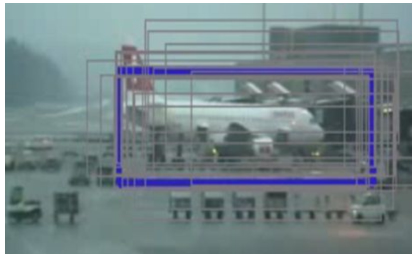
UNIVERSITÀ DEGLI STUDI DELLA BASILICATA

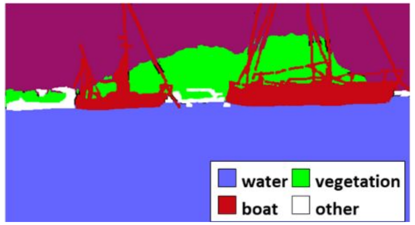
Corso di Visione e Percezione

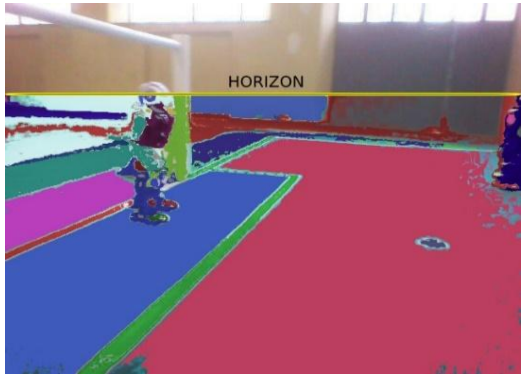
Docente

Domenico D. Bloisi

Robot mobili su ruote



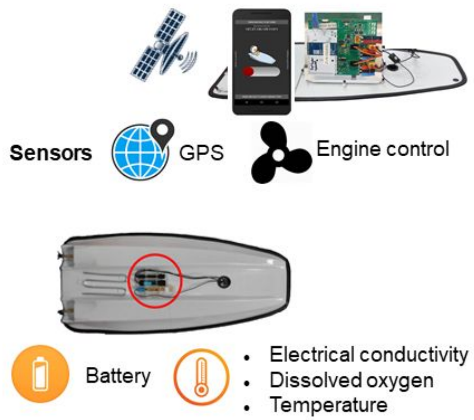


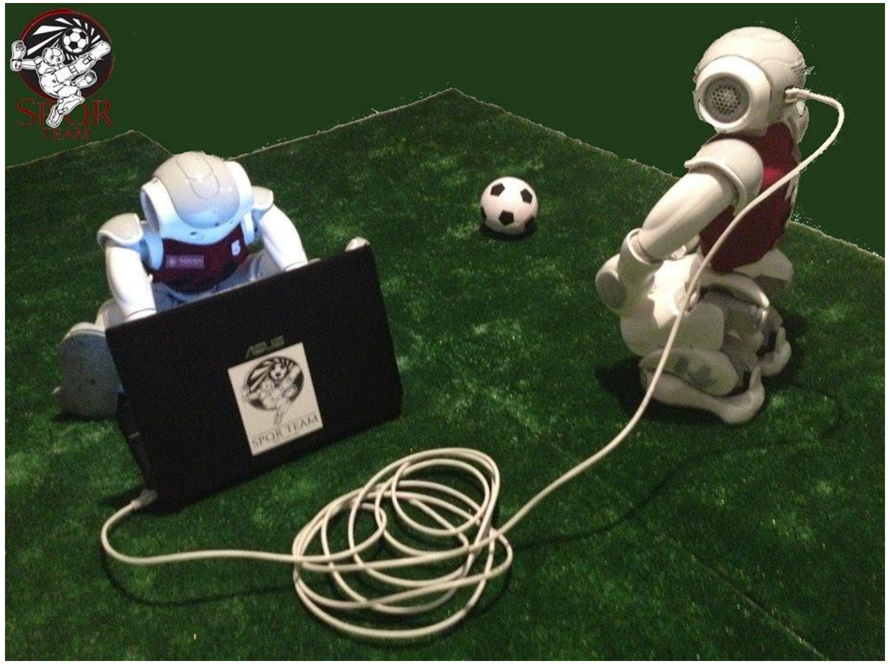


Domenico Daniele Bloisi

- Ricercatore RTD B
  Dipartimento di Matematica, Informatica ed Economia
  Università degli studi della Basilicata
  http://web.unibas.it/bloisi

- SPQR Robot Soccer Team
  Dipartimento di Informatica, Automatica e Gestionale Università degli studi di Roma “La Sapienza”
  http://spqr.diag.uniroma1.it





Informazioni sul corso

- Home page del corso
  http://web.unibas.it/bloisi/corsi/visione-e-percezione.html

- Docente: Domenico Daniele Bloisi

- Periodo: II semestre marzo 2021 – giugno 2021

  Martedì 17:00-19:00 (Aula COPERNICO)
  Mercoledì 8:30-10:30 (Aula COPERNICO)

Codice corso Google Classroom:
https://classroom.google.com/c/NjI2MjA4MzgzNDFa?cjc=xgolays

Ricevimento

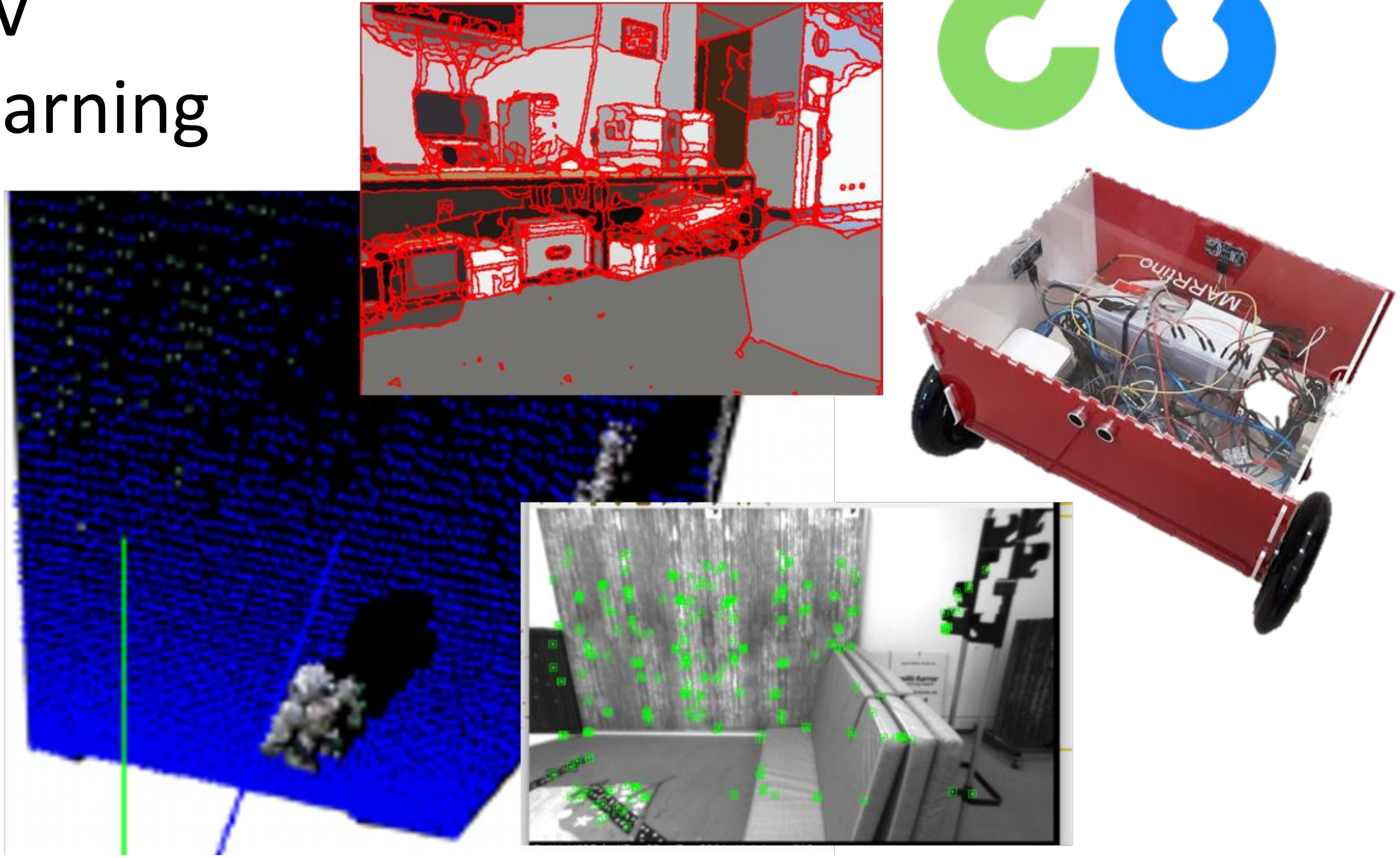
- Su appuntamento tramite Google Meet

Per prenotare un appuntamento inviare una email a

domenico.bloisi@unibas.it

Programma – Visione e Percezione

- Introduzione al linguaggio Python
- Elaborazione delle immagini con Python
- Percezione 2D – OpenCV
- Introduzione al Deep Learning
- ROS
- Il paradigma publisher and subscriber
- Simulatori
- Percezione 3D - PCL



Manipolatori vs robot mobili

- I bracci robotici sono ancorati al terreno e hanno, di solito, un'unica catena di giunti

- Il workspace di un manipolatore definisce il range (relativamente al punto di ancoraggio) delle possibili posizioni che possono essere raggiunte dagli end-effector del robot

https://www.youtube.com/watch?v=sWgvIAkfqXQ

Manipolatore - Position estimation

- Un manipolatore ha un'estremità ancorata ad un punto dell'ambiente

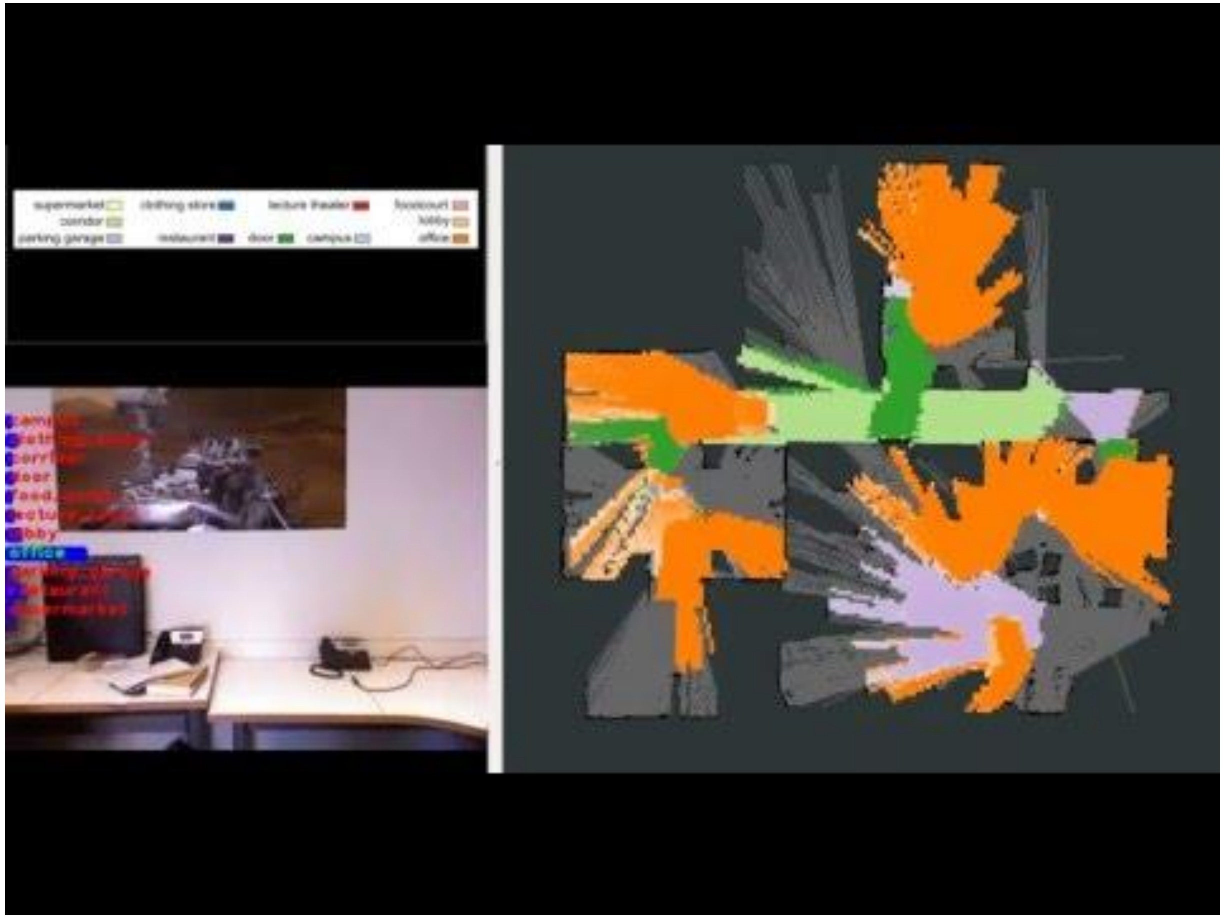
- Misurare la posizione dell'end-effector di un braccio richiede unicamente di conoscere la cinematica del robot e di misurare la posizione dei giunti intermedi

- La posizione di un manipolatore è sempre calcolabile avendo a disposizione i dati dei sensori

Robot mobili

- Il movimento di un robot mobile può essere definito attraverso i vincoli di **rotolamento** e **scivolamento** che agiscono al punto di contatto tra ruota e terreno

- Il workspace di un robot mobile definisce il range delle possibili **pose** che il robot può raggiungere nell’ambiente operativo



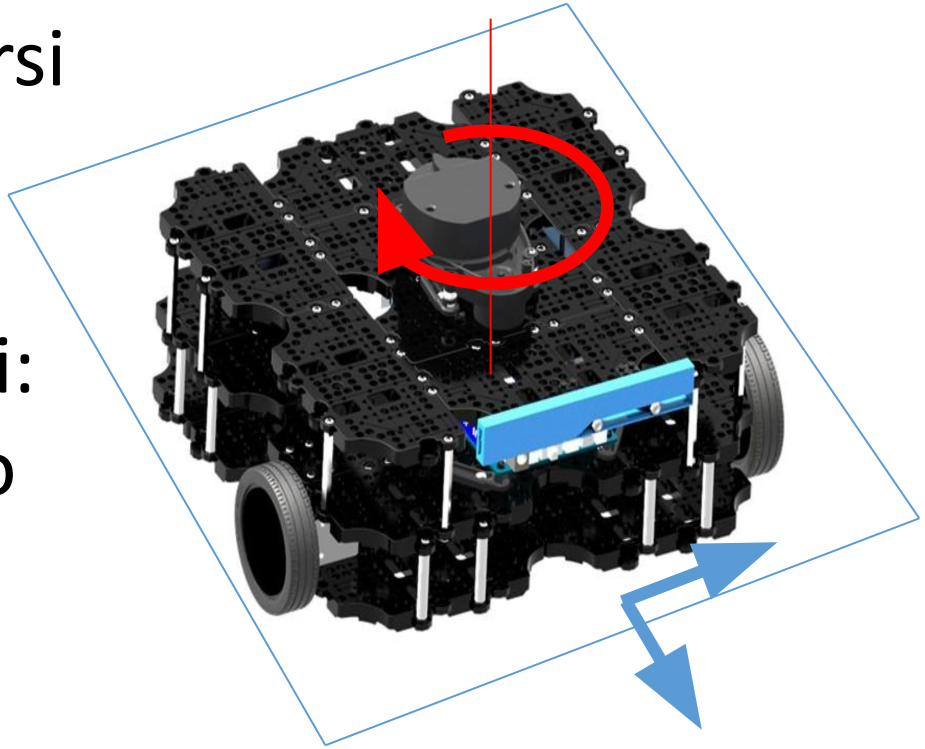
https://www.youtube.com/watch?v=E8OKp31eMpE

Introduction to Autonomous Mobile Robots
Roland Siegwart, Illah Nourbakhsh, Davide Scaramuzza

Modello del robot mobile

- Il nostro robot verrà modellato come un corpo rigido su ruote, in grado di muoversi su un piano orizzontale

- Il modello semplificato avrà 3 dimensioni:
  - 2 per descrivere la posizione nel piano
  - 1 per rappresentare l'orientazione del robot lungo l'asse verticale (che è ortogonale al piano su cui avviene il movimento)



Introduction to Autonomous Mobile Robots
Roland Siegwart, Illah Nourbakhsh, Davide Scaramuzza

Robot mobile - Position estimation

- Un robot mobile è un sistema auto-contenuto che si muove interamente rispetto all'ambiente (non ci sono punti fissi di contatto)
- Non c'è un modo diretto di misurare la posizione del robot mobile istantaneamente
- E' possibile integrare il movimento del robot al passare del tempo, ottenendo una *stima* del movimento

Cinematica

- La cinematica studia gli aspetti geometrici e temporali del moto delle strutture robotiche, senza riferimento alle cause che lo provocano

