Universidad San Francisco de Quito

Colegio de Ciencias de Ingeniería

Desarrollo de un Vehículo Autónomo

Fase 2: Sincronización del sistema de adquisición de video con el sistema de georreferencia vehicular usando ROS

Alex Hernán Díaz Conterón

Ingeniería Electrónica

Trabajo fin de carrera para la presentado como requisito
para la obtención del título de
Ingeniero Electrónico

Quito, 27 de julio de 2020

UNIVERSIDAD SAN FRANCISCO DE QUITO

Colegio de Ciencias e Ingenierías

HOJA DE CALIFICACIÓN
DE TRABAJO DE FIN DE CARRERA

Desarrollo de un Vehículo Autónomo

Fase 2: Sincronización del sistema de adquisición de video con el sistema de georreferencia vehicular usando ROS

Alex Hernán Díaz Conterón

Nombre del profesor, Título académico **René Játiva Espinoza, PhD**

Quito, 27 de julio de 2020

DERECHOS DE AUTOR

Por medio del presente documento certifico que he leído todas las Políticas y Manuales de la Universidad San Francisco de Quito USFQ, incluyendo la Política de Propiedad Intelectual USFQ, y estoy de acuerdo con su contenido, por lo que los derechos de propiedad intelectual del presente trabajo quedan sujetos a lo dispuesto en esas Políticas.

Asimismo, autorizo a la USFQ para que realice la digitalización y publicación de este trabajo en el repositorio virtual, de conformidad a lo dispuesto en el Art. 144 de la Ley Orgánica de Educación Superior.

Nombres y Apellidos: Alex Hernan Diaz Conteron

Código: 00109864

Cedula de identidad: 1003412663

Lugar y fecha: Quito, 27 de julio de 2020

ACLARACIÓN PARA PUBLICACIÓN

Nota: El presente trabajo, en su totalidad o cualquiera de sus partes, no debe ser considerado como una publicación, incluso a pesar de estar disponible sin restricciones a través de un repositorio institucional. Esta declaración se alinea con las prácticas y recomendaciones presentadas por el Committee on Publication Ethics COPE descritas por Barbour et al. (2017) Discussion document on best practice for issues around theses publishing, disponible en http://bit.ly/COPETheses.

UNPUBLISHED DOCUMENT

Note: The following capstone project is available through Universidad San Francisco de Quito USFQ institutional repository. Nonetheless, this project – in whole or in part – should not be considered a publication. This statement follows the recommendations presented by the Committee on Publication Ethics COPE described by Barbour et al. (2017) Discussion document on best practice for issues around theses publishing available on http://bit.ly/COPETheses.

RESUMEN

En la actualidad los sistemas autónomos han ganado importancia en las plataformas tecnológicas. Las aplicaciones robóticas y la informática han logrado un avance significativo en la integración de robots de diferentes tipos, dando lugar a la nube robótica. El Sistema Operativo de Robots (ROS) otorga una mayor accesibilidad de funciones robóticas para una aplicación educativa e industrial. El presente proyecto es la continuación del desarrollo del Vehículo Autónomo de la Fase 1. De tal forma, se sigue con el objetivo de dotar capacidades independientes que permita una libre exploración.

Este documento se enfoca principalmente en la visualización del entorno de exploración del Vehículo Autónomo y su georreferenciación. Esta reseña describe el procedimiento para implementar una cámara web USB y un GPS al Raspberry Pi 3B. El sistema de visualización en esta fase del proyecto permite al usuario observar el entorno de exploración en una simulación sobre RVIZ.

Esta implementación se realizó usando los repositorios, bibliotecas y herramientas de ROS. Con la ayuda de paquetes de ROS y scripts personalizados se proporciona un procedimiento detallado de la funcionalidad de la Webcam y el GPS.

Palabras claves: Vehículo autónomo, ROS, visualización y seguimiento, GPS, RViz, RaspBerry Pi 3B

ABSTRACT

Currently, autonomous systems have gained importance in technological platforms. Robotic applications and computing have made significant progress in integrating robots of different types, giving rise to the robotic cloud. The Robot Operating System (ROS) provides greater accessibility to robotic functions for educational and industrial applications. This project extends the development of the Autonomous Vehicle performed at Phase 1. Thus, it stands with the aim of providing independent capabilities that allow free exploration.

This document mainly focuses on the visualization of the Autonomous Vehicle exploration environment and its geo-positioning. This extended summary describes the procedure to integrate an USB Webcam and a GPS to a Raspberry 3B. Visualization system at this phase allows the user to observe the exploration environment on a RViz simulation.

This implementation was performed using the ROS repositories, libraries, and tools. A detailed procedure of Webcam and GPS functionality is provided with the help of ROS packages and custom scripts.

Key words: Autonomous vehicle, ROS, visualization and tracking, GPS, RViz, RaspBerry Pi 3B.

CONTENIDO

Introducción 9
1. DESARROLLO DEL TEMA 10
1.1 Tecnologías emergentes 10
1.2 Introducción a ROS 12
1.2.1 Robot Operating System (ROS) 12
1.2.2 Conceptos y comandos de ROS 13
1.3 Descripción de Community Level 14
1.4 Distribución de ROS 14
1.5 Herramientas de ROS 15
1.5.1 RVIZ 15
1.5.3 IMAGEN_VIEW 15
1.6 PUPNUB 16
2 Microcontroladores y dispositivos para ROS 16
2.2 Raspberry Pi 3B 16
2.3 GPS Ublox NEO-6M 17
2.4 Cámara web GENIUS 18
3 Instalación 19
3.1 Ubuntu Mate en Raspberry Pi 19
3.2 Instalación de ROS Kinetic en Raspberry 19
4 Implementación 20
4.1 Cámara Web 20
4.2 GPS 26
4.2.1 Configuraciones del software del GPS 26
4.2.2 Página web de seguimiento de geolocalización a tiempo real 29
5. Simulación de la cámara web con el sistema de georreferencia vehicular usando ROS 31
Conclusiones 32
REFERENCIAS BIBLIOGRAFICAS 34
ANEXOS A: Código de la página web 35
ANEXO B: Código de Python para enviar datos a la página web 38

ÍNDICE DE FIGURAS

Figura 1.- Ciclo de Sobre expectación de Gartner 11
Figura 2.- Ciclo de Sobre expectación de Gartner para Tecnologías Emergentes 2019 11
Figura 3.- Comunicación de ROS 14
Figura 5.- Raspberry Pi 3B 17
Figura 6.- GPS Ublox NEO-6M 18
Figura 7.- Cámara Web Genius 18
Figura 8.- verificación de dispositivo 20
Figura 9.- Especificaciones de la webcam 20
Figura 10.- Roscore 21
Figura 11.- Implementación de la webcam en Imagen view 22
Figura 12.- Rviz 22
Figura 13.- Opciones para agregar a dispositivo 23
Figura 14.- Agrega topic image_raw 23
Figura 15.- Implementación de la webcam en Rviz 24
Figura 16.- Topics activos 24
Figura 17.- Característica de la cámara 25
Figura 18.- Nodos creados 25
Figura 19.- Verificación de nodos, mensajes y topic creados 26
Figura 20.- Conexión de Raspberry y modulo GPS Neo 6M 26
Figura 21.- Obtención de datos del módulo GPS 28
Figura 22.- Obtención de datos modificados del GPS 29
Figura 23.- Clave del publicador y suscriptor de PubNub 30
Figura 24.- Pagina web con rastreo GPS 30
Figura 25.- Simulación de la Webcam y el GPS 32

Introducción

La robótica en los últimos años ha evolucionado de una forma considerable dentro de los campos de la educación y la industria. El robot es una máquina controlada por un ordenador y que ejecuta automáticamente funciones complejas conforme a su programación el campo de ejecución. Los avances tecnológicos y el crecimiento en la complejidad de estos sistemas y sus aplicaciones han dado lugar a entornos de trabajo específicos para la robótica, es decir para el desarrollo de sistemas y equipos capaces de realizar actividades repetitivas de forma eficiente cubriendo exigencias autónomas. Existen varios entornos de robótica como, por ejemplo: Carmen, Player, Yarp, Oroscos, etc. Sin embargo, ROS se diferencia de los demás por ser más robusto y soportar plataformas robóticas más complejas. ROS es un entorno para el desarrollo de algoritmos de robótica de libre distribución de código abierto. Esta plataforma es el soporte principal para la implementación del sistema de visualización y su georreferenciación en el presente proyecto.

