Uchile Homebreakers Robotics Team
Facultad de Ciencias Físicas y Matemáticas
Universidad de Chile

Uchile Peppers

Maqui for Dummies

Versión marzo 2023

Primera Ley

Un robot no hará daño a un ser humano,
ni por inacción permitirá que un ser humano sufra daño.

Segunda Ley

Un robot debe cumplir las órdenes dadas
por los seres humanos, a excepción de aquellas
que entren en conflicto con la primera ley.

Tercera Ley

Un robot debe proteger su propia existencia
en la medida en que esta protección no entre en
conflicto con la primera o con la segunda ley.

Ley Cero

Un robot no puede dañar a la humanidad o,
por inacción, permitir que la humanidad sufra daños.

Índice

1. Prefacio

El laboratorio de robótica del departamento de Ingeniería Eléctrica de la Universidad te da la bienvenida a este curso intensivo: Pepper 101.

Históricamente el laboratorio se ha caracterizado por presentar proyectos memorables y del agrado de toda la comunidad Universitaria. Entre estos podemos encontrar al equipo de fútbol robótico de la universidad de Chile, los hoy retirados *Naos*. También te vas a encontrar con el robot ganador de nada menos que dos premios de innovación por su capacidad para detectar y expresar emociones: *Bender*.

Actualmente el laboratorio se encuentra trabajando en tres proyectos principales, estos involucran al ya mencionado Bender, también se está desarrollando un nuevo robot, Jaime (el robot mayordomo. ¿Cuál es nombre de mayordomo? James. Ya pero, ¿en español? Jaime). Jaime es un brazo robótico móvil que planea simular una planta para que la interacción con las personas se agradable y se sientan cómodas.

Finalmente, pero no menos importante, el robot que tiene su propio apunte al más puro estilo DIM, pero sin ecuaciones feas. Pepper, es un robot fabricado por Softbank Robotics diseñado como un robot social. Equipado con múltiples sensores es capaz de navegar por su entorno, interactuar con personas, detectar objetos, etc. ¿La gracia?, uno de los principales atractivos del robot, de acuerdo a Aldebraran, es que *es un robot que no está terminado*, en este sentido, cada dueño o dueña de un robot Pepper tiene la misión de prepararle para lo que estime conveniente. En el laboratorio, se bautizó a Pepper con el nombre *Maqui*. Así, Maqui es un robot que ha logrado grandes hazañas, compitiendo en varias ocasiones en la *RoboCup*, una competencia internacional de robótica, llegando a quedar en segundo lugar en la categoría *Social Standar Platform* de la liga *RoboCup@Home*.

Ahora, ¿tienes ganas de aprender a controlar a Pepper con todos los avances que se han logrado en el laboratorio? Si la respuesta es sí, **¡Bienvenid@ al Team Pepper!**.

¿Qué necesitas para empezar? Antes de comenzar a leer este manual, *Maqui for Dummies*, necesitas saber ROS (ROBOT OPERATING SYSTEM). ¿Qué es ROS?, primero que todo, ROS NO es un sistema operativo, ROS es un framework/middleware que permite, en palabras muy simples, que todo el robot funcione como un solo ente, es decir, permite la intercomunicación de todos sus sensores y actuadores, logrando así tener un sistema robusto.

Así que ya sabes, partiste a instalar Ubuntu 18, ROS Melodic y hacer los tutoriales de ROS [1]. *Maqui For Dummies* te estará esperando cuando termines.

2. UChile ROS Framework: Instalación del Sisitema

Para poder trabajar con cada uno de los robots del laboratorio se creó un espacio de trabajo en ROS [2] que tiene como objetivo proveer herramientas para manejar el código de los equipos @Home de UChile Robotics. Actualmente contiene los siguientes módulos:

- Manejo e instalación del la estructura de archivos y workspaces ROS.
- Configuración centralizada para cargar ROS en bash y zsh.
- Hooks para repositorios git.
- Herramientas para bash-zsh.

En el caso particular del robot Pepper, nos centraremos en la instalación del sistema mostrada en la rama *maqui-fix-2022* [3], que posee la versión más actualizada del repositorio con el que se trabaja en Pepper.

2.1. Prerrequisitos

Esta guía ha sido probada en Ubuntu 18.04.

IMPORTANTE: Es recomendado seguir las instrucciones directamente desde el repositorio de github [3], pues ahí siempre estará la versión más reciente de estas instrucciones.

2.1.1. Instalar ROS

AVISO: Esta sección se puede ignorar si es que ROS ya está instalado en la máquina. En tal caso, asegurate de tener instalado rosdep (Ctrl+Alt+T):

```
sudo apt-get install python-{rosinstall,pip,rosdep}
```

Ejecutar en terminal (Ctrl+Alt+T):

```
# ROS Keys
# Evite instalar la versión full (sudo apt-get
#install ros-melodic-desktop-full) o alguna de las otras variantes.
# ver: http://wiki.ros.org/melodic/Installation/Ubuntu
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc)
main" >/etc/apt/sources.list.d/ros-latest.list'

sudo -E apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-key
C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654

# actualizar base de software
sudo apt-get update

# instalar ROS base
sudo apt-get install ros-melodic-ros-base curl openssl pv python
-rosinstall python-pip python-rosdep

# inicializar rosdep
sudo rosdep init # ignorar si es que falla con "ERROR: default
#sources list file already exists:..."

rosdep update    # NO EJECUTAR CON SUDO!
```

2.1.2. Instalar Dependencias para system

Ejecutar en terminal (Ctrl+Alt+T):

```
sudo apt-get update

sudo apt-get install git python-flake8 shellcheck libxml2-utils
python-yaml cppcheck curl openssl pv python-rosinstall python-pip
openssh-client python-termcolor openssh-server python-rosdep jq
```

2.2. Instalación de Uchile ROS Framework

Procurar ejecutar las veces que sea necesario, pues puede fallar el clone de algún repositorio, por ejemplo, al introducir mal la clave.

Observación: Actualmente, hay dos repositorios que son privados y por lo tanto, pedirán usuario y contraseña de GitHub al intentar descargarlos. Revisar la salida del instalador para ver que

todo funcionó OK; También puedes revisar manualmente que uchile_perception y uchile_high se hayan descargado correctamente. En caso de que haya algún error, ejecutar nuevamente la línea del instalador.

Ejecutar en terminal (Ctrl+Alt+T):

```
# descargar uch_system
git clone https://github.com/uchile-robotics/uchile_system.git ~/tmp_repo

cd "$HOME"/tmp_repo && git checkout maqui-fix-2022

# Obtener repositorios y crear workspaces
bash "$HOME"/tmp_repo/install/ws_installer.bash

# limpiar
rm -rf ~/tmp_repo
```

2.2.1. Habilitar Workspace para uso en consola

Antes de ejecutar el siguiente paso, es necesario que revises el archivo de configuración correspondiente a tu consola .bashrc o .zshrc, para eliminar toda línea relacionada con ROS. Por ejemplo, debes comentar toda línea de la forma **source /opt/ros/melodic/setup.bash o source workspace-ros .bash.**

Hint: .bashrc y .zshrc se encuentran ocultos en "$HOME". Puedes mostrar/ocultar éstos archivos utilizando **Ctrl+H**. En cada caso puedes copiar el bloque de código directo en la terminal **(Ctrl+Alt+T)**.

Sólo usuarios de bash

```
cat >> ~/.bashrc <<"EOF"

## ------------------------------------------------
## UCHILE ROS FRAMEWORK Settings

# workspace location
export UCHILE_WS="$HOME"/uchile_ws

# settings file location
export UCHILE_SHELL_CFG="$HOME"/uchile.sh

# UChile Robotics Framework for BASH
# comment this line to prevent sourcing the framework
. "$UCHILE_WS"/system/setup.bash
## ------------------------------------------------

EOF
```

Sólo usuarios de zsh

```
cat >> ~/.zshrc <<"EOF"

## ------------------------------------------------
## UCHILE ROS FRAMEWORK Settings

# workspace location
export UCHILE_WS="$HOME"/uchile_ws

# settings file location
export UCHILE_SHELL_CFG="$HOME"/uchile.sh

# UChile Robotics Framework for ZSH
# comment this line to prevent sourcing the framework
. "$UCHILE_WS"/system/setup.zsh
## ------------------------------------------------

EOF
```

Para continuar la instalación y que las configuraciones anteriores surtan efecto, es necesario abrir un nuevo terminal. De lo contrario, no existirán las variables de entorno ni funciones necesarias, como UCHILE_SYSTEM, bgit o cdb.

