ARQUITECTURA E IMPLEMENTACIÓN DE UN SISTEMA DE ADQUISICIÓN DE DATOS PARA SISTEMAS GLOBALES DE NAVEGACIÓN POR SATÉLITE (GNSS)

DIEGO FERNANDO ACOSTA ORTIZ

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FISICOMECÁNICAS
ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
BUCARAMANGA
2015

ARQUITECTURA E IMPLEMENTACIÓN DE UN SISTEMA DE ADQUISICIÓN DE DATOS PARA SISTEMAS GLOBALES DE NAVEGACIÓN POR SATÉLITE (GNSS)

DIEGO FERNANDO ACOSTA ORTIZ

Trabajo de grado para optar al título de
Ingeniero de Sistemas

DIRECTOR:
RAÚL RAMOS POLLÁN
DOCTOR EN INGENIERÍA INFORMÁTICA

CO-DIRECTOR:
SANTIAGO SOLEY RIMBLAS
MSc. EN INGENIERÍA DE TELECOMUNICACIONES

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FISICOMECÁNICAS
ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
BUCARAMANGA
2015

DEDICATORIA

A mis padres Fernando Acosta Castillo y Ana Virginia Ortiz por su inmenso sacrificio, ambos han sido mi ejemplo e inspiración durante mis días.

A mi hermana Carolina, quien cuidó de mi durante muchos años.

A todas las personas y amigos con los que he compartido y de quienes en los últimos años he aprendido valiosas lecciones.

AGRADECIMIENTOS

A mis padres quienes decidieron apoyarme con su esfuerzo y dedicación durante esta etapa universitaria para llevar a cabo mis objetivos.

A todos los docentes que han sido participes de mi proceso de formación académica y crecimiento personal, especialmente al Dr. Raúl Ramos Pollan de quien he aprendido mucho, gracias a su acompañamiento, instrucción y paciente orientación me ha ayudado a desarrollar un perfil más científico.

A Santiago, Fernando, Daniel, Marc, nuevamente a Raúl y a todo el equipo de Pildo Labs por ofrecerme esta interesante oportunidad de aprendizaje y entrenamiento.

A la Universidad Industrial de Santander y al grupo de Supercomputación y Cálculo Científico por todas las herramientas que me han otorgado durante mi proceso de formación académica.

A NCK por brindarme la oportunidad de llevar a cabo un proceso de crecimiento personal y formación profesional.

A Satsuki Furukawa por su apoyo en momentos de austeridad.

A mi novia por su cariño, apoyo, comprensión y paciencia.

TABLA DE CONTENIDO

Pág.

RESUMEN

TÍTULO: ARQUITECTURA E IMPLEMENTACIÓN DE UN SISTEMA DE ADQUISICIÓN DE DATOS PARA SISTEMAS GLOBALES DE NAVEGACIÓN POR SATÉLITE (GNSS)*

AUTOR: ACOSTA ORTIZ, DIEGO FERNANDO**

PALABRAS CLAVE: GNSS, Sistemas Embebidos, GPS, Galileo.

DESCRIPCIÓN:

Los Sistemas Satelitales de Posicionamiento Global (GNSS), han abierto la posibilidad de crear nuevas aplicaciones y solucionar nuevos problemas. Actualmente están surgiendo otros sistemas complementarios al GPS, conformado por una constelación 24 satélites en órbita media controlados por el Departamento de Defensa de EEUU, como la constelación GLONASS (Rusia, 24 satélites) o la constelación Galileo (Europa, 30 satélites todavía en despliegue). Por tanto nos dirigimos hacia un mundo GNSS multi-constelación donde hay espacio y oportunidad para nuevas aplicaciones basadas en el geo-posicionamiento. La disponibilidad de varias constelaciones permite un posicionamiento de mayor precisión e integridad con una mayor independencia geoestratégica. Por tanto la recepción y el monitoreo de las señales es cada vez más importante en aplicaciones como la navegación aérea, terrestre y marítima, la marcación de territorios, agricultura de precisión, etc. entre otros para así garantizar confiabilidad e integridad.

Al día de hoy las redes de monitoreo GNSS representan un alto costo por los componentes hardware y software requeridos. En un contexto multi-constelación, surge una necesidad aún mayor de estas redes. Teniendo en cuenta las nuevas oportunidades que brindan las plataformas hardware de bajo coste por su versatilidad, portabilidad, etc.

El presente proyecto plantea implementar un receptor de datos GPS/Galileo integrando hardware y software de bajo costo que permitirá una recolección de datos GNSS para múltiples áreas de investigación y desarrollo.

* Trabajo de grado

** Facultad de Ingenierías Físico-Mecánicas. Escuela de Ingeniería de Sistemas. Director: Raúl Ramos Pollán. Co-Director: Santiago Soley Rimblas

ABSTRACT

TITLE: ARCHITECTURE AND IMPLEMENTATION OF A DATA ACQUISITION SYSTEM FOR GLOBAL NAVIGATION SATELLITE SYSTEMS (GNSS)*

AUTHOR: ACOSTA ORTIZ, DIEGO FERNANDO**.

KEYWORDS: GNSS, EMBEDDED SYSTEMS, GPS, and Galileo.

DESCRIPTION:

The Global Navigation Satellite Systems (GNSS), have opened up the possibility to create a lot of new applications and solve new problems. The GPS constellation (based on a constellation of 24 satellites placed in a medium earth orbit), is managed by the US Department of Defense but now alternative constellations had been created like the GLONASS (made by Russia, 24 satellites) or the Galileo constellation (made by European Union, 24 satellites still in deployment). Consequently nowadays we are heading to a GNSS multi-constellation world which space and opportunity for new applications based on geo-positioning. The availability of various constellations allows a higher accuracy and integrity with a bigger geo-strategic independence.

Consequently the signal reception and monitoring is increasingly important in high-tech sectors such as air, marine and earth navigation, oil prospecting, precision crop management, freight management, etc, in order to ensure reliability and availability.

Nowadays GNSS monitoring networks are based on expensive hardware and software but in a multi-constellation context, arises a necessity even bigger of this networks, taking into account the new possibilities which provides new low cost platforms because of its portability, efficiency, adaptability, etc., this project plans to implement a GPS/Galileo data receptor integrating low cost hardware and software which will allows to gather GNSS data for different research and developing areas.

* Degree Work

** Faculty of Physico-Mechanical Engineering. School of Systems Engineering. Director Raúl Ramos Pollán. Co-Director: Santiago Soley Rimblas

INTRODUCCIÓN

En los últimos años los sistemas de navegación global por satélite GNSS han cobrado importancia en diferentes áreas productivas e incluso en la vida cotidiana esto ha hecho que diferentes gobiernos como el de Estados Unidos, Rusia y recientemente los países miembros de la unión europea empiecen a financiar las iniciativas de investigación y desarrollo relacionadas con esta área. Como cualquier sistema de la vida real, un sistema GNSS está sujeto a una serie de incertidumbres, aquí es donde también cobra importancia la implementación de modelos adecuados que se ajusten a la realidad y permitan mejorar el cálculo de la posición de un receptor.

El modelamiento de las incertidumbres asociadas a la ionósfera no es un proceso trivial y está sujeto a muchas variables, entre estas la latitud sobre la superficie terrestre donde se encuentre el receptor ubicado ya que en zonas tropicales y ecuatoriales como Colombia, la ionósfera es susceptible a generar incertidumbres aún mayores que en otras partes para el cálculo de la posición de un receptor. Para resolver este problema se ha pensado en utilizar modelos de inteligencia artificial sobre los datos que se reciben sobre los receptores, pero para lograr esto primero hace falta implementar y masificar los sistemas de recolección de estos datos para sus estudios posteriores; Aquí es donde reside la utilidad de este trabajo de grado.

1. PLANTEAMIENTO Y JUSTIFICACION DEL PROBLEMA

Los Sistemas Satelitales de Posicionamiento Global (GNSS), han abierto la posibilidad de crear nuevas aplicaciones y solucionar nuevos problemas. Actualmente están surgiendo otros sistemas complementarios al GPS, conformado por una constelación 24 satélites en órbita media controlados por el Departamento de Defensa de EEUU [1], como la constelación GLONASS (Rusia, 24 satélites)[2] o la constelación Galileo (Europa, 30 satélites todavía en despliegue)[3]. Por tanto nos dirigimos hacia un mundo GNSS multi-constelación donde hay espacio y oportunidad para nuevas aplicaciones basadas en el geo-posicionamiento. La disponibilidad de varias constelaciones permite un posicionamiento de mayor precisión e integridad con una mayor independencia geoestratégica. Por tanto la recepción y el monitoreo de las señales es cada vez más importante en aplicaciones [4] como la navegación aérea, terrestre y marítima, la marcación de territorios, agricultura de precisión, etc. entre otros para así garantizar confiabilidad e integridad.

Al día de hoy las redes de monitoreo GNSS representan un alto costo por los componentes hardware y software requeridos. En un contexto multi-constelación, surge una necesidad aún mayor de estas redes. Teniendo en cuenta las nuevas oportunidades que brindan las plataformas hardware de bajo coste por su versatilidad, portabilidad, etc.

PILDO LABS es una organización de origen español que ha identificado todas estas necesidades que han ido surgiendo dentro del creciente mercado relacionado con GNSS y ha visto una oportunidad de invertir no solo en infraestructura y tecnología sino en talento humano e investigación para mejorar la calidad y las prestaciones de sus servicios destinados principalmente a la aviación civil. Como resultado de esto una de sus soluciones propuestas es la Pildobox.

El presente proyecto plantea implementar un receptor de datos GPS/Galileo integrando hardware y software de bajo costo que permitirá una recolección de datos GNSS para múltiples áreas de investigación y desarrollo; de una manera a análoga a la Pildobox pero explorado nuevas alternativas a nivel de hardware al ya previamente implementado y así mejorar las capacidades de procesamiento mientras se reduce su tamaño y/o costo.

2. OBJETIVOS

2.1. OBJETIVO GENERAL

- Diseñar e implementar un sistema de adquisición de datos de bajo coste para GNSS que integre elementos hardware de recepción de señales con un componente software para el procesamiento y análisis de los datos.

2.2. OBJETIVOS ESPECÍFICOS

- Definir los componentes de hardware de bajo coste para la recepción de señales GNSS.
- Integrar un componente de recepción de la señal (front end) sobre una plataforma genérica de hardware embebida.
- Integrar un flujo de trabajo para la recogida y almacenamiento de los datos.
- Validar el sistema completo con un caso de uso real.

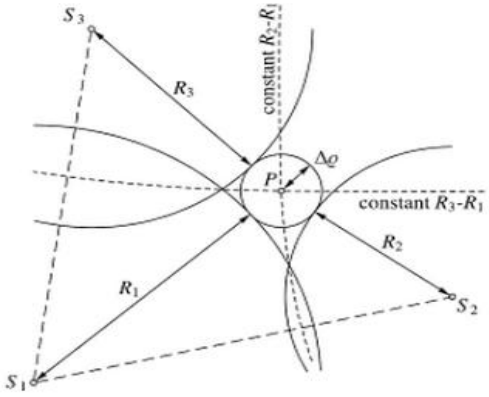
3. MARCO DE REFERENCIA

3.1. SISTEMAS DE NAVEGACIÓN GLOBAL POR SATÉLITE

Los Sistemas de Navegación Global por Satélite (GNSS) consisten en un conjunto de satélites orbitando la tierra, estos pueden ser usados para determinar la posición de cualquier dispositivo receptor de señales sobre la superficie terrestre siempre y cuando este receptor capture las señales de los satélites visibles. Para calcular la posición acertadamente, estos satélites están ubicados en órbitas muy estables llamadas MEOs (Medium Earth Orbits) a una altura aproximada de 22000 kilómetros, por otro lado el receptor tiene que recibir la señal de al menos 4 satélites. El receptor determina la distancia desde cada uno de los satélites tomando el tiempo que le tomó a la señal viajar desde el satélite hasta la antena receptora.

La distancia medida a partir de dos satélites indica que el receptor está ubicado en algún lugar del círculo en el que dos esferas se interceptan. Cada esfera contiene a uno de los dos satélites en su centro y su radio son las distancias satélite-receptor. Conociendo la distancia desde un tercer satélite corrige la posición de uno de los dos puntos donde el círculo intersecta con la tercera esfera. Uno de los puntos de intersección puede usualmente ser descartado, por ejemplo, este puede estar a miles de kilómetros arriba de la superficie terrestre.

Figura 1. Representación bidimensional de los pseudo-rangos.[1]



[1] Hofmann-Wellenhof, B., Lichtenegger, H., & Wasle, E. (2007). *GNSS–global navigation satellite systems: GPS, GLONASS, Galileo, and more*. Springer.

En la práctica un cuarto satélite es necesario para sincronizar el reloj del receptor con un tiempo estándar el cual está estrictamente respetado por los relojes a bordo de todos los satélites, utilizar mínimo cuatro satélites también resuelve la posición ambigua que ocurre con solo tres satélites. En general, a mayor número de satélites usado, mayor precisión posicional. Muchos receptores tienen canales para recibir señales hasta de 15 satélites [6].

3.2. COMPONENTES DE UN GNSS

Los Sistemas de Navegación Global por Satélite (GNSS) permiten determinar la posición de un objeto mediante triangulación, calculando la distancia de varios satélites a un receptor. Consta de tres segmentos [21]:

- **Segmento Espacial:** El conjunto de satélites orbitando la tierra que forman una constelación, los satélites transmiten señales que portan datos del reloj, parámetros de la forma de órbita de cada satélite en una determinada época.

- **Segmento de Terrestre (control):** Se encarga de vigilar el segmento espacial, estaciones de seguimiento en todo el planeta que se encargan de corregir y ajustar las órbitas de los satélites.

- **Segmento de Usuario:** Cualquier dispositivo receptor de señales sobre la superficie terrestre siempre que capture las señales de los satélites visibles. Cuenta con herramientas de hardware y software para hacer procesamiento de los datos para aplicaciones de posicionamiento.

3.3. ERROR DE ESTIMACIÓN

Una situación ideal incluye la perfecta sincronización de los satélites, órbitas, velocidad constante de propagación de la señal y ausencia de interferencias. Cualquier desviación de este escenario ideal puede contribuir a errores en la medida de las distancias. Algunas de las fuentes de error son:

3.3.1. Error en la ephemeris (Órbitas). Se refiere al error en la posición del satélite que es transmitido en el mensaje de navegación por intervalos, en GPS estos intervalos son de treinta segundos.

3.3.2. Error en relojes satelitales (Clock offset) Es el error de sincronización de los relojes del satélite y receptor con respecto al sistema GNSS

- **Reloj receptor:** Es calculado junto con las coordenadas del receptor, y no necesita ser modelado.

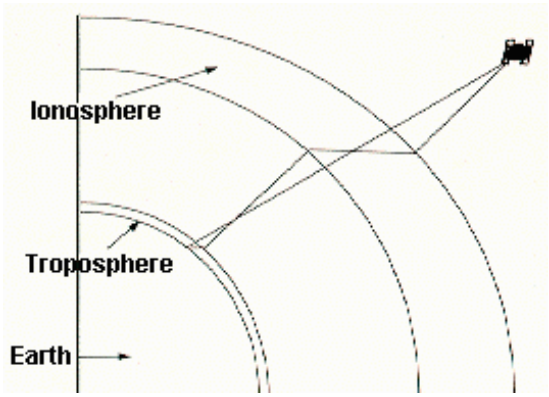
- **Reloj satelital**: Aunque los relojes presentes en los satélites son atómicos aún contienen error. Este consta de dos componentes: un error inherente al reloj que es fácilmente corregido con valores presentes en el mensaje de navegación más el error por efecto de la diferencia del potencial gravitacional en la tierra y en la ubicación del satélite.

3.3.3. Error atmosférico Las señales electromagnéticas experimentan cambios en la velocidad cuando pasan de un medio a otro debido a la refracción, por este motivo la atmósfera es una de las fuentes de error que más afecta el desempeño de los sistemas GNSS [17][18].

3.3.3.1. Ionósfera La ionósfera es la parte de la atmósfera terrestre ionizada permanentemente producto de la radiación solar, esta capa se extiende entre los 80 y 500 kilómetros de altura. La ionósfera causa una desviación en la señal resultando en el retraso en el tiempo que viaja entre el satélite y el receptor, afectando el cálculo de la distancia y por ende de la posición. Este error depende del ciclo solar y la latitud de la ubicación, de modo que las latitudes bajas se ven más perjudicadas que las medias y altas. Para mitigar este error se han implementado varias formas de hacerlo entre las que destacan los receptores de doble frecuencia que permiten mitigar totalmente el error ionosférico, pero estos receptores son muy costosos en el mercado. Con receptores de una sola frecuencia hace falta aplicar modelos ionosféricos entre los que destaca el modelo Klobuchar usado en GPS, en el sentido que algunos coeficientes dentro de los mensajes de navegación emitidos por los satélites son usados como parámetros del modelo según el segmento de tierra de GPS los va calibrando de manera continua. De esta manera puede estimar cerca del 50% del retraso ionosférico [19], sin embargo la ionosfera presenta cambios complejos que los modelos físicos no alcanzan detectar por lo cual los métodos basados en datos (Data Driven Models) están cobrando importancia [20].

