AMPLIACIÓN DE ROBÓTICA

Comparativa de técnicas de SLAM empleando un robot movil

Grado en Ingeniería Electrónica, Mecatrónica y Robótica

Índice

Autores:
López Gil, Miguel
Montes Grova, Marco Antonio
Osuna Cañas, Alfonso Carlos

1. Introducción al proyecto

En este proyecto se implementan diferentes técnicas de SLAM visuales, así como diversas fuentes de odometría, sobre un robot móvil de bajo presupuesto. El objetivo final ha sido diseñar e implementar una plataforma de desarollo donde poder estudiar los conceptos vistos durante el curso. Para ello, y teniendo en mente el tiempo y los recursos limitados de los que se disponen, se han optado por soluciones baratas o fabricadas *in-house* y de sencilla implementación para favorecer un rápido prototipado.

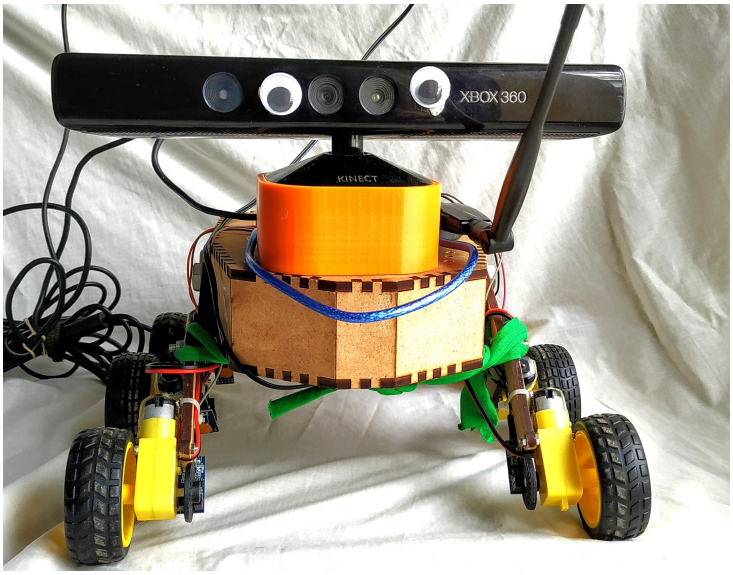


Figura 1: El robot, también conocido como *Wheele*

El robot, o, como ha sido apodado cariñosamente; *Wheele*, consta de una *Raspberry Pi 3B+* como ordenador de a bordo, que hace de interfaz con la cámara, lee las medidas de los encoders y establece un bus i2c con una IMU *mpu-6050*. Un microcontrolador ATmega328 de una placa Arduino, comunicado por puerto serie con la Raspberry, ejecuta un PI y actúa sobre los motores. Todo el sistema, exceptuándo el control de bajo nivel del micro, corre en ROS Kinetic, en una configuración de multi-masters que relega los procesos computacionalmente costosos (algoritmos de SLAM, data fusion...) a un portátil conectado a la Raspberry a través de una red *wireless* dedicada. Como drivers para los motores se han usado tres circuitos integrados *L293D* (dos puentes H por cada chip), que reciben como *input* una señal generada por el micro mediante *Pulse-Width Modulation*. En la sección de hardware de este mismo documento se detallan en más profundidad los sistemas electrónicos empleados, los buses de comunicaciones y se discute la estructura del robot.

La odometría del robot se obtiene a partir de una estimación de la posición a partir de encoders ópticos, y de la velocidad y aceleración medidas por la IMU. La fusión sensorial se realiza con un filtro de Kalman extendido, implementado en el paquete *robot localization* de ROS. El código desarollado para este proyecto y el uso de software externo es expuesto en la seccion4.
Las imágenes y medidas de profundidad utilizadas para las técnicas SLAM provienen de una *Kinect for Xbox 360* de Microsoft, que cuenta con una cámara RGB y un sensor de profundidad matricial. Una breve explicación teórica de las metodologías SLAM empleadas puede encontrarse en la sección5.