2.3. Configuraciones

En el archivo /uchile.sh se deben pueden configurar aspectos del framework como el robot a utilizar y opciones de red. Pon atención en las variables especificadas en tal archivo, pues deberás modificarlas constantemente.

Al menos, debieras configurar la variable de entorno UCHILE_ROBOT, que por defecto es bender. Ésta permite seleccionar que overlay de workspaces ROS se utilizarán. Todos los overlays diponibles se encuentran en el directorio $UCHILE_WS/ros/. Según el valor escogido, el workspace ROS linkeado proveerá distintos packages, y por lo tanto, requerirá distintos pasos de instalación.

2.3.1. Configuraciones MUY Recomendadas

Estas configuraciones son opcionales, pero se recomiendan para facilitar el desarrollo. Leer con atención y sólo habilitar las realmente deseadas.

Ejecutar en terminal (Ctrl+Alt+T)

```
# Configuraciones globales de git.
# 1.- colores y aliases para comandos git.
# 2.- configurar nombre y mail para commits.
#(ojalá mail matchee con el de github!)
# 3.- configurar caché para contraseña git a 1 día
# 4.- por defecto sólo pushear la rama actual.
#Evita subir commits que pueden no estar listos.

cp -bfS.bkp "$UCHILE_SYSTEM"/templates/default.gitconfig ~/.gitconfig
git config --global user.name 'Replace Your Name Here'
git config --global user.email 'replace.your.email.here@gmail.com'
git config --global credential.helper 'cache --timeout=86400'
git config --global push.default simple

# Promt de bash muestra rama actual y estado del repositorio git.
cp -bfS.bkp "$UCHILE_SYSTEM"/templates/bash_aliases ~/.bash_aliases

# Variable utilizada por "rosed" y algunos utilitarios.
# Poner el nombre del ejecutable deseado. Ejemplo: "subl" para sublime.
echo 'export EDITOR="gedit"' >> ~/.bashrc
```

2.3.2. Configuraciones MENOS Recomendadas

Cuidado con las siguientes configuraciones!. Pueden ser útiles, pero no útiles para todos.

Ejecutar en terminal (Ctrl+Alt+T)

```
# Herramienta meld para git diffs. (OBS!, puede ser molesta para algunos!)

# - permite ver diffs más bellos y hermosos.
sudo apt-get install meld

cp "$UCHILE_SYSTEM"/templates/gitconfig_meld_launcher.py
~/.gitconfig_meld_launcher.py

git config --global diff.external  '~/.gitconfig_meld_launcher.py'
```

2.4. Compilación de Workspaces

Esta rama está focalizada en compilar para maqui, por lo tanto se debe modificar el archivo "uchile.sh", específicamente, cambiar en la línea de UCHILE_ROBOT "all"por "maqui".

En esta fase es importante el orden de compilación.

El sistema se divide en 5 workspaces, que en orden son: ROS, forks_ws, base_ws, soft_ws y high_ws.

Los pasos a seguir dependerán del robot a utilizar, según la variable $UCHILE_ROBOT.

Ejecutar en terminal (Ctrl+Alt+T)

```
# Actualizar base de datos del repositorio de software.
sudo apt-get update
```

2.4.1. Instalación de fork_ws

fork_ws

```
# instalar dependencias

sudo apt install ros-melodic-yocs-msgs
sudo apt install ros-melodic-naoqi-bridge-msgs

cdb forks && rosdep install --from-paths . --ignore-src --rosdistro=melodic -r

# Compilar
cdb forks && cd .. && catkin_make
```

2.4.2. Instalación de base_ws

base_ws

```
# instalar dependencias
cdb base && rosdep install --from-paths . --ignore-src --rosdistro=melodic -r

cdb uchile_gazebo && bash install/install_models.sh
```

IMPORTANTE

Entrar a uchile_ws/ros/maqui/base_ws/src/bender_core, copiar (o cortar) bender_sensors y bender_joy, después pegarlos en uchile_ws/ros/maqui/base_ws/src y borrar bender_core

base_ws

```
#Compilar
cdb base && cd .. && catkin_make
```

2.4.3. Instalación soft_ws

soft_ws

```
# instalar dependencias
cdb soft && rosdep install --from-paths . --ignore-src
--rosdistro=melodic -y -r


# instalar dependencias de speech
cdb uchile_speech_pocketsphinx && bash install/install.sh
sudo apt-get install python-alsaaudio

#Compilar
cdb soft && cd .. && catkin_make
```

2.4.4. Instalación high_ws

high_ws

```
# instalar dependencias
cdb high && rosdep install --from-paths . --ignore-src --rosdistro=melodic -r
sudo apt-get install python-aiml

# Compilar
cdb high && cd .. && catkin_make
```

2.5. Configuraciones Obligatorias

Para trabajar con los archivos más recientes de Pepper, hay ciertos repositorios dentro del *Uchile_Workspace* que acabas de compilar, que se encuentran en ramas diferentes a las principales, en general, estas ramas se llaman *feat-NewMaqui*. A continuación se listarán los diferentes repositorios en los que es necesario cambiarse de rama para trabajar con Maqui:

- maqui_bringup
- uchile_maps

TIP!

Para cambiar de rama en una consola, se usa el siguiente comando, tomando como ejemplo el cambio de rama a *feat-NewMaqui*;

```
git checkout feat-NewMaqui
```

2.6. Cargar Meshes

Las *meshes*, son las "mallas de personaje"para poder ver el modelo 3D de Pepper dentro de los ambientes simulados. Actualmente se está trabajando por incluirlas dentro de la compilación general del espacio de trabajo, sin embargo, de momento para poder ver a Maqui con todas sus características físicas en el computador donde se encuentra compilado el repositorio, es necesario agregar determinados archivos de manera externa:

- Consulta con alguno de los tutores del laboratorio o algún miembro del team Pepper las *"Meshes"* de Pepper.
- Una vez se te entreguen los archivos, cópialos y pégalos en: /fork_ws/src/pepper/pepper_meshes

Con el repositorio descargado y compilado, ya está todo listo para empezar a trabajar con Maqui.

3. Conectarse al Robot

Es importante recordar que, cuando hablamos de un robot, estamos hablando de un computador que tiene bastantes addons. En este sentido, si hablamos de Pepper, cuando nos queremos conectar de manera remota a su computador (de ahora en adelante la *cabeza* de Pepper), lo haremos siguiendo el protocolo SSH para acceder, remotamente, de un computador a otro [4].

3.1. Navegador e IP

Para lograr lo anterior, es importante conocer cuál es la red a la que está conectada Pepper y cuál es la IP que tiene asociada dentro de esa red. Obtener la IP es sencillo: con el robot encendido, basta con presionar 1 vez el botón de encendido del robot (NO mantener presionado, esto provocará que se apague) que se encuentra en su tronco, detrás del tablet. Una vez hecho esto, Pepper nos dirá los números correspondientes a su dirección IP.

Si por algún motivo, no sabemos cuál es la red a la que está conectado o alguna otra situación nos impide avanzar, detrás de su cabeza física, *se puede retirar una tapa* que nos permitirá acceder a un puerto ethernet. Si se conecta un computador directamente a este puerto, en un navegador podremos acceder a la dirección ***pepper.local*** y se nos redirigirá a la configuración del robot, donde podremos seleccionar una red wifi para conectarse. A esta página web también podremos acceder si nuestro computador se encuentra conectado a la misma red wifi que Pepper, conociendo la IP de Pepper o con la misma dirección (*pepper.local*).

Si se pide algún usuario o contraseña estos son:

- user: nao
- password: maqui

Para mayor información sobre la interfaz Web de Pepper, se puede visitar la documentación oficial de Sofbank [5].