1. DESARROLLO DEL TEMA

1.1 Tecnologías emergentes

Los avances tecnológicos se encuentran en constante evolución con la intención de solucionar problemas relacionados con el espacio global. Sin embargo, las soluciones propuestas en muchos casos no son las óptimas o son incapaces de seguir la dinámica de la complejidad social, científica y académica, tornándose en obsoletas.

El Ciclo de Sobre expectación de Gartner muestra los niveles de expectativa que tiene un nuevo producto difundido al mercado. Es una representación gráfica de la madurez y la adopción de tecnologías y aplicaciones, con la intención de resolver problemas reales. En la Figura 1 indica la evolución de la tecnología a través del tiempo, donde se observa un enfoque panorámico de la tecnología emergente.

El ciclo de sobre expectación tiene 5 fases: Innovation Trigger (Disparador de Innovación), en el cual se detecta un avance tecnológico que se difunde sin disponer de productos utilizables; Peak of Inflated Expectations (Pico de sobreexpectación), fase en la que se cautiva sustancialmente el interés del público. Trough of Disillusionment (Curva de Desilusión), período en el cual se pierde interés a medida que los experimentos realizados no muestran desarrollos factibles. Slope of Enlightenment (Pendiente de Iluminación), fase en la cual la tecnología se vuelve consistente con la intención ser implementada. Plateau of Productivity (Escenario de Productividad), el producto se consolida ante las pruebas para su producción.

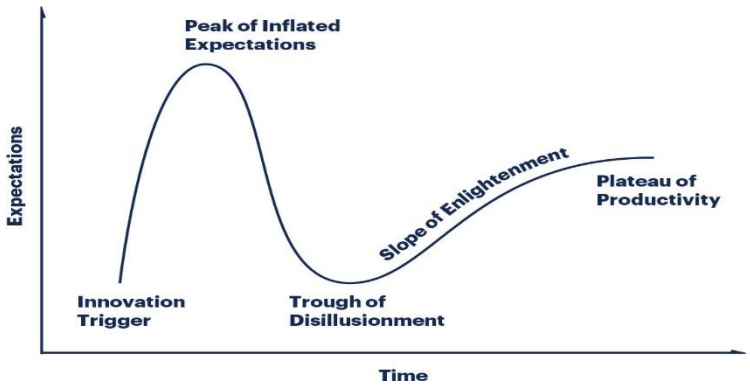

Figura 1.- Ciclo de Sobre expectación de Gartner

El Ciclo de Sobre expectación de Gartner para Tecnologías emergentes, muestra un amplio abanico de tecnologías que están siendo analizadas. En la actualidad, los robots ocupan un interés por la comunidad académica y empresarial. Según Gartner la tendencia número 1 en tecnologías emergentes son los Robots Autónomos que estarán vigentes por más de 10 años. Son capaces de actuar con un sistema de programación en un mundo real. Los ambientes de exploración de un robot autónomo son: el mar, tierra, aire y digital, que operan con capacidades autónomas. Entonces, como factor motivacional el tema de este proyecto contempla una amplia aceptación académica para su estudio e investigación.

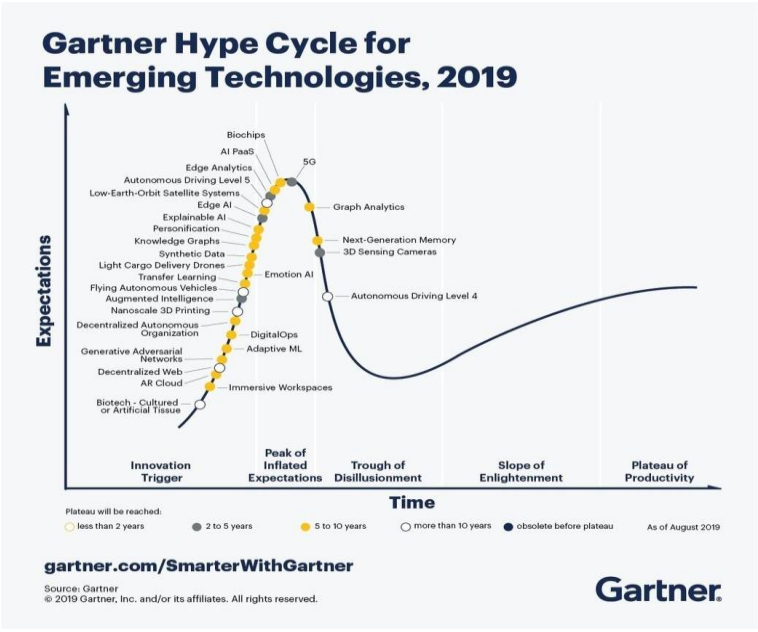

Figura 2.- Ciclo de Sobre expectación de Gartner para Tecnologías Emergentes 2019

1.2 Introducción a ROS

1.2.1 **Robot Operating System (ROS)**

En el laboratorio de Inteligencia Artificial de Stanford se dio inicio a los primeros entornos robóticos. ROS que fue desarrollado por el Dr. Morgan Quigley. Él desarrolló el software de Open Robotics, siendo esta base principal para dar lugar a ROS. En el 2007 la empresa Willow Garage logró el desarrollo de ROS, bajo la licencia de Berkeley Software Distribution (BSD) y Apache 2.0, que permite a los usuarios modificar, reutilizar y redistribuir todo el material disponible dentro de ROS. (Quigley,2009)

ROS es un sistema meta operativo de código abierto que ofrece grandes ventajas de acceso a una comunidad de usuarios, bibliotecas y paquetes, para desarrollar el control de dispositivos. ROS intenta unificar funciones autónomas y facilitar el desarrollo de los robots. ROS es versátil para los usuarios que ofrece la construcción y la ejecución de códigos creados por otros usuarios, que pueden ser modificados de acuerdo con los requerimientos que se desee.

Una de las características principales de ROS su programa es reutilizable, el usuario tiene el fácil acceso al código y puede modificarlo, para luego difundirlo a la comunidad robótica. En términos de comunicación, ROS se destaca por proporcionar un servicio para la implementación de controladores, actuadores y sensores. ROS tiene herramientas para la visualización del estado del robot y el proceso del algoritmo. ROS tiene una comunidad robótica activa en todo el mundo, y más de 5000 paquetes se han desarrollado de forma voluntaria. También tiene la capacidad de otorgar ayuda especializada por medio de su página de preguntas y respuestas.

1.2.2 **Conceptos y comandos de ROS**

- Tópico (topic): es el canal de comunicación a través de los cuales los nodos pueden intercambiar mensaje. La comunicación semántica de publicación / suscriptor, el nodo envía un mensaje publicador en un tema determinado. Además, al tópico se puede nombrar para identificar el mensaje. El tópico tiene la facilidad de comunicarse con varios suscriptores para un solo tema.
- Paquete (Package): es la estructura y contenido necesario para crear un programa en ROS. Dentro de un paquete tiene un código diseñados para compartir y descargarse para su uso de forma gratuita. Estos códigos son desarrollados por programadores de todo el mundo con intención de aplicarlos.
- Nodo(node): el nodo es un proceso diseñado de comunicación capaz de realizar cálculos. El nodo se crea en la biblioteca cliente de ROS descrita con diferentes lenguajes de programación como Cpp o Python.
- Mensajes (msg): es una estructura de datos que comprende campos escritos por lo que el nodo envía un mensaje al publicar en el topic determinado.
- Servicios (srv): es el medio donde realiza la función de comunicación entre el publicador y el suscriptor por medio de mensajes para la solicitud y respuesta de un nodo proveedor.