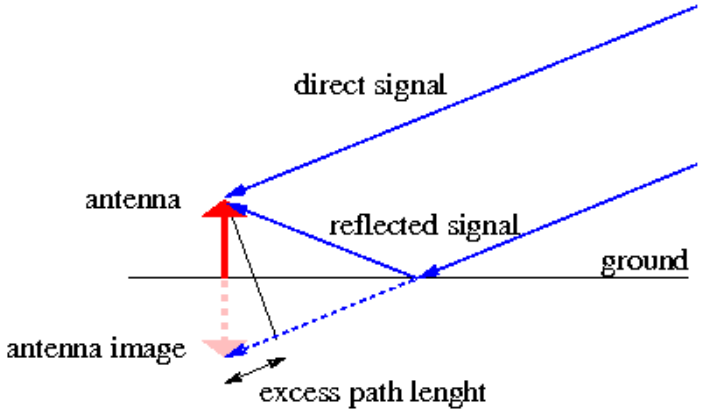
3.3.3.2. Troposfera Ubicada entre los 12 y 20 kilómetros de altura, el retraso de la señal generado por la troposfera dependerá de la temperatura, presión y humedad. Su error puede ser disminuido pues esta capa puede ser modelada fácilmente, ya que dicho error depende de las condiciones meteorológicas de la estación mencionadas anteriormente.

Figura 2. Retraso de radioseñales por el medio atmosférico.[2]



3.3.4. Error multicaminos (Multipath) Causado por las reflexiones de la señal, ya que hay varios caminos al receptor, este error suele ser pequeño, sin embargo es posible que estas reflexiones aumenten considerablemente en grandes en ciudades. Para mitigar el error multipath se debe mejorar la calidad de las antenas.

Figura 3. Diferencia entre la señal directa y la señal reflejada[3]



[2] (2010). Public Roads - Navigating the Future , Autumn 1995 -. Retrieved October 8, 2015, from https://www.fhwa.dot.gov/publications/publicroads/95fall/p95au4.cfm.
[3] (2012). Multipath - Navipedia. Retrieved October 8, 2015, from http://www.navipedia.net/index.php/Multipath.

La siguiente tabla muestra el error de cada factor sobre la precisión del sistema de posicionamiento [14], [15]:

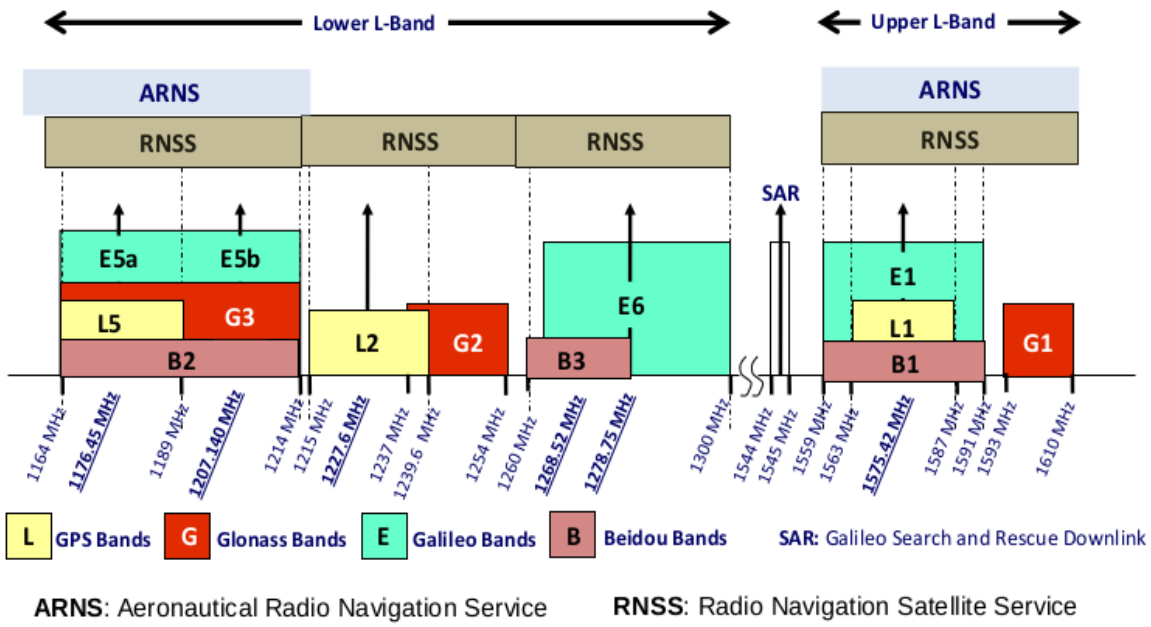
Tabla 1.

Tipo de Error	**Error [m] + / -**	**Segmento Asociado**
Órbitas	2.5	Señal en el espacio
Relojes Satelitales	1.5	Señal en el espacio
Ionosfera	5	Atmósfera
Troposfera	0.7	Atmósfera
Multicaminos	0.6	Receptor
Ruido en Receptor	0.7	Receptor
Total	11	

3.4. FRECUENCIAS ESPECTRALES EN GNSS

Elegir las bandas para alojar las frecuencias para las señales de sistemas GNSS fue un proceso complicado debido a que en el espectro electromagnético se encuentran viajando señales de televisión, radio, telefonía, microondas entre otras. La ITU es la agencia internacional que coordina el uso compartido del radio espectro, en la figura se puede observar cómo se encuentran repartidas las frecuencias para los sistemas GPS (Estados Unidos), GLONASS (Rusia), GALILEO (Unión Europea) y BEIDOU(China).

Figura 4, Bandas de frecuencia de los sistemas posicionamiento GNSS. Fuente ESA book[4]



Los sistemas de posicionamiento global transmiten sus señales usando señales UHF en la banda L del espectro (1 - 2 GHz), por ejemplo de la cual el Servicio de radionavegación aeronáutica tiene secciones reservadas (1559 – 1610 MHz) especialmente para aplicaciones de seguridad por la vida (Safety of Life - SoL) y no debe tener interferencia de otras señales GNSS, este servicio contiene la Banda L1 y L5 de GPS, E1 y E5 de Galileo, además de B1 y B2 de Beidou. Las demás bandas como L2 de GPS son usadas para servicios de posicionamiento y son más vulnerables a interferencia.

3.5. CARACTERÍSTICAS DE LAS SEÑALES GNSS

Las señales transmitidas por los satélites poseen algunas características que son objeto de análisis y procesado por parte de los receptores en el cálculo la posición del dispositivo que las recepciona.

[4] J. Sanz Subirana, J.M. Juan Zornoza and M. Hernández-Pajares (2013). GNSS Data Processing, Vol. I: Fundamentals and Algorithms (pp. 31), ESA TM-23/ESA Publications.
http://www.esa.int/About_Us/ESA_Publications/ESA_TM-23_GNSS_DATA_PROCESSING.

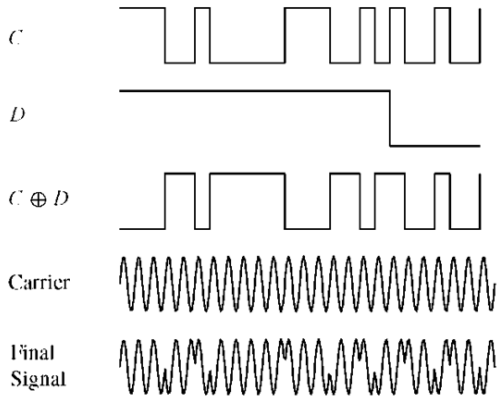
3.5.1. Señal carrier (portadora) Radio frecuencia sinusoidal en la banda L (1559 – 1610 MHz) del espectro electromagnético. La frecuencia fundamental es *f0 =10,23 Mhz, e*n GPS dos señales son creadas a partir de esta multiplicando por 154 para el canal L1 *f1=1575.42* Mhz, longitud de onda 19 cm; y se multiplica por 120 para el canal L2 con frecuencia *f2=1227.6 MHz*, longitud de onda 24.4 cm. En la actualidad existen otros canales como L2C y L5 que se mencionaron anteriormente, para tener más precisión en posicionamiento y estimación del error. La información se codifica como bits en un proceso que se denomina modulación de la fase.

3.5.2. Código El código son secuencias de unos y ceros que permiten al receptor determinar el tiempo de la señal viajando desde el satélite al receptor. Estas secuencias se denominan códigos PRN (Pseudo Random Noise) y se modulan sobre las señales carrier:

- **Código civil (C):** Contiene 1023 bits que se repiten cada milisegundo, la "longitud de chip" o distancia física entre transiciones binarias es 293 metros, de modo que la duración de cada chip es de 1 microsegundo, el código C se modula únicamente en la frecuencia L1 [21].

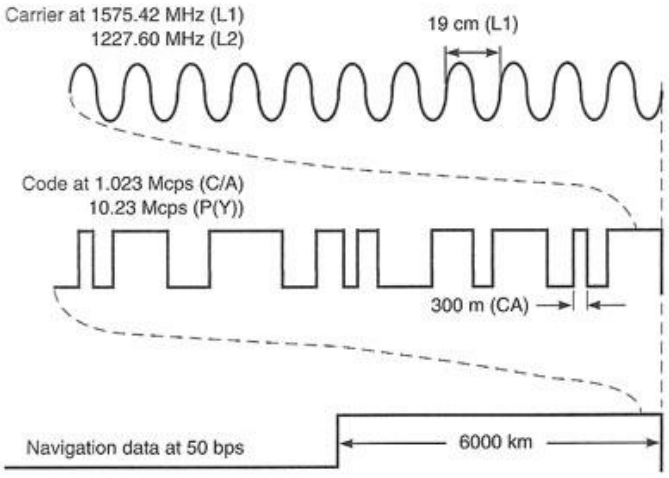
- **Código de precisión (P):** De uso militar y para civiles autorizados para una posición más precisa pues el tiempo del satélite transmitido tiene hasta 10 veces más resolución que el código C, con una frecuencia de 10 Mbps y una longitud de onda o chip de 29,31 metros. Se modula en ambas carriers L1 y L2. Puede ser encriptado en un proceso conocido como *anti-spoofing* [21].

Figura 5. Gráfico de modulación de una señal GNSS. [5]



3.5.3. Datos de navegación Mensaje codificado binario que provee información de la órbita de los satélites entre estos datos se encuentran parámetros de Kepler, correcciones del reloj entre otros datos como coeficientes para el modelamiento ionosférico. El mensaje de navegación NAV al igual el código pueden ser modulados sobre L1 y L2 a una tasa muy lenta de 50 bps, consiste en una secuencia de 1500 bits y toma 30 segundos en ser transmitida.

Figura 6. Señal portadora en L1, código y datos de navegación. [6]



[5] (2012). GPS Signal (GPS and Galileo Receiver) Part 2. Retrieved October 9, 2015, from http://what-when-how.com/a-software-defined-gps-and-galileo-receiver/gps-signal-gps-and-galileo-receiver-part-2/.
[6] (2013). Part 2: The Origins of GPS, Fighting to Survive : GPS World. Retrieved October 9, 2015, from http://gpsworld.com/origins-gps-part-2-fighting-survive/.

3.6. FORMATOS ESTÁNDAR GNSS

En el proceso de captura de los datos GNSS pueden intervenir diferentes protocolos utilizados por parte de los receptores, como resultado de estos protocolos, los datos generados pueden estar en formatos estándares regulados por normas internacionales o en formatos propietarios establecidos por los fabricantes de algún receptor determinado.

Existen varios formatos de mensajes, entre los más conocidos se encuentran:
ANTEX: The Antenna Exchange Format, BINEX: Binary Exchange Format, IONEX: The IONosphere map EXchange format, RINEX: Receiver Independent Exchange Format, NMEA 0183, entre otros. A continuación se describirán los dos formatos de mensajes de interés para este proyecto:

3.6.1. Formato RINEX (Receiver Independent Exchange Format) [22],[23] fue desarrollado por el Instituto Astronómico de la Universidad de Berne para facilitar el intercambio de datos GPS, los primeros mensajes fueron generados durante la campaña EUREF 89, contaba con 60 receptores GPS de diferentes fabricantes. El aspecto más relevante durante su desarrollo fue definir el conjunto de observables. Los observables o datos para computar el posicionamiento más fundamental son:

3.6.1.1. Pseudo-rango Es la distancia del satélite al receptor calculada teniendo en cuenta factores como el tiempo, donde el tiempo entre la emisión y recepción de la señal.

t= tr-ts Donde tr: tiempo del receptor, ts: tiempo de emisión del satélite

Este es idéntico para las mediciones de fase, rango y todos los satélites observados en una misma época.

Entonces teniendo en cuenta el código y los desfases entre los respectivos relojes junto con otros factores como la atmósfera que retrasa el tiempo causando una estimación mayor que la distancia real. El pseudo-rango se mide en unidades de metros.

PR= p + c(clock offset receptor- clock offset satélite)+ otros retrasos*

3.6.1.2. Fase: La fase de la señal carrier medida en número de ciclos. Usada para medir el rango entre satélites y receptores en aplicaciones que necesitan alta precisión. Existe siempre una ambigüedad en el número de ciclos medidos que se resuelve por métodos avanzados de posicionamiento, gracias a los cuales se pueden obtener precisiones de posicionamiento del orden de los centímetros.

3.6.1.3. Doppler El efecto doppler es ampliamente conocido en acústica, denota el cambio de frecuencia de una onda de sonido: un objeto respecto al observador, a medida que este se acerca el sonido parece más agudo que cuando se aleja. La fase puede ser utilizada para medir tiempos y desarrollar un modelo de pseudo rangos basados en la observación de fases [21]. Los receptores guardan una señal réplica del satélite que es combinada con la real emitida, estas son mezcladas resultando en una señal "beating" es igual a la fase de referencia sin la fase de la carrier. Sin embargo hay un inconveniente en este método ya que la fase contiene una ambigüedad por un número de ciclos N.

$$\Phi S(T) = \varphi(T) - \varphi S(T) - N$$

$\Phi S(T)$: Fase de la señal beating
$\varphi(T)$:Fase réplica
$\varphi S(T)$:Fase transmitida
N: Número entero de ciclos, ambigüedad.

El formato RINEX se ha optimizado con los años pero por lo general se transmiten 2 tipos de mensajes: De navegación (parámetros de órbita) y de observación. Cada uno se divide en encabezado y datos. La sección de encabezado contiene la información de las etiquetas de los campos o columnas que suelen ser de 61 a 80 para cada línea. La estructura permite utilizar un mínimo de espacio independiente del número de tipos de observaciones que el receptor capta. La figura 7 y 8 muestran ejemplos de los formatos en la versión 2.11.