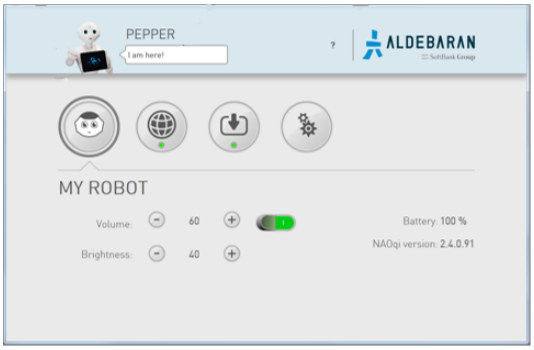


Figura 1: Pepper Web Page.

3.2. SSH

Como se mencionó, para conectarnos a Maqui lo haremos con el protocolo SSH. Una vez tenemos la IP del robot y sabemos cuál es la red a la que está conectado, con un computador conectado a la misma red, en una terminal, podemos conectarnos gracias al siguiente comando:

Ejecutar en terminal (Ctrl+Alt+T)

```
#ip_pepper será la ip que obtengamos de la subsección
#anterior. O también podemos intentar con pepper.local
ssh nao@pepper.local
```

Una vez lanzado el comando, se nos pediran las credenciales correspondientes, que son las mismas de la parte anterior. Así, ya estaremos en la cabeza de Pepper donde podremos navegar por sus archivos como en cualquier terminal.

Ahora bien, si queremos administrar archivos o movernos dentro de las carpetas de una manera más gráfica (sin terminales) podemos hacer esto mediante **SFTP** (Safe File Transfer Protocol) y nautilus. Esto lo haremos con el siguiente comando:

Ejecutar en terminal (Ctrl+Alt+T)

```
#ip_pepper será la ip que obtengamos de la subsección
#anterior. O también podemos intentar con pepper.local
nautilus sftp://pepper.local
```

4. Primeros Pasos

4.1. ROS Launch

Como ya se tiene el repositorio de Pepper instalado en el computador, es posible comenzar a controlarlo utilizando ROS. Para comenzar, es necesario lanzar ROS además de todos los nodos y tópicos de tal forma que Maqui sea capaz de funcionar (cámaras, lasers, ruedas, etc.), para esto, existe un archivo en la cabeza de Maqui donde están todas las instrucciones para lanzar todos los paquetes, nodos y tópicos de manera simple, lo que llamaremos un archivo *launch*. Para hacerlo, basta ejecutar un comando en la cabeza de Pepper al estar conectados por ssh:

Conectados a Maqui por SSH

```
roslaunch maqui_bringup maqui.launch
```

Es importante correr el archivo *.launch* cada vez que se quiera usar a Pepper, de lo contrario ninguna de las funcionalidades propias del laboratorio que utilicen ROS funcionarán.

4.2. Hello World: Movimiento y Text To Speech

Al lanzar todo lo necesario para hacer funcionar a Pepper, si ahora revisamos los tópicos activos en Maqui (*rostopic list*), nos encontraremos con una larga lista con todos los tópicos que se activaron, es más, podemos publicar de manera directa en algunos de ellos utilizando el siguiente comando estándar:

Conectados a Maqui por SSH o exportando el ROS Master

```
rostopic pub topic_name message_type message
```

En la mayoría de los casos, se podrá autocompletar el mensaje a enviar solamente escribiendo hasta el nombre del tópico y utilizando la tecla *tab* para obtener el tipo de mensaje y el formato en el que se envía el mensaje, ese es el caso, por ejemplo, cuando queremos publicar velocidad directamente en el tópico $\backslash cmd\_vel$, basta con escribir *rostopic pub* $\backslash cmd\_vel$ y completar con *tab* el resto del comando, para finalmente modificar a mano los parámetros x, y y z de las velocidades angular y lineal. De la misma forma, esto se puede hacer con el tópico $\backslash speech$, donde, en ambos casos, los comandos completos quedan como se muestra a continuación:

Mover y hacer hablar a Maqui publicando mensajes directamente en los tópicos

```
#publicar en cmd_vel
rostopic pub /cmd_vel geometry_msgs/Twist "linear:
  x: 0.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.0"

#publicar en speech
rostopic pub /speech std_msgs/String "data: Hello World''"
```

4.3. Teleoperación

Como se vio en la subsección anterior, es posible lograr que maqui se mueva publicando directamente en el tópico de velocidad, sin embargo, esto no es lo óptimo, puesto que en la mayoría de las situaciones necesitaremos controlar a Maqui con cierto nivel de naturalidad. Para ello utilizamos la teleoperación, esto se hará utilizando diferentes nodos que vayan publicando frecuentemente la velocidad que queremos que Maqui lleve en cada instante con la ayuda de distintos periféricos: un joystick o el teclado del computador.

4.3.1. Teleoperando con Joystick

Para usar un joystick externo, no se necesitará instalar nada más aparte del repositorio en el, sin embargo, sí es necesario conectar un control por USB al PC.

Con el computador conectado y Maqui corriendo ROS, seguir los siguientes comandos:

Ejecutar en terminal (Ctrl+Alt+T):

```
export ROS_MASTER_URI http://pepper.local:11311
rosrun joy joy_node
```

Ahora, sin cerrar la terminal anterior:

Ejecutar en nueva terminal (Ctrl+Alt+T):

```
export ROS_MASTER_URI http://pepper.local:11311
rosrun maqui_joy joy_base.py
```

¡Y listo! Utilizando los diferentes botones del control, Pepper debería comenzar a moverse.

Errores comunes:

- Fijarse que Pepper no tenga el freno puesto.
- Fijarse que el nodo esté publicando en el tópico \cmd_vel y no en \maqui\cmd_vel.

4.3.2. Teleoperando con el Teclado

Para utilizar esta funcionalidad, es necesario instalar un nuevo paquete de ROS llamado *teleop_twist_keyboard* [6]. Se puede hacer siguiendo las siguientes instrucciones para ROS melodic:

Ejecutar en terminal (Ctrl+Alt+T):

```
sudo apt-get install ros-melodic-teleop-twist-keyboard
```

Y para utilizarlo para controlar a Maqui, se lanzará un nodo que el paquete trae incluido:

Ejecutar en terminal (Ctrl+Alt+T):

```
rosrun teleop_twist_keyboard teleop_twist_keyboard.py
```

El nodo publica por defecto en el tópico \cmd_vel, pero si se quiere publicar en un tópico distinto a ese las instrucciones dadas por el teclado, se puede cambiar en los argumentos del comando, por ejemplo, si se quiere publicar en el tópico \baymax\cmd_vel:

Ejecutar en terminal (Ctrl+Alt+T):

```
rosrun teleop_twist_keyboard teleop_twist_keyboard.py cmd_vel:=\baymax\cmd_vel
```

4.4. Modo Autónomo

Cada vez que se enciende a Maqui, este entra en el modo *demo* de fábrica, lo cual no está mal, sin embargo, este modo puede llegar a interferir en determinadas acciones que se quieran realizar. El principal ejemplo de lo anterior es entorpecer el *speech recognition*. Para ello, se desactivará el modo autónomo. Esto se hace utilizando los servicios de ROS con los siguientes comandos conectados a Pepper por SSH:

Conectados a Pepper por SSH:

```
rosservice call \maqui\pose\life\disabled #esperar a que quede en modo espera

rosservice call \maqui\pose\wakeup #devuelve la postura predeterminada
```

Teniendo en cuenta lo anterior, se puede deducir que el modo autónomo y la pose *rest* se activan con los siguientes comandos análogos a los anteriores:

Conectados a Pepper por SSH:

```
rosservice call \maqui\pose\life\enable #activa modo autónomo

rosservice call \maqui\pose\rest #activa pose de descanso
```

Lo anterior se pude hacer también utilizando Choreographe (una plataforma de la que se hablará más adelante), esto se logra, en primer lugar, deshabilitar el **corazón** que se muestra en la esquina superior derecha de la plataforma, como se muestra en la figura 2. Esto realizará la misma acción que el primer comando usado al llamar los servicios de ROS, por lo que ahora será necesario devolverle a Maqui su postura normal, lo cual se hace presionando el **sol** se encuentra en la misma esquina. Luego, si se vuelve a presionar el **corazón** se activará la vida autónoma, y si se presiona la **luna**, se activa la pose de descanso.

