

Robot Teleoperado

Clase 8
Ing. Alexander López

Teleoperado



Controlador de teclado



Nodo de teleoperación

```
alex@alex-X555LA: ~/turtlebot_ws/src
alex@alex-X555LA:~$ sudo gedit .bashrc
[sudo] password for alex:
alex@alex-X555LA:~$ cd turtlebot_ws/src/
alex@alex-X555LA:~/turtlebot_ws/src$ catkin_create_pkg turtle_teleop rospy
Created file turtle_teleop/CMakeLists.txt
Created file turtle_teleop/package.xml
Created folder turtle_teleop/src
Successfully created files in /home/alex/turtlebot_ws/src/turtle_teleop. Please
adjust the values in package.xml.
alex@alex-X555LA:~/turtlebot_ws/src$
```

Controlador de teclado

```
#!/usr/bin/env python
import sys, select, tty, termios
import rospy
from std_msgs.msg import String

if __name__ == '__main__':
    key_pub = rospy.Publisher('keys', String, queue_size=1)
    rospy.init_node("keyboard_driver")
    rate = rospy.Rate(100)
    old_attr = termios.tcgetattr(sys.stdin)
    tty.setcbreak(sys.stdin.fileno())
    print "Publishing keystrokes. Press Ctrl-C to exit..."
    while not rospy.is_shutdown():
        if select.select([sys.stdin], [], [], 0) == [sys.stdin]:
            key_pub.publish(sys.stdin.read(1))
            rate.sleep()
        termios.tcsetattr(sys.stdin, termios.TCSADRAIN, old_attr)
```

Las librerías *termios* y *tty* permiten capturar la pulsación de las teclas.

Tipo de mensaje

Establecer que se reciban los caracteres sin presionar Enter

100 Hz revisa si se ha pulsado una tecla

100 Hz revisa si se ha pulsado una tecla

Controlador de teclado

```
#!/usr/bin/env python
import sys, select, tty, termios
import rospy
from std_msgs.msg import String

if __name__ == '__main__':
    key_pub = rospy.Publisher('keys', String, queue_size=1)
    rospy.init_node("keyboard_driver")
    rate = rospy.Rate(100)
    old_attr = termios.tcgetattr(sys.stdin)
    tty.setcbreak(sys.stdin.fileno())
    print "Publishing keystrokes. Press Ctrl-C to exit..."
    while not rospy.is_shutdown():
        if select.select([sys.stdin], [], [], 0) == [sys.stdin]:
            key_pub.publish(sys.stdin.read(1))
            rate.sleep()
        termios.tcsetattr(sys.stdin, termios.TCSADRAIN, old_attr)
```

Las librerías *termios* y *tty* permiten capturar la pulsación de las teclas.

Tipo de mensaje

Establecer que se reciban los caracteres sin presionar Enter

100 Hz revisa si se ha pulsado una tecla

100 Hz revisa si se ha pulsado una tecla

Generador de movimiento


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

key_mapping = { 'w': [ 0, 1], 'x': [ 0, -1], 'a': [-1, 0], 'd': [ 1, 0], 's': [ 0, 0] }

def keys_cb(msg, twist_pub):
    if len(msg.data) == 0 or not key_mapping.has_key(msg.data[0]):
        return # unknown key
    vels = key_mapping[msg.data[0]]
    t = Twist()
    t.angular.z = vels[0]
    t.linear.x = vels[1]
    twist_pub.publish(t)
```

Tipo de mensaje: String
Tipo de mensaje: velocidades en todos los ejes

Diccionario los comandos de movimiento



Generador de movimiento

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

key_mapping = { 'w': [ 0, 1], 'x': [ 0, -1], 'a': [-1, 0], 'd': [ 1, 0], 's': [ 0, 0] }

def keys_cb(msg, twist_pub):
    if len(msg.data) == 0 or not key_mapping.has_key(msg.data[0]):
        return # unknown key
    vels = key_mapping[msg.data[0]]
    t = Twist()
    t.angular.z = vels[0]
    t.linear.x = vels[1]
    twist_pub.publish(t)
```

Función callback que recibe parámetros "msg" y "twis_pub"

Esta condicional se asegura que si no se recibe mensaje alguno o no se presiona una de las teclas del diccionario, el robot no se mueve.

