Asociación Argentina
de Mecánica Computacional

Mecánica Computacional Vol XXXIV, págs. 3473-3480 (artículo completo)
Sebastián Giusti, Martín Pucheta y Mario Storti (Eds.)
Córdoba, 8-11 Noviembre 2016

ARQUITECTURA DE COMUNICACIÓN PARA SISTEMAS DE TELEOPERACIÓN DE UAV SOBRE INTERNET

Fernando Corteggiano[a], Sebastián Tosco[a], Cristian Benegas[b] y Livio S. Maglione[c]

[a]*Grupo de Investigación y Desarrollo Aplicado a las Telecomunicaciones (GIDAT), Universidad Nacional de Río Cuarto, Ruta Nacional 36 km 601, 5800 Río Cuarto, Cba., Argentina, fcorteggiano@ing.unrc.edu.ar, stosco@ing.unrc.edu.ar, http://www.ing.unrc.edu.ar/gidat*

[b]*benegascristian@gmail.com*

[c]*smaglione@ing.unrc.edu.ar*

Keywords: ROS, middleware, UAV, teleoperación.

Resumen. En la actualidad, las telecomunicaciones han evolucionado permitiendo realizar la operación remota de diversos dispositivos, en especial en situaciones donde hay peligro para la vida humana o cuando resulta imposible el desplazamiento de personal debido a la inaccesibilidad de la zona. Los sistemas de teleoperación de vehículos no tripulados están constituidos por: un operador humano que maneja un dispositivo de telecomando, un vehículo no tripulado, un canal de comunicación entre ellos, y el medio ambiente sobre el cual se desplaza el vehículo no tripulado. El objetivo principal de este trabajo es proponer una arquitectura de comunicación para sistemas de teleoperación de drones en situaciones de búsqueda y recate de personas. Dicha arquitectura de transmisión de datos se basa en el middleware ROS que permite la comunicación basada en mensajes, la incorporación de nuevos participantes de la red y la transmisión en tiempo real. De los resultados experimentales se observa que la arquitectura propuesta es capaz de satisfacer los requerimientos de un sistema de comunicación para la teleoperación de drones en escenarios de búsqueda y rescate.

1 INTRODUCCIÓN

Cuando se plantea una arquitectura de comunicación, se hace necesario un middleware (capa intermedia de software) que facilite la abstracción de los detalles de la red y sus parámetros para centrarse en lo que es relevante desde el punto de vista del diseño y operación (Curry, 2004; McAfferty, 2010). Por tanto, desde el punto de vista del operador humano, el comando remoto de un proceso/sistema no debería ser sustancialmente diferente en comparación con el manejo local del mismo (Callaghan y otros, 2003).

Existen varios tipos de plataformas para este fin (por ejemplo Golberg y Mascha, 1995; Xue, Yang y Meng, 2005; Lum, Rosen y otros, 2009), una de estas son los Middleware Orientados a Mensajes (MOM) (Tosco y Corteggiano, 2014). Éstos proporcionan comunicación distribuida sobre la base del modelo de interacción asíncrona; este modelo sin bloqueo permite resolver muchas de las limitaciones que se encuentran en otras soluciones. Los participantes en un sistema basado en MOM no están obligados a bloquear y esperar a enviar un mensaje, ya que se les permite continuar con el proceso una vez que un mensaje ha sido enviado. Esto posibilita la entrega de mensajes cuando el remitente o el receptor no está activo o disponible para responder en el momento de la ejecución. En los sistemas distribuidos basados en MOM se ofrece una comunicación asíncrona entre procesos orientada a la prestación de servicios.

En el presente trabajo se han realizado pruebas sobre un MOM particular llamado Robot Operating System (ROS) el cual es un meta-sistema operativo de código abierto pensado para robótica que corre con todo su potencial sobre plataformas basadas en UNIX. ROS ofrece los servicios que se esperarían de un sistema operativo, incluida la abstracción de hardware de bajo nivel de control del dispositivo, la aplicación de la funcionalidad de uso común, el paso de mensajes entre los procesos y la gestión de paquetes. También proporciona herramientas y bibliotecas para la obtención, construcción, escritura y ejecución de código en varios equipos (Quigley y otros, 2009).

El objetivo que se persigue en el presente trabajo es analizar el comportamiento de ROS como capa intermedia de un sistema de UAV para situaciones de búsqueda y rescate.