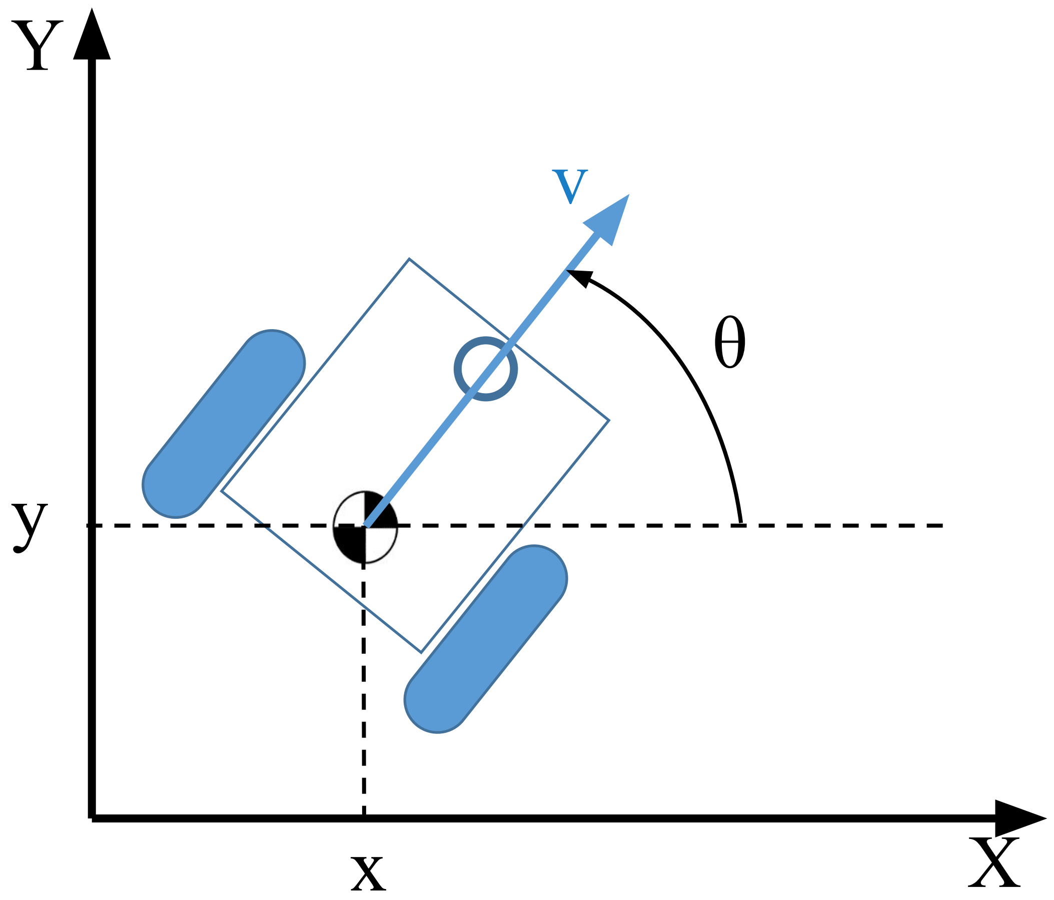
- La *cinematica diretta* è una trasformazione dallo spazio dei giunti allo spazio fisico

- La *cinematica inversa* è una trasformazione dallo spazio fisico allo spazio dei giunti. E’ necessaria per controllare il movimento del robot

Robot pose

- La *robot pose* è definita come la posizione del robot e la sua orientazione in un dato sistema di riferimento

- Per un robot mobile che si muove su un piano, la *pose* è definita dalla tripla [x, y, θ]



Introduction to Autonomous Mobile Robots
Roland Siegwart, Illah Nourbakhsh, Davide Scaramuzza

Costruzione del modello cinematico

- Derivare il modello cinematico per un robot mobile è un processo bottom-up
- Ogni ruota contribuisce individualmente al movimento del robot e, al tempo stesso, impone dei vincoli al movimento
- Poiché le ruote sono collegate tra loro in base alla geometria della scocca, i vincoli posti dalla singola ruota si combinano per formare vincoli che si applicano all'intero sistema

Limitazioni

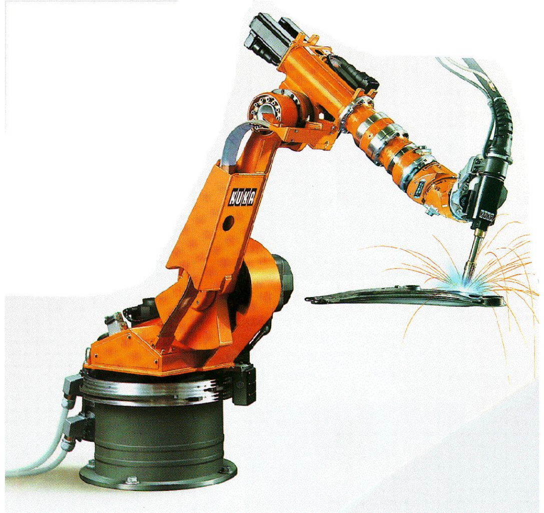
- Il movimento di un robot mobile è limitato dalla dinamica

- Per esempio, ad alte velocità, un centro di massa molto alto limita il raggio di curvatura (può esserci pericolo di cappottamento)

https://www.youtube.com/watch?v=0iui1ACWw-c

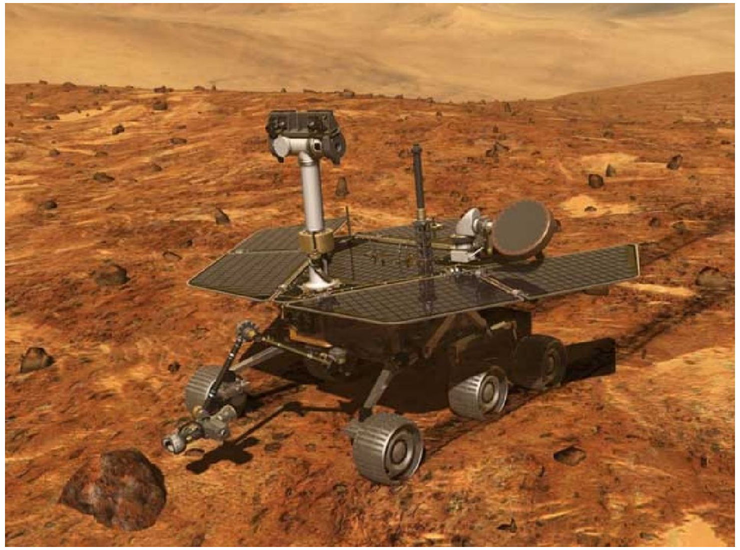
Locomozione e Manipolazione

Nella manipolazione, il braccio robotico è fisso e muove gli oggetti nello spazio di lavoro (*workspace*) impartendo loro delle forze



An Introduction to Mobile Robotics
Steve Goldberg

Nella locomozione, l'ambiente è fisso e il robot si muove impartendo forze all'ambiente



Introduction to Autonomous Mobile Robots
Roland Siegwart, Illah Nourbakhsh, Davide Scaramuzza

Aspetti chiave nella locomozione

Stabilità

- numero di punti di contatto
- centro di gravità
- stabilizzazione statica/dinamica
- inclinazione del terreno

Tipo di ambiente

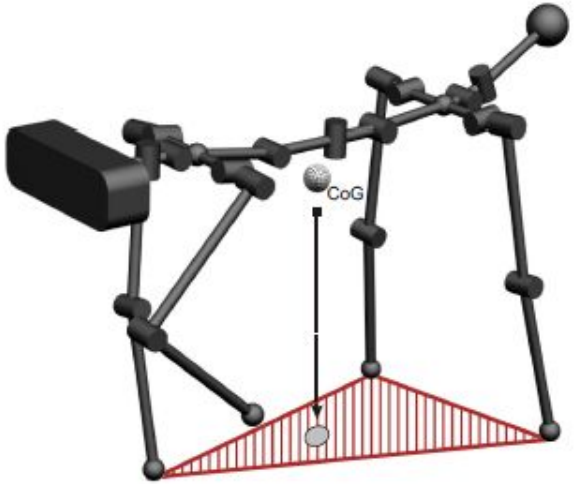
- struttura
- mezzo (acqua, aria, terreno soffice, terreno duro)

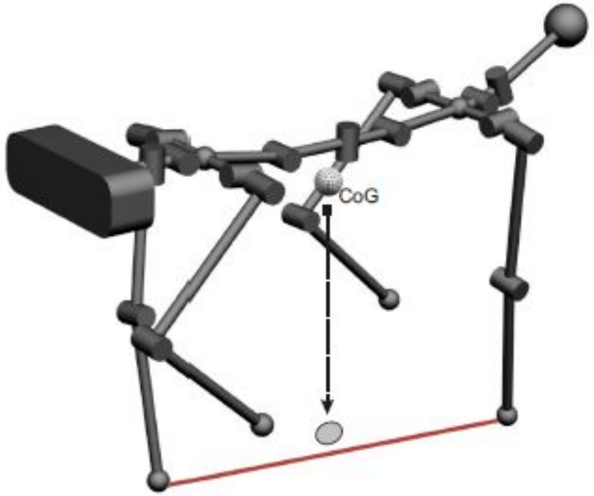
Natura del contatto

- punto/area di contatto
- angolo di contatto
- attrito

Stabilità statica/dinamica

Almeno tre gambe in contatto con il terreno sono richieste per avere stabilità statica





Stabilità statica
- Peso del corpo sostenuto da almeno tre gambe
- Anche in caso di blocco di tutti i giunti, il robot non cade
- Camminata lenta e sicura

Stabilità dinamica
- Il robot cade se non rimane in continuo movimento
- Meno di tre gambe possono essere in contatto con il terreno
- Camminata veloce, ma più onerosa per gli attuatori

Autonomous Mobile Robots
Péter Fankhauser, Marco Hutter
Roland Siegwart, Margarita Chli, Martin Rufli

Camminata NAO – RomeCup 2009

https://www.youtube.com/watch?v=vy25hEiHn98

Camminata NAO – RoboCup 2015

https://www.youtube.com/watch?v=Yfitj_-6Rxc

Robot Mobili con Ruote

Per la maggioranza delle applicazioni l'uso delle ruote è la soluzione migliore

- 3 ruote sono sufficienti a garantire stabilità
- Se si usano più di 3 ruote, è necessario un sistema di sospensioni per garantire che tutte le ruote siano in contatto con il terreno
- Il tipo di ruote da usare dipende dall'applicazione

Tipi di Ruota

- Ruota semplice sterzante
- Ruota semplice non sterzante
- Castor
- Swedish wheel
- Sferica

Ruote Attive e Passive

Le ruote possono essere attive o passive

- Ruota attiva
  collegata con un motore che fornisce una coppia motrice esterna
- Ruota passiva
  si muove per trascinamento perchè priva di coppia motrice applicata

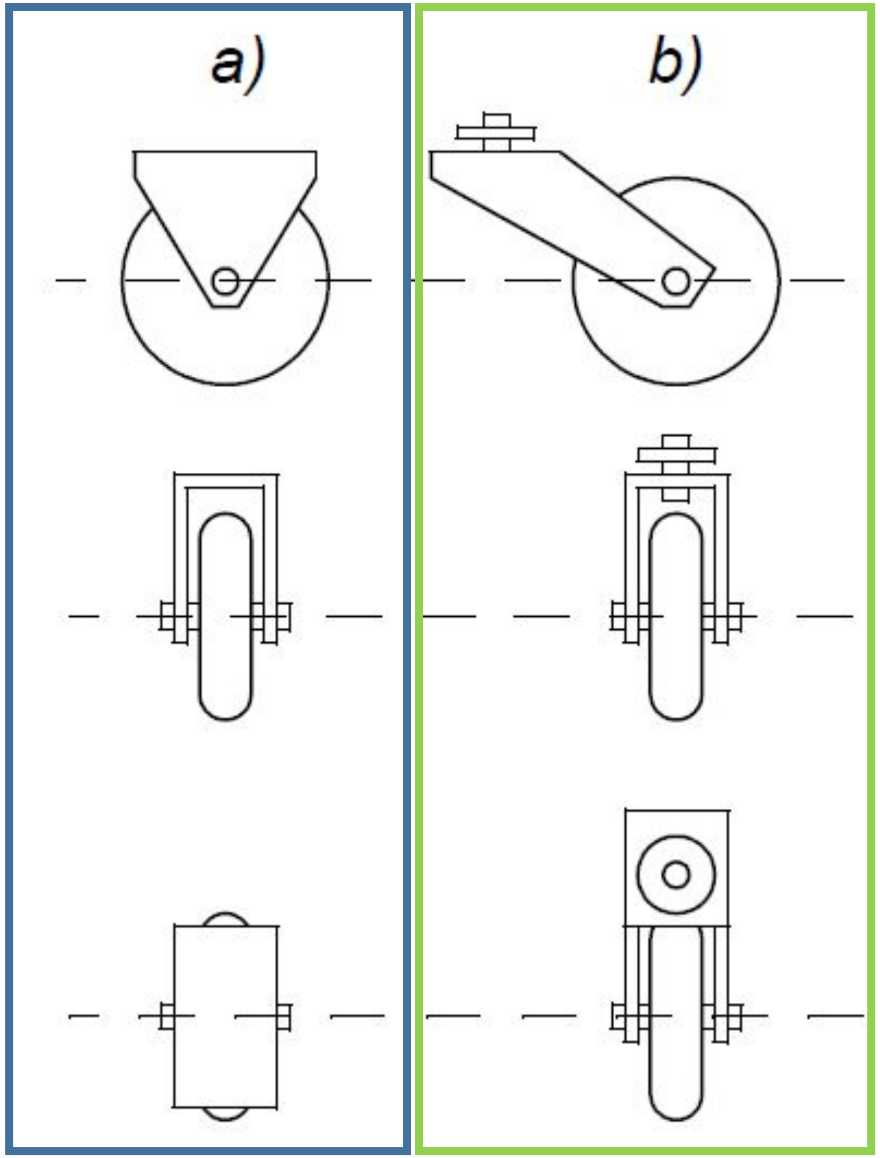
Ruota semplice e Castor

a) Ruota semplice

rotazione intorno all'asse della ruota e al punto di contatto

b) Castor

rotazione intorno al punto di contatto e all'asse del castor (offset rispetto al giunto sterzante)

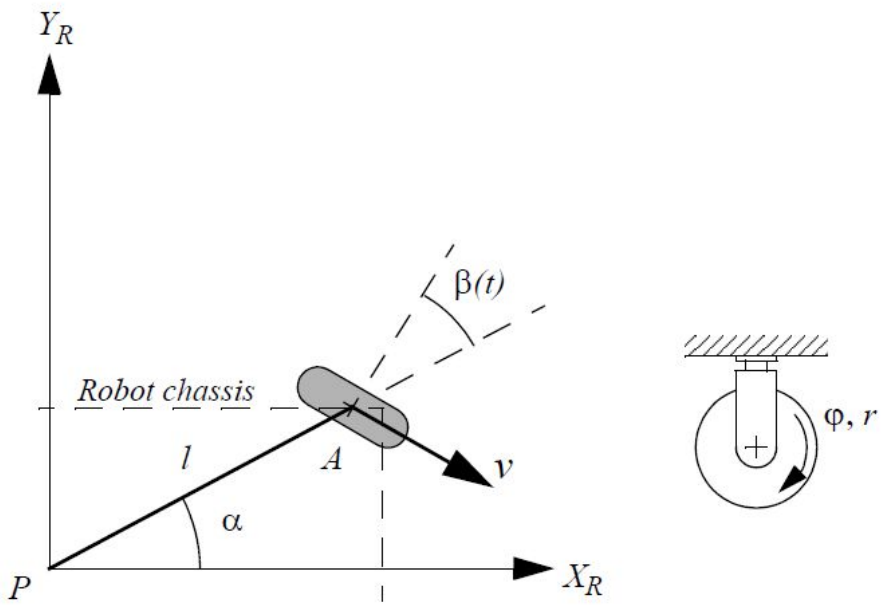


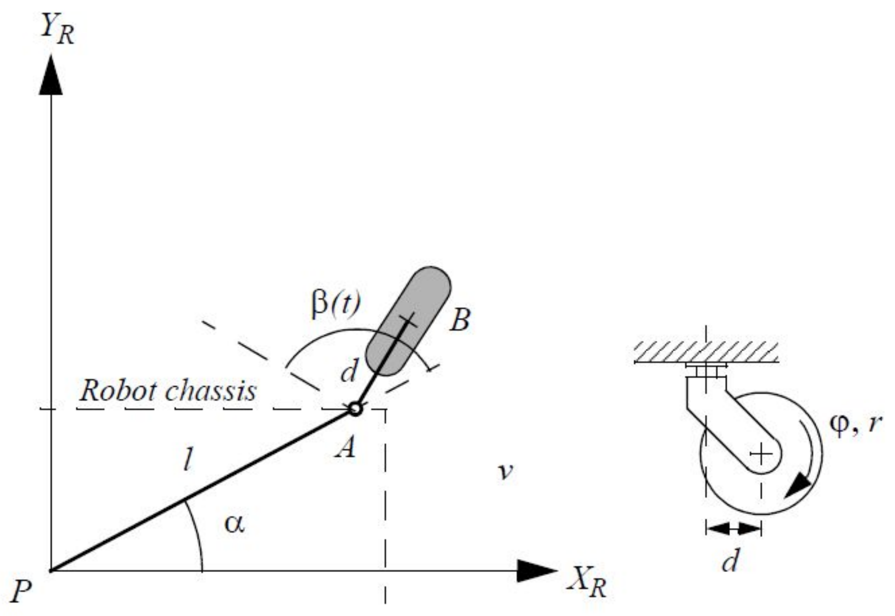
Introduction to Autonomous Mobile Robots
Roland Siegwart, Illah Nourbakhsh, Davide Scaramuzza

Ruota semplice vs Castor

La ruota semplice permette di direzionare il robot senza che ci sia un *side effect*, poichè il centro di rotazione passa attraverso il punto di contatto con il terreno

Il castor ruota intorno ad un asse che ha un offset, impartendo così una forza alla scocca del robot durante la sterzata





Introduction to Autonomous Mobile Robots
Roland Siegwart, Illah Nourbakhsh, Davide Scaramuzza

