic\\_light.urdf](#)

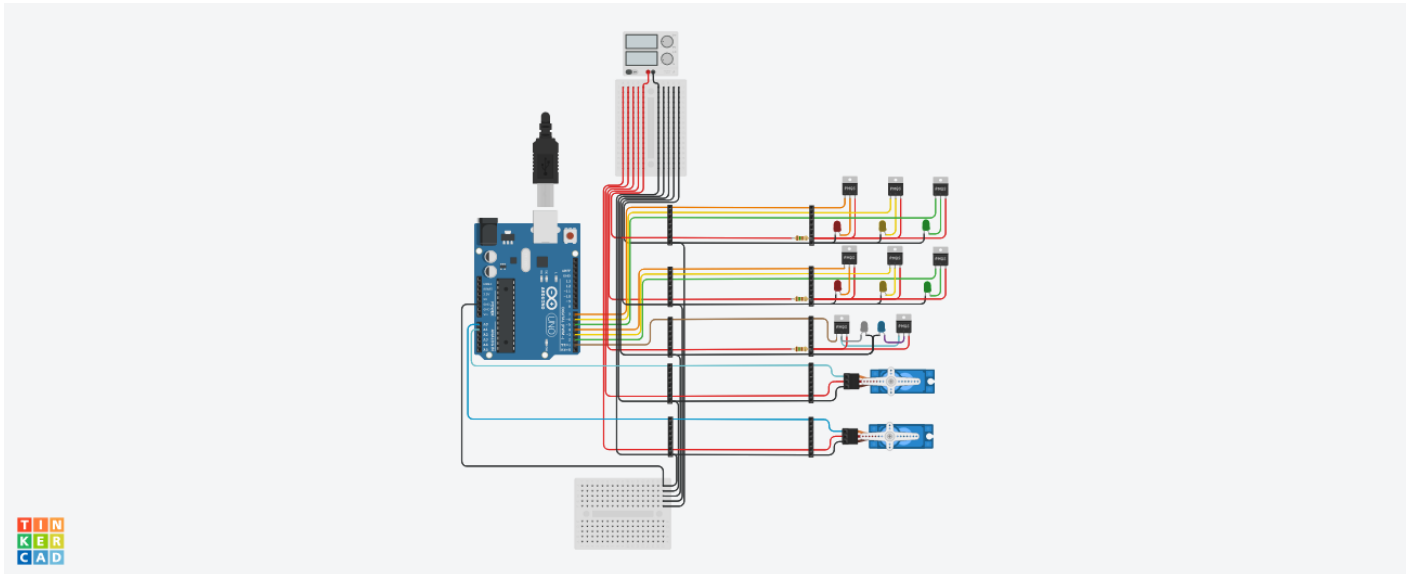
Description de la barrière: [barrier.urdf](#)

Monde Gazebo du circuit: [arene\\_2.world](#)

Script de contrôle des modules: [world\\_control.py](#)

# Circuit sur TinkerCad

Pour la réalisation de l'arène il faut créer le circuit électrique qui connectera les différents modules de l'arène à l'alimentation et à une arduino qui permet son contrôle.



Ce circuit permet de gérer 2 feux tricolores ainsi que 2 barrières automatiques, chaque barrière est assemblée dans le code avec un feu, ainsi, le feu passe au vert seulement lorsque la barrière est complètement ouverte. Il y a aussi un feu bicolore qui alterne entre les deux led allumées, cela nous permettra d'inclure un choix binaire dans l'arène.



# Gazebo - Méthode Actionneurs

Une autre méthode pour contrôler tous les modules est d'utiliser des plugins intégrés à Gazebo qui agiront comme des moteurs attachés aux liaisons.

Pour la barrière, en plus de la liaison de type *revolute* on appliquera le plugin *gazebo\_ros\_control*:

```
<gazebo>
  <plugin name="gazebo_ros_control" filename="libgazebo_ros_control.so">
    <robotNamespace>/barrier1</robotNamespace>
  </plugin>
</gazebo>

<transmission name="barrier1_transmission">
  <type>transmission_interface/SimpleTransmission</type>
  <actuator name="motor1">
    <mechanicalReduction>1</mechanicalReduction>
  </actuator>
  <joint name="base_to_barrier">
    <hardwareInterface>hardware_interface/EffortJointInterface</hardwareInterface>
  </joint>
</transmission>
```