Figura 7. Mensaje de observación RINEX v. 2.11.[7]

Observation RINEX 2.11 Format

```
     2.11           OBSERVATION DATA    M (MIXED)           RINEX VERSION / TYPE
BLANK OR G = GPS,  R = GLONASS,  E = GALILEO,  M = MIXED    COMMENT
gLAB                gAGE                17-MAR-10 12:14    PGM / RUN BY / DATE
EXAMPLE OF A MIXED RINEX FILE                               COMMENT
MRKR                                                        MARKER NAME
9080.1.34                                                   MARKER NUMBER
gAGE                UPC: Technical University of Catalonia  OBSERVER / AGENCY
THIS FILE IS PART OF THE gLAB TOOL SUITE                    COMMENT
FILE PREPARED BY: ADRIA ROVIRA GARCIA                       COMMENT
PLEASE EMAIL ANY COMMENT OR REQUEST TO:   glab @ gage.es    COMMENT
IR2200716006        ASHTECH UZ-12       CQ00                REC # / TYPE / VERS
482                 AOAD/M_T        NONE                    ANT # / TYPE
   4789028.4701    176610.0133   4195017.0310               APPROX POSITION XYZ
         0.9030         0.0000         0.0000               ANTENNA: DELTA H/E/N
     1     1                                                WAVELENGTH FACT L1/2
     1     2     3   G14   G18   G19                        WAVELENGTH FACT L1/2
     7    L1    L2    P1    P2    C1    S1    S2           # / TYPES OF OBSERV
    30.000                                                  INTERVAL
  2010     3     5     0     0     0.0000000     GPS       TIME OF FIRST OBS
  2010     3     5    23    59    30.0000000     GPS       TIME OF LAST OBS
     1                                                      RCV CLOCK OFFS APPL
    15                                                      LEAP SECONDS
    14                                                      # OF SATELLITES
   G07   815   815   815   815   815   815   815           PRN / # OF OBS
   G09   246   246   246   246   246   246   246           PRN / # OF OBS
   G12   687   687   687   687   687   687   687           PRN / # OF OBS
   G13   762   762   762   762   762   762   762           PRN / # OF OBS
   G15   454   454   454   454   454   454   454           PRN / # OF OBS
   G20   599   599   599   599   599   599   599           PRN / # OF OBS
   G21   636   636   636   636   636   636   636           PRN / # OF OBS
   G26   210   210   210   210   210   210   210           PRN / # OF OBS
```

Figura 8. Mensaje de Navegación para GPS en formato RINEX v.2.11. [8]

GPS Navigation RINEX 2.11 Format

```
     2.11           N: GPS NAV. MESSAGE                     RINEX VERSION / TYPE
gLAB                gAGE / UPC          17-MAR-10 12:14    PGM / RUN BY / DATE
EXAMPLE OF A NAVIGATION RINEX FILE                          COMMENT
THIS FILE IS PART OF THE gLAB TOOL SUITE                    COMMENT
FILE PREPARED BY: ADRIA ROVIRA GARCIA                       COMMENT
PLEASE EMAIL ANY COMMENT OR REQUEST TO:   glab @ gage.es    COMMENT
     0.1676D-07     0.2235D-07    -0.1192D-07    -0.1192D-07ION ALPHA
     0.1208D+06     0.1310D+06    -0.1310D+06    -0.1966D+06ION BETA
   0.133179128170D-06   0.107469588780D-12   552960     1025DELTA-UTC: A0,A1,T,W
    15                                                      LEAP SECONDS
                                                            END OF HEADER
 6 99  9  2 19  0  0.0 -.839701388031D-03 -.165982783074D-10  .000000000000D+00
     .910000000000D+02  .934062500000D+02  .116040547840D-08  .162092304801D+00
     .484101474285D-05  .626740418375D-02  .652112066746D-05  .515365489006D+04
     .409904000000D+06 -.242143869400D-07  .329237003460D+00 -.596046447754D-07
     .942817490922D+00  .326593750000D+03  .206958726335D+01 -.638312302555D-08
     .307155651409D-09  .000000000000D+00  .102500000000D+04  .000000000000D+00
     .000000000000D+00  .000000000000D+00  .000000000000D+00  .910000000000D+02
     .406800000000D+06  .000000000000D+00
13 99  9  2 19  0  0.0  .490025617182D-03  .204636307899D-11  .000000000000D+00
     .133000000000D+03 -.963125000000D+02  .146970407622D-08  .292961152146D+01
    -.498816370964D-05  .200239347760D-02  .928156077862D-05  .515328476143D+04
     .414000000000D+06 -.279396772385D-07  .243031939942D+01 -.558793544769D-07
     .986307770581D+00  .271187500000D+03 -.232757915425D+01 -.619632953057D-08
    -.785747015231D-11  .000000000000D+00  .102500000000D+04  .000000000000D+00
     .000000000000D+00  .000000000000D+00  .000000000000D+00  .389000000000D+03
     .410400000000D+06  .000000000000D+00
```

[7] (2010). Observation Rinex v2.11.html - gAGE Learning Material - UPC. Retrieved October 12, 2015, from http://gage14.upc.es/gLAB/HTML/Observation_Rinex_v2.11.html.

[8] (2010). Observation Rinex v2.11.html - gAGE Learning Material - UPC. Retrieved October 12, 2015, from http://gage14.upc.es/gLAB/HTML/Observation_Rinex_v2.11.html.

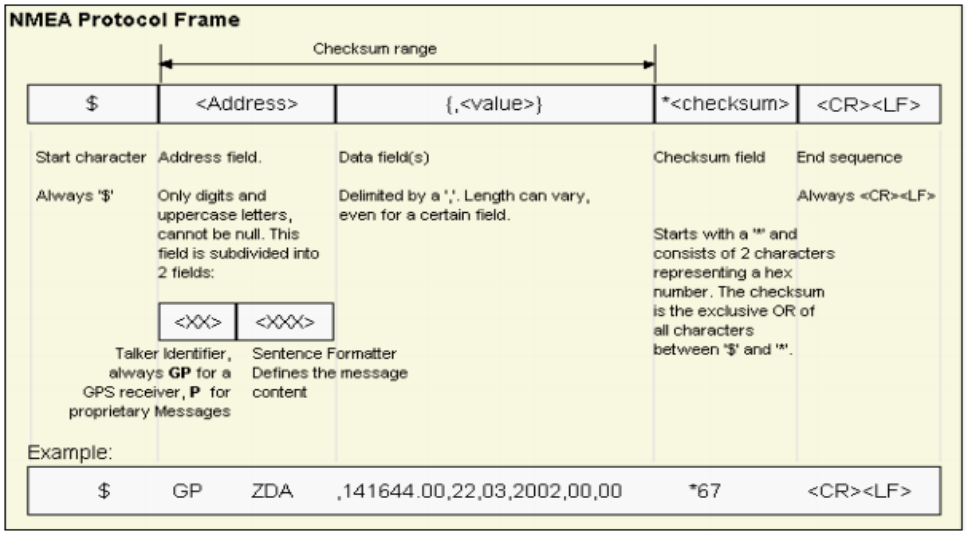
Otro ejemplo de datos capturados en formato RINEX se puede encontrar en este repositorio: **https://raw.githubusercontent.com/rramosp/20152.gnss.uis/master/data/bogt0010.15o**

3.6.2. Formato NMEA 0183

El nombre NMEA[30] National Marine Electronics Association, hace referencia a la entidad gubernamental norteamericana encargada de establecer estándares para la comunicación entre dispositivos marítimos electrónicos. Uno de los estándares más reconocidos es el NMEA 0183, este inicialmente fue establecido para propósitos de navegación marítima, pero con el paso de los años ha ganado gran aceptación para aplicaciones diferentes a las marítimas [24].

Cabe señalar que estos datos NMEA ofrecen información sobre la posición del receptor más no ofrece gran detalle sobre las señales GNSS recibidas desde los satélites. Este formato es usado por chips GNSS que ofrecen directamente la posición del receptor para aplicaciones en la que los observables de pseudo.rango y fase no son tan relevantes. Estos chips son los que se usan en receptores de gran consumo (celulares, tablets, etc..).

Figura 9, Trama Protocolo NMEA 0183 Versión 4.0[9]



[9] (2014). u-blox M8 receiver description and protocol specification (pp. 68). Retrieved January 30, 2015, from http://www.u-blox.com/images/downloads/Product_Docs/u-bloxM8_ReceiverDescriptionProtocolSpec_(UBX-13003221)_Public.pdf.

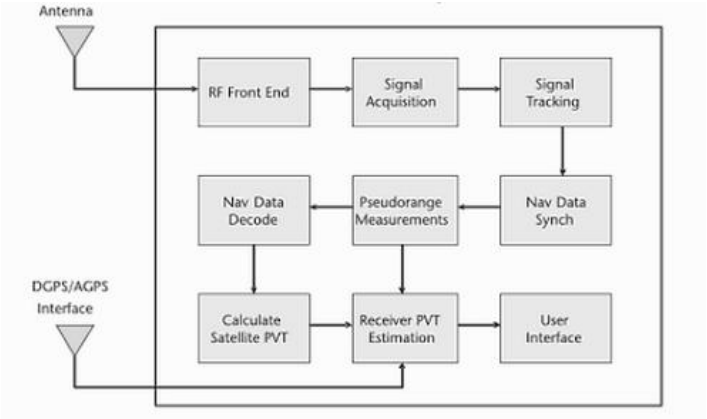
Ejemplo de una salida NMEA, tomada directamente desde un receptor, los datos señalados en rojo corresponden a la posición del receptor.

```
$GPGGA,204443.00,0708.4572,N,07307.3596,W,1,06,02.0,1005.5,M,-4.4,M,,*62
$GPRMC,204443.00,A,0708.4572,N,07307.3596,W,00.00,058.8,290115,,,A*43
$GPGSV,3,1,11,08,00,000,33,14,33,012,25,18,48,140,36,22,88,093,36*78
$GPGSV,3,2,11,25,34,044,33,29,00,000,27,31,47,316,33,33,25,094,*79
$GPGSV,3,3,11,35,13,267,,48,22,265,,51,50,259,*4C
$GLGSV,1,1,01,65,48,114,26*5B
$GNGSA,A,3,14,18,22,25,31,,,,,,,,04.7,02.0,04.2*12
$GNGSA,A,3,65,,,,,,,,,,,,04.7,02.0,04.2*18
$PORZD,A,009.0*35
```

3.7. RECEPTORES CONVENCIONALES

Un receptor GNSS es una combinación de hardware y software capaz de recibir señales de diferentes satélites GNSS y procesarlas en una información útil de posición, velocidad y tiempo.
Una arquitectura convencional de un receptor GNSS consiste en una antena conectada a una serie de Circuitos Integrados de Aplicación Específica (ASICs), todos controlados por un procesador central. En conjunto, esta combinación de hardware y software desempeña todas las funciones del receptor. Una configuración común consiste en una antena y un chip RF front-end, que a su vez alimenta las señales muestreadas en los chips de correlaciones digitales. Todos los pasos subsecuentes del procesado de la señal son controlados a través del uso de interfaces desde el ASIC al procesador central. El resultado es una estimación de la posición, velocidad y tiempo (PVT) del receptor [7].

Figura 10. Diagrama en bloque de un receptor GNSS típico[10]



Los receptores comunes ofrecen una posición YA CALCULADA (con los errores corregidos con modelos estándar) en formato NMEA [12], [13], pero en este proyecto los datos deseados serán capturados con el front-end ya que este brinda el RAW DATA, es decir, el mensaje recibido directamente desde el satélite con los datos de la posición del satélite, la distancia estimada al mismo, la hora entre otros datos; de tal manera que posibilita el uso de nuestra solución (de bajo coste) en estudios sobre GNSS, como por ejemplo, estudios sobre las afectaciones de la ionosfera, modelos de corrección de errores, aplicaciones de posicionamiento preciso (posicionamiento diferencial, uso de las fases de las señales del receptor y del satélite), etc.

3.8. SISTEMAS EMBEBIDOS

Un sistema embebido (SE) está definido como un sistema con una fuerte integración entre componentes de hardware y software, construido para desempeñar una función dedicada. La palabra embebida refleja el hecho que esos sistemas son usualmente una parte integral de un sistema mayor.[8]

[10] Gleason, S., & Gebre-Egziabher, D. (Eds.). (2009). *GNSS applications and methods*. Artech House.

Las principales características de un gran número de sistemas embebidos es que, respecto a los ordenadores convencionales, los SE usualmente son de tamaño reducido, poseen bajo consumo energético, un costo reducido de fabricación. Es por estas cualidades, hoy día, son utilizados masivamente en un gran número de dispositivos y allanando así el terreno para nuevas tendencias tecnológicas como el internet de las cosas.

3.9. COMPILACIÓN

El término compilación denota la conversión de un algoritmo expresado en un lenguaje fuente (lenguaje de alto nivel o lenguaje humano) a un algoritmo equivalente expresado en un lenguaje de máquina [10]. Usualmente pero no siempre, el lenguaje fuente es de alto nivel como Pascal o Java y el lenguaje objetivo es de bajo nivel, como código assembler o lenguaje puro de máquina.
Al igual que cualquier ordenador convencional, un sistema embebido necesita que todo el código, que se desea ejecutar sobre el sistema, sea convertido a lenguaje de máquina, para ello se puede compilar el código directamente sobre el hardware embebido siempre y cuando exista una capa intermedia con sistema operativo y un compilador sobre el hardware, de otro modo el código debe ser compilado utilizando herramientas de compilación cruzada, es decir el código que se desea ejecutar sobre el hardware objetivo debe ser compilado sobre otra arquitectura usualmente diferente a la del sistema embebido pero que emula a esta última.

3.9.1. Compilación cruzada La compilación cruzada permite construir en una plataforma un binario que se ejecute en otra plataforma, la plataforma que genera el ejecutable *build platform* y la que la *host platform* es donde espera que se ejecute [25]. Algunas ventajas de la compilación cruzada de acuerdo a Landley[26] son:

- **Velocidad:** Ya que la plataforma host suele ser mucho más lenta, pues estas son diseñadas para ser de bajo costo y consumo.

- **Capacidad:** Los dispositivos embebidos no tienen los recursos necesarios para la compilación porque no tiene suficiente memoria ni espacio de almacenamiento en disco para construir grandes y complicados paquetes.

- **Disponibilidad:** Llevar linux a una plataforma de hardware que nunca lo ha ejecutado requiere un compilador cruzado.

- **Flexibilidad:** Una distribución de Linux consiste en cientos de paquetes, pero en una compilación cruzada puede depender de la distribución del host para la mayoría de tareas. Se concentra en construir los paquetes a ser desplegados en el sistema embebido, sin gastar tiempo satisfaciendo pre-requisitos.

4. ESTADO DEL ARTE

En el mercado se pueden encontrar receptores GNSS multi constelación de una sola frecuencia y de doble frecuencia. En el caso de los receptores de una sola frecuencia, estos operan sobre la banda de L1/E1 para GPS/Galileo y la banda G1 para GLONASS; en cambio los receptores doble frecuencia, pueden capturar datos modulados en dos o más frecuencias, por ejemplo para GPS se usan frecuencias en L1, L2 y recientemente en L5; pero se tiene que señalar que la frecuencia en L2 está usualmente limitada para propósitos militares por el gobierno de EEUU. Por lo tanto, para aplicaciones civiles los receptores usados son de una sola frecuencia; estos receptores son mucho más pequeños y baratos en comparación con los receptores multifrecuencia cuyos precios multiplican varias veces los de receptores de una sola frecuencia. Los receptores más comunes son usados usualmente ofrecen datos en formato NMEA con una posición ya calculada, otros receptores permiten obtener datos *raw* o binarios que pueden ser convertidos a formatos estándares como el RINEX, estos últimos son objeto de interés para realizar estudios sobre señales GNSS.

Usualmente los receptores que ofrecen raw data tienen que ser conectados a computadores desde los cuales se han de controlar varios parámetros dentro las campañas de medición como por ejemplo la frecuencia de muestreo. También otro de los motivos para usar el receptor en conjunto con un computador es que los datos pueden ser procesados en tiempo real o ser almacenados para estudios posteriores.

Finalmente surge la necesidad de implementar un dispositivo standalone para realizar monitoreo de señales GNSS, los mismos fabricantes de receptores se han puesto en esta tarea pero las soluciones ofrecidas son muy costosas respecto al valor de receptor en sí, además estas soluciones carecen de la versatilidad ofrecida por un ordenador multipropósito.

4.1. PILDOBOX

La Pidobox[11] es un producto implementado por una organización europea de origen español llamada PILDO LABS, este producto de bajo costo es utilizado como un componente clave que interviene en varios de los servicios ofrecidos por Pildo, estos están principalmente basados en la nube y son usados para dar soporte a diferentes tipos de operaciones en las que interviene el monitoreo de señales GNSS, principalmente aquellas operaciones relacionadas con aviación civil. La Pildobox consiste en un receptor multi-constelación GNSS, un Sistema Embebido y unas capas de software para la comunicación con el receptor, el almacenamiento de datos, además de la sincronización y envío de datos a la nube. La primera versión de la Pildobox utiliza como SE una Raspberry Pi, pero se ha visto la necesidad de que para la nueva versión se implemente algún SE alternativo que ofrezca mejores prestaciones a la Raspberry, cuente con mayor capacidad de procesamiento y menor tamaño, sobre esto se ha concentrado gran parte del objeto de estudio y trabajo de este proyecto.

A continuación se describirán algunos de los componentes software que intervienen en la Pildobox y que se pueden aprovechar para la nueva versión de la Pildobox propuesta en el marco de este proyecto.

4.2. RTKLIB

RTKLIB es un software desarrollado por Tomoji Takasu consiste en una serie de aplicaciones y librerías Open Source para el cálculo estándar y preciso del posicionamiento a través de GNSS. Dentro de sus características se destacan:

- Soporte a algoritmos para posicionamiento estándar y preciso con GPS, GLONASS, Galileo, QZSS, BeiDou y SBAS.
- Soporte para procesado en tiempo real y post procesado de señales GNSS.
- Soporte a varios formatos, estándares y protocolos usados en GNSS.
- Soporte a varios mensajes propietarios de receptores GNSS.
- Soporte a comunicación externa vía serial, TCP (IP, NTRIP, local log file y FTP/HTTP.