2 INTRODUCCIÓN A LA PLATAFORMA DE COMUNICACIÓN ORIENTADA A MENSAJES: ROS

En el presente trabajo se ha escogido utilizar el Robot Operation System (ROS) dentro de un sistema de teleoperación en el contexto de los laboratorios remotos de nueva generación para evaluar su desempeño. Las razones de esta elección se fundamentan en tres aspectos: las prestaciones de esta arquitectura de software, la calidad de documentación y repositorios, y la buena compatibilidad con el lenguaje de programación Python.

2.1 Aspectos generales

ROS es un meta-sistema operativo de código abierto pensado para robótica que corre sobre plataformas basadas en UNIX. Ofrece los servicios que se esperarían de un sistema operativo, incluida la abstracción de hardware de bajo nivel de control del dispositivo, la aplicación de la funcionalidad de uso común, el paso de mensajes entre los procesos y la gestión de paquetes. También proporciona herramientas y bibliotecas para la obtención, construcción, escritura y ejecución de código en varios equipos (Quigley y otros, 2009). Además, la documentación sobre éste es muy amplia y completa.

La filosofía de ROS se puede resumir en pocos items:

- Punto a Punto
- Basado en herramientas
- Multilenguaje (actualmente son soportados cuatro lenguajes: C++, Python, Octave y LISP.)
- Modular (microkernel con funciones específicas + módulos con librerías y drivers)
- Código abierto (licencia BSD)

2.2 Funcionamiento

En ROS se considera que todo proceso, o parte de un sistema que realiza cálculos, es un *Nodo*. Además hay un *Maestro*, el cual es un servicio de nombres para ROS que ayuda a los nodos a contactar con sus pares y obtener información útil. Entre los nodos puede compartirse información a través de dos paradigmas: *Cliente-Servidor* (esto sería tipo Petición-Respuesta, modelo síncrono, o sea bloqueante, basado en XML-RPC) y *Publicador-Subscriptor*. Ambos paradigmas están muy bien resumidos en la figura 1.

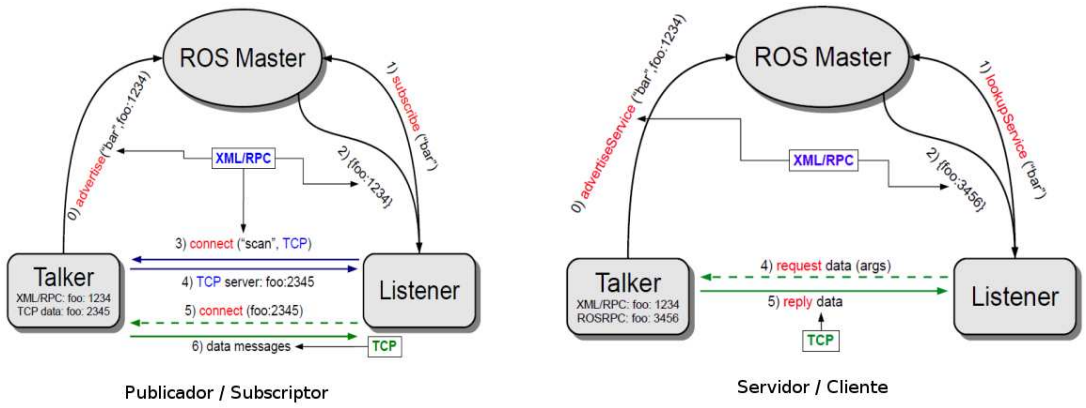

Figura 1: Paradigmas de comunicación en ROS (Ros-Tutorial, 2016)

En el paradigma Publicador-Subscriptor, un nodo publica datos a un canal de comunicación llamado *Tópico* y otro se subscribe al mismo para acceder a los datos. Pueden haber varios publicadores y subscriptores. Es un modelo de funcionamiento flexible (asincrónico). Aquí el maestro actúa sólo como un servicio de nombres almacenando la información de registro de los temas (tópicos) y servicios de los nodos ROS. Los nodos se comunican con éste para reportar su información de registro. A medida que los nodos se comunican con el maestro, pueden recibir información sobre otros nodos inscriptos y hacer las conexiones, según corresponda. También éstos recibirán notificación de los cambios en la información de registro, así pueden crear dinámicamente las conexiones con los nuevos nodos. *Los nodos se conectan a otros nodos directamente, el Maestro sólo proporciona información de búsqueda*.

