CAPÍTULO 1

Nodos de ROS

1.1. Hardware

Sólo los nodos de esta carpeta interactúan con hardware. Todos los demás nodos sólo usan tópicos y servicios para obtener información del hardware.

1.1.1. Head

Dentro del paquete `head` se encuentra el nodo `head_node.py`, 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 articulación, entonces se posicionará en la cota superior o inferior según sea el caso. En modo de simulación existe un nodo llamado `head_simul_node.py`, dicho nodo publica y se suscribe a los mismos tópicos.

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

Tabla 1.1: 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"/>
```

1.1.2. Arms

El paquete `arms` contiene los nodos encargados del control de posición de las articulaciones de los brazos, los nombres de los ejecutables son `left_arm_node.py` y, `right_arm_node.py`. Debido a que ambos nodos operan del mismo modo, únicamente se explicará el nodo `left_arm_node.py`, la diferencia radica en los nombres dados a los tópicos, por ejemplo, el tópico */hardware/left_arm/current_pose* se llama */hardware/right_arm/current_pose* para el caso del nodo correspondiente al brazo derecho.

El nodo `left_arm_node.py` 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. Para el caso de simulación se encuentran los nodos `right_arm_simul_node.py` y `left_arm_simul_node.py`, los cuales funcionan de manera simular a los nodos de puesta en marcha del robot.

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 1.2: Nodo /hardware/left_arm

Sintaxis en un archivo launch

Para correr el nodo `left_arm_node.py` 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="--port1
/dev/justinaLeftArm --baud1 1000000"/>
```

1.1.3. Mobile base

En el paquete `mobile_base` se encuentra el nodo `omni_base_node.py`, este nodo se encarga de controlar el movimiento de la base estableciendo la rapidez deseada para 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. Además 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/odometry [nav_msgs/Odometry]	Odometría calculada con las lecturas de los encoder de los motores
	/hardware/robot_state/base_battery [std_msgs/Float32]	Voltaje de alimentación de los motores de la base
	/tf [tf/tfMessage]	Transformación de *base-link* a *odom*
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 de la base en el plano *xy*, y velocidad angular deseada de la base en el eje *z*
	/hardware/mobile_base/speeds [std_msgs/Float32MultiArray]	Rapideces instantáneas deseadas para las ruedas derecha e izquierda