[11] (2005). Pildo Labs: Home. Retrieved October 8, 2015, from http://pildo.com/.

- Soporte a muchas funciones, librerías y APIs para procesado de datos GNSS.

Información más detallada se puede encontrar en el sitio web. www.rtklib.com

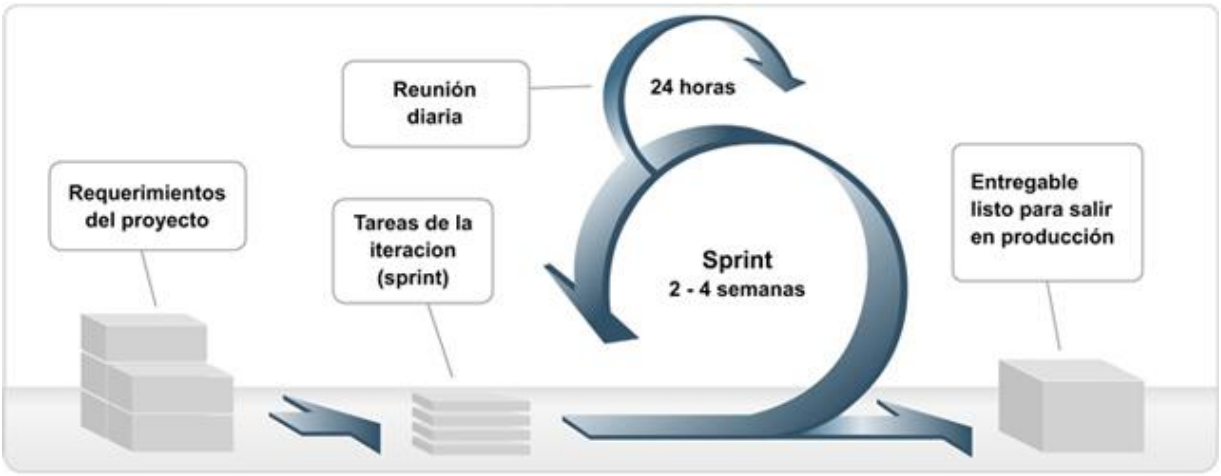
4.3. PAF WEB APPLICATION

Esta es una web application creada sobre el framework de Pildo "PAF" (Pildo Application Framework) que a su vez está basado en el framework Grails para el desarrollo de aplicaciones web. Como su nombre lo indica, esta es la capa de software que reside en el servidor y es utilizada para la gestión de la recepción y almacenamiento de los datos recibidos por cada una de las Pildobox.
Para llevar a cabo esta operación existe una aplicación llamada Data Collector también desarrollada por Pildo que corre sobre la PildoBox, esta es una aplicación desarrollada en Java por Pildo, fue pensada para integrarse con los servidores de Pildo y realizar el envío de datos.

5. METODOLOGÍA

Las metodologías ágiles usualmente son aplicadas netamente al desarrollo de software, a pesar de que este proyecto posee una serie de características particulares que lo diferencian de un proyecto de tradicional de desarrollo de software comparte algunas de las necesidades y dificultades que caracterizan a estos mismos, como por ejemplo la necesidad de una metodología flexible y adaptable ya que las hipótesis planteadas, soluciones formuladas o dicho de otro modo, requisitos del sistema propuesto, pueden estar sujetos a incertidumbre como consecuencia de los cambios repentinos en las dependencias de hardware y software. Debido a lo anterior surge la necesidad de llevar a cabo iteraciones (sprints) incrementales sobre el modelo básico del Ciclo de Vida del Desarrollo de un Sistema (SDLC). De esta manera, a medida que se Analiza, Diseña, Desarrolla, Implementa y Validan pequeños atributos y funcionalidades del sistema, se puede garantizar la operatividad de cada uno de los componentes implementados a medida que el proyecto avanza.

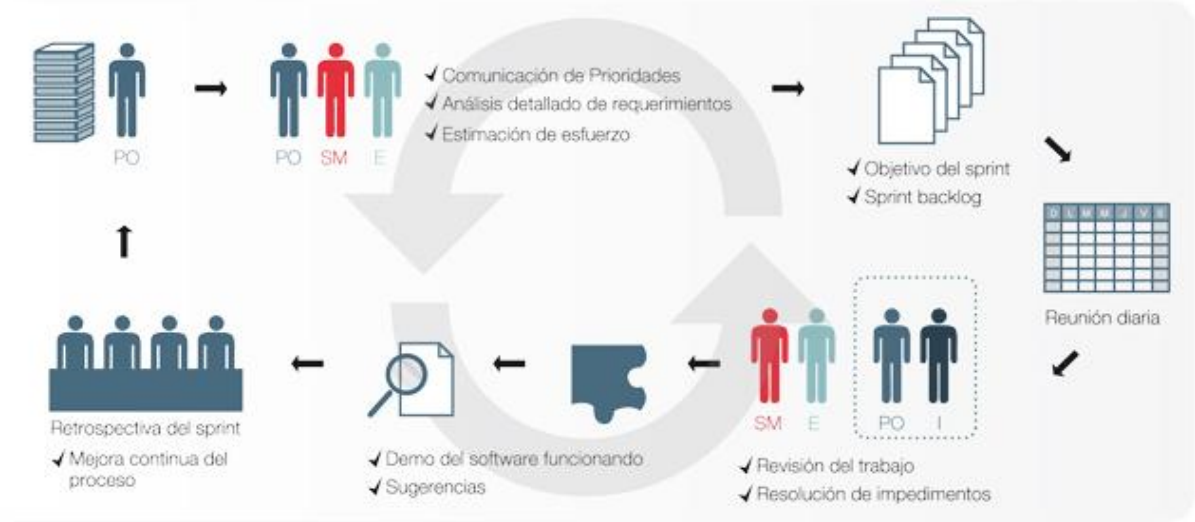
Figura 11. Diagrama de funcionamiento para la metodología ágil de desarrollo de proyectos SCRUM[12]



[12] (2010). Reserv IT Solutions - Servicios. Retrieved October 14, 2015, from http://www.reserv.com.ar/servicios.php. http://www.reserv.com.ar/metodologia.php

Teniendo en cuenta todo lo mencionado anteriormente se ha optado por implementar una adaptación de la metodología Scrum. La metodología Scrum[16] consiste en una serie de sprints, cada uno de estos sprints consiste en una iteración que usualmente tienen una duración de 30 días, a medida que se lleva a cabo cada sprint, una serie de retroalimentaciones son llevadas a cabo diariamente para ejercer un control en el desarrollo del sprint, al final de cada sprint un pequeño módulo o funcionalidad del sistema es entregado y validado. En la metodología Scrum existen unos roles identificados en la estructura del llamado equipo Scrum, estos son el *Product Owner* (dueño del producto), el *Development Team* (equipo de desarrollo) y el *Scrum Master* (líder de desarrollo) y se encuentran representados en la figura 12.

Figura 12. Roles en la metodología SCRUM.[13] *PO:* Product Owner *SM:* Scrum Master *E:* Equipo que construye el proyecto.



En la adaptación de la metodología para este proyecto en particular se han llevado a cabo al menos 2 sesiones de retroalimentación semanalmente, con una duración en promedio de 1 hora y 30 minutos cada una; los sprints fueron contemplados en periodos flexibles entre 3 y 5 semanas cada uno. Respecto a los roles asignados, el director de proyecto ha desempeñado el rol de Scrum Master, el codirector de proyecto ha actuado como el Product Owner y el estudiante autor del desarrollo del proyecto desempeñó el rol correspondiente al equipo de desarrollo.

[13] (2010). Reserv IT Solutions - Servicios. Retrieved October 14, 2015, from http://www.reserv.com.ar/servicios.php.

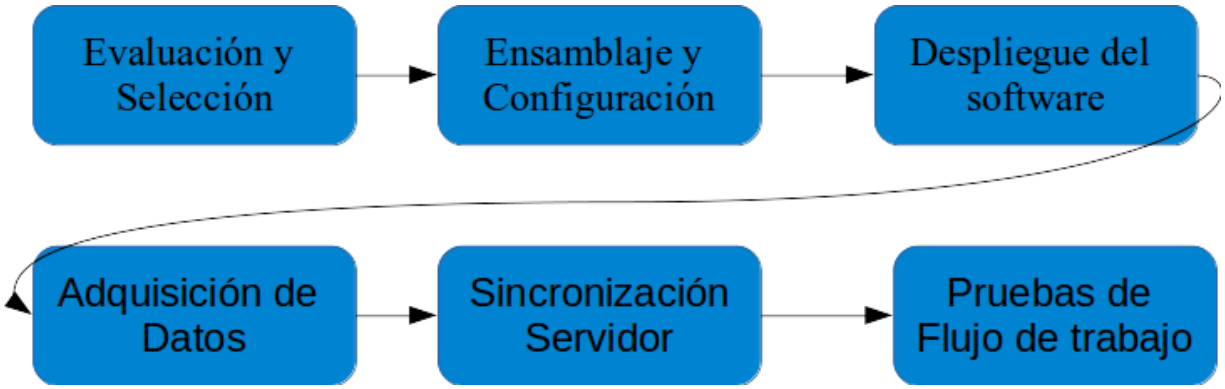
Los pasos básicos del SDLC serán recorridos, y la documentación será generada, en la medida que cada una de las iteraciones correspondientes a los siguientes sprints sean ejecutadas:

1. Evaluación y selección de hardware.
2. Configuración inicial para el uso del SE y el Receptor GNSS. (Incluye comunicación básica con el receptor).
3. Configuración y despliegue del software sobre el SE. (Despliegue del entorno de creación distribución propia basada en Yocto Linux y trabajo sobre el compilador cruzado).
4. Validación de la adquisición de datos GNSS sobre el SE.
5. Validación de la sincronización del SE con servidor.
6. Pruebas del flujo de trabajo del sistema completo integrado.

Como es contemplado dentro de la metodología Scrum, durante el transcurso del proyecto estos sprints fueron sujetos sujetos a algunas modificaciones respecto al plan original.

6. DESARROLLO

Figura 13. Gráfico de los Sprints.



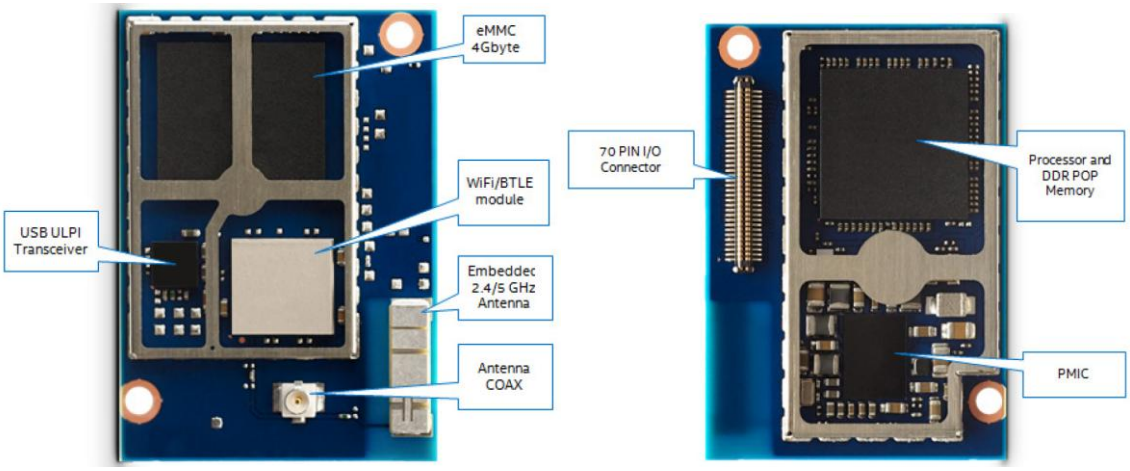
6.1. EVALUACIÓN Y SELECCIÓN DE HARDWARE.

Teniendo en cuenta las características principales de la PildoBox es deseable que la siguiente versión de esta cuente con un tamaño aún más reducido pero sin perder potencia de cálculo ni las ventajas que ofrece tener instalado un sistema operativo totalmente funcional sobre un sistema embebido.

Siguiendo la filosofía del paradigma del internet de las cosas "IOT" (Internet of Things) que busca dotar de capacidad de procesamiento a cualquier dispositivo que usamos en nuestra vida cotidiana, se buscó utilizar alguna de las nuevas plataformas embebidas desarrolladas para este objeto se planteó usar para este proyecto una tarjeta Intel® Edison[27] como una alternativa al SE Raspberry Pi.

La Intel® Edison es un sistema embebido basado en un módulo SoC (System-On-Chip) es decir, la Intel® Edison posee sobre un solo chip todos los componentes básicos como lo son: Procesador, Memoria Primaria, Memoria Secundaria, Microcontroladores, entre otros.

Figura 14. Intel® Edison.



Fuente: Intel

Al comienzo de este proyecto no se contaba con una Intel® Edison para iniciar el desarrollo e implementación del sistema así que se optó por usar una tarjeta Intel® Galileo mientras la adquisición de la Edison era gestionada. Otro de los motivos por los cuales se usó en este proyecto la placa Intel Galileo fue el hecho de que esta placa de desarrollo es predecesora de la Intel® Edison, por lo tanto Intel® ha garantizado que ambos SE posean una gran compatibilidad entre sí en donde la más llamativa es que ambos dispositivos tienen implementados por defecto el mismo sistema operativo Linux Yocto.[28]

6.1.1. Comparación entre SE

Tabla 2. Comparación de características generales de los SE utilizados.

	Raspberry Pi - Model B	Intel® Galileo	Intel® Edison
SoC	Broadcom BCM2835 - Single Core	Intel® Quark X1000 - Single Core	"Tangier" Atom SOC 22nm - Dual Core Intel® Atom™ x86 & 32-bit Intel® Quark MCU 100MHz.
Arquitectura	ARMv6	i586	i686
CPU	ARM1176	Intel® X1000	Intel® Atom™
Reloj	700 MHz	400 MHz	500 MHz por núcleo
RAM	512 MB	256 MB	1024 MB LPDDR3 POP memory (2 channel 32bits @ 800MT/sec)
GPU	Broadcom VideoCore IV	No	No
Almacenamiento Interno	Ninguno	8 MB	4 GB eMMC
Almacenamiento Externo	Tarjeta SD	Tarjeta MicroSD	Expansión tarjeta MicroSD
Ethernet	10/100M Ethernet	10/100M Ethernet	No
WiFi	No	Expansión Mini-PCI Express.	Broadcom* 43340 802.11 a/b/g/n; Dual-band (2.4 & 5 GHz), Bluetooth 4.0
Dimensiones	85.6mm x 56mm	106.7mm x 71.1mm	35.5 × 25.0 × 3.9 mm

Comparando los SE se puede observar que de entrada la placa Intel® Edison ofrece una serie de prestaciones interesantes como por ejemplo ofrece la posibilidad de conectarse a redes de manera inalámbrica gracias a su módulo integrado con interfaz Wi-Fi y bluetooth 4.0, lo cual deja abierta la

posibilidad de que la PildoBox sea totalmente inalámbrica como lo es un Smartphone convencional. La Intel Edison también puede ser usada con bloques de expansión.[14]

6.1.2 Receptor GNSS

El sensor elegido es el NV08C-CSM[29] ya que este es un receptor multi-constelación, es decir, recibe señales de diferentes sistemas GNSS como lo son GPS, GLONASS y GALILEO. Este receptor posee dos interfaces de conexión UART enumeradas como A y B, donde la interfaz A arroja datos utilizando el protocolo NMEA 0183 [24][30], la interfaz B arroja datos a través protocolo BINR[31] que es propietario del fabricante del receptor. Este último es relevante para el presente proyecto puesto que los datos arrojados utilizando este protocolo son crudos "RAW" es decir, que son datos generados sin ningún preprocesado y que pueden ser convertidos a otros formatos estándar como el formato RINEX que conservan la información de la señal recibida desde los satélites.

Figura 15. Receptor NV08C-CSM v4.1 conectado a través de interfaz UART A (NMEA).

[14] (2015). General Guide to SparkFun Blocks for Intel® Edison - learn ... Retrieved October 14, 2015, from https://learn.sparkfun.com/tutorials/general-guide-to-sparkfun-blocks-for-intel-edison.

Fig 16. Receptor NV08C-CSM v4.1, interfaz UART B (BINR).

