Driver de ROS para el robot móvil RoMAA

Gonzalo Perez-Paina, David Gaydou, Gastón Araguás

Centro de Investigación en Informática para la Ingeniería
Universidad Tecnológica Nacional, FRC
`http://ciii.frc.utn.edu.ar`

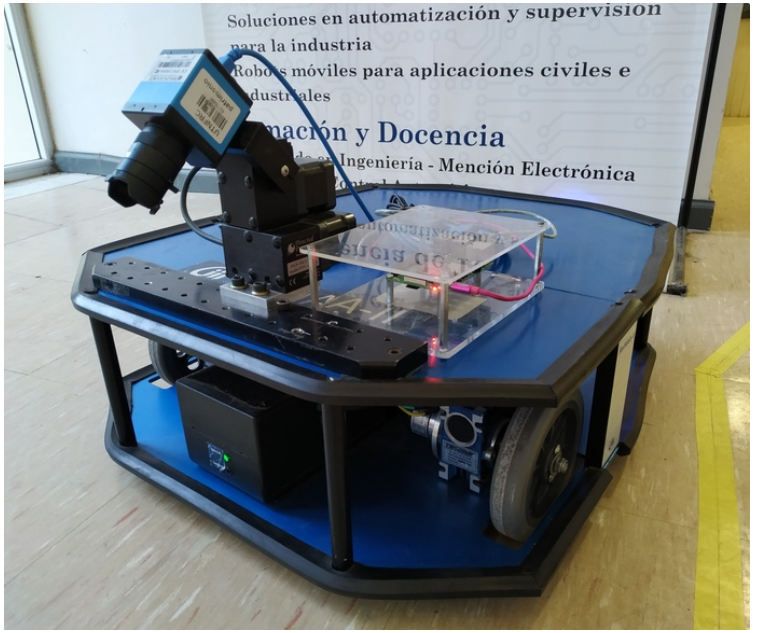
X Jornadas Argentinas de Robótica
– 2019 –

Contenido

Motivación y objetivo

Motivación

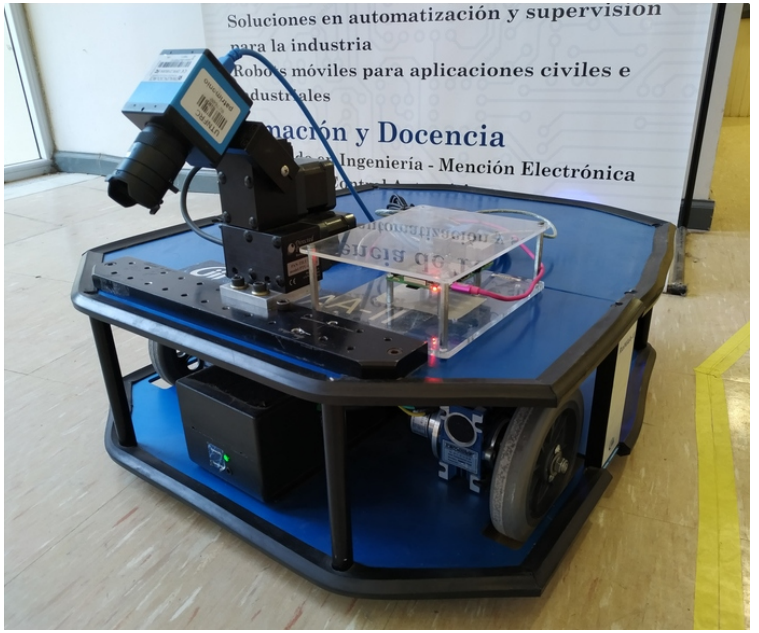
- En el CIII se desarrolló un el robot móvil RoMAA como plataforma experimental en robótica móvil y visión por computadoras
- En la actualidad el framework de desarrollo ROS es utilizado por gran cantidad de robots en universidades y compañías alrededor del mundo



Motivación y objetivo

Motivación

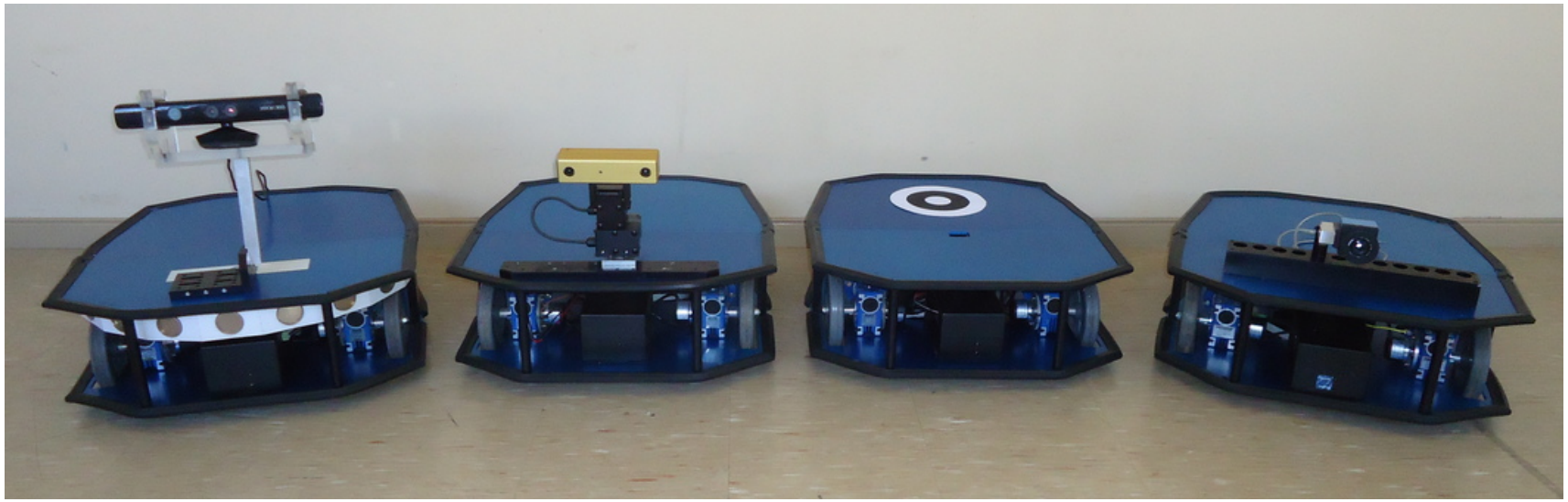
- En el CIII se desarrolló un el robot móvil RoMAA como plataforma experimental en robótica móvil y visión por computadoras
- En la actualidad el framework de desarrollo ROS es utilizado por gran cantidad de robots en universidades y compañías alrededor del mundo