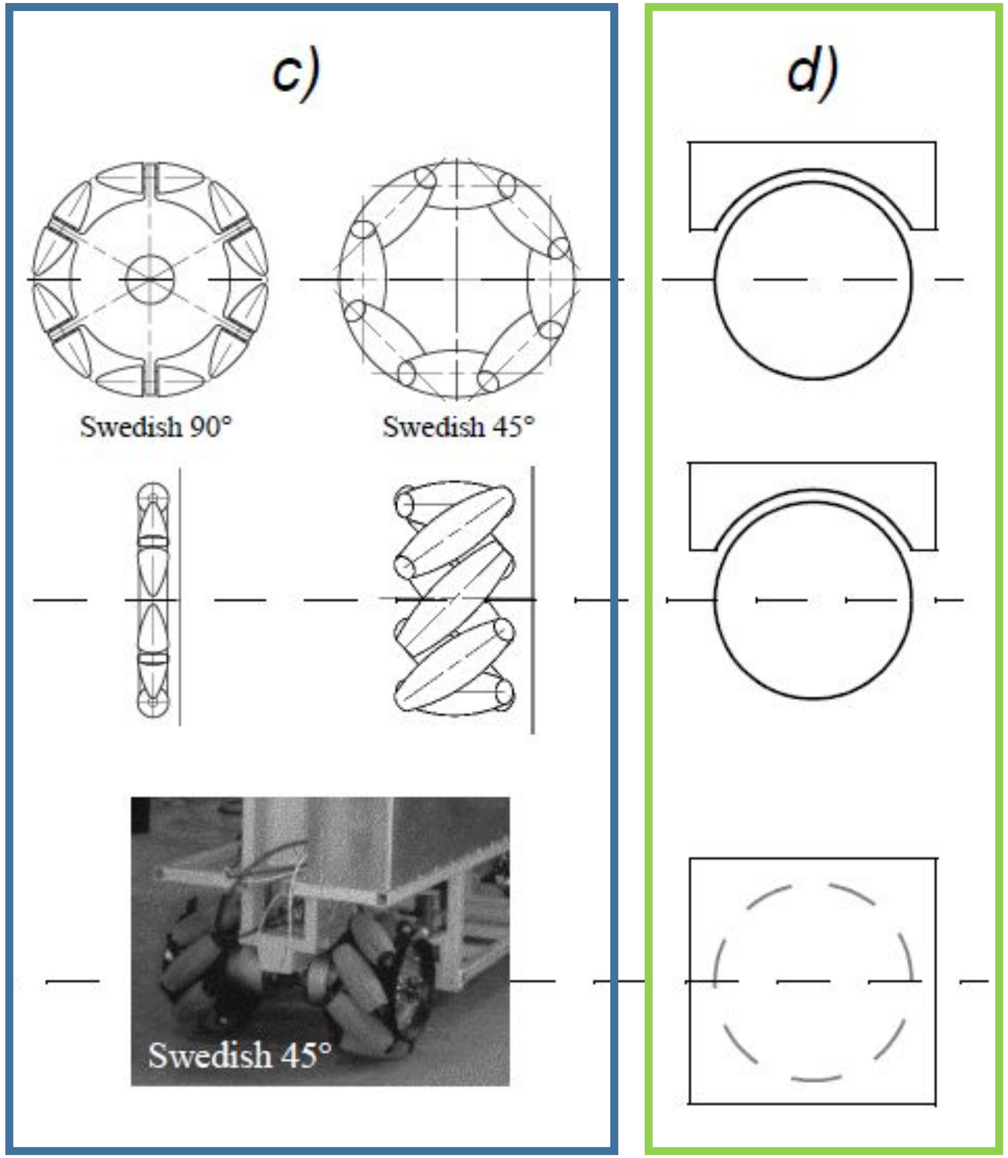
Swedish wheel e ruota sferica

c) Swedish wheel

rotazione intorno all'asse della ruota, ai rulli e al punto di contatto

d) Sferica

- difficile da realizzare
- simile alla vecchia pallina del mouse



Introduction to Autonomous Mobile Robots
Roland Siegwart, Illah Nourbakhsh, Davide Scaramuzza

Condizioni di stabilità statica

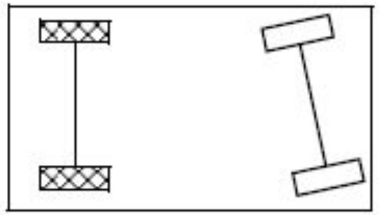
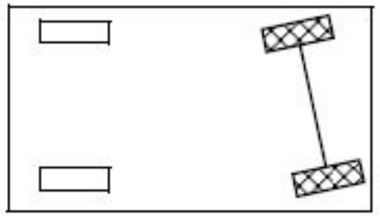
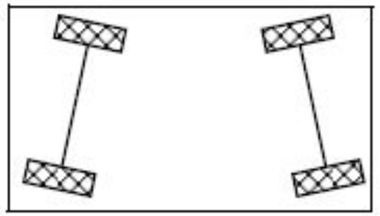
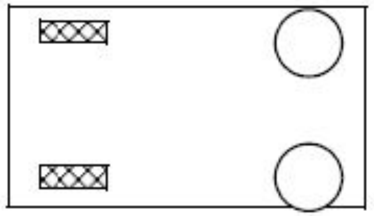
La stabilità è garantita con 3 ruote

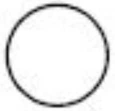
- a condizione che il centro di gravità sia all'interno del triangolo formato dai punti di contatto delle ruote con il terreno

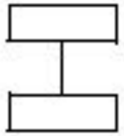
La stabilità può essere migliorata usando 4 o più ruote

- la natura iperstatica della geometria del sistema richiede un sistema di sospensioni su terreni accidentati

4 ruote

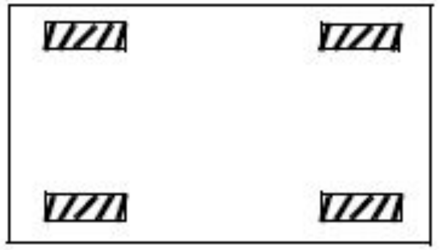
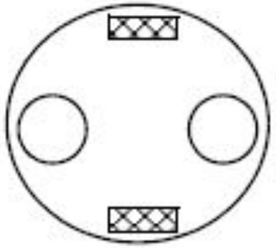
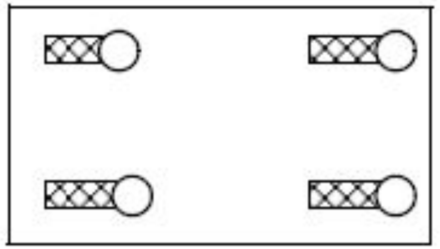
	Two motorized wheels in the rear, two steered wheels in the front; steering has to be different for the two wheels to avoid slipping/skidding.
	Two motorized and steered wheels in the front, two free wheels in the rear; steering has to be different for the two wheels to avoid slipping/skidding.
	Four steered and motorized wheels
	Two traction wheels (differential) in rear/front, two omnidirectional wheels in the front/rear

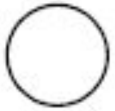
 ruota non motorizzata omnidirezionale (sferica, castor, swedish)

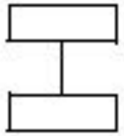
 ruote connesse

 ruota semplice motorizzata

4 ruote

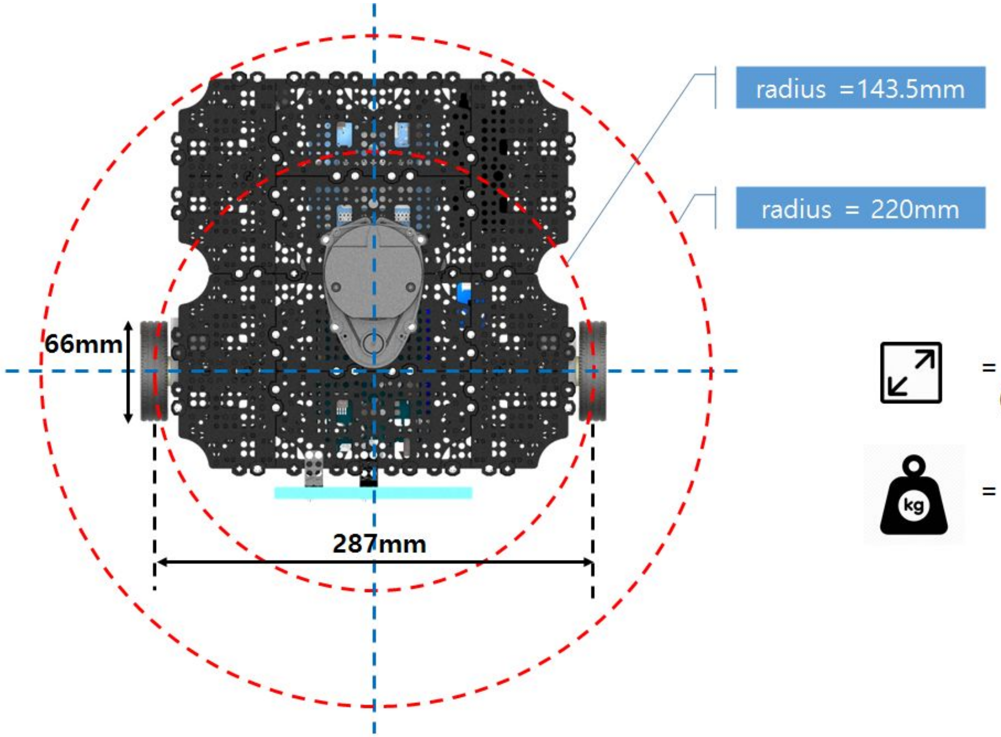
	Four omnidirectional wheels
	Two-wheel differential drive with two additional points of contact
	Four motorized and steered castor wheels

 ruota non motorizzata omnidirezionale (sferica, castor, swedish)

 ruote connesse

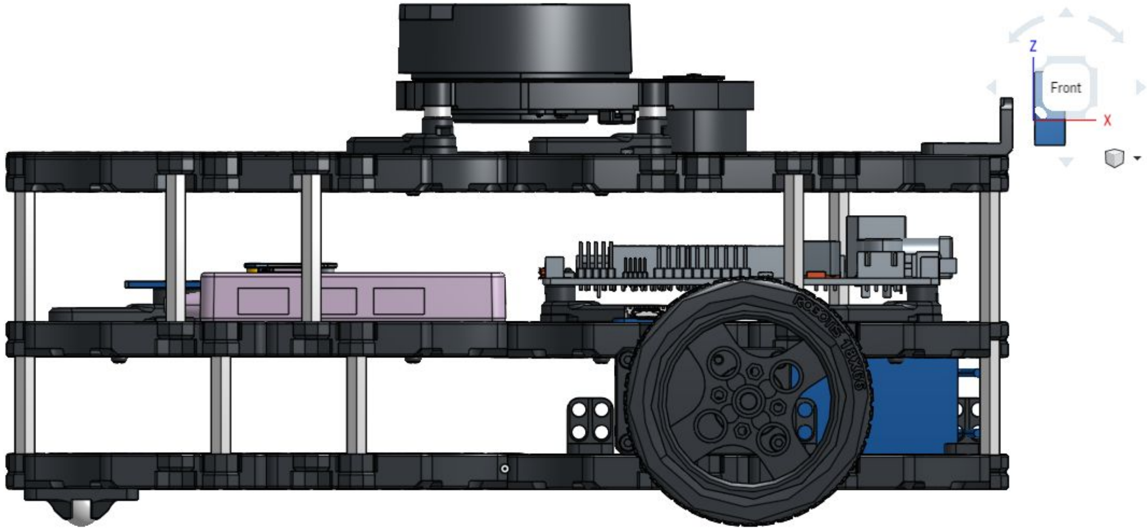
 ruota semplice motorizzata

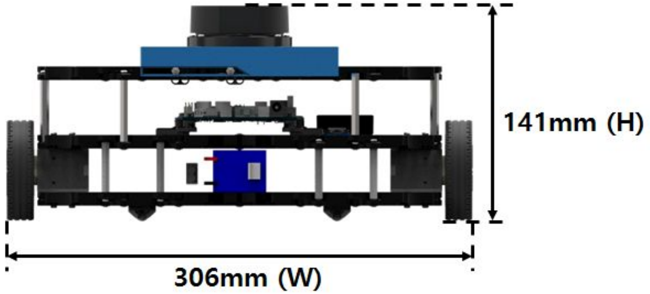
Turtlebot 3 waffle

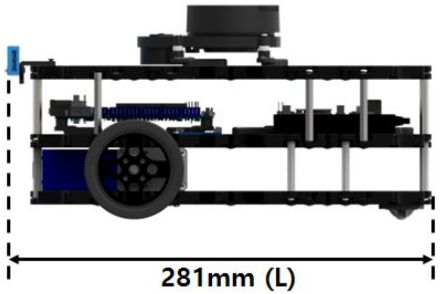


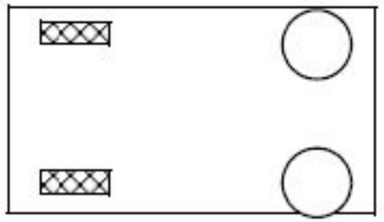
= 281 x 306 x 141
(L x W x H, mm)

= 1.8 Kg







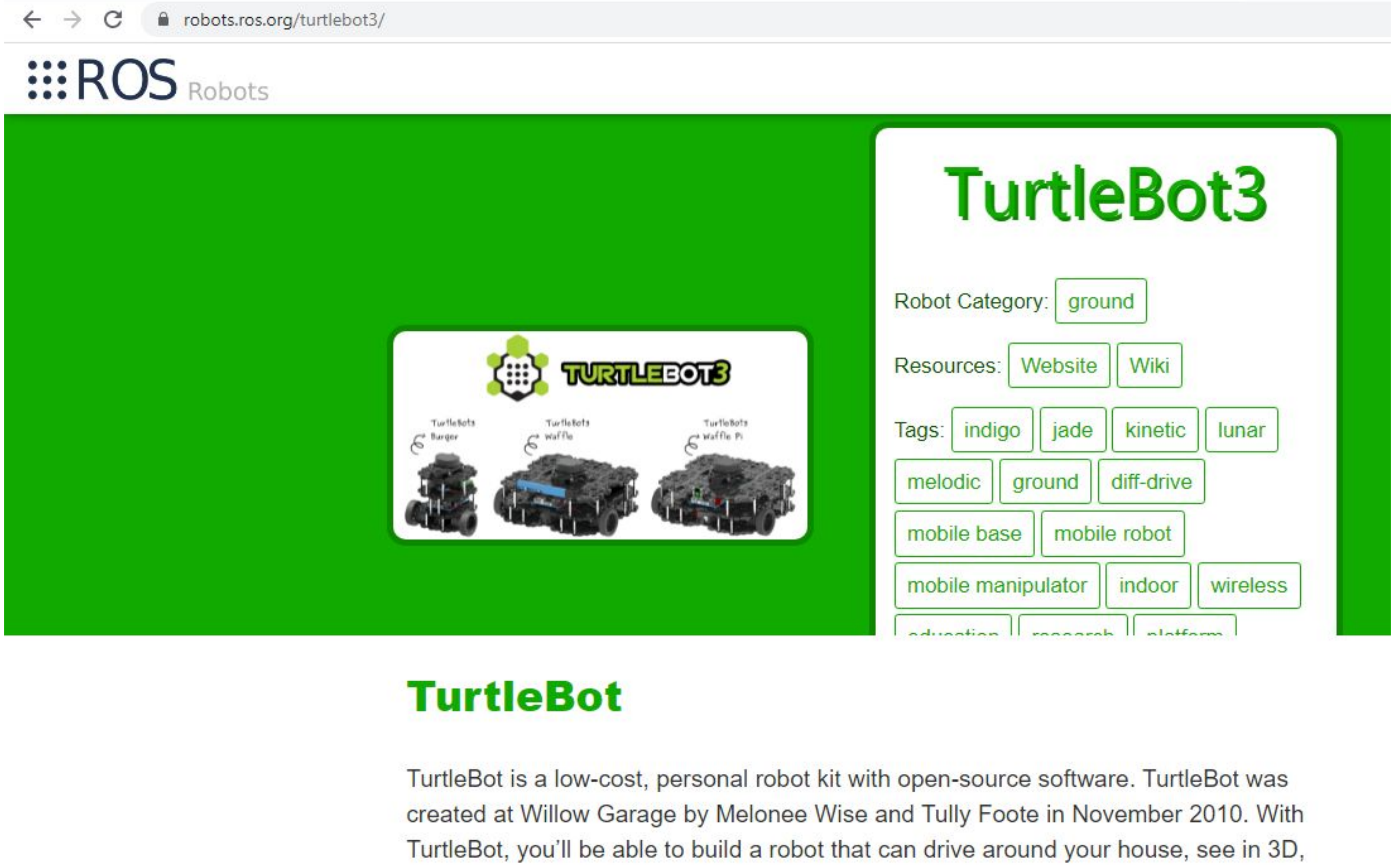


Two traction wheels (differential) in rear/front, two omnidirectional wheels in the front/rear

Esempio Youbot

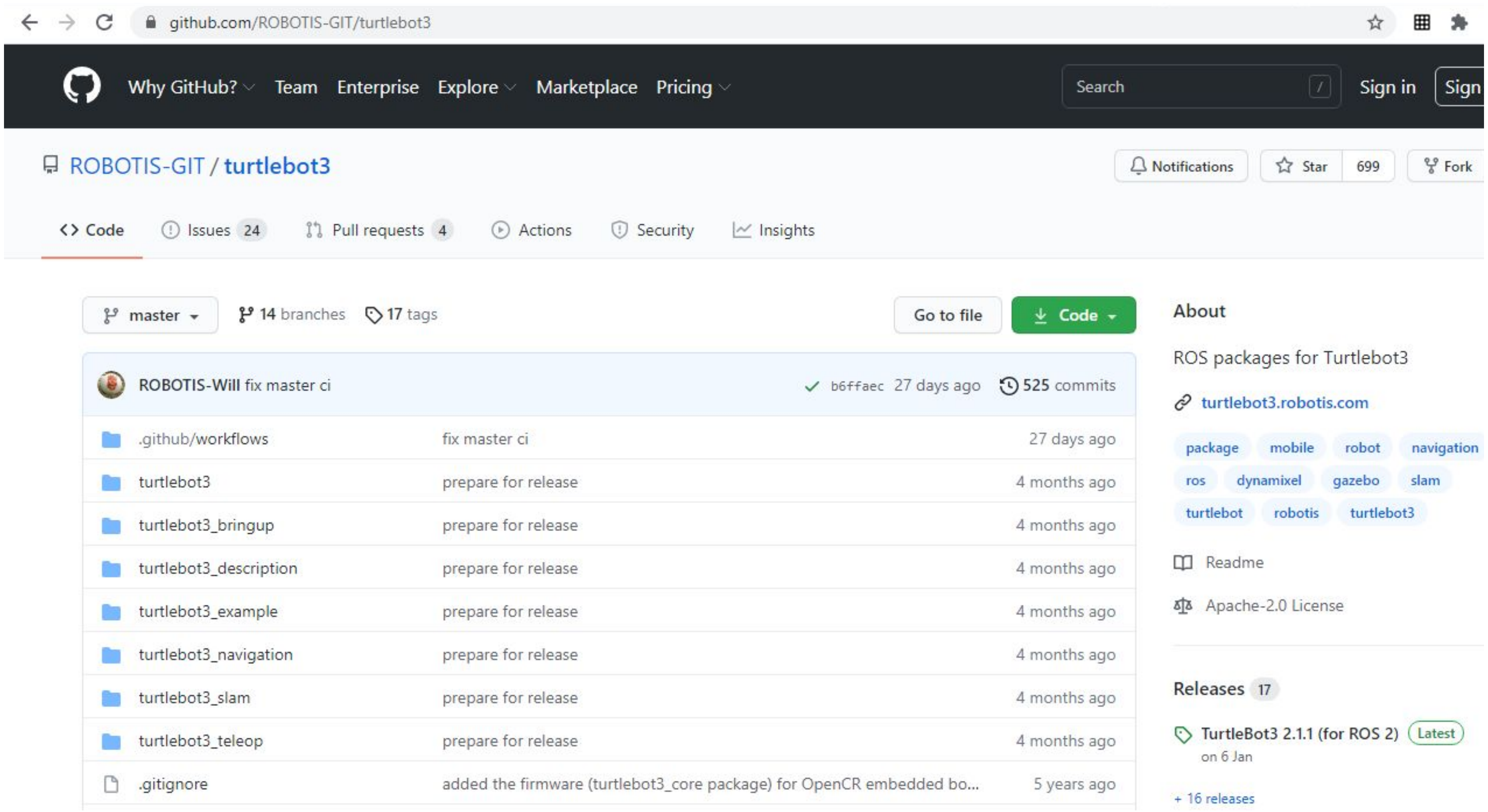
https://www.youtube.com/watch?v=SfwCXyuxgQs