Se termina la función

Generador de movimiento

```
if __name__ == '__main__':
    rospy.init_node('keys_to_twist')
    twist_pub = rospy.Publisher('cmd_vel', Twist, queue_size=1)
    rospy.Subscriber('keys', String, keys_cb, twist_pub)
    rospy.spin()
```

El nodo publica el tópic "cmd_vel" (las velocidades del móvil)

El nodo se suscribe al tópic "keys", instancia una función callback llamada "keys_cb", se usa como parámetro de entrada llamada "twist_pub".

Generador de movimiento

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

key_mapping = { 'w': [ 0, 1], 'x': [ 0, -1], 'a': [-1, 0], 'd': [ 1, 0], 's': [ 0, 0] }

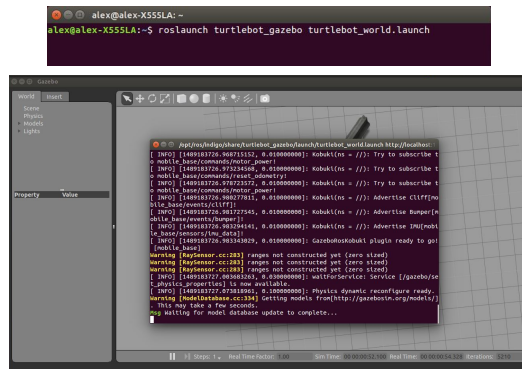
def keys_cb(msg, twist_pub):
    if len(msg.data) == 0 or not key_mapping.has_key(msg.data[0]):
        return # unknown key
    vels = key_mapping[msg.data[0]]
    t = Twist()
    t.angular.z = vels[0]
    t.linear.x = vels[1]
    twist_pub.publish(t)
```

Función callback que recibe parámetros "msg" y "twis_pub"

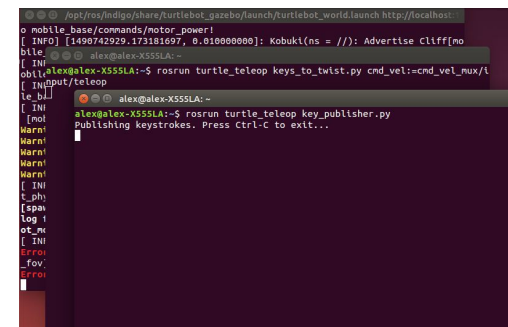
Esta condicional se asegura que si no se recibe mensaje alguno o no se presiona una de las teclas del diccionario, el robot no se mueve.

Se termina la función

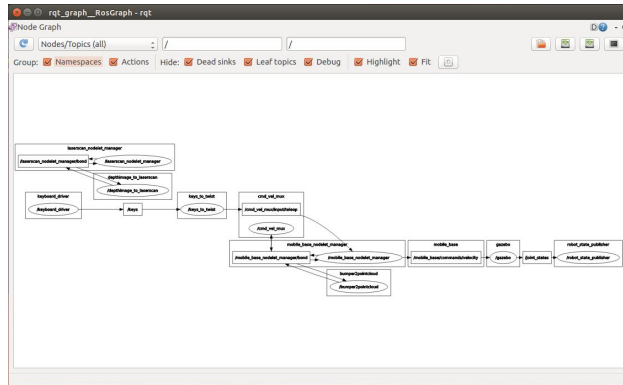
Generador de movimiento



Generador de movimiento



Generador de movimiento



Generador de movimiento