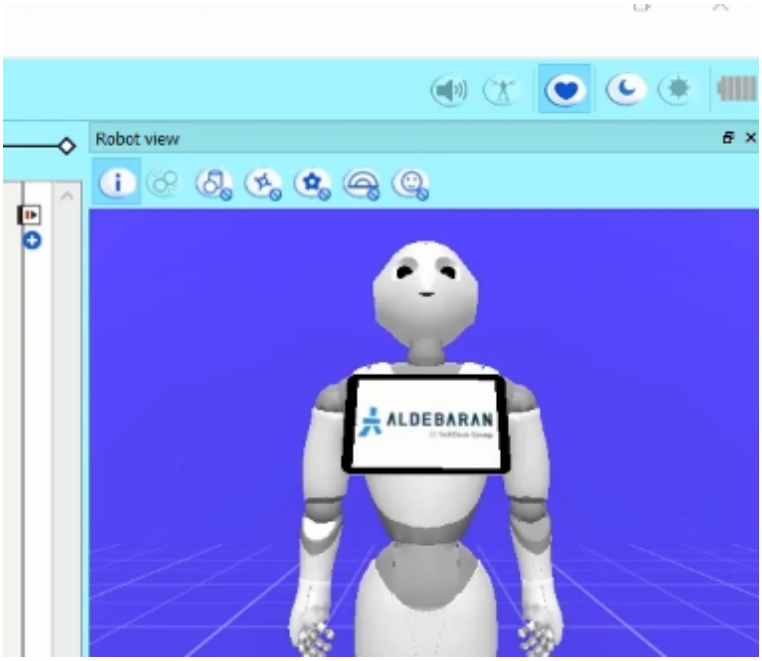


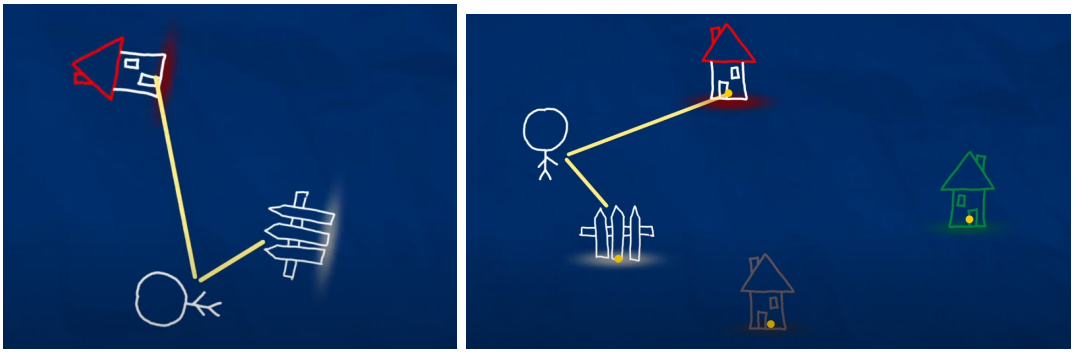
Figura 2: Modo autónomo Choreographe.

5. SLAM: Simultaneous Localization and Mapping

Imagina que quieres hacer un mapa de la calle donde vives, así que comienzas a caminar, mientras tú vas tomando nota de todo lo que ves: Comienzas a avanzar y hay una casa anaranjada a tu izquierda, luego una casa verde a tu derecha y llegas a una curva, doblas a la izquierda y a tu derecha hay una casa roja, finalmente, avanzas y poco más y hay una valla blanca a tu izquierda. Guardas el registro de todo lo que viste y cómo te moviste y listo: ya tienes el mapa de tu calle [3].

Figura 3: Recreación del mapeo.

Ahora imagina que estás en un punto, a priori, desconocido del espacio, sin embargo eres capaz de ver, como se muestra en la figura 4.a, una valla blanca y una casa roja frente a ti, la valla está a la derecha y la casa a la izquierda, ahora, como tienes tu mapa creado con anterioridad, pues comprobar que, si eres capaz de visualizar lo anterior, significa que estás en la posición que se muestra en la figura 4.b, es decir, fuiste capaz de encontrar tu posición en el sistemas de coordenadas global del mapa que habías creado: te localizaste.



(a) (b)

Figura 4: Localización en el mapa creado.

Ahora bien, ya tienes tu mapa y eres capaz de localizarte en el mapa, sin embargo aún no eres capaz de realizar lo que buscamos con todo esto: navegar por tu entorno, es decir, moverte de un

punto a otro considerando las condiciones del entorno. Esto es, como se muestra en la figura 5, considerando que te encuentras en un determinado punto del mapa y quieres llegar a otro punto, pasando por la casa roja, debes ser capaz de encontrar una ruta óptima que cumpla esas condiciones considerando los obstáculos presentes en el mapa y también obstáculos nuevos que podrían aparecer (por ejemplo, si hay un vehículo estacionado, que no está presente en el mapa, justo en medio de la trayectoria calculada). Así, se debe ser capaz de actualizar, dinámicamente, la trayectoria mientras los obstáculos vayan apareciendo en el camino [7].

Figura 5: Recreación de la navegación.

Ahora, lo que hace Maqui (y los demás robots del laboratorio), no es fijarse si hay una casa es roja, si hay una valla o no, sino que crea un mapa en dos dimensiones al ir escaneando su entorno, donde irá marcando distintas celdas como *ocupadas* o *no ocupadas*, es decir, los lugares que están ocupados por algún objeto y por ende no puede pasar por ahí, y los lugares que se encuentran libres y puede navegar por ellos. A continuación se estudiarán los pasos necesarios para que Pepper pueda mapear, localizarse y navegar por su entorno.

5.1. RViz

Una herramienta que conviene conocer antes de comenzar con SLAM, es RViz [8]. En palabras simples, Rviz es una plataforma que permite *ver lo que el robot está viendo*, es decir, al utilizar rviz podemos acceder de manera visual a lo que el robot está procesando en cada instante, esto es, gracias a Rviz, se puede tener una forma más intuitiva de analizar los tópicos *robot_description*, además de otros tópicos correspondientes a láser, las cámaras, etc.

Para abrir un nuevo proyecto de rviz, basta con ejecutar el comando ***rviz*** en una terminal.

Abrir Rviz desde terminal (Ctrl+Alt+T):

```
rviz
```

Al lanzar el comando anterior, se abrirá una ventana como la de la figura 6:

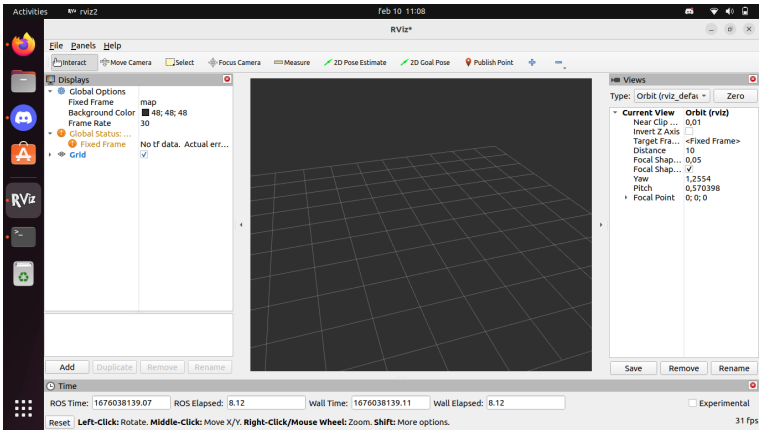


Figura 6: Ejemplo de proyecto Rviz vacío.

Se puede comprender con mayor profundidad el funcionamiento de la plataforma al leer la documentación oficial [8], sin embargo, ya existe un archivo de guardado en el repositorio con todas las configuraciones para Maqui, para lanzarlo, basta exportar el *ROS master* de Maqui al computador (como se enseñó con anterioridad) y lanzar el comando ***mviz*** en una terminal:

Abrir archivo RViz configurado para Maqui desde terminal (Ctrl+Alt+T):

```
mviz
```

Al lanzar el comando anterior, debería abrirse una ventana como la que se muestra a continuación en la figura 7:

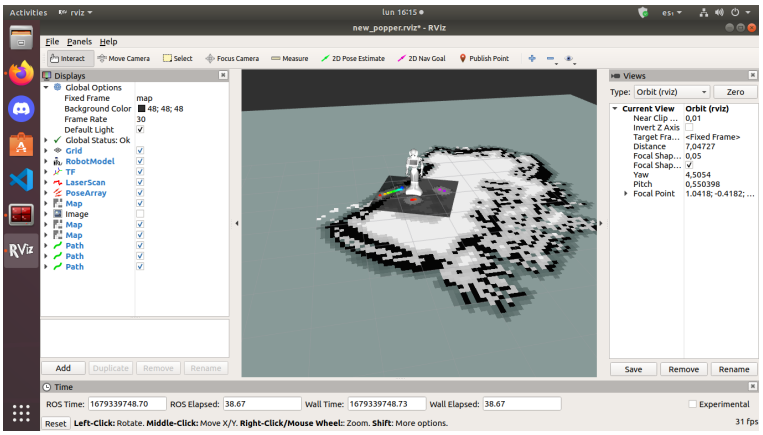


Figura 7: mviz.