ROS e i robot mobili su ruote



https://robots.ros.org/turtlebot3/

Quanti nodi?



https://github.com/ROBOTIS-GIT/turtlebot3

roslaunch

roslaunch è un tool per semplificare

- il lancio di più nodi ROS
- il settaggio dei parametri

roslaunch utilizza i cosiddetti “launch file” che sono file XML contenenti la lista dei nodi da lanciare con i rispettivi parametri

roslaunch - sintassi

`roslaunch <package> <launch file>`

- i launch file hanno per convenzione un nome che termina con .launch

- `roscore` viene automaticamente lanciato quando si esegue `roslaunch`

Esempio launch file

```
<launch>
   <node name="talker" pkg="hello_ros" type="talker.py" output="screen"/>
   <node name="listener" pkg="hello_ros" type="listener.py" output="screen"/>
</launch>
```

Il tag <node> contiene gli attributi per l’esecuzione del nodo

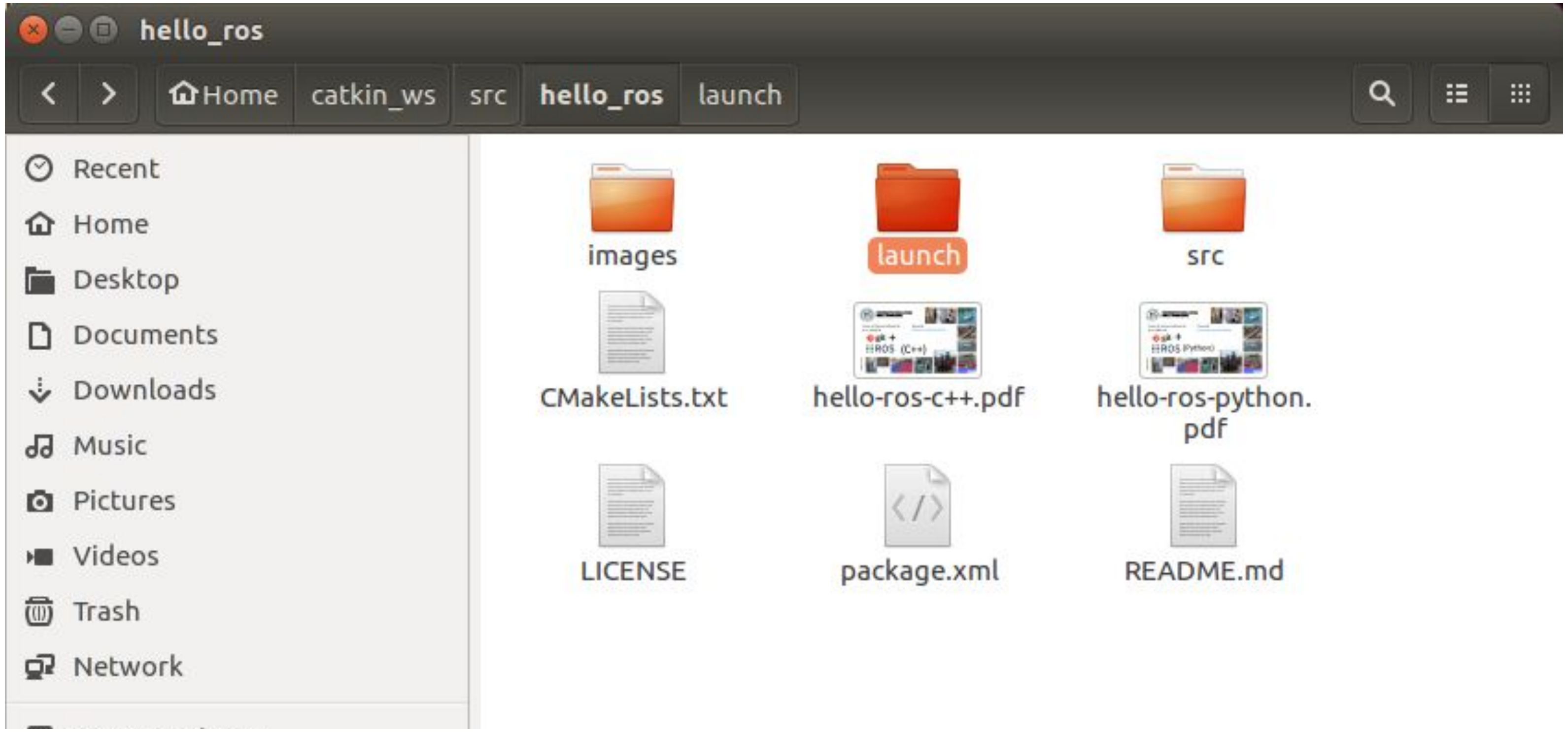
- name è il nome con cui il nodo verrà inserito nel grafo di ROS
- pkg indica il package nel quale può essere trovato il nodo
- type specifica il filename dell’eseguibile
- output posto a “screen” indica che i messaggi di log di ROS verranno mostrati sul terminale su cui verrà eseguito il comando `roslaunch`

hello_ros: git repo recap

```
bloisi@bloisi-U36SG:~/catkin_ws/src$ git clone https://github.com/dbloisi/hello_ros.git
Cloning into 'hello_ros'...
remote: Enumerating objects: 26, done.
remote: Counting objects: 100% (26/26), done.
remote: Compressing objects: 100% (26/26), done.
remote: Total 74 (delta 13), reused 0 (delta 0), pack-reused 48
Unpacking objects: 100% (74/74), done.
Checking connectivity... done.
bloisi@bloisi-U36SG:~/catkin_ws/src$ cd hello_ros
bloisi@bloisi-U36SG:~/catkin_ws/src/hello_ros$ ls
CMakeLists.txt     hello-ros-python.pdf  LICENSE      README.md
hello-ros-c++.pdf  images                package.xml  src
bloisi@bloisi-U36SG:~/catkin_ws/src/hello_ros$ 
```

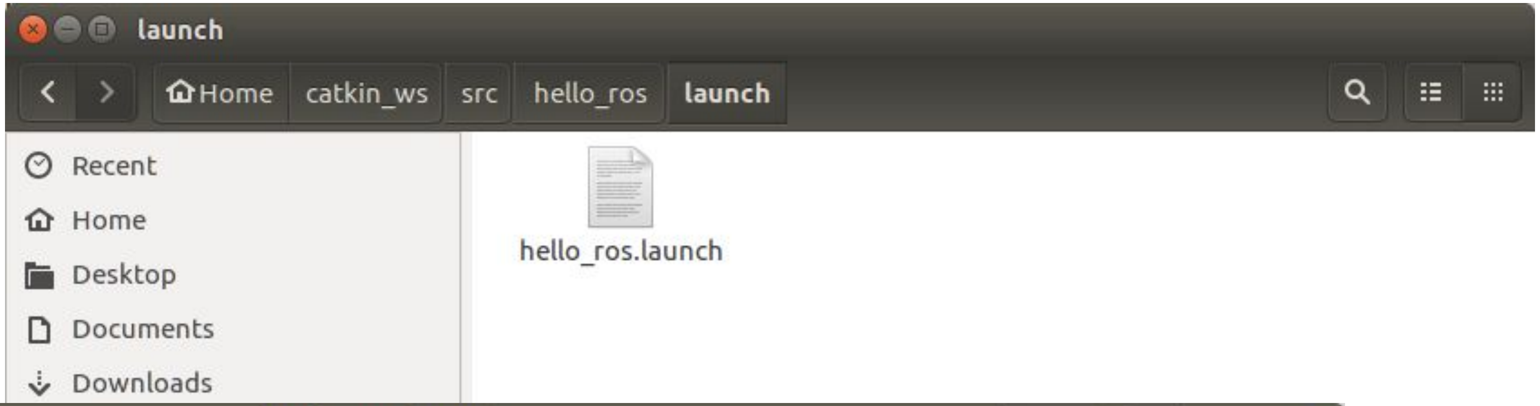
hello_ros launch file

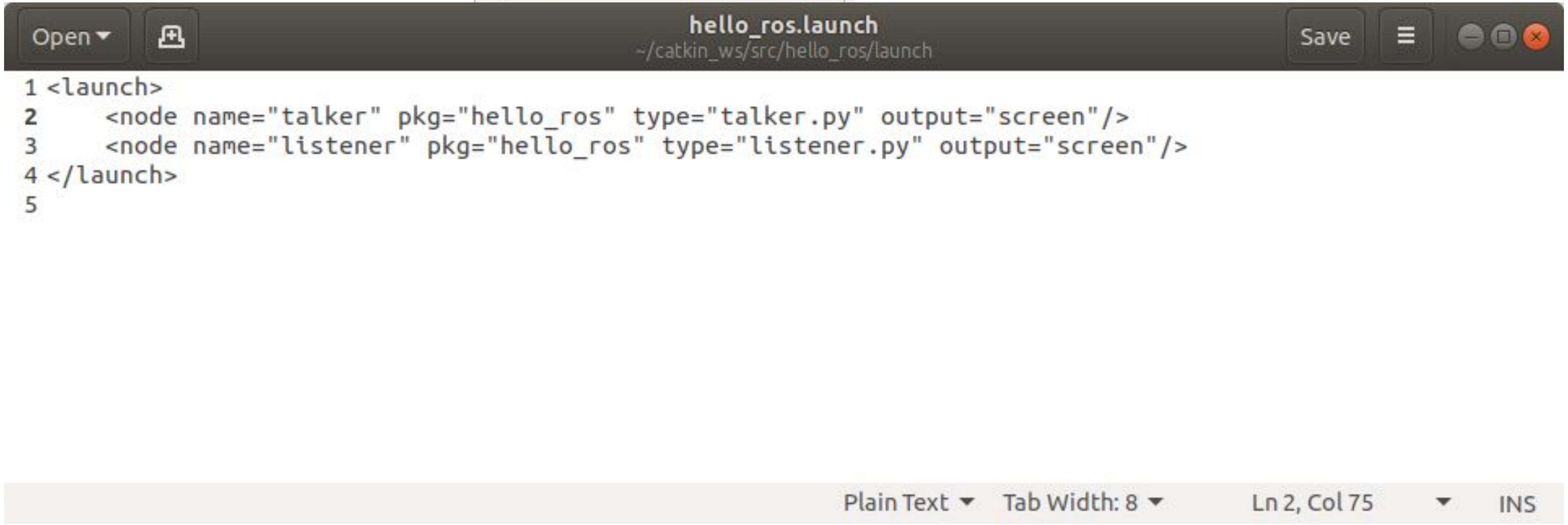
Creiamo una cartella launch



hello_ros launch file

Creiamo dentro la cartella `launch` il file `hello_ros.launch`





hello_ros.launch

esecuzione

```
bloisi@bloisi-U36SG:~/catkin_ws$ roslaunch hello_ros hello_ros.launch
... logging to /home/bloisi/.ros/log/56139d64-adb3-11eb-90a4-9de4bf981568/roslaunch-bloisi-U36SG-17297.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://localhost:41847/

SUMMARY
========

PARAMETERS
 * /rosdistro: noetic
 * /rosversion: 1.15.11

NODES
  /
    listener (hello_ros/listener.py)
    talker (hello_ros/talker.py)

auto-starting new master
process[master]: started with pid [17322]
ROS_MASTER_URI=http://localhost:11311

setting /run_id to 56139d64-adb3-11eb-90a4-9de4bf981568
process[rosout-1]: started with pid [17332]
started core service [/rosout]
process[talker-2]: started with pid [17335]
process[listener-3]: started with pid [17336]
[INFO] [1620227133.908454]: hello world 1620227133.9083347
[INFO] [1620227133.909943]: /listenerI heard hello world 1620227133.9083347
[INFO] [1620227134.008725]: hello world 1620227134.0085406
[INFO] [1620227134.010525]: /listenerI heard hello world 1620227134.0085406
[INFO] [1620227134.108713]: hello world 1620227134.1085317
[INFO] [1620227134.110436]: /listenerI heard hello world 1620227134.1085317
[INFO] [1620227134.208630]: hello world 1620227134.208502
[INFO] [1620227134.210462]: /listenerI heard hello world 1620227134.208502
```

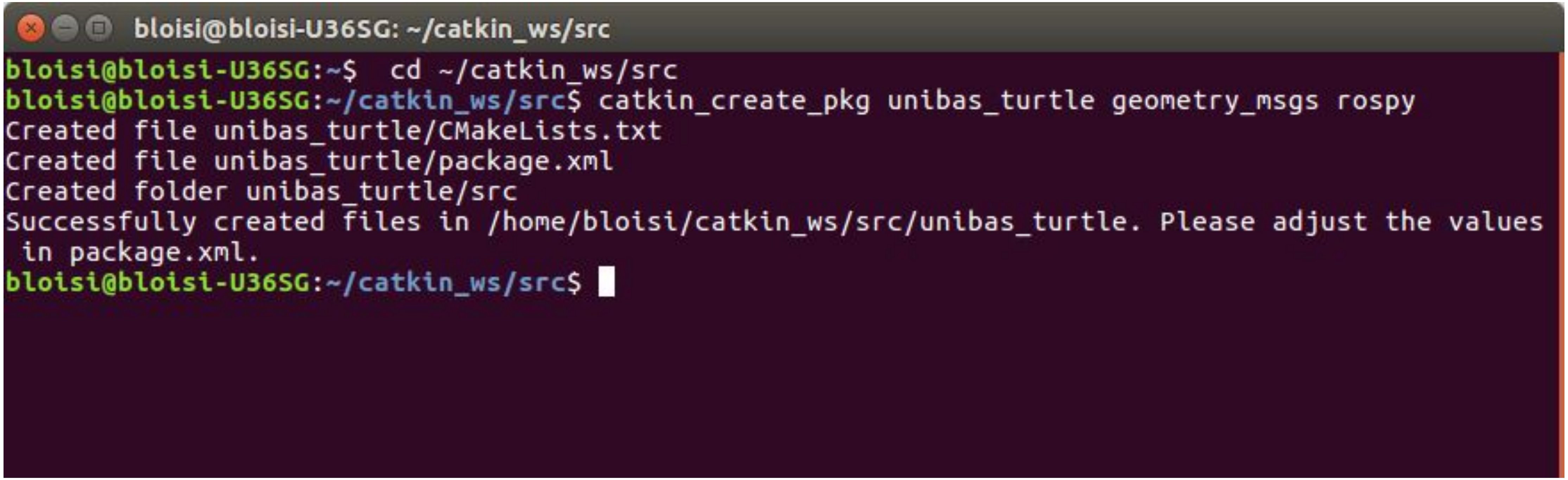
hello_ros launch file: git

```
bloisi@bloisi-U36SG: ~/catkin_ws/src/hello_ros
bloisi@bloisi-U36SG:~/catkin_ws/src/hello_ros$ ls
CMakeLists.txt     hello-ros-python.pdf  launch   package.xml  src
hello-ros-c++.pdf  images                LICENSE  README.md
bloisi@bloisi-U36SG:~/catkin_ws/src/hello_ros$ git add launch
bloisi@bloisi-U36SG:~/catkin_ws/src/hello_ros$ git commit -m "adding launch fold
er and launch file"
[master eeefdd0] adding launch folder and launch file
 1 file changed, 5 insertions(+)
 create mode 100644 launch/hello_ros.launch
bloisi@bloisi-U36SG:~/catkin_ws/src/hello_ros$ git pull
Already up-to-date.
bloisi@bloisi-U36SG:~/catkin_ws/src/hello_ros$ git push origin master
Username for 'https://github.com': dbloisi
Password for 'https://dbloisi@github.com':
Counting objects: 5, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (5/5), 519 bytes | 0 bytes/s, done.
Total 5 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 1 local object.
To https://github.com/dbloisi/hello_ros.git
   03675ef..eeefdd0  master -> master
bloisi@bloisi-U36SG:~/catkin_ws/src/hello_ros$
```

