

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

POSGRADO EN INGENIERÍA ELÉCTRICA

LABORATORIO “BIOROBOTICS“



Manual de usuario del robot Justina.

Índice general

1. Introducción	7
2. Aspectos generales	9
2.1. Cabeza mecatronica	10
2.2. Brazos mecatronicos	11
2.3. Torsó y base holonomica	11
2.4. Diagramas esquemáticos	12
2.4.1. Diagrama esquemático conexiones en las clemas	13
2.4.2. Diagrama esquemático Roboclaws	14
2.5. Comunicación hardware-computadora	15
2.6. Interfaces e interruptores de	17
2.6.1. Paro de emergencia o apagado forzado	17
2.7. Uso y seguridad de la batería	17
2.7.1. Instrucciones de uso y seguridad para baterías LiPo (POLÍMERO DE LITIO)	18
2.7.2. Proceso de carga:	18
2.8. Precauciones y conexión de la Roboclaw	19
2.8.1. Precauciones	19
2.8.2. Modo de conexión de la roboclaw	22
3. Primeros pasos	23
3.1. Software necesario	23
3.2. Obtención de la carpeta de Git hub	24
3.3. Instalación completa del software de Justina	25
3.4. Cómo compilar los repositorios de Justina	26
3.5. RViz y GUI de Justina	27
3.6. Simulación en el RViz y GUI de Justina	28
4. Software	29
4.1. VIRBOT	29
4.2. Guia principal para el desarrollo de codigo	29
4.3. Estructura de la carpeta	30
4.4. Nodos de ROS	32
4.4.1. Nodo /robot_state_publisher	34

4.4.2. Nodo /hardware/head	38
4.4.3. Nodo /hardware/hokuyo_node	38
4.4.4. Nodo /hardware/joy	39
4.4.5. Nodo /hardware/left_arm	40
4.4.6. Nodo /hardware/mobile_base	42
4.4.7. Nodo /hardware/torso	43
4.4.8. Nodo /hri/human_follower	43
4.4.9. Nodo /hri/justina_gui	44
4.4.10. Nodo /hri/leg_finder	53
4.4.11. Nodo /hri/qr_reader	53
4.4.12. Nodo /hri/rviz	54
4.4.13. Nodo /hri/sp_gen	55
4.4.14. Nodo /interoperation/joystick_teleop	55
4.4.15. Nodo /manipulation/ik_geometric	57
4.4.16. Nodo /manipulation/manip_pln	58
4.4.17. Nodo /navigation/localization/loc_amcl	60
4.4.18. Nodo /navigation/localization/map_server	62
4.4.19. Nodo /navigation/mvn_pln	64
4.4.20. Nodo /navigation/obs_avoid/obstacle_detector	70
4.4.21. Nodo /navigation/path_planning/path_calculator	70
4.4.22. Nodo /navigation/path_planning/simple_move	72
4.4.23. Nodo /vision/face_recog	74
4.4.24. Nodo /vision/line_finder	75
4.4.25. Nodo /vision/obj_reco	76
4.4.26. Nodo /vision/skeleton_finder	76
A. Hardware	77
A.1. Actuadores y sus controladores	77
A.1.1. Servomotor MX-106	77
A.1.2. Servomotor MX-64	78
A.1.3. Servomotor MX-28	80
A.1.4. Motor-DCX32L GB KL 12V	81
A.1.5. USB2Dynamixel adapter	85
A.1.6. Roboclaw 2x30A	86
A.1.7. Roboclaw 2x15A	86
A.2. Vision, navegación y sonido	88
A.2.1. Kinect	88
A.2.2. Hokuyo UHG-08LX	89
A.2.3. Microfono RODE	90
A.3. Alimentación de Justina	91
A.3.1. Alimentación bateria Li-po	91
A.3.2. ATX configuración	92
A.4. HUBs	92

A.4.1. Cisco-Linksys USB2HUB4 USB 4-Port Hub	92
A.4.2. HUB Startech ST4300PBU3	93
B. Software	95
B.1. ROS Introducción	95
B.1.1. Instalación de ROS indigo para Ubuntu 14.04	96
B.1.2. Configura tus repositorios de Ubuntu	96
B.1.3. Prepara tus sources.list	97
B.1.4. Configura tus llaves	97
B.1.5. Instalación	97
B.1.6. Inicializar rosdep	98
B.1.7. Configuración del entorno	99
B.1.8. rosinstall	99
B.2. Instalación de PrimeSense drivers	99
B.3. Instalación de OpenCV 2.4.9	99
B.4. Instalando otros paquetes de ROS	100

CAPÍTULO 1

Introducción

El presente manual de usuario tiene como propósito ser una guía clara y específica para garantizar el uso y funcionamiento óptimo de Justina “el robot de servicio”, resolver las dudas más comunes de esta, así como el desarrollo de una replica para la investigación sobre los robots de servicio.

Comprende de la descripción general de Justina, así como de: el hardware empleado en ésta y, el software desarrollado para su funcionamiento.

Contempla todo lo relacionado al hardware empleado (datasheet), así como las ligas para ver especificaciones más avanzadas de este, son dados para el lector interesado en una explicación más detallada. También, este documento contiene los algoritmos creados por los desarrolladores del laboratorio de Biorobotics. Para cada modulo está incluida una breve descripción del algoritmo, técnicas o enfoques usados para el diseño.

Cabe señalar, que, este documento esta sujeto a actualizaciones conforme sean requeridas por actualizaciones en Justina.

El robot de servicio Justina fue diseñado en el laboratorio Biorobotics, de la facultad de ingeniería de la UNAM.

En la figura 1.1 se muestra el robot de servicio Justina.



Figura 1.1: El Robot Justina

CAPÍTULO 2

Aspectos generales

En este capítulo abordaremos los aspectos más comunes sobre el robot Justina en el aspecto físico. Exploraremos la configuración mecánica de Justina. Veremos de forma general el hardware que integra a Justina y marcaremos aspectos importantes para su correcto funcionamiento y un desempeño óptimo.

El cuerpo de Justina esta conformado por una cabeza mecatronica, brazos mecatronicos, una columna que soporta la cabeza y los brazos y una base que se encarga del desplazamiento de Justina; así como de controlar gran parte del hardware que la integra.

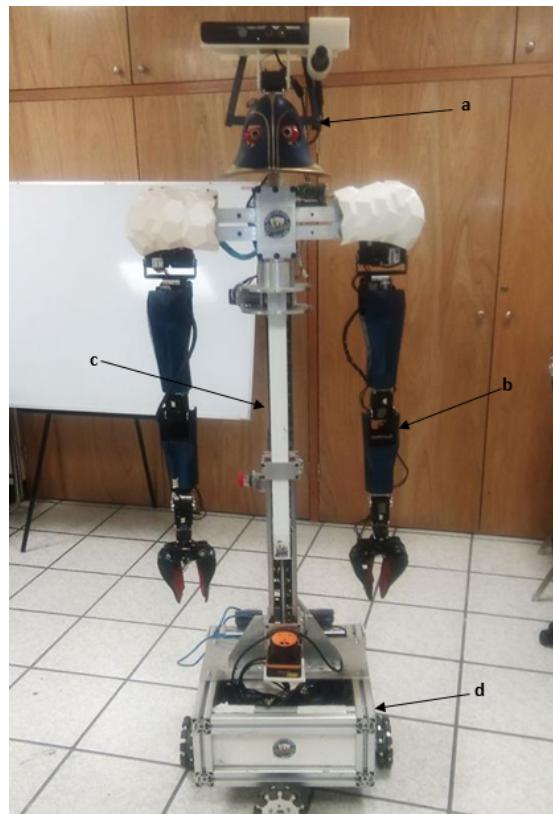


Figura 2.1: (a):Cabeza mecatronica-(b):Brazo mecatronico-(c):Torso-(d):Base holonomica

2.1. Cabeza mecatronica

La cabeza mecatronica está formada por un sensor kinect el cual tiene integrado varios componentes para ayudar a llevar a cabo "ciertas" tareas. Cuenta con una cámara la cual será utilizada para la visión y la detección de objetos apoyado por la cámara de color, estos son utilizados para reconocimiento de patrones y compararlos en la base de datos para reconocer el objeto.

La cabeza mecatronica cuenta con 2 grados de libertad, 1 hace un movimiento vertical (de inclinación hacia adelante y atras) y otro para un movimiento rotatorio; a los dos movimientos se les conoce como til y pan respectivamente.

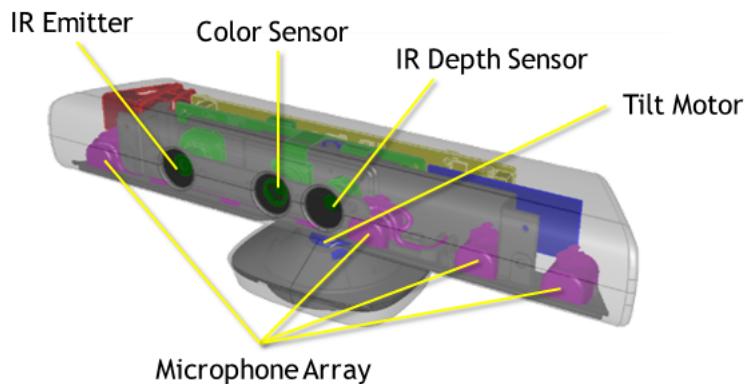


Figura 2.2: Componentes internos del sensor kinect

Para la adquisición de audio cuenta con un micrófono RODE. Para el movimiento cuenta con dos servomotores para darle dos grados de libertad, cuenta con un MX-106 y un MX-64. Para controlar los servomotores se hace uso de un USB2Dynamixel el cual tiene una conexión R232.

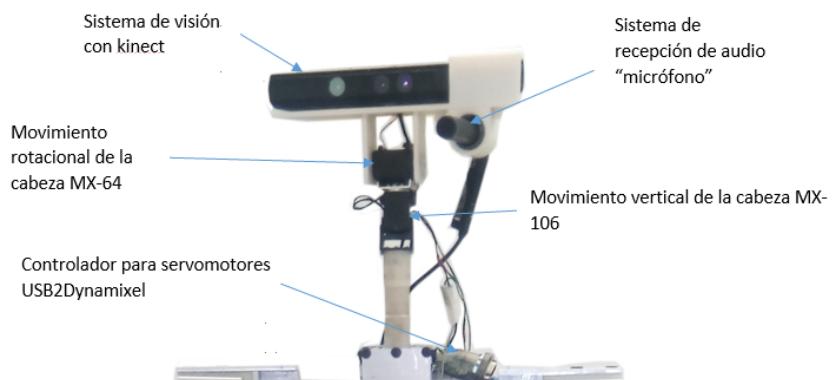


Figura 2.3: Cabeza mecatronica de Justina

2.2. Brazos mecatronicos

Los brazos mecatronicos cuentan con 10 servomotores para darle 7 grados de libertad simulando un brazo humano, además de contar con un griper capaz de agarrar objetos. Está articulado para que cumpla casi los mismos movimientos de un brazo humano.

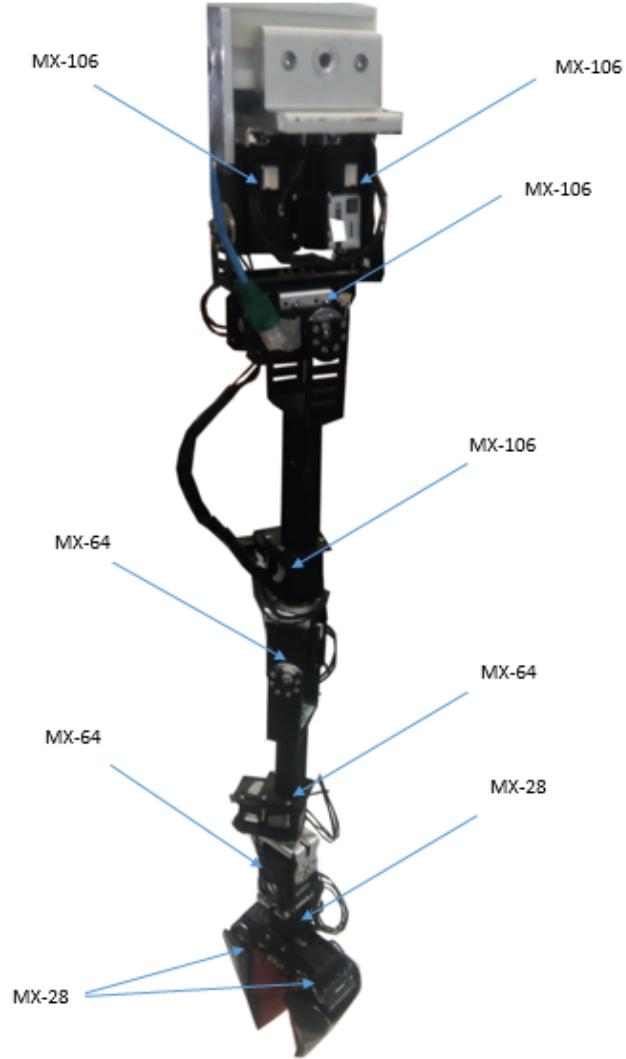


Figura 2.4: Brazo mecatronico de Justina

2.3. Torso y base holonomica

El torso de justina aparte de ser el eje central y soporte de los brazos y la cabeza tiene una guía para llevar todo el cableado de la cabeza y los brazos a la base. El torso sirve principalmente para ayudar a Justina a agarrar los objetos con el griper más fácilmente.

La base holonomica es donde se encuentra toda la etapa de potencia y ejecuta el desplazamiento de Justina. Para la etapa de potencia cuenta con unas clemas, las cuales distribuyen el voltaje entre todos los dispositivos que integran a Justina. Las clemas más grandes son utilizadas para conectar todas las tierras y las más pequeñas son utilizadas para distribuir el voltaje. La alimentación de los servomotores es controlada utilizando dos tipos de roboclaw.

Para el desplazamiento de Justina se utilizaron cuatro motores, los cuales tienen ensambladas llantas cada uno las cuales le permiten un desplazamiento en cualquier dirección.

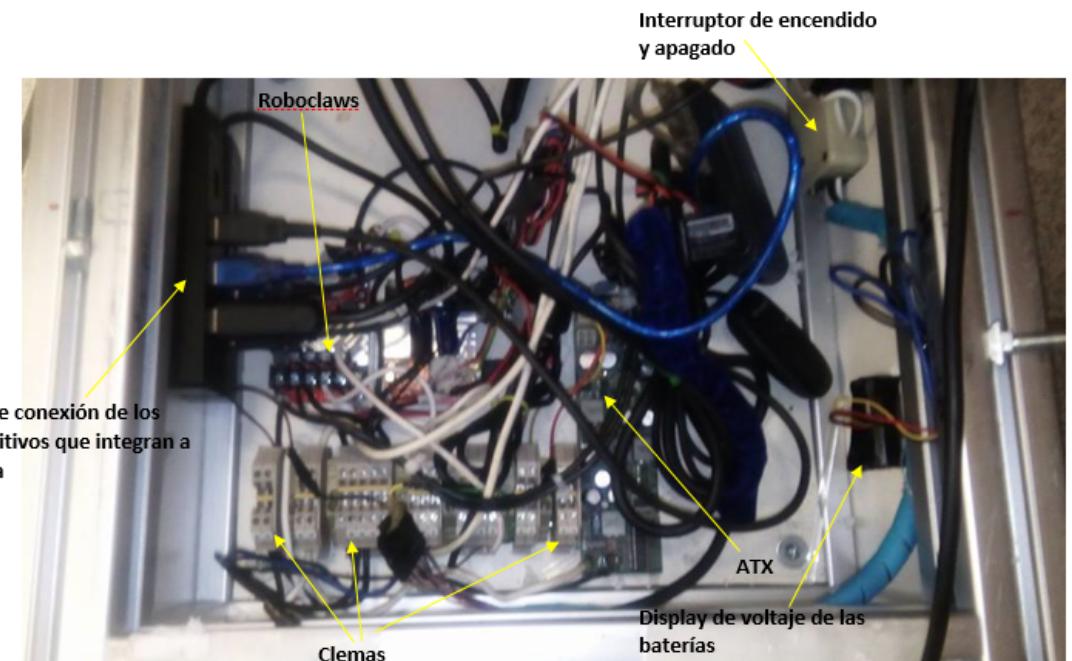


Figura 2.5: Interior de la base de Justina

2.4. Diagramas esquemáticos

Se muestran los diagramas esquemáticos de las conexiones del hardware de Justina.