2.3 Ventajas y desventajas de ROS

Es de interés que se ha realizado una comparación entre varias plataformas de software y se ha discriminado las virtudes y algunos puntos débies de la plataforma escogida.

Entre las virtudes se destacan: orientado a mensajes, multisistema operativo, código libre, permite trabajar en entornos distribuidos, posee simulador y está bien documentado. Entre los puntos débiles se tiene que no posee seguridad implementada (puede salvarse este escollo utilizando conexiones seguras "ssh") y que no es el middleware ideal para aplicaciones exigentes en tiempo real (existen otras como OROCOS diseñadas para tal fin (Elkady y Sobh, 2012)). Sin embargo, se han realizado pruebas con buena tasa de respuesta con tiempos de intergeneración de paquetes cercanos a los 50ms (Tosco y Corteggiano, 2014).

3 ANÁLISIS DEL FUNCIONAMIENTO DEL DRONE (UAV)

Además de conocer la plataforma de comunicación que se utilizó para teleoperar el drone, fue necesario analizar el funcionamiento del mismo. En este caso se utilizó el Ardrone Parrot 2.0 dado que estaba disponible para el estudio.

Este dispositivo tiene un software de control de vuelo brindado por el fabricante el cual no puede modificarse debido a que es de código cerrado. Dicho software implementa un algoritmo de control para corregir la estabilidad del drone en el aire y compensar los efectos del viento. Además, es posible usar un conjunto de librerías que permiten comandar, monitorizar y adquirir flujo de video en tiempo real del drone desde un nodo externo utilizando middleware ROS.

Luego de observarse el comportamiento del drone, se pudieron establecer algunos criterios para la vinculación entre el drone y la arquitectura propuesta.

4 VINCULACIÓN ENTRE EL DRONE Y LA ARQUITECTURA PROPUESTA

Hasta aquí se pudo establecer el funcionamiento del drone y cómo transmitir la información por la red de datos a través del ROS. Se propone utilizar un package de ROS denominado Ardrone Autonomy desarrollado por Mani Monajjemi y otros colaboradores en Autonomy Lab de la Universidad Simon Fraser. Este es un controlador para el quadcopter Ardrone Parrot 1.0 y 2.0 basado en el SDK ArDrone oficial, de la versión 2.0.1. El mismo, contiene un nodo que permite vincular la información recibida del drone por WiFi con ROS a través de la red de datos. Para ello, es necesario ejecutar el nodo ardrone_driver, el cual se encarga de vincular la red de datos con el drone, publicando en mensajes ROS la información brindada por el quadcopter.

Con esto en mente, se puede plantear la arquitectura mostrada en la figura 2. Allí puede observarse un nodo repetidor estacionario (por ejemplo montado en un globo, con la idea que consuma la menor cantidad de energía para lograr una mayor autonomía) el cual repite las señales de los drones voladores desde y hacia una estación terrena de procesamiento. En esta última se halla emplazado tanto el controlador ROS (Ros Master) como todo lo necesario para procesamiento de imágenes y detección autómata de personas en situaciones de búsqueda y rescate.

También en la figura supracitada puede notarse los dos ámbitos de red claramente nomenclados: El ámbito ROS, donde se utiliza una red wlan para teleoperación y el ámbito de Internet, donde pueden publicarse datos de interés para usuarios remotos vía web. Estos dos ámbitos quedan interconectados gracias a un puente ("ROS-Brigde") que permite transportar en ambos sentidos información de esos dos ámbitos.

En el caso del drone utilizado, el puerto de comunicación es Wifi (802.11n). Con lo cual se puede armar una arquitectura con un Ubiquity Rocket M2 como repetidor y un Bullet en 2.4 Ghz en la estación base (ubtn,2016).