Tabla 1.3: 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"/>
```

1.1.4. Laser simulator

1.1.5. 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 1.4: Nodo /hardware/torso

1.1.6. Descripción del robot: URDF

head_pan head_node head_simul_node
head_tilt.

En el archivo *justina.xml* se encuentra el modelo del robot Justina; en la Figura 1.1 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 1.2 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`.

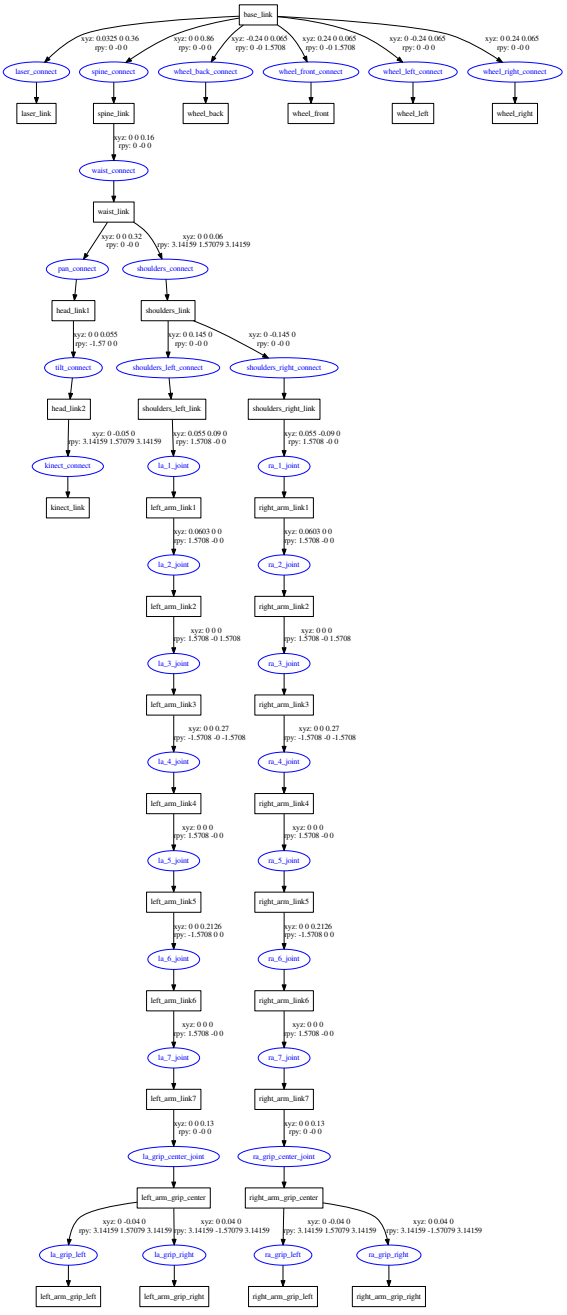

Figura 1.1: Árbol de transformaciones

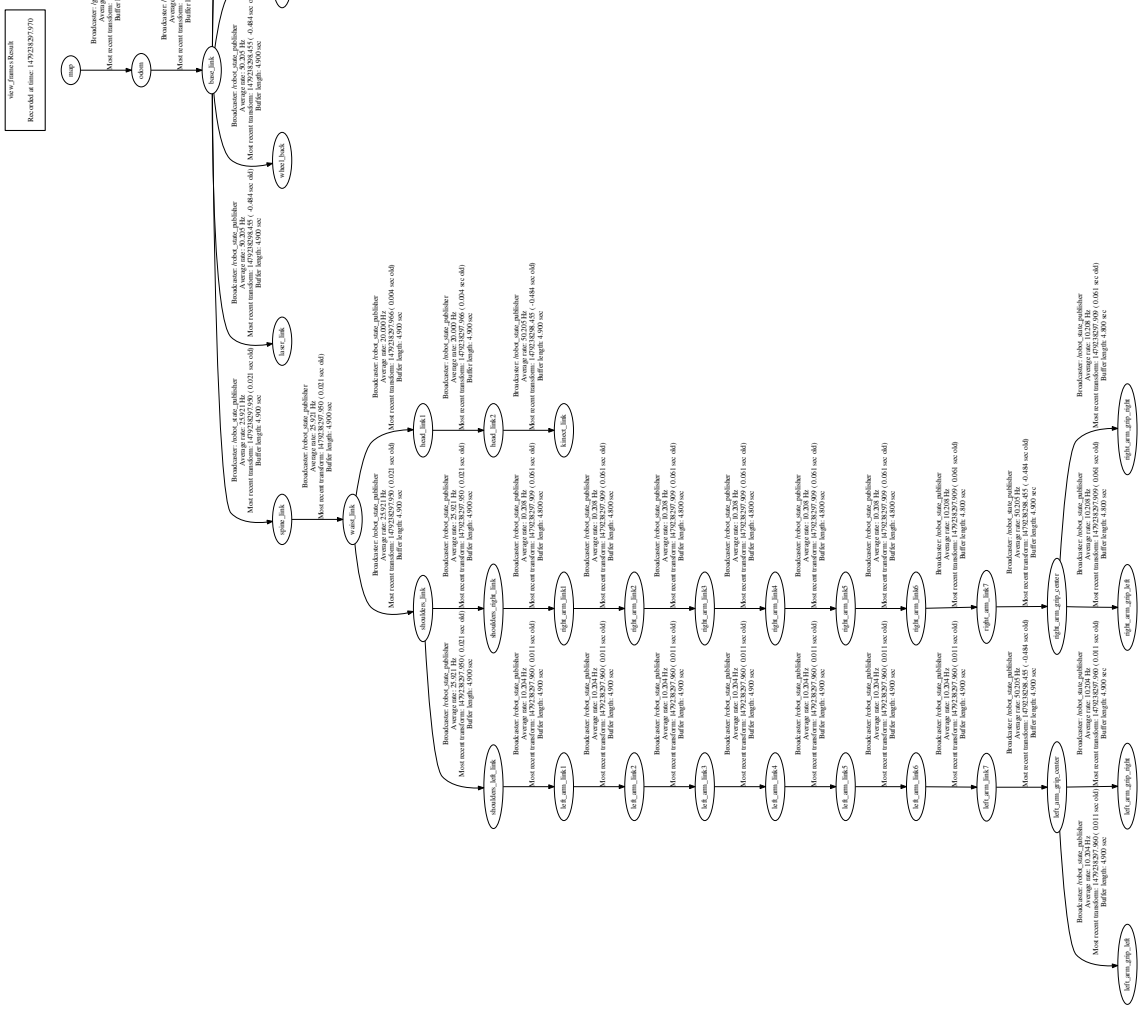

Figura 1.2: Árbol de transformaciones con los marcos de referencia *map* y *odom*

1.2. Navigation

1.2.1. 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 1.5: Nodo /navigation/obs_avoid/obstacle_detector

1.2.2. Mvn Planner

Tópicos publicados	/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]	

Tabla 1.6: Nodo /navigation/mvn_pln

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

Tabla 1.7: Nodo /navigation/mvn_pln

Tópicos publicados	/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]	

Tabla 1.8: 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 1.9: Nodo /navigation/mvn_pln

Tópicos suscritos	/hardware/robot_state/stop [std_msgs/Empty] /navigation/mvn_pln/get_close_xya [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]	

Tabla 1.10: 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 1.11: Nodo /navigation/mvn_pln

1.3. Vision

1.3.1. Face recognition

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 1.12: Nodo /vision/face_recog

Tópicos suscritos	/vision/face_recognizer/ run_face_recognizer [std_msgs/Empty] /vision/face_recognizer/run_face_trainer [std_msgs/String] /vision/face_recognizer/stop_recog [std_msgs/Empty] /vision/face_recognizer/start_recog [std_msgs/Empty]	

Tabla 1.13: Nodo /vision/face_recog

1.3.2. Line finder

Tópicos suscritos	/hardware/head/current_pose [std_msgs/Float32MultiArray]	
Servicios	/vision/line_finder/find_lines_ransac [vision_msgs/FindLines]	

Tabla 1.14: Nodo /vision/line_finder

1.3.3. Object recognition

El nodo se utiliza para detectar e identificar objetos sobre planos horizontales. Además, contiene las funciones para generar datos de entrenamiento que son utilizados durante el proceso de identificación. Actualmente, lo primero que se realiza para hacer la detección de objetos es una identificación y extracción de planos horizontales, para después segmentar la nube de puntos sobre estos planos y así identificar y clusterizar los objetos. Todo el proceso descrito anteriormente se realiza utilizando solamente información 3D. Una vez segmentados los objetos, estos se tratan de identificar utilizando 3 características:

- Altura (distancia del punto mas alejado del plano al plano)
- Forma (se obtienen momentos de Hu de la proyección de los puntos del objeto sobre el plano)
- Color (Se comparan Histogramas en HSV)

Estas tres características pasan por una etapa de clasificación similar a un clasificador en cascada y se devuelve el nombre del objeto entrenado más semejante al detectado. Además de esto, calcula el centroide de los objetos utilizando simplemente la media de todos los puntos 3D que lo componen.

Tópicos publicados	/vision/obj_reco /recognizedObjectes [vision_msgs/VisionObjectList]	Si el tópico enableRecognizeTopic recibe el valor True en el callback, entonces el nodo detecta y trata de identificar todos los objetos que se encuentren sobre los planos horizontales utilizando los datos de entrenamiento previos, cada que hay un nuevo frame de Kinect. En el tópico se publican los nombres de los objetos identificados, así como las coordenadas de su centroide.

Tabla 1.15: Nodo /vision/obj_reco

Tópicos suscritos	/vision/obj_reco /enableRecognizeTopic [std_msgs/Bool]	Habilita o deshabilita la función de reconocimiento de objetos cada que existe un nuevo frame de Kinect.