2.4.1. Diagrama esquemático conexiones en las clemas

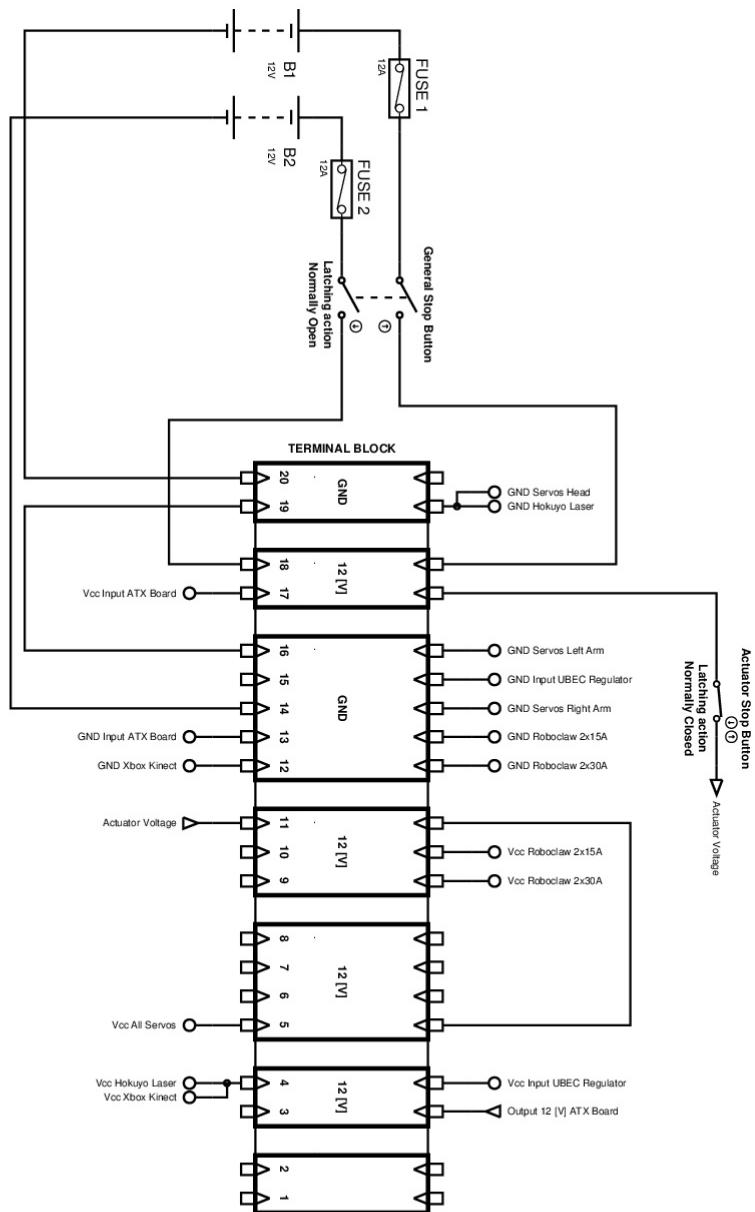


Figura 2.6: Diagrama general

2.4.2. Diagrama esquemático Roboclaws

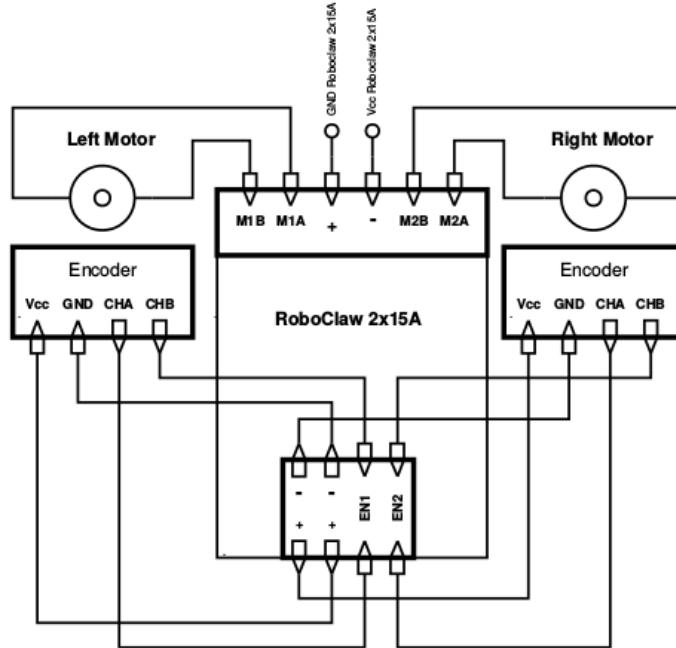


Figura 2.7: Diagrama de la roboclaw 2x15A

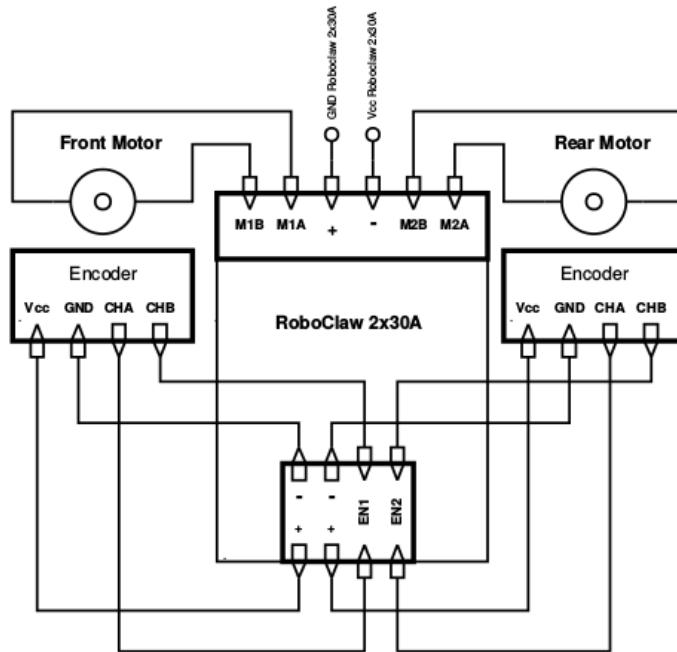


Figura 2.8: Diagrama de la roboclaw 2x30A

2.5. Comunicación hardware-computadora

Para la conexión de Justina con la computadora se hace uso de 3 HUB los cuales están conectados de manera tal que sólo es necesario conectar un HUB a la laptop. El primer HUB (linksys) es utilizado para conectar los actuadores de los dos brazos y la cabeza mecatronica, el segundo HUB (el primer Startech) se utiliza para conectar las dos roboclaw, la 2x30A y la 2x15A, por ultimo tenemos el HUB (el segundo Startech) que es donde llegan los dos HUB anteriores así como el hokuyo y un control de xbox (que es utilizado para mover a Justina), para ser enviado a la laptop.

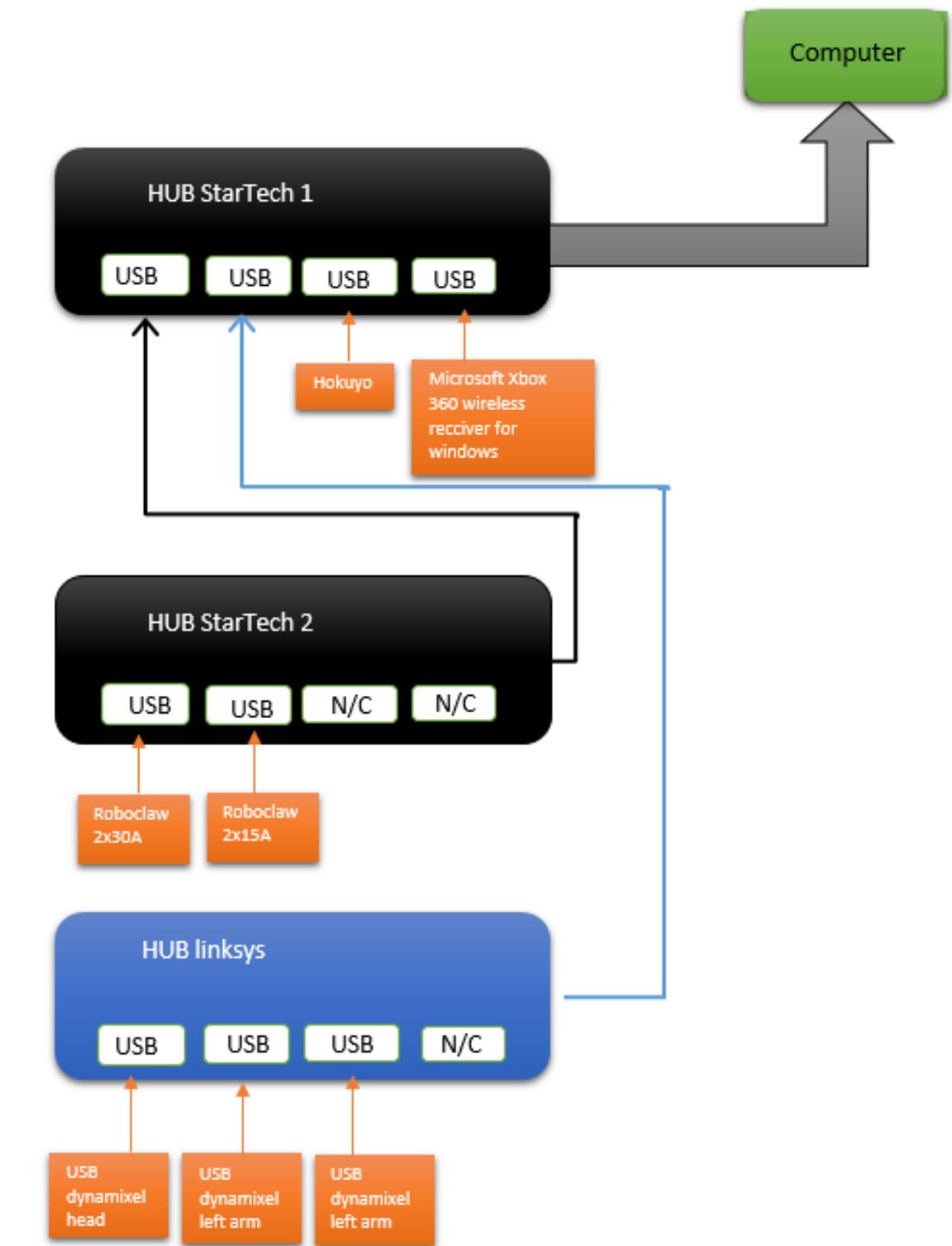


Figura 2.9: Diagrama de conexiones de los HUBs

2.6. Interruptores de encendido y apagado

En la base Justina cuenta con dos botones, el negro que está etiquetado como “ON” sirve para encender a Justina y el botón rojo está etiquetado como “OFF” el cual sirve para apagar a Justina.



Figura 2.10: Encendido y apagado de Justina

También cuenta con dos fusibles para protección del sistema para cuando exista un mal funcionamiento en la alimentación de Justina. Además tiene un display el cual muestra el voltaje actual de las baterías que alimentan a Justina, esto nos sirve para ver que el voltaje con el que cuentan las baterías no este por debajo del voltaje de operación recomendado y no cause un mal funcionamiento en éstas.

2.6.1. Boton de paro de emergencia

El botón de paro de emergencia se encuentra a lado del torso, este botón es de color rojo. La función de este botón es detener todos los actuadores de Justina ante una emergencia como un mal funcionamiento.

2.7. Uso y seguridad de la batería

Es importante saber los límites y voltajes de operación recomendados por el fabricante para el correcto uso de las baterías. A continuación ponemos información pertinente para un correcto uso de las baterías.

2.7.1. Instrucciones de uso y seguridad para baterías LiPo (POLÍMERO DE LITIO)

Normas a seguir para evitar cualquier peligro o mal funcionamiento:

- Emplee sólo cargadores específicos para baterías de Polímero de Litio (LiPo). En caso contrario puede provocar un incendio que derive en daños personales y/o materiales.
- Nunca cargue las baterías LiPo sin estar presente. Siempre debe vigilar el proceso para poder reaccionar ante cualquier problema que se pudiese plantear.
- Si en cualquier momento observa que una batería Lipo se hincha o derrama líquido, desconéctela y obsérvela durante 15 minutos en un lugar seguro y alejado de cualquier material combustible.
- Tenga mucho cuidado de que NUNCA se toquen los dos terminales de la batería, este cortocircuito podría hacer que la batería se incendiase.
- Una batería que haya sufrido un golpe, cortocircuito u otro problema puede llegar a incendiarse incluso 10-15 minutos después de haberse producido este hecho. Lleve rápidamente la batería a un lugar seguro y obsérvela durante 15 minutos.
- NUNCA almacene sus baterías en un vehículo ni en cualquier lugar donde se puedan alcanzar temperaturas altas. Las temperaturas extremas pueden causar el incendio de la batería.
- Tenga mucho cuidado de NO PERFORAR ningún pack de baterías LiPo, puede provocar un incendio.

2.7.2. Proceso de carga:

- Nunca cargue las baterías sin vigilarlas y utilice solo cargadores especiales para las baterías de lipo, así como tener en cuenta el número de elementos que contiene su batería.
- Cargue las baterías en un área segura y aislada de cualquier material inflamable.
- Deje enfriar la batería a la temperatura ambiente antes de comenzar la carga.
- Valores nominales de una batería de lipo cargada.

Lipos 2S (2 elementos): entre 8,32 y 8,44V

Lipos 3S (3 elementos): entre 12,48 y 12,66V

Lipos 4S (4 elementos): entre 16,64 y 16,88V

Nunca descargue una batería por debajo de 3V por elemento, puede dañar la batería. Para ello debe tener cuidado de no agotarla más de lo debido empleando dispositivos de corte por bajo voltaje o variadores especialmente diseñados para baterías LiPo.

Fin de vida de las baterías LiPo:

Cuando la capacidad de la batería haya disminuido un 30 %, deberá desecharla. Para ello descárguela a 3V por elemento, aíslle sus terminales, envuélvala en plástico y depositélas en los contenedores especiales para el desecho responsable de pilas.

2.8. Precauciones y conexión de la Roboclaw

2.8.1. Precauciones

Estas son precauciones sumamente importantes que se deberán seguir para evitar daños a la Roboclaw y los sistemas conectados.

1. Desconectar la terminal negativa de la alimentación no es la mejor forma para apagar el motor. Si conectas cualquier entrada o salida a la Roboclaw obtienes un ciclo de tierra a los pines de entrada/salida como resultado. Puede causar daños a la Roboclaw y cualquier dispositivo conectado. Para apagar el controlador del motor debe removese primero la conexión positiva de la alimentación después de que los motores dejen de moverse.
2. El motor de DC puede trabajar como un generador cuando este gira. Siempre detenga el motor antes de apagar la Roboclaw.
3. Apagar en caso de emergencia, un interruptor y/o contacto de un tamaño adecuado debe ser utilizado. Se debe usar un diodo correcto para hacer un puente entre el apagador y el contacto.
4. Dependiendo del modelo de RoboClaw hay un requisito mínimo de potencia de al menos 6V. Bajo cargas pesadas, si la batería lógica y la batería principal se combinan, pueden suceder caídas de tensión. Esto puede causar un comportamiento errático de la RoboClaw.
Vista general de los conectores

En el control principal de entrada/salida, están puestos para una fácil conectividad para controlar dispositivos como controladores RC. Los cabezales están también arreglados para proveer un fácil acceso a tierra y alimentación para suministrar poder a controladores externos.