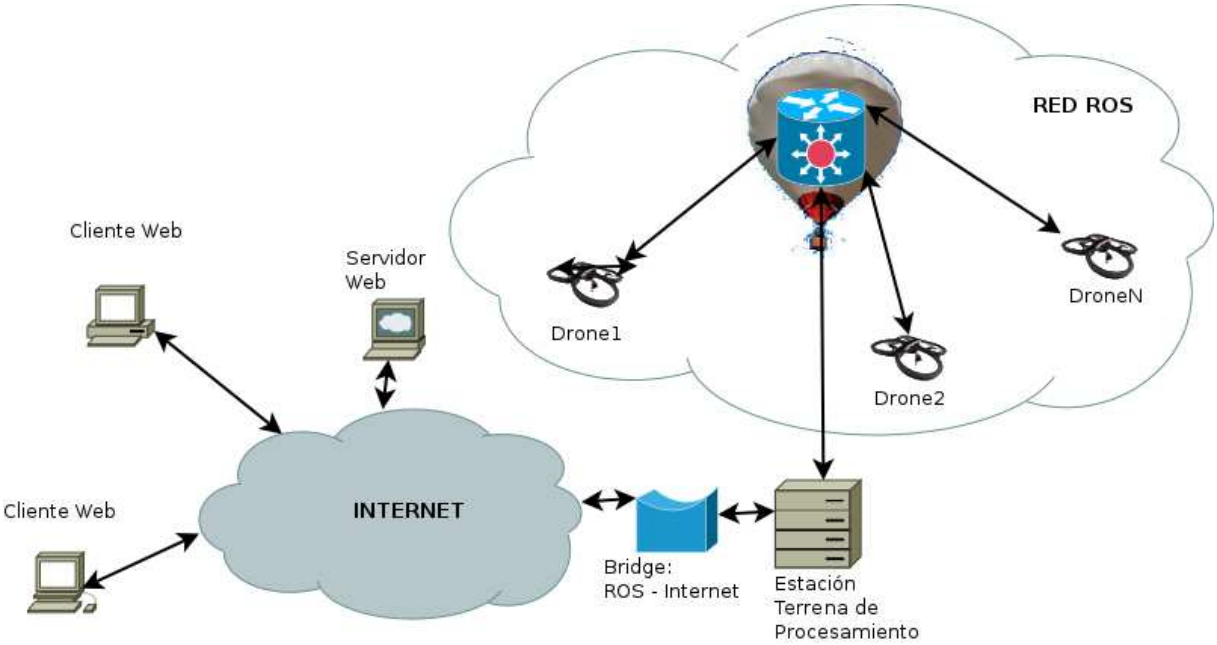

Figura 2: Arquitectura propuesta

5 PRUEBAS REALIZADAS

Se realizaron pruebas de tres aspectos de interés para el presente trabajo:

- Prueba de ROS como arquitectura de comunicaciones
- Prueba de la API de funcionamiento de AR-Drone Parrot y su vinculación con ROS.
- Pruebas relacionadas con la obtención del flujo de video y su procesamiento.

5.1 Funcionamiento de ROS como arquitectura de comunicaciones

Para probar la capacidad de comunicación del middleware ROS se armó una arquitectura para control de un proceso eléctrico con varios nodos que transmitan distintos tipos de información:

- Información para control del proceso
- Información de realimentación de estado del proceso
- Flujo de video.

En las sendas pruebas que se realizaron pudieron establecerse resultados interesantes en cuanto a desempeño y funcionamiento de la arquitectura propuesta. En dichas pruebas se estableció es que ROS funciona de forma aceptable en rangos normales de Delay-Jitter (≤200ms - 100ms) y con una adecuada selección de tasas de muestreo (Ts >40Hz). La tasa de datos total requerida fue de 300kbps (Tosco y Corteggiano, 2014)

5.2 Vinculación del AR-DRONE con la arquitectura de comunicaciones propuesta

Luego de obtener la vinculación del drone con el ROS mediante el package ardrone_autonomy y el SDK oficial de Ardrone, se desarrolló un nodo en Python que se suscribía a la información de los parámetros de vuelo que publica el nodo ardrone_driver en el tema (topic) ardrone/navdata con el tipo de mensaje Navdata; donde se obtenía información como: Nivel de batería, velocidad de los motores, temperatura sensada, altitud, estado del drone (volando, aterrizado, desconocido, etc.), rotación, magnetómetro, entre otros. Este nodo, lanzaba una ventana que mostraba algunos de estos parámetros. Además, dicho nodo permitía comandar el drone utilizando el teclado de la computadora para enviar mensajes ROS al nodo ardrone_driver. Por ejemplo: La barra espaciadora despegaba y aterrizaba el drone; la tecla Escape, realizaba un aterrizaje de emergencia; las flechas deslizaban hacia adelante/atrás/izquierda/derecha; las teclas W, S, A, D deslizaban hacia arriba, abajo, giro a la izquierda y giro a la derecha, respectivamente. Estos comandos ejecutaban la orden por un tiempo establecido por defecto de 0.2 segundo, y luego el drone quedaba en estado "suspendido en el aire". También se podía editar por teclado la potencia con la que se movía el drone utilizando la tecla "+" y "-" que permitía aumentar y disminuir respectivamente este valor.

