Ciencia de la Computación

Campos Potenciales

LARVIC

Ximena Pocco Lozada

Angel Sucapuca Diaz

Karelia Vilca Salinas

4to Semestre

11-2015

"Los alumnos declaran haber realizado el presente trabajo de acuerdo a las normas de la Universidad Católica San Pablo"

Resumen

Informe del grupo encargado del proyecto Campos Potenciales en el cual se explicarán los objetivos alcanzados, los problemas que surgieron durante el desarrollo de las tareas asi como sus soluciones encontradas y se mencionaran los problemas, tareas y objetivos que faltan realizar.

Contenidos

Capítulo 1

Introducción

Un comportamiento humano trivial es el hecho de transportarse, pero para un robot acciones simples como estas pueden ser más complejas ya que el individuo debe conocer su posición global para poder definir su trayectoria, pero en un camino estandar los obstáculos son impresindibles y el robot debería ser capáz de evitar colisiones. Entonces parte la necesidad de elaborar estrategias para cubrir esta necesidad de manera eficiente.

Este podría considerar un camino virtual o buscar un objetivo para guiar su transcurso, para el camino virtual, este debería ser definido.

En el presente trabajo se buscará manejar mejor las trayectorias y sobretodo la evasión de obstáculos.

Capítulo 2

Trabajo realizado

2.1 Palabras Clave

- Nodo : Es un ejecutable que usa ROS para comunicarse con otros nodos mediante una biblioteca cliente.
- Bbilioteca Cliente: Es la que permite escribir los nodos de ROS en diferentes lenguajes : Python y C++.
- Mensajes: Es un tipo de dato de ROS que es utilizado durante la suscripción y publicación de un topic.
- Topic: Es un medio por el cual los nodos pueden publicar mensajes a través de un topic.
- Odometría: Estimación de la posición de vehículos con ruedas.

2.2 Especificaciones ROS y máquinas

Para poder utilizar NXT Lego, se piden ciertos requerimientos que obedecen al tutorial segido "RobotsNXTdiamondback" [1]. La conexión más adecuada es con la distribución ROS Diamondback en una máquina de Ubuntu (10.04 o más reciente). Estas instrucciones no funcionarán en Windows.

2.2.1 NXT

NXT es un kit de robótica modular fabricado por Lego. Hay interfaces de ROS para el firmware de Lego por defecto (v1.28) en base al-NXT python biblioteca por Douglas Lau.

La pila de software NXT-ROS ofrece muchas herramientas útiles para interactuar con los robots NXT ROS. Actualmente los usuarios NXT pueden tomar modelos de robots creados con Lego Digital Designer, y automáticamente convertirlos en modelos de robots compatibles con ROS.

Una vez que el robot está conectado a ROS, puede iniciar la ejecución de aplicaciones, tales como el controlador base, odometría, teleoperación teclado / joystick, e incluso teleoperación asistida.[3]

2.2.2 ROS (Robotic Operative System)

El sistema operativo del robot (ROS) es un marco flexible para la escritura de software del robot. Es una colección de herramientas, bibliotecas y convenciones que tienen como objetivo simplificar la tarea de crear un comportamiento complejo y robusto robot en una amplia variedad de plataformas robóticas. [2]

- ROS Diamondback fue puesto a servicio el 2 de marzo de 2011 y es la tercera versión de distribución ROS. Contiene más de cuarenta nuevas pilas, incluyendo soporte para Kinect, contribuyó pilas de la creciente comunidad de ROS, y una versión estable. Diamondback ha sido diseñado para ser más pequeño, más ligero y más configurable que ROS C Turtle.[4]
- Otras distribución probada sin mucho éxito fue ROS electric con una máquina Ubuntu 11.10, esta cuarta distribucion de ROS soporta varias plataformas como Android y Arduino, pero no existe mayor documentación para su uso con Lego.