Objetivos

- Desarrollar un nodo driver para el robot móvil RoMAA para poder utilizar el robot dentro del ecosistema ROS
- Para poder utilizar dicho entorno de desarrollo para evaluar nuevos algoritmos de robótica y aprovechar la gran cantidad de paquetes de software disponibles

El Robot Móvil de Arquitectura Abierta – RoMAA

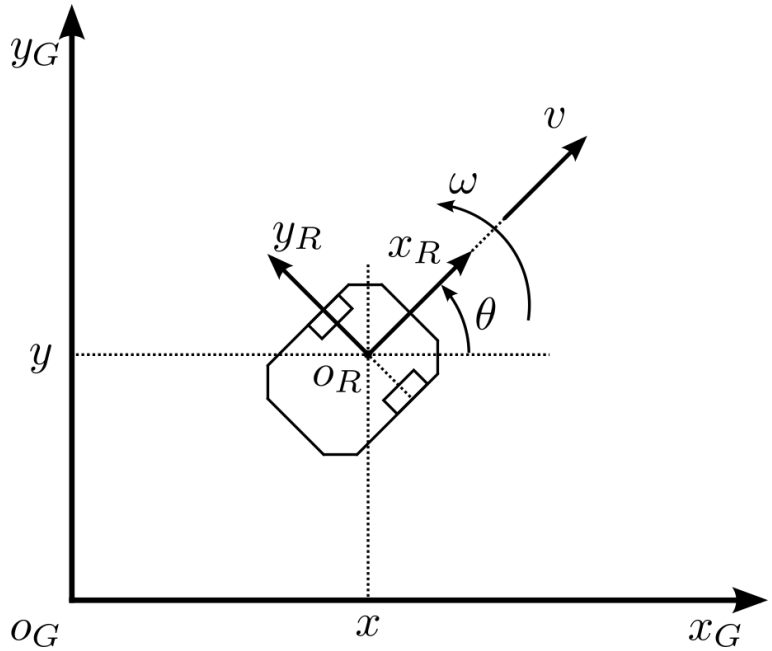


- El RoMAA es un robot de tracción diferencia diseñado como plataforma de experimentación en las áreas de investigación de robótica móvil y visión por computadoras
- Permite adaptarse a diferentes experimentos dada su flexibilidad en el montaje de sensores y actuadores

El Robot Móvil de Arquitectura Abierta – RoMAA

El Robot Móvil de Arquitectura Abierta – RoMAA

- El robot se controla mediante velocidad linean v y angular ω, en el sistema de coordenadas local al robot
- La odometría permite conocer la localización (x, y, θ) del robot en un sistema de coord. global



El Sistema Operativo de Robótica – ROS

No es exactamente un SO como el que se instala en la PC.
Es mas bien un middle-ware que vive entre el SO y
los programas que se desarrollan para el robot.

El Sistema Operativo de Robótica – ROS

No es exactamente un SO como el que se instala en la PC.
Es mas bien un middle-ware que vive entre el SO y
los programas que se desarrollan para el robot.

Filosofía ROS

- Programas individuales que se comunican (P2P) sobre una API definida

El Sistema Operativo de Robótica – ROS

No es exactamente un SO como el que se instala en la PC.
Es mas bien un middle-ware que vive entre el SO y
los programas que se desarrollan para el robot.

Filosofía ROS

- Programas individuales que se comunican (P2P) sobre una API definida
- Debido al tipo de comunicación, puede ser distribuido en múltiples PCs

El Sistema Operativo de Robótica – ROS

No es exactamente un SO como el que se instala en la PC.
Es mas bien un middle-ware que vive entre el SO y
los programas que se desarrollan para el robot.

Filosofía ROS

- Programas individuales que se comunican (P2P) sobre una API definida
- Debido al tipo de comunicación, puede ser distribuido en múltiples PCs
- Permite construir un sistema grande y complejo distribuidos

El Sistema Operativo de Robótica – ROS

No es exactamente un SO como el que se instala en la PC.
Es mas bien un middle-ware que vive entre el SO y
los programas que se desarrollan para el robot.

Filosofía ROS

- Programas individuales que se comunican (P2P) sobre una API definida
- Debido al tipo de comunicación, puede ser distribuido en múltiples PCs
- Permite construir un sistema grande y complejo distribuidos
- Soporta múltiples lenguajes de programación: `C++`, `python`, `MATLAB`

El Sistema Operativo de Robótica – ROS

No es exactamente un SO como el que se instala en la PC.
Es mas bien un middle-ware que vive entre el SO y
los programas que se desarrollan para el robot.

