***** Archivo README ROS*****

Johan Castro

Richard García

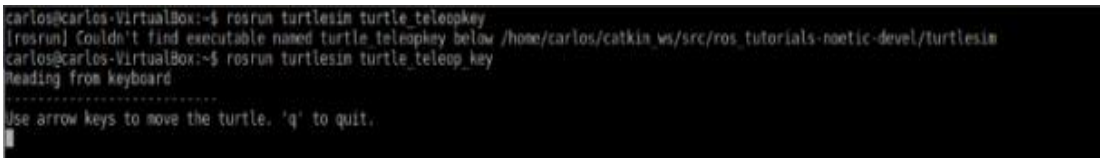
Carlos Tibaduiza

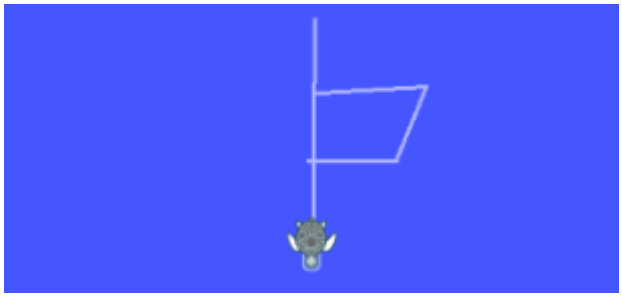
Este repositorio fue creado con la ayuda de @herohernan

En el actual archivo se explica generalmente el que y el cómo utilizar los paquetes o ejercicios realizados durante el primer semestre del 2022 en la materia de ROS, mayormente se utilizó el lenguaje de programación de python, y en algunas ocasiones se probaron los códigos en c++, pero eso fue a decisión de cada grupo. Tambien se tuvo en cuenta la página oficial de ROS, para guiarse en cada uno de los ejercicios. Se recomienda utilizar el ROS NOETIC(UBUNTU 20.04), para evitar daños en maquina virtual o algunos otros problemas de actualizaciones. Se explicará según la información que se pudo subir a Github debido a que se perdieron muchos datos por daños en máquinas virtuales.

****Primer Corte****

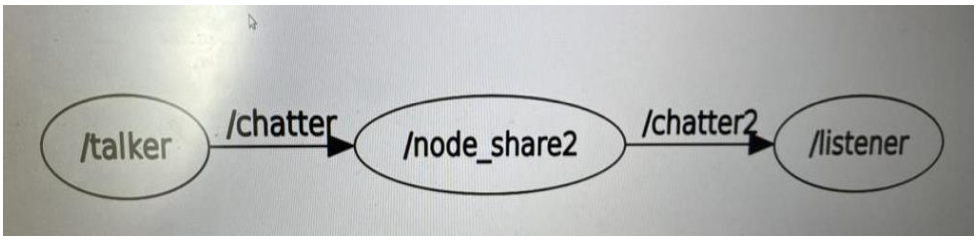
- **Ros-tutorials-noetic-devel:** En este paquete esta básicamente todo lo relacionado con la instalación de ros, se siguió el paso a paso de la página oficial de Ros, en este caso en el grupo se probó tanto el ros kinetic, como el noetic, al final cada integrante se quedó con el que le pareció más adecuado y tambien teniendo en cuenta la versión de Ubuntu. Tambien todo lo relacionado con herramientas, y paquetes necesarios para utilizar todo en Ros. Se verifica tambien el uso de rostopic, rosinfo, etc... El topic de turtlesim fue el ejemplo y se verificaba mediante el movimiento de la tortuga en una terminal con las flechas.