```

➤ jupyter/robidyn/jupyterlab_alex@alex@launch/turtlebot_world:launch http://localhost:
o mobile_base/commands/motor_power!
[ INFO ] [1490742929.173181697, 0.010000000]: Kobuki(ns = //): Advertise Clifffino
bille
➤ alex@alex-X55SLA:~$
[ INFO ]
obile@alex-X55SLA:~$ rosrun turtle_teleop keys-to-twist.py cmd_vel:=cmd_vel_mux/1
[ INFO ] turtle/teleop
te, id:
[ INFO ]
➤ alex@alex-X55SLA:~$
[ INFO ]
alex@alex-X55SLA:~$ rosrun turtle_teleop key_publisher.py
Publishing keystrokes. Press Ctrl-C to exit...
[ INFO ]
➤ alex@alex-X55SLA:~$
[ INFO ]
alex@alex-X55SLA:~$ rostopic list
/camera/depth/camera_info
/camera/depth/image_raw
/camera/depth/points
/camera/parameter_descriptions
/camera/parameter_updates
/camera/rgb/camera_info
/camera/rgb/image_raw
/camera/rgb/image_raw/compressed
/camera/rgb/image_raw/compressed/parameter_descriptions
/camera/rgb/image_raw/compressed/parameter_updates
/camera/rgb/image_raw/compressedDepth
/camera/rgb/image_raw/compressedDepth/parameter_descriptions
/camera/rgb/image_raw/compressedDepth/parameter_updates

```

Generador de movimiento

```
➜ ~ curl -o /dev/null -s https://raw.githubusercontent.com/alexxxa/turtlebot_world/main/launch/turtlebot_world.launch | sed 's|@|&|g' > turtlebot_world.launch
➜ ~ cat turtlebot_world.launch
[ INFO ] [1499742929.173181697, 0.0100000000]: Kobuki(ns = //): Advertise Cliff[no
    alex@alex-XSS5SLA:~$ rosrun turtle_teleop keys_to_twist.py cmd_vel:=cmd_vel_mux/I
Input/teleop
    alex@alex-XSS5SLA:~$ 
[ INFO ]
[ NOT ]
warn]
warn]
warn]
warn]
[ INFO ]
[ PH ]
[ CAP ]
log f
at_m
[ INI
Error
Fov:
Error

    alex@alex-XSS5SLA:~$ rosrun turtle_teleop key_publisher.py
Publishing keystrokes. Press Ctrl-C to exit...

    alex@alex-XSS5SLA:~$ rostopic info cmd_vel_mux/Input/teleop
Type: geometry_msgs/Twist

Publishers:
 * /keys_to_twist (http://alex-XSS5SLA:46283/)

Subscribers:
 * /mobile_base_nodelet_manager (http://alex-XSS5SLA:41811/)

    alex@alex-XSS5SLA:~$
```

Generador de movimiento

```

$ cd ~/catkin_ws/src
$ git clone https://github.com/robertnixon/turtlebot_azebo.launch
$ cd ..
$ catkin build
$ source ~/catkin_ws/devel/setup.sh
$ roslaunch turtlebot_azebo.launch http://localhost:
no mobile_base/commands/motor_power!
[ INFO] [1490742029.173181607, 0.010000000]: Kobuki(ns = //): Advertise Cliff[no
bille
$ alias alex=alex.XSS5SLA-
$ alex
no mobile_base/commands/motor_power!
$ alex=alex.XSS5SLA-$ rosrun turtle_teleop keys_to_twist.py cmd_vel:=cmd_vel_mux/1
[ INFO] /teleop
t_twist
$ alias alex=alex.XSS5SLA-
$ alex
no mobile_base/commands/motor_power!
$ alex=alex.XSS5SLA-$ rosrun turtle_teleop key_publisher.py
Publishing keystrokes. Press Ctrl+C to exit...
$
$ alias alex=alex.XSS5SLA-
$ alex
no mobile_base/commands/motor_power!
$ alex=alex.XSS5SLA-$ rostopic show geometry_msgs/Twist
geometry_msgs/Vector3 linear
float64 x
float64 y
float64 z
geometry_msgs/Vector3 angular
float64 x
float64 y
float64 z
$ alias alex=alex.XSS5SLA-
$ alex
no mobile_base/commands/motor_power!
$ alex=alex.XSS5SLA-$

```

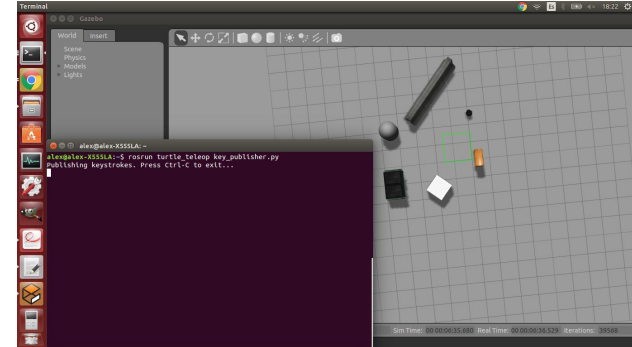
Generador de movimiento