5.2. Mapeo: Project Tingle

Para crear un mapa 2D del entorno por el que se moverá Maqui, se utilizará *gmapping* [9]. En este sentido, los pasos a seguir para poder crear un mapa son:

- Iniciar los nodos correspondientes para teleoperar a Pepper, ya sea con el teclado, joystick o algún otro método.
- Abrir rviz (*mviz*) y asegurarse de que todo esté bien configurado (probablemente se abra con algún mapa antiguo, no hay que preocuparse por esto).
- Iniciar los archivos de configuración de gmapping para crear el mapa con el siguiente comando (no olvidar exportar el ROS master):

Ejecutar en terminal (Ctrl+Alt+T):

```
roslaunch maqui_bringup gmapping.launch
```

- Una vez se lance lo anterior, en rviz debería verse un mapa en blanco que comienza a rellenarse poco a poco con la información que se recibe de la odometría y de los láser de Maqui. Para poder obtener toda la información del entorno necesaria, es necesario ir moviendo al robot con ayuda de la teleoperación. Este proceso debe hacerse lento y con calma para evitar errores en las mediciones (con Maqui es complejo que el mapeo resulte a la perfección, pero errores en el mapeo como secciones giradas o cosas mal posicionadas se pueden corregir con software externo).
- Una vez se haya recorrido todo el entorno y se obtenga el mapa, es necesario guardarlo con ayuda del siguiente comando en una nueva terminal, conectados al mismo ROS master:

  Ejecutar en terminal (Ctrl+Alt+T):

  ```
  rosrun map_server map_saver
  ```

Lo anterior generará dos archivos en la carpeta raíz del computador: un archivo .pgm y un archivo .yaml. Si se está conforme con lo obtenido, para cargar el mapa a utilizar, se deben copiar ambos archivos en la carpeta (/uchile_ws/ros/maqui/base_ws/src/uchile_knowledge/uchile_maps) del repositorio compilado en el computador con algún nombre descriptivo, es importante notar que, si se le cambia el nombre a los archivos .pgm y .yaml, también será necesario actualizar el nombre del mapa DENTRO del archivo .yaml. Una vez hecho esto, en el repositorio *maqui_bringup*, se tendrá que actualizar el mapa a cargar en los archivos .launch de localización y de navegación.

5.2.1. LifeHack: GIMP [10]

Si no se está conforme con el mapeo realizado por Maqui, algunos errores del proceso, como rotaciones, objetos mal posicionados o no detectados, se pueden corregir directamente en el archivo .pgm. Un editor de código abierto que ha demostrado ser útil para esta acción es ***Gimp*** [10], el cual hasta ahora se ha utilizado, básicamente, como un *Paint-mapping*.

De cualquier forma, los otros robots del laboratorio son capaces de realizar mapas de mucha mejor calidad, por lo que, a priori, no hay problema en utilizar los archivos utilizados por ellos para la localización y la navegación, solo deben guardarse de la misma forma que si los hubiera generado Maqui.

5.3. Localización y Navegación: Project Columbus

Una vez se tenga el mapa creado, lo siguiente que sigue es que Maqui pueda localizarse y navegar por su entorno. Para lo anterior, con rviz (mviz) abierto, se debe lanzar el comando:

Ejecutar en terminal (Ctrl+Alt+T):

```
roslaunch maqui_bringup nav_full.launch
```

Es importante recordar que, para lo anterior, debe estar corriendo ROS en la cabeza de Pepper y el Ros Master exportado en el computador.

Una vez listo lo anterior, en rviz se debe visualizar una pantalla como la de la figura 8 donde, si la mayor cantidad de flechitas en el suelo se encuentran en los pie de Maqui y la posición de Maqui en rviz coincide con su posición física, significa que está bien localizado y listo para navegar. Si la posición de rviz no coincide con la posición real, será necesario *ayudar* a Pepper a localizarse, esto se hace utilizando el botón con la flecha verde ***2D pose estimate*** en la barra de herramientas superior de rviz, con este, basta seleccionar la posición en el mapa donde *más o menos* se encuentra Pepper para que luego él mismo haga el resto. Ahora bien, si las flechas del suelo están muy dispersas, basta con mover ligeramente a Maqui con la ayuda de la teleoperación o darle un *nav goal* a un lugar cercano como se enseñará a continuación.

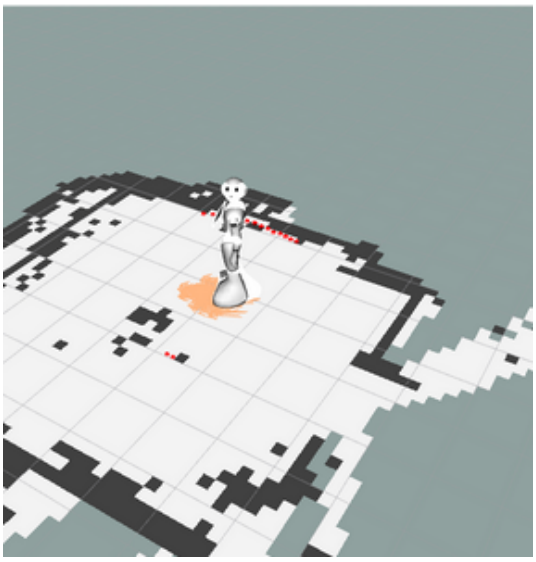


Figura 8: Maqui localizado en un mapa creado por él mismo.

Para lograr que Pepper navegue autónomamente a una poición específica del mapa utilizando rviz, basta con presionar el botón morado ***2D nav goal*** en la barra de herramientas superior de Rviz y orientar la flecha el lugar del mapa al que se quiere que Pepper llegue, el sentido de la flecha indica la orientación en la que llegará Maqui a su destino. Un ejemplo de Pepper localizado y navegando se ve en la figura 9:

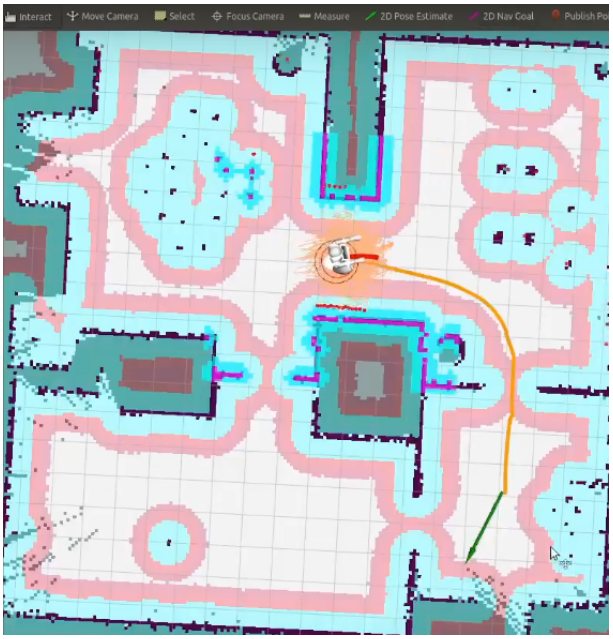


Figura 9: Maqui Navegando.

6. Choreographe

Choreographe es una interfaz gráfica diseñada para los robots de Aldebaran Robotics: Nao y Pepper. Choreographe permite crear aplicaciones con diálogos, servicios y diferentes acciones, como interactuar con personaas, bailes, enviar *e-mails* sin necesidad de escribir código [11]. Estas rutinas no solo se pueden ejecutar desde choreographe, sino que también se pueden llamar desde máquinas de estado, permitiendo una integración completa con ROS.

