Robot Teleoperado

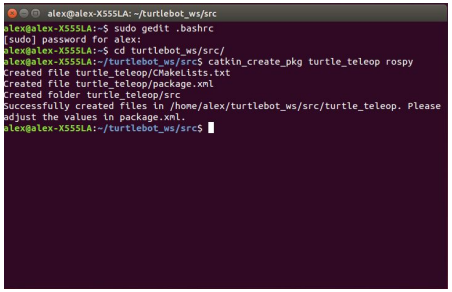
Clase 8

Ing. Alexander López

Teleoperado

Controlador de teclado

Nodo de teleoperación



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)[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)[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 pública el tópico "cmd_vel" (las velocidades del móvil)

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

Generador de movimiento

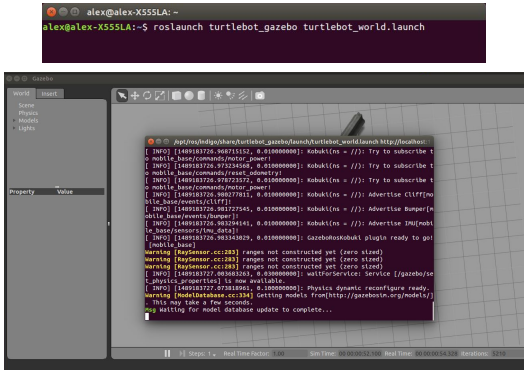
```
#!/usr/bin/env python
import rospy
from std_msgs.msg import String
from geometry_msgs.msg import Twist

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"

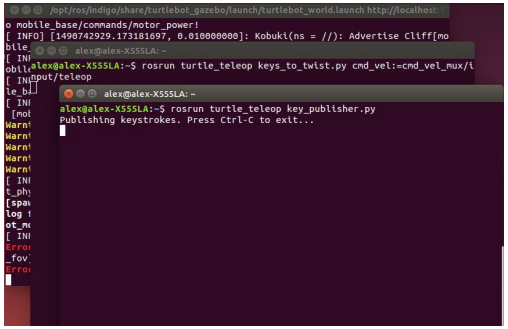
Esta condicional se asegura que si no se recibe mensaje alguno o no se presiona una de las teclas del diccionario, el robot no se mueve.

Se termina la función

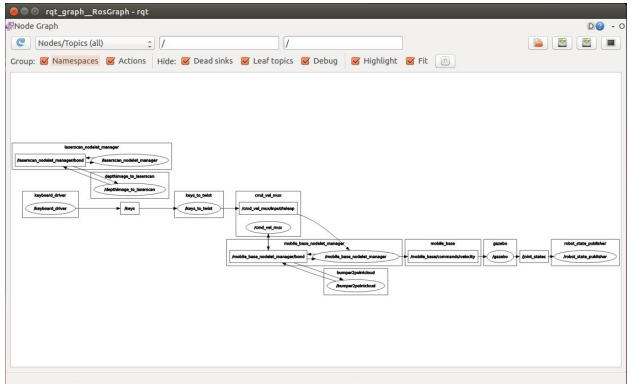
Generador de movimiento



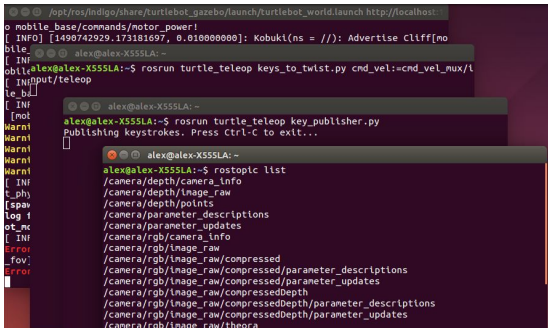
Generador de movimiento



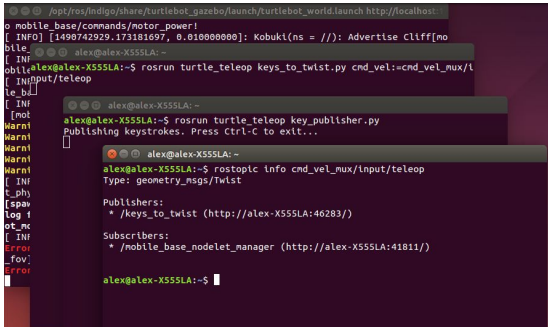
Generador de movimiento



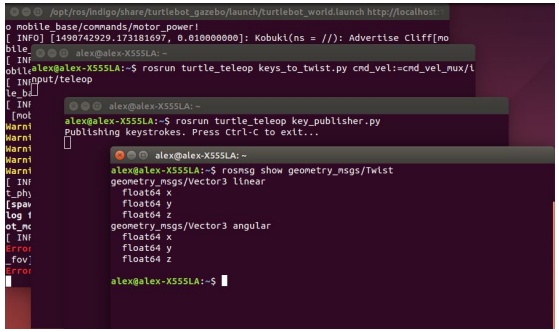
Generador de movimiento



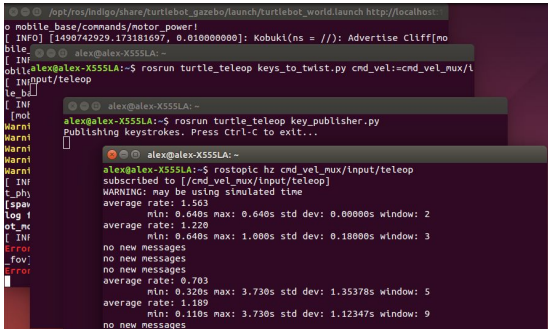
Generador de movimiento



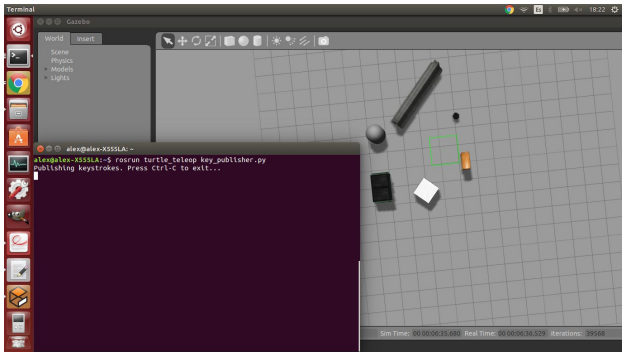
Generador de movimiento



Generador de movimiento



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()
```