	/vision/obj_reco /enableDetectWindow [std_msgs/Bool]	Habilita o deshabilita la ventana donde se muestra el reconocimiento de objetos cada que existe un nuevo frame de Kinect. Para evitar consumir recursos, se recomienda que esta este en false, salvo para depuración o entrenamiento.
	/hardware/point_cloud_man/ rgbd_wrt_robot [sensor_msgs/PointCloud2]	Si alguno de los dos tópicos anteriores esta habilitado, el nodo espera este tópico para realizar la detección e identificación de objetos con cada nuevo frame.

Tabla 1.16: Nodo /vision/obj_reco

Servicios	/vision/obj_reco/det_objs [vision_msgs/DetectObjects]	El nodo detecta el plano horizontal más grande en la escena, y detecta todos los objetos sobre este usando información 3D. Una vez detectados, estos objetos tratan de ser reconocidos utilizando las características extraídas durante la fase de entrenamiento. El servicio regresa una lista de todos los objetos **reconocidos**, utilizando su nombre y su centroide con respecto del robot.
	/vision/geometry_finder /findPlane [vision_msgs/FindPlane]	Detecta y muestra en una ventana todos los planos **horizontales** detectados.
	/vision/obj_reco/trainObject [vision_msgs/TrainObject]	Se le envía un string con el nombre de un objeto y el nodo detecta y extrae las características 3D y 2D del objeto más cercano al centro del robot (los objetos, para ser detectados, deben encontrarse sobre un plano horizontal). Un archivo XML con las características del objeto, nombre y una imagen del objeto es guardado en el directorio: \src\vision\obj_reco\TrainingDir \NombreDelObjeto

Tabla 1.17: Nodo /vision/obj_reco

1.3.4. 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 1.18: Nodo /vision/skeleton_finder

1.4. Human-robot interaction

1.4.1. gui

Sirve para operar todo el robot desde una interfaz gráfica cuyas funciones se detallan en la sección **??**.

Tópicos publicados	/hardware/point_cloud_man/save_cloud [std_msgs/String] /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] /hardware/robot_state/stop [std_msgs/Empty] /vision/face_recognizer/start_recog_old [std_msgs/Empty] /manipulation/manip_pln/ra_goto_loc [std_msgs/String] /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]	

Tabla 1.19: Nodo /hri/justina_gui

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

Tabla 1.20: Nodo /hri/justina_gui

Tópicos publicados	/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]	

Tabla 1.21: Nodo /hri/justina_gui

Tópicos publicados	/vision/thermal_vision/stop_video [std_msgs/Empty] /vision/skeleton_finder/stop_recog [std_msgs/Empty] /vision/face_recognizer/clearfacesdb [std_msgs/Empty] /manipulation/manip_pln/ hd_goto_angles [std_msgs/Float32MultiArray] /hardware/left_arm/goal_torque [std_msgs/Float32MultiArray] /vision/face_recognizer/clearfacesdbbyid [std_msgs/String] /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] /hardware/left_arm/goal_pose [std_msgs/Float32MultiArray]	

Tabla 1.22: Nodo /hri/justina_gui

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

Tabla 1.23: Nodo /hri/justina_gui

Tópicos publicados	/navigation/path_planning/simple_move /goal_path [nav_msgs/Path] /vision/skeleton_finder/start_recog [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]	

Tabla 1.24: Nodo /hri/justina_gui

Tópicos suscritos	/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]	

Tabla 1.25: Nodo /hri/justina_gui

Tópicos suscritos	/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]	

Tabla 1.26: 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 1.27: Nodo /hri/justina_gui

1.4.2. 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 1.28: Nodo /hri/human_follower

1.4.3. 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 1.29: Nodo /hri/leg_finder

1.4.4. 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 1.30: Nodo /hri/qr_reader

1.4.5. SP gen

Tópicos suscritos	/hri/sp_gen/say [std_msgs/String]	

Tabla 1.31: Nodo /hri/sp_gen

1.5. Interoperation

1.5.1. 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 1.32: 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" />
```