6.2. CONFIGURACIÓN INICIAL DE USO PARA SE Y EL RECEPTOR GNSS

Como se mencionó anteriormente, al inicio del proyecto no se tenía la Intel Edison, así que el trabajo se empezó a elaborar utilizando la placa Galileo. Teniendo en cuenta la metodología elegida para el desarrollo de este proyecto, para este segundo sprint fue necesario realizar un breve análisis de requisitos para poder identificar los puntos claves necesarios para establecer las configuraciones iniciales para el uso del SE, como resultado de este análisis los siguientes puntos fueron identificados:

- Preparar el Sistema Embebido para su uso, en este caso la Intel Galileo.
- Configurar el SE para ser accedido remotamente.
- Conectar el Receptor GNSS.
- Verificar el funcionamiento del Receptor.

Una vez planteado lo anterior, se procedió a diseñar un plan de trabajo para este sprint. A medida que se identificaban tareas más específicas se continuó revisando toda la documentación disponible, en donde las fuentes que fueron más consultadas corresponden a la documentación ofrecida por Intel[32]. (Para más detalle siga el link en la referencia.)

A continuación se describe el procedimiento de desarrollo de este segundo sprint:

- Desde la página de Intel se descargó una imagen del sistema operativo Yocto Linux para Galileo.[15]
- La imagen descargada fue instalada sobre una micro SD que luego fue insertada en el módulo SD de la galileo.
- Se conectó la tarjeta a la fuente alimentación y a un equipo host a través el puerto micro USB cliente de la Galileo.
- Se estableció una conexión serial a través de USB entre la galileo y el equipo host utilizando la aplicación "screen".
- Se actualizó el firmware de la Galileo siguiendo el manual en la referencia. [33]
- Para validar el correcto funcionamiento de la Galileo y descartar posibles fallas a nivel de hardware, se ejecutaron varios de los sketches de arduino utilizando el IDE de arduino.[16]
- Para establecer una comunicación más robusta se conectó la Galileo con el equipo host a través de cable ethernet. Para este propósito, se instaló sobre el host un servidor DHCP para asignar dirección IP a la Galileo.
- En el equipo host se escaneó la tabla de direcciones ARP y se identificó la dirección IP que correspondía a la dirección MAC impresa sobre la galileo.
- Se accedió de manera remota a la Galileo a través de una conexión SSH.

Luego de esto se continuó de una manera más detenida el estudio de la documentación relacionada con el Receptor GNSS NV08C-CSM, como resultado se identificó que el receptor una vez es conectado sobre la interfaz UART A, comienza el envío mensajes en formato NMEA a través de una conexión serial simple. Pero si el receptor es conectado sobre la interfaz UART B, es necesario usar alguna aplicación que controle la recepción y almacenamiento de los datos enviados desde el receptor

[15] SD-Card Linux Image for Intel® Galileo. Retrieved October 13, 2015, from https://downloadcenter.intel.com/downloads/eula/24272/Intel-Galileo-Software-Package-1-0-3?httpDown=https%3A%2F%2Fdownloadmirror.intel.com%2F24272%2Feng%2FSDCard.1.0.3.zip.

[16] Intel® Arduino IDE 1.5.3 for Intel® Galileo Board ... Retrieved October 13, 2015, from https://downloadcenter.intel.com/downloads/eula/24782/Intel-Galileo-software-package-1-0-4?httpDown=https%3A%2F%2Fdownloadmirror.intel.com%2F24782%2Feng%2FIntelArduino-linux64-1.5.3.tgz

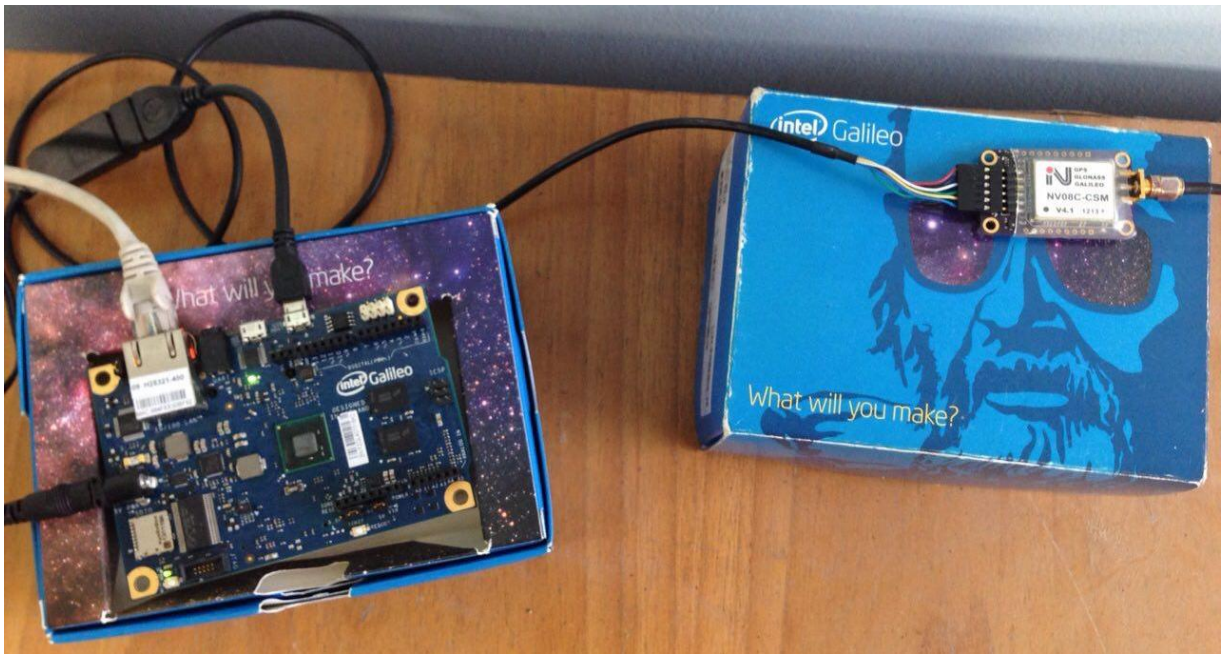
puesto que estos son datos binarios, luego de su captura estos datos pueden ser procesados, para estas labores se utilizó una aplicación de comunicación llamada str2str del paquete open source RTKLIB que usualmente es usado en adquisición, comunicación y postprocesado de los datos capturados con un Receptor GNSS.

En vista de lo anterior, se procedió inicialmente a probar el receptor conectándolo sobre la interfaz UART A a la Galileo para verificar el funcionamiento de este:

- Una vez conectado el Receptor con la Galileo utilizando una interfaz FTDI FT232R entre el puerto UART A del Receptor y el puerto microUSB de la Galileo.
- Para validar la conexión entre el SE y el Receptor, se revisaron los dispositivos conectados al SE dentro del directorio /dev/
- Para identificar el nuevo dispositivo conectado se redirigió a un fichero la salida estándar del listado de dispositivos antes de conectar el receptor al SE y a otro fichero la salida correspondiente después de conectar el receptor, luego se compararon los ficheros a través de un diff.
- Se revisaron los mensajes de entrada a través de la conexión identificada en /dev/, para esto se estableció una comunicación serial a través de screen, se tuvo en cuenta las indicaciones de la conexión descritas en la documentación, es decir, para establecer la conexión serial se establecieron los siguientes parámetros:

    - Velocidad de transmisión: 115200 bps
    - Bit de arranque: 1
    - Número de bits: 8
    - Verificación de paridad: ninguna
    - Control de flujo: ninguno
    - Bit de parada: 1
- Una vez establecida la conexión, se verificó la recepción de mensajes NMEA a través de screen.

Figura 17. Intel Galileo y Receptor conectados.



6.3. CONFIGURACIÓN Y DESPLIEGUE DEL SOFTWARE SOBRE EL SE.

Como es descrito en el proyecto RTKLIB, str2str es una a la versión CUI (Console User Interface) de la aplicación de RTKLIB encargada de la comunicación de un receptor con un computador, esto es importante y hay que tenerlo en cuenta puesto que tanto la Intel Galileo como la Edison son sistemas embebidos destinados a control y a IOT más que procesamiento gráfico, por lo tanto no cuentan con soporte a ningún entorno o aplicaciones gráfica. Las aplicaciones CUI de RTKLIB puede ser compiladas a partir de sus códigos fuentes escritos en C utilizando GNU-Make. Para esto, sobre el equipo host fue necesario desplegar el entorno de construcción de la imagen personalizada de Yocto linux y utilizando este entorno se pudo construir y desplegar el entorno de compilación cruzada para la Galileo.[35]

Para este Tercer Sprint se identificaron los siguientes puntos claves:

- Instalar y desplegar el entorno para la creación y personalización de la imagen de Yocto Linux sobre el equipo host.
- Instalar la imagen personalizada sobre la Intel Galileo.
- Desplegar las herramientas de compilación cruzada sobre el equipo host.
- Instalar y ejecutar la aplicación str2str sobre el SE.
- Establecer la comunicación entre el Receptor y el SE.

Una vez identificadas estas necesidades se procedió a leer con detenimiento los manuales proporcionados por intel para la creación de las imágenes personalizadas a través del proyecto yocto linux, también se revisó documentación relacionada con el proyecto yocto, openembedded y Bitbake. También se estableció el siguiente procedimiento que está descrito a mayor detalle en el Anexo #1:

- Se preparó el equipo host, para esto se instaló una serie de dependencias, tanto aplicaciones como librerías.
- Se instaló bitbake, las capas de software "poky" correspondientes al proyecto yocto linux y las capas de software (metadatos) para Galileo.
- Se ejecutó Bitbake llamando a la receta de construcción de la imagen yocto linux.
- La imagen generada se instaló sobre una memoria microSD y se colocó sobre la Galileo.
- Se verificó el funcionamiento de la Galileo con la nueva imagen y se configuró la conexión remota.
- Sobre el equipo host se construyó y desplegó el entorno de compilación cruzada.
- Las aplicaciones CUI de RTKLIB fueron recompiladas sobre el equipo host colocando como arquitectura target la Galileo.
- Se copiaron las aplicaciones CUI de RTKLIB sobre la Galileo
- Se conectó el sensor a través de la interfaz UART B que arroja los datos RAW en el formato propietario BINR.
- Se configuró el str2str para recoger los datos y almacenarlos en un fichero, teniendo en cuenta las indicaciones de la conexión descritas en la documentación, es decir, para establecer la conexión serial se establecieron los siguientes parámetros:

- Velocidad de transmisión: 115200 bps.
- Bit de arranque: 1
- Número de bits: 8
- Verificación de paridad: Impar
- Control de flujo: ninguno
- Bit de parada: 1

- Se capturaron datos y monitoreo el tamaño del fichero de salida.

Salida estándar de la aplicación str2str:

```
stream server start
2015/05/20 21:26:31 [CC---]            0 B      0 bps
2015/05/20 21:26:36 [CC---]        29271 B   38519 bps
2015/05/20 21:26:41 [CC---]        53616 B   38782 bps
2015/05/20 21:26:46 [CC---]        77898 B   39208 bps
2015/05/20 21:26:51 [CC---]       102241 B   39597 bps
2015/05/20 21:26:56 [CC---]       126549 B   38772 bps
2015/05/20 21:27:01 [CC---]       150906 B   39005 bps
2015/05/20 21:27:06 [CC---]       175054 B   38585 bps
2015/05/20 21:27:11 [CC---]       199377 B   39002 bps
```

6.4. VALIDACIÓN DE LA ADQUISICIÓN DE DATOS GNSS SOBRE EL SE.

Una vez fueron compiladas e instaladas las aplicaciones CUI del RTKLIB en la Galileo se procedió a validar la adquisición de los datos a través del Receptor. A su vez, este sprint coincidió con la llegada de la tarjeta Intel Edison, así se planteó la necesidad de replicar todo el proceso llevado a cabo con la Galileo. Para todo lo mencionado anteriormente en este sprint se identificaron los siguientes puntos claves:

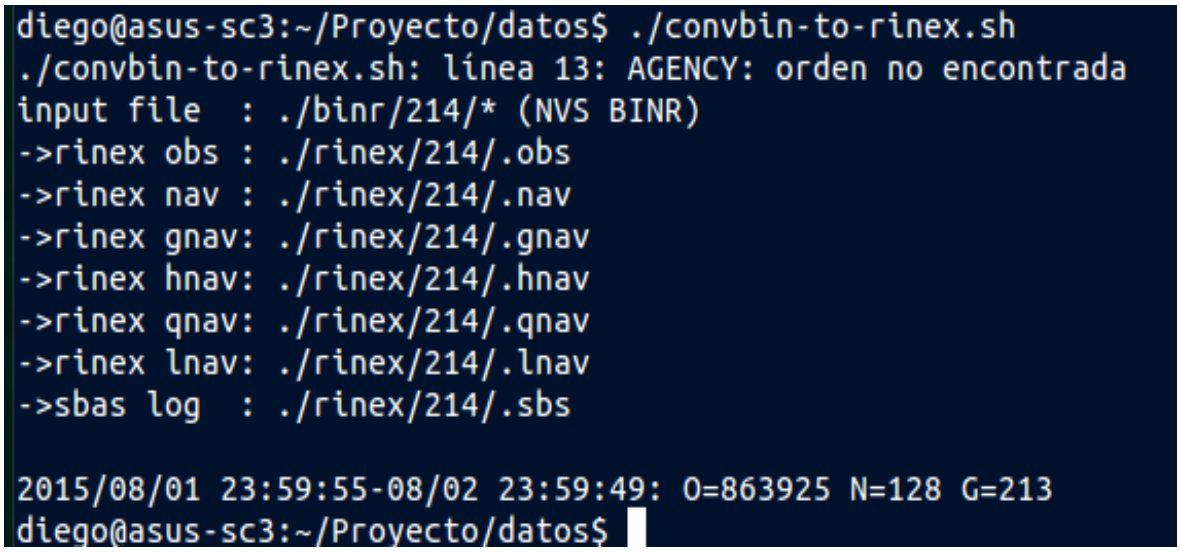
- Replicar todo el trabajo realizado hasta el momento sobre la Galileo en la Edison.
- Realizar campañas de medición.
- Elaboración de un script para automatizar y parametrizar las campañas de medición.
- Convertir los datos RAW BINR a un formato estándar como el RINEX.
- Analizar estos datos.
- Validar los datos teniendo en cuenta la posición calculada, número de satélites observados, etc.

Una vez fueron identificadas estas necesidades se elaboró y llevó a cabo el siguiente procedimiento:

- Explorar la usabilidad de la Intel Edison.
- Sobre el equipo host, construir la imagen personalizada de yocto linux de manera análoga como se hizo para la Galileo.
- Instalar la imagen personalizada sobre la Edison.
- Construir las herramientas de compilación cruzada.
- Compilar para la Edison las aplicaciones CUI de RTKLIB.
- Instalar estas aplicaciones sobre la Edison.
- Establecer unas campañas de medición de mínimo una hora.
- Llevar a cabo las campañas de medición.
- Luego se enviaron los datos capturados durante las campañas de medición al equipo host.
- En el equipo host se convirtieron los datos a formato RINEX utilizando la aplicación CUI de RTKLIB llamada convbin.
- Se tomaron estos datos y se llevaron unos análisis básicos de estos utilizando otra aplicación llamada GPSTK.
- Se validaron los datos recibidos desde ambos SE.

A continuación se colocan unas capturas de pantalla utilizando las librerías de gpstk para el procesado de datos en formato RINEX.