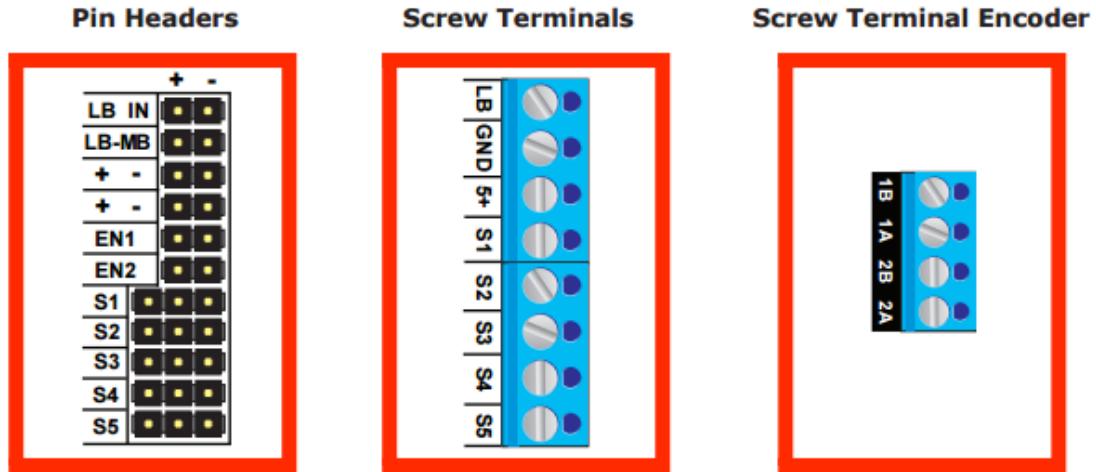


Figura 2.11: Configuración de pines

Batería lógica (LB IN)

La parte lógica de la RoboClaw puede ser alimentada por una batería secundaria conectada a LB IN. La terminal positiva (+) está localizada al borde de la tarjeta y la tierra (-) es el pin más cercano al disipador. Remueva el jumper LB-MB para que la batería secundaria pueda ser usada.

Encoder power (+ -)

Los pines marcados como (+) y (-) son los pines de alimentación de los encoders. El positivo (+) está localizado al borde de la tarjeta y la fuente +5VDC. El pin de tierra (-) está cercano al disipador. En todos los modelos ST la alimentación debe venir del único borne de 5v y a GND

Entradas de los encoder (EN1 / EN2 – 1B / 1A / 2B / 2A)

EN1 y EN2 son las entradas de los encoders en versión pin del RoboClaw. 1B, 1A, 2B y 2A son las entradas de los encoders a los bornes de la RoboClaw. El canal A de ambos EN1 y EN2 están localizados en los pines del borde de la tarjeta. Los pines del canal B están localizados cercanos al disipador en los pines. Los canales A y B están debidamente etiquetados en los bornes.

Cuando conectes los encoders asegúrese que el canal para la dirección de giro esté conectado en A. si un encoder es llevado hacia atrás a el otro, tendrás un contador interno que contara hacia adelante y hacia atrás.

Control de entradas (S1 / S2 / S3 / S4 / S5)

S1, S2, S3, S4 y S5 están configuradas para pines de servo estándar de estilo (tipo) entrada/salida (excepto en modelos ST) , +5V y GND. S1 y S2 son las entradas de los modos de control serial, analógico y RC. S3, S4 y S5 pueden ser usadas como entrada de corte de emergencia o como salidas de control de voltaje.

Bornes de la batería principal

La alimentación de entrada principal puede ser desde 6VDC a 34VDC en la RoboClaw estándar y de 10.5VDC a 60VDC en la Roboclaw de alto voltaje. Las conexiones son hechas en los bornes principales (+) y (-). El símbolo de mas (+) marca la terminal positiva y el negativo (-) marca la terminal negativa. El cableado de la batería principal debe ser lo más corto posible

Desconectar

La batería principal debe ser desconectada en caso de situaciones donde se salga de control y la energía necesite ser cortada. El interruptor debe estar estimado para manejar la máxima corriente y voltaje de la batería. Esto puede variar dependiendo del tipo de motores y/o la fuente de alimentación que se este utilizando.

Bornes del motor

Los bornes del motor están hechos con M1A/M1B para el canal 1 y M2A/M2B para el canal 2. Para que ambos motores giren en la misma dirección, el cableado de uno de los motores debe ser contrario hacia el otro en un robot diferencial típico. El cableado de los motores y la batería deben ser lo más cortos posibles. Los cables largos pueden incrementar la inductancia y por lo tanto incrementan potencialmente los picos de voltaje perjudiciales.

Leds de estado y error

La Roboclaw tiene tres leds. Dos leds de estado, STAT1 y STAT2, y un led de error ERR. Cuando la Roboclaw es alimentada por primera vez, hasta los 3 leds deben parpadear brevemente para indicar que todos los led están funcionando. Los leds se comportaran differently dependiendo en qué modo está configurada la Roboclaw.

Cableado básico

El diagrama de cableado de abajo ilustra la batería básica y conexiones de motor para la Roboclaw. M1A y M1B es el canal de motor 1, junto a M2A y M2B como canal del motor 2.

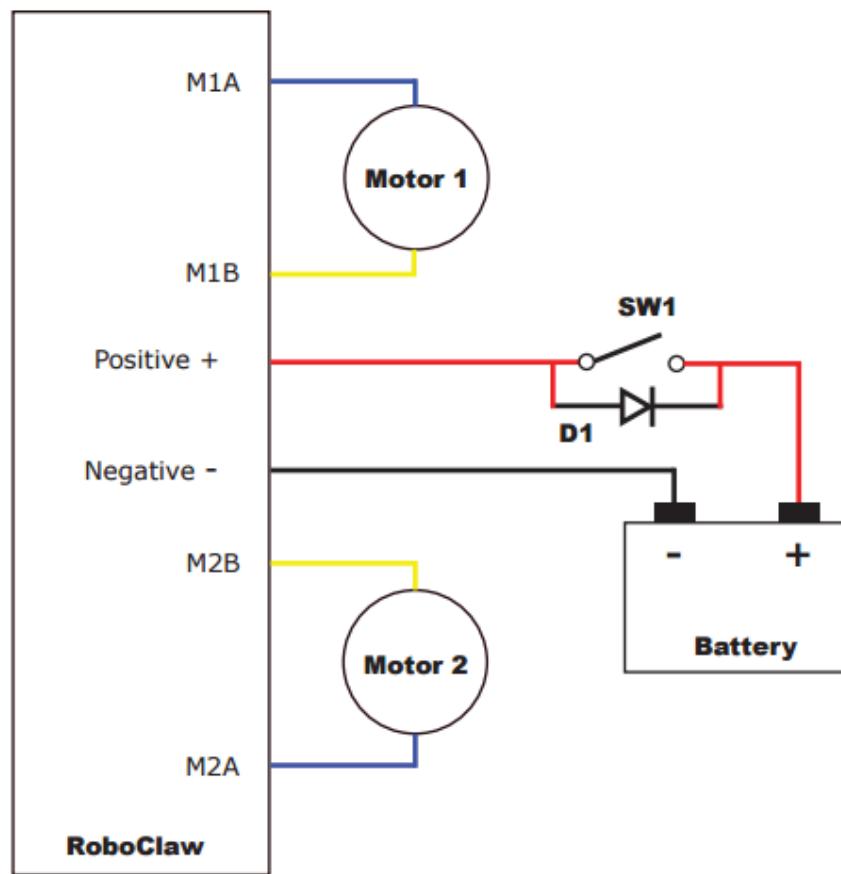


Figura 2.12: Conexión básica de motores

Nunca desconecte la terminal negativa de la batería antes de desconectar la positiva.

2.8.2. Modo de conexión de la roboclaw

CAPÍTULO 3

Primeros pasos

Antes de poner en funcionamiento a Justina, debemos conocer los software con los que trabaja y los requerimientos para su correcto funcionamiento. También debes saber que Justina no funciona únicamente con una computadora, trabaja con 2 actualmente; en un computadora tenemos integrado ubuntu 14.04.1 (esta es la version ya probada), esta es la computadora principal, la cual se encargara de todos los procesos principales de Justina, y tenemos la segunda computadora con windows 7, la cual únicamente utilizaremos para la comunicación el reconocimiento por voz y el habla de Justina

Como primer paso debemos conocer el software necesario para el funcionamiento de Justina.

3.1. Software necesario

Se requiere lo siguiente:

-Ubuntu

- ROS Indigo desktop full
- OpenNI + PrimeSense drivers
- OpenCV 2.4.8 or 2.4.9. Compiled with OpenNi, WITHOUT OpenCL, WITHOUT Cuda, with Eigen
- PCL 1.6

Para conocer la forma de instalar ROS, OpenNI, los drivers PrimeSense y OpenCV 2.4.9 por favor acude al apéndice B (software).

-Windows 7

- Blackboard
- SpRecV1

- SpRecV2

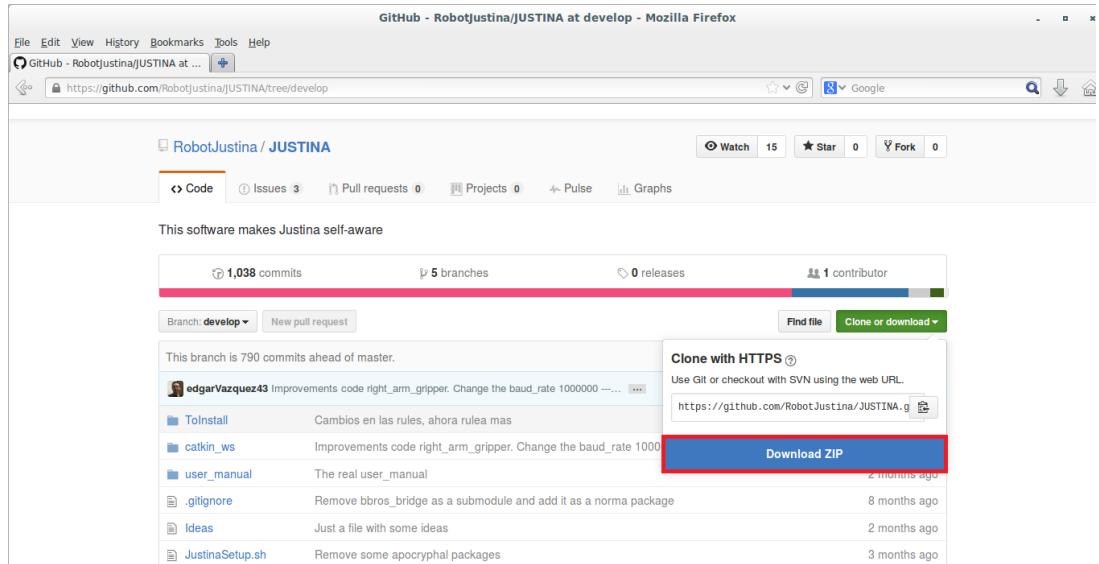
La comunicacion entre las dos computadoras se da por medio de conexion ethernet, para esto se debe configurar una computadora para que sea detectada como red. Para conocer la configuración de la red por favor visite el apendice de software.

3.2. Obtención de la carpeta de Git hub

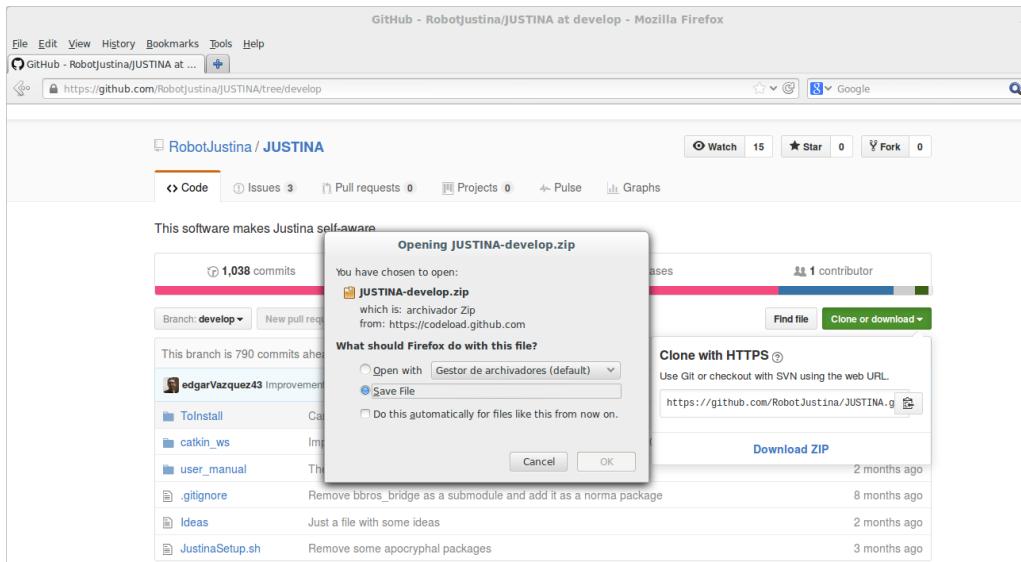
Como siguiente paso obtener el software de Justina, para esto debemos descargar todas las carpetas con las que se ha trabajado Justina.

Todos los repositorios del software de Justina se encuentran en Git hub (así como este manual y es de donde podrás descargar futuras versiones). Existen dos formas para obtener la carpeta contenedora con todo lo necesario para empezar:

La primera es ir a la dirección “<https://github.com/RobotJustina/JUSTINA/tree/develop>” y descargarlo con el botón color verde que dice **clone or download**”.



te saldrá una opción para seleccionar la ubicación en la que deseas guardar el archivo .zip



Busca la carpeta contenedora y descomprime el archivo. Al descomprimirlo obtendrás una carpeta llamada "JUSTINA" la cual contiene todo lo necesario para utilizar a Justina.

La segunda opción consiste en que de desde la terminal clones la carpeta de Justina, usando el siguiente comando: "git clone https://github.com/RobotJustina/JUSTINA".

3.3. Instalación completa del software de Justina

Una vez instalado ROS procedemos a instalar el software de Justina, para esto abrir una terminal y seguir las siguientes instrucciones.

1. ingresamos a la carpeta JUSTINA y ejecutar JustinaSetup.sh

```
kreauznik@K: ~/JUSTINA
Archivo Editar Ver Buscar Terminal Ayuda
kreauznik@K:~/JUSTINA$ cd JUSTINA/
kreauznik@K:~/JUSTINA$ .
catkin_ws/      JustinaSetup.sh  user_manual/
.git/           ToInstall/
kreauznik@K:~/JUSTINA$ ./JustinaSetup.sh
```

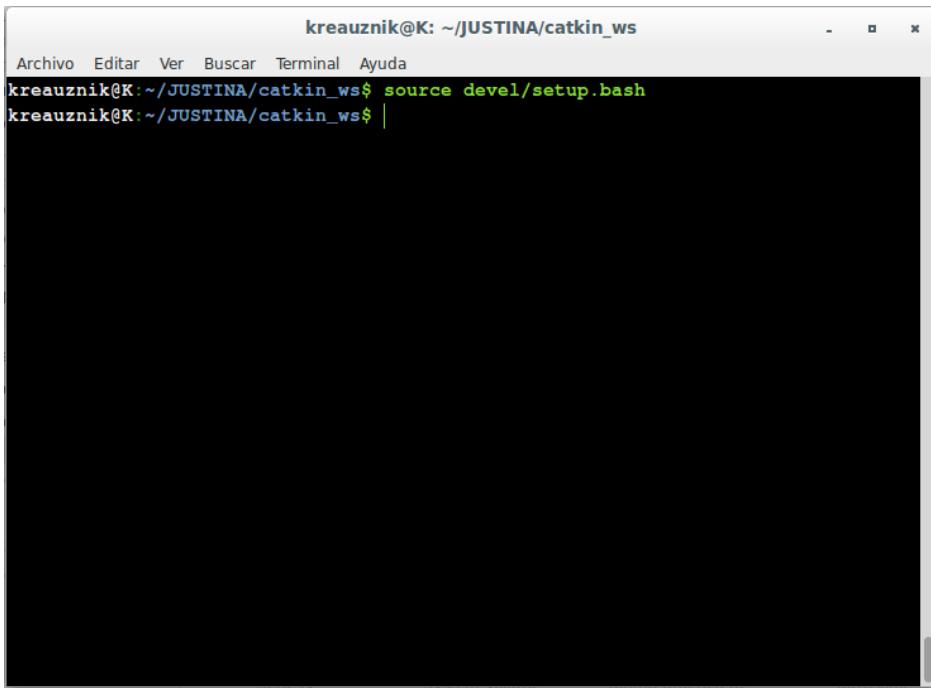
2. Aceptar cada que pregunte. Esto nos llevara varios minutos.
3. Una vez instalado el software, debemos habilitar el uso de los puertos USB para ROS, para esto ingresamos al directorio ”JUSTINA/ToInstall /USB (si quieres seguir las instrucciones más detalladamente, en la misma dirección abrir el archivo “instructions”) una vez dentro de la carpeta ejecutar el siguiente comando ”sudo cp 80-justinaRobot.rules /etc/udev/rules.d/”
4. Te pedirá la contraseña. Una vez termines de ejecutar el comando, se debe ejecutar el siguiente: ”sudo udevadm control –reload-rules && sudo service udev restart && sudo udevadm trigger”

Listo, ya tienes instalado el software de Justina.