2. Hardware empleado en el proyecto

Con el fin de obtener una vista general de todo el proyecto, así como del coste del mismo se lista a continuación todos los componentes del robot y su coste:

- Chasis del vehículo. Fabricado a partir de un tablón plano de edf cortado en una cnc láser de forma gratuita gracias al Fablab de la Universidad de Sevilla en la facultad de Arquitectura.

- Motores de corriente continua(1pavos). Adquiridos a través de tiendas digitales por un módico precio, lo cual se refleja en una calidad más bien pésima. Sin embargo, formaban parte de un kit que incluye ruedas y acople al eje de transmisión, suponiendo un ahorro importante de tiempo y dinero.

- Encoders ópticos (6pavos). Compuestos por un diodo emisor de luz comprados también a través de internet y por un disco que interrumpe el paso del haz lumínico diseñado específicamente e impreso en 3D.

- Microcontrolador ATmega328 (10pavos). Aunque la gran parte del código implementado en el micro no utiliza las librerías de arduino, por comodidad se ha optado por un microchip ya montado en una placa tipo arduino (un modelo no oficial, clónico).

- IMU mpu-6050 (2pavos).

- Raspberry Pi 3B+ (30pavos). Constituye el cerebro del robot, y centraliza la comunicación con los demás sistemas. Aunque puede resultar el componente más caro del proyecto, se disponían de varias de antemano, por lo que no hubo que asumir su coste.

- Kinect for Xbox 360 de Microsoft (15pavos). Esta popular cámara merece la fama que tiene por lo asequible que es (especialmente el modelo más antiguo) y las utilidades que trae. Se demuestra como la opción más socorrida para desarollar con poco presupuesto aplicaciones que requieran de imágenes o sensores de profundidad.

2.1. Estructura

El robot ha sido diseñado con una topología inspirada en los vehículos de exploración espacial conocidos como *rovers*, utilizando materiales y métodos de construcción asequibles y manteniendo como objetivo una sencilla sustitución e iterabilidad de cada componente. La estructura está diseñada en *FreeCad* y compuesta por piezas de madera mecanizadas en una cortadora láser cnc. Algunos elementos sin funcionalidad estructural, como los encoders o el soporte de la cámara, han sido fabricados mediante impresión 3D.

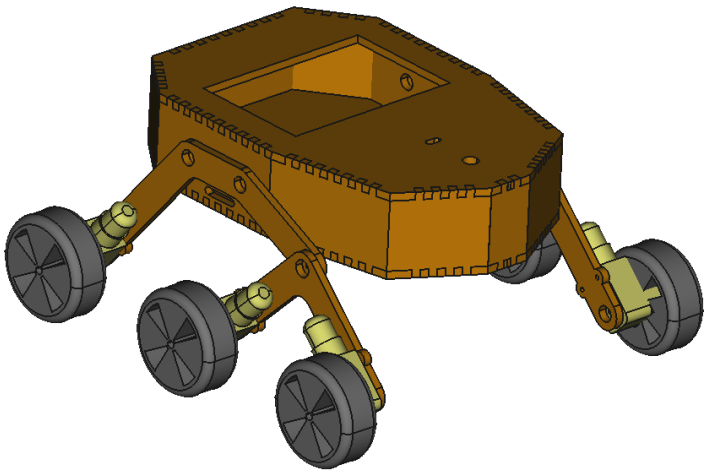


Figura 2: Modelo CAD del robot

La disposición geométrica es la de un vehículo diferencial, con tres pares de ruedas, cada una con su respectiva pareja de motor y encoder.

3. Software empleado en el proyecto

3.1. ROS

ROS (Robotic Operating System) consite en un framework para la robótica sobre el que desarrolladores de cualquier ambito pueden aportar soluciones modulares a los sistemas de forma general. La modularidad de ROS hace que desarrollar un nuevo sistema o prototipo pueda ser una tarea tan sencilla como interconectar módulos de otros desarrolladores con los propios.
Esta característica lo hace idoneo para un proyecto como este en el que el uso de técnicas de percepción, mapeado, localization, etc, complejas se escapa del alcance; y solo sea necesario el desarrollo de modulos específicos de este robot que sean capaces de comunicarse con aquellos más complejos y sean capaces de proporcionarle los datos necesarios para implementar los algoritmos de SLAM.