- rospack: comando para obtener información de los paquetes.
- rosdep: comando para instalar dependencias del paquete.
- rosmake: comando para compilar el paquete.
- roscd: comando para abrir directorio
- rosnode list: indica los nodos activos.
- rosnode info: indica la información del nodo.

- roscore: activa el nodo principal para la ejecución de los nodos creados.
- rosrun: inicializa los nodos creados.

En la siguiente figura se muestra el proceso de ejecución de los nodos en ROS, donde el nodo 1 está publicando un mensaje sobre el topic y el nodo 2 se suscribe a este topic para obtener el mensaje.

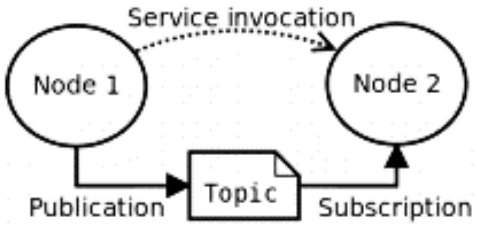

Figura 3.- Comunicación de ROS

1.3 Descripción de Community Level

- Distribuciones: son versiones de colecciones de meta-paquetes que se pueden instalar.
- Repositorios: es un medio donde instituciones o programadores aportan su desarrollo a la comunidad.
- Ros Wiki: es un medio donde los usuarios pueden realizar preguntas para una asistencia personalizada que resuelven los problemas de forma oportuna.

1.4 Distribución de ROS

Para este proyecto se realizó la instalación de ROS Kinetic que fue lanzado el 23 de mayo de 2016 y es compatible con el sistema operativo Ubuntu 16.04.

Figura 4.- Logo de la distribución de ROS Kinetic

1.5 Herramientas de ROS

Las herramientas de ROS son útiles para el usuario, muchos de ellos fueron desarrollados de forma voluntaria para luego ser aplicada por la comunidad robótica, a continuación, se observa las herramientas que se utilizó en este proyecto.

1.5.1 RVIZ

Es una herramienta de visualización en 3D que controla las funciones de ROS mostrando imágenes de la cámara montada en el robot, tiene funciones compatibles con sensores como Kinect, RealSense y sensores de distancia para estimar la posición y punto de navegación del robot. Rviz tiene la función de visualizar mapas, objetos y robots en un campo virtual indicando los movimientos que ocurre en el mundo real o físico.

1.5.2 ROS GUI (RQT_GRAPH)

ROS tiene unas herramientas GUI para el desarrollo de los robots. Es una herramienta que viene integrada cuando se instala ROS, esta herramienta se ejecuta con la línea de comando “rosrun”. Se encarga de visualizar los nodos y mensajes activos, también indica la estructura actual de los nodos que se están ejecutando en el robot.

1.5.3 IMAGEN_VIEW

Tiene la función de visualizar los datos de la imagen de una cámara es bastante útil para verificar el entorno que se encuentra el robot mientras se mueve.

1.6 PUPNUB

Es un servicio publicador/ suscriptor para enviar datos entre dispositivos en tiempo real. La utilidad de esta plataforma es amplia, incluye la construcción de aplicación de chat, señalización de dispositivos IoT (Internet of Things), seguimiento de ubicación GPS en tiempo real y para desarrolladores de juegos en línea. PubNub dispone una versión gratuita para su uso.

2 Microcontroladores y dispositivos para ROS

2.2 Raspberry Pi 3B

ROS es compatible con microcontroladores, pero no todos ellos son compatibles. Para el desarrollo de este proyecto se optó por el modelo Raspberry Pi 3B.

Características de Raspberry Pi 3B

- Dimensiones: 85 x 56 x 17mm
- Ranura de tarjeta microSD 16GB
- Antena de chip
- 40 pines GPIO
- 4 puertos USB 2.0
- 1 conector de cámara CSI
- 1 conector de video/audio RCA
- 1 entrada HDMI
- 1 puerto Ethernet

- Bluetooth 4.1
- Chipset Broadcom BCM2837
- 1 micro USB power input 2.5Amps.
- Procesador Quad Core ARM Cortex-A53 de 64 bits a 1,2 Ghz
- 1Gb RAM

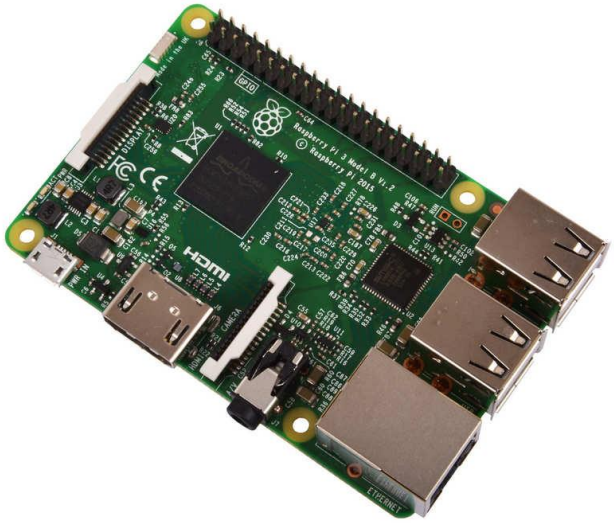

Figura 5.- Raspberry Pi 3B

2.3 GPS Ublox NEO-6M

Es un módulo de posicionamiento geo referencial, equipado con una antena y EEPROM. La comunicación del módulo se realiza por un puerto serial. Dispone de 4 pines VCC, RX, TX y GND. Los datos se obtienen en formato NMEA (National Marine Electronics Association).

Características GPS

- Modelo GY-GPS6M V2
- EEPROM para el registro de datos
- LED indicador de señal

- Velocidad de transmisión estándar 9600bps
- Voltaje de alimentación 3.0 a 5.0 Volts
- Dimensiones: 25mm x 35mm

Figura 6.- GPS Ublox NEO-6M

2.4 Cámara web GENIUS

La cámara web tiene las siguientes características:

- Enfoque manual
- Formato MPEG
- Micrófono
- Dimensiones: 150x49x48 mm
- Puerto USB

Figura 7.- Cámara Web Genius

3 Instalación

3.1 Ubuntu Mate en Raspberry Pi

Raspberry Pi tiene compatibilidad con varios sistemas operativos. ROS.org recomienda la instalación de la distribución de Kinetic debido a su compatibilidad con la versión Ubuntu 16.04 para la instalación en el Raspberry. Para la instalación se sigue los pasos de indicados por Ubuntu_Mate.org.

Primero en la página de Ubuntu en la sección de descargas se obtiene el sistema operativo para grabar la imagen en la tarjeta SD. Se abre el terminal y se ejecuta el siguiente comando:

```
sudo apt-get install gddrescue xz-utils
```

se descomprime el archivo que se descargó

```
unxz ubuntu-16.04.2-desktop-armhf-raspberry-pi.img.xz
```

se conecta la tarjeta SD al ordenador y se graba la imagen del sistema

```
sudo ddrescue -D --force ubuntu-mate-16.04.2-desktop-armhf-raspberry-pi.img /dev/sdx
```

finalmente, se coloca la tarjeta SD en la Raspberry y listo para el desarrollo del proyecto.

3.2 Instalación de ROS Kinetic en Raspberry

Para la instalación de ROS Kinetic se sigue todos los pasos detallados en la página web de ROS.org tal como se realizó en la Fase 1 de proyecto de vehículo autónomo.