3.4. Cómo compilar los repositorios de Justina

Para compilar los repositorios de Justina simplemente ve al directorio ”JUSTINA/catkin_ws”, en este directorio ejecutamos el siguiente comando ”catkin_make”. Esto nos llevara varios minutos.

Una vez compilados todos los repositorios ejecutar el siguiente comando dentro de la misma carpeta ”source devel/setup.bash”.

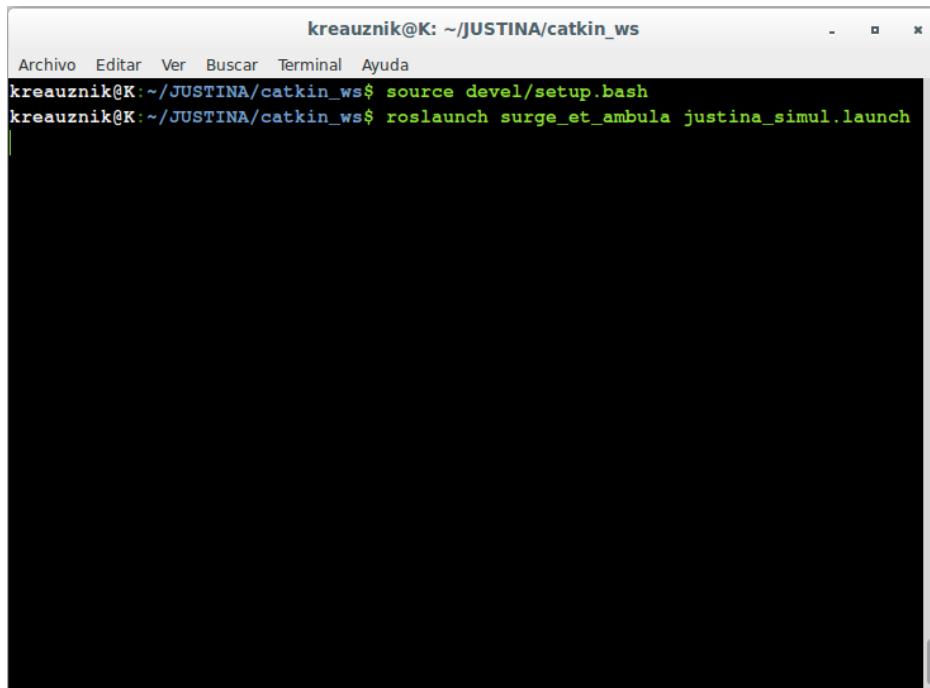


A screenshot of a terminal window titled "kreauznik@K: ~/JUSTINA/catkin_ws". The window has a menu bar with options: Archivo, Editar, Ver, Buscar, Terminal, Ayuda. The main area of the terminal shows the command "source devel/setup.bash" being typed at the prompt. The terminal is black with white text.

Listo, ahora el software de Justina está instalado y los repositorios compilados y listos para usarse.

3.5. RViz y GUI de Justina

Para probar el funcionamiento del hardware y software de Justina utilizaremos RViz y la GUI. Para ejecutar estos programas utilizamos el comando "roslaunch surge_et_ambula justina.launch".

A screenshot of a terminal window titled "kreauznik@K: ~/JUSTINA/catkin_ws". The window has a standard Linux-style title bar with icons for minimize, maximize, and close. Below the title bar is a menu bar with options: Archivo, Editar, Ver, Buscar, Terminal, Ayuda. The main area of the terminal is black and contains white text. It shows two commands being run: "source devel/setup.bash" and "roslaunch surge_et_ambula justina_simul.launch". The cursor is visible as a vertical line at the bottom left of the terminal window.

```
kreauznik@K: ~/JUSTINA/catkin_ws
Archivo Editar Ver Buscar Terminal Ayuda
kreauznik@K:~/JUSTINA/catkin_ws$ source devel/setup.bash
kreauznik@K:~/JUSTINA/catkin_ws$ roslaunch surge_et_ambula justina_simul.launch
```

3.6. Simulación en el RViz y GUI de Justina

Cuando no tenemos conectado el robot Justina a nuestras laptops lo único que podemos hacer es simular a Justina en nuestras laptops, para esto ejecutamos el siguiente comando "roslaunch surge_et_ambula justina_simul.launch".

CAPÍTULO 4

Software

- AI (VIRBOT)
- Navegación
- Visión
- Habla

4.1. VIRBOT

El sistema VIRBOT consiste de varios subsistemas los cuales controlan la operación del robot móvil.

4.2. Guia principal para el desarrollo de codigo

- Todo código fuente DEBE estar contenido en la carpeta *catkin_ws/src*.
- Sólo el código contenido en la carpeta *catkin_ws/src/hardware* puede interactuar con el hardware del robot
- El punto anterior implica que todos los otros programas deberán implementar SÓLO algoritmos. Todas las interacciones con el hardware (e.g.. obtener una imagen desde la cámara, leer el láser, mover la base o la cabeza, hablar, etc.) debe hacerse intercambiando información con los paquetes contenidos en la carpeta **hardware**, a través de los tópicos y servicios de ROS.
- Los códigos contenidos en todas las carpetas dentro de *catkin_ws/src*, excepto las carpetas de herramientas, DEBEN contener sólo código escrito por el propio desarrollador (de cualquier paquete). Todas las bibliotecas necesarias o código de otras fuentes (bibliotecas serial, arduino, julius, dynamixel, etc.), si no están instaladas en algún default path

(/opt/ros, /usr/local/, etc.), deben ser puestas dentro de la carpeta *catkin_ws/src/tools* en una sub-carpeta apropiada.

- Los desarrolladores deben tratar de usar sólo mensajes ya definidos en algún paquete de ROS o pila, sin embargo, si mensajes personalizados son requeridos, éstos deben ser puestos dentro de *catkin_ws/src/subsystem/subsystem_maga*, así que, muchos mensajes pueden ser usados sin necesidad de ejecutar todos los demás subsistemas.

4.3. Estructura de la carpeta

```
catkin_ws
  build
  devel
  src
    hardware
    arms
    battery
    hardware_state
    justina_urdf
    hardware_msgs
    head
    mobile_base
    point_cloud_manager
    speakers
    torso
      hri
    gesture_recog
    hri_msgs
    justina_gui
    natural_language
    speech_recog
      interoperation
    bbros_bridge
    joy_teleop
    pc_teleop
    roah_rsbb
      manipulation
    arms_predef_movs
    arms_path_planning
    arms_trajectory_planning
    head_predef_movs
```

```
head_tracking_point
manipulation_msgs
    navigation
localization
mapping
moving
navigation_msgs
path_planning
point_traking
    planning
planning_msgs
pomdp
rule_based
semantic_database
state_machines
    surge_et_ambula
launch
rviz_files
    testing
any_not_stable_node
    tools
ros_tools
libraries
    serial_arduino
    serial_dynamixel
    julius
    festival
    vision
door_detector
furniture_recog
object_detector
object_recog
person_detection
person_recog
vision_msgs
user_manual
```

Cada paquete en la carpeta de *hardware* debe tener su versión simulada, así que, el resto del software (todas las otras carpetas se supone que contienen sólo algoritmos y no interacción con el hardware del robot) puedes correr inmediatamente el modo de simulación. Eligiendo entre simulado o real debe ser hecho en la carpeta de ejecución.

4.4. Nodos de ROS

Con el comando *rosrun* se puede ejecutar un nodo de un paquete sin tener que conocer la ruta completa, indicando el nombre del paquete dentro del cual se encuentra el nodo que deseamos lanzar y el nombre del ejecutable.

Uso:

```
rosrun package_name executable_name
```

Por otro lado, *roslaunch* es una herramienta para lanzar fácilmente múltiples nodos de ROS de forma local y remota a través de SSH, así como establecer parámetros en el Servidor de Parámetros. Antes de iniciar cualquier nodo, *roslaunch* determinará si *roscore* ya está en ejecución y, en caso contrario, lo iniciara automáticamente. Los archivos *.launch* del robot Justina se encuentran dentro del paquete *surge_et_ambula* en la carpeta *launch*.

Uso:

```
roslaunch package_name file_name.launch
```

Los archivos *launch* son documentos XML, y cada documento XML debe tener un elemento raíz, para el caso de los archivos *launch* de ROS, el elemento raíz se define mediante un par de etiquetas *launch*:

```
<launch>
</launch>
```

Todos los demás elementos del archivo se deben incluir dentro de estas etiquetas.

La etiqueta *<node>* especifica un nodo ROS que se desea lanzar y tiene tres atributos requeridos, ésta etiqueta se ve de esta forma:

```
<node name="node-name" pkg="package-name" type="executable-name" />
```

Los atributos *pkg* y *type* identifican qué programa debe ejecutar ROS para iniciar este nodo. Estos son los mismos dos argumentos que toma el comando *rosrun*, especificando el nombre del paquete y el nombre del ejecutable, respectivamente. El atributo *name* asigna un nombre al nodo. Esto anula cualquier nombre que el nodo se asignaría normalmente a sí mismo en su llamada a *ros::init*.

Existen otros atributos opcionales utilizados en la etiqueta *<node>*:

- `args='arg1 arg2 arg3'`: Pasar argumentos al nodo.
- `output='screen'`: Los nodos iniciados con este atributo mostrarán sus salidas std-out/stderr en la pantalla.

La etiqueta `<group>` facilita la aplicación de configuraciones a un grupo de nodos. Tiene un atributo `ns` que le permite definir un *namespace* independiente para un grupo de nodos.

```
<group ns="namespace">
    <node name="node-name" pkg="package-name" type="executable-name" />
</group>
```

La etiqueta `<group>` es equivalente a la etiqueta de nivel superior `<launch>` y simplemente actúa como un contenedor para las etiquetas que se encuentran dentro. Esto significa que puede usar cualquier etiqueta como se usaría normalmente dentro de una etiqueta `<launch>`.

Los nodos de ROS también admiten reasignaciones, que proporcionan un nivel de control para modificar los nombres utilizados por los nodos. Las reasignaciones se basan en la idea de sustitución: cada reasignación proporciona un nombre original y un nuevo nombre. Para reasignar nombres dentro de un archivo *launch* se utiliza la etiqueta `<remap>`:

```
<remap from="original-name" to="new-name"/>
```

Si la etiqueta `<remap>` aparece en el nivel superior como hijo de la etiqueta `<launch>`, esta reasignación se aplicará a todos los nodos subsiguientes. Estos elementos de reasignación también pueden aparecer como hijos de una etiqueta `<node>`, en este caso, los reasignamientos dados se aplican solamente al nodo en el que estén contenidos, como en este ejemplo:

```
<node name="node-name" pkg="package-name" type="executable-name" >
    <remap from="original-name" to="new-name" />
</node>
```

La etiqueta `<param>` define un parámetro que se establecerá en el Servidor de Parámetros. Esta etiqueta, como uno habría de esperar, asigna el valor dado al parámetro con el nombre dado.

```
<param name="param-name" value="param-value" />
```

La etiqueta `<param>` se puede colocar dentro de una etiqueta `<node>`, en cuyo caso el parámetro se trata como un parámetro privado.

```
<node name="node-name" pkg="package-name" type="executable-name" >
    <param name="param-name" value="param-value" />
</node>
```

Para obtener más información acerca de estas etiquetas y de los archivos *launch*, por favor consulte: <http://wiki.ros.org/roslaunch/XML>.

4.4.1. Nodo `/robot_state_publisher`

Este nodo se encuentra dentro del paquete *robot_state_publisher*, y permite publicar el estado del robot a *tf*. Una vez que el estado se publica, está disponible para todos los componentes del sistema que utilizan *tf*. El paquete toma las posiciones de las juntas del robot como entrada y publica las poses 3D de los eslabones usando un modelo de árbol cinemático. *tf* es un paquete que permite al usuario realizar el seguimiento de varios marcos de referencia a lo largo del tiempo.

robot_state_publisher usa el URDF especificado por el parámetro *robot_description*, y las posiciones de las juntas del tópico *joint_states* para calcular la cinemática directa del robot y publicar los resultados a través de *tf*. URDF (Unified Robot Description Format) es un formato XML para representar el modelo de un robot.

Tópicos suscritos	/joint_states [sensor_msgs/JointState]	Se suscribe a la información de la posición de las juntas
Tópicos publicados	/tf [tf/tfMessage]	Publica el estado del robot
Parámetros	robot_description [urdf map, default:]	Archivo XML del modelo del robot
	tf_prefix [string, default:]	Establece el prefijo tf para el namespace
	publish_frequency [double, default: 50Hz]	Frecuencia a la que publica el nodo
	use_tf_static [bool, default: false]	Define si se quiere utilizar /tf_static

Tabla 4.1: Nodo /robot_state_publisher

Sintaxis en un archivo launch

Lanzamiento del nodo y ajuste del parámetro *robot_description*.

```
<param name="robot_description" command="cat $(find knowledge)/hardware/justina.xml"/>
<node name="robot_state_publisher" pkg="robot_state_publisher" type="state_publisher"/>$
```

En el archivo *justina.xml* se encuentra el modelo del robot Justina; en la Figura 4.1 se tiene el árbol de transformaciones, los ovalos azules representan las juntas, mientras que los recuadros negros representan los sistemas de referencia asociados a los eslabones del robot. En el grafo dirigido se muestran los offset de traslación en *x*, *y* y *z*, y los offset de los ángulos de rotación *roll*, *pitch* y *yaw* que se tienen entre los sistemas de referencia, las unidades están en metros y radianes respectivamente.

En el grafo dirigido de la Figura 4.2 se muestran todas las transformaciones entre sistemas de referencia para el robot, en éste grafo se incluyen además los marcos *odom* y *map*. Para más información acerca de los sistemas de referencia para plataformas móviles consulte: <http://www.ros.org/reps/rep-0105.html>.