Package unibas_turtle

Per creare il package digitiamo

```
cd ~/catkin_ws/src
catkin_create_pkg unibas_turtle geometry_msgs rospy
```



http://wiki.ros.org/turtlesim/Tutorials/Moving%20in%20a%20Straight%20Line

Package unibas_turtle: package.xml

Open | *package.xml ~/catkin_ws/src/unibas_turtle | Save

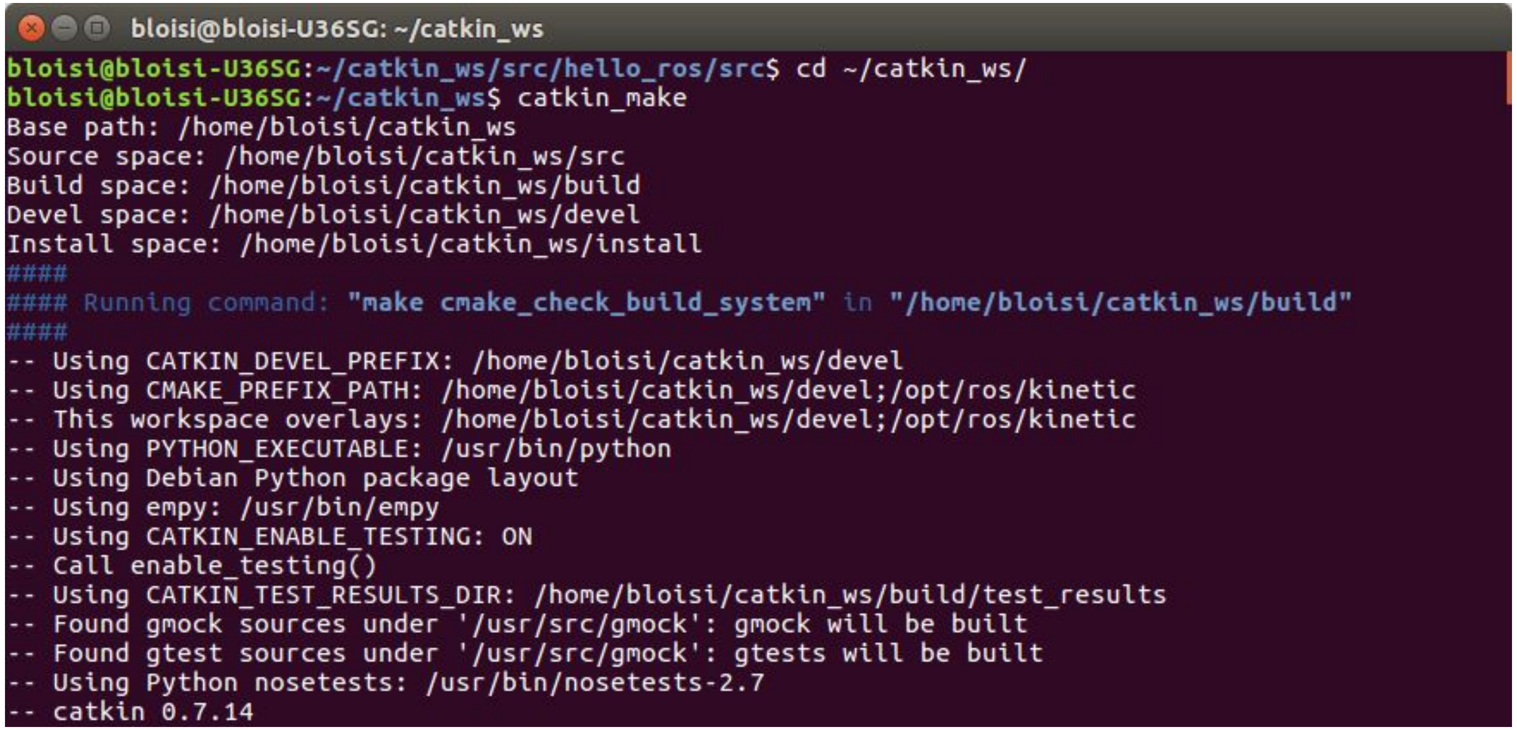
```xml
<?xml version="1.0"?>
<package format="2">
  <name>unibas_turtle</name>
  <version>0.0.0</version>
  <description>The unibas_turtle package</description>

  <!-- One maintainer tag required, multiple allowed, one person per tag -->
  <!-- Example:  -->
  <!-- <maintainer email="jane.doe@example.com">Jane Doe</maintainer> -->
  <maintainer email="domenico.bloisi@gmail.com">domenico bloisi</maintainer>


  <!-- One license tag required, multiple allowed, one license per tag -->
  <!-- Commonly used license strings: -->
  <!--   BSD, MIT, Boost Software License, GPLv2, GPLv3, LGPLv2.1, LGPLv3 -->
  <license>GPLv3</license>


  <!-- Url tags are optional, but multiple are allowed, one per tag -->
  <!-- Optional attribute type can be: website, bugtracker, or repository -->
  <!-- Example: -->
  <!-- <url type="website">http://wiki.ros.org/unibas_turtle</url> -->


  <!-- Author tags are optional, multiple are allowed, one per tag -->
  <!-- Authors do not have to be maintainers, but could be -->
  <!-- Example: -->
  <!-- <author email="jane.doe@example.com">Jane Doe</author> -->

```

XML Tab Width: 8 Ln 16, Col 17 INS

Package unibas_turtle: catkin_make

Compiliamo con catkin_make

```
cd ~/catkin_ws
catkin_make
```



```
bloisi@bloisi-U36SG: ~/catkin_ws
bloisi@bloisi-U36SG:~/catkin_ws/src/hello_ros/src$ cd ~/catkin_ws/
bloisi@bloisi-U36SG:~/catkin_ws$ catkin_make
Base path: /home/bloisi/catkin_ws
Source space: /home/bloisi/catkin_ws/src
Build space: /home/bloisi/catkin_ws/build
Devel space: /home/bloisi/catkin_ws/devel
Install space: /home/bloisi/catkin_ws/install
####
#### Running command: "make cmake_check_build_system" in "/home/bloisi/catkin_ws/build"
####
-- Using CATKIN_DEVEL_PREFIX: /home/bloisi/catkin_ws/devel
-- Using CMAKE_PREFIX_PATH: /home/bloisi/catkin_ws/devel;/opt/ros/kinetic
-- This workspace overlays: /home/bloisi/catkin_ws/devel;/opt/ros/kinetic
-- Using PYTHON_EXECUTABLE: /usr/bin/python
-- Using Debian Python package layout
-- Using empy: /usr/bin/empy
-- Using CATKIN_ENABLE_TESTING: ON
-- Call enable_testing()
-- Using CATKIN_TEST_RESULTS_DIR: /home/bloisi/catkin_ws/build/test_results
-- Found gmock sources under '/usr/src/gmock': gmock will be built
-- Found gtest sources under '/usr/src/gmock': gtests will be built
-- Using Python nosetests: /usr/bin/nosetests-2.7
-- catkin 0.7.14
```

http://wiki.ros.org/turtlesim/Tutorials/Moving%20in%20a%20Straight%20Line

Package unibas_turtle: src

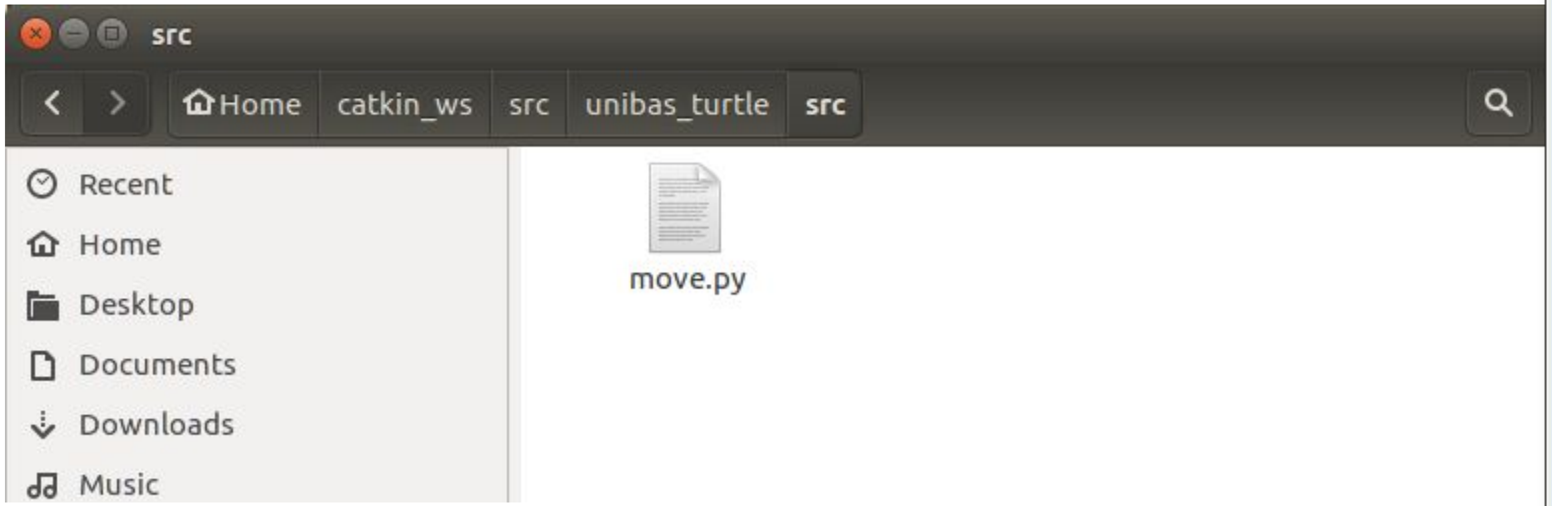
Creiamo una cartella source che conterrà il codice sorgente

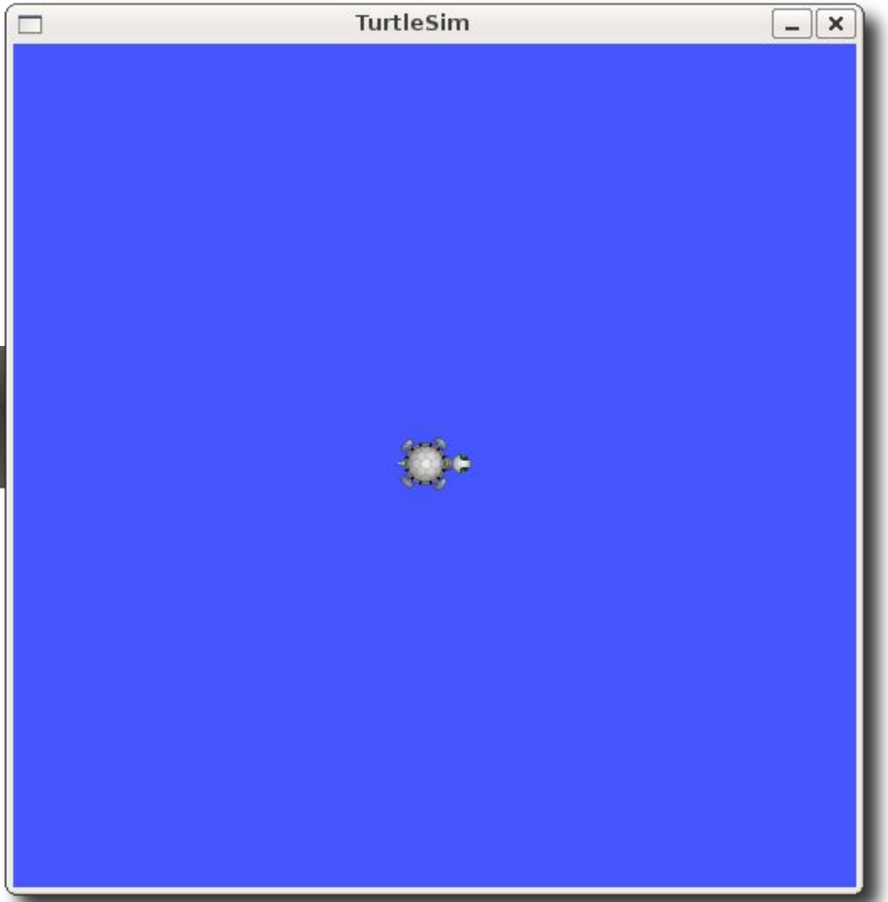
```
bloisi@bloisi-U36SG: ~/catkin_ws/src/unibas_turtle/src
bloisi@bloisi-U36SG:~/catkin_ws$ cd ~/catkin_ws/src/unibas_turtle
bloisi@bloisi-U36SG:~/catkin_ws/src/unibas_turtle$ ls
CMakeLists.txt  package.xml  src
bloisi@bloisi-U36SG:~/catkin_ws/src/unibas_turtle$ cd src
bloisi@bloisi-U36SG:~/catkin_ws/src/unibas_turtle/src$ ls
bloisi@bloisi-U36SG:~/catkin_ws/src/unibas_turtle/src$
```

http://wiki.ros.org/turtlesim/Tutorials/Moving%20in%20a%20Straight%20Line

Package unibas_turtle: src

Creiamo un file `move.py` per far muovere la tartaruga di `turtlesim`





http://wiki.ros.org/turtlesim/Tutorials/Moving%20in%20a%20Straight%20Line
http://wiki.ros.org/turtlesim

idea

- Vogliamo far muovere la tartaruga controllandone la velocità

- Adottiamo per la tartaruga il modello di un robot differenziale

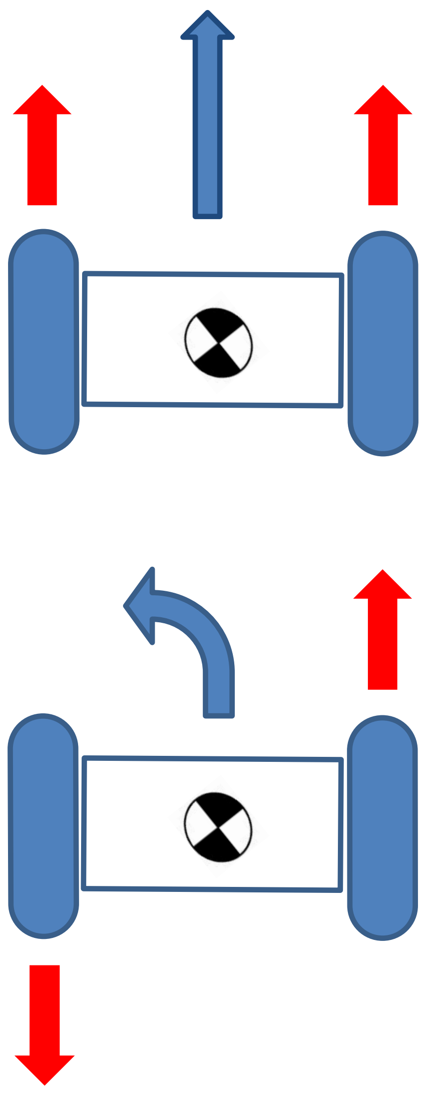
- Modifichiamo i valori di velocità lineare e angolare per controllare il moto

Differential drive robot

- Un robot differenziale su ruote è una base mobile avente due ruote motorizzate indipendenti

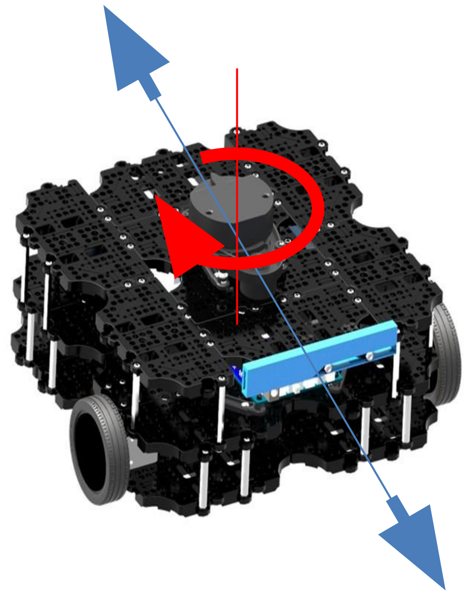
- Le ruote sono posizionate ai due lati opposti della scocca

- Il robot si muove in avanti quando entrambe le ruote gira in avanti, mentre gira sul posto quando una ruota gira in avanti e l’altra gira all’indietro



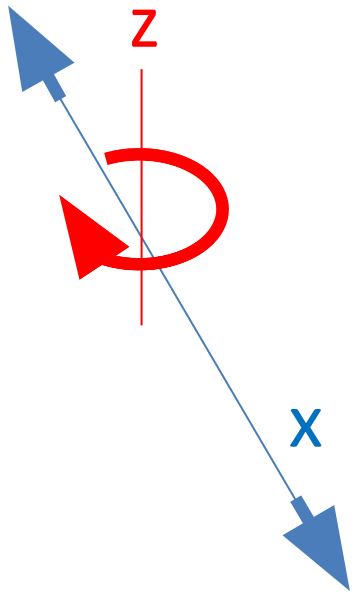
Movimento di un robot differenziale

Data la sua configurazione, un robot differenziale può muoversi solo in avanti o indietro lungo il suo asse longitudinale e può ruotare solo lungo il suo asse verticale



Movimento di un robot differenziale

- Il robot non potrà muoversi di lato o verticalmente
- Per tali motivi ci bastano la componente lineare x e la componente angolare z per controllare il movimento
- Nel caso di un robot omnidirezionale, avremo anche una componente y per lo spostamento laterale
- Quante componenti avremo per un robot underwater?