Figura 18. Conversión de datos BINR a RINEX usando un script que llama a convbin

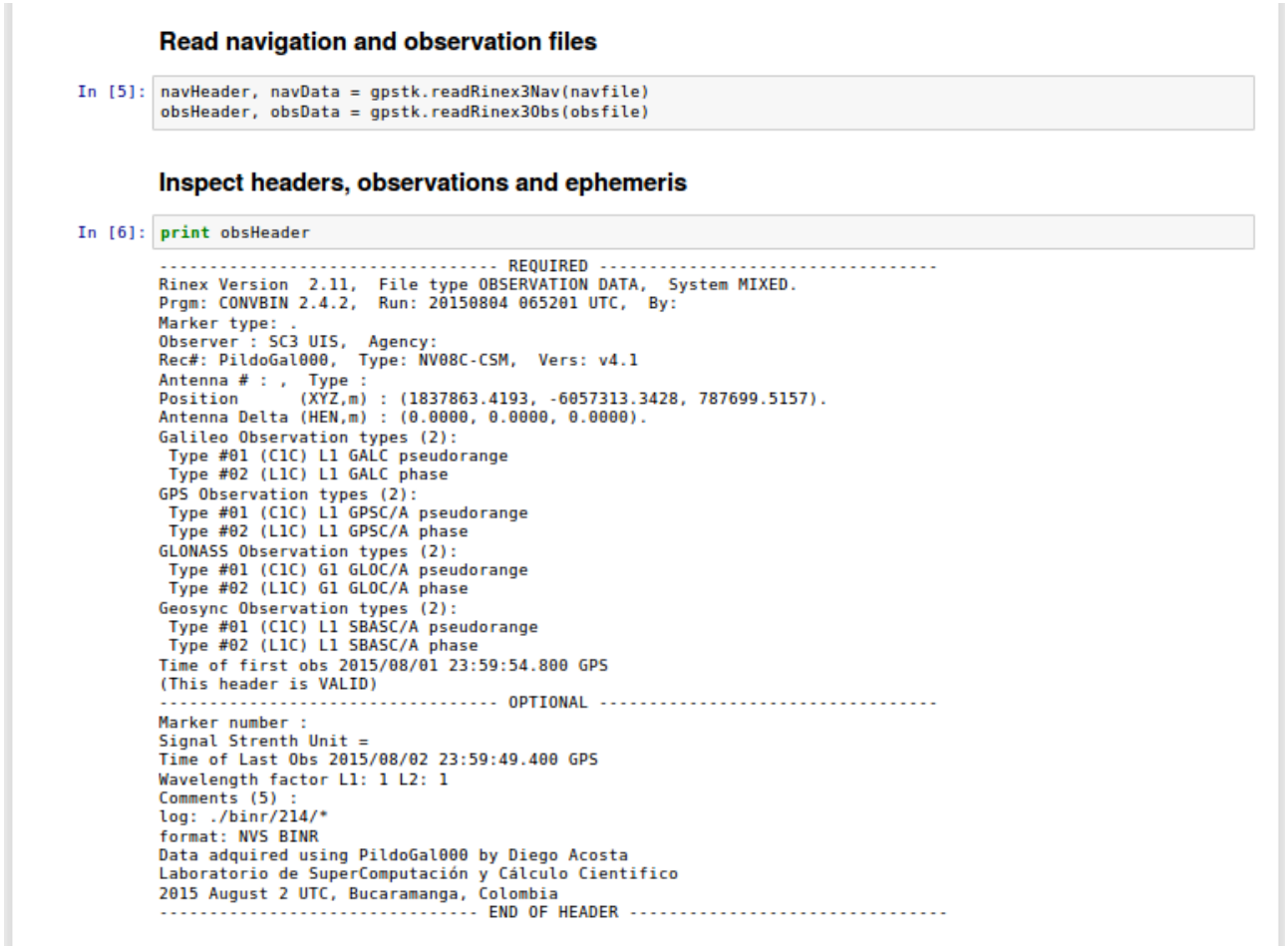

diego@asus-sc3:~/Proyecto/datos$ ./convbin-to-rinex.sh
./convbin-to-rinex.sh: línea 13: AGENCY: orden no encontrada
input file : ./binr/214/* (NVS BINR)
->rinex obs : ./rinex/214/.obs
->rinex nav : ./rinex/214/.nav
->rinex gnav: ./rinex/214/.gnav
->rinex hnav: ./rinex/214/.hnav
->rinex qnav: ./rinex/214/.qnav
->rinex lnav: ./rinex/214/.lnav
->sbas log : ./rinex/214/.sbs
2015/08/01 23:59:55-08/02 23:59:49: O=863925 N=128 G=213
diego@asus-sc3:~/Proyecto/datos$

Encabezado de los datos capturados y ya convertidos a RINEX.

```
       2.11           OBSERVATION DATA      M (MIXED)             RINEX VERSION / TYPE
CONVBIN 2.4.2                        20150804 065201 UTC PGM / RUN BY / DATE
log: ./binr/214/*                                                  COMMENT
format: NVS BINR                                                   COMMENT
Data adquired using PildoGal000 by Diego Acosta                    COMMENT
Laboratorio de SuperComputación y Cálculo Científico              COMMENT
2015 August 2 UTC, Bucaramanga, Colombia                           COMMENT
                                                                   MARKER NAME
                                                                   MARKER NUMBER
SC3 UIS                                                            OBSERVER / AGENCY
PildoGal000           NV08C-CSM             v4.1                   REC # / TYPE / VERS
                                                                   ANT # / TYPE
  1837863.4193 -6057313.3428   787699.5157                         APPROX POSITION XYZ
       0.0000         0.0000         0.0000                        ANTENNA: DELTA H/E/N
       1      1                                                    WAVELENGTH FACT L1/2
       2      C1     L1                                            # / TYPES OF OBSERV
  2015  8      1      23     59   54.8000000       GPS    TIME OF FIRST OBS
  2015  8      2      23     59   49.4000000       GPS    TIME OF LAST OBS
                                                                   END OF HEADER
 15  8  1 23 59 54.8000000  0 13R 1R20G17R 7R 8R22R21G20G13G15G 6G 2
                             G 5
  22622360.682 -7274366.394
  24024167.301         3190207.684
  23882758.099         1031454.852
  21679270.550
  20085614.199
  20670103.978
  20625595.618
  22545619.825
  21023158.983
  22085618.067   -14234681.722
  23712860.572
  22480439.347 -8543465.709
  21944310.477   -15800644.973
 15  8  1 23 59 54.9000000  0 13R 1R20G17R 7R 8R22R21G20G13G15G 6G 2
                             G 5
  22622320.798 -7274578.851
  24024229.342         3190539.400
  23882824.750         1031805.113
  21679341.240
  20085633.140
```

```
20670075.996
20625615.167
22545617.037
21023139.649
22085569.177   -14234938.623
23712872.437
22480427.625 -8543528.665
21944350.968   -15800432.132
```

Figura 19. Inspección de los encabezados, las observaciones y efemérides.



Read navigation and observation files

In [5]:
```
navHeader, navData = gpstk.readRinex3Nav(navfile)
obsHeader, obsData = gpstk.readRinex3Obs(obsfile)
```

Inspect headers, observations and ephemeris

In [6]:
```
print obsHeader
```

```
-------------------------------- REQUIRED --------------------------------
Rinex Version  2.11,  File type OBSERVATION DATA,  System MIXED.
Prgm: CONVBIN 2.4.2,  Run: 20150804 065201 UTC,  By:
Marker type: .
Observer : SC3 UIS,  Agency:
Rec#: PildoGal000,  Type: NV08C-CSM,  Vers: v4.1
Antenna # : ,  Type :
Position      (XYZ,m) : (1837863.4193, -6057313.3428, 787699.5157).
Antenna Delta (HEN,m) : (0.0000, 0.0000, 0.0000).
Galileo Observation types (2):
 Type #01 (C1C) L1 GALC pseudorange
 Type #02 (L1C) L1 GALC phase
GPS Observation types (2):
 Type #01 (C1C) L1 GPSC/A pseudorange
 Type #02 (L1C) L1 GPSC/A phase
GLONASS Observation types (2):
 Type #01 (C1C) G1 GLOC/A pseudorange
 Type #02 (L1C) G1 GLOC/A phase
Geosync Observation types (2):
 Type #01 (C1C) L1 SBASC/A pseudorange
 Type #02 (L1C) L1 SBASC/A phase
Time of first obs 2015/08/01 23:59:54.800 GPS
(This header is VALID)
------------------------------- OPTIONAL --------------------------------
Marker number :
Signal Strenth Unit =
Time of Last Obs 2015/08/02 23:59:49.400 GPS
Wavelength factor L1: 1 L2: 1
Comments (5) :
log: ./binr/214/*
format: NVS BINR
Data adquired using PildoGal000 by Diego Acosta
Laboratorio de SuperComputación y Cálculo Cientifico
2015 August 2 UTC, Bucaramanga, Colombia
------------------------------ END OF HEADER ------------------------------
```

Figura 20. Inspección de los datos de navegación.

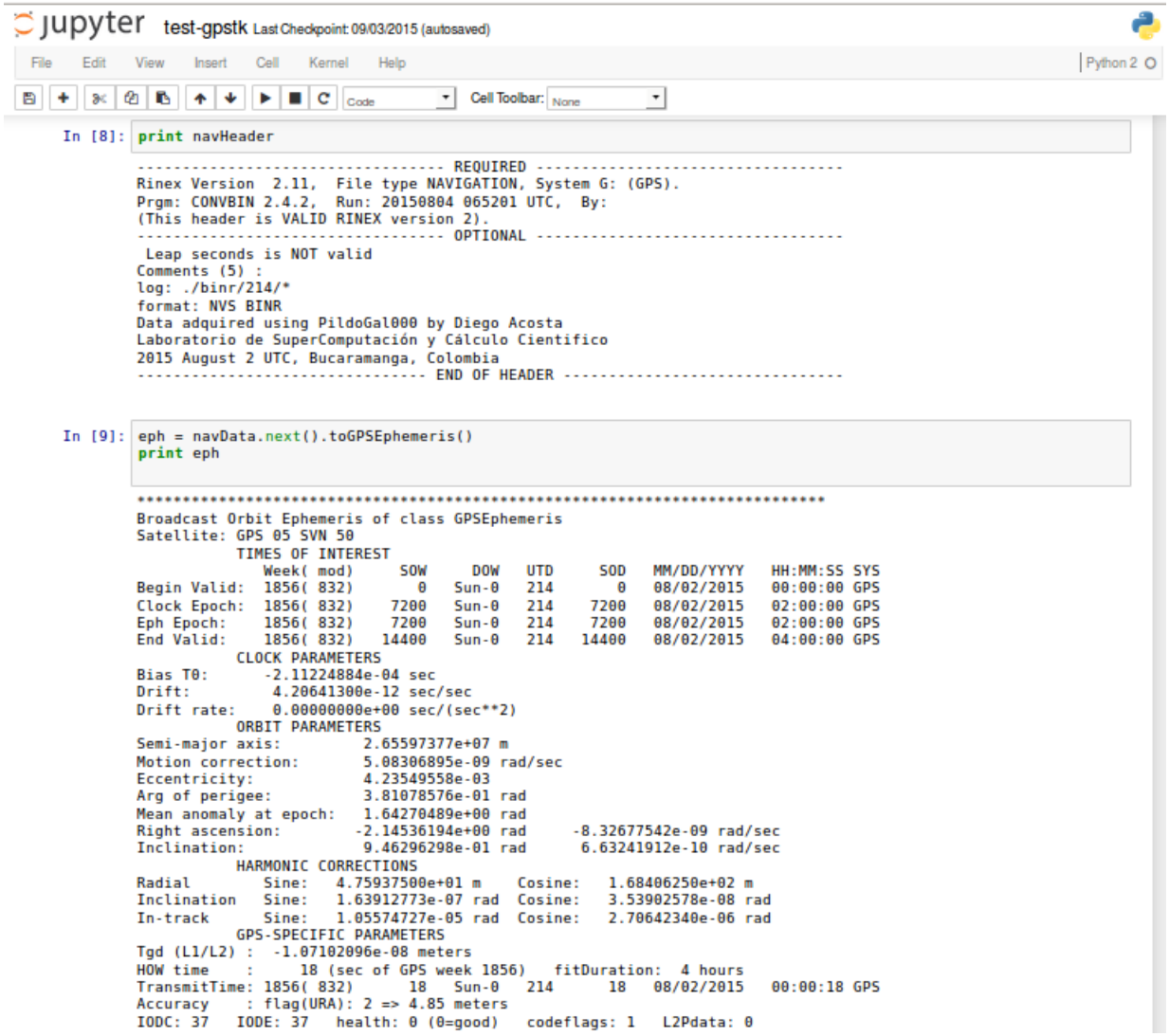
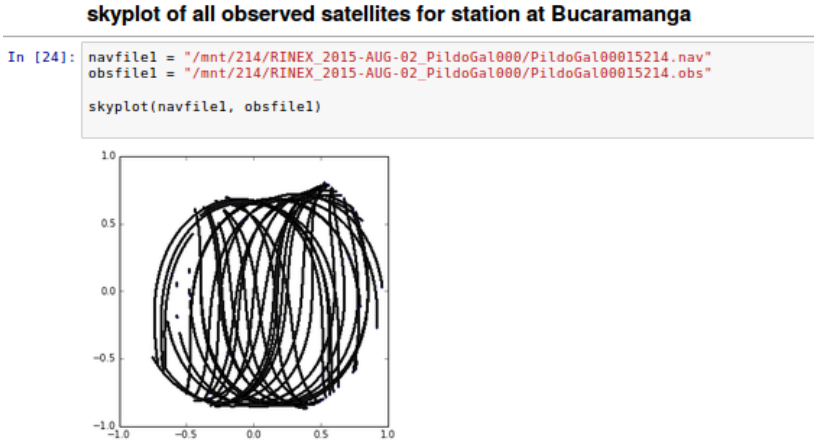


Figura 21. Gráfica de las trayectorias sobre la posición del receptor en el globo terráqueo.



6.5. VALIDACIÓN SINCRONIZACIÓN SE/HARDWARE CON SERVIDOR.

Para este sprint se planificó la comunicación, sincronización y envío de datos desde el SE al servidor destino, para llevar a cabo esto se utilizó una aplicación propietaria de Pildo llamada Data Collector, esta aplicación está desarrollada en java así que se identificó también la necesidad de instalar java sobre la Galileo y la Edison, también fue necesario implementar un servidor de pruebas para la recepción de los datos, Pildo tiene un web application llamado PAF que puede ser desplegado sobre un servidor web estándar tipo J2EE, por ejemplo Tomcat. En resumen se identificaron las siguientes necesidades:

- Instalar Java sobre la Galileo y la Edison.
- Instalar el Data Collector de Pildo sobre ambos SE.
- Montar un servidor de pruebas.
- Desplegar la web application de Pildo (PAF).
- Crear servicios (daemons) para Data Collector y str2str
- Establecer campañas de medición y pruebas.
- Validación de datos.

A partir de estos puntos claves se empezó a llevar a cabo el siguiente procedimiento:

- Instalación de java sobre la Galileo y la Edison.

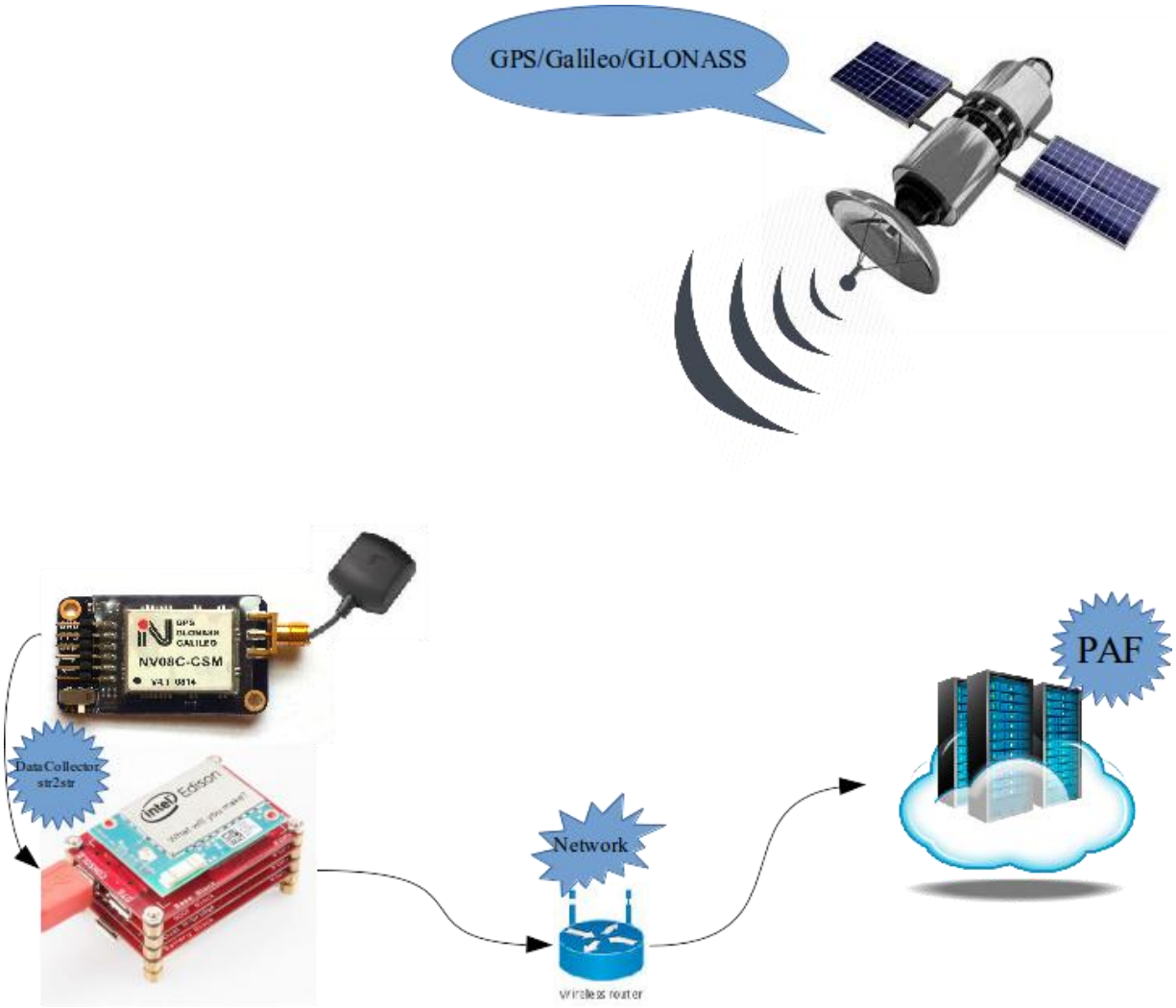
La instalación de Java se podía llevar a cabo de diferentes maneras, una de ellas era descargando los paquetes desde los repositorios de Intel e instalarlos sobre cada SE respectivamente, la otra forma es incluyendo la capa de metadatos de bitbake para que Java sea compilado, empaquetado e instalado sobre la imagen personalizada durante su proceso de construcción. Inicialmente se intentó instalar Java a través de los repositorios pero en el proceso se descubrió que la partición asignada al FileSystem de Yocto era muy pequeña por defecto, así que se optó por personalizar la imagen de yocto, a la imagen construida se le aumentó el tamaño de la partición sobre la cual se instalaría el FileSystem.