Ensuite, il faut ajouter une configuration qui permettra à Gazebo d'interpréter ce moteur:

*config.yaml*

```
barrier_1:
  joint_1_state_controller:
    type: joint_state_controller/JointStateController
    publish_rate: 50

  gear_to_barrier_1_position_controller:
    type: effort_controllers/JointEffortController
```

```
joint: gear_to_barrier_1
```

Enfin, on pourra publier sur le topic généré par le plugin:

```
pub = rospy.Publisher('/barrier1/gear_to_barrier_1_position_controller/command', Float64, queue_size=10)
```

# Realisation des pièces

Dans un deuxième temps, il faut réaliser les différentes pièces qui vont composer notre arène.

# Barrière automatique



Contrôler notre barrière automatique, actuellement, nous utilisons un servo moteur pour montrer sa faisabilité.



Cette barrière de démonstration est trop petite et fragile nous allons donc utiliser une barrière plus robuste.

Cette nouvelle barrière fera 30 cm de long pour 8 cm de haut avec 3 mm d'épaisseur, le système d'attache au servo moteur restera le même.

[barriere.DXF](#)

La partie amovible de notre barrière est maintenant plus lourde il faut donc trouver un moyen de faire contre-poids sur le servo moteur pour qu'il ne bascule pas. Une solution pour ça est de faire un module autour du servo qui sera ensuite fixé sur le parcours pas aimantation.

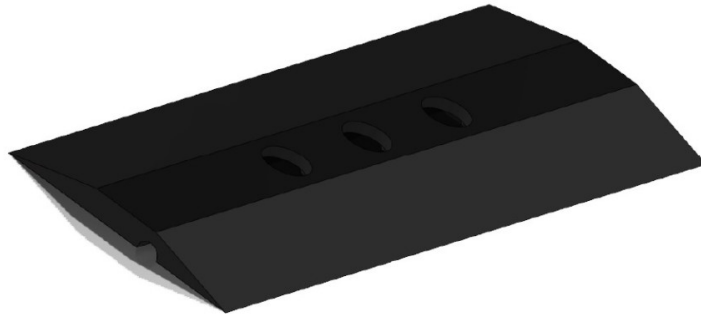
# Support pour feux de signalisation

Nous allons créer des feux de signalisation, ces feux tricolores indiqueront au turtlebot le comportement à avoir:

Un feu rouge signifie que la voie est bloquée par une barrière

Un feu vert signifie que la voie est libre

Pour la réalisation de ces feux nous les poseront sur le sol, la caméra des robots à un champ de vision dirigé vers le sol pour voir les lignes sur le sol ainsi il n'est pas capable voir des feux en hauteur comme on en voit dans la rue tous les jours. Ces feux seront alors sur la route mais il doivent être robustes pour ne pas être endommagés par les robots qui roulent dessus et aussi avoir suffisamment de place pour faire passer la connectique. On va donc utiliser un modèle de passe-câbles souvent utiliser pour les événements et utiliser pour faire passer des câbles à travers des voies où des véhicules passent.



### [Support\\_feux.STL](#)

Nous allons utiliser des LED à forte luminosité et les colorer avec un papier calque pour chaque orifice, ceci va aussi nous permettre de diffuser la lumière un peu plus efficacement et augmenter la visibilité pour les robots.

# Boitier contenant le circuit de l'arène

Nous utiliserons un boitier pour contenir tout le circuit de l'arène.

Il y aura donc la raspberry pi qui fournira un contrôle à une arduino et qui permettra le lien avec ROS et les turtlebots, il y aura aussi l'alimentation et le routage des cables vers de prises qui permettront de rattacher tous les modules au circuit.

Pour fabriquer ce boitier nous allons découper une plaque de bois de 6mm et nous utiliserons [MakerCase](#) pour obtenir le plan de découpe.

On va ensuite découper le fichier en six parties pour chaque face car la boite est trop grande pour être découpée en une seule fois. On aura alors une boite de dimensions 400x200x100 mm avec 6mm d'épaisseur de plaque.

[box\\_front.svg](#)

[box\\_back.svg](#)

[box\\_right.svg](#)

[box\\_left.svg](#)

[box\\_top.svg](#)

[box\\_bottom.svg](#)

Voici tout de même le fichier complet pour voir la boite dans son ensemble.

[box.svg](#)

Realisation des pièces

# Support pour caméra raspicam rev2.2