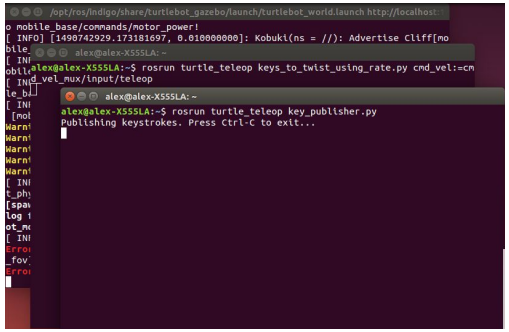
Publicar en el tópico “cmd_vel” al cual se suscribe el robot

Suscribirse al tópico “keys” que publica el nodo Keyboard_driver

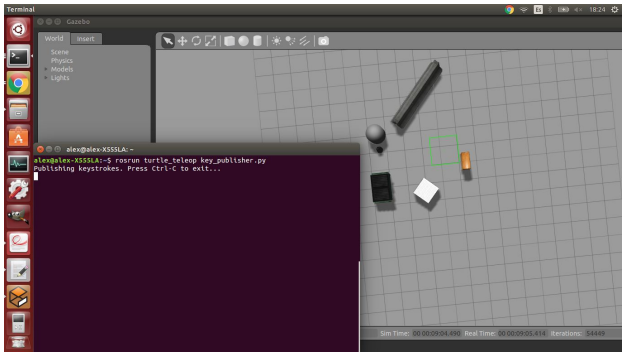
Se publica los valores iniciados en cero

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

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]):
        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]
    twist_pub.publish(g_last_twist)
```

Se especifica los parámetros de velocidad para el robot

Variables como globales que se usan en distintas funciones

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

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]):
        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]
    twist_pub.publish(g_last_twist)
```

Variables como globales que se usan en distintas funciones

Se recibe las velocidades del nodo keyboard_driver

Aplicar los parámetros a la velocidad angular y lineal

Publicar el mensaje “Twist” en el tópico “cmd_vel”

Parametros de servidor

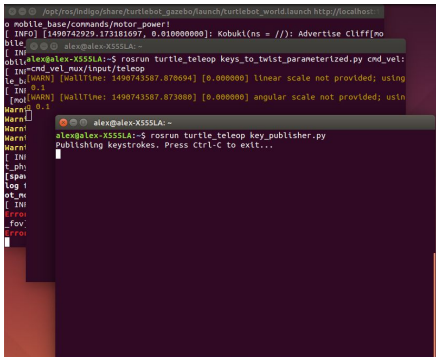
```
if __name__ == '__main__':
    rospy.init_node('keys_to_twist')
    twist_pub = rospy.Publisher('cmd_vel', Twist, queue_size=1)
    rospy.Subscriber('keys', String, keys_cb, twist_pub)
    g_last_twist = Twist() # initializes to zero
    if rospy.has_param('~linear_scale'):
        g_vel_scales[1] = 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()
```