4 Implementación

4.1 Cámara Web

En la página web de ROS.org se realiza una búsqueda del paquete USB_CAM, para la instalación se verifica que sea para la distribución de Kinetic. Para la implementación se conecta la cámara web en el puerto USB de la Raspberry, con el siguiente código de línea se verifica si el dispositivo está conectado.

```
$ lsusb
```

Se verifica que está conectado en el bus 001 la webcam

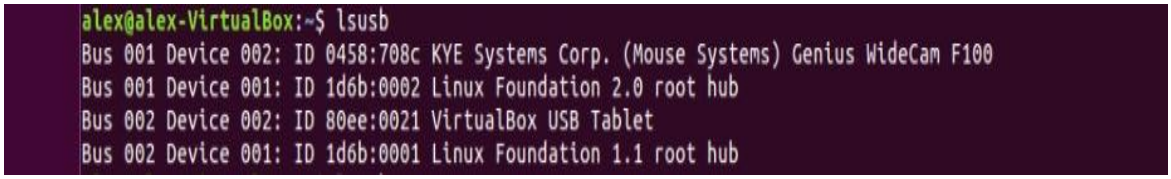

Figura 8.- verificación de dispositivo

Las especificaciones de la cámara son determinadas con

```
$ v4l2-ctl -d /dev/video0
```

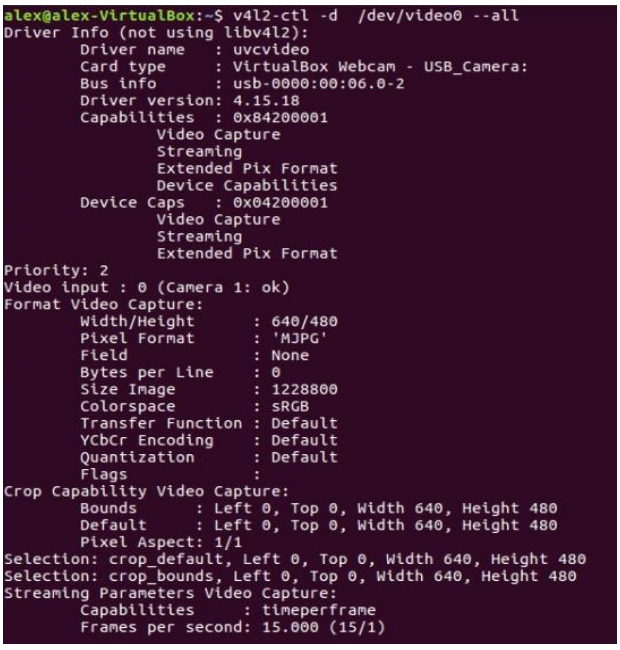

Figura 9.- Especificaciones de la webcam

Se instala el paquete USB-CAM de ROS

```
$ sudo apt-get install ros-kinetic-usb-cam
```

Se instala la herramienta de visualización de ROS

```
$ sudo apt-get install ros-kinetic-image-view
```

En otro terminal, se ejecuta el nodo master

```
$ roscore
```

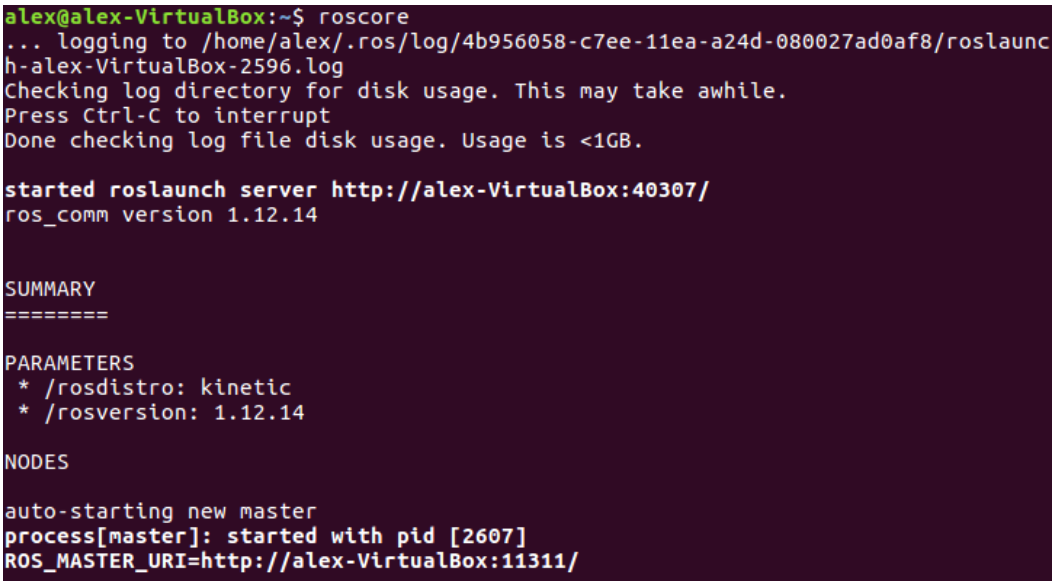

Figura 10.- Roscore

Finalmente, se ejecuta el siguiente código en la carpeta de launch del paquete instalado

```
$ roslaunch usb_cam-test.launch
```

Se visualiza la video imagen de la webcam en la herramienta imagen view de ROS

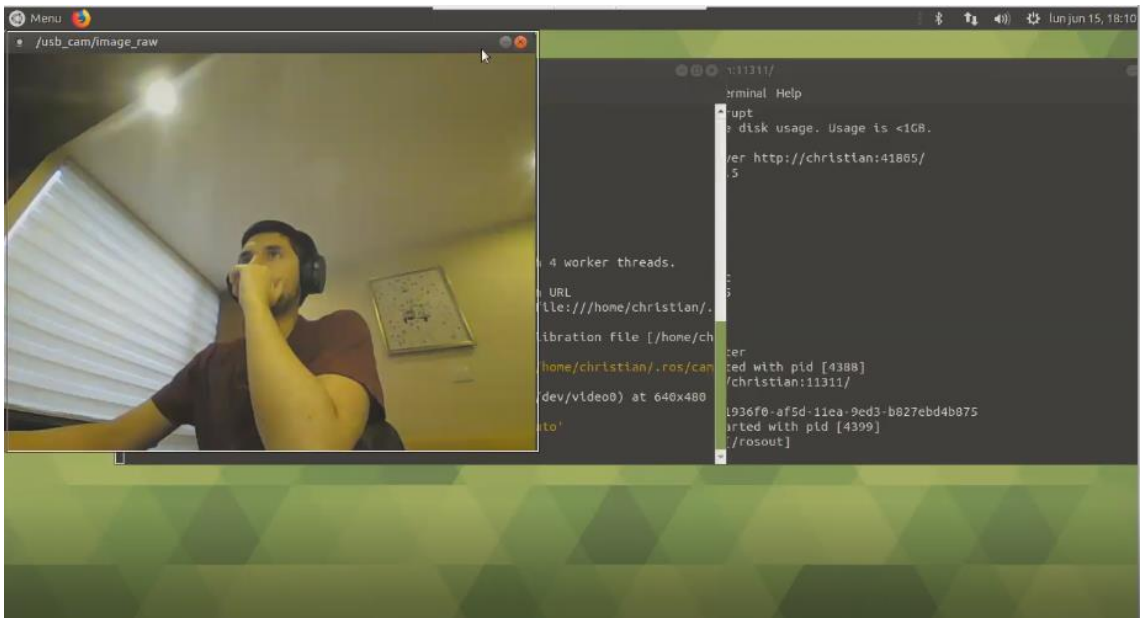

Figura 11.- Implementación de la webcam en Imagen view

Ahora se visualiza con la herramienta Rviz de ROS, para ejecutar Rviz se ingresa el siguiente comando

```
$ rosrun rviz rviz
```