2.2.3 Recursos

En este apartado consideraremos los elementos físicos que dispone el grupo constantemente, es decir los elementos a disposición que serán empleados durante el proceso de investigación y programación.El hardware con el que contamos consta de:

1. Laptop Toshiba:
   - Fabricante: TOSHIBA
   - Modelo: Satelite L755
   - Procesador: Intel® Core™ i3-2350M CPU @ 2.30GHz 4
   - Memoria instalada (RAM): 4.00 GB (2.70 GB utilizable)
   - Tipo de sistema: Sistema operativo de 32 bits
   - Sistema Operativo : Windows 7 Ultimate / Ubuntu 14.04
   - Maquina virtual usada: VM VritualBox en Ubuntu 14.04
   - Sistemas Emulados: Ubuntu 11.04 , Ubuntu 11.10
   - Nulcleos: 4
2. Laptop Dell:
   - Fabricante: DELL
   - Modelo: Inspiron 14

- Procesador: Intel (R) Core(TM) i5-3317U CPU @ 1.70GHz 4
- Memoria instalada (RAM): 4.00 GB (3.70 GB utilizable)
- Tipo de sistema: Sistema operativo de 64 bits
- Sistema Operativo : Windows 8 / Ubuntu 14.04
- Maquina virtual usada: VM Ware Player en Ubuntu 14.04
- Sistemas Emulados: Ubuntu 11.04 , Ubuntu 11.10
- Nulcleos: 4

3. Laptop Toshiba:

   - Fabricante: TOSHIBA
   - Modelo: Satellite L50D-CBT2NX2
   - Procesador: Intel Core i7
   - Memoria instalada (RAM): 4.00 GB (2.70 GB utilizable)
   - Tipo de sistema: Sistema operativo de 64 bits
   - Sistema Operativo : Windows 10
   - Maquina virtual usada: VM VritualBox en Ubuntu 14.04
   - Sistemas Emulados: Ubuntu 11.04 , Ubuntu 11.10
   - Nulcleos: 4

2.3 Tareas y objetivos alcanzados

2.3.1 Investigación

Fue necesario revisar documentación teórica relacionada como el paper "Control y Comportamiento de Robots Omnidireccionales Campos Potenciasles"[5] de Santiago Martińez, Rafael Siesto , de la cual estragimos las sigientes ideas claves como guia.

- Para alcanzar un objetivo con un programa aplicado en róbotica existen dos estrategias claves "Growing Obstacles" (Agrandar Obstáculos) y "Force Fields" (Campos Potenciales)

- Agrandar Obstáculos consiste en redimensionar al objeto a una dimension similar a la del robot para llevar al robot al objetivo, pero no es la forma más segura

- Campos Potenciales consiste en implementar campos de repulsión que emanen los obstáculos, estos campos pueden variar según la distancia del objeto con el robot.
  Esta idea es bastante fuerte y usada en otras competencias como por ejemplo el soccer de robots al implementar un campo atractor con respecto a la pelota y uno de repulsión con los oponentes y límites de la plataforma.

2.3.2 Primer Objetivo

Nuestro primer objetivo fue intstalar y aprender a utilizar ROS (Robotic Operative System) debido a que será el entorno de trabajo de el proyecto en su totalidad.

El primer paso fue instalar ROS, siendo elegida la distribución Diamondback. Para lograrlo fue necesario la instalación des sistema operativo Ubuntu 11.04 (Natty) el cual es la versión mas reciente del S.O. Ubuntu sobre la que es posible trabajar con ROS; fue instalado en una máquina virtual utilizando VMWare Player.

El segundo paso fue la instalación de ROS en la máquina virtual para lo cual fue necesario ejecutar:

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu
    natty main" > /etc/apt/sources.list.d/ros-latest.list'
wget http://packages.ros.org/ros.key -O - | sudo apt-key add
    -
sudo apt-get install ros-diamondback-desktop-full
sudo apt-get install ros-diamondback-desktop-full
```

Luego, el tercer paso fue establecer el entorno de trabajo de ROS lo cual conseguimos ejecutando:

```
echo "source /opt/ros/diamondback/setup.bash" >> ~/.bashrc
. ~/.bashrc
```

Una vez realizado esto ROS queda listo para ser usado y para probarlo ejecutamos los siguientes comandos: » roscore » roscd » rosls

El primero inicia ROS para poder ser utilizado el segundo permite cambiar de directorios y el tercero muestra lo que contiene un directorio.

El cuarto paso fue aprender a utilizar ROS para lo cual nos basamos en el libro: "Learining ROS for Robotics Programing". En este paso aprendimos a:

- Utilizar los comandos básicos de ROS
- Crear borrar y modificar dierctorios de ROS
- Contruir un paquete de ROS
- Manejar nodos y tópicos

En este punto evualuamos que habíamos aprendido las nociones básicas del manejo de ROS para continuar con el objetivo.

2.3.3 Segundo Objetivo

Nuestro segundo objetivo fue hacer la conección NXT-ROS y utilizar los robots NXT mediante ROS.

El primer paso fue instalar lo necesario para realizar la conección NXT-ROS lo cual se logró mediante el uso de los siguientes comandos:

-Comados de configuración:

```
sudo groupadd lego
sudo usermod -a -G lego <username>
echo "BUS==\"usb\", ATTRS{idVendor}==\"0694\", GROUP=\"lego
    \", MODE=\"0660\"" > /tmp/70-lego.rules && sudo mv /tmp
    /70-lego.rules /etc/udev/rules.d/70-lego.rules
sudo restart udev
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu
    natty main" > /etc/apt/sources.list.d/ros-latest.list'
wget http://packages.ros.org/ros.key -O - | sudo apt-key add
     -
```

-Comandos de instalación:

```
sudo apt-get update
sudo apt-get install ros-diamondback-nxtall
```

Luego de esto ya es posible realizar la conección NXT-ROS y para probarla ejecutamos:

```
roscore
rosrun nxt_python touch_sensor_test.py
```

Éste último fallará si no hay un NXT conectado al puerto USB.

El segundo paso fue lograr leer los datos de un sensor de tacto de un robot NXT conectado:

1. Es necesario crear un paquete de ROS y construirlo, para esto se ejecutan los siguientes comandos:

```
    roscd nxt
    roscreate-pkg learning_nxt rospy nxt_ros
    rosmake
```

2. Se deben crear dos archivos dentro del paquete robot.yaml y robot.launch, cada uno debe contener:

```
    En robot.yaml:
    nxt_robot:
  - type: touch
    frame_id: touch_frame
    name: my_touch_sensor
    port: PORT_1
```

```
7       desired_frequency: 20.0
```

Con esto decimos que queremos crear un tópico con nombre my_touch_sensor que escuche a un sensor de tacto conectado al puerto 1 del NXT con una frecuencia de 20.0.

En robot.launch:

```
1       <launch>
2
3       <node pkg="nxt_ros" type="nxt_ros.py" name="nxt_ros"
            output="screen" respawn="true">
4           <rosparam command="load" file="$(find
                learning_nxt)/robot.yaml" />
5         </node>
6
7       </launch>
```

Este es alrchivo que ejecutará robot.yaml.

3. Para ejecutar robot.launch:

```
1       robot.launch
```

y para visualizar los datos que recibimos:

```
1       rostopic echo my_touch_sensor
```

El tercer paso fue lograr recibir los datos de un motor que se realizó de forma similar:

1. En el archivo robot.yaml:

```
1       nxt_robot:
2         - type: motor
3           name: l_motor_joint
4           port: PORT_A
5           desired_frequency: 10.0
```

Con esto decimos que queremos crear un tópico con nombre my_touch_sensor que escuche a un sensor de tacto conectado al puerto 1 del NXT con una frecuencia de 20.0.

2. En robot.launch:

```
<launch>

<node pkg="nxt_ros" type="nxt_ros.py" name="nxt_ros"
     output="screen" respawn="true">
    <rosparam command="load" file="$(find
        learning_nxt)/robot.yaml" />
  </node>
<node pkg="nxt_ros" type="joint_states_aggregator.py
    " name="joint_state_publisher" output="screen"
    />

</launch>
```

3. Para ejecutar robot.launch:

```
robot.launch

y para visualizar los datos que recibimos:

rostopic echo joint_states
```

El cuarto paso fue lograr controlar un motor de acuerdo a los datos recibidos de un sensor de ultrasonido, ésta tarea implicó la utilización de dos tópicos adicionales comúnmente llamados "talker" y "listener". Para lograr mover un motor de NXT el tópico listener será el que recibe los datos y controla el motor y tópico "talker" será el que enviará la potencia con la cual el motor se moverá para lograr esto debemos crear el archivo: controller.py y colocar lo siguiente:

```
#!/usr/bin/env python
# license removed for brevity
import roslib; roslib.load_manifest('nxt_ros')
import rospy
from std_msgs.msg import String
from nxt_msgs.msg import JointCommand

def talker():
    pub = rospy.Publisher('joint_command',JointCommand)
    rospy.init_node('talker', anonymous=True)
    rate = rospy.Rate(10) # 10hz
    while not rospy.is_shutdown():
        pub.publish('motor',10)
        rate.sleep()

if __name__ == '__main__':
   try:
        talker()
```

```
    except rospy.ROSInterruptException:
        pass
```

En el archivo robot.yaml:

```
nxt_robot:
  - type: motor
    name: l_motor_joint
    port: PORT_A
    desired_frequency: 10.0
```

En robot.launch:

```
<launch>

<node pkg="nxt_ros" type="nxt_ros.py" name="nxt_ros"
    output="screen" respawn="true">
    <rosparam command="load" file="$(find learning_nxt)/
        robot.yaml" />
  </node>
<node pkg="nxt_ros" type="joint_states_aggregator.py"
    name="joint_state_publisher" output="screen" />

</launch>
```

Para ejecutar todo:

```
roscore
roscd learning_nxt
roslaunch robot.launch
python controller.py
```

El quinto paso fue lograr mover un motor de NXT de acuerdo a los datos recibidos de un sensor de ultrasonido para esto:

En controller.py:

```
#!/usr/bin/env python
# license removed for brevity
import roslib; roslib.load_manifest('nxt_ros')
import rospy
from std_msgs.msg import String
from nxt_msgs.msg import JointCommand
from nxt_msgs.msg import Range

pub = rospy.Publisher('joint_command',JointCommand)
def callback(data):
    rospy.loginfo(rospy.get_caller_id()+"I heard %s",str(
        data.range))
```

```
    if(data.range>1.0):
        pub.publish('motor',10)
    else:
        pub.publish('motor',00)

def talker():

   rospy.init_node('talker', anonymous=True)
   rospy.Subscriber('ultrasonic_sensor', Range, callback)
   rate = rospy.Rate(10) # 10hz
   rospy.spin()

if __name__ == '__main__':
   try:
       talker()
   except rospy.ROSInterruptException:
       pass
```

En el archivo robot.yaml:

```
    nxt_robot:
    - type: ultrasonic
        frame_id: r_ultrasonic_link
        name: ultrasonic_sensor
        port: PORT_2
        spread_angle: 0.2
        min_range: 0.01
        max_range: 2.5
        desired_frequency: 10.0
    - type: motor
        name: motor
        port: PORT_A
        desired_frequency: 10.0
```

En robot.launch:

```
    <launch>

    <node pkg="nxt_ros" type="nxt_ros.py" name="nxt_ros"
        output="screen" respawn="true">
    <rosparam command="load" file="$(find learning_nxt)/
        robot.yaml" />
    </node>
    <node pkg="nxt_ros" type="joint_states_aggregator.py"
        name="joint_state_publisher" output="screen" />
    </launch>
```

Para ejecutar todo:

```
roscore
roscd learning_nxt
roslaunch robot.launch
python controller.py
```

Cabe resaltar que para este paso no encontramos ningún tutorial disponible y tuvimos que realizarlo haciendo uso de lo que habíamos aprendido sobre manejo de tópicos en ROS.

Con éste último paso completamos la conección y el manejo de sensores y motores de un NXT mediante ROS lo cual la deja lista para ser utilizada en el proyecto Campos Potenciales.

2.4 Problemas al desarrollar las tareas

1. La máquina virtual de VirtualBox posee conflictos con el reconocimiento de dispositivos externos como las entradas USB , lo que limita enormemente la conexión de motores y sensores a probar con ROS.
2. AL instalación de VM Ware Player en máquinas de 32 bits es problemática ya que varias versiones para equipos de esta naturaleza ya estan decontinuadas. O poseen conflictos con el host.
3. Al intentar actualizar con: sudo apt-get update no se ejecutaba correctamento debido a que no es una versión actual de Ubuntu.
4. Instalación incorrecta de ROS.
5. Error al ejecutar el comando rosmake.
6. Error de permisos al ejecutar el comando roscreate-pkg.
7. Error en el archivo

```
/opt/ros/diamondback/stacks/nxt/nxt_ros/scripts/nxt_ros.
    py
```

   cannot find nxt.locator

8. Error al conectar el robot NXT al puerto USB.
9. Error en el archivo

```
/opt/ros/diamondback/ros/nxt/learning_nxt/controller.py
```

   cannot find rospy

2.5 Solución de los problemas

1. Para el empleo de maquinas virtuales debe garantizarse que esta tenga un correcto manejo de sus terminales y que los reconozca, la maquina virtual recomendada es VM Ware Player, esto tambien nos permite no realizar mas particiones en el disco para probar las diferentes versiones de Ubuntu y su compatibilidad con cada distribución de ROS.
2. Se recomienda instalar vmware Workstation e intentar dar los permisos mediante consola para poder modificar el host.Esta solución aun esta siendo probada.
3. Modificar el archivo sources.list de Ubuntu ubicado en etc/apt y reemplazar todas las coincidencias de 'archive.ubuntu.com' por 'old-releases.ubuntu.com'.
4. Eliminar lo instalado y reinstalar completamente ROS diamondback
5. Ejecutar:

```
$export ROS_PACKAGE_PATH = /ruta/del/paquete${
    ROS_PACKAGE_PATH}
```

6. Ejecutar:

```
$sudo chown usuario  -R /dev/rosbook
```

7. Comentar la linea 365.
8. Activar todos los puertos USB en la barra inferior de VM Ware Player.
9. Incluir la siguiente linea:

```
import roslib; roslib.load_manifest('nxt_ros')
```

Capítulo 3

Proyecciones

3.1 Problemas y pendientes

- Dado que ya se profundizó en el uso de ROS y su conexión, se podría ver casos mas relacionados a la detección con sensores de proximidad para orientar la investigación al tema de Campos Potenciales.
- Comenzar en la implementación de estrategias que usen las técnicas de "Growing Obstacles" y "Force Fields".
- Solucionar los problemas pendientes correspondientes a la compatibilidad con la arquitectura de ciertas máquinas.
- Orientar la investigación a proyectos múltiples como el caso de la aplicaión de campos potenciales en competencias de soccer o rescate con robots.

Capítulo 4

Conclusiones

- Se vio la importancia de alcanzar objetos con los robots mediante las técnicas de "Growing Obstacles" y "Force Fields".
- Se establecieron las herramientas usadas para este trabajo como el empleo del framework ROS, el kit robotico NXT de Lego.
- Se logró instalar la plataforma par a las posteriores investigaciones y aplicaciones con ROS.
- Se hicieron pruebas de algunos sensores y motores NXT controlados mediante ROS Diamondback.
- Las proyecciones del proyecto son vastas por su gran aplicación en otras competencias que empleen trayectorias de robots.

Bibliography

[1] http://wiki.ros.org/Robots/NXT/diamondback

[2] http://www.ros.org/about-ros/

[3] http://wiki.ros.org/Robots/NXTNXT

[4] http://wiki.ros.org/diamondback

[5] Santiago Martińez, Rafael Siesto Control y Comportamiento de Robots Omnidireccionales Campos Potenciasles, Instituto de Computación, Facultad de Ingenieria - Universidad de la Republica, Montevideo - Uruguay 2009