Filosofía ROS

- Programas individuales que se comunican (P2P) sobre una API definida
- Debido al tipo de comunicación, puede ser distribuido en múltiples PCs
- Permite construir un sistema grande y complejo distribuidos
- Soporta múltiples lenguajes de programación: `C++`, `python`, `MATLAB`
- Es liviano: no incluye los algoritmos de robótica, son bibliotecas aparte (ROS wrapper)

El Sistema Operativo de Robótica – ROS

No es exactamente un SO como el que se instala en la PC.
Es mas bien un middle-ware que vive entre el SO y
los programas que se desarrollan para el robot.

Filosofía ROS

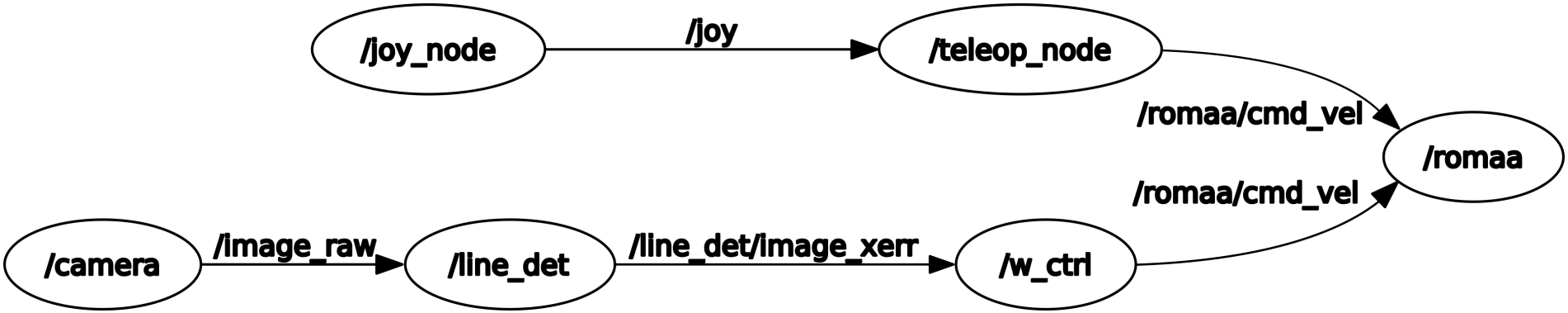
- Programas individuales que se comunican (P2P) sobre una API definida
- Debido al tipo de comunicación, puede ser distribuido en múltiples PCs
- Permite construir un sistema grande y complejo distribuidos
- Soporta múltiples lenguajes de programación: `C++`, `python`, `MATLAB`
- Es liviano: no incluye los algoritmos de robótica, son bibliotecas aparte (ROS wrapper)
- Free and open-source

Terminología de ROS

El software de robots se desarrolla como una colección de programas llamados nodos que se ejecutan al mismo tiempo.

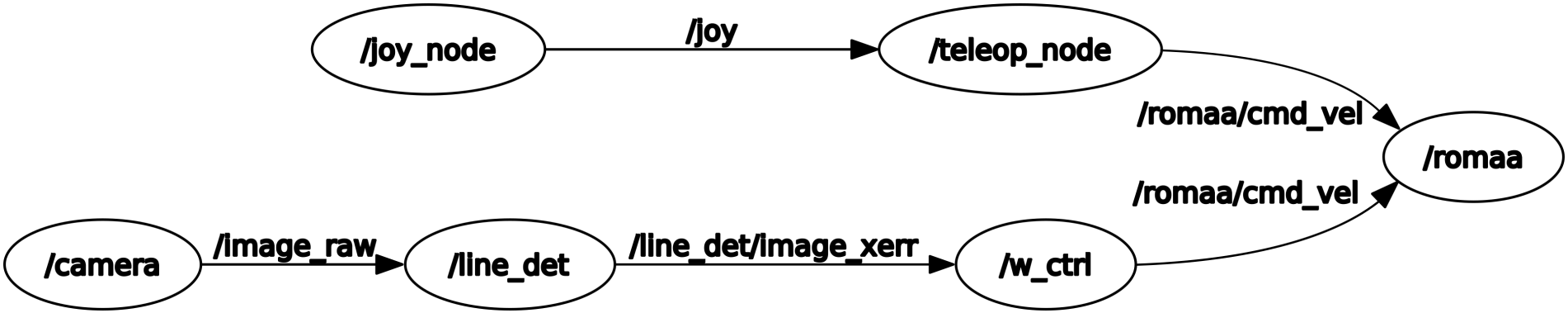
Terminología de ROS

El software de robots se desarrolla como una colección de programas llamados nodos que se ejecutan al mismo tiempo.



Terminología de ROS

El software de robots se desarrolla como una colección de programas llamados nodos que se ejecutan al mismo tiempo.