3.1.1. Computación distribuida con ROS-Multimaster Package

Las técnicas de SLAM a testear en el robot desarrollado son computacionalmente costosas. De manera genérica a los metodos de SLAM visual usados, para cada frame capturado por la camara el algoritmo extrae 'features' de la imagen, las alamacena y compara los conjuntos con los de otro frame. Entre ambos frames el algoritmo calcula una matriz de transformación homogenea con la cual se es capaz de saber, al menos localmente, cuanto y como se ha desplazado el robot. Sumado a eso cada algoritmo tiene su forma propia de localizar globalmente en el mapa creado al robot. Todo este coste computacional es imposible asumirlo desde una Raspberry-Pi, por lo que surge la necesidad de hacer el procesado en un servidor remoto al robot en tiempo real con las dificultades que eso conlleva, entre ellas y la mas importante, la sincronización de datos.
Bajo estas condiciones de diseño y restricciones aparece una solución de la comunidad: ROS multimaster_fkie. La premisa que lo rige es simple, dado un sistema mutirobot cada elemento mantiene la ejecución de 2 nodos locales, *master_discovery* y *master_sync*; el primero es el encargado de mantener una lista del resto de *roscores* disponibles en la red y de publicar el suyo propio como activo. El otro es el encargado de suscribir su *core* los nodos de otros *cores*.
Des esta forma todos tienen acceso de una forma hasta cierto punto 'real-time' a los recursos del resto.

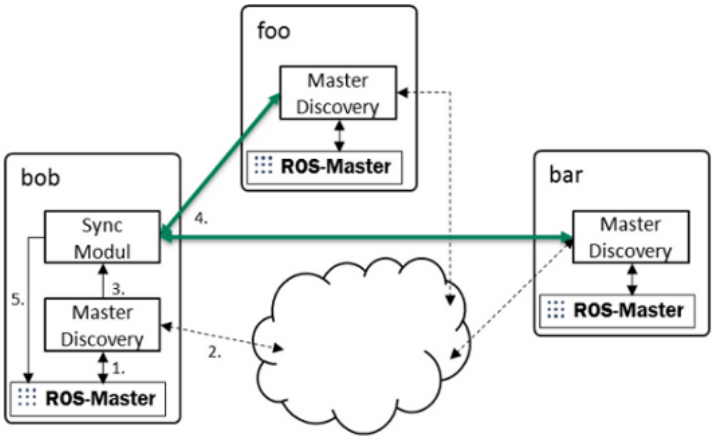


Figura 3: Esquema del funncionamiento básico del paquete Multimaster

En este proyecto su integración se ha llevado a cabo lanzando en la Raspberry-Pi y en otro PC los nodos correspondientes siendo la Raspberry-Pi la suministradora de información y el PC el encargado de procesarla.

3.1.2. Interfaz con IMU

Qué hace, como se procesan los datos y que se publica.

3.1.3. Interfaz con encoders

Mismito que lo de la IMU

3.1.4. Odometría

Explicar qué se hace, qué y donde se publica y a quién se suscribe

3.1.5. Interfaz Raspberry-Pi/Arduino

Qué hace, a quién se suscribe y donde publica el qué

3.1.6. Interfaz Arduino/Motores

Same old same old

3.2. Implementación filtro estadístico

Introducción de porqué es interesante y eso, lo de vender la moto as usual

3.2.1. Filtro de Kalman Extendido

Background teorico que coma espacio, poner la imagen del filtro esa con colores bonicos y eso

3.2.2. ROS-Robot Localization Package

pues eso, lo mismo de lo otro, como funca y tal

3.3. Arquitectura general del sistema

Aqui lo de rqt_plot and eso

4. Introducción al SLAM *(Simultaneous Localization and Mapping)*

¿Qué cojones es el SLAM?