5.3 Obtención y procesamiento del flujo de video

Aprovechando la potencialidad de ROS, fue posible suscribirse de manera remota para obtener la imagen captada desde la cámara a bordo del drone. Esto permitió realizar un procesamiento digital de la imagen con un procesador potente, específico para tal fin, y así tomar decisiones de vuelo y/u obtener información de interés.

Así, se desarrolló un nodo en el lenguaje de programación Python, el cual se suscribía a la publicación de la cámara a bordo del drone, y determinaba el histograma de la imagen por color RGB. El histograma de la imagen es una herramienta que se utiliza mucho para obtener los niveles de color de una imagen, con fines de posterior ecualización para lograr una mejor apreciación de la misma. Se muestran en la figura 3 los resultados de dicho procesado.

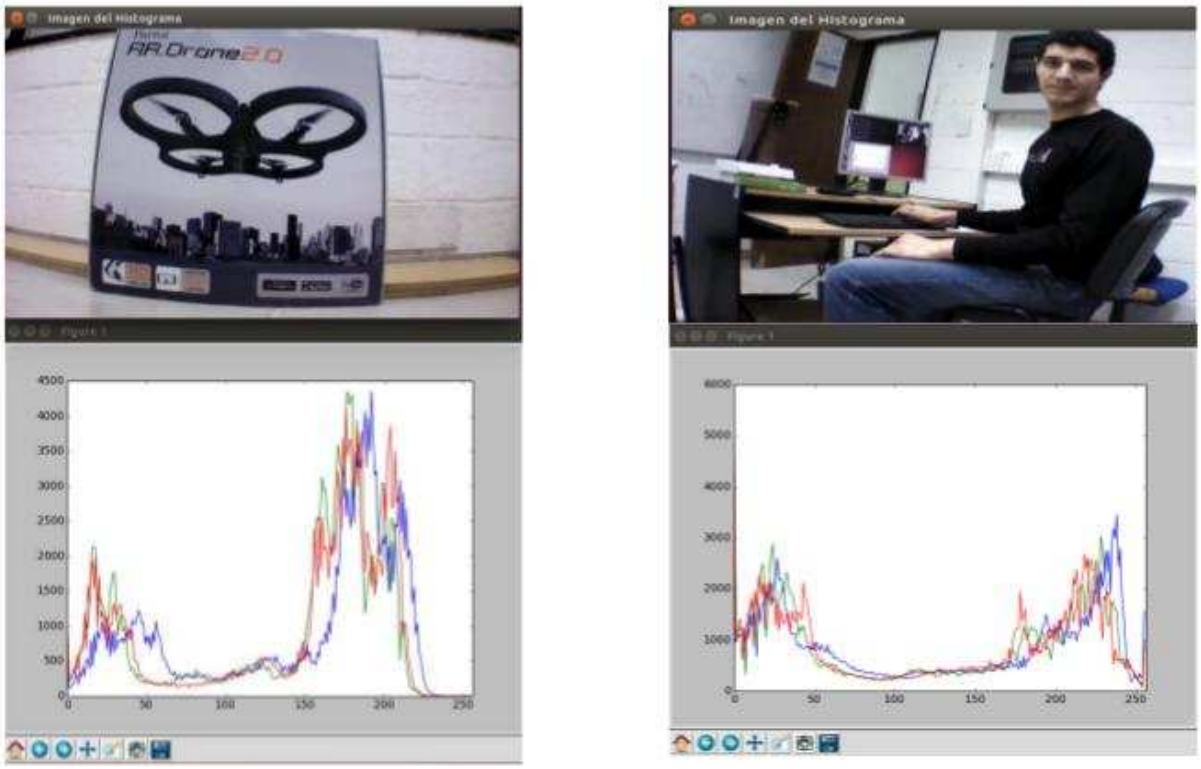

Figura 3: Procesamiento de imagen para obtener histograma

De manera similar, se podría implementar un nodo que realice un procesamiento más complejo para obtener otro tipo de información, como podría ser:

- Detectar la presencia de vehículos, personas y/o animales
- Determinar el índice de verde de un campo y/o su tamaño
- Contar animales u objetos en general
- Armar un plano integrando las imágenes
- Buscar patrones que permitan localizar personas, etc.