Conexiones entre nodos

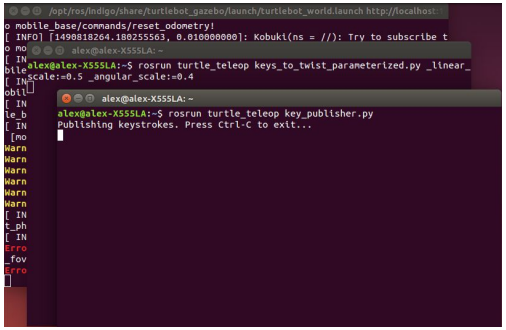
Si se ingresa un parámetro con el nombre “linear_scale”

Si se ingresa un parámetro con el nombre “angular_scale”

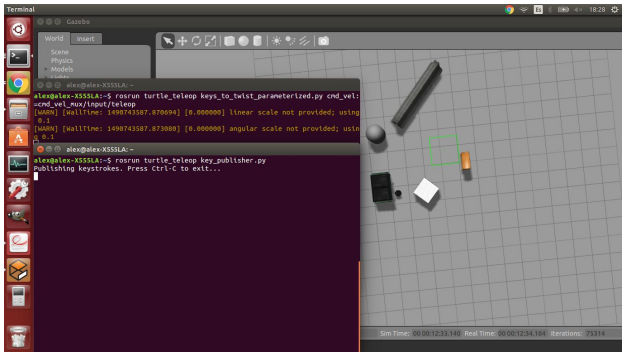
Parametros de servidor



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


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
```

Función que implementa la rampa de velocidad

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


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
```

Función que implementa la rampa de velocidad

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)

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 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

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. Defaulting to %.3f" % (name, default)
        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

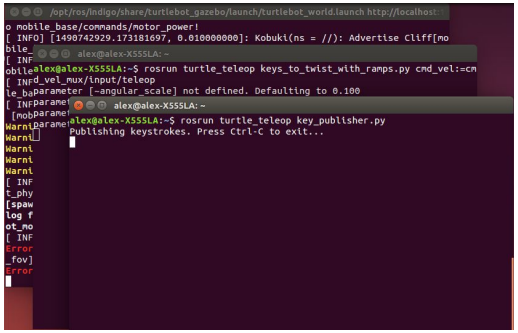
```
if __name__ == '__main__':
    rospy.init_node('keys_to_twist')
    g_last_twist_send_time = rospy.Time.now()
    twist_pub = rospy.Publisher('cmd_vel', Twist, queue_size=1)
    rospy.Subscriber('keys', String, keys_cb)
    g_target_twist = Twist() # initializes to zero
    g_last_twist = Twist()
    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:

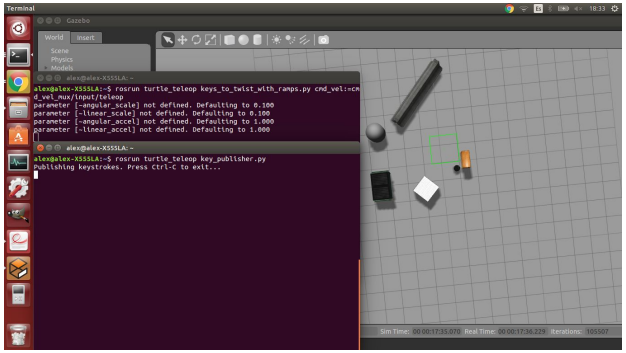
- Máximo de velocidad lineal y angular
- Aceleración lineal y angular

Transimisión a 20 hz

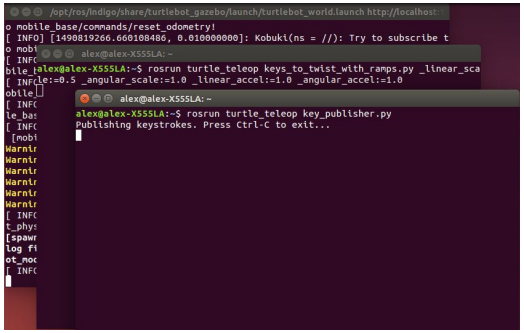
Rampas de velocidad



Rampas de velocidad



Rampas de velocidad



¡Gracias!

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