```

alex@alex-X555LA:~$ rosrun turtle_teleop keys_to_twist.py cmd_vel:=cmd_vel_mux/l
[ INFO] [1490742929.173181697, 0.010000000]: Kobuki(ns = //): Advertise Clifff(mo
ble
alex@alex-X555LA:~$ rosrun turtle_teleop key_publisher.py
Publishing keystrokes. Press Ctrl-C to exit...
alex@alex-X555LA:~$ rostopic hz cmd_vel_mux/input/teleop
subscribed to [/cmd_vel_mux/input/teleop]
WARNING: may be using simulated time
average rates: 1.563
min: 0.640s max: 0.640s std dev: 0.00000s window: 2
average rate: 1.220
min: 0.640s max: 1.000s std dev: 0.18000s window: 3
no new messages
no new messages
no new messages
average rate: 0.703
min: 0.320s max: 3.730s std dev: 1.35378s window: 5
average rate: 1.189
min: 0.110s max: 3.730s std dev: 1.12347s window: 9
no new messages

```

Generador de movimiento



Generador de movimiento con hz

```

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

```

```

key_mapping = { 'w': [ 0, 1], 'x': [ 0, -1], 'a': [-1, 0], 'd': [ 1, 0], 's': [ 0, 0] }
g_last_twist = None

```

```

def keys_cb(msg, twist_pub):
    global g_last_twist
    if len(msg.data) == 0 or not key_mapping.has_key(msg.data[0]):
        return # unknown key
    vels = key_mapping[msg.data[0]]
    g_last_twist.angular.z = vels[0]
    g_last_twist.linear.x = vels[1]
    twist_pub.publish(g_last_twist)

```

Función callback que
recibe parámetros
"msg" y "twis_pub"

Esta condicional se asegura que
si no se recibe mensaje alguno o
no se presiona una de las teclas
del diccionario, el robot no se
mueve.

Se termina la función

Generador de movimiento con hz

```

if __name__ == '__main__':
    rospy.init_node('keys_to_twist')
    twist_pub = rospy.Publisher('cmd_vel', Twist, queue_size=1)
    rospy.Subscriber('keys', String, keys_cb, twist_pub)
    rate = rospy.Rate(10)
    g_last_twist = Twist() # initializes to zero
    while not rospy.is_shutdown():
        twist_pub.publish(g_last_twist)
        rate.sleep()

```

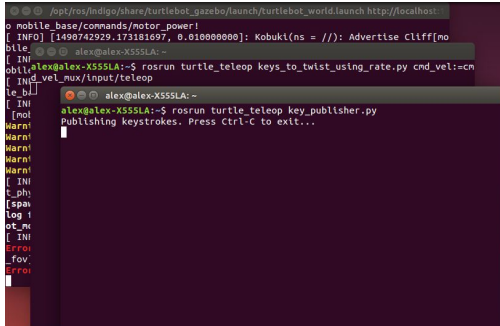
Publicar en el tópico
"cmd_vel" al cual se suscribe
el robot

Suscribirse al tópico "keys"
que publica el nodo
Keyboard_driver

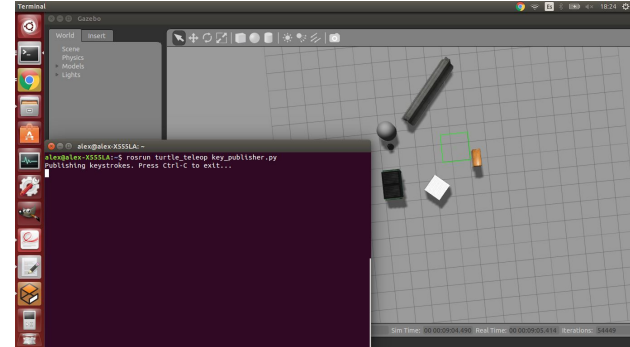
Se publica los valores
iniciados en cero

El proceso se realiza
a 10 Hz

Generador de movimiento con hz



Generador de movimiento con hz



Parametros de servidor

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