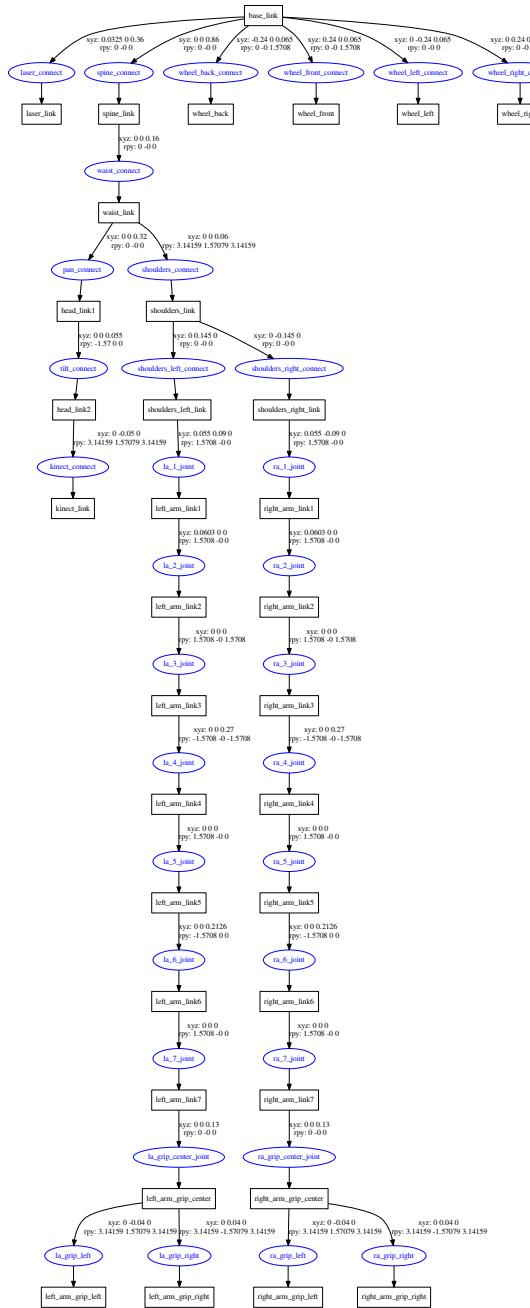


Figura 4.1: Árbol de transformaciones

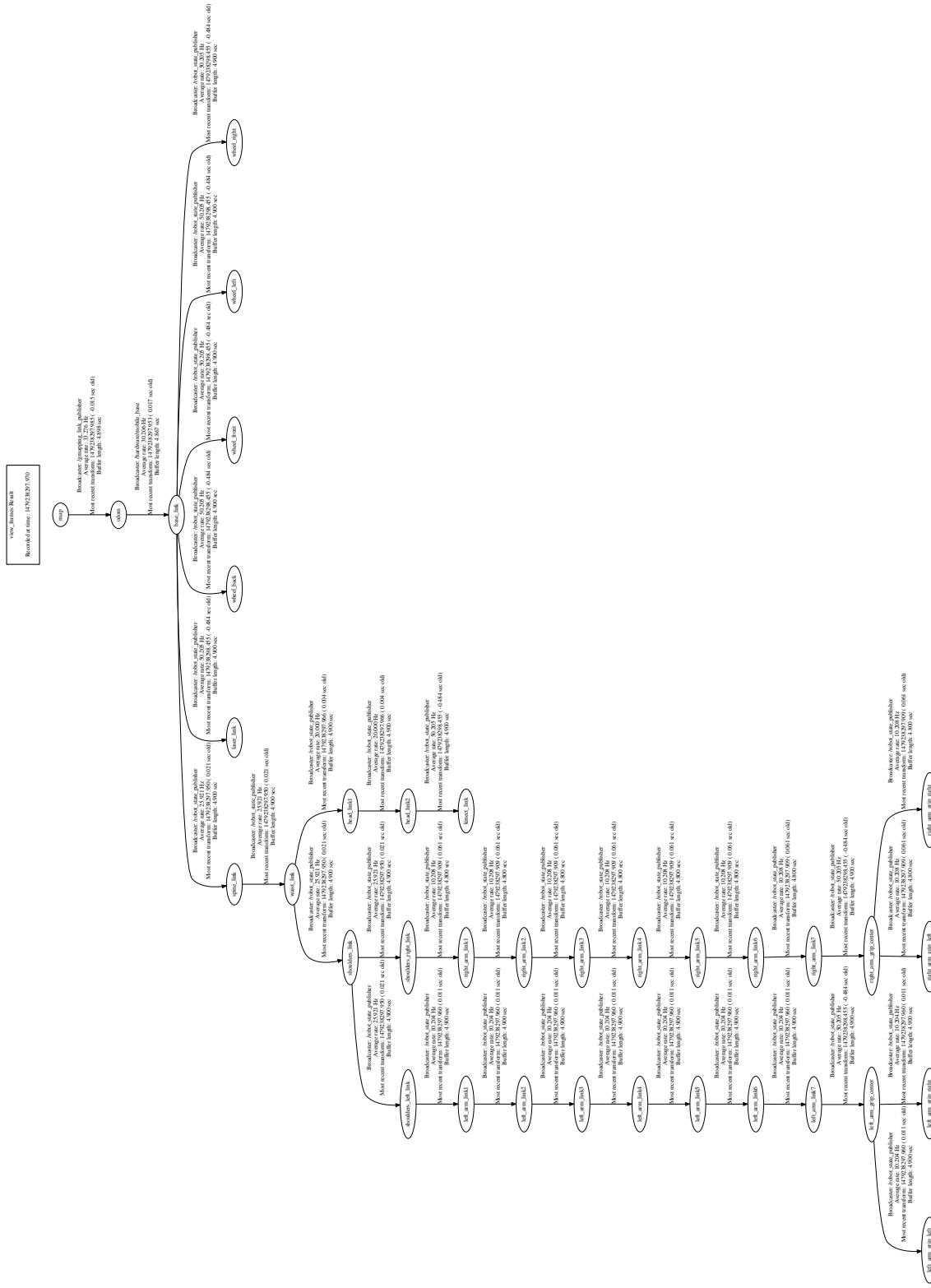


Figura 4.2: Árbol de transformaciones con los marcos de referencia *map* y *odom*

4.4.2. Nodo /hardware/head

Este nodo se encarga de controlar la posición de la cabeza mediante los grados de libertad *pan* y *tilt*. La posición deseada se establece en radianes, en caso de que estos valores se encuentren fuera del rango de alcance de la junta, entonces se posicionará en la cota superior o inferior según sea el caso.

Tópicos suscritos	/hardware/head/goal_pose [std_msgs/Float32MultiArray]	Posición deseada de las juntas de revolución de la cabeza
Tópicos publicados	/hardware/head/current_pose [std_msgs/Float32MultiArray] /joint_states [sensor_msgs/JointState] /hardware/robot_state/head_battery [std_msgs/Float32]	Posición actual de las juntas de la cabeza Descripción del estado de las juntas de la cabeza Voltaje de alimentación de los servo motores de la cabeza

Tabla 4.2: Nodo /hardware/head

Sintaxis en un archivo launch

Para correr este nodo es necesario indicar como argumentos el puerto serial en el cual esta conectado el dispositivo *USB2Dynamixel* asociado a los servo motores de la cabeza, así como el baudaje al cual se establecerá la comunicación.

```
<node name="head" pkg="head" type="head_node.py" output="screen" args="--port
/dev/justinaHead --baud 1000000"/>
```

4.4.3. Nodo /hardware/hokuyo_node

Este nodo se encarga de la adquisición de datos en un sensor Hokuyo Láser y, los hace accesibles mediante un mensaje de tipo *LaserScan*. Los escaneos del Hokuyo se toman en sentido contrario a las agujas del reloj, así mismo, los ángulos se miden en sentido contrario a las agujas del reloj con 0 apuntando directamente hacia delante.

Tópicos publicados	/hardware/scan [sensor_msgs/LaserScan]	[sen- sor_msgs/LaserScan]	Datos de un escaneo
Parámetros	port [string, default: /dev/ttyACM0] frame_id [string, default: laser]	Puerto donde se encuentra el dispositivo Hokuyo Marco de referencia asociado al láser	

Tabla 4.3: Nodo /hardware/hokuyo_node

Sintaxis en un archivo launch

Para lanzar este nodo por medio de un archivo *lauch* es necesario indicar como parámetros el puerto en el que se encuentra el dispositivo y el marco de referencia asociado al láser, dicho marco se encuentra definido en el archivo *justina.xml*.

```
<node name="hokuyo_node" pkg="hokuyo_node" type="hokuyo_node" output="screen">
  <param name="port" type="string" value="/dev/justinaHokuyo" />
  <param name="frame_id" type="string" value="laser_link" />
</node>
```

4.4.4. Nodo /hardware/joy

Este nodo conecta un joystick genérico de Linux a ROS; publica un mensaje de tipo *Joy* que contiene el estado actual de cada uno de los botones y ejes del joystick.

Tópicos publicados	/hardware/joy [sensor_msgs/Joy]	Reporta el estado de los ejes y botones del joystick
Parámetros	dev [string, default: /dev/input/js0]	Dispositivo desde el cual se leen los eventos

Tabla 4.4: Nodo /hardware/joy

Sintaxis en un archivo launch

Para lanzar este nodo por medio de un archivo *lauch* únicamente es necesario indicar el nombre

que se le desea dar al nodo, el paquete en el que se encuentra y el nombre del ejecutable.

```
<node name="joy" pkg="joy" type="joy_node" output="screen"/>
```

4.4.5. Nodo /hardware/left_arm

Este nodo se encarga de controlar la posición de los 7 grados de libertad del brazo izquierdo del robot Justina, de igual modo controla el agarre del *gripper*; la posición deseada para cada uno de los GDL se establece en radianes.

Tópicos publicados	/hardware/left_arm/current_pose [std_msgs/Float32MultiArray]	Posición actual de las juntas del brazo izquierdo
	/hardware/left_arm/current_gripper [std_msgs/Float32]	Posición actual del gripper
	/joint_states [sensor_msgs/JointState]	Descripción del estado de las juntas del brazo izquierdo
	/hardware/robot_state/left_arm_battery [std_msgs/Float32]	Voltaje de alimentación de los servo motores del brazo izquierdo
Tópicos suscritos	/hardware/left_arm/goal_gripper [std_msgs/Float32]	Posición deseada del gripper
	/hardware/left_arm/torque_gripper [std_msgs/Float32]	Par deseado en el gripper para tareas de manipulación de objetos
	/hardware/left_arm/goal_pose [std_msgs/Float32MultiArray]	Posición deseada de las juntas de revolución del brazo izquierdo. Si se especifican 7 datos, éstos serán las posiciones deseadas de los 7 GDL, si son 14 datos, los 7 adicionales serán las rapideces de los servo motores a las que se desea alcanzar dicha posición

Tabla 4.5: Nodo /hardware/left_arm

Sintaxis en un archivo launch

Para correr este nodo es necesario indicar como argumentos el puerto serial en el cual esta conectado el dispositivo *USB2Dynamixel* asociado a los servo motores del brazo izquierdo, además del baudaje al cual se establecerá la comunicación.

```
<node name="left_arm" pkg="arms" type="left_arm_node.py" output="screen" args="--po
/dev/justinaLeftArm --baud1 1000000"/>
```

4.4.6. Nodo /hardware/mobile_base

Este nodo se encarga de controlar el movimiento de la base estableciendo la rapidez de los motores por medio de controladores *RoboClaw*, la velocidad y rapidez lineal están dadas en metros por segundo y, la velocidad angular en radianes por segundo. Aparte de esto, se determina la ubicación del robot en un mapa estático utilizando *tf* y un mensaje de tipo *Odometry*, para obtener información más detallada consulte por favor: <http://wiki.ros.org/navigation/Tutorials/RobotSetup/Odom>.

Tópicos publicados	/hardware/mobile_base/odometr [nav_msgs/Odometry]	Odometría calculada con las lecturas de los encoder de los motores
	/hardware/robot_state/base_batt [std_msgs/Float32]	Voltaje de alimentación de los motores de la base
Tópicos suscritos	/hardware/robot_state/stop [std_msgs/Empty]	Tópico para parar los motores de la base
	/hardware/mobile_base/cmd_vel [geometry_msgs/Twist]	Velocidad lineal deseada en el plano <i>xy</i> , y velocidad angular deseada en <i>z</i>
	/hardware/mobile_base/speeds [std_msgs/Float32MultiArray]	Rapideces instantáneas derecha e izquierda deseadas

Tabla 4.6: Nodo /hardware/mobile_base

Sintaxis en un archivo launch

La base móvil con la que cuenta el robot Justina actualmente posee cuatro motores, es por ello que se requieren dos controladores *RoboClaw*. Para lanzar este nodo se especifica mediante argumentos los dos puertos en los cuales están conectados los controladores.

```
<node name="mobile_base" pkg="mobile_base" type="omni_base_node.py" output="screen"
args="--port1 /dev/justinaRC15 --port2 /dev/justinaRC30"/>
```

4.4.7. Nodo /hardware/torso

Tópicos publicados	/hardware/torso/goal_reached [std_msgs/Bool] /tf [tf/tfMessage] /joint_states [sensor_msgs/JointState] /hardware/torso/current_pose [std_msgs/Float32MultiArray]	
Tópicos suscritos	/hardware/torso/goal_pose [std_msgs/Float32MultiArray] /hardware/torso/goal_rel_pose [std_msgs/Float32MultiArray]	

Tabla 4.7: Nodo /hardware/torso

4.4.8. Nodo /hri/human_follower

Tópicos publicados	/hardware/mobile_base/speeds [std_msgs/Float32MultiArray]	
Tópicos suscritos	/hri/human_following/start_follow [std_msgs/Bool] /hri/leg_finder/leg_poses [geometry_msgs/PointStamped]	

Tabla 4.8: Nodo /hri/human_follower

4.4.9. Nodo /hri/justina_gui

Tópicos publicados	<pre>/hardware/point_cloud_man/save_cloud [std_msgs/String]</pre> <pre>/navigation/path_planning/ simple_move/goal_lateral [std_msgs/Float32]</pre> <pre>/hardware/torso/goal_pose [std_msgs/Float32MultiArray]</pre> <pre>/manipulation/manip_pln/hd_goto_loc [std_msgs/String]</pre> <pre>/hardware/robot_state/stop [std_msgs/Empty]</pre> <pre>/vision/face_recognizer/start_recog_old [std_msgs/Empty]</pre> <pre>/manipulation/manip_pln/ra_goto_loc [std_msgs/String]</pre> <pre>/manipulation/manip_pln/ la_goto_angles [std_msgs/Float32MultiArray]</pre> <pre>/manipulation/manip_pln/ la_pose_wrt_robot [std_msgs/Float32MultiArray]</pre> <pre>/navigation/path_planning/ simple_move/goal_dist [std_msgs/Float32]</pre>
--------------------	---

Tabla 4.9: Nodo /hri/justina_gui

Tópicos publicados	<pre>/vision/face_recognizer/ run_face_recognizer_id [std_msgs/String]</pre> <pre>/hardware/point_cloud_man/ stop_saving_cloud [std_msgs/Empty]</pre> <pre>/hardware/mobile_base/cmd_vel [geometry_msgs/Twist]</pre> <pre>/manipulation/manip_pln/ ra_pose_wrt_robot [std_msgs/Float32MultiArray]</pre> <pre>/navigation/mvn_pln/get_close_xy [std_msgs/Float32MultiArray]</pre> <pre>/hardware/right_arm/goal_torque [std_msgs/Float32MultiArray]</pre> <pre>/manipulation/manip_pln/la_move [std_msgs/String]</pre> <pre>/navigation/mvn_pln/get_close_loc [std_msgs/String]</pre> <pre>/vision/obj_reco/enableRecognizeTopic [std_msgs/Bool]</pre> <pre>/hardware/left_arm/goal_gripper [std_msgs/Float32]</pre>	
--------------------	--	--

Tabla 4.10: Nodo /hri/justina_gui

Tópicos publicados	<pre>/hardware/mobile_base/speeds [std_msgs/Float32MultiArray] /recognizedSpeech [hri_msgs/RecognizedSpeech] /hardware/head/goal_pose [std_msgs/Float32MultiArray] /navigation/path_planning/ simple_move/goal_rel_pose [geometry_msgs/Pose2D] /hri/human_following/start_follow [std_msgs/Bool] /vision/obj_reco/enableDetectWindow [std_msgs/Bool] /hardware/right_arm/torque_gripper [std_msgs/Float32] /vision/face_recognizer/ run_face_trainer_frames [vision_msgs/VisionFaceTrainObject] /hardware/torso/goal_rel_pose [std_msgs/Float32MultiArray] /navigation/obs_avoid/enable [std_msgs/Bool]</pre>	
--------------------	--	--

Tabla 4.11: Nodo /hri/justina_gui

Tópicos publicados	<pre>/vision/thermal_vision/stop_video [std_msgs/Empty]</pre> <pre>/vision/skeleton_finder/stop_recog [std_msgs/Empty]</pre> <pre>/vision/face_recognizer/clearfacesdb [std_msgs/Empty]</pre> <pre>/manipulation/manip_pln/ hd_goto_angles [std_msgs/Float32MultiArray]</pre> <pre>/hardware/left_arm/goal_torque [std_msgs/Float32MultiArray]</pre> <pre>/vision/face_recognizer/clearfacesdbbyid [std_msgs/String]</pre> <pre>/hardware/left_arm/torque_gripper [std_msgs/Float32]</pre> <pre>/navigation/path_planning/ simple_move/goal_dist_angle [std_msgs/Float32MultiArray]</pre> <pre>/manipulation/manip_pln/ la_pose_wrt_arm [std_msgs/Float32MultiArray]</pre> <pre>/hardware/left_arm/goal_pose [std_msgs/Float32MultiArray]</pre>
--------------------	--

Tabla 4.12: Nodo /hri/justina_gui

Tópicos publicados	<pre>/navigation/mvn_pln/add_location [navig_msgs/Location]</pre> <pre>/hri/leg_finder/enable [std_msgs/Bool]</pre> <pre>/vision/face_recognizer/ run_face_recognizer [std_msgs/Empty]</pre> <pre>/hri/sp_rec/recognized [std_msgs/String]</pre> <pre>/vision/thermal_vision/start_video [std_msgs/Empty]</pre> <pre>/vision/face_recognizer/run_face_trainer [std_msgs/String]</pre> <pre>/manipulation/manip_pln/la_goto_loc [std_msgs/String]</pre> <pre>/vision/face_recognizer/stop_recog [std_msgs/Empty]</pre> <pre>/hardware/right_arm/goal_pose [std_msgs/Float32MultiArray]</pre> <pre>/vision/face_recognizer/start_recog [std_msgs/Empty]</pre>
--------------------	--