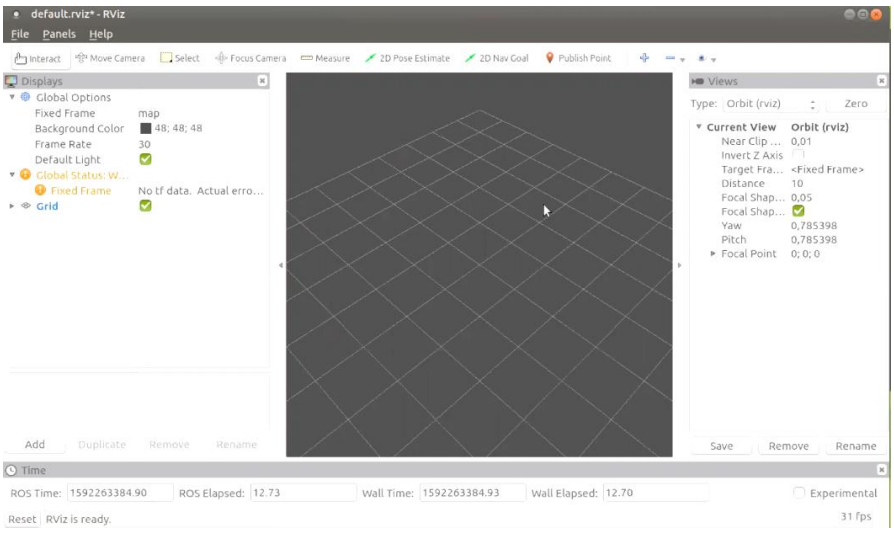

Figura 12.- Rviz

Se agrega el dispositivo de visualización, click en image

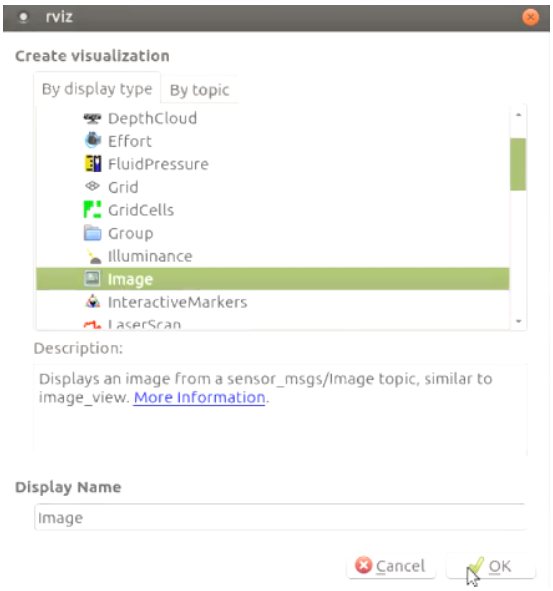

Figura 13.- Opciones para agregar a dispositivo

Luego se agrega el topic creado previamente.

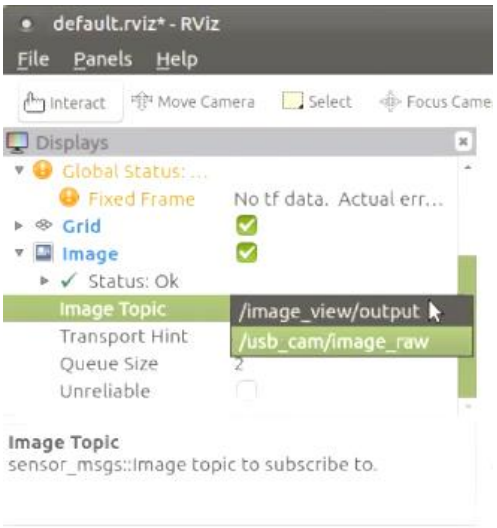

Figura 14.- Agrega topic image_raw

Finalmente se visualiza la imagen en Rviz

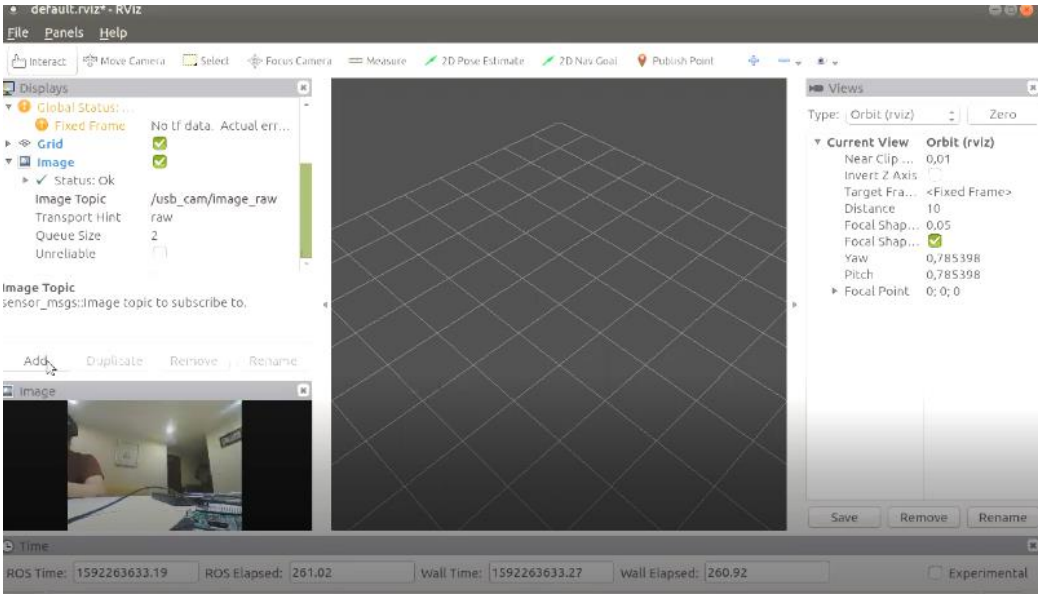

Figura 15.- Implementación de la webcam en Rviz

Verificación

Para verificar los topic que están activo se ingresa el siguiente comando

```
$ rostopic list
```

En la siguiente figura se observa los topic activo del paquete usb_cam

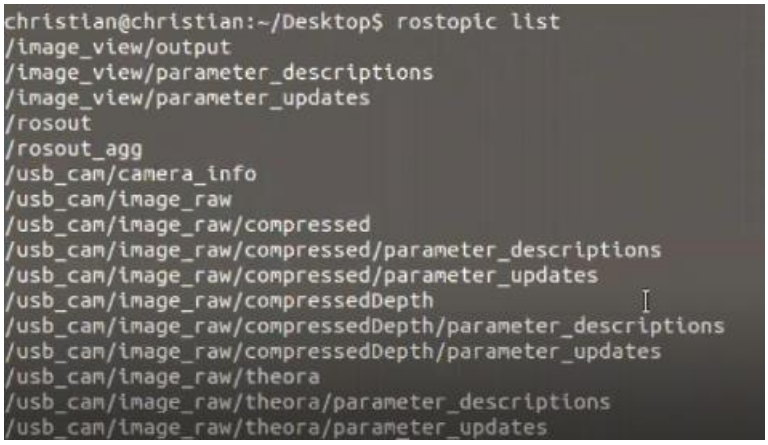

Figura 16.- Topics activos

En la siguiente figura se observa las especificaciones de la cámara, el tipo de video MPGE, el tamaño del video 640x480 a 30FPS.