Comandi di velocità in ROS

Per far muovere un robot in ROS è necessario pubblicare Twist messages sul topic cmd_vel

geometry_msgs/Twist Message

File: geometry_msgs/Twist.msg

Raw Message Definition

```
# This expresses velocity in free space broken into its linear and angular parts.
Vector3  linear
Vector3  angular
```

Compact Message Definition

```
geometry_msgs/Vector3 linear
geometry_msgs/Vector3 angular
```

move.py

```python
#!/usr/bin/env python
import rospy
from geometry_msgs.msg import Twist

def move():
    # Starts a new node
    rospy.init_node('move', anonymous=True)
    velocity_publisher = rospy.Publisher('/turtle1/cmd_vel', Twist, queue_size=10)
    vel_msg = Twist()
```

move.py

```
#Receiveing the user's input
print("Let's move your robot")
speed = float(input("Input your speed:"))
distance = float(input("Type your distance:"))
isForward = int(input("Foward?:")) #True or False

#Checking if the movement is forward or backwards
if(isForward):
   vel_msg.linear.x = abs(speed)
else:
   vel_msg.linear.x = -abs(speed)

#Since we are moving just in x-axis
vel_msg.linear.y = 0
vel_msg.linear.z = 0
vel_msg.angular.x = 0
vel_msg.angular.y = 0
vel_msg.angular.z = 0
```

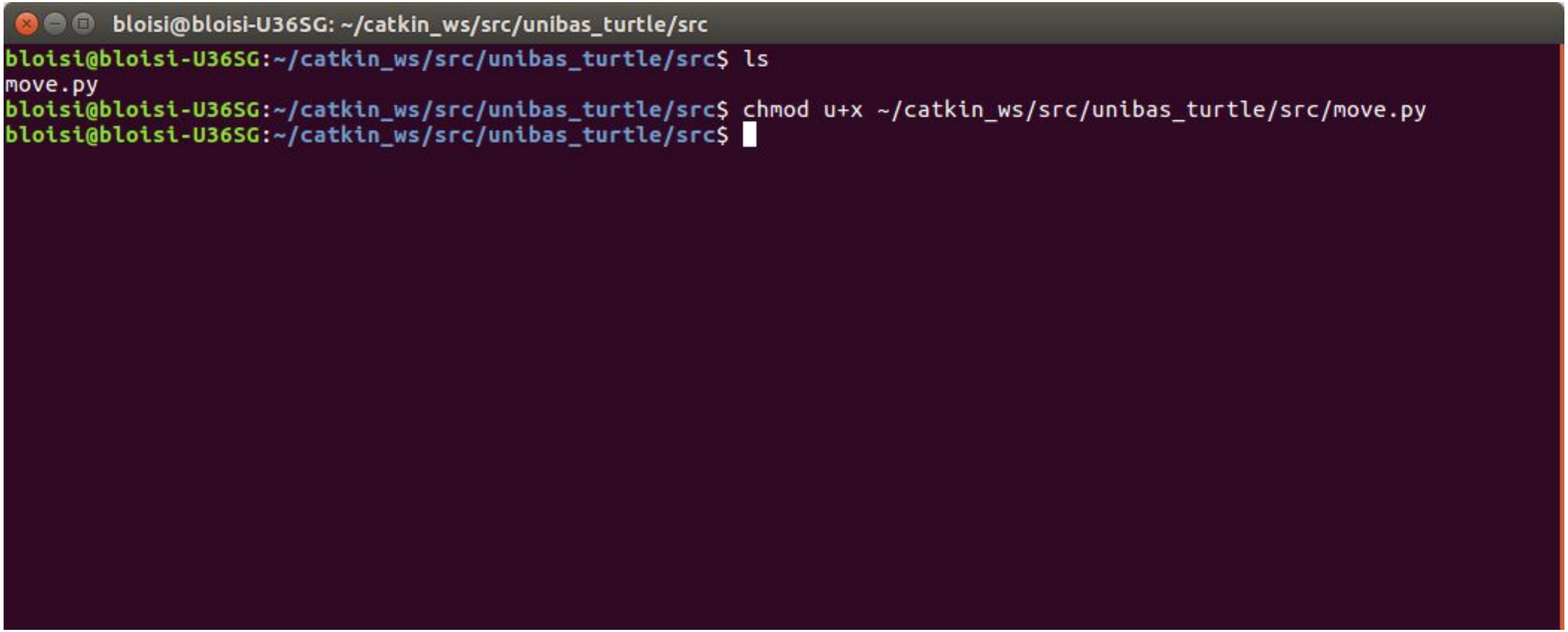
move.py

```
while not rospy.is_shutdown():
    #Setting the current time for distance calculus
    t0 = rospy.Time.now().to_sec()
    current_distance = 0

    #Loop to move the turtle in an specified distance
    while(current_distance < distance):
        #Publish the velocity
        velocity_publisher.publish(vel_msg)
        #Takes actual time to velocity calculus
        t1=rospy.Time.now().to_sec()
        #Calculates distancePoseStamped
        current_distance= speed*(t1-t0)
    #After the loop, stops the robot
    vel_msg.linear.x = 0
    #Force the robot to stop
    velocity_publisher.publish(vel_msg)
```

move.py

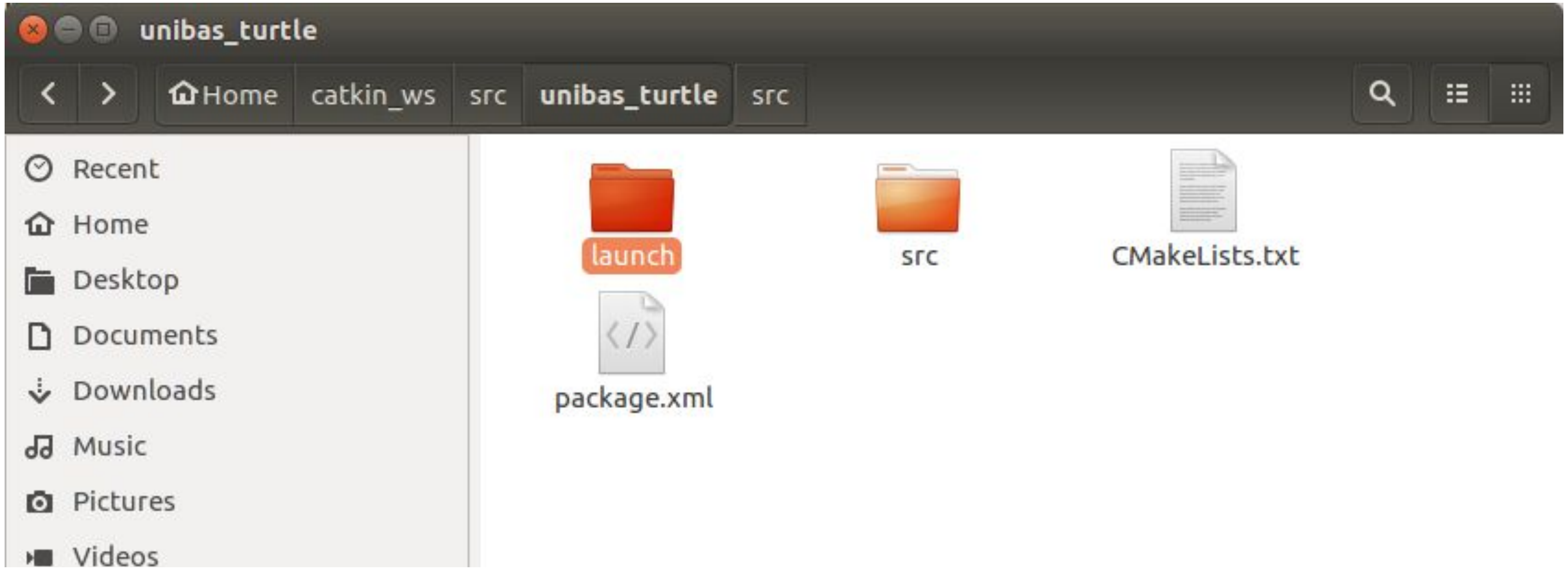
```
if __name__ == '__main__':
    try:
        #Testing our function
        move()
     except rospy.ROSInterruptException: pass
```

permessi per move.py



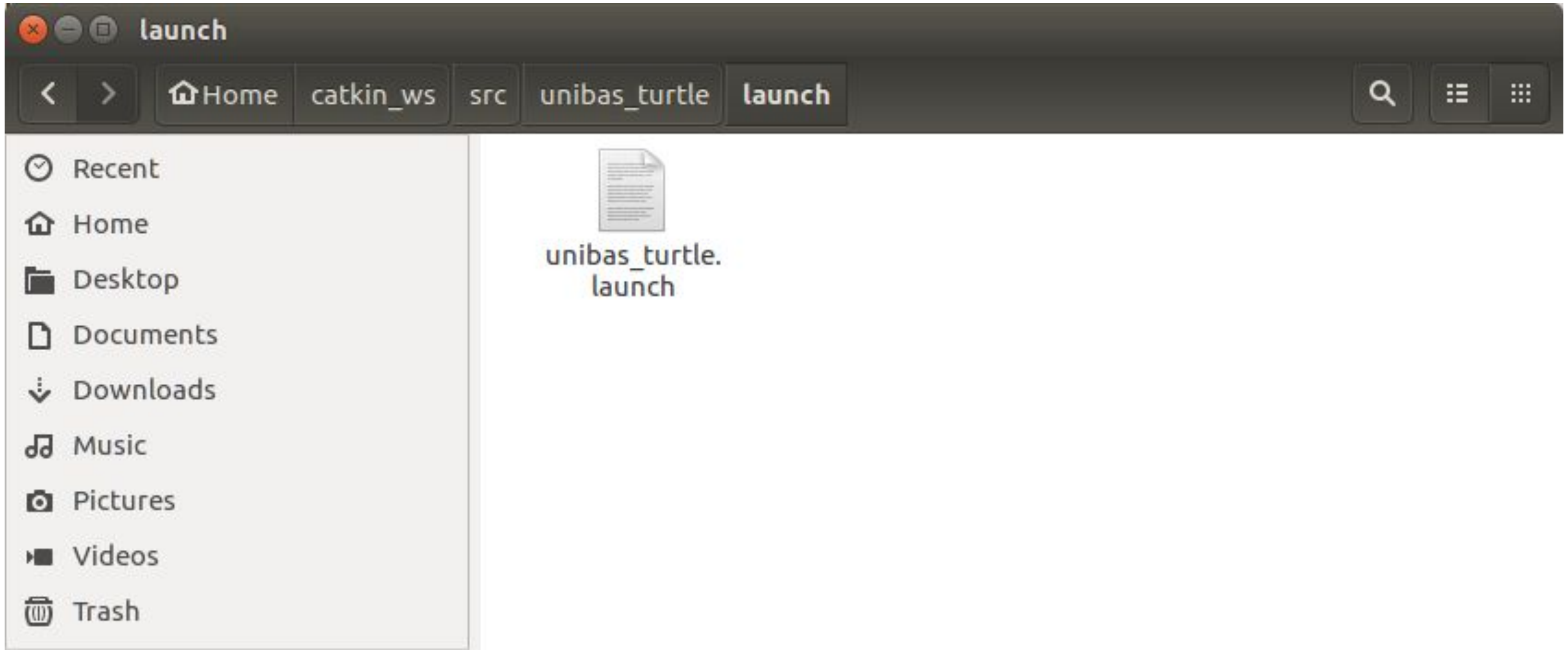
Launch file per unibas_turtle

Creiamo una cartella launch



Launch file per unibas_turtle

Creiamo un file unibas_turtle.launch dentro la cartella launch



unibas_turtle.launch

unibas_turtle.launch (~/catkin_ws/src/unibas_turtle/launch) - gedit

Open

```
<launch>
    <node name="turtlesim_node" pkg="turtlesim" type="turtlesim_node" output="screen"/>
    <node name="move" pkg="unibas_turtle" type="move.py" output="screen"/>
</launch>
```

git clone unibas_turtle

bloisi@bloisi-U36SG: ~/catkin_ws/src

```
bloisi@bloisi-U36SG:~/catkin_ws$ cd src
bloisi@bloisi-U36SG:~/catkin_ws/src$ git clone https://github.com/dbloisi/unibas_turtle.git
Cloning into 'unibas_turtle'...
remote: Enumerating objects: 37, done.
remote: Counting objects: 100% (14/14), done.
remote: Compressing objects: 100% (14/14), done.
remote: Total 37 (delta 4), reused 3 (delta 0), pack-reused 23
Unpacking objects: 100% (37/37), 14.50 KiB | 362.00 KiB/s, done.
bloisi@bloisi-U36SG:~/catkin_ws/src$
```

https://github.com/dbloisi/unibas_turtle

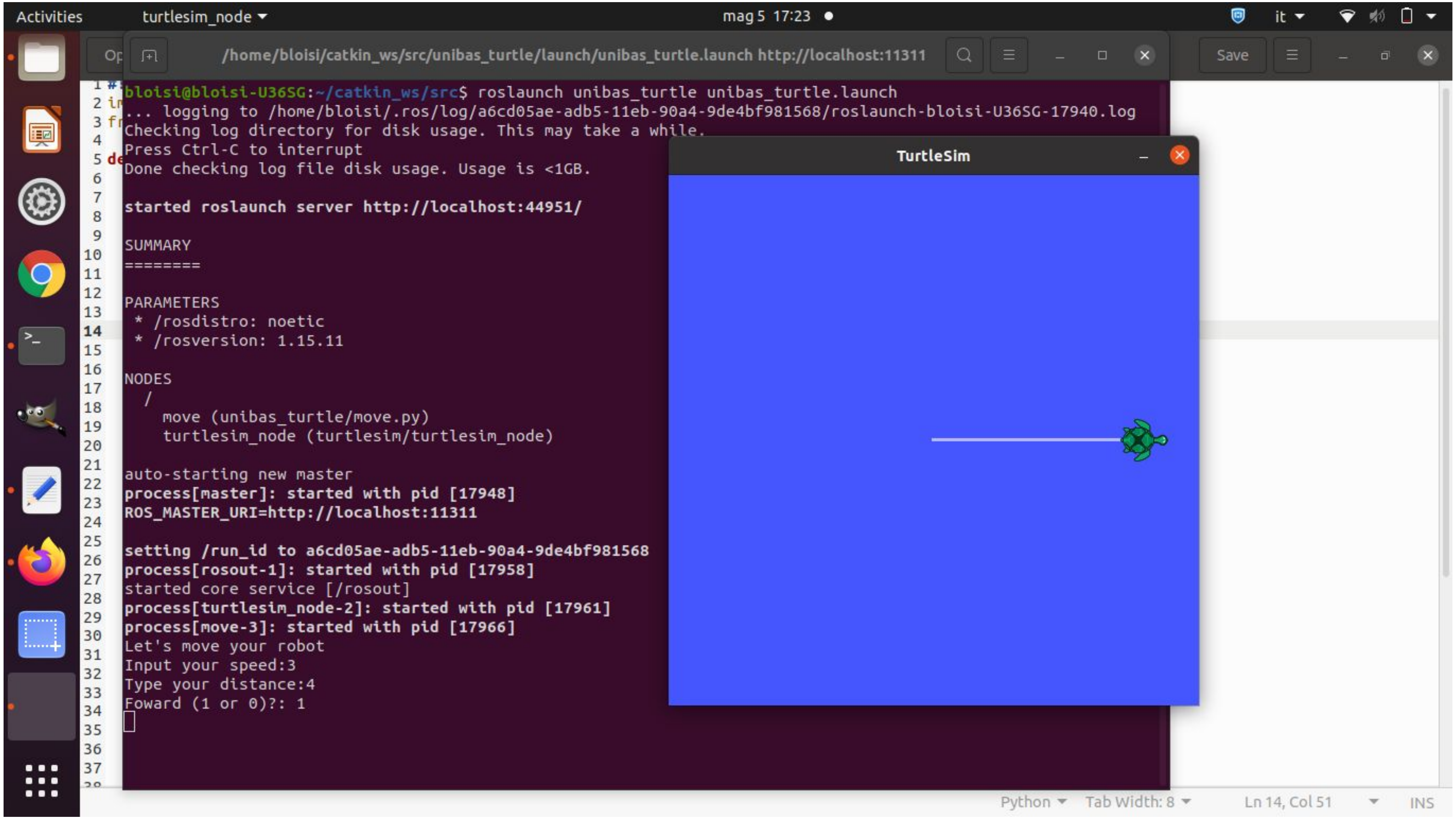
Esempio roslaunch

```
roslaunch unibas_turtle unibas_turtle.launch
```