4.1. Clasificación técnicas de SLAM

La forma en que se toma información de las imagenes que recibe y de su entorno y cómo la procesa permite distinguir las siguientes tipos de sistemas SLAM visual:

- *Denso vs Disperso*
  En función de la cantidad de datos tomada de cada imagen, se realizará ésta clasificación. Las técnicas de SLAM disperso sólo emplean una pequeña región de pixeles, cómo pueden ser puntos característicos. Sin embargo, las técnicas densas emplean la mayoría o todos los píxeles de cada frame que reciben.
  Debido a que la cantidad y tipo de información que toman son diferentes, el tipo de mapa que se obtendrá también lo es. Con las técnicas de SLAM disperso se obtendrá una nube de puntos que será una representación de los puntos característicos de las escena y se usará sobre todo para hacer un tracking de la pose de la cámara. Por otro lado, un mapa dentro tendrá muchos más detalles y, requerirá de una mucho más elevada carga computacional.

- *Directo vs Indirecto*
  En función de cómo las técnicas de SLAM empleen y traten la información que reciben, se podrá clasificar en SLAM directo o indirecto.
  El SLAM infirecto, se basa en extraer primero las features de la imagen, puntos característicos, para posteriormente emplearlos para localizarse y contruir el mapa. Para extraer estos puntos caracteristicos existen muchos descriptores: ORB, SIFT, FAST, etc.
  En contraste, el SLAM Directo,emplean directamente la intensidad de los pixeles, de tal modo que se obtengan features intermedios. Estos metodos tratan de obtener la profundidad y estructura del entorno a partir de una optimización del mapa. Los procesos de extracción de puntos característicos son mucho más pesados computacionalmente si se trabaja al mismo rate que un slam indirecto.
  Por último, destacar, que los metodos indirectos de SLAM son más tolerantes a los cambios de iluminación del entorno.

  A continuación, se monstarán una comparativa de una serie de técnicas de SLAM:

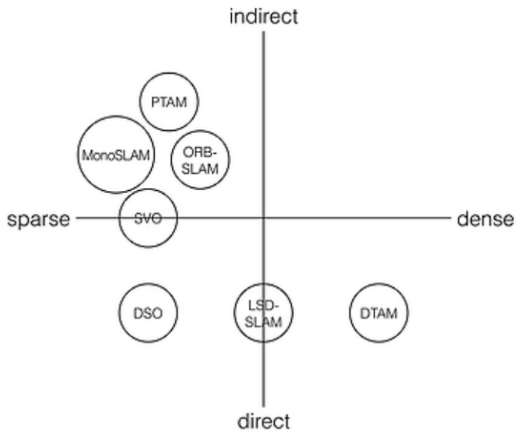


Figura 4: Comparativa de técnicas SLAM

En éste proyecto, se probará ORB-SLAM2 y RTAB-Map, los cuales abordan el problema del SLAM desde dos puntos de vista completamente distinto, cómo se verá a continuación.

4.2. *RTAB-Map SLAM*

RTAB-Map *(Real-Time Appearance-Based Mapping)* es una técnica de Graph-SLAM[1] basada en la detección de bucles cerrados incrementales. Es totalmente funcional con sensores RGB-D, Stereo y LIDAR.
El detector de bucles cerrados se basará en la comparativa de cuán semenjantes son la imagenes en una localización y la previa. Cuándo una hipótesis de bucle cerrado es aceptada, se añade una nueva restricción al *graph* del mapa y, tras ello, el optimizador minimiza el error del mapa.