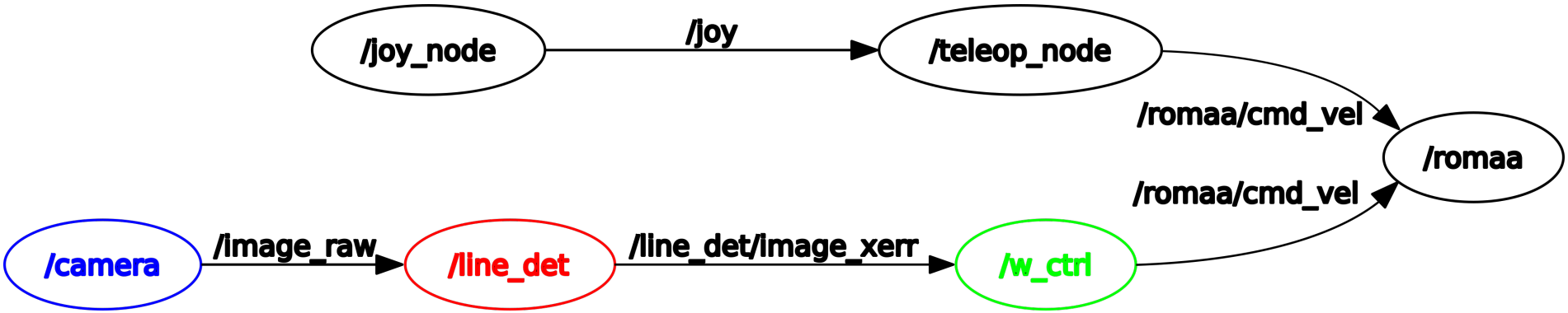


1. Nombre de recurso

Cadena de texto que identifica a un recurso dentro del grafo computacional de ROS

Terminología de ROS

El software de robots se desarrolla como una colección de programas llamados nodos que se ejecutan al mismo tiempo.

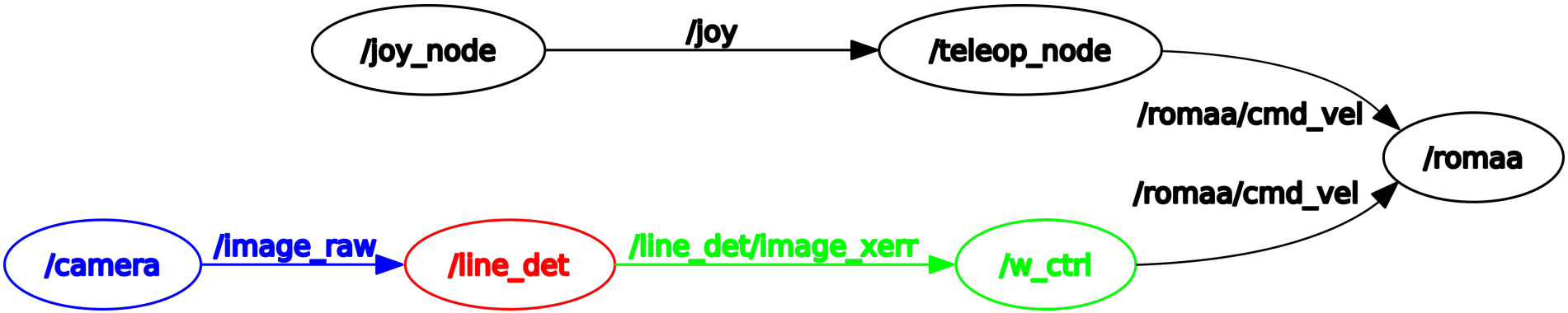


1. Nombre de recurso
2. Nodo

Cualquier proceso que utiliza una API cliente de ROS

Terminología de ROS

El software de robots se desarrolla como una colección de programas llamados nodos que se ejecutan al mismo tiempo.

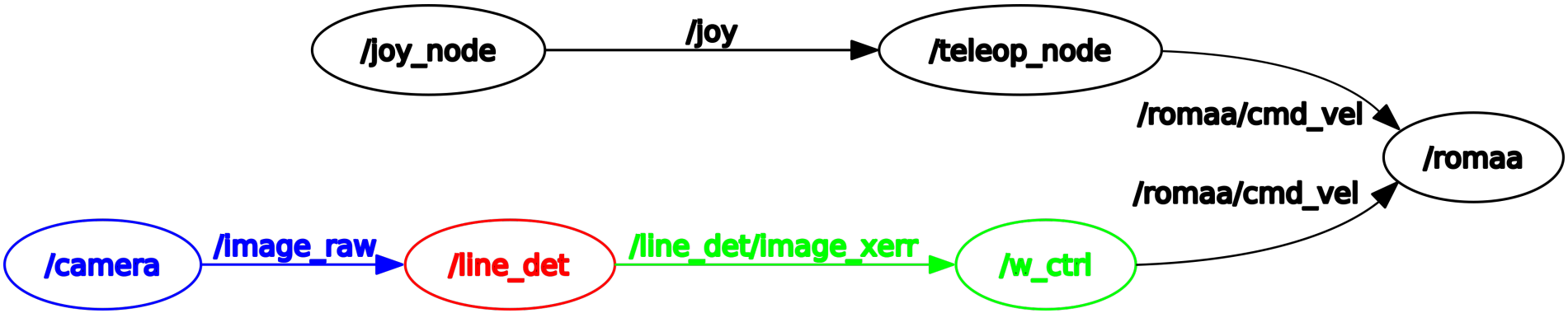


1. Nombre de recurso
2. Nodo
3. Tópico

Nombre del canal de comunicación (unidireccional, asíncrono, fuertemente tipado) con mecanismo de comunicación *Publicador/Subscriptor*

Terminología de ROS

El software de robots se desarrolla como una colección de programas llamados nodos que se ejecutan al mismo tiempo.