- Instalación del Data Collector de Pildo sobre ambos SE.
- Instalación y configuración de un servidor de pruebas J2EE (Tomcat).
- Despliegue del .war de la web application de Pildo (PAF) sobre el servidor de pruebas.
- Configuración del Data Collector sobre cada uno de los SE.
- Creación de scripts para la ejecución del Data Collector y el str2str.
- Creación de los scripts para implementar los servicios en el arranque del sistema que ejecutan la adquisición de datos con el str2str y el envío de estos con el Data Collector.
- Pruebas de envío de datos.
- Campañas de medición en ambos SE.
- Validación de los datos recibidos sobre el servidor de pruebas.

6.6. PRUEBAS DEL FLUJO DE TRABAJO DEL SISTEMA COMPLETO INTEGRADO

En la parte final del proyecto se vió la necesidad de integrar todo el flujo de trabajo para garantizar el correcto funcionamiento de la Pildobox, como se puede observar a continuación en el gráfico.

Figura 22. Flujo de trabajo.



Para el estudio final, se buscó comparar el rendimiento entre las distintas versiones de la Pildobox, más puntualmente, se busca comparar los diferentes Sistemas Embebidos usados en el proyecto. Se toma como marco de referencia a la Raspbery Pi B, ya que esta fue utilizada en la primera versión de la Pildobox.

Para este sprint se identificaron los siguientes puntos:

- Automatizar el proceso de configuración de la Pildobox.
- Llevar a cabo pruebas de consumo de recursos (Consumption resources).
- Analizar los resultados de las pruebas.
- Generar documentación final.

Se desea que la Pildobox sea integrada a un sistema de producción masiva, así que es necesario reducir al mínimo el trabajo y los tiempos de relacionados con el proceso de instalación y configuración del software. En el caso de la Raspberry Pi, Pildo preparó una imagen de Raspbian personalizada con todo el software instalado para ser copiado sobre una micro SD booteable, de manera análoga se puede realizar el proceso para la Galileo clonando una imagen personalizada de Yocto Linux con todo el software necesario cargado y configurado; pero este proceso no puede ser llevado a cabo de esta manera para la Edison dado que la fecha presente, la Edison no es booteable desde una tarjeta microSD por lo tanto el proceso se ha dividido en dos:

1. Construir la imagen personalizada usando las herramientas del proyecto yocto, e instalando esta imagen utilizando el procedimiento estándar de Intel que consiste en el flasheo de la memoria interna de la Edison.
2. Copiar el software que fue cross compilado previamente e instalando este último con un script.

Para llevar a cabo las pruebas de consumo de recursos se pensó en utilizar herramientas avanzadas como profilers e incluso sistemas de monitoreo de recursos como ganglia pero usualmente estas herramientas generan un mínimo consumo de recursos durante su ejecución, que en el caso de plataformas embebidas, no será despreciable. Entonces se pensó en lo básico y se utilizaron las herramientas estándar del paquete systat que usualmente está instalado en cualquier distribución de Linux. Dentro de las aplicaciones de systat, se identificó que la aplicación más conveniente para llevar

a cabo este estudio fue pidstat, esta aplicación permite visualizar el consumo de las aplicaciones en ejecución usando su identificador de proceso PID. Así que en el caso de la Pildobox, 2 aplicaciones deben ser monitoreadas: DataCollector y str2str.

Se establecieron 3 campañas de medición de 24, una sobre cada Sistema Embebido, en donde se monitoreo el porcentaje de consumo de CPU y el porcentaje de consumo de RAM para ambas aplicaciones. La frecuencia de monitoreo fue de una muestra por segundo, resultando así en 86400 muestras para 24 horas de ejecución.

En lo que respecta a la configuración, del receptor, este fue configurado para que capturara datos a 10Hz (máxima capacidad).

El DataCollector fue configurado para Galileo para que enviara datos cada 2 horas, para la Raspberry Pi se configuró para que enviara datos cada hora y para la Raspberry se configuró para que intentara enviar datos cada 10 minutos.

En las gráficas correspondientes al DataCollector se puede observar los picos en el consumo de la CPU a la hora de hacer el envío de los datos.

En las gráficas correspondientes al consumo de CPU para la aplicación str2str se puede apreciar que la Edison presenta un mejor desempeño en comparación con los otros dos sistemas embebidos.

También se puede observar que, sobre la Edison, ambas aplicaciones consumen menor porcentaje memoria de RAM tomando como referencia la memoria total disponible del sistema.

Figura 23. Intel Edison sobre board Spark Fun.

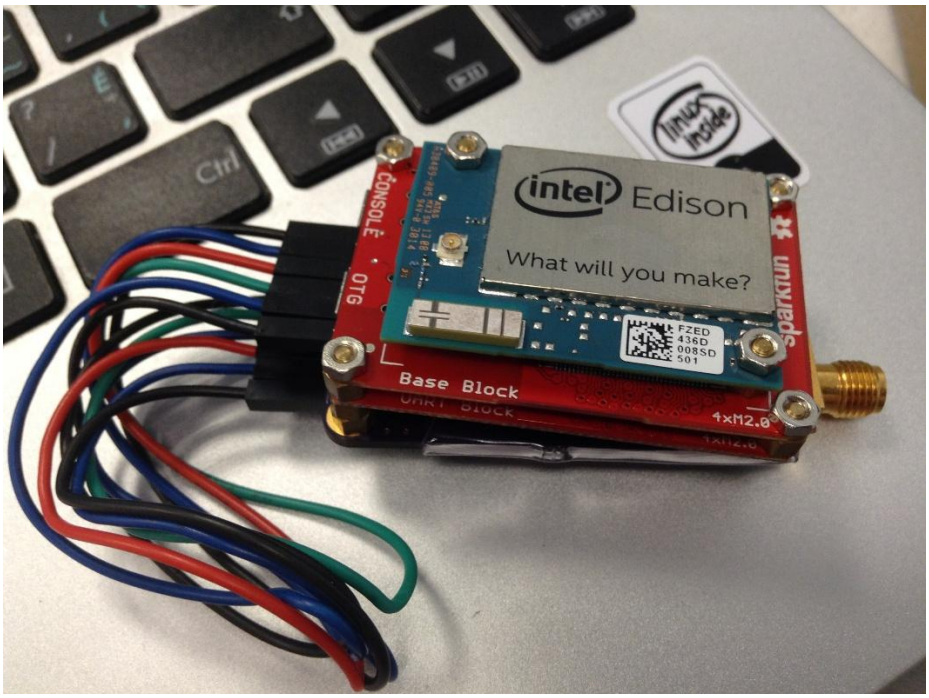
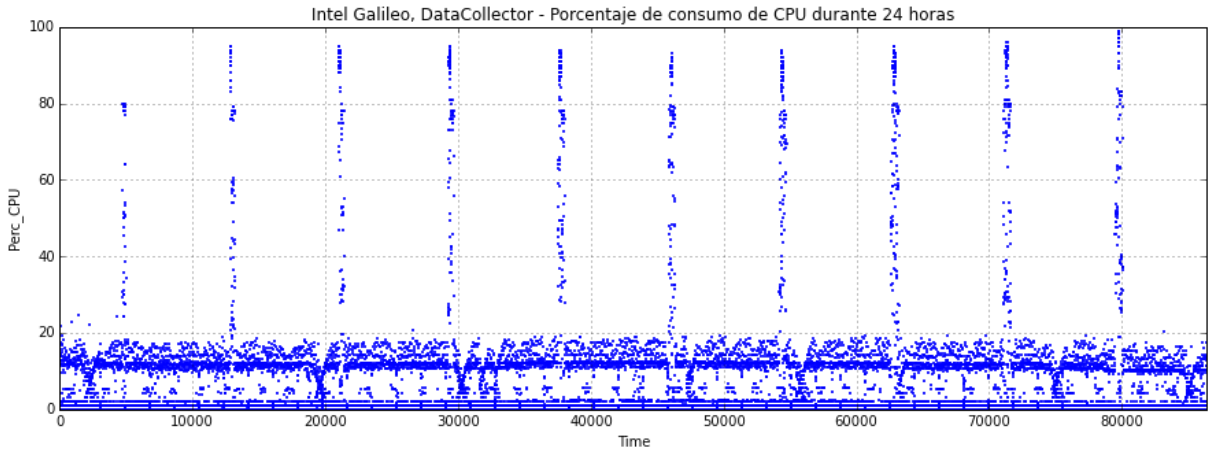
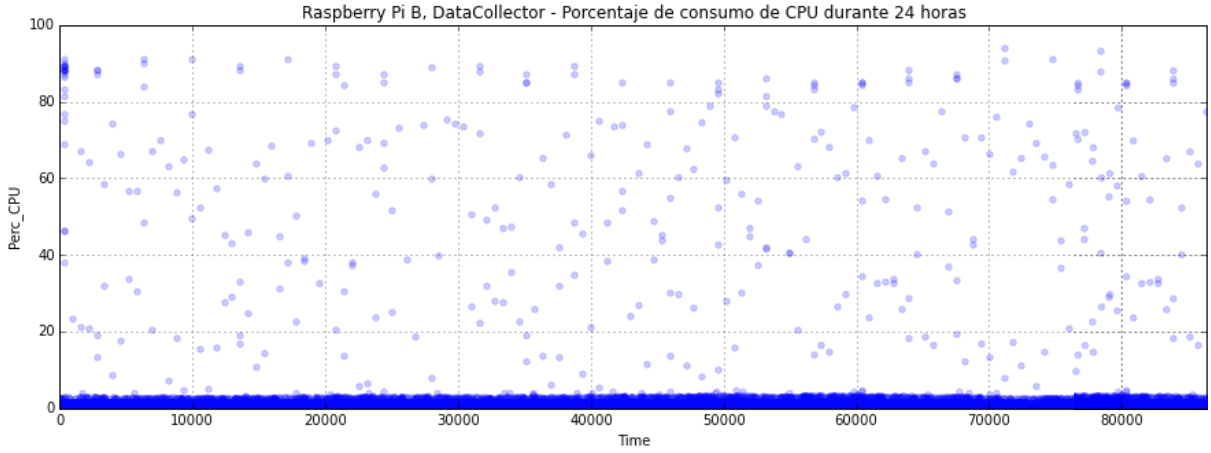


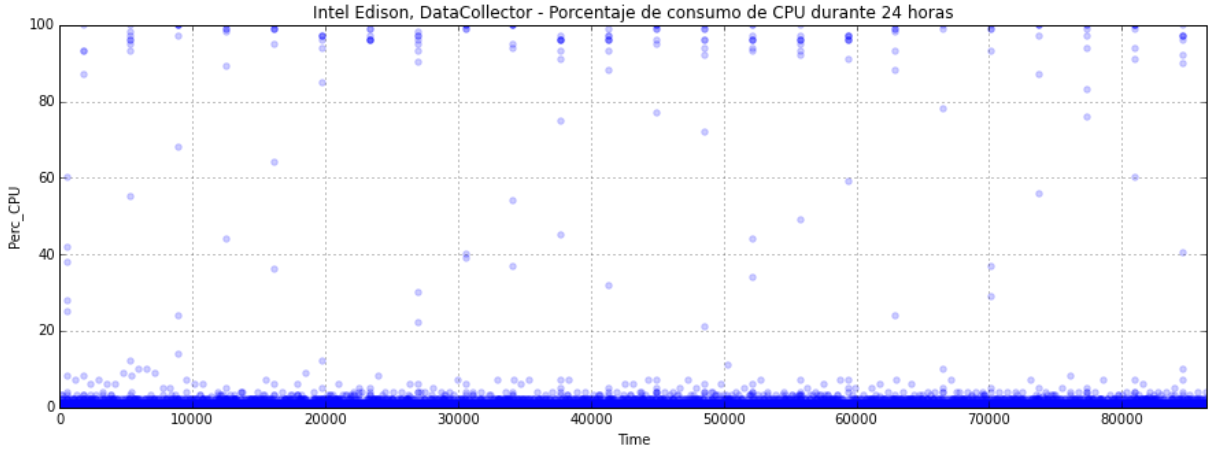
Figura 24. Receptor conectado a la Intel Edison a través de interfaz UART.

6.6.1. Comparación de gráficas

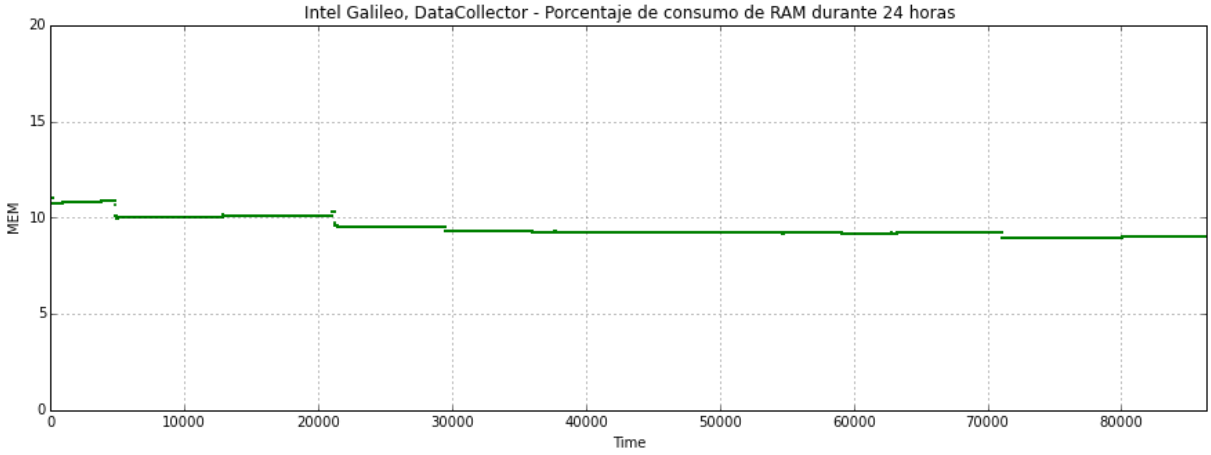
Porcentaje de consumo de CPU, DataCollector