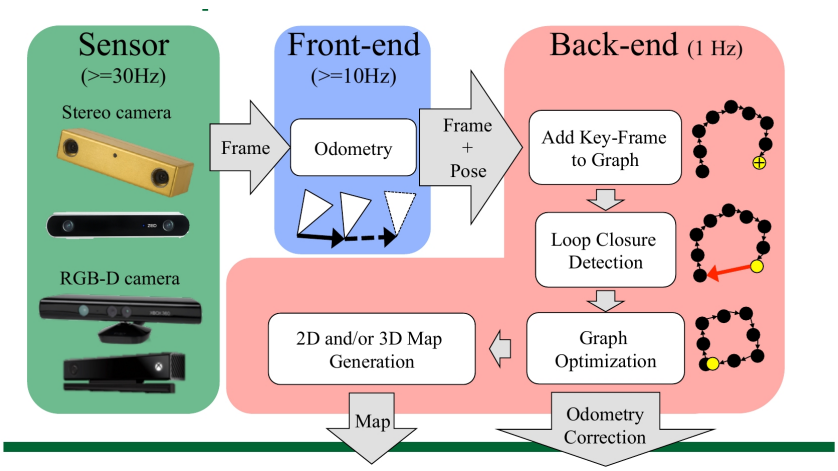


Figura 5: Esquema del Back-End y Front-End de RTAB-Map

4.2.1. Fundamento teórico de la técnica

Este algoritmo plantea una estrategia de particionado de memoria que pretende asemejarse al funcionamiento de la memoria humana, dónde ésta se estructura en:
Memoria de trabajo del robot *(Working Memory)*, la memoria a largo plazo *(Long Term Memory)*, la memoria a corto plazo *(Short-term memory)* y la memoria sensorial *(Sensory Memory)*.
De ese modo, se mantendrán en la memoria de trabajo del robot aquellas localizaciones que se han visitado recientemente y con más frecuencia, mientras que el resto pasarán a la memoria de largo plazo.

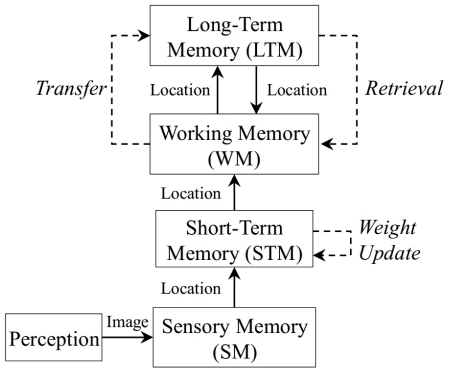


Figura 6: Estructura de memoria de la técnica RTAB-Map

Se partirá de la premisa de que aquellas localizaciones que son visitadas de forma más frecuente son más propensas a crear bucles cerrados. Por ello, el número de veces que una localización sea visitada será

[1] http://robots.stanford.edu/papers/thrun.graphslam.pdf

empleado como peso, de esta forma serán transferidas desde la memoria de trabajo a la memoria de largo plazo aquellas observaciones que tengan mayor peso.
La memoria a corto plazo, *STM*, tiene como misión buscar las similitudes que existan entre dos imágenes consecutivas, mientras que la memoria de trabajo, *WM*, es la encargada de detectar los bucles cerrados entre las localizaciones en el espacio. El número de localizaciones almacenadas en la memoria del trabajo del robot es limitado. El tamaño de la memoria a corto plazo, *STM*, está basado en la velocidad del robot y en la frecuencia de adquisición de las localizaciones.