6.1. Instalación

En primer lugar, es necesario descargar la última versión de Choreographe para linux que se encuentra disponible en el sitio de Aldebaran [12], es decir, se debe descargar la versión *2.5.10* para linux que se encuentra en la página de la bibliografía [12]. Esto descargará un archivo *.run* que tendremos que abrir.

Para comenzar la instalación, en una nueva terminal es necesario dirigirse a la carpeta donde se guardó el archivo descargado (por defecto, debería irse a Downloads).

Ejecutar en terminal (Ctrl+Alt+T):

```
cd Downloads/

sudo chmod +x choregraphe-suite-2.5.10.7-linux64-setup.run

./choregraphe-suite-2.5.10.7-linux64-setup.run
```

Lo anterior, debería pedir la contraseña de administrador del computador, una vez se ingrese, se abrirá el instalador. Basta aceptar los términos, escoger instalación rápida, y el programa se instalará.

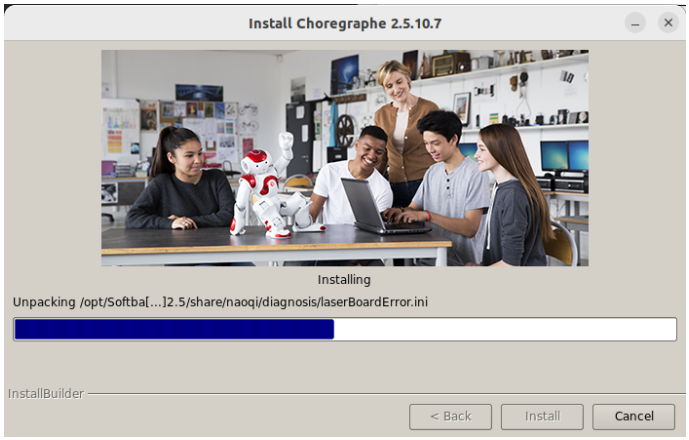


Figura 10: We've got to celebrate our differences.

Cuando termine la instalación, antes de lanzar el programa será necesario ingresar el siguiente comando en una nueva terminal:

Ejecutar en terminal (Ctrl+Alt+T):

```
sudo ln -sf /usr/lib/x86_64-linux-gnu/libz.so /opt/'Softbank Robotics'/
'Choregraphe Suite 2.5'/lib/libz.so.1

#Tener cuidado con el salto de línea del comando al copiarlo en la terminal!!
```

Una vez se abra el programa, se pedirá una clave de activación del producto, esta es:

Clave de activación de Choreographe:

```
654e-4564-153c-6518-2f44-7562-206e-4c60-5f47-5f45
```

Una vez esté lista la instalación, se puede utilizar el software para manipular el robot en tiempo real o guardar rutinas y subirlas a Pepper para utilizarlas más adelante.

Para comprender el funcionamiento general de Choreographe se recomienda aventurarse a probar diferentes cosas en el software y complementar con recursos y tutoriales en línea [13].

Como se mencionó, se pueden subir rutinas ya creadas a Pepper, para poder utilizarlas, basta con importarlas a la aplicación.

Otra cosa importante a mencionar es que, las rutinas que se creen utilizando Choreographe, tienen compatibilidad directa con *máquinas de estado* utilizando ROS. En la siguiente sección se enseñará sobre las máquinas de estado y se explicará como incluir rutinas creadas en Choreographe en las mismas.

7. Máquinas de Estado: Project Plato

Si se le quiere dar cierto grado de autonomía a Pepper, el tener que utilizar rviz constantemente para hacer maqui navegue, de la misma forma que tener que estar publicando a mano todo lo que se quiere que Pepper diga, no es lo más útil, esto es, al utilizar un joystick para mover a Pepper, darle instrucciones de movimiento sin mayor profundidad o utilizarla nétamente como un modulador de voz, hace que se pierda idea de que Pepper es una máquina autónoma. Por ello, en esta sección se profundizará en lo que son las máquinas de estado para lograr que Maqui tome decisiones sobre qué decir, hacia dónde moverse y, en general, cómo reaccionar a diferentes situaciones de manera autónoma.

7.1. Super Smach ROS: Melee

Figura 11: A new foe has appeared!

No, tristemente Pepper no es el nuevo luchador confirmado para Smash, y no, tampoco vamos a armar una batalla campal entre Pepper, Jaime y Bender, este es otro Smach.

Si bien a continuación se explicará de manera genérica cómo realizar determinadas máquinas de estado para Pepper, es MUY recomendado revisar previamente la documentación de SMACH y, particularmente, realizar los tutoriales del paquete [14].

Antes de comenzar a crear máquinas de estado con Pepper, es importante conocer un componente importante dentro del sistema Maqui, nos referimos a las *maqui skills*. En palabras muy simples, son *puentes* entre las habilidades de fábrica de Pepper y la construcción que se le dio en el laboratorio. Gracias a ellas, se puede acceder de manera sencilla a las *habilidades* de Pepper.

Pese a ser una parte fundamental dentro del trabajo con Maqui, el desarrollo de las principales *skills* ya se encuentra hecho por lo que el trabajo en ellas será como si fueran cajas negras.

Teniendo en cuenta lo anterior y asumiendo un manejo mínimo en SMACH, como se mencionó, a continuación se presentan diferentes máquinas de estado muy simples donde se utilizan algunas de las principales *skills* de Maqui. Si bien no son todas las máquinas que se encuentran disponibles,

sí son un buen inicio para comenzar a crear tus propias máquinas de estado tanto con las cosas ya existentes, como con utilidades que se implementen en Pepper a futuro. Los archivos correspondientes a los siguientes códigos se encuentran ubicados en la cabeza de Maqui en la carpeta /MaquiDemo /2022_working. Para ejecutarlos basta situarse en la carpeta y correr el comando:

Run State Machine

```
python your_state_machine.py
```

7.2. Top Most Popular Pepper State Machines

7.3. Dummy State Machine

Dummy State

```
#!/usr/bin/env python
__autor__ = "Poppers"
from maqui_skills import robot_factory
import rospy
import smach

class DummyState(smach.State):
    def __init__(self,robot):
        smach.State.__init__(self, outcomes=['succeeded','aborted'])
    def execute(self, userdata):
        return 'succeeded'


def getInstance(robot):

    sm = smach.StateMachine(outcomes=['succeeded','aborted'])
    sm.userdata.test_data = "data"
    with sm:
        smach.StateMachine.add('Dummy1', DummyState(robot),
            transitions={
                'succeeded':'Dummy2',
                'preempted':'Dummy1'
            })
```

```
        smach.StateMachine.add('Dummy2', DummyState(robot),
            transitions={
                'succeeded':'succeeded',
                'preempted':'Dummy2'
            })
    return sm

if __name__ == '__main__':

    import os
    if os.environ['UCHILE_ROBOT']=="bender":
        from bender_skills import robot_factory
    else:
        from maqui_skills import robot_factory

    rospy.init_node('test')
    robot = robot_factory.build(["add_needed_skills_here"], core=False)
    sm = getInstance(robot)

    outcome = sm.execute()
```

7.4. Speech

Basta con agregar un estado que llame la función *SpeakGestures(str)*, como se mostrará acontinuación. Algo interesante de esta función es que permite agregar gestos importados de Choreographe dentro del discurso, como se muestra a continuación:

Speak With Gestures

```
smach.StateMachine.add('INTRODUCE',SpeakGestures(robot,text="Greetings.
^start(animations/Stand/Gestures/Hey_1) My name is "+robot.name),
transitions={'succeeded':'succeeded'}
)
```

Otra State Machine análoga a la anterior resultaría utilizando la función *Speak(robot,text="str")* en lugar de *SpeakGestures()*, sin embargo en esta no se pueden llamar animaciones desde Choreographe.