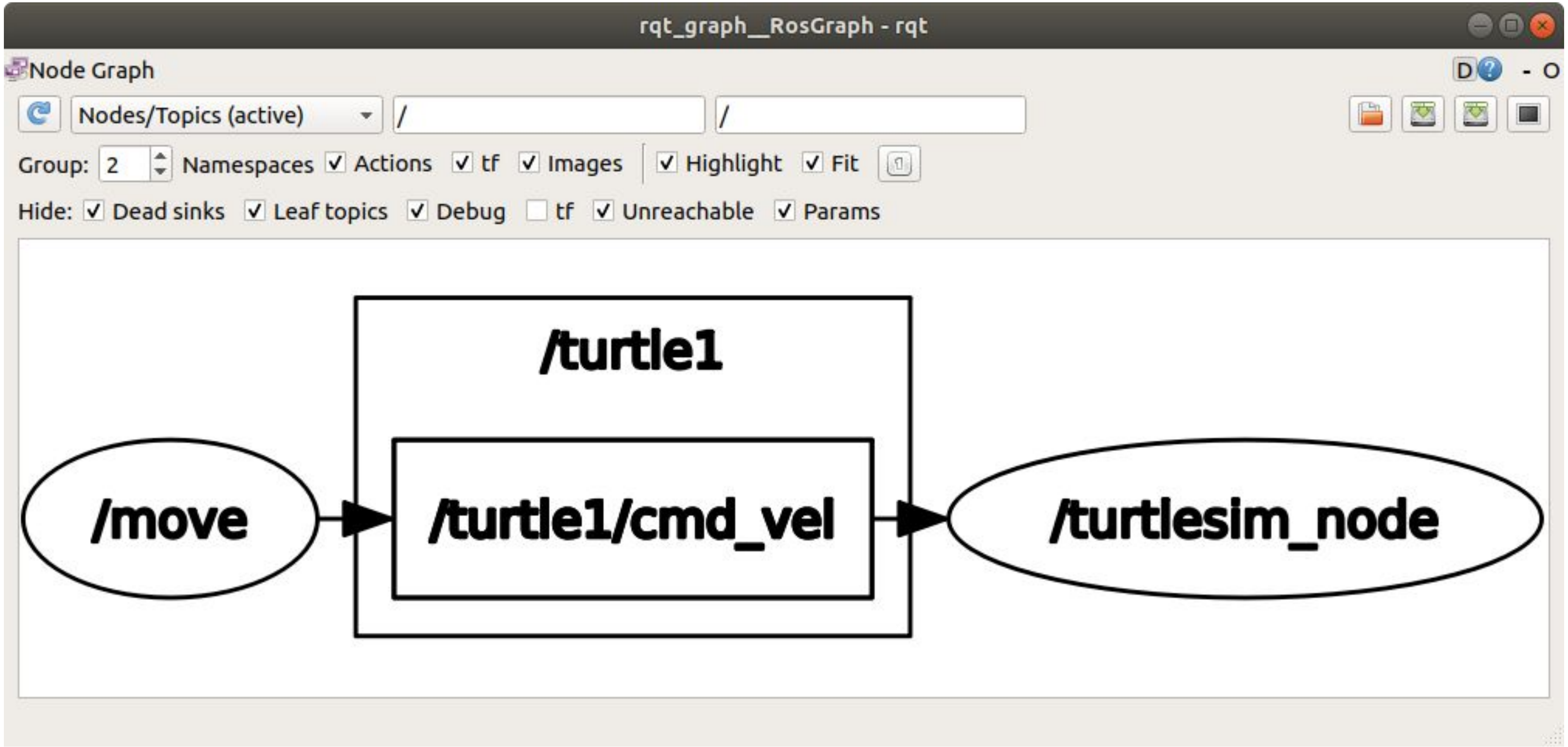
ROS package name

launch file name

Esecuzione unibas_turtle.roslaunch



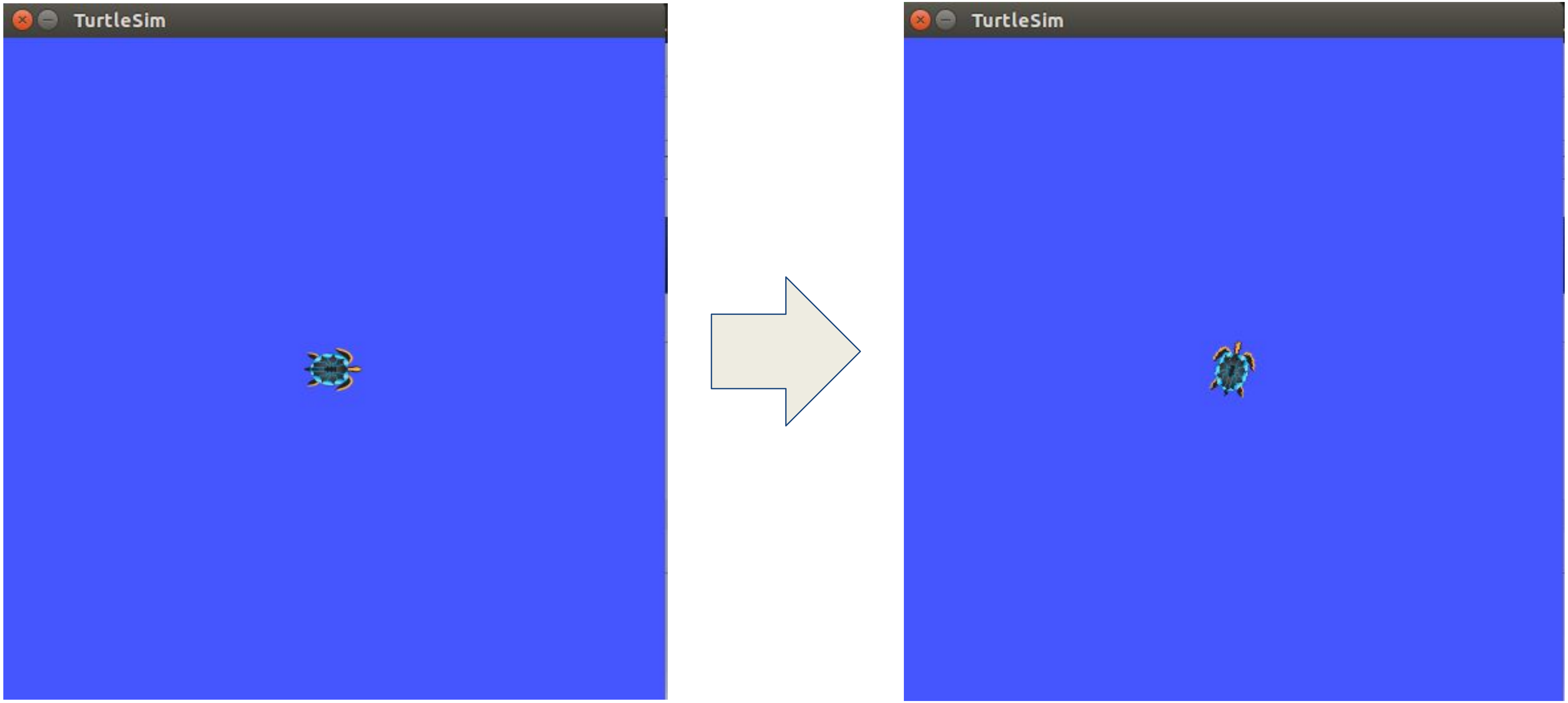
rqt_graph


rqt_graph__RosGraph - rqt
Node Graph
D
- O
Nodes/Topics (active)
/
/
Group: 2
Namespaces
Actions
tf
Images
Highlight
Fit
Hide:
Dead sinks
Leaf topics
Debug
tf
Unreachable
Params
/turtle1
/move
/turtle1/cmd_vel
/turtlesim_node

topic e message

```
bloisi@bloisi-U36SG: ~
File  Edit  View  Search  Terminal  Help
bloisi@bloisi-U36SG:~$ rostopic info /turtle1/cmd_vel
Type: geometry_msgs/Twist

Publishers:
 * /move (http://localhost:38655/)

Subscribers:
 * /turtlesim_node (http://localhost:42133/)


bloisi@bloisi-U36SG:~$ rosmsg show geometry_msgs/Twist
geometry_msgs/Vector3 linear
  float64 x
  float64 y
  float64 z
geometry_msgs/Vector3 angular
  float64 x
  float64 y
  float64 z

bloisi@bloisi-U36SG:~$ 
```

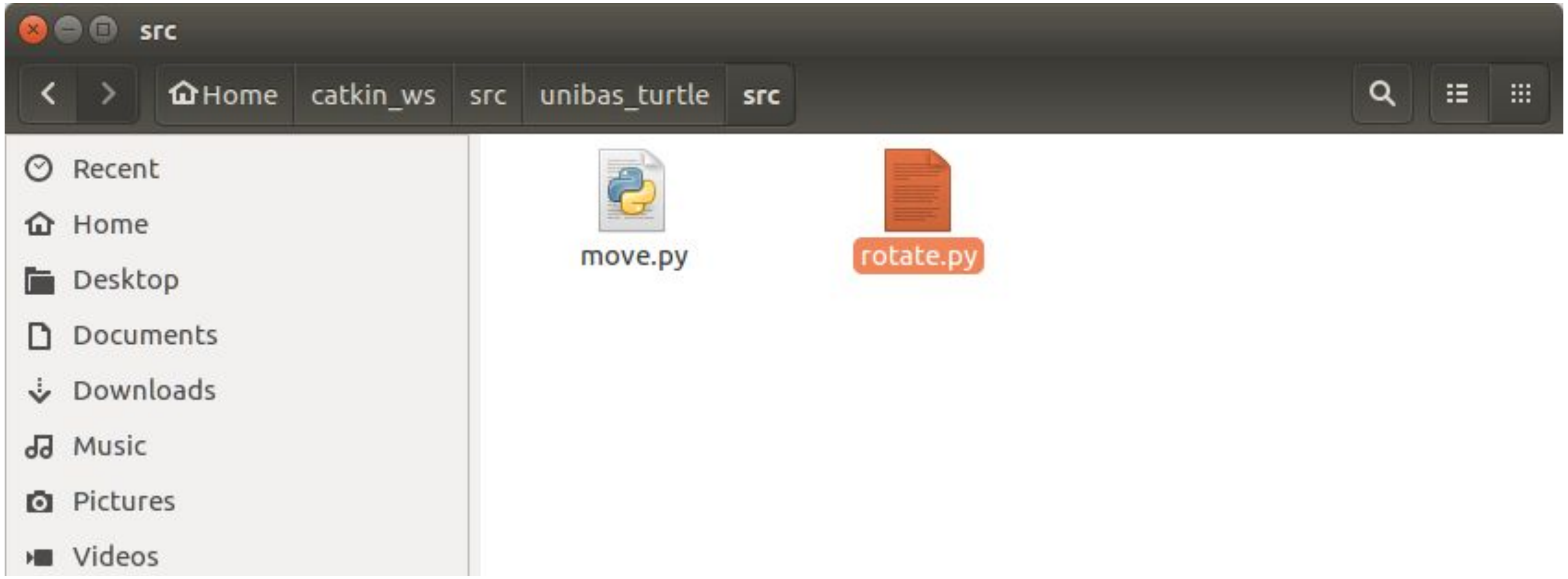
Rotating left and right



http://wiki.ros.org/turtlesim/Tutorials/Rotating%20Left%20and%20Right

creazione di rotate.py

Creiamo un file rotate.py dentro la cartella src



http://wiki.ros.org/turtlesim/Tutorials/Rotating%20Left%20and%20Right

rotate.py

```
#!/usr/bin/env python
import rospy
from geometry_msgs.msg import Twist
PI = 3.1415926535897

def rotate():
    #Starts a new node
    rospy.init_node('rotate', anonymous=True)
    velocity_publisher = rospy.Publisher('/turtle1/cmd_vel', Twist, queue_size=10)
    vel_msg = Twist()
```

rotate.py

```
# Receiveing the user's input
print("Let's rotate your robot")
speed = float(input("Input your speed (degrees/sec):"))
angle = float(input("Type your distance (degrees):"))
clockwise = int(input("Clockwise (1 or 0)?:")) #True or false

#Converting from angles to radians
angular_speed = speed*2*PI/360
relative_angle = angle*2*PI/360

#We wont use linear components
vel_msg.linear.x=0
vel_msg.linear.y=0
vel_msg.linear.z=0
vel_msg.angular.x = 0
vel_msg.angular.y = 0
```

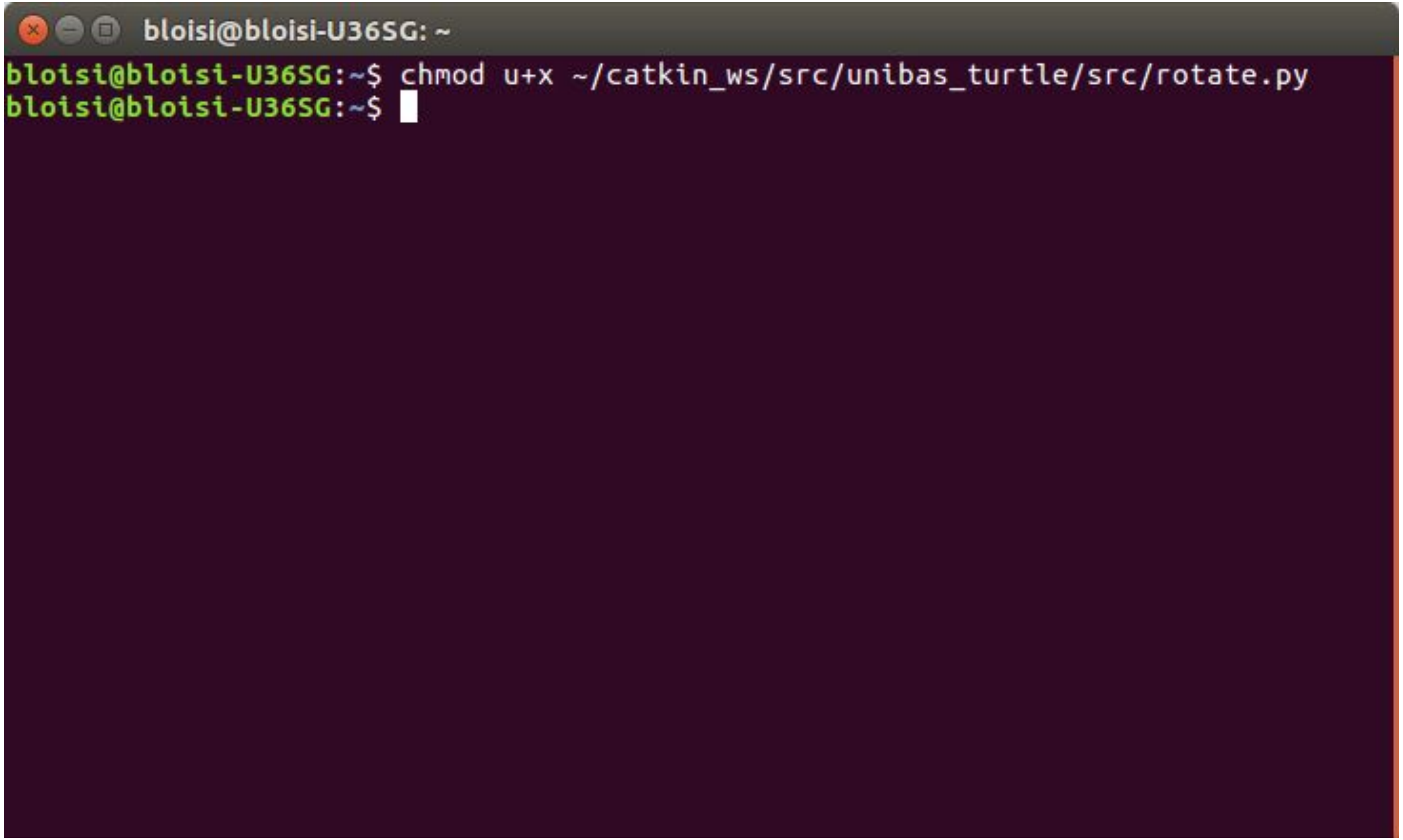
rotate.py

```
# Checking if our movement is CW or CCW
if clockwise:
    vel_msg.angular.z = -abs(angular_speed)
else:
    vel_msg.angular.z = abs(angular_speed)
# Setting the current time for distance calculus
t0 = rospy.Time.now().to_sec()
current_angle = 0

while(current_angle < relative_angle):
    velocity_publisher.publish(vel_msg)
    t1 = rospy.Time.now().to_sec()
    current_angle = angular_speed*(t1-t0)
```

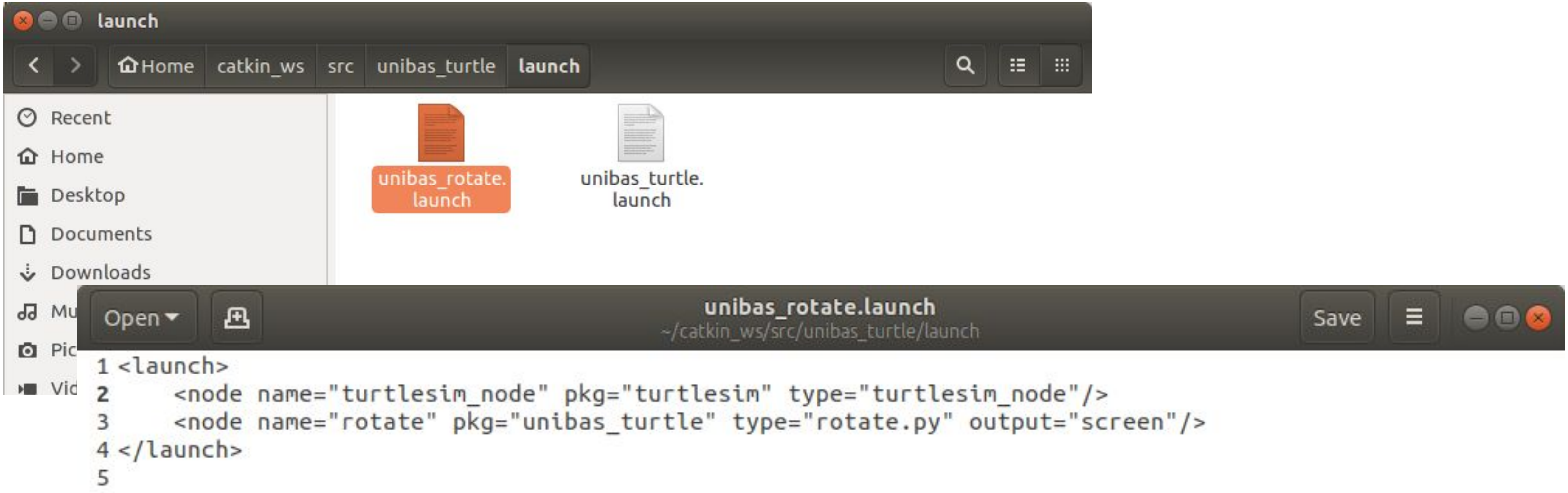
rotate.py

```
    #Forcing our robot to stop
    vel_msg.angular.z = 0
    velocity_publisher.publish(vel_msg)
    rospy.spin()

if __name__ == '__main__':
    try:
        # Testing our function
        rotate()
    except rospy.ROSInterruptException:
         pass
```

permessi per rotate.py



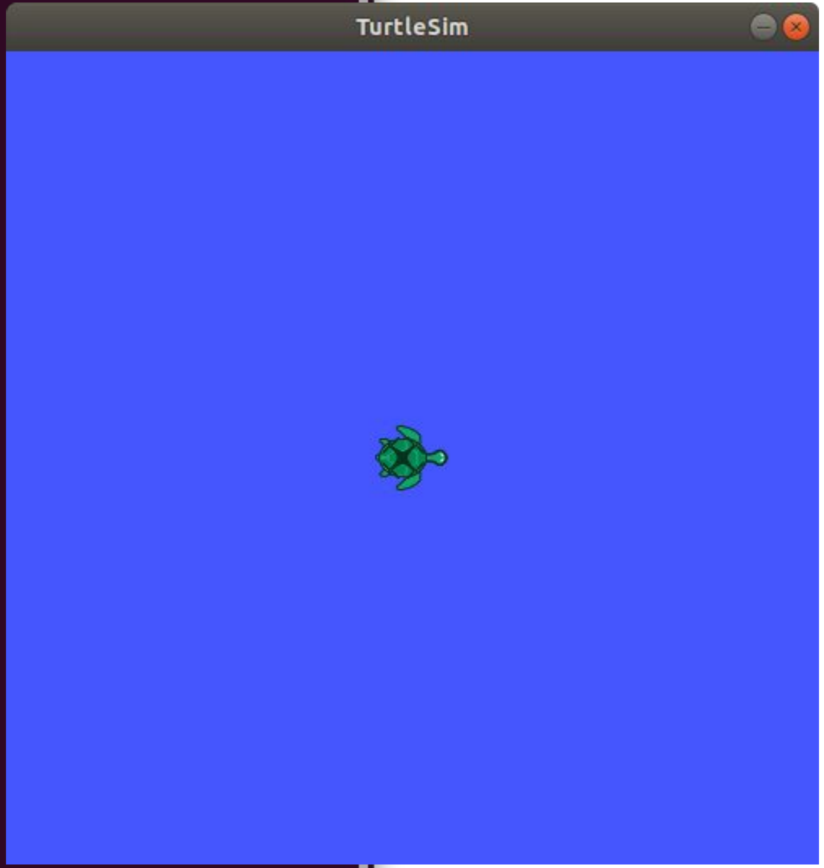
launch file per il nodo rotate



esecuzione per il nodo rotate

/home/bloisi/catkin_ws/src/unibas_turtle/launch/unibas_rotate.launch http://localhost:11311