4.2.2. Análisis de resultados obtenidos

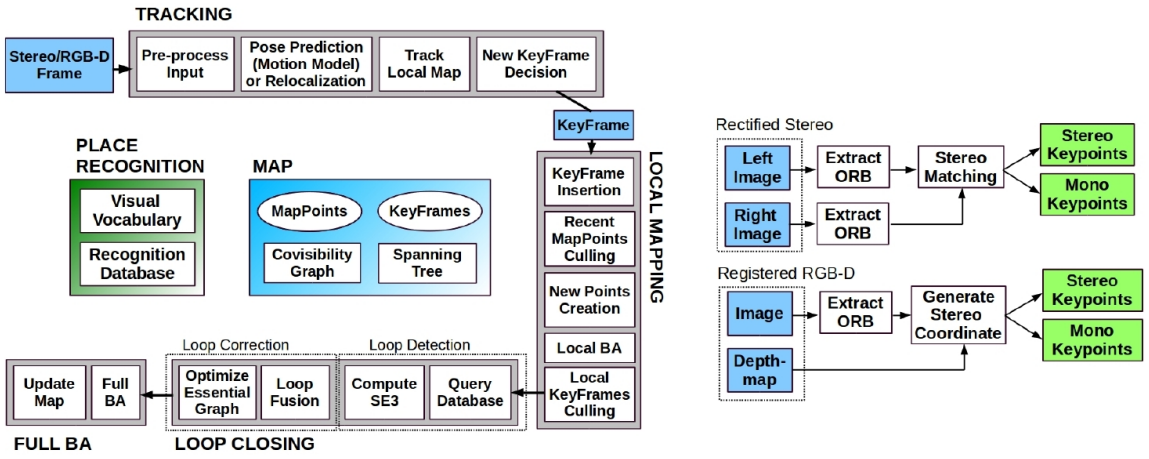
4.3. *ORB-SLAM 2*

ORB-SLAM2 es una técnica de SLAM en tiempo real para cámaras Monocular, Stereo y RGB-D que se engloba dentro de las técnicas de *Sparse-Slam*. Se computa la trayectoria y hace una reconstrucción dispersa 3D del entorno. Al igual que todas las técnicas de SLAM, se basa en la detección de bucles cerrados y se relocaliza en tiempo real.
Se basa en la detección de *keyframes* que emplea para hacer un tracking de los mismos y a partir de ello crear el mapa local que, posteriormente, se optimizará junto al mapa global.
Unos de los puntos destacables de ésta técnica de SLAM son los siguientes:

- Emplea *grafo de covisibilidad*. Tanto el seguimiento como el mapping se focalizan en el área covisible, independientemente del tamaño del mapa completo, consiguiendo así explorar entornos amplios sin aumentar el tiempo y la carga de computación.
- La estrategia para detectar los bucles cerrados de visión en tiempo real se basa en la optimización de un de grafo denominado *Essencial Graph*, lo cuál se desarrollará mas adelante.

4.3.1. Fundamento teórico de la técnica

A continuación, se tratará un poco el funcionamiento interno del algoritmo, el cuál presenta el siguiente esquema:



(a) System Threads and Modules. (b) Input pre-processing

Figura 7: Estructura interna de ORB-SLAM2

Se observa cómo existen 6 grandes módulos, los cuales de lanzarán en 3 hilos paralelos.
En primer lugar, se tendrá el preprocesamiento de la imagen, de la cuál se obtendrán ORB features *(Oriented Fast and Rotated BRIEF)*.
Los hilos desempeñarán las siguientes funciones:

- El *tracking* se encargará de localizar la cámara en cada frame buscando marches entre las features del mapa local y minimizando la reprojección del error aplicando un *Bundle Adjustment*[2] sólo de movimiento.
- El *Local Mapping* se encargará de gestionar y optimizar el mapa local aplicando un *BA* local.
- El hilo de *Loop Closing* se encargará de detectar bucles cerrados grandes y corregir la deriva acumulada realizando una optimización del grafo del *pose* obtenido. Este hilo, lanza 4 hilos internos que se encagarán de realizar un *Bundle Adjustement* completo tras la optimización del grafo del *pose*, para hayar la solución más optima.

[2]https://homes.cs.washington.edu/ sagarwal/bal.pdf

En caso de que el sistema pierda el tracking, lleva integrado un modulo de *Plane recognition* basado en un vocabulario de palabras que básicamente es una especie de preentrenamiento de la técnica, para poder obtener y asociar keyframes más fácilmente.

4.3.2. Análisis de resultados obtenidos

En la imagen a continuación, puede observarse la técnica corriendo en un ordenador. Destacar que ésta técnica de SLAM presenta una carga computacional bastante elevada.