De manera similar, se pueden crear funciones más complejas que produzcan los mismos resultados, esto es, escribir el texto que Maqui debe decir en una clase, para luego llamarla en un estado

Speech Class

```
class Introduce(smach.State):
def __init__(self,robot):
    smach.State.__init__(self, outcomes=['succeeded'])
def execute(self, userdata):
    robot.tts.say_with_gestures("Hello John and hello")
    robot.tts.say_with_gestures("our guest has arrived to the party.")
    return 'succeeded'
)
.
.
.
smach.StateMachine.add('Introduce',Introduce(robot),
        transitions={'succeeded':'succeeded'})
```

Para usar este estado es necesario importar la skill ***tts***, además de: from uchile_states.interaction.states import *

7.5. Hear

7.5.1. Compilar Diccionarios

Antes de poder utilizar las habilidades de escuchar de Maqui, es necesario compilar los bancos de palabras que se esperan como respuesta al momento de escuchar. Para ello, se debe crear un archivo de texto con extensión .bnf con las distintas palabras que se podrían esperar como respuestas. Un ejemplo de un diccionario con nombres, se muestra a continuación:

Diccionario

```
!grammar names;
!start <_main_>;

<_main_> : [<_response>] <_names>;

<_names> : <_males_names> | <_females_names>;

<_males_names> : alex | charlie | francis | james | john | michael;

<_females_names> : elizabeth | jennifer | linda | mary | patricia;

<_response> : my name is;
<_vocative> : robot | pepper | maqui;
<_noise> :  hum | wa | sh | ch | s | mm | pu | tu | ss | huh | naa ;
```

Una vez esté listo el diccionario, es necesario compilarlo. En general en Maqui se guardan los diccionarios en la carpeta Grammar. Se recomienda crear una carpeta dentro de Grammar y guardar el archivo .bnf ahí, al compilar se generará un archivo con el mismo nombre que el creado, pero con extensión .lcf.

Para usar el compilador, se necesita abrir una nueva terminal en Pepper, con el modo autónomo desactivado, dentro de la carpeta grammar, se sigue el siguiente formato:

compile_dics.py

```
python compile_dics.py
    --path=[folder inside of /grammar that you want to compile]
    --filename=[ name of the diccionary to compile ]
    --language=[Language]
- Example:

    Compile the dictionary confirmation on the folder restaurant:
        python compile_dics.py --path=/restaurant --filename=confirmation.lcf

    Compile all dictionarys in folder restaurant
        python compile_dics.py --path=/restaurant
```

7.6. Hear State

Para ejecutar Hear, es necesario entregarle al estado el path y el nombre del diccionario a utilizar, lo cual debe estar dentro de Grammar.

Hear State

```
smach.StateMachine.add('HEAR', Hear(robot, dictionary = 'folder/dictionary'),
            transitions = {
                'succeeded': 'succeeded',
                'preempted': 'preempted'
            },
            remapping = {
                'recognized_sentence':'recognized_sentence'}
        )
```

Lo que sea que escuche Maqui, se guardará como *string* en una variable a la cual se puede acceder entregándola como input a otro estado, como se muestra a continuación, donde se re-mapeo el output de Hear como *recognized_name1*:

Hear State

```
class Introduce(smach.State):
    def __init__(self,robot):

        smach.State.__init__(self, outcomes=['succeeded'],
                             input_keys=['recognized_name1'])
    def execute(self, userdata):
        robot.tts.say_with_gestures("Hello John.")
        robot.tts.say_with_gestures("our guest" + userdata.recognized_name1
        + "has arrived to the party.")
        return 'succeeded'
```

Para usar el estado *Hear()* es necesario importar la skill ***audition***, además de:

from uchile_states.interaction.states import *

7.7. MoveTo

MoveTo corresponde a la forma que existe de publicar posiciones a las que se quiere llegar en el mapa cargado, sin necesidad de utilizar rviz con *nav_goal*. Para usarlo en una máquina de estado, se debe crear una clase como la que se muestra a continuación y luego usarla en un estado.

Move To

```
class GoNav(smach.State):
    def __init__(self,robot):
        smach.State.__init__(self, outcomes=['succeeded'])
    def execute(self, userdata):
        m = Move()
        x = 2.77
        y = 3.25
        w = 92
        m.set_pose(x,y,w)
        m.go()
        return 'succeeded'
.
.
.
        smach.StateMachine.add('GO_DOOR', GoNav(robot),
            transitions={
                'succeeded':'suceeded'
            }       )
```

La función debe recibir los parámetros x e y del mapa, es decir, el punto al que queremos llegar. Esto lo hacemos haciendo navegar de al robot al punto que queremos, como se mostró en la subsección 5.3 y luego ver la posición del robot el el tópico /amcl_pose:

amcl_pose

```
rostopic echo \amcl_pose
```

Se deben copiar los valores $\boldsymbol{x}$ e $\boldsymbol{y}$. El valor $\boldsymbol{w}$ representa la orientación del robot y se obtiene al correr el script de python que se encuentra en *maqui_bringup/src* llamado *pose.py*. El valor que se imprima en la terminal y que no sea 0, representa la orientación del robot en ese momento en el mapa. Es necesario transformar este valor a grados sexagesimales, ya que *pose.py* los entrega en radianes.

Para usar la navegación en en máquinas de estado, basta importar: from uchile_states.navigation.move_to import *

7.8. Choreographe states

Al abrir Choreographe e ir al apartado de Robot applications. Ahí se debe buscar el *beheavior* que se quiere usar en la máquina de estadoy tomar nota de la "carpeta.[en] la que se encuentra y de la ID del *beheavior*. La ID se obtiene dándole click derecho al *beheavior* que se quiera y copiar su ID, para luego llamarlo desde un script de python como se muestra a continuación:

Choreographe State

```
class choreographe(smach.State):
    def __init__(self,robot):
        smach.State.__init__(self, outcomes=['succeeded'])
    def execute(self, userdata):
                robot.behavior_manager.run_behavior("behaviors/beheavior_ID")
                return 'succeeded'


.
.
.
        smach.StateMachine.add('industrial', industrial(robot),
            transitions={
                'succeeded':'succeeded'
            }       )
```

Para usar este estado es necesario importar la skill ***behavior_manager***

Para terminar, es importarse darse cuenta que, cualquier aplicación que se le agregue al robot, es fácilmente utilizable en una máquina de estado. Además, estos no son los únicos states con los que ya cuenta Pepper, es muy importante que te des una vuelta por la carpeta *uchile_states* donde hay bastantes cositas entretenidas y muy útiles que ya están escritas y puedes utilizar como *plug and play* en tus máquinas de estado.

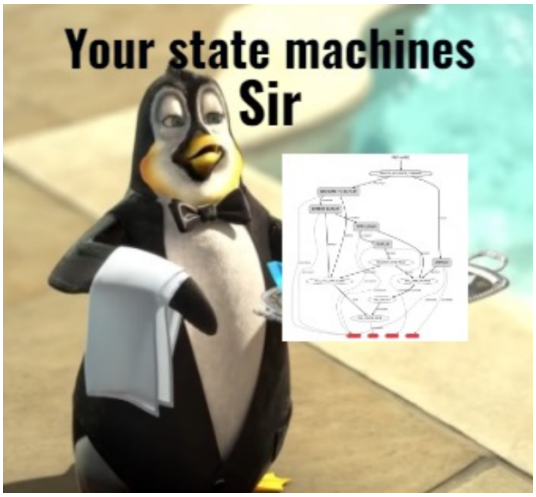


Figura 12: Your state machines sir.

8. YOLOv5

YOLOv5 es un modelo de la familia de modelos computacionales *You Only Look Once*. YOLOv5 es comúnmente utilizado en la detección de objetos [15].

Si bien, YOLOv5 funciona únicamente con python 3, el equipo Jaime [16] del laboratorio realizó las configuraciones para poder utilizar esta versión de YOLO en python 2 y con ROS melodic, a continuación se muestra el proceso de instalación para obtener detecciones de objetos con Maqui como las que se muestran en la figura 13.

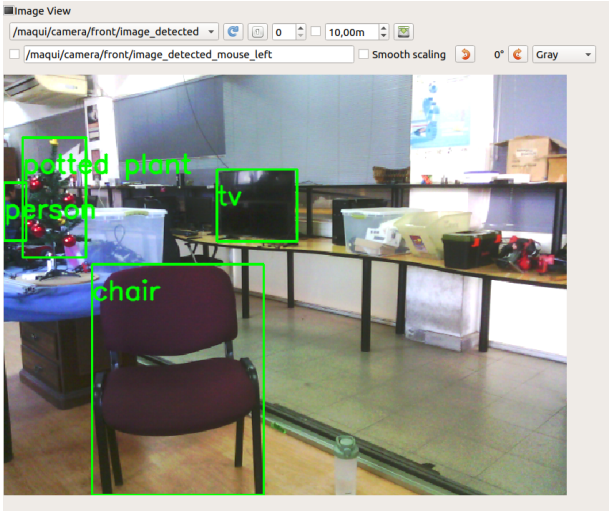


Figura 13: YOLO siendo utilizado con Pepper.