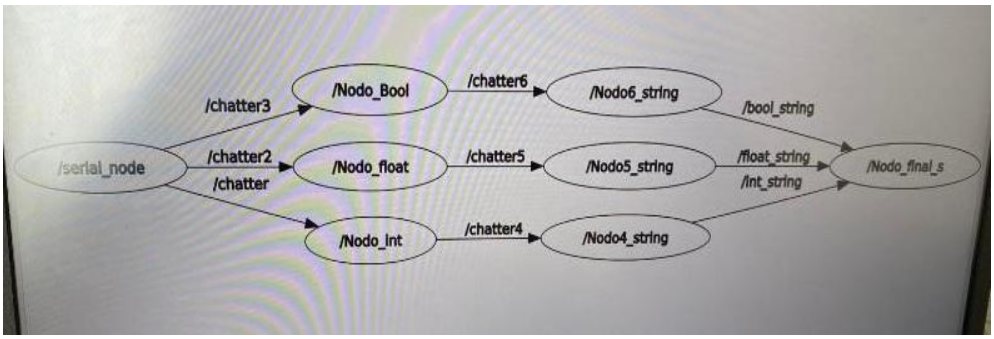


****Segundo Corte****

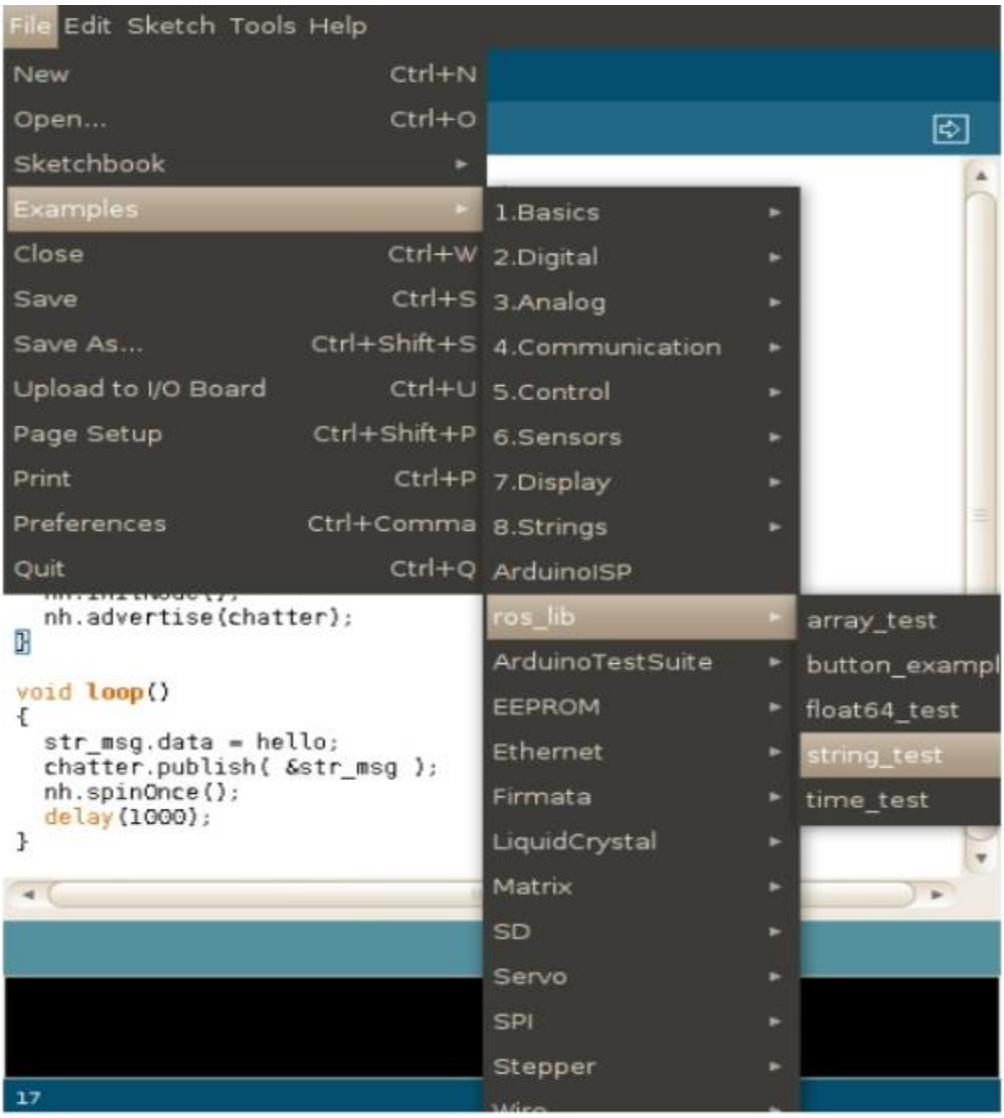
- **Begginer Tutorials:** En esta carpeta se encuentran los ejercicios que inicialmente fueron de prueba cuando se comenzó a ver nodos, topics, donde simplemente hay un nodo Publisher y un nodo Suscriber, en este caso esta en python. El nodo publisher, básicamente publica un dato o una cadena de dato en cualquier tipo, en este caso se envia un entero, mediante un topic y lo recibe cualquier otro nodo con el mismo topico. El nodo Suscriber recibe informacion mediante un topico de un publisher, guarda esa informacion en una funcion y la muestra si lo desea el usuario.



- **Ros Arduino:** Basado en la carpeta anterior, este paquete contiene todos los nodos que se utilizaron para el parcial del segundo corte. Inicialmente se recibe información de 3 tipos de datos(Int,float,bool) en nodos que son publisher y Subscriber a la vez, seguidamente los nodos 4,5,6 reciben esa información, y según la clasificación de los datos que se manejaron, por ejemplo en el dato del Int(despacio, normal o rápido) y según que tipo de dato es en los rangos que se manejaron, sale un dato String, finalmente esos 3 nodos(nodos 4,5 y 6) se reciben en un nodo Final, que recibe tambien 3 datos String, se intercambian las variables segun el nombre del dato que se ingreso y se publica en una minitabla las 3 variables finales. Cabe resaltar que se hizo uso del paquete Rosserial y arduino para la comunicación de Ros y arduino.