```
key_mapping = {'w': [0, 1], 'x': [0, -1], 'a': [-1, 0], 'd': [1, 0], 's': [0, 0]}
g_last_twist = None
g_vel_scales = [0.1, 0.1] # default to very slow
```

- Se especifica los parámetros de velocidad para el robot

```
def keys_cb(msg, twist_pub):
```

- Variables como globales que se usan en distintas funciones

```
global g_last_twist, g_vel_scales
if len(msg.data) == 0 or not key_mapping.has_key(msg.data[0]):
    return msg.unknown_key
vels = key_mapping[msg.data[0]]
g_last_twist.angular.z = vels[0] * g_vel_scales[0]
g_last_twist.linear.x = vels[1] * g_vel_scales[1]
twist_pub.publish(g_last_twist)
```

Esta condicional se asegura que si no se recibe mensaje alguno o no se presiona una de las teclas del diccionario, el robot no se mueve.

Parametros de servidor

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

```
key_mapping = {'w': [0, 1], 'x': [0, -1], 'a': [-1, 0], 'd': [1, 0], 's': [0, 0]}
g_last_twist = None
g_vel_scales = [0.1, 0.1] # default to very slow
```

- Variables como globales que se usan en distintas funciones

```
def keys_cb(msg, twist_pub):
```

```
global g_last_twist, g_vel_scales
if len(msg.data) == 0 or not key_mapping.has_key(msg.data[0]):
```

Se recibe las velocidades del
nodo keyboard_driver

```

return # unknown key
vels = key_mapping[msg.data[0]]
g_last_twist.angular.z = vels[0] * g_vel_scales[0]

```

Aplicar los parámetros a la
velocidad angular y lineal

```
g_last_twist.linear.x = vels[1] * g_vel_scales[1]
twist_pub.publish(g_last_twist)
```

Publicar el mensaje "Twist"
en el tópico "cmd_vel"

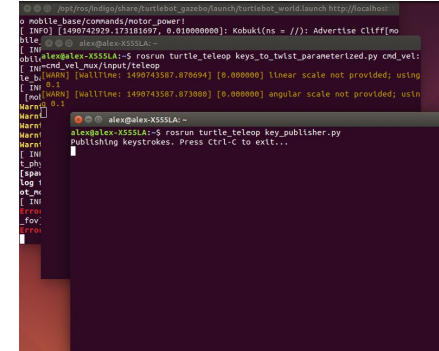
Parametros de servidor