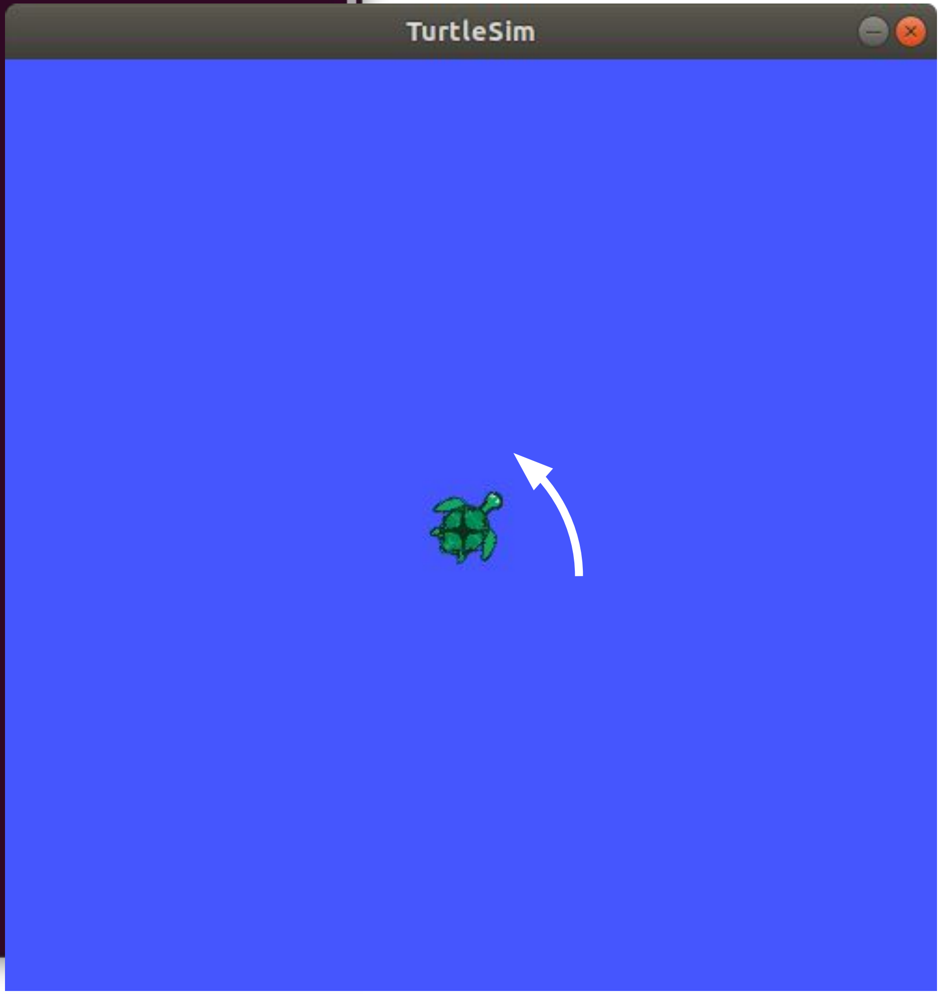
```
bloisi@bloisi-U36SG:~/catkin_ws/src$ roslaunch unibas_turtle unibas_rotate.launch
... logging to /home/bloisi/.ros/log/a6cd05ae-adb5-11eb-90a4-9de4bf981568/roslaunch-bloisi-U36SG-18522.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://localhost:34775/

SUMMARY
========

PARAMETERS
 * /rosdistro: noetic
 * /rosversion: 1.15.11

NODES
  /
    rotate (unibas_turtle/rotate.py)
    turtlesim_node (turtlesim/turtlesim_node)

ROS_MASTER_URI=http://localhost:11311

process[turtlesim_node-1]: started with pid [18536]
process[rotate-2]: started with pid [18537]
Let's rotate your robot
Input your speed (degrees/sec):3
Type your distance (degrees):40
Clockwise (1 or 0)?:0
```



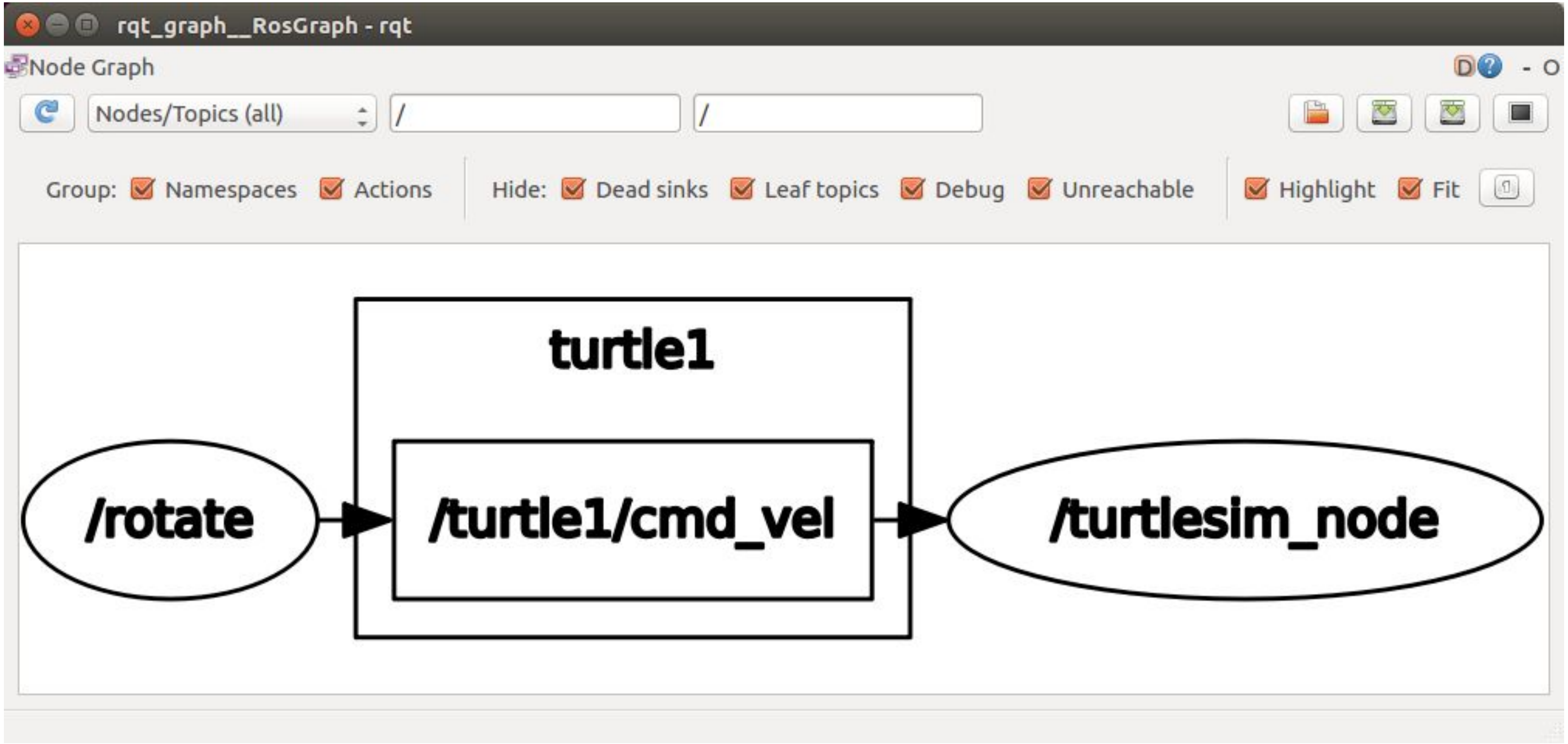
esecuzione per il nodo rotate

/home/bloisi/catkin_ws/src/unibas_turtle/launch/unibas_rotate.launch http://localhost:11311

```
bloisi@bloisi-U36SG:~/catkin_ws/src$ roslaunch unibas_turtle unibas_rotate.launch
... logging to /home/bloisi/.ros/log/a6cd05ae-adb5-11eb-90a4-9de4bf981568/roslaunch-bloisi-U36SG-18522.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://localhost:34775/

SUMMARY
========

PARAMETERS
 * /rosdistro: noetic
 * /rosversion: 1.15.11

NODES
  /
    rotate (unibas_turtle/rotate.py)
    turtlesim_node (turtlesim/turtlesim_node)

ROS_MASTER_URI=http://localhost:11311

process[turtlesim_node-1]: started with pid [18536]
process[rotate-2]: started with pid [18537]
Let's rotate your robot
Input your speed (degrees/sec):3
Type your distance (degrees):40
Clockwise (1 or 0)?:0
```



rqt_graph


rqt_graph__RosGraph - rqt
Node Graph
Nodes/Topics (all)
/
/
Group:
Namespaces
Actions
Hide:
Dead sinks
Leaf topics
Debug
Unreachable
Highlight
Fit
turtle1
/rotate
/turtle1/cmd_vel
/turtlesim_node

Il package unibas_teleop

Branch: master ▾ unibas_teleop / src / key_teleop.py / <> Jump to ▾

dbloisi first commit

0 contributors

Executable File | 91 lines (69 sloc) | 1.92 KB

```
#!/usr/bin/env python

from __future__ import print_function

import roslib; roslib.load_manifest('unibas_teleop')
import rospy

from geometry_msgs.msg import Twist

import sys, select, termios, tty

```

https://github.com/dbloisi/unibas_teleop

Il package unibas_teleop

```
msg = """
Reading from keyboard
---------------------------
Use the following keys to move the robot.
   w
a     d
   z

ESC key to quit

"""

```

https://github.com/dbloisi/unibas_teleop

Il package unibas_teleop

```
linear_ = 0.
angular_ = 0.
l_scale_ = 0.5
a_scale_ = 0.5
dirty = False

KEYCODE_R = 'd'
KEYCODE_L = 'a'
KEYCODE_U = 'w'
KEYCODE_D = 'z'

bindings = {
    KEYCODE_L:(0.0, 1.0, True),
    KEYCODE_R:(0.0, -1.0, True),
    KEYCODE_U:(1.0, 0.0, True),
    KEYCODE_D:(-1.0, 0.0, True)
}

def getKey():
    tty.setraw(sys.stdin.fileno())
    select.select([sys.stdin], [], [], 0)
    key = sys.stdin.read(1)
    termios.tcsetattr(sys.stdin, termios.TCSADRAIN, settings)
    return key

```

https://github.com/dbloisi/unibas_teleop

Il package unibas_teleop

```python
if __name__=="__main__":
    settings = termios.tcgetattr(sys.stdin)

    pub = rospy.Publisher('turtle1/cmd_vel', Twist, queue_size = 1)
    rospy.init_node('key_teleop')

    try:
        print(msg)
        run = True
        while(run):
            key = getKey()
            linear_ = 0.
            angular = 0.
            dirty = False

            if key in bindings.keys():
                linear_ = bindings[key][0]
                angular_ = bindings[key][1]
                dirty = bindings[key][2]
            elif ord(key) == 27: #ESC key
                print('quit')
                run = False
                continue
```

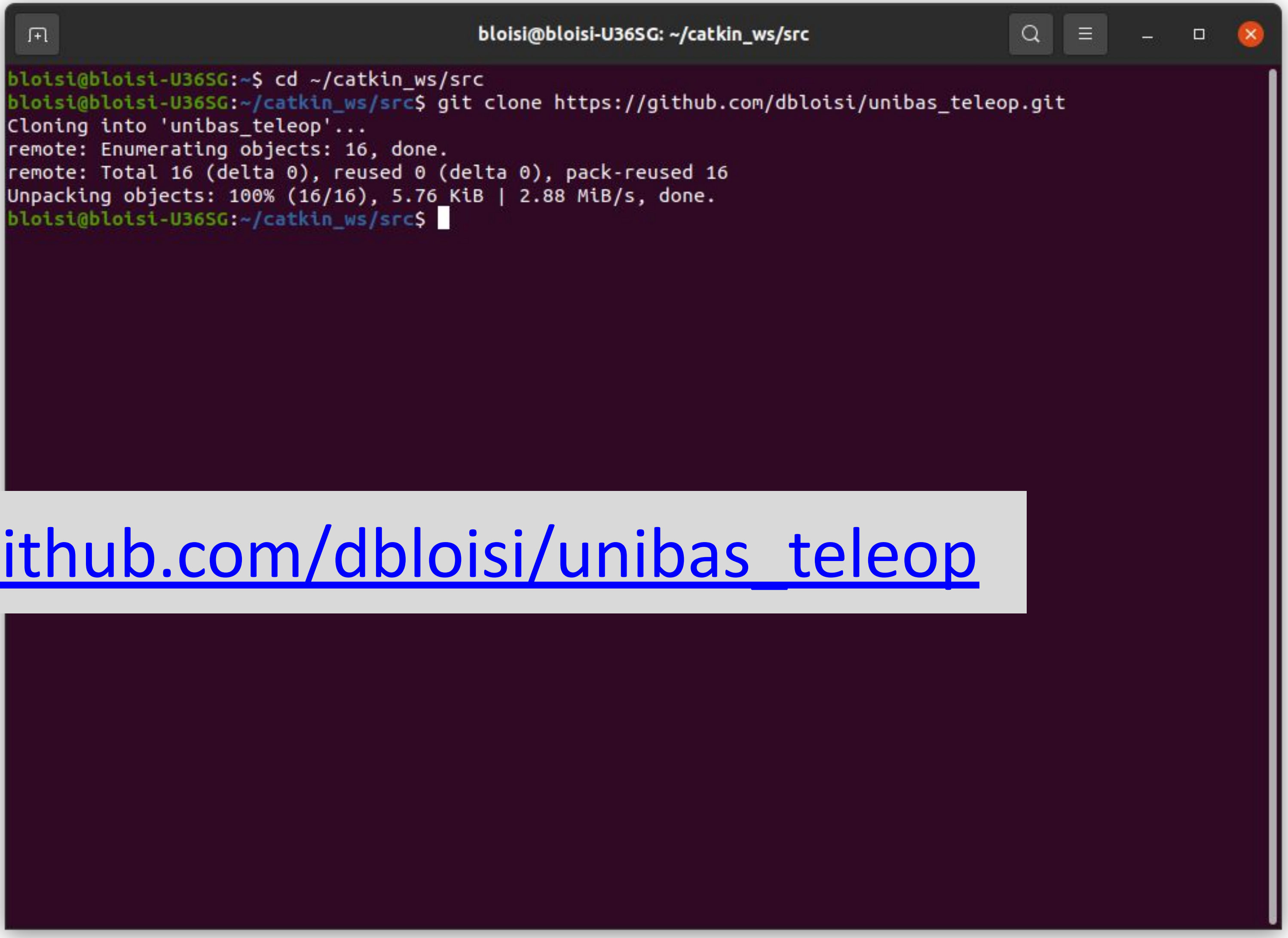
https://github.com/dbloisi/unibas_teleop

Il package unibas_teleop

```python

            twist = Twist()
            twist.linear.x = l_scale_*linear_
            twist.linear.y = 0;
            twist.linear.z = 0;
            twist.angular.x = 0;
            twist.angular.y = 0;
            twist.angular.z = a_scale_*angular_
            if dirty is True:
                pub.publish(twist)
                termios.tcsetattr(sys.stdin, termios.TCSADRAIN, settings)
                dirty = False


    except Exception as e:
        print(e)

```

https://github.com/dbloisi/unibas_teleop

git clone unibas_teleop



https://github.com/dbloisi/unibas_teleop

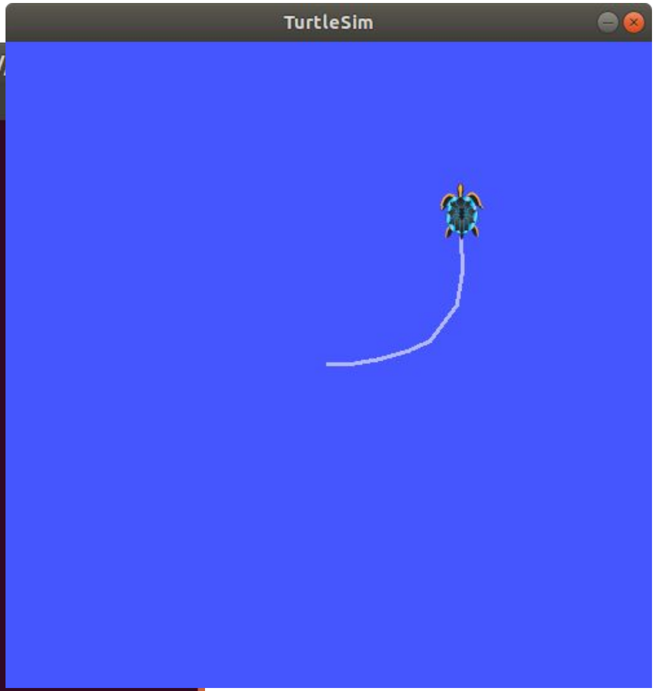
unibas_key_teleop.launch

unibas_key_teleop.launch
~/catkin_ws/src/unibas_teleop/launch

```xml
<launch>
    <node name="turtlesim_node" pkg="turtlesim" type="turtlesim_node"/>
    <node name="key_teleop" pkg="unibas_teleop" type="key_teleop.py" output="screen"/>
</launch>
```

/home/bloisi/catkin_ws/src/unibas_teleop/launch/unibas_key_teleop.launch http:/

```
    key_teleop (unibas_teleop/key_teleop.py)
    turtlesim_node (turtlesim/turtlesim_node)

auto-starting new master
process[master]: started with pid [4062]
ROS_MASTER_URI=http://localhost:11311

setting /run_id to 76a4e73a-8dee-11ea-be24-50465dde6884
process[rosout-1]: started with pid [4073]
started core service [/rosout]
process[turtlesim_node-2]: started with pid [4079]
process[key_teleop-3]: started with pid [4081]

Reading from keyboard
---------------------------
Use the following keys to move the robot.
   w
a     d
   z

ESC key to quit
```



https://github.com/dbloisi/unibas_teleop

UNIVERSITÀ DEGLI STUDI DELLA BASILICATA

Corso di Visione e Percezione

Docente

Domenico D. Bloisi

Robot mobili su ruote