Por otro lado, dado que se propone realizar el procesamiento en la estación terrena, es posible escalar la capacidad de procesamiento mediante el armado de un cluster dedicado a tal fin.

6 CONCLUSIONES

Este trabajo se desarrolló dada la motivación inicial de utilizar drones como instrumento para la búsqueda y rescate de personas en situaciones de riesgo, con el aporte del tratamiento digital de la imagen obtenida de la cámara a bordo, obteniéndose buenos resultados. Entre ellos, se logró demostrar que la utilización de la plataforma ROS como herramienta para organizar la comunicación entre diferentes computadoras y uno o varios drones, es muy potente. Ésta permite compartir en la red de datos (Internet) la información tanto de los parámetros de vuelo para comandar el UAV, como la visualización de la/s cámara/s a bordo con el fin de aplicar los algoritmos que permitan la detección de los objetos de interés. Además, la posibilidad de dividir la carga del procesamiento en múltiples computadoras, permite implementar algoritmos más complejos y robustos debido a que no se limita el cálculo computacional en un solo procesador.

Finalmente, con respecto al drone, cabe recordar que se utilizó el Ardrone Parrot 2.0 debido a que se encontraba disponible al momento de realizar este trabajo, pero en realidad no está diseñado para usos en campos bajo situaciones de búsqueda y rescate. Por tal motivo, es recomendable seleccionar un drone más robusto y con mayor capacidad de carga y autonomía. En cada caso habrá que idear una vinculación (hardware y software) entre ROS y el drone seleccionado.

BIBLIOGRAFÍA

Callaghan MJ., J. Harkin J., Prasad G., MeGinnity TM., Maguire LP., Integrated Architecture for Remote Experimentation, *IEEE International Conference on System Man and Cybernetics*, 2003.

Curry, E., "Message-Oriented Middleware" in "Middleware for Communications." Edited by Mahmoud, Qusay, ISBN 0-470-86206-8, *Ed.: John Wiley & Sons* Ltd., 2004.

Elkady, A., Sobh, T., Robotics Middleware: A Comprehensive Literature Survey and Attribute-Based Bibliography, Journal of Robotics Volume 2012 (2012), Article ID 959013, 15 pages, 2012.

Golberg, K., Mascha, M., Desktop Teleoperation via the World Wide Web," Proceedings of the 1995 IEEE International Conference on Robotics and Automation, May 1995.

Lum, M.J.H., Rosen, J., King, H., Friedman, D.C.W., Lendvay, T. S., Wright, A.S., Sinanan, M.N., Hannaford, B., Teleoperation in Surgical Robotics – Network Latency Effects on Surgical Performance, 31st Annual International Conference of the IEEE EMBS, USA, 2009.

Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler,. R., Ng, A., ROS: an open-source Robot Operating System, Proc. Open-Source Software workshop of the International Conference on Robotics and Automation (ICRA), 2009.

McCafferty, B., Message-Based Systems for Maintainable, Asynchronous Development, *Blog on devlicio.us*, 2010.

Tosco S.J., Corteggiano F., Evalución de una plataforma basada en mensajes para la operación remota de procesos. Mecánica Computacional. San Carlos de Bariloche: Graciela Bertolino, Mariano Cantero, Mario Storti y Federico Teruel (Eds)., 33:2595–2618, 23-26 Setiembre 2014. issn 1666-6070.

Xue, X., Yang, S. X., Meng, M.Q.-H., Remote Sensing and Teleoperation of a Mobile Robot via the Internet, *Proceedings of the 2005 IEEE International Conference on Information Acquisition*, China, 2005.

https://courses.cs.washington.edu/courses/cse466/12au/calendar/16f-ros_tutorial.pdf (Consultado: 09/2016)

https://www.ubnt.com (Consultado: 09/2016)