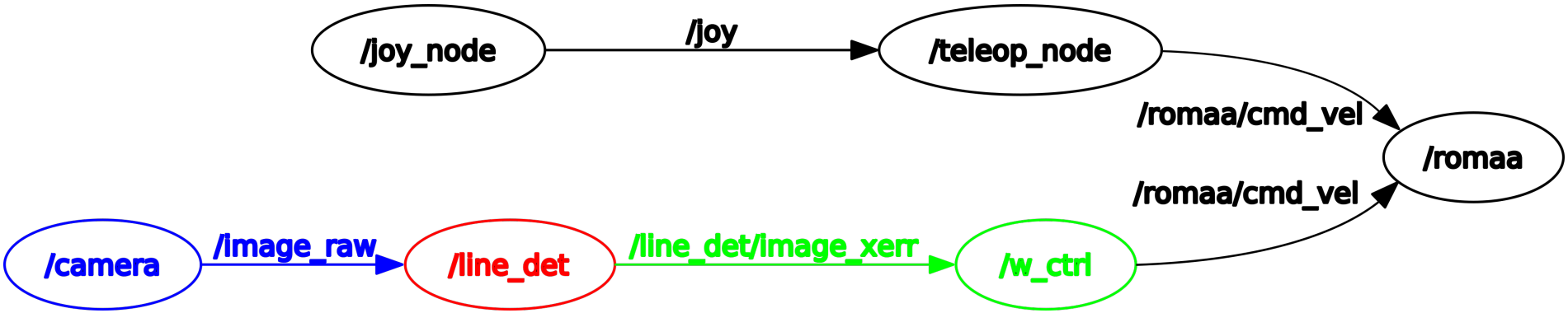


1. Nombre de recurso
2. Nodo
3. Tópico
4. Mensaje

Estructura de dato específica (puede ser anidada)

Terminología de ROS

El software de robots se desarrolla como una colección de programas llamados nodos que se ejecutan al mismo tiempo.



1. Nombre de recurso
2. Nodo
3. Tópico
4. Mensaje
5. Servicio

Llamada a procedimiento remoto síncrono

Nodo driver del RoMAA

Comunicación a bajo nivel y biblioteca de comunicación

- Trama de comunicación por puerto serie que implementa diferentes comandos (`CMD_SET_SPEED`, `CMD_GET_ODOMETRY`, `CMD_ENABLE_MOTOR`, etc.)

Nodo driver del RoMAA

Comunicación a bajo nivel y biblioteca de comunicación

- Trama de comunicación por puerto serie que implementa diferentes comandos (`CMD_SET_SPEED`, `CMD_GET_ODOMETRY`, `CMD_ENABLE_MOTOR`, etc.)
- Biblioteca de comunicación implementada como clase de C++ (POO) con funciones miembros que se corresponden uno a uno a los comandos de bajo nivel (`set_speed`, `get_odometry`, `enable_motor`, etc.)

Nodo driver del RoMAA

Comunicación a bajo nivel y biblioteca de comunicación

- Trama de comunicación por puerto serie que implementa diferentes comandos (`CMD_SET_SPEED`, `CMD_GET_ODOMETRY`, `CMD_ENABLE_MOTOR`, etc.)
- Biblioteca de comunicación implementada como clase de C++ (POO) con funciones miembros que se corresponden uno a uno a los comandos de bajo nivel (`set_speed`, `get_odometry`, `enable_motor`, etc.)

Nodo driver de ROS: utiliza el mecanismo de comunicación *publicador/subscriptor* y llamada a *servicios*

Nodo driver del RoMAA

Comunicación a bajo nivel y biblioteca de comunicación

- Trama de comunicación por puerto serie que implementa diferentes comandos (`CMD_SET_SPEED`, `CMD_GET_ODOMETRY`, `CMD_ENABLE_MOTOR`, etc.)
- Biblioteca de comunicación implementada como clase de C++ (POO) con funciones miembros que se corresponden uno a uno a los comandos de bajo nivel (`set_speed`, `get_odometry`, `enable_motor`, etc.)

Nodo driver de ROS: utiliza el mecanismo de comunicación *publicador/ subscriptor* y llamada a *servicios*

- Publicador: publica mensajes de odometría bajo el tópico `romaa/odom`

Nodo driver del RoMAA

Comunicación a bajo nivel y biblioteca de comunicación

- Trama de comunicación por puerto serie que implementa diferentes comandos (`CMD_SET_SPEED`, `CMD_GET_ODOMETRY`, `CMD_ENABLE_MOTOR`, etc.)
- Biblioteca de comunicación implementada como clase de C++ (POO) con funciones miembros que se corresponden uno a uno a los comandos de bajo nivel (`set_speed`, `get_odometry`, `enable_motor`, etc.)

Nodo driver de ROS: utiliza el mecanismo de comunicación *publicador/subscriptor* y llamada a *servicios*

- Publicador: publica mensajes de odometría bajo el tópico `romaa/odom`
- Subscriptor: subscribe a comandos de velocidad bajo el tópico `romaa/cmd_vel`

Nodo driver del RoMAA

Comunicación a bajo nivel y biblioteca de comunicación

- Trama de comunicación por puerto serie que implementa diferentes comandos (`CMD_SET_SPEED`, `CMD_GET_ODOMETRY`, `CMD_ENABLE_MOTOR`, etc.)
- Biblioteca de comunicación implementada como clase de C++ (POO) con funciones miembros que se corresponden uno a uno a los comandos de bajo nivel (`set_speed`, `get_odometry`, `enable_motor`, etc.)

Nodo driver de ROS: utiliza el mecanismo de comunicación *publicador/subscriptor* y llamada a *servicios*

- Publicador: publica mensajes de odometría bajo el tópico `romaa/odom`
- Subscriptor: subscribe a comandos de velocidad bajo el tópico `romaa/cmd_vel`
- Servicios:
  1. Reiniciar la odometría: `reset_odometry`

Nodo driver del RoMAA

Comunicación a bajo nivel y biblioteca de comunicación

- Trama de comunicación por puerto serie que implementa diferentes comandos (`CMD_SET_SPEED`, `CMD_GET_ODOMETRY`, `CMD_ENABLE_MOTOR`, etc.)
- Biblioteca de comunicación implementada como clase de C++ (POO) con funciones miembros que se corresponden uno a uno a los comandos de bajo nivel (`set_speed`, `get_odometry`, `enable_motor`, etc.)