Tabla 4.13: Nodo /hri/justina_gui

Tópicos publicados	<pre>/navigation/path_planning/simple_move /goal_path [nav_msgs/Path] /vision/skeleton_finder/start_recognition [std_msgs/Empty] /navigation/path_planning/simple_move /goal_pose [geometry_msgs/Pose2D] /vision/qr/start_qr [std_msgs/Bool] /manipulation/manip_pln /ra_goto_angles [std_msgs/Float32MultiArray] /hardware/right_arm/goal_gripper [std_msgs/Float32] /manipulation/manip_pln /ra_pose_wrt_arm [std_msgs/Float32MultiArray]</pre>
--------------------	---

Tabla 4.14: Nodo /hri/justina_gui

Tópicos suscritos	<pre>/hardware/left_arm/current_pose [std_msgs/Float32MultiArray] /hardware/robot_state/stop [std_msgs/Empty] /vision/face_recognizer/faces [vision_msgs/VisionFaceObjects] /recognizedSpeech [hri_msgs/RecognizedSpeech] /hardware/left_arm/current_gripper [std_msgs/Float32] /hri/leg_finder/legs_found [std_msgs/Empty] /hardware/right_arm/current_gripper [] /navigation/localization/current_pose [geometry_msgs/ PoseWithCovarianceStamped] /hardware/head/current_pose [std_msgs/Float32MultiArray] /hardware/torso/goal_reached [std_msgs/Bool]</pre>	
-------------------	--	--

Tabla 4.15: Nodo /hri/justina_gui

Tópicos suscritos	<pre>/tf [tf/tfMessage] /navigation/obs_avoid/obs_in_front [std_msgs/Bool] /tf_static [tf2_msgs/TFMessage] /manipulation/hd_goal_reached [std_msgs/Bool] /hri/sp_rec/recognized [std_msgs/String] /hardware/robot_state/ right_arm_battery [] /hardware/right_arm/current_pose [] /hardware/torso/current_pose [std_msgs/Float32MultiArray] /manipulation/ra_goal_reached [std_msgs/Bool] /navigation/global_goal_reached [std_msgs/Bool]</pre>	
-------------------	---	--

Tabla 4.16: Nodo /hri/justina_gui

Tópicos suscritos	/navigation/obs_avoid/collision_risk [std_msgs/Bool] /navigation/goal_reached [std_msgs/Bool] /vision/face_recognizer/trainer_result [std_msgs/Int32] /hardware/robot_state/left_arm_battery [std_msgs/Float32] /hardware/robot_state/head_battery [std_msgs/Float32] /hri/qr/recognized [std_msgs/String] /manipulation/la_goal_reached [std_msgs/Bool] /hardware/robot_state/base_battery [std_msgs/Float32]
-------------------	---

Tabla 4.17: Nodo /hri/justina_gui

4.4.10. Nodo /hri/leg_finder

Tópicos publicados	/hri/leg_finder/legs_found [std_msgs/Empty] /hri/leg_finder/leg_poses [geometry_msgs/PointStamped]	
Tópicos suscritos	/hardware/scan [sensor_msgs/LaserScan] /tf [tf/tfMessage] /tf_static [tf2_msgs/TFMessage] /hri/leg_finder/enable [std_msgs/Bool]	

Tabla 4.18: Nodo /hri/leg_finder

4.4.11. Nodo /hri/qr_reader

Tópicos publicados	/hri/qr/recognized [std_msgs/String]	
Tópicos suscritos	/tf [tf/tfMessage] /tf_static [tf2_msgs/TFMessage] /vision/qr/start_qr [std_msgs/Bool]	

Tabla 4.19: Nodo /hri/qr_reader

4.4.12. Nodo /hri/rviz

Tópicos publicados	<pre>/clicked_point [geometry_msgs/PointStamped] /move_base_simple/goal [geometry_msgs/PoseStamped] /initialpose [geometry_msgs/ PoseWithCovarianceStamped]</pre>	
Tópicos suscritos	<pre>/hri/rviz/location_markers [visualization_msgs/Marker] /hardware/scan [sensor_msgs/LaserScan] /hri/rviz/location_markers_array [] /tf [tf/tfMessage] /tf_static [tf2_msgs/TFMessage] /hri/leg_finder/leg_poses [geometry_msgs/PointStamped] /navigation/localization/map_updates [] /navigation/mvn_pln/last_calc_path [nav_msgs/Path] /navigation/localization/map [nav_msgs/OccupancyGrid]</pre>	

Tabla 4.20: Nodo /hri/rviz

4.4.13. Nodo /hri/sp_gen

Tópicos suscritos	/hri/sp_gen/say []	
-------------------	--------------------	--

Tabla 4.21: Nodo /hri/sp_gen

4.4.14. Nodo /interoperation/joystick_teleop

Este nodo se suscribe a un mensaje de tipo *Joy* para controlar mediante un joystick de una consola Xbox el movimiento de la base, la posición de la cabeza y del torso. Las posiciones angulares y lineales, así como las velocidades lineales y angulares están dadas en radianes, metros, metros por segundo y radianes por segundo respectivamente.

Para mover la cabeza del robot se utiliza el *stick* izquierdo, la base se opera por medio del *stick* derecho, mientras que el botón rojo del joystick es para el paro de la base.

Tópicos publicados	/hardware/robot_state/stop [std_msgs/Empty]	Tópico de paro para los motores de la base
	/hardware/mobile_base/cmd_vel [geometry_msgs/Twist]	Velocidad lineal deseada deseada de la base en el plano xy , y velocidad angular deseada en z
	/hardware/head/goal_pose [std_msgs/Float32MultiArray]	Posición deseada de la cabeza
	/hardware/torso/goal_spine [std_msgs/Float32]	Posición deseada de la junta prismática para cambiar la altura del torso
	/hardware/torso/goal_shoulders [std_msgs/Float32]	Posición deseada de la junta de revolución para orientar el torso (roll)
	/hardware/torso/goal_waist [std_msgs/Float32]	Posición deseada de la junta de revolución para orientar el torso (yaw)
Tópicos suscritos	/hardware/joy [sensor_msgs/Joy]	Estado de los ejes y botones de un joystick

Tabla 4.22: Nodo /interoperation/joystick_teleop

Sintaxis en un archivo launch

Para lanzar este nodo por medio de un archivo *lauch* sólo se necesita indicar el nombre que se le desea dar al nodo, el paquete dentro del que se encuentra y el nombre del ejecutable.

```
<node name="joystick_teleop" pkg="joystick_teleop" type="joystick_teleop_node.py"
      output="screen" />
```

4.4.15. Nodo /manipulation/ik_geometric

Servicios	/manipulation/ik_geometric/ ik_float_array /manipulation/ik_geometric/ik_path /manipulation/ik_geometric/ik_pose /manipulation/ik_geometric/ direct_kinematics
-----------	---

Tabla 4.23: Nodo /manipulation/ik_geometric

4.4.16. Nodo /manipulation/manip_pln

Tópicos publicados	<p>/hardware/right_arm/goal_torque [std_msgs/Float32MultiArray]</p> <p>/hardware/head/goal_pose [std_msgs/Float32MultiArray]</p> <p>/hardware/left_arm/goal_torque [std_msgs/Float32MultiArray]</p> <p>/hardware/left_arm/goal_pose [std_msgs/Float32MultiArray]</p> <p>/manipulation/hd_goal_reached [std_msgs/Bool]</p> <p>/manipulation/ra_goal_reached [std_msgs/Bool]</p> <p>/hardware/right_arm/goal_pose [std_msgs/Float32MultiArray]</p> <p>/hardware/head/goal_torque [std_msgs/Float32MultiArray]</p> <p>/manipulation/la_goal_reached [std_msgs/Bool]</p>	
--------------------	--	--

Tabla 4.24: Nodo /manipulation/manip_pln

Tópicos suscritos	<pre>/hardware/left_arm/current_pose [std_msgs/Float32MultiArray]</pre> <pre>/manipulation/manip_pln/ra_goto_loc [std_msgs/String]</pre> <pre>/manipulation/manip_pln/hd_goto_loc [std_msgs/String]</pre> <pre>/manipulation/manip_pln/ la_goto_angles [std_msgs/Float32MultiArray]</pre> <pre>/manipulation/manip_pln/ la_pose_wrt_robot [std_msgs/Float32MultiArray]</pre> <pre>/manipulation/manip_pln/ ra_pose_wrt_robot [std_msgs/Float32MultiArray]</pre> <pre>/manipulation/manip_pln/la_move [std_msgs/String]</pre> <pre>/hardware/head/current_pose [std_msgs/Float32MultiArray]</pre> <pre>/manipulation/manip_pln/hd_move []</pre> <pre>/manipulation/manip_pln/ra_move []</pre>
-------------------	---

Tabla 4.25: Nodo /manipulation/manip_pln

Tópicos suscritos	<pre>/manipulation/manip_pln/ hd_goto_angles [std_msgs/Float32MultiArray] /tf [tf/tfMessage] /manipulation/manip_pln/ la_pose_wrt_arm [std_msgs/Float32MultiArray] /tf_static [tf2_msgs/TFMessage] /hardware/right_arm/current_pose [] /manipulation/manip_pln/la_goto_loc [std_msgs/String] /manipulation/manip_pln/ ra_pose_wrt_arm [std_msgs/Float32MultiArray] /manipulation/manip_pln/ ra_goto_angles [std_msgs/Float32MultiArray]</pre>	
-------------------	--	--

Tabla 4.26: Nodo /manipulation/manip_pln

4.4.17. Nodo /navigation/localization/loc_amcl

Este nodo implementa el enfoque adaptativo de localización de Monte Carlo, que utiliza un filtro de partículas para rastrear la pose de un robot en un mapa conocido. AMCL es un sistema de localización probabilística para un robot que se mueve en un plano.

AMCL transforma los escaneos láser entrantes al sistema de referencia *odometry*. Por lo tanto, debe existir un camino a través del árbol *tf* desde el sistema de referencia en el que los escaneos láser se publican hacia el sistema de referencia de odometría.

Durante la operación AMCL estima la transformación del marco de referencia de la base con respecto al marco de referencia global(*map* para este caso), pero solamente publica la transformación entre el marco de referencia global(*map*) y el marco de referencia de odometría(*odometry*). Esencialmente, esta transformación considera la deriva que ocurre usando

Dead Reckoning. *Dead Reckoning* es el proceso de calcular la posición estimando la dirección y la distancia recorrida.

Para obtener información más detallada consulte: <http://wiki.ros.org/amcl>.

Tópicos publicados	/navigation/localization/current_pose [geometry_msgs/ PoseWithCovarianceStamped]	Posición estimada del robot en el mapa, con covarianza
	/tf [tf/tfMessage]	Publica la transformación de odom (que se puede reasignar a través del parámetro <i>odom_frame_id</i>) a map
	/navigation/localization/particlecloud [geometry_msgs/PoseArray]	Conjunto de poses estimadas mantenidas por el filtro
Servicios	/navigation/localization/ global_localization [std_srvs/Empty]	Inicio de la localización global, donde todas las partículas se dispersan al azar a través del espacio libre en el mapa

Tabla 4.27: Nodo /navigation/localization/loc_amcl

Tópicos suscritos	/navigation/localization/initialpose [geometry_msgs/ PoseWithCovarianceStamped] /hardware/scan [sensor_msgs/LaserScan] /tf [tf/tfMessage]	Media y covarianza con la cual se (re-)inicializa el filtro de partículas Escaneos láser Transformaciones del robot
Parámetros	update_min_a [double, default: $\pi/6.0$ radians] laser_min_range [double, default: -1.0] odom_model_type [string, default: "diff"]	Movimiento de rotación requerido antes de realizar una actualización del filtro Rango de escaneo mínimo a considerar Configuración del robot, ya sea "diff", "omni", "diff-corrected" o "omni-corrected"

Tabla 4.28: Nodo /navigation/localization/loc_amcl

Sintaxis en un archivo launch

Para correr este nodo se necesita indicar el tópico en el cual se publican los datos del láser (*/hardware/scan* para este caso), de igual forma se requiere modificar los parámetros *update_min_a*, *laser_min_range* y *odom_model_type*.

```
<node name="loc_amcl" pkg="amcl" type="amcl" output="acreen" args="scan:=/hardware/scan">
  <param name="update_min_a" value="0.3"/>
  <param name="laser_min_range" value="0.3"/>
  <param name="odom_model_type" value="omni"/>
</node>
```

4.4.18. Nodo /navigation/localization/map_server

Este nodo ofrece datos de un mapa como un servicio de ROS. También proporciona la utilidad de línea de comandos *map_saver*, que permite guardar en un archivo los mapas generados

dinámicamente.

Los mapas manipulados por las herramientas de este paquete se almacenan en un par de archivos, un archivo YAML describe los metadatos del mapa y una imagen codifica los datos de ocupación. La imagen describe el estado de ocupación de cada celda del mundo en el color del píxel correspondiente. Los píxeles más blancos están libres, los píxeles más negros están ocupados y los píxeles entre estos colores son desconocidos. Los campos requeridos en el archivo YAML son seis: *image*, *resolution*, *origin*, *occupied_thresh*, *free_thresh* y *negate*.

Para obtener información más específica por favor consulte: http://wiki.ros.org/map_server.

Tópicos publicados	/navigation/localization/map_metadata [nav_msgs/MapMetaData] /navigation/localization/map [nav_msgs/OccupancyGrid]	Metadatos del mapa Mapa
Servicios	navigation/localization/static_map [nav_msgs/GetMap]	Obtención del mapa a través de este servicio
Parámetros	frame_id [string, default: "map"]	Marco de referencia establecido en el encabezado (<i>header</i>) del mapa publicado

Tabla 4.29: Nodo /navigation/localization/map_server

Sintaxis en un archivo launch

Para ejecutar este nodo se requiere especificar como argumento el archivo YAML que contiene los metadatos del mapa que se quiere proveer. Para este ejemplo, el archivo *bioroboanexo3.yaml* se encuentra dentro del paquete *knowledge* en la ruta *navigation/occupancy_grids*.

```
<node name="map_server" pkg="map_server" type="map_server" output="screen"
      args="$(find knowledge)/navigation/occupancy_grids/bioroboanexo3.yaml"/>$
```

4.4.19. Nodo /navigation/mvn_pln

Tópicos publicados	<pre>/navigation/path_planning/simple_move /goal_lateral [std_msgs/Float32] /hardware/torso/goal_pose [std_msgs/Float32MultiArray] /manipulation/manip_pln/hd_goto_loc [std_msgs/String] /manipulation/manip_pln/ra_goto_loc [std_msgs/String] /hri/rviz/location_markers [visualization_msgs/Marker] /manipulation/manip_pln/ la_goto_angles [std_msgs/Float32MultiArray] /manipulation/manip_pln/ la_pose_wrt_robot [std_msgs/Float32MultiArray] /navigation/path_planning/simple_move /goal_dist [std_msgs/Float32] /manipulation/manip_pln/ ra_pose_wrt_robot [std_msgs/Float32MultiArray]</pre>
--------------------	--

Tabla 4.30: Nodo /navigation/mvn_pln

Tópicos publicados	<pre>/navigation/mvn_pln/get_close_xy [std_msgs/Float32MultiArray]</pre> <pre>/manipulation/manip_pln/la_move [std_msgs/String]</pre> <pre>/navigation/mvn_pln/get_close_loc [std_msgs/String]</pre> <pre>/hardware/left_arm/goal_gripper [std_msgs/Float32]</pre> <pre>/navigation/path_planning/simple_move /goal_rel_pose [geometry_msgs/Pose2D]</pre> <pre>/hardware/right_arm/torque_gripper [std_msgs/Float32]</pre> <pre>/hardware/torso/goal_rel_pose [std_msgs/Float32MultiArray]</pre> <pre>/navigation/obs_avoid/enable [std_msgs/Bool]</pre> <pre>/manipulation/manip_pln/ hd_goto_angles [std_msgs/Float32MultiArray]</pre>
--------------------	--

Tabla 4.31: Nodo /navigation/mvn_pln

Tópicos publicados	<pre>/hardware/left_arm/torque_gripper [std_msgs/Float32] /navigation/path_planning/simple_move /goal_dist_angle [std_msgs/Float32MultiArray] /manipulation/manip_pln/ la_pose_wrt_arm [std_msgs/Float32MultiArray] /navigation/mvn_pln/add_location [navig_msgs/Location] /navigation/mvn_pln/last_calc_path [nav_msgs/Path] /navigation/global_goal_reached [std_msgs/Bool] /manipulation/manip_pln/la_goto_loc [std_msgs/String] /navigation/path_planning/simple_move /goal_path [nav_msgs/Path] /navigation/path_planning/simple_move /goal_pose [geometry_msgs/Pose2D]</pre>
--------------------	--