YOLOv5 installation

```
cd soft_ws/src
git clone https://github.com/uchile-robotics/uchile_vision
cd  uchile_vision/yolov5/
sudo python setup.py install
cd src/yolov5/
python2.7 -m pip --no-cache-dir install -r requirements.txt
python2.7 -m pip install future

cd .. #usar este comando hasta volver al directorio soft_ws/

#Compilar
catkin_make

git checkout feat-NewMaqui
```

Para usar YOLO en el robot, basta correr el archivo ***maqui_detector.py***. Por defecto, se utilizan los pesos del archivo yolov5_jp.pt. Un error común al correr el archivo detector viene dado por una función *SiLU* que no existe, para solucionarlo, basta con correr los siguientes comandos en una nueva terminal:

Ejecutar en nueva terminal

```
cd .local/lib/python2.7/site_packages/torch/nn/modules
```

En la terminal anterior ejecutar lo siguiente como un solo comando:

```
cat >> ~/activation.py <<"EOF"

class SiLU(Module):  # export-friendly version of nn.SiLU()
    @staticmethod
    def forward(x):
        return x * torch.sigmoid(x)
EOF
```

Una vez se haya agregado la información necesaria al archivo *activation.py*, se puede volver a ejecutar *maqui_detector.py* y si todo salió bien, al correr el comando rqt_image_view en una nueva terminal (no olvidar exportar el ROS master), se abrirá una ventana que permitirá ver los tópicos que contienen imágenes, estos son las cámaras de Maqui, además de un nuevo tópico donde, además de la imagen recibida por Pepper, se encontrarán las detecciones hechas por la red/maqui/camera/front/image_detected. Si bien al momento de revisar el tópico de detecciones se puede notar un retraso considerable en las imágenes recibidas, las detecciones son muy precisas.

9. Maqui Cosas

9.1. Botón de emergencia

En el posible caso de que Maqui deje de responder, esté ejecutando cosas que podrían lastimarle o se derramó líquido cerca de éste, existe un botón de emergencia que apagará instantáneamente a Maqui, solamente debes presionar cuidadosamente la parte blanda que tiene en su espalda.

ADVERTENCIA: No se guardará ningún dato asociado a programas/códigos en curso al momento de presionar el botón. Procura usarlo cuando realmente sea necesario.

Figura 14: Activar el botón de emergencia.

PD: Al presionar el botón Maqui tiende a realizar una prueba de fé, así que asegurate de que alguien esté con los brazos extendidos para que Maqui sienta el cariño humano evitando que caiga al piso y no quede peor de lo que ya está.

Después de presionar el botón, éste quedará bloqueado, para desbloquearlo y poder volver encender a Maqui debes abrir la parte blanda y girar el botón en sentido horario.

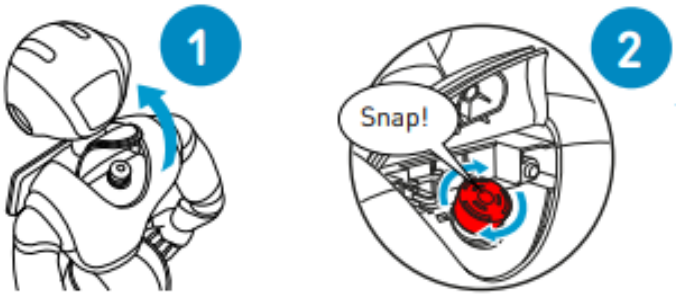


Figura 15: Desbloquear el botón de emergencia.

9.2. Retirar tapa de la cabeza de Maqui

En el caso que necesites conectarte a Maqui vía ethernet (o LAN) tendrás que retirar la tapa que se encuentra en la parte trasera de su cabeza. Para esto debes abrir la parte blanda de su espalda superior, luego verás que al lado del botón de emergencia hay una llave con un llavero de Aldebaran, esa llave debes insertarla en los dos agujeros que están debajo de la tapa que quieres retirar.

¡Felicidades! Abriste la cabeza de Maqui.

9.3. Cambiar el Idioma de Maqui

Para cambiar el idioma de Pepper, basta con acceder a la web como se indicó en la subsección 3.1. En el apartado de configuración avanzada será posible seleccionarle el idioma de entre los que se encuentran disponibles. Obs: Sólo están disponibles los idiomas Chino, Inglés y Español. Los demás son DLC's que se deben comprar aparte.

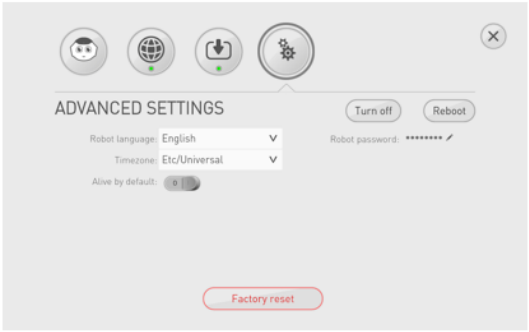


Figura 16: Advanced Settings Pepper Web Interface.

Referencias

[1] ROS Wiki. *ROS Tutorials*. 2022. URL: http://wiki.ros.org/ROS/Tutorials (visitado 21-03-2023).

[2] Uchile HomeBreakers. *uchile_system*. 2022. URL: https://github.com/uchile-robotics/uchile_system/tree/develop (visitado 28-12-2022).

[3] Uchile Peppers. *uchile_system*. 2022. URL: https://github.com/uchile-robotics/uchile_system/blob/maqui-fix-2022/doc/installation.md (visitado 28-12-2022).

[4] SSH. *SSH Protocol – Secure Remote Login and File Transfer*. 2022. URL: https://www.ssh.com/academy/ssh/protocol (visitado 28-12-2022).

[5] Softbank Robotics. *Pepper Web Page*. 2018. URL: http://doc.aldebaran.com/2-5/family/pepper_user_guide/webpage.html (visitado 21-03-2023).

[6] Mike Purvis. *teleop twist keyboard*. 2015. URL: http://wiki.ros.org/teleop_twist_keyboard (visitado 29-01-2023).

[7] Articulated Robotics. *Easy SLAM with ROS using slam_toolbox*. 2022. URL: https://youtu.be/ZaiA3hWaRzE (visitado 31-01-2023).

[8] RViz. *RViz*. 2018. URL: http://wiki.ros.org/rviz (visitado 10-02-2023).

[9] Brian Gerkey. *gmapping*. 2019. URL: http://wiki.ros.org/gmapping (visitado 16-02-2023).

[10] The GIMP Team. *GNU IMAGE MANIPULATION PROGRAM*. 2022. URL: https://www.gimp.org/ (visitado 09-02-2023).

[11] Aldebaran United Robotics Group. *What is Choreographe*. URL: https://fileadmin.cs.lth.se/robot/nao/doc/software/choregraphe/choregraphe_overview.html (visitado 22-02-2023).

[12] Aldebaran United Robotics Group. *Choreographe*. 2022. URL: https://www.aldebaran.com/en/support/pepper-naoqi-2-9/downloads-softwares (visitado 22-02-2023).

[13] Elke Schneider. *Primeros pasos con el robot Pepper virtual y Choregraphe*. 2019. URL: https://youtu.be/ubMuqIF9yRY (visitado 27-02-2023).

[14] Daniel Stonier. *SMACH Tutorials*. 2015. URL: http://wiki.ros.org/smach/Tutorials (visitado 19-03-2023).

[15] Jacob Solawetz. *What is YOLOv5?* 2020. URL: https://blog.roboflow.com/yolov5-improvements-and-evaluation/#:~:text=YOLOv5%5C%20is%5C%20a%5C%20model%5C%20in,offering%5C%20progressively%5C%20higher%5C%20accuracy%5C%20rates. (visitado 20-03-2023).

[16] Jaime's Team. *uchile_vision*. 2022. URL: https://github.com/uchile-robotics/uchile_vision (visitado 20-03-2023).