Nodo driver de ROS: utiliza el mecanismo de comunicación *publicador/subscriptor* y llamada a *servicios*

- Publicador: publica mensajes de odometría bajo el tópico `romaa/odom`
- Subscriptor: subscribe a comandos de velocidad bajo el tópico `romaa/cmd_vel`
- Servicios:
  1. Reiniciar la odometría: `reset_odometry`
  2. Fijar valor de odometría: `set_odometry`

Nodo driver del RoMAA

Comunicación a bajo nivel y biblioteca de comunicación

- Trama de comunicación por puerto serie que implementa diferentes comandos (`CMD_SET_SPEED`, `CMD_GET_ODOMETRY`, `CMD_ENABLE_MOTOR`, etc.)
- Biblioteca de comunicación implementada como clase de C++ (POO) con funciones miembros que se corresponden uno a uno a los comandos de bajo nivel (`set_speed`, `get_odometry`, `enable_motor`, etc.)

Nodo driver de ROS: utiliza el mecanismo de comunicación *publicador/subscriptor* y llamada a *servicios*

- Publicador: publica mensajes de odometría bajo el tópico `romaa/odom`
- Subscriptor: subscribe a comandos de velocidad bajo el tópico `romaa/cmd_vel`
- Servicios:
  1. Reiniciar la odometría: `reset_odometry`
  2. Fijar valor de odometría: `set_odometry`
  3. Habilitar/desbabilitar los motores: `enable_motor`

Nodo driver del RoMAA

```
1  #include <ros/ros.h>
2  // Include header files of used messages.
3
4  int main(int argc, char **argv)
5  {
6    ros::init(argc, argv, "romaa");
7    ros::NodeHandle nh("~");
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23   return 0;
24 }
```

Nodo driver del RoMAA

```
1  #include <ros/ros.h>
2  // Include header files of used messages.
3
4  int main(int argc, char **argv)
5  {
6    ros::init(argc, argv, "romaa");
7    ros::NodeHandle nh("~");
8
9    // Define publisher and subscribers and subscriber's callback functions.
10
11
12
13
14
15
16
17
18
19
20
21
22
23   return 0;
24 }
```

Nodo driver del RoMAA

```
1  #include <ros/ros.h>
2  // Include header files of used messages.
3
4  int main(int argc, char **argv)
5  {
6    ros::init(argc, argv, "romaa");
7    ros::NodeHandle nh("~");
8
9    // Define publisher and subscribers and subscriber's callback functions.
10
11   ros::Time current_time;
12   ros::Rate loop_rate(10);// 10 Hz
13   while(ros::ok())
14   {
15
16
17
18
19
20
21
22   }
23   return 0;
24 }
```

Nodo driver del RoMAA

```cpp
#include <ros/ros.h>
// Include header files of used messages.

int main(int argc, char **argv)
{
  ros::init(argc, argv, "romaa");
  ros::NodeHandle nh("~");

  // Define publisher and subscribers and subscriber's callback functions.

  ros::Time current_time;
  ros::Rate loop_rate(10);// 10 Hz
  while(ros::ok())
  { // Fill messages with data to publish and publish them.
    // Use 'current_time' in msg timestamp.

    // Process incoming messages via callbacks.
    ros::spinOnce();

    // Sleep for the time remaining to achieve the publish rate.
    loop_rate.sleep();
  }
  return 0;
}
```

Nodo driver del RoMAA – Publicador/Subscriptor

```
1  #include <nav_msgs/Odometry.h>
2
3
4  nav_msgs::Odometry odom_msg; // Odometry message.
5  ros::Publisher odom_pub = nh.advertise<nav_msgs::Odometry>("odom", 100);
6
7
8
9
10 /* In the while loop: fill 'odom' message with data */
11 odom_pub.publish(odom_msg); // Publish de message.
```

Nodo driver del RoMAA – Publicador/Subscriptor

```
#include <nav_msgs/Odometry.h>
#include <geometry_msgs/Twist.h>

nav_msgs::Odometry odom_msg; // Odometry message.
ros::Publisher odom_pub = nh.advertise<nav_msgs::Odometry>("odom", 100);

ros::Subscriber cmd_vel_sub = nh.subscribe<geometry_msgs::Twist>(
  "cmd_vel", 1, cmdVelCallback);

/* In the while loop: fill 'odom' message with data */
odom_pub.publish(odom_msg); // Publish de message.
```

Nodo driver del RoMAA – Servicios

```
1  // Register servers with the Master
2  ros::ServiceServer reset_odom_srv = nh.advertiseService(
3    "reset_odometry", &resetOdometrySrvCb); // Standard empty msg
4
5  ros::ServiceServer set_odom_srv = nh.advertiseService(
6    "set_odometry", &setOdometrySrvCb);
7
8
9
10
11
12
13
14
15
16
17
18
```

Nodo driver del RoMAA – Servicios

```
1  // Register servers with the Master
2  ros::ServiceServer reset_odom_srv = nh.advertiseService(
3    "reset_odometry", &resetOdometrySrvCb); // Standard empty msg
4
5  ros::ServiceServer set_odom_srv = nh.advertiseService(
6    "set_odometry", &setOdometrySrvCb);
7
8  bool resetOdometrySrvCb(std_srvs::Empty::Request &req,
9      std_srvs::Empty::Response &resp) {
10   // Send reset command to the robot
11   return true;
12 }
13
14
15
16
17
18
```