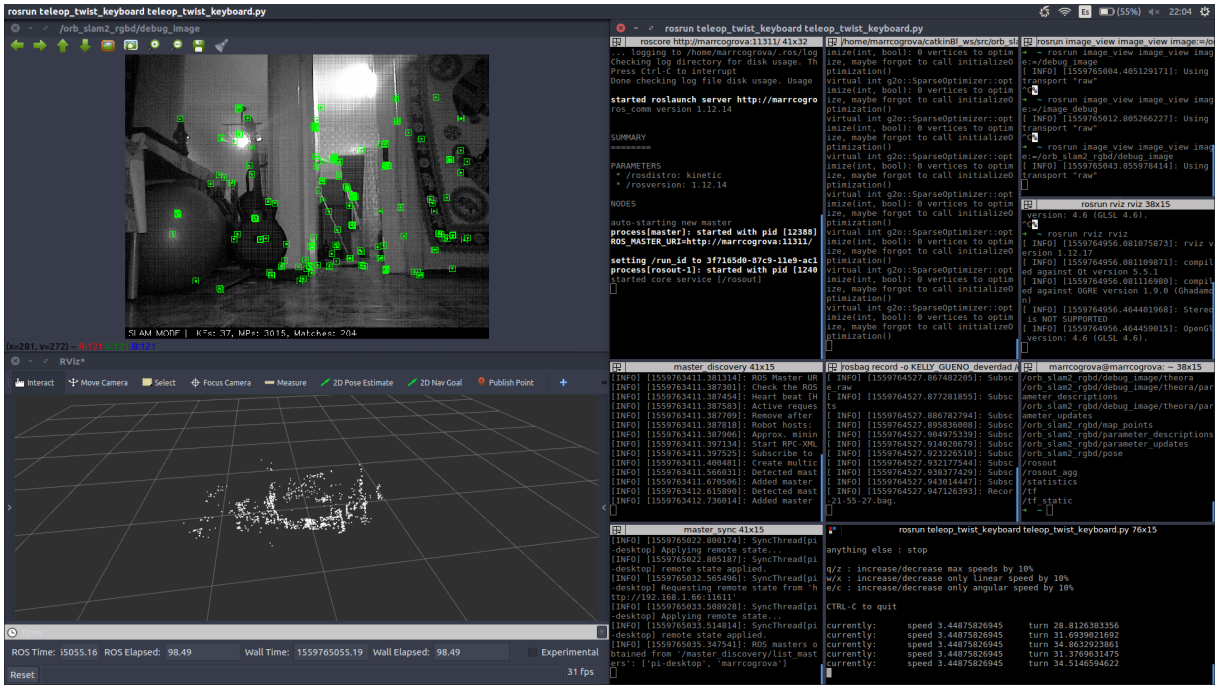


Figura 8: ORB-SLAM2 en funcionamiento

Se observa en la imagen superior izquierda las features que está obteniendo en ese frame, a partir de las cuales se ha creado en mapa local y, por extensión el global. En la imagen inferior izquierda, se puede observar el mapa creado, el cuál se analizará más tarde.
Por último, a la derecha se tienen las terminales a partir de las cuales se ha lanzado el SLAM y se realiza la comunicación con el robot.

Referencias

[1] R. Mur-Artal and J. D. Tardós, "ORB-SLAM2: an open-source SLAM system for monocular, stereo and RGB-D cameras," *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, 2017.

[2] IntRoLab, "RTAB-Map(real-time appearance-based mapping): RGB-D, stereo and lidar graph-based SLAM." https://github.com/introlab/rtabmap, 2014.

[3] T. Moore and D. Stouch, "A generalized extended kalman filter implementation for the robot operating system," in *Proceedings of the 13th International Conference on Intelligent Autonomous Systems (IAS-13)*, Springer, July 2014.

[4] M. Harms, "Python interface for cmdmessenger arduino serial communication library." https://github.com/harmsm/PyCmdMessenger, February 2017.

[5] T. Elenbaas, "Command messenger communication library for arduino & .NET." https://github.com/thijse/Arduino-CmdMessenger, September 2017.

[6] "Gammon software solutions." http://www.gammon.com.au.

[7] Atmel, *Atmel ATmega640 Datasheet*, February 2014.