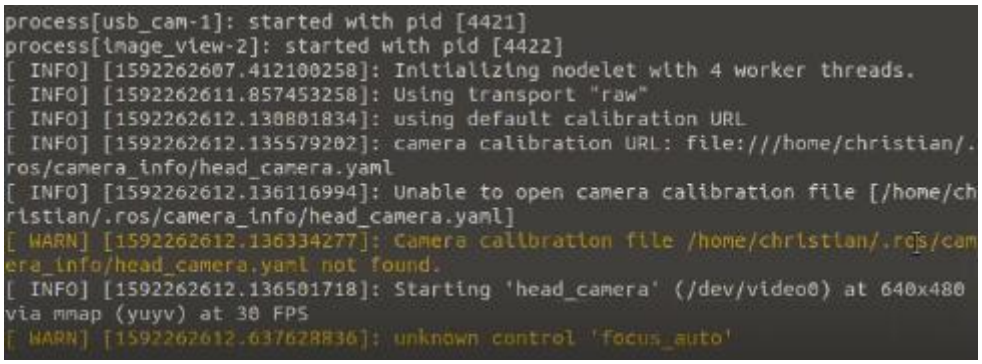

Figura 17.- Característica de la cámara

Para verificar los nodos creados se ingresa el siguiente comando

```
$ rqt_graph
```

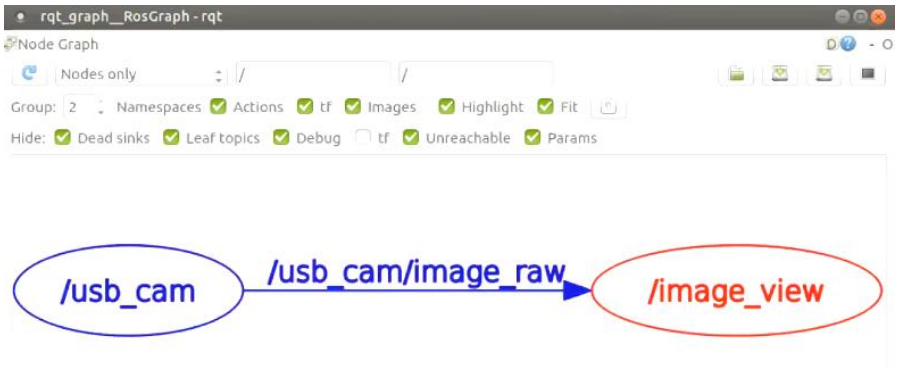

Figura 18.- Nodos creados

En la siguiente figura se observa /usb_cam como nodo publicador que está unido por topic /usb_cam con el mensaje /usb_cam/image_raw que conecta el nodo suscriptor /image_view.

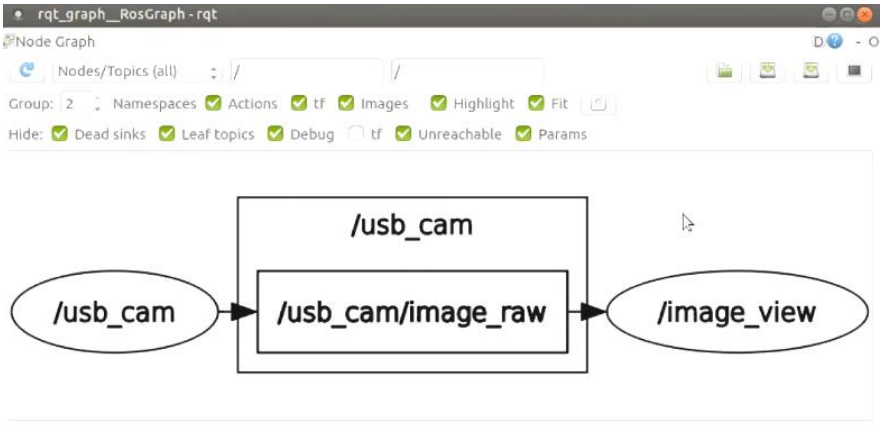

Figura 19.- Verificación de nodos, mensajes y topic creados

4.2 GPS

Para realizar la implementación primero se realiza la conexión del GPS con la Raspberry de la siguiente forma. También se conecta el módulo Wi-Fi y se alimenta con una batería Lipo de 5Amps, para que el módulo GPS pueda enviar datos al Raspberry a través de la conexión serie.

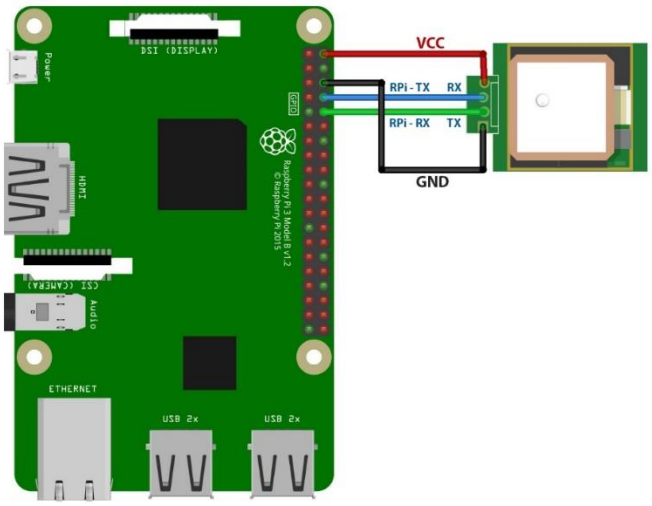

Figura 20.- Conexión de Raspberry y modulo GPS Neo 6M

4.2.1 Configuraciones del software del GPS

Con respecto al software para obtener los datos de modulo GPS se realiza las siguientes configuraciones. Se modifica el archivo /boot/config.txt con el siguiente comando

```
$sudo nano /boot/config.txt
```

Se añade la línea de parámetros del gps

```
$ dtparm=spi=on

dtpverplay=pi3=disable-bt

core_freq=250
```

```
enable_uart=1

force_turbo=1
```

Presionar Ctrl +x y Enter para guardar el archivo modificado. UART (Universal Asynchronous Receiver) es un dispositivo que controla los puestos y dispositivos serie, la cual se debe desactivar para modificar el archivo para la comunicación serial.

Por seguridad antes de editar el archivo /boot/cmdline.txt. se realiza una copia de seguridad mediante el siguiente comando:

```
$ sudo cp /boot/cmdline.txt /boot/cmdline_backup.txt
```

Para editar el archivo se ingresa la línea de comando:

```
$ sudo nano /boot/cmdline.txt
```

Se ingresa nuevos valores, específicos del dispositivo

```
$ dwc_otg.lpm_enable=0 console=tty1 root=/dev/mmcblk0p2
rootfstype=ext4 elevator=deadline fsck.repair=yes rootwait
quiet splash plymouth.ignore-serial-consoles
```

Presionar Ctrl +x y Enter para guardar el archivo modificado

Finalmente, se reinicia el Raspberry Pi

```
$ sudo reboot
```

Ahora, tanto el hardware y como el software del GPS son configurados, se observa que el LED azul del Neo 6M parpadea que significa la recepción de datos del módulo. Se ejecuta el comando cat para recibir los datos del módulo GPS del puerto serial ttyAMA0

```
$ sudo cat /dev/ttyAMA0
```

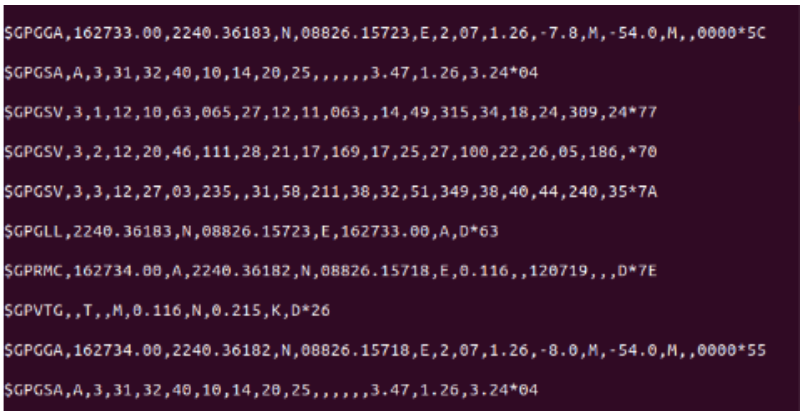

Figura 21.- Obtención de datos del módulo GPS

Para modificar los datos de la Figura 21 se escribe el siguiente código Python, primero se instala la biblioteca de Python

```
$ pip install pynmea2
```