Nodo driver del RoMAA – Servicios

```
1  // Register servers with the Master
2  ros::ServiceServer reset_odom_srv = nh.advertiseService(
3    "reset_odometry", &resetOdometrySrvCb); // Standard empty msg
4
5  ros::ServiceServer set_odom_srv = nh.advertiseService(
6    "set_odometry", &setOdometrySrvCb);
7
8  bool resetOdometrySrvCb(std_srvs::Empty::Request &req,
9      std_srvs::Empty::Response &resp) {
10   // Send reset command to the robot
11   return true;
12 }
13
14 bool setOdometrySrvCb(romaa_ros::SetOdometry::Request &req,
15   romaa_ros::SetOdometry::Response &resp) {
16   // Set robot odometry using: req.x, req.y and req.theta
17   return true;
18 }
```

Ejemplos de integración – Archivos de lanzamiento

```
1  <?xml version="1.0"?>
2  <launch>
3
4    <arg name="port" default="/dev/ttyUSB0" />
5    <arg name="baudrate" default="115200" />
6
7    <node pkg="romaa_ros" type="romaa_driver" name="romaa" >
8      <param name="port" value="$(arg port)" />
9      <param name="baud" value="$(arg baudrate)" />
10   </node>
11
12   <node pkg="joy" type="joy_node" name="joy" />
13
14   <node pkg="teleop_twist_joy" type="teleop_node" name="teleop" >
15     <remap from="/cmd_vel" to="/romaa/cmd_vel" />
16   </node>
17 </launch>
```

Ejemplos de integración – Archivos de lanzamiento

```
1  <?xml version="1.0"?>
2  <launch>
3
4    <arg name="port" default="/dev/ttyUSB0" />
5    <arg name="baudrate" default="115200" />
6
7    <node pkg="romaa_ros" type="romaa_driver" name="romaa" >
8      <param name="port" value="$(arg port)" />
9      <param name="baud" value="$(arg baudrate)" />
10   </node>
11
12   <node pkg="joy" type="joy_node" name="joy" />
13
14   <node pkg="teleop_twist_joy" type="teleop_node" name="teleop" >
15     <remap from="/cmd_vel" to="/romaa/cmd_vel" />
16   </node>
17 </launch>
```

1. `> roslaunch romaa_ros romaa.launch`

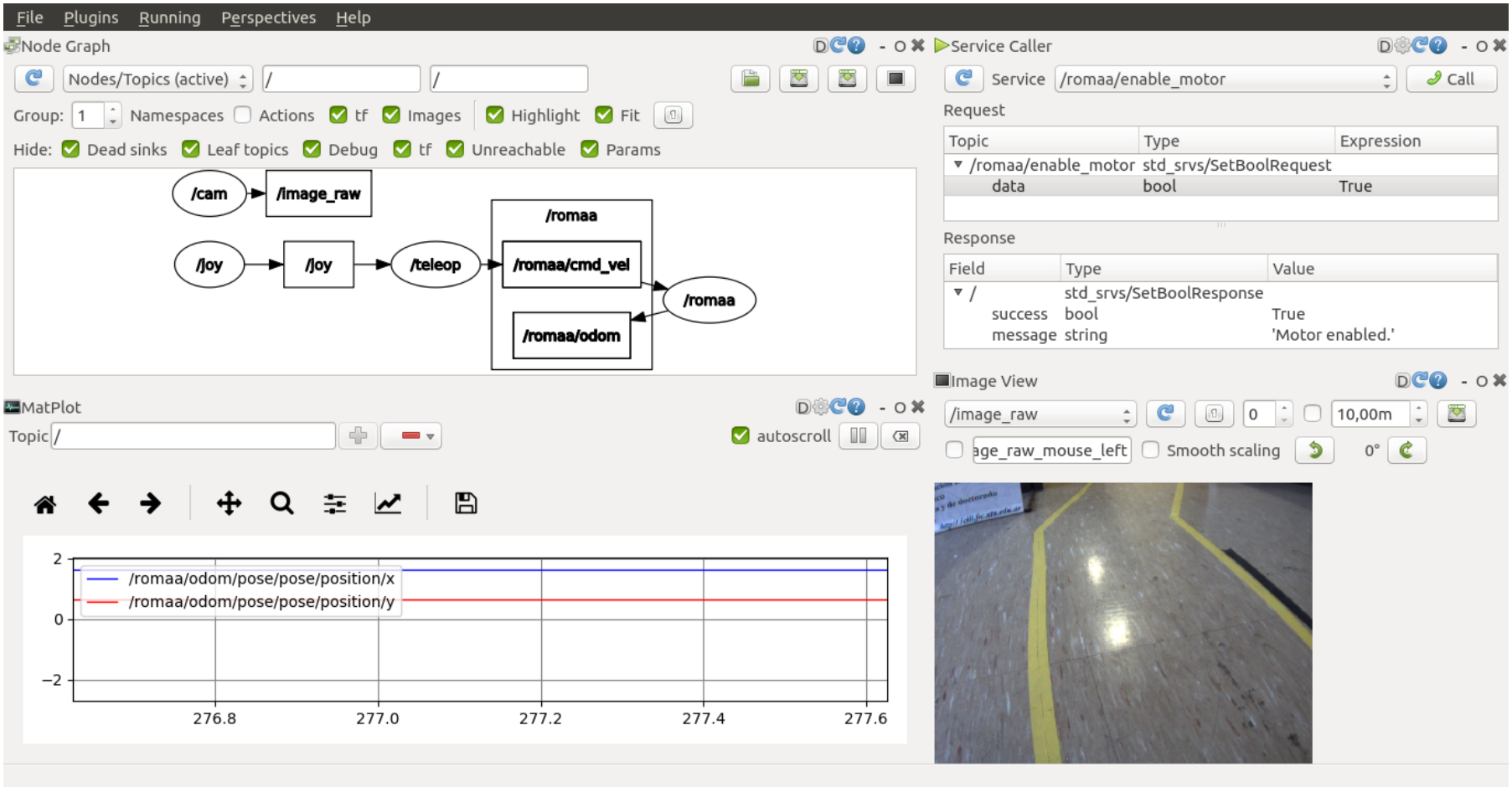
Ejemplos de integración – Archivos de lanzamiento

```
1  <?xml version="1.0"?>
2  <launch>
3
4    <arg name="port" default="/dev/ttyUSB0" />
5    <arg name="baudrate" default="115200" />
6
7    <node pkg="romaa_ros" type="romaa_driver" name="romaa" >
8      <param name="port" value="$(arg port)" />
9      <param name="baud" value="$(arg baudrate)" />
10   </node>
11
12   <node pkg="joy" type="joy_node" name="joy" />
13
14   <node pkg="teleop_twist_joy" type="teleop_node" name="teleop" >
15     <remap from="/cmd_vel" to="/romaa/cmd_vel" />
16   </node>
17 </launch>
```