```

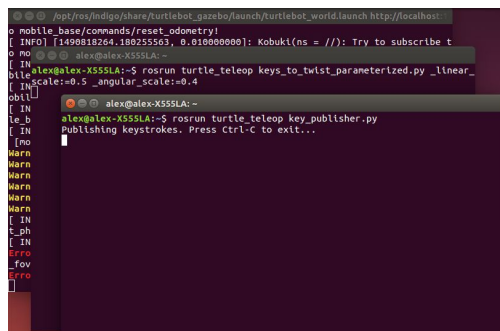
if __name__ == '__main__':
    rospy.init_node('keys_to_twist')
    twist_pub = rospy.Publisher('cmd_vel', Twist, queue_size=1)
    rospy.Subscriber('keys', String, keys_cb, twist_pub)
    g_last_twist = Twist() # initializes to zero
    if rospy.has_param('~linear_scale'):
        g_vel_scales[1] = rospy.get_param('~linear_scale')
    else:
        rospy.logwarn("linear scale not provided; using %.1f" % g_vel_scales[1])
    if rospy.has_param('~angular_scale'):
        g_vel_scales[0] = rospy.get_param('~angular_scale')
    else:
        rospy.logwarn("angular scale not provided; using %.1f" % g_vel_scales[0])
    rate = rospy.Rate(10)
    while not rospy.is_shutdown():
        twist_pub.publish(g_last_twist)
        rate.sleep()

```

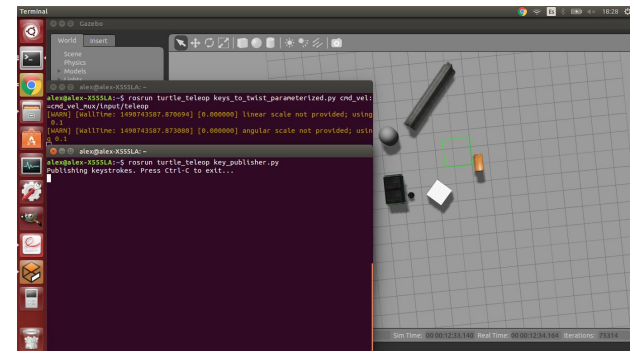
Parametros de servidor



Parametros de servidor



Parametros de servidor



Rampas de velocidad

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

```
key_mapping = {'w': [0, 1], 'x': [0, -1], 'a': [-1, 0], 'd': [1, 0], 's': [0, 0]}
g_twist_pub = None
g_target_twist = None
g_last_twist = None
g_last_send_time = None
g_vel_scales = [0.1, 0.1] # default to very slow
g_vel_ramps = [1, 1] # units: meters per second^2
```

← Iniciar variables con valor cero

← Aceleración del móvil

Rampas de velocidad

```
def ramped_vel(v_prev, v_target, t_prev, t_now, ramp_rate):
    # compute maximum velocity step
    step = ramp_rate * (t_now - t_prev).to_sec()
    sign = 1.0 if (v_target > v_prev) else -1.0
    error = math.fabs(v_target - v_prev)
    if error < step: # we can get there within this timestep-we're done.
        return v_target
    else:
        return v_prev + sign * step # take a step toward the target
```

← Función que implementa la rampa de velocidad

```
def ramped_twist(prev, target, t_prev, t_now, ramps):
    tw = Twist()
    tw.angular.z = ramped_vel(prev.angular.z, target.angular.z, t_prev, t_now, ramps[0])
    tw.linear.x = ramped_vel(prev.linear.x, target.linear.x, t_prev, t_now, ramps[1])
    return tw
```

← Implementa la rampa en el mensaje tipo "Twist"

Rampas de velocidad

```
def ramped_vel(v_prev, v_target, t_prev, t_now, ramp_rate):
    # compute maximum velocity step
    step = ramp_rate * (t_now - t_prev).to_sec()
    sign = 1.0 if (v_target > v_prev) else -1.0
    error = math.fabs(v_target - v_prev)
    if error < step: # we can get there within this timestep-we're done.
        return v_target
    else:
        return v_prev + sign * step # take a step toward the target
```

← Función que implementa la rampa de velocidad

```
def ramped_twist(prev, target, t_prev, t_now, ramps):
    tw = Twist()
    tw.angular.z = ramped_vel(prev.angular.z, target.angular.z, t_prev, t_now, ramps[0])
    tw.linear.x = ramped_vel(prev.linear.x, target.linear.x, t_prev, t_now, ramps[1])
    return tw
```

← Variable tipo "Twist"

← Retorna el resultado

← Llama la función ramped_vel para aplicar la rampa con la velocidad lineal y angular

Rampas de velocidad

```
def send_twist():
    global g_last_twist_send_time, g_target_twist, g_last_twist, \
           g_vel_scales, g_vel_ramps, g_twist_pub
    t_now = rospy.Time.now()
    g_last_twist = ramped_twist(g_last_twist, g_target_twist,
                                g_last_twist_send_time, t_now, g_vel_ramps)
    g_last_twist_send_time = t_now
    g_twist_pub.publish(g_last_twist)
```

← Función que publica el mensaje tipo Twist en el nodo

← Inicia un conteo del tiempo

← Llama a la función rampa con datos de entrada como los parámetros de entrada y el tiempo actual

← El contador de tiempo lo registra como una variable para usarlo la próxima vez que se llame la función