Tabla 4.32: Nodo /navigation/mvn_pln

Tópicos publicados	/manipulation/manip_pln/ ra_goto_angles [std_msgs/Float32MultiArray] /hardware/right_arm/goal_gripper [std_msgs/Float32] /manipulation/manip_pln/ ra_pose_wrt_arm [std_msgs/Float32MultiArray]	
Servicios	/navigation/mvn_pln/plan_path	

Tabla 4.33: Nodo /navigation/mvn_pln

Tópicos suscritos	<pre>/hardware/robot_state/stop [std_msgs/Empty] /navigation/mvn_pln/get_close_xy [std_msgs/Float32MultiArray] (clicked_point [geometry_msgs/PointStamped] /navigation/mvn_pln/get_close_loc [std_msgs/String] /navigation/localization/current_pose [geometry_msgs/ PoseWithCovarianceStamped] /hardware/scan [sensor_msgs/LaserScan] /hardware/torso/goal_reached [std_msgs/Bool] /tf [tf/tfMessage] /navigation/obs_avoid/obs_in_front [std_msgs/Bool]</pre>	
-------------------	---	--

Tabla 4.34: Nodo /navigation/mvn_pln

Tópicos suscritos	/tf_static [tf2_msgs/TFMessage] /manipulation/hd_goal_reached [std_msgs/Bool] /navigation/mvn_pln/add_location [navig_msgs/Location] /manipulation/ra_goal_reached [std_msgs/Bool] /navigation/global_goal_reached [std_msgs/Bool] /navigation/obs_avoid/collision_risk [std_msgs/Bool] /navigation/goal_reached [std_msgs/Bool] /navigation/obs_avoid/collision_point [geometry_msgs/PointStamped] /manipulation/la_goal_reached [std_msgs/Bool]
-------------------	---

Tabla 4.35: Nodo /navigation/mvn_pln

4.4.20. Nodo /navigation/obs_avoid/obstacle_detector

Tópicos publicados	/navigation/obs_avoid/obs_in_front [std_msgs/Bool] /navigation/obs_avoid/collision_risk [std_msgs/Bool] /navigation/obs_avoid/collision_point [geometry_msgs/PointStamped]	
Tópicos suscritos	/hardware/scan [sensor_msgs/LaserScan] /navigation/obs_avoid/enable [std_msgs/Bool] /tf [tf/tfMessage] /tf_static [tf2_msgs/TFMessage] /navigation/mvn_pln/last_calc_path [nav_msgs/Path]	

Tabla 4.36: Nodo /navigation/obs_avoid/obstacle_detector

4.4.21. Nodo /navigation/path_planning/path_calculator

Este nodo se encarga de calcular una ruta y suavizarla desde una pose inicial hasta una pose objetivo utilizando el algoritmo de búsqueda A*, ésto mediante dos servicios de ROS.

Servicios	/navigation/path_planning/ path_calculator/wave_front_from_map [navig_msgs/PathFromMap]	
	/navigation/path_planning/ path_calculator/a_star_from_map [navig_msgs/PathFromMap]	Cálculo de una ruta utilizando el algoritmo de búsqueda A*

Tabla 4.37: Nodo /navigation/path_planning/path_calculator

Sintaxis en un archivo launch

Para correr este nodo sólo se requiere especificar el nombre que se le desea dar al nodo, el paquete en el que se encuentra y el nombre del ejecutable.

```
<node name="path_calculator" pkg="path_calculator" type="path_calculator_node"
output="screen"/>
```

4.4.22. Nodo /navigation/path_planning/simple_move

Tópicos publicados	<p>/hardware/mobile_base/cmd_vel [geometry_msgs/Twist]</p> <p>/hardware/mobile_base/speeds [std_msgs/Float32MultiArray]</p> <p>/hardware/head/goal_pose [std_msgs/Float32MultiArray]</p> <p>/navigation/goal_reached [std_msgs/Bool]</p>	
Tópicos suscritos	<p>/hardware/robot_state/stop [std_msgs/Empty]</p> <p>/navigation/path_planning/simple_move /goal_lateral [std_msgs/Float32]</p> <p>/navigation/path_planning/simple_move /goal_dist [std_msgs/Float32]</p> <p>/navigation/path_planning/simple_move /goal_rel_pose [geometry_msgs/Pose2D]</p>	

Tabla 4.38: Nodo /navigation/path_planning/simple_move

Tópicos suscritos	<pre>/navigation/localization/current_pose [geometry_msgs/ PoseWithCovarianceStamped] /tf [tf/tfMessage] /navigation/path_planning/simple_move /goal_dist_angle [std_msgs/Float32MultiArray] /tf_static [tf2_msgs/TFMessage] /navigation/obs_avoid/collision_risk [std_msgs/Bool] /navigation/path_planning/simple_move /goal_path [nav_msgs/Path] /navigation/path_planning/simple_move /goal_pose [geo- metry_msgs/Pose2D]</pre>
-------------------	--

Tabla 4.39: Nodo /navigation/path_planning/simple_move

4.4.23. Nodo /vision/face_recog

Tópicos publicados	/vision/face_recognizer/faces [vision_msgs/VisionFaceObjects] /vision/face_recognizer/trainer_result [std_msgs/Int32]	
Tópicos suscritos	/vision/face_recognizer/start_recog_old [std_msgs/Empty] /vision/face_recognizer/ run_face_recognizer_id [std_msgs/String] /vision/face_recognizer/ run_face_trainer_frames [vision_msgs/VisionFaceTrainObject] /vision/face_recognizer/clearfacesdb [std_msgs/Empty] /vision/face_recognizer/clearfacesdbbyid [std_msgs/String]	

Tabla 4.40: Nodo /vision/face_recog

Tópicos suscritos	<pre>/vision/face_recognizer/ run_face_recognizer [std_msgs/Empty]</pre> <pre>/vision/face_recognizer/run_face_trainer [std_msgs/String]</pre> <pre>/vision/face_recognizer/stop_recog [std_msgs/Empty]</pre> <pre>/vision/face_recognizer/start_recog [std_msgs/Empty]</pre>	
-------------------	--	--

Tabla 4.41: Nodo /vision/face_recog

4.4.24. Nodo /vision/line_finder

Tópicos suscritos	<pre>/hardware/head/current_pose [std_msgs/Float32MultiArray]</pre>	
Servicios	<pre>/vision/line_finder/find_lines_transac</pre>	

Tabla 4.42: Nodo /vision/line_finder

4.4.25. Nodo /vision/obj_reco

Tópicos publicados	/vision/obj_reco/recognizedObjects [vision_msgs/VisionObjectList]	
Tópicos suscritos	/vision/obj_reco/enableRecognizeTopic [std_msgs/Bool] /vision/obj_reco/enableDetectWindow [std_msgs/Bool] /hardware/point_cloud_man/ rgbd_wrt_robot []	
Servicios	/vision/obj_reco/det_objs /vision/geometry_finder/findPlane /vision/obj_reco/trainObject	

Tabla 4.43: Nodo /vision/obj_reco

4.4.26. Nodo /vision/skeleton_finder

Tópicos publicados	/vision/skeleton_finder/skeletons [vision_msgs/Skeletons]	
Tópicos suscritos	/vision/skeleton_finder/start_recog [std_msgs/Empty] /vision/skeleton_finder/stop_recog [std_msgs/Empty]	

Tabla 4.44: Nodo /vision/skeleton_finder

APÉNDICE A

Hardware

En la siguiente sección mostramos el hardware utilizado para el desarrollo del robot Justina así como especificaciones del mismo y algunas configuraciones que deben seguirse para su correcto funcionamiento.

A.1. Actuadores y sus controladores

En ésta sección se mostrarán los componentes utilizados para ensamblar al robot Justina y especificaciones técnicas como algunas configuraciones y recomendaciones del mismo para su correcto funcionamiento.

A.1.1. Servomotor MX-106



Figura A.1: MX-106

MX-106			
Voltaje de operación	14.8[V]	12[V]	11.1[V]
	102[kg*cm]	85.6[Kg*cm]	81.5[kg*cm]
Torque	10.0[N*m]	8.4[N*m]	8[N*m]
Velocidad sin carga	55[RPM]	45[RPM]	41[RPM]
Masa	153[g]		
Medidas	40.2[mm]x65.1[mm]x46[mm]		
Resolución	0.088[grados]		
Radio de redicción	1/225		
Ángulo de operación	360 grados o giro continuo		
Corriente máxima	5.2[A] @ 12[V]		
Corriente en espera	55[mA]		
Temperatura de operación	-5[C] 85[C]		
Protocolo	TTL Asynchronous serial		
Límite de modulos	254 direcciones validas		
Velocidad	8000bps 3Mbps		
Realimentación de posición	Sí		
Realimentación de temperatura	Sí		
Realimentación de voltaje de carga	Sí		
Realimentación de voltaje de entrada	Sí		
PID	Sí		
Materiales	Engranes de metal y cuerpo plastico		
Lista de controladores	USB2Dynamixel		
	CM-530		
	CM-700		
	Arbotix		

Tabla A.1: MX-106

A.1.2. Servomotor MX-64



Figura A.2: MX-64

MX-64			
Voltaje de operación	14.8[V]	12[V]	11.1[V]
Torque	74[kg*cm]	61[Kg*cm]	56[kg*cm]
Velocidad sin carga	7.3[N*m]	6[N*m]	5.5[N*m]
Masa	78[RPM]	63[RPM]	58[RPM]
Medidas	126[g]		
Resolución	40.2[mm]x61.1[mm]x41[mm]		
Radio de redicción	0.088[grados]		
Ángulo de operación	1/200		
Corriente máxima	360 grados o giro continuo		
Corriente en espera	4.1[A] @ 12[V]		
Temperatura de operación	100[mA]		
Protocolo	-5[C] 85[C]		
Límite de modulos	TTL Asynchronous serial		
Velocidad de transmisión	254 direcciones validas		
Realimentación de posición	8000bps 3Mbps		
Realimentación de temperatura	Sí		
Realimentación de voltaje de carga	Sí		
Realimentación de voltaje de entrada	Sí		
PID	Sí		
Materiales	Engranes de metal y cuerpo plastico		
Lista de controladores	USB2Dynamixel		
	CM-530		
	CM-700		
	Arbotix		

Tabla A.2: MX-64

A.1.3. Servomotor MX-28



Figura A.3: MX-28

MX-28			
Voltaje de operación	14.8[V]	12[V]	11.1[V]
	31[kg*cm]	25.5[Kg*cm]	23.4[kg*cm]
Torque	3.1[N*m]	2.5[N*m]	2.3[N*m]
Velocidad sin carga	67[RPM]	55[RPM]	50[RPM]
Masa	72[g]		
Medidas	35.6[mm]x50.6[mm]x35.5[mm]		
Resolución	0.088[grados]		
Radio de redicción	193:1		
Ángulo de operación	360 grados o giro continuo		
Corriente máxima	1.4[A] @ 12[V]		
Corriente en espera	100[mA]		
Temperatura de operación	-5[C] 80[C]		
Protocolo	TTL Asynchronous serial		
Límite de modulos	254 direcciones validas		
Velocidad de transmisión	8000bps 3Mbps		
Realimentación de posición	Sí		
Realimentación de temperatura	Sí		
Realimentación de voltaje de carga	Sí		
Realimentación de voltaje de entrada	Sí		
PID	Sí		
Materiales	Engranes de metal y cuerpo plastico		
Lista de controladores	USB2Dynamixel		
	CM-530		
	CM-700		
	Open CM 9		

Tabla A.3: MX-28

A.1.4. Motor-DCX32L GB KL 12V

Existen 2 tipos de motores DCX32L, el GPX32 LN 16:1 y el GPX32 G1 35:1 los cuales tienen cambios en sus funciones pero escencialmente conservan el diseño.



Figura A.4: Motor DCX32L

GPX32 G1 35:1	
Funciones	
Gearhead type	Versión estándar
Reducción	35:1
Número de etapas	2
Comutación	Graphote brushes
Fuente de voltaje	Voltaje nominal 12[V]
Motor bearings	preloaded ball bearing
Conteos por vuelta	1024
Hysteresis	0.17m
Forma y ajuste	
Gear shaft	With flat
Shaft bore	Without transverse bore
Shaft length L1	21[mm]
Length of flat L2	12[mm]
Height of flat D2	7[mm]
Gear flange	Standard flange
Amount of threads	4
Thread diameter	M3
Pitch circle diameter TK	26[mm]
Conexión eléctrica, motor	cable
Tipo de conector, motor	Sin conector
Longitud del cable L1 para el motor	200[mm]
Tipo de cable	AWG18
Conexión electrica, enconder	Estándar
Longitud del cable L1 para el encoder	200[mm]
Tipo de cable para el enconder	TPE ribbon cable
Tipo de conector, encoder	10-pol 2.54[mm] pin
Orientación de la conexión (motor)	0 grados
Orientación de la conexión (enconder)	0 grados
Your entries	
Voltaje disponible	12[V]
Velocidad	180[rpm]
Torque	2000[mNm]
Valores de el dispositivo con voltaje	
Máx. speed at given load	190[rpm]
Máximo torque continuo	2440.62[mNm]
Máxima corriente continua	6[A]

Tabla A.4: GPX32 G1 35:1

GPX32 G1 16:1	
Funciones	
Gearhead type	Nivel de ruido reducido
Reducción	16:1
Número de etapas	2
Commutación	Graphote brushes
Fuente de voltaje	Voltaje nominal 12[V]
Motor bearings	preloaded ball bearing
Conteos por vuelta	1024
Hysteresis	0.17m
Forma y ajuste	
Gear shaft	With flat
Shaft bore	Without transverse bore
Shaft length L1	21[mm]
Length of flat L2	12[mm]
Height of flat D2	7[mm]
Gear flange	Standard flange
Amount of threads	4
Thread diameter	M3
Pitch circle diameter TK	26[mm]
Conexión eléctrica, motor	Terminal (bent radially)
Conexión electrica, enconder	Estándar
Longitud del cable L1 para el encoder	200[mm]
Tipo de cable para el enconder	TPE ribbon cable
Tipo de conector, encoder	10-pol 2.54[mm] pin
Orientación de la conexión (motor)	0 grados
Orientación de la conexión (enconder)	0 grados
Your entries	
Voltaje disponible	12[V]
Velocidad	400[rpm]
Torque	900[mNm]
Valores de el dispositivo con voltaje	
Máx. speed at given load	417[rpm]
Máximo torque continuo	1115.71[mNm]
Máxima corriente continua	6[A]

Tabla A.5: Motor DCX32L

Motor - DCX32L GB KL 12V	
Valores en voltaje nominal	
Voltaje nominal	12[V]
Velocidad sin carga	7120[rpm]
Corriente sin carga	274[mA]
Velocidad nominal	6560[rpm]
Torque nominal (máx. torque continuo)	89.4[mNm]
Corriente nominal	6[A]
Stall Torque	1730[mNm]
Stall Corriente	111[A]
Eficiencia máxima	85.5 %
Características	
Máxima salida de potencia	90.2[W]
Resistencia de terminal	0.108[Ohm]
Inductancia de terminal	0.03362[mH]
Torque constante	15.6[mNm/A]
Velocidad constante	612[rpm/V]
Gradiente de velocidad/torque	4.24[rpm/mNm]
Mechanical time constant	3.44[ms]
Inercia del rotor	77.6[gcm ²]
Datos térmicos	
Resistencia térmica housing-ambient	7.28[K/W]
Resistencia térmica winding-housing	2.3[K/W]
Thermal time constant Of the winding	45[s]
Constante de tiempo térmica del motor	837[s]
Temperatura ambiente	-40 a 100[Grados C]
Max. winding temperatura	155[grados C]
Datos mecánicos	
Velocidad máxima permisible	11300[rpm]
Min. axial play	0[mm]
Máx. axial play	0.1[mm]
Radial backlash	0.02[mm]
Max. axial load (dynamic)	7[N]
Max. force for press fits	22.6[N]
Max. radial load	65.3[N]
Especificaciones	
Número de pares de polos	1
Número de segmentos del conmutador	11
Peso	0[mm]
Nivel de ruido típico	47dbA