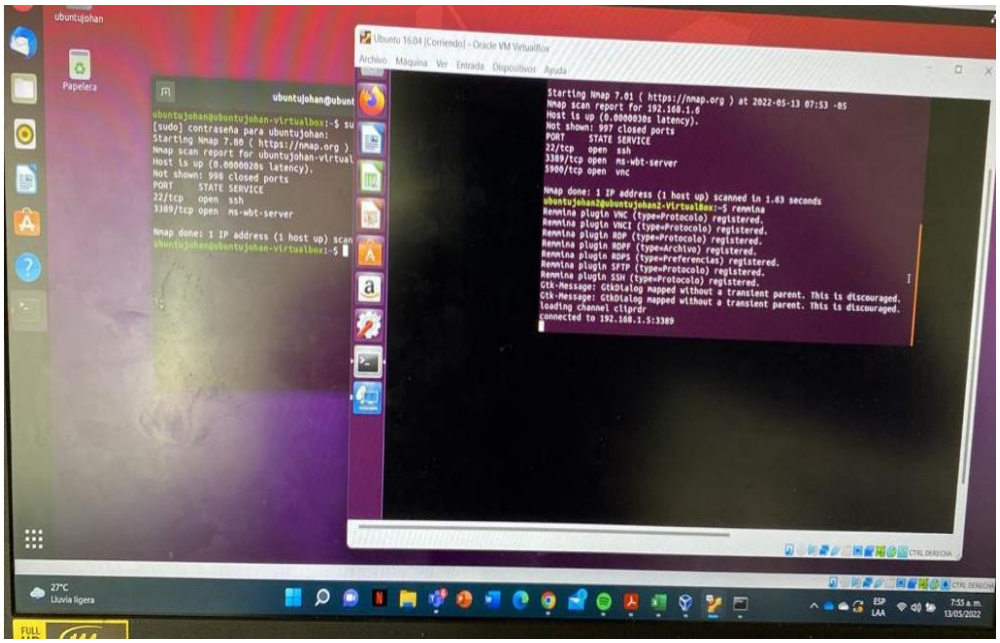


- **Rosserial:** Se descargo este paquete para poder usar ROS directamente con el IDE de Arduino. Rosserial nos proporciona básicamente un protocolo de comunicación ROS que funciona sobre el UART de Arduino. Permite que nuestro arduino sea un nodo ROS completo que puede publicar y suscribirse directamente a mensajes ROS, publicar transformaciones TF y obtener la hora del sistema R, entre otras múltiples herramientas que se pueden utilizar en este paquete. Nuestro uso fue principalmente para la comunicación entre Ros y arduino, y para el parcial del tercer corte de Opencv.



****Tercer Corte****

- **Conexión Remoto Remmina&SSH:** Básicamente se implementó una conexión remoto entre dos máquinas virtuales de Ubuntu, se habilitaron puertos para poder realizar la conexión, en el caso de SSH se habilito el puerto 22, y la conexión fue simplemente de un terminal a otro y con respecto a VSH se habilito el puerto 3389 y en esta parte si se podía interactuar con toda la interfaz obteniendo así un control remoto completo literalmente. Cabe aclarar que se hizo instalación de aplicaciones y tambien algo muy importante la configuración de red, en nuestro caso fue bajo la misma red y con conexión "adaptador puente"



OpenCV:

Del primer punto se instalo el opencv, el paquete de usb node y luego se descargaron los ejemplos de opencv del github.

Para el segundo punto se modificó el código de identificación de círculos de los ejemplos de opencv, en el código se adiciono dos nodos publisher que publican las coordenadas X y Y del centro del circulo

```
class ImageConverter
{
    private:
      ros::NodeHandle nh_;           // Iniciación del nodo

      // Image used
      image_transport::ImageTransport it_;
      image_transport::Subscriber topic1_sub__image_input;
      image_transport::Publisher topic1_pub__image_output;
      ros::Publisher chatter_pub = nh_.advertise<std_msgs::Int16>("ValueX", 100); //el publicador de la coordenada en x
      ros::Publisher chatter_pub2 = nh_.advertise<std_msgs::Int16>("ValueY", 100); //el publicador de la coordenada en y
    public:
```

```
  std_msgs::Int16 msg;
  //Cargamos dicha variable en el .data de la variable Int16 anteriormente creada (la Convertimos a un dato del tipo i
  msg.data = (int)cvRound(circles[i][0]);
  //Publicamos el dato por chatter_pub
  chatter_pub.publish(msg);
  //Realizamos el mismo proceso para la variable en Y
  msg.data = (int)cvRound(circles[i][1]);
  chatter_pub2.publish(msg);
```