1. > roslaunch romaa_ros romaa.launch
2. > roslaunch romaa_ros romaa.launch port:=/dev/ttyUSB1

Ejemplos de integración – Herramienta rqt

Aplicación de interfaz gráfica `rqt`



Conclusiones y trabajo futuro

Conclusiones

- Se presentó el desarrollo de un nodo driver para el robot móvil RoMAA a fin de poder integrar dicho robot de diseño propio al ecosistema ROS.

Conclusiones y trabajo futuro

Conclusiones

- Se presentó el desarrollo de un nodo driver para el robot móvil RoMAA a fin de poder integrar dicho robot de diseño propio al ecosistema ROS.
- Se describió en detalle la programación de dicho nodo y los tipos de comunicación utilizados.

Conclusiones y trabajo futuro

Conclusiones

- Se presentó el desarrollo de un nodo driver para el robot móvil RoMAA a fin de poder integrar dicho robot de diseño propio al ecosistema ROS.
- Se describió en detalle la programación de dicho nodo y los tipos de comunicación utilizados.
- Los ejemplos de integración del robot RoMAA con ROS muestra la flexibilidad que brinda ROS de generar diferentes configuraciones para llevar a cabo los diferentes experimentos.

Conclusiones y trabajo futuro

Conclusiones

- Se presentó el desarrollo de un nodo driver para el robot móvil RoMAA a fin de poder integrar dicho robot de diseño propio al ecosistema ROS.
- Se describió en detalle la programación de dicho nodo y los tipos de comunicación utilizados.
- Los ejemplos de integración del robot RoMAA con ROS muestra la flexibilidad que brinda ROS de generar diferentes configuraciones para llevar a cabo los diferentes experimentos.

Trabajo futuro

- Ampliar las posibilidades del nodo driver para incluir el ajuste de los parámetros del controlador PID de los motores

Conclusiones y trabajo futuro

Conclusiones

- Se presentó el desarrollo de un nodo driver para el robot móvil RoMAA a fin de poder integrar dicho robot de diseño propio al ecosistema ROS.
- Se describió en detalle la programación de dicho nodo y los tipos de comunicación utilizados.
- Los ejemplos de integración del robot RoMAA con ROS muestra la flexibilidad que brinda ROS de generar diferentes configuraciones para llevar a cabo los diferentes experimentos.

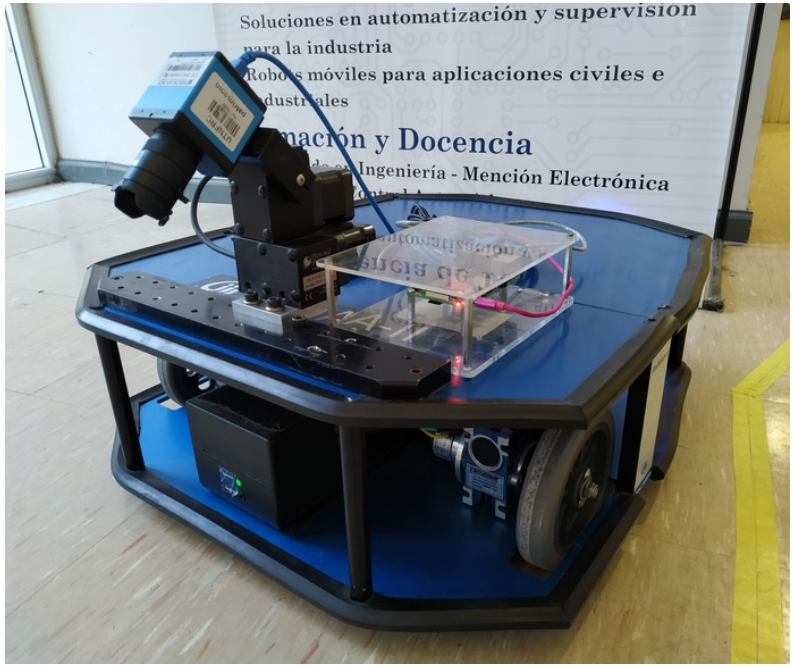
Trabajo futuro

- Ampliar las posibilidades del nodo driver para incluir el ajuste de los parámetros del controlador PID de los motores
- Desarrollar los modelos de visualización del robot para utilizar la herramienta RViz y los modelos de simulación para el simulador dinámico Gazebo.

Driver de ROS para el robot móvil RoMAA

Gonzalo Perez-Paina, David Gaydou, Gastón Araguás

Gonzalo F. Perez Paina
gperez@frc.utn.edu.ar



Gracias por su atención