Tabla A.6: Motor DCX32L

A.1.5. USB2Dynamixel adapter



Figura A.5: Adaptador USBDynamixel

Para controlar una red de Robotics Dynamixels desde el puerto USB de la computadora. El adaptador USB2Dynamixel tiene tres opciones de salida:

-Nivel TTL RS232: conector de 3 pines, usado con un Dynamixel serie AX y MX-T

- AX-12A
- AX-18A
- AX-12W
- MX-28T
- MX-64T
- MX-106T

-S485: conector de 4 pines, usado con RX, EX y MX-R de la serie Dynamixel

- RX-24F
- RX-28
- RX-64
- RX-28R
- MX-64R
- MX-106R
- EX-106

A.1.6. Roboclaw 2x30A

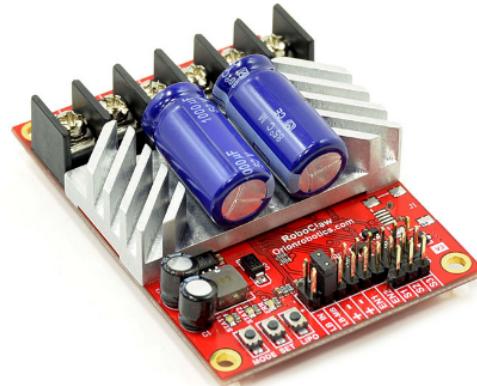


Figura A.6: Roboclaw 2x30A

Roboclaw 2x30A	
Canales para motor	2
Voltaje de operacion	6[V] a 34[V]
corriente continua de salida	20[A]
pico de corriente de salida	60[A]
5V BEC(1) corriente máxima	3[A]
Ancho	5.2[cm]
Largo	7.4[cm]
Peso	63[g]

Tabla A.7: Roboclaw 2x30A

A.1.7. Roboclaw 2x15A



Figura A.7: Roboclaw 2x15A

Roboclaw 2x15A	
Canales para motor	2
Voltaje de operacion	6[V] a 34[V]
corriente continua de salida	15[A]
pico de corriente de salida	30[A]
5V BEC(1) corriente máxima	3[A]
Ancho	5.2[cm]
Largo	7.4[cm]
Peso	54[g]

Tabla A.8: Roboclaw 2x15A

A.2. Vision, navegación y sonido

A.2.1. Kinect



Figura A.8: Kinect

Kinect	
Características	Sensores
Campo de visión	57.5 grados horizontal por 43.5 grados vertical
Profundidad resoluble	0.8[m]-4.0[m]
Flujo de color	640x480x24 bpp 4:3 RGB @ 30fps 640x480x16bpp 4:3 YUV @ 15fps
Infrarrojo	Sin flujo IR
Registro	Color + ruta
Ruta de datos	USB 2.0
Latencia	90 ms con procesos
Motor de inclinación	Sólo vertical

Tabla A.9: Kinect

A.2.2. Hokuyo UHG-08LX



Figura A.9: Hokuyo UHG-08LX

Hokuyo UHG-08LX Scanning Laser	
Alimentación	12[V]
Rango de detección	De 20 a 8000[mm]
Exactitud	De 100 a 1000[mm]
Resolución angular	0.36grados(360grados/1,024 pasos)
Fuente de luz	Diodo laser semiconductor
Tiempo de escaneo	67[msec/scan]
Nivel de sonido	menos de 25dB
Interface	USB2.0 (velocidad completa)
Salida sincrona	NPN colector abierto
Comando del sistema	Comanda diseñado exclusivamente SCIP ver. 2.0
Conexión	Salida de voltaje y sincronía: 2
Iluminación ambiente	Lampara de alogeno/mercurio: 10,000lx o menos, fluorescente: 6,000lx(máx)
Ambiente (temperatura/humedad)	-10 a 50 [grados C], menos del 85 % RH
Resistencia a la vibración	Amplitud doble 1.5[mm], de 10 a 55[Hz], 2 veces en cada dirección X, Y y Z
Resistencia al impacto	196[m/s], 10 veces en las direcciones X, Y y Z
Peso	Aprox. 500[g](con el cable conectado)

Tabla A.10: Hakuyo UHG-08LX

A.2.3. Microfono RODE



Figura A.10: Microfono Rode NTG-2

Microfono Rode NTG-2	
Principio acustico	Line Gradient
Electronica	Conversor de impedancia JFET con un transformador de salida balanceado
Capsula	0.50"
Tipo de dirección	End
Rango de frecuencia	20Hz-20kHz
Impedancia de salida	250[ohms]
Nivel de sonido	131dB SPL(@ 1kHz, 1% THD en carga de 1kohm)
Máximo nivel de salida	6.9[mV]
Sensibilidad	-36.0dB re 1[Volt/pascal] (15[mV] @ 94dB SPL)+/- 2dB
Nivel de ruido equivalente	18dB-A
Opciones de alimentación	Pilas AA o P48
Peso	161[gm]
Dimensiones	280[mmH]x22[mmW]x22[mmD]
Salida	XLR

Tabla A.11: Microfono Rode

A.3. Alimentación de Justina

A.3.1. Alimentación bateria Li-po

Para el robot Justina se utilizan 3 baterías conectadas en paralelo



Figura A.11: Bateria Li-po

Batería Li-po 4000mAh a 11.1[V]	
Voltaje	11.1[V] en 3 celdas
Corriente de descarga por hora	4000[mAh]
Tasa de descarga	35C
Plug de carga	JST-XH
Plug de descarga	”T”
Medidas	25x46x144[mm]
Peso	335[gr]

Tabla A.12: Bateria Li-po

A.3.2. ATX configuración

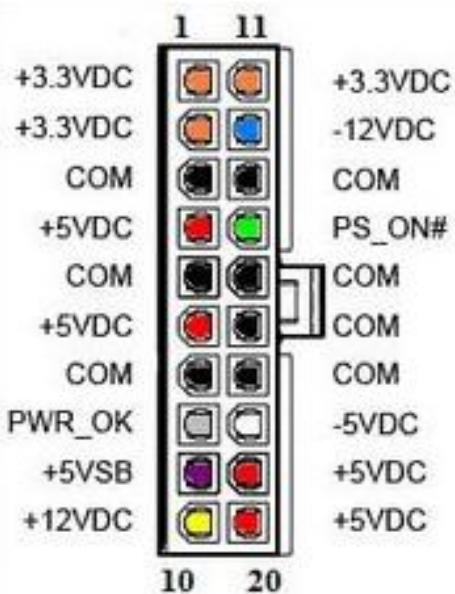


Figura A.12: Pines ATX

A.4. HUBs

A.4.1. Cisco-Linksys USB2HUB4 USB 4-Port Hub



Figura A.13: Cisco-Linksys USB2HUB4 USB 4-Port Hub

USB2HUB4	
Estándar	OHCI UHIC USB 1.1 USB 2.0
Puertos	USB type B Root port 4 USB type A device ports
Número máximo de dispositivos	127
Cable	Shielded USB 2.0
Environmental	
Dimensiones	4.52" x 0.75" x 2.675"
Masa	70[g]
Alimentación	5[V] DC a 2.4[A]
Temperatura de operación	0 a 70 grados
Temperatura en almacenamiento	-20 a 176 grados
Humedad de operación	0 a 95 % sin condensación
Humedad en almacenamiento	0 a 95 % sin condensación

Tabla A.13: USB2HUB4

A.4.2. HUB Startech ST4300PBU3



Figura A.14: HUB Startech ST4300PBU3

USB2HUBST4300PBU3	
Hardware	
Tipo de Bus	USB 3.0
Chipset ID	VLI-VL812
Interface	USB 3.0
Puertos	4
Rendimiento	
Rango máximo de transferencia de datos	5Gbps
Tipo y rango	USB 3.0-5Gbit/s
Conectores	
Puertos externos	1-USB tipo A (9 pines) USB 3.0 macho 4-USB tipo A (9 pines) USB 3.0 hembra
Software	
Compatibilidad con SO	SO independiente; Sin software o drivers adicionales requeridos
Notas especiales/Requerimientos	
Requerimientos de sistema y cables	Puerto USB disponible
Indicadores	
LED indicador	1 - power
Power	
Fuente de poder	USB-Powered
Entorno	
Humedad	20 80 % RH
Temperatura de operación	-5 grados C a 45 gradosC
Temperatura en almacenamiento	-10 grados C a 75 grados C

Tabla A.14: USB2HUB4

APÉNDICE B

Software

B.1. ROS Introducción

ROS es un *middleware* de código abierto (open source) que provee la funcionalidad comúnmente necesaria en el desarrollo de software para robots móviles autónomos, como paso de mensajes y manejo de paquetes. La robot Justina utiliza ROS como plataforma de desarrollo.

ROS puede describirse en dos niveles conceptuales: el sistema de archivos y el grafo de procesos.

El sistema de archivos. Se refiere al modo en que están organizados los recursos en disco:

- **Workspace:** Se refiere a las carpetas que contienen paquetes de ROS.
- **Paquete:** Es la principal unidad de organización de software en ROS. Pueden contener nodos, bibliotecas, datasets, archivos de configuración y otros.
- **Manifiesto:** Definido por el archivo package.xml en cada paquete. Provee meta-datos acerca de cada paquete.
- **Mensaje:** Archivos con extensión .msg. Definen estructuras de datos para el paso de mensajes en ROS.
- **Servicio:** Archivos con extensión .srv. Definen estructuras de tipo request-response. Utilizan mensajes para dicha definición.

Grafo de procesos. Es una red *peer-to-peer* de procesos. Los componentes básicos son:

- **Roscore:** Inicializa el sistema ROS: un master + rosout + un servidor de parámetros.
- **Nodos:** Es simplemente un ejecutable que usa ROS para comunicarse con otros nodos.
- **Tópicos:** Algo similar a una variable cuyo contenido puede ser compartido entre todos los nodos mediante un patrón de publicación y suscripción.
- **Servicios:** Otra forma de comunicar nodos pero con un patrón de petición y respuesta.
- **Servidor de parámetros:** Es un diccionario compartido. Todos los nodos pueden leer y escribir parámetros en tiempo de ejecución.

B.1.1. Instalación de ROS indigo para Ubuntu 14.04

Hemos creado paquetes Debian para varias plataformas de ubuntu listadas abajo. Estos paquetes son más eficientes que los creados basados en la fuente y son nuestro método preferido de instalación para Ubuntu.

Si tu necesitas instalar desde la fuente (no recomendado), por favor revisa la sección de ayuda y referencias.

B.1.2. Configura tus repositorios de Ubuntu

Configura tus repositorios de Ubuntu para permitir restringido”, “universo” y ”multiverso”. Puedes seguir ”la guía de Ubuntu”(El enlace se encuentra en ayuda y referencias) para instrucciones para hacer esto.

B.1.3. Prepara tus sources.list

Prepara tu computadora para aceptar software de packages.ros.org. ROS indigo **sólo** soporta Saucy(13.10) y Trusty(14.04) para paquetes debian.

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
```

B.1.4. Configura tus llaves

```
sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net --recv-key 0xB01FA116
```

Puedes intentar el siguiente comando adhiriendo :**80** si tienes el error **gpg: keyserver timed**

```
sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-key 0xB01FA116
```

B.1.5. Instalación

Primero asegúrate que el índice de tu paquete “Debian” este actualizado

```
sudo apt-get update
```

Si estás usando Ubuntu Trusty 14.04.2 y experimentas problemas de dependencia durante la instalación de ROS, debes instalar algunas dependencias del sistema adicionales.

⚠ Do not install these packages if you are using 14.04, it will destroy your X server:

```
sudo apt-get install xserver-xorg-dev-lts-utopic mesa-common-dev-lts-utopic libxatracker-dev-lts-utopic libopenvg1-mesa-dev-lts-utopic libgles2-mesa-dev-lts-utopic libgles1-mesa-dev-lts-utopic libgl1-mesa-dev-lts-utopic libgbm-dev-lts-utopic libegl1-mesa-dev-lts-utopic
```

(Do not install the above package if you are using 14.04, it will destroy your X server)
Alternativamente, intenta instalando esto para arreglar problemas de dependencia:

```
sudo apt-get install libgl1-mesa-dev-lts-utopic
```

Hay muchas diferentes bibliotecas y herramientas en ROS. proveen 4 diferentes configuraciones para que inicies. Puedes también instalar los paquetes de ROS individualmente.

Desktop-Full Install: (Recomendado): ROS, rqt, rviz, bibliotecas generales de robot, simuladores 2D/3D y percepción 2D/3D

```
sudo apt-get install ros-indigo-desktop-full
```

Desktop install:ROS, rqt, rviz, y bibliotecas de robots en general

```
sudo apt-get install ros-indigo-desktop
```

ROS-Base: (Bare Bones) Paquetes de ROS, construcción y bibliotecas de comunicación. Sin herramientas GUI

```
sudo apt-get install ros-indigo-ros-base
```

Paquete individual: Puedes además instalar un paquete específico de ROS (reemplaza)

```
sudo apt-get install ros-indigo-PACKAGE
```

e.g.

```
sudo apt-get install ros-indigo-slam-gmapping
```

Para encontrar paquetes disponibles, utiliza:

```
apt-cache search ros-indigo
```

B.1.6. Inicializar rosdep

Antes de que puedas usar ROS, necesitaras inicializar rosdep. Rosdep le permite instalar fácilmente las dependencias del sistema para la fuente que buscas compilar y requiere correr algunos componentes del núcleo (Core) en ROS.

```
sudo rosdep init  
rosdep update
```

B.1.7. Configuración del entorno

Es conveniente si las variables del entorno de ROS son añadidas automáticamente a tu bash session cada vez que un nuevo shell es ejecutado.

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

Si tienes mas de una distribución de ROS instalada; `./.bashrc` sólo debe generarse la configuración. `bash` para la version que utilizas actualmente.

Si sólo buscas cambiar el entorno de tu shell actual, puedes escribir:

```
source /opt/ros/indigo/setup.bash
```

Si usas `zsh` en lugar de `bash` necesitas correr los siguientes comandos para configurar tu shell:

```
echo "source /opt/ros/indigo/setup.zsh" >> ~/.zshrc
source ~/.zshrc
```

B.1.8. rosinstall

`Rosinstall` es una linea de comando frecuentemente usada en ROS que es distribuida separadamente. te permite descargar fácilmente muchos arboles de fuente para los paquetes de ROS con un comando.

```
sudo apt-get install python-rosinstall
```

B.2. Instalación de PrimeSense drivers

```
sudo apt-get install freeglut3-dev pkg-config build-essential libxmu-dev libxi-dev
libusb-1.0-0-dev doxygen graphviz mono-complete
```

B.3. Instalación de OpenCV 2.4.9

```
sudo apt-get update
sudo apt-get install build-essential libgtk2.0-dev libjpeg-dev libtiff4-dev
libjasper-dev libopenexr-dev cmake python-dev python-numpy python-tk
libtbb-dev libeigen3-dev yasm libfaac-dev libopencore-amrnb-dev
libopencore-amrwb-dev libtheora-dev libvorbis-dev libxvidcore-dev
libx264-dev libqt4-dev libqt4-opengl-dev sphinx-common texlive-latex-extra
```

```
libv4l-dev libdc1394-22-dev libavcodec-dev libavformat-dev libswscale-dev
default-jdk ant libvtk5-qt4-dev
cd ~
wget http://sourceforge.net/projects/opencvlibrary/files/opencv-unix/2.4.9/opencv-2.4.9.zip
unzip opencv-2.4.9.zip
cd opencv-2.4.9
mkdir build
cd build
cmake -D WITH_TBB=ON -D BUILD_NEW_PYTHON_SUPPORT=ON -D WITH_V4L=ON -D INSTALL_C_EXAMPLES=ON
-D INSTALL_PYTHON_EXAMPLES=ON -D BUILD_EXAMPLES=ON -D WITH_QT=ON -D WITH_OPENGL=ON
-D WITH_VTK=ON -D WITH_OPENNI=ON -D WITH_OPENCL=OFF ..
make
sudo make install
sudo echo "/usr/local/lib" >> /etc/ld.so.conf.d/opencv.conf
sudo ldconfig
```

B.4. Instalando otros paquetes de ROS

```
sudo apt-get install ros-indigo-amcl
sudo apt-get install ros-indigo-tf2-bullet
sudo apt-get install ros-indigo-fake-localization
sudo apt-get install ros-indigo-map-server
sudo apt-get install ros-indigo-sound-play
sudo apt-get install ros-indigo-pocketsphinx
```