```
def keys_cb(msg):
    global g_target_twist, g_last_twist, g_vel_scales
    if len(msg.data) == 0 or not key_mapping.has_key(msg.data[0]):
        return # unknown key
    vels = key_mapping[msg.data[0]]
    g_last_twist.angular.z = vels[0] * g_vel_scales[0]
    g_last_twist.linear.x = vels[1] * g_vel_scales[1]
```

← Función conocida, recibe la data del nodo Keyboard_drive

Rampas de velocidad

```
def fetch_param(names, default):
    if rospy.has_param(name):
        return rospy.get_param(name)
    else:
        print "parameter [%s] not defined" % name
        return default
```

Función que define los parámetros del nodo:

- Velocidad lineal y angular
- Aceleración lineal y angular

En caso no se defina el parámetro, se define los variables por default

Rampas de velocidad

```
if __name__ == '__main__':
    rospy.init_node('keys_to_twist')
    g_last_twist_send_time = rospy.Time.now()
    twist_pub = rospy.Publisher('cmd_vel', Twist, queue_size=1)
    rospy.Subscriber('keys', String, keys_cb)
    g_target_twist = Twist() # initializes to zero
    g_last_twist = Twist()
    g_vel_scales[0] = fetch_param('~angular_scale', 0.1)
    g_vel_scales[1] = fetch_param('~linear_scale', 0.1)
    g_vel_ramps[0] = fetch_param('~angular_accel', 1.0)
    g_vel_ramps[1] = fetch_param('~linear_accel', 1.0)

    rate = rospy.Rate(20)
    while not rospy.is_shutdown():
        send_twist()
        rate.sleep()
```

Establece los parámetros del nodo:

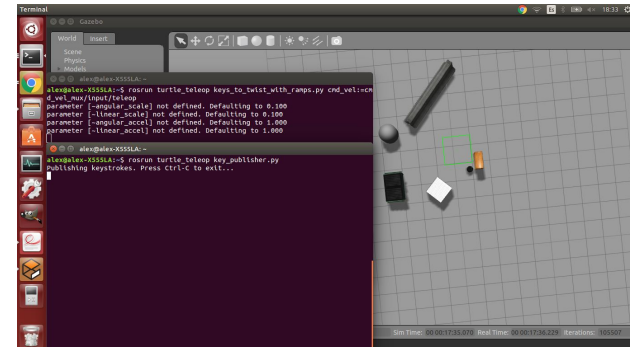
- Máximo de velocidad lineal y angular
- Aceleración lineal y angular

Transmisión a 20 hz

Rampas de velocidad

[illegible]

Rampas de velocidad



Rampas de velocidad

```
alex@alex-X555LA: ~/ros/indigo/share/turtlebot_gazebo/launch$ roslaunch turtlebot_world.launch http://localhost:
o mobile_base/commands/reset_odometry!
[ INFO ] [1490819266.666108486, 0.010000000]: Kobuki(ns = //): Try to subscribe t
o mobi
[ INFO ] alex@alex-X555LA: ~
ble alex@alex-X555LA:~$ roslaunch turtle_teleop keys_to_twist_with_ramps.py _linear_sca
[ INFO ] _linear_scale:=0.5 _angular_scale:=1.0 _linear_accel:=1.0 _angular_accel:=1.0
obile
[ INFO ] alex@alex-X555LA: ~
alex@alex-X555LA:~$ roslaunch turtle_teleop key_publisher.py
Publishing keystrokes. Press Ctrl-C to exit...
Warntr
Warntr
Warntr
Warntr
[ INFO ]
t_phys:
[spawn
log fi
ot_moc
[ INFO ]
```

¡Gracias!

¡La única pregunta tonta es la que no se hace!