Luego se escribe el siguiente código

```
$ import serial
import time
import string
import pynmea2

while True:
     port="/dev/ttyAMA0" //nombre del puerto serial
     ser=serial.Serial(port, baudrate=9600, timeout=0.5)
     dataout = pynmea2.NMEAStreamReader()
     newdata=ser.readline()

     if newdata[0:6] == "$GPRMC":
          newmsg=pynmea2.parse(newdata)
          lat=newmsg.latitude // formato de datos
          lng=newmsg.longitude // formato de datos
```

```
            gps = "Latitude=" + str(lat) + "and Longitude=" +
str(lng)
            print(gps)
```

Se ejecuta el código de Python

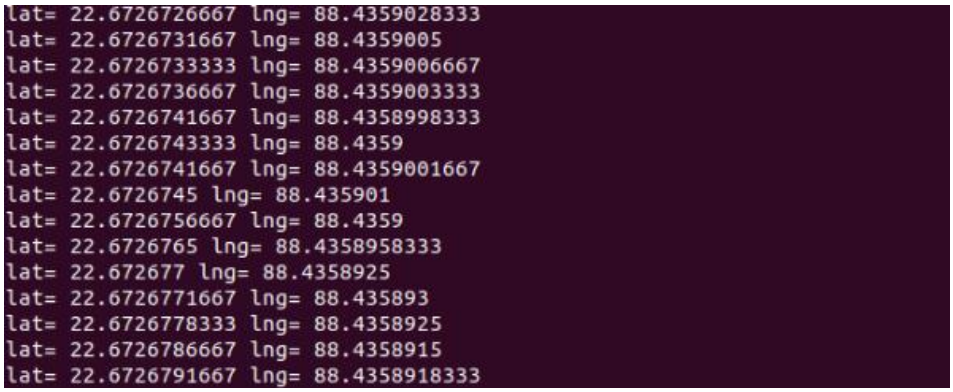

Figura 22.- Obtención de datos modificados del GPS

4.2.2 Página web de seguimiento de geolocalización a tiempo real

Para visualizar los datos obtenidos GPS Traker a tiempo real en una página web

Ahora con los datos obtenidos del GPS, se crea el sistema de rastreo a tiempo real del robot, la cual puede ser visualizada en cualquier parte del mundo.

Primero se realiza la página web por medio de una API llamada PubNub para enviar dato del GPS de Raspberry Pi a nuestra página web. Ver ANEXO A: Código de la página web.

En el código de la página web se cambia los datos de publicación y suscripción, para ello es necesario registrarse en PubNub que es totalmente gratis. En la sección de panel de administración de PubNub se observa las claves del publicador y del suscriptor, que posteriormente serán ingresadas al código.

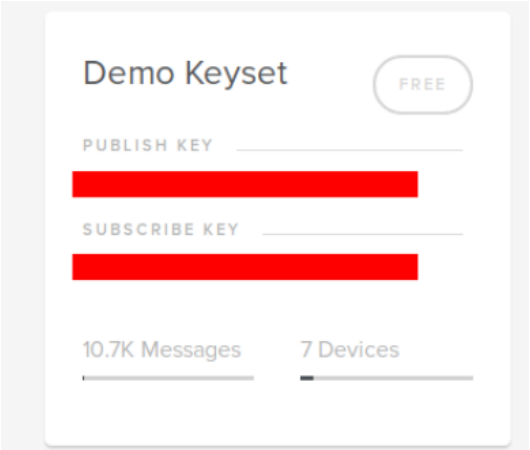

Figura 23.- Clave del publicador y suscriptor de PubNub

Se dirige a la página Google Cloud Console para obtener la clave de API del mapa de Google. Se habilita “Maps JavaScript API”, luego se crea las credenciales y se copia la clave de API. Esta clave API se pega en el código.

```
<script
src="https://maps.googleapis.com/maps/api/js?v=3.exp&key=YO
UR_GOOGLE_API_KEY&callback=initialize"></script>
```

Luego se guarda el archivo con la extensión .html y se ejecuta.

Figura 24.- Pagina web con rastreo GPS

En la Figura 24 se observa el marcador rojo indicando la ubicación del robot. Además, en el código de la página web se agrega el siguiente código que genera una coordenada aleatoria (latitud y longitud) cada 500 milisegundos y se publica en el mismo canal que muestra el marcador automáticamente donde crea la ruta a través de los putos obtenidos.

```
<script>
function newPoint(time) {
      var radius = 0.01;
      var x = Math.random() * radius;
      var y = Math.random() * radius;
      return {lat:window.lat + y, lng:window.lng + x};
        }
      setInterval(function() {
      pubnub.publish({channel:pnChannel,
message:newPoint()});
      }, 500);
</script>
```

Se modifica el código de Python para que envié los datos del GPS a la página web.
Se instala la biblioteca de Python para usar Pubnub en el código creado.

```
pip install 'pubnub>=4.1.4'
```

Se configura el código de Python. Ver Anexo B: Código de Python para enviar datos a la página web.

5. Simulación de la cámara web con el sistema de georreferencia vehicular usando ROS

Finalmente, se ejecuta el nodo de la cámara web con la herramienta imagen_view descrita en la sección 4. También se ejecuta el archivo Python y se abre el archivo .html

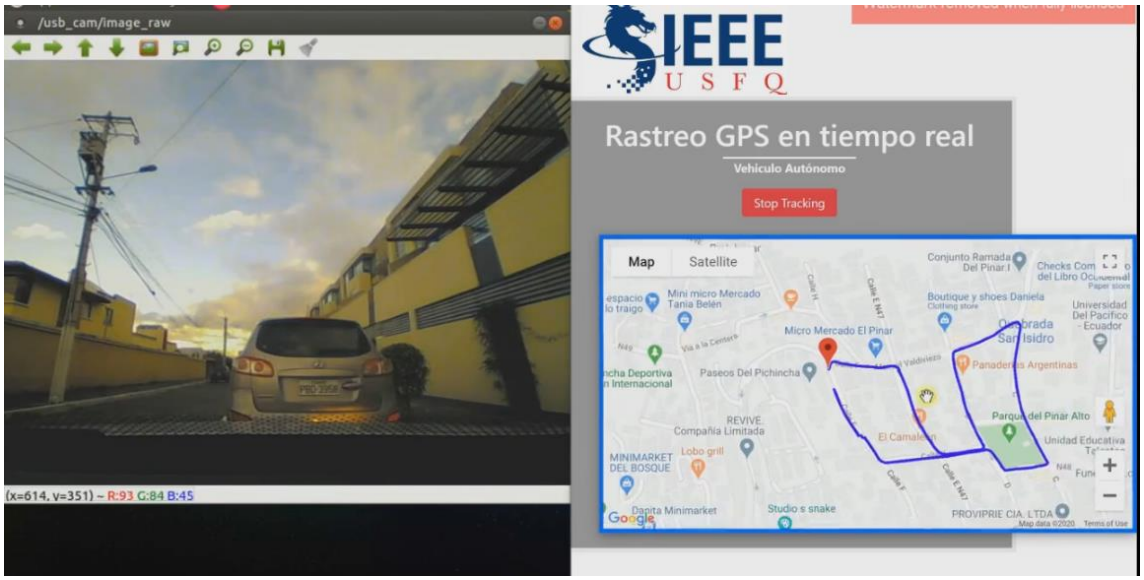

Figura 25.- Simulación de la Webcam y el GPS

Conclusiones

El desarrollo tecnológico en el campo de vehículos autónomos empuja también el crecimiento de las necesidades de servicio y operación, fomentando el interés en sectores industrial y académico. Ante el interés de integrar estrategias de interacción de la robótica en entornos del mundo real, las capacidades que presenta ROS adquieren un potencial de motivación para el desarrollo del proyecto. La flexibilidad y las oportunidades que ofrece ROS para explorar aplicaciones en la robótica son muy extensas. La alta disponibilidad de paquetes y librerías, y su portabilidad a este sistema robótico condujo el proyecto de forma favorable, alcanzando resultados exitosos.