1.6. Manipulation

1.6.1. Manipulation Planner

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

Tabla 1.33: Nodo /manipulation/manip_pln

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

Tabla 1.34: Nodo /manipulation/manip_pln

Tópicos suscritos	/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 [std_msgs/Float32MultiArray] /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]	

Tabla 1.35: Nodo /manipulation/manip_pln

1.6.2. IK Geometric

Servicios	/manipulation/ik_geometric/ ik_float_array [manip_msgs/InverseKinematicsFloat Array] /manipulation/ik_geometric/ik_path [manip_msgs/InverseKinematicsPath] /manipulation/ik_geometric/ik_pose [manip_msgs/InverseKinematicsPose] /manipulation/ik_geometric/ direct_kinematics [manip_msgs/DirectKinematics]	

Tabla 1.36: Nodo /manipulation/ik_geometric

1.7. Planning

1.7.1. 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 [na- vig_msgs/PathFromMap]	 Cálculo de una ruta uti- lizando el algoritmo de búsqueda A*

Tabla 1.37: 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"/>
```

1.7.2. Simple Move

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

Tabla 1.38: Nodo /navigation/path_planning/simple_move

Tópicos suscritos	/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 [geometry_msgs/Pose2D]	

Tabla 1.39: Nodo /navigation/path_planning/simple_move

1.8. Paquetes de terceros

1.8.1. 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]	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 1.40: 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>
```

1.8.2. 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 *odom_frame_id*) 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 1.41: Nodo /navigation/localization/loc_amcl

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

Tabla 1.42: 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_typ*.

```
<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>
```

1.8.3. 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 utlizan *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 1.43: Nodo /robot_state_publisher

Sintaxis en un archivo launch
Lanzamiento del nodo y ajuste del parámetro *robot_description*.

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

1.8.4. 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 (*header*) del mapa publicado

Tabla 1.44: 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="$(find knowledge)/navigation/occupancy_grids/bioroboanexo3.yaml"/>$
```

1.8.5. Rviz

Tópicos publicados	/clicked_point [geometry_msgs/PointStamped] /move_base_simple/goal [geometry_msgs/PoseStamped] /initialpose [geometry_msgs/ PoseWithCovarianceStamped]	
Tópicos suscritos	/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]	

Tabla 1.45: Nodo /hri/rviz

1.8.6. joy node

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 1.46: 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"/>
```