Sin embargo, el proyecto en sus inicios atravesó por problemas de implementación debido a que a pesar de la amplia gama de librerías que ROS dispone, muchas de ellas son adecuadas solamente para determinados dispositivos diferentes a los que el proyecto dispone. La investigación posterior permitió sobrellevar esta problemática con el

soporte del Community Level que está integrada por estudiantes e investigadores que aporta a la comunidad robótica.

En cuanto a la implementación de la cámara web que se integró en una Raspberry, primero se identificó las características y especificaciones del dispositivo: el ancho y el largo de la imagen, el formato de pixeles de imagen (YUYV) y la velocidad de transmisión en términos de tramas (frames) por segundo, para conseguir una óptima comunicación entre el dispositivo y ROS.

En relación con las herramientas de visualización descritas anteriormente, se optó por imagen_view, ya que ésta cuenta con una interfaz simple para obtener la video imagen en tiempo real y también calcula el valor de los colores RGB. Por otra parte, Rviz muy útil durante la fase 1 del proyecto, se descartó en esta fase por estar direccionada a la virtualización del robot, lo cual no corresponde al objetivo de este trabajo.

Para la implementación del GPS, se aprovechó la API de libre acceso llamada PubNub. La gratuidad de la aplicación contribuyó enormemente para la realización del proyecto y conseguir la ubicación del robot georreferenciado por Google Maps que puede ser visualizada por medio de la API de PubNub

Reconociendo las ventajas de ROS para la realización de futuros proyectos y previendo por tanto el uso de repositorios no verificados, se reconoce que esta práctica podría provocar la vulnerabilidad de nuestro robot. Por este motivo, previamente a la implementación del robot físico es recomendable realizar la virtualización del robot en Rviz para determinar errores, dotarlo de seguridad y precautelar el sistema.

REFERENCIAS BIBLIOGRAFICAS

Quigley, M. (2009). “ROS: an open-source Robot Operating System”. In:

ICRA Workshop on Open Source Software. Vol. 3.

Lentin, J. (2015). Mastering ROS for robotics Programming. Pack Publiching

ROS.org. (2019). ROS Introduction. Recuperado Julio 18,2020 de ROS: https://www.ros.org/about-ros/

ROS.org. (2019). Official website. Recuperado Julio 18,2020 de ROS: http://www.ros.org/. Recuperado 18/07/20

ROSWIKI, (2012). Ask ROS. Recuperado Julio 18,2020 de ROS: http://wiki.ros.org/.

RViz (2016).Vizualitation tool. Recuperado Julio 18,2020 de ROS: http://wiki.ros.org/rviz. Recuperado 18/07/20

ROS Packages,(2016). Packages. Recuperado Julio 18,2020 de ROS: http://www.ros.org/browse

rqt ROS (2016). Packages. Recuperado Julio 18, 2020 de ROS: http://wiki.ros.org/rqt_graph. [Accessed 21 7 2017

ANEXOS A: Código de la página web

```
<!doctype html>

<html>

  <head>

    <title>Realtime GPS Tracker</title>

    <script
src="https://cdn.pubnub.com/sdk/javascript/pubnub.4.19.0.mi
n.js"></script>

    <link rel="stylesheet" href="map.css">

    <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/b
ootstrap.min.css" integrity="sha384-
Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAi
S6JXm" crossorigin="anonymous">

<script
src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/boo
tstrap.min.js" integrity="sha384-
JZR6Spejh4U02d8jOt6vLEHfe/JQGiRRSQQxSfFWpi1MquVdAyjUar5+76P
VCmYl" crossorigin="anonymous"></script>

  </head>

  <body>

      <div class="bg"></div>

     <div class="bg-others">

    <div class="container">

      <h1 >Realtime GPS Tracker with Raspberry PI</h1>

        <center><hr style="height:2px; border:none;
color:#ffffff; background-color:#ffffff; width:35%; margin:
0 auto 0 auto;"></center>

        <p>by SPARKLERS: We Are The Makers</p>

        <center><button class="btn btn-success col-sm-3"
id="action">Start Tracking</button></center><br>

        <center><div id="map-canvas"></div></center>

    </div>

          </div>


    <script>
```

```
    var map;
    var mark;
    var lineCoords = [];

    var initialize = function() {
      map  = new
google.maps.Map(document.getElementById('map-canvas'),
{center:{lat:lat,lng:lng},zoom:12});
      mark = new google.maps.Marker({position:{lat:lat,
lng:lng}, map:map});
    };

    window.initialize = initialize;

    var redraw = function(payload) {
      if(payload.message.lat){
      lat = payload.message.lat;
      lng = payload.message.lng;

      map.setCenter({lat:lat, lng:lng, alt:0});
      mark.setPosition({lat:lat, lng:lng, alt:0});

      lineCoords.push(new google.maps.LatLng(lat, lng));

      var lineCoordinatesPath = new google.maps.Polyline({
        path: lineCoords,
        geodesic: true,
        strokeColor: '#2E10FF'
      });

      lineCoordinatesPath.setMap(map);}
    };
    var pnChannel = "raspi-tracker";
```

```
    var pubnub = new PubNub({
      publishKey:   'YOUR_PUBLISH_KEY',
      subscribeKey: 'YOUR_SUBSCRIBE_KEY'
    });

document.querySelector('#action').addEventListener('click',
function(){
        var text =
document.getElementById("action").textContent;
        if(text == "Start Tracking"){
            pubnub.subscribe({channels: [pnChannel]});
            pubnub.addListener({message:redraw});

document.getElementById("action").classList.add('btn-
danger');

document.getElementById("action").classList.remove('btn-
success');
            document.getElementById("action").textContent =
'Stop Tracking';
        }
        else{
            pubnub.unsubscribe( {channels: [pnChannel] });

document.getElementById("action").classList.remove('btn-
danger');

document.getElementById("action").classList.add('btn-
success');
            document.getElementById("action").textContent =
'Start Tracking';
        }
        });
    </script>
    <script
src="https://maps.googleapis.com/maps/api/js?v=3.exp&key=YO
UR_GOOGLE_API_KEY&callback=initialize"></script>
```

```
  </body>
</html>
```

ANEXO B: Código de Python para enviar datos a la página web

```
<import serial
import time
import string
import pynmea2
from pubnub.pnconfiguration import PNConfiguration
from pubnub.pubnub import PubNub
from pubnub.exceptions import PubNubException

pnChannel = "raspi-tracker";

pnconfig = PNConfiguration()
pnconfig.subscribe_key = "Your Subscribe key"
pnconfig.publish_key = "Your Publish key"
pnconfig.ssl = False

pubnub = PubNub(pnconfig)
pubnub.subscribe().channels(pnChannel).execute()

while True:
    port="/dev/ttyAMA0"
    ser=serial.Serial(port, baudrate=9600, timeout=0.5)
    dataout = pynmea2.NMEAStreamReader()
    newdata=ser.readline()

    if newdata[0:6] == "$GPRMC":
        newmsg=pynmea2.parse(newdata)
        lat=newmsg.latitude
```

```
        lng=newmsg.longitude
        try:
            envelope =
pubnub.publish().channel(pnChannel).message({
            'lat':lat,
            'lng':lng
            }).sync()
            print("publish timetoken: %d" %
envelope.result.timetoken)
        except PubNubException as e:
            handle_exception(e)
```