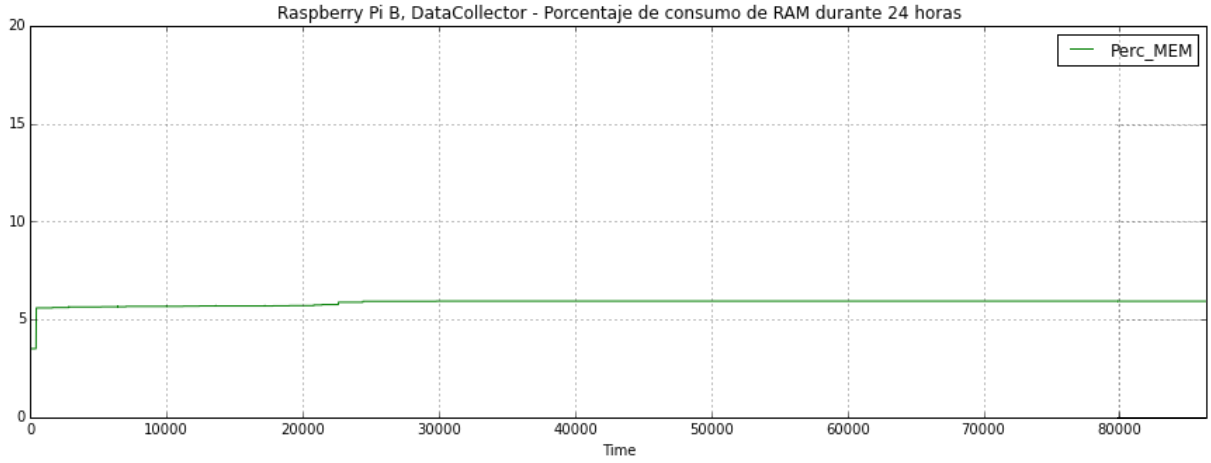


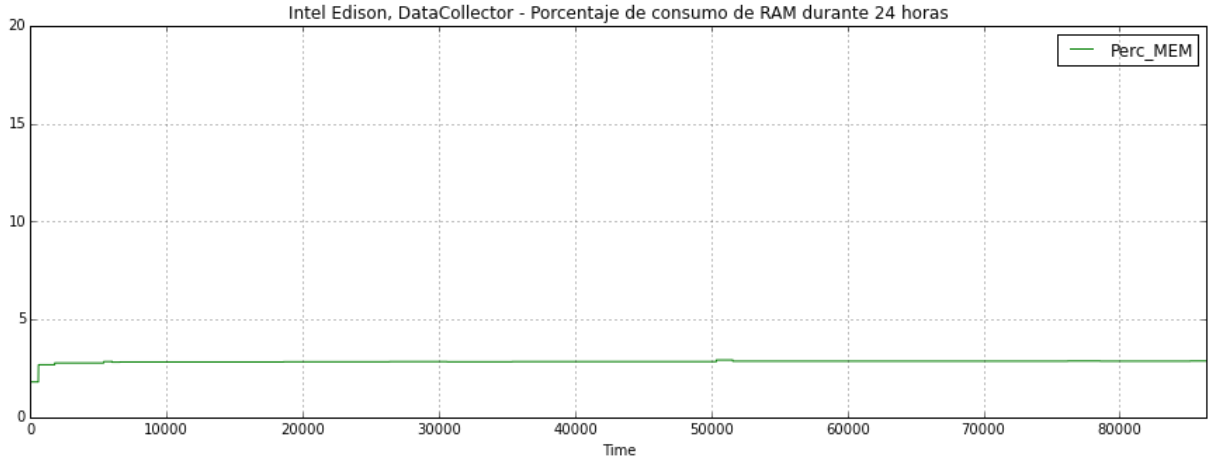




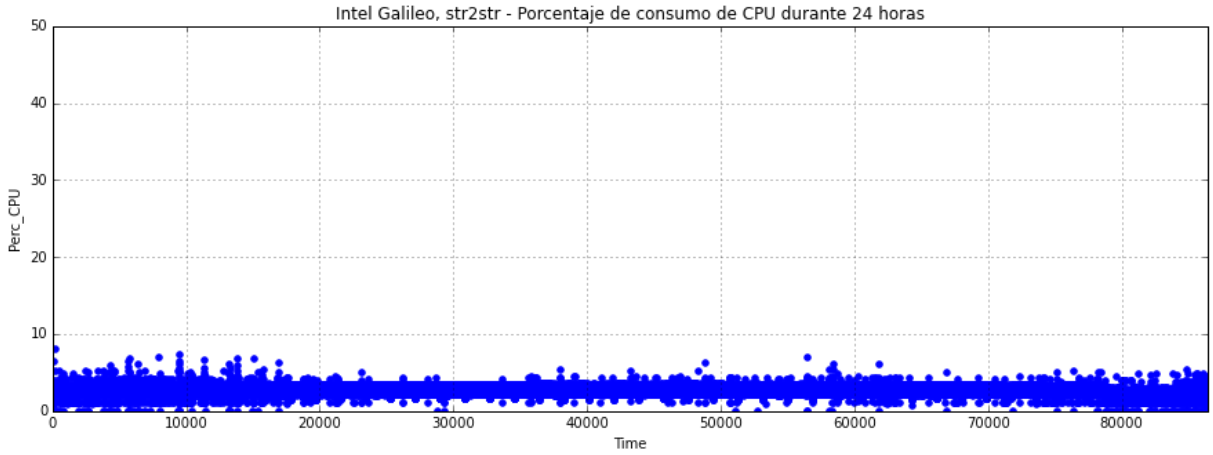
Porcentaje de consumo de RAM, DataCollector

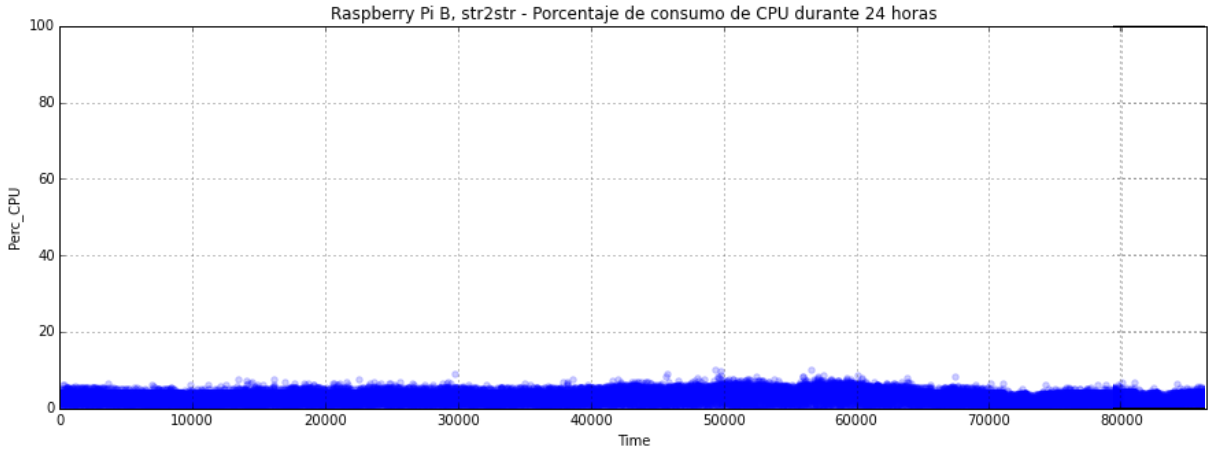


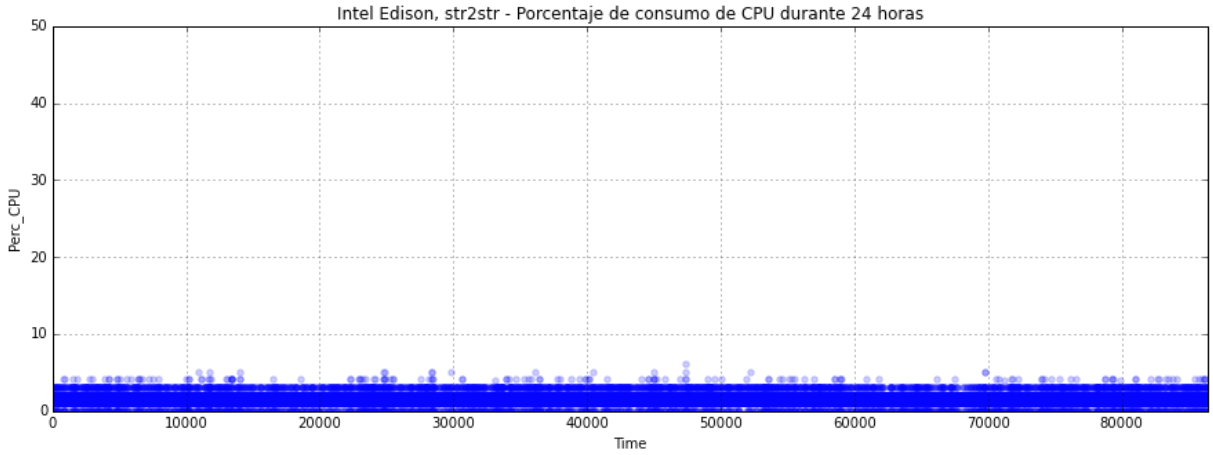




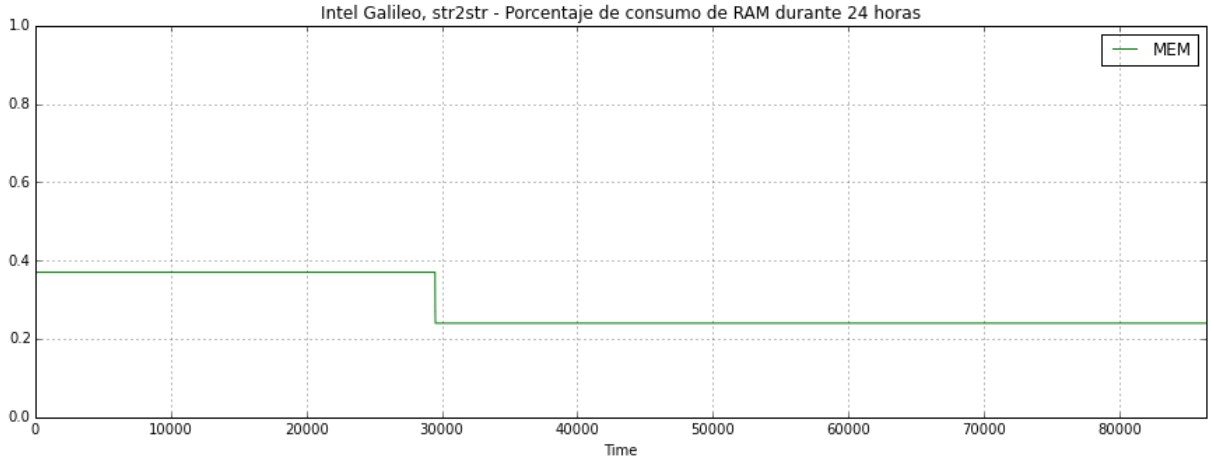
Porcentaje de consumo de CPU, str2str

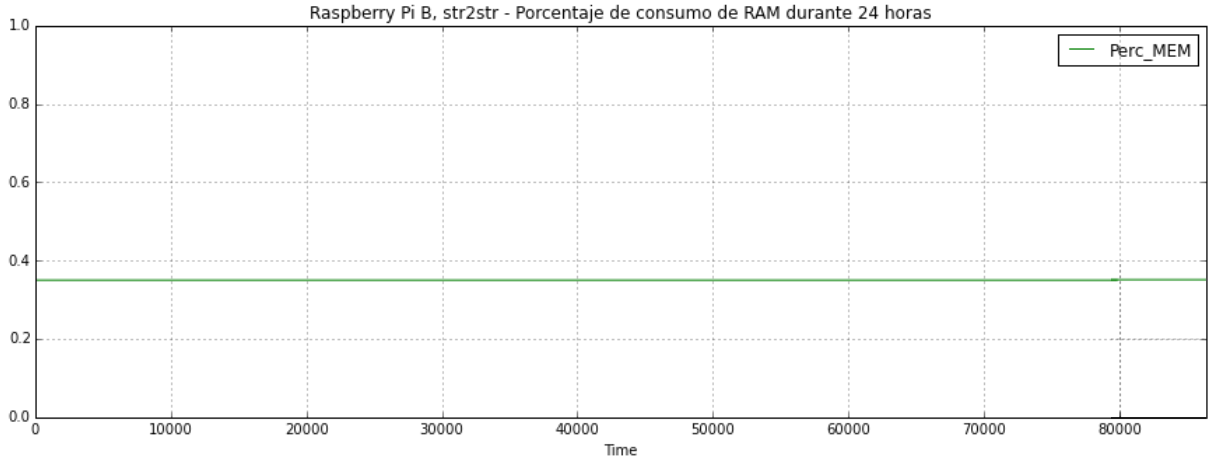


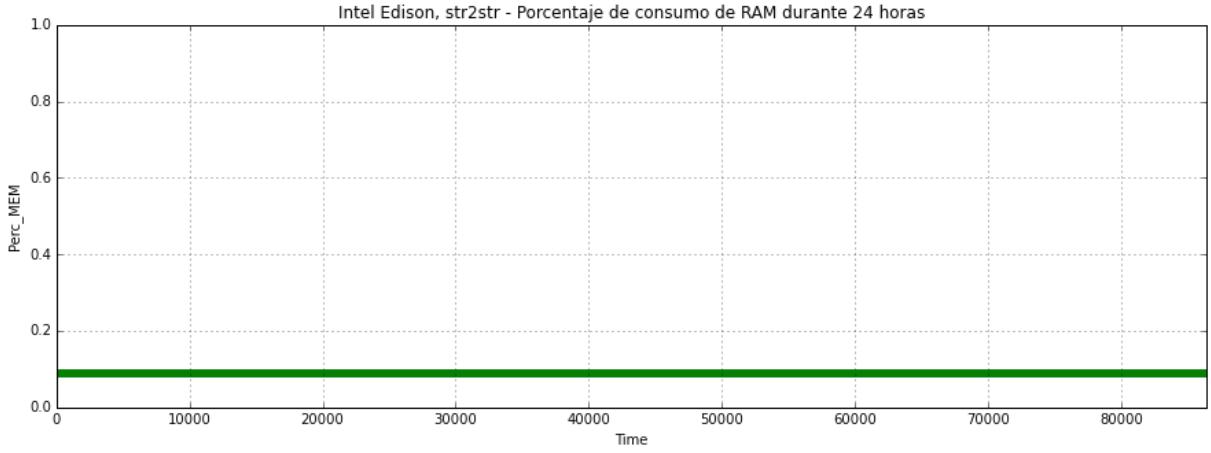




Porcentaje de consumo de memoria RAM, str2str







7. CONCLUSIONES

En función de los resultados obtenidos a partir de las pruebas de consumo de recursos, llevadas a cabo en la fase final del proyecto, se pudo evidenciar que el SE Intel Edison presenta un mejor desempeño respecto a la Raspberry Pi B y la Intel Galileo. Lo cual hace viable la utilización de la Intel Edison dentro de la PildoBox como el componente encargado del procesamiento. Además la Intel Edison ofrece la posibilidad de conectarse a redes inalámbricas a través de WiFi.

La metodología elegida para el proyecto brindó la flexibilidad necesaria para lidiar con los posibles imprevistos, facilitando así una planeación realista y ordenada para el desarrollo de la solución propuesta.

El proceso de configuración de un sistema embebido y la migración de software a estos sistemas no es un proceso trivial debido a la complejidad asociada al proceso de compilación cruzada y a todas las dependencias de librerías que puede tener un software.

El presente proyecto de grado ha sido útil para el aprendizaje y el desarrollo de habilidades para realizar trabajos en proyectos futuros en temas relacionados con GNSS, Telecomunicaciones, Sistemas Embebidos e Internet de las cosas IOT.

Se descubrió la necesidad de buscar una alternativa al DataCollector para el envío de los datos adquiridos dentro de la Pildobox hacia el servidor puesto que el DataCollector está implementado en Java, lo cual lo hace un poco pesado para ser ejecutado sobre Sistemas Embebidos que cuentan con limitaciones computacionales y restricciones energéticas.

8. RECOMENDACIONES

Integrar los bloques de expansión para la Intel Edison así como también integrar el receptor GNSS a uno de estos, ya que brindan una mayor integración física entre los componentes hardware y reduciendo el espacio ocupado.

Para futuros trabajos sobre la Pildobox se puede explorar la posibilidad de utilizar varias Pildobox repartidas en una amplia área geográfica para adquirir datos GNSS y que estos datos sean procesados en distribuido sobre las Pildobox aprovechando así el potencial de cálculo que está disponible en cada dispositivo.

Se recomienda que para trabajos futuros se creen nuevas capas de metadatos para la generación e instalación de paquetes para la adquisición y procesado de señales GNSS utilizando BitBake y el proyecto Yocto Linux.

CITAS

[1] (2011). GPS.gov: Space Segment. Retrieved January 12, 2015, from http://www.gps.gov/systems/gps/space/.

[2] (2011). Glonass. Retrieved January 12, 2015, from http://www.glonass-center.ru/en/GLONASS/.

[3] (2012). What is Galileo? / The future - Galileo / Navigation ... - ESA. Retrieved January 12, 2015, from http://www.esa.int/Our_Activities/Navigation/The_future_-_Galileo/What_is_Galileo.

[4] (2010). Galileo/EGNOS: range of potential applications - European ... Retrieved January 29, 2015, from http://ec.europa.eu/enterprise/policies/satnav/galileo/applications/index_en.htm.

[5] Hofmann-Wellenhof, B., Lichtenegger, H., & Wasle, E. (2007). *GNSS–global navigation satellite systems: GPS, GLONASS, Galileo, and more*. Springer.

[6] (2012). About satellite navigation / Navigation / Our Activities / ESA. Retrieved January 27, 2015, from http://www.esa.int/Our_Activities/Navigation/About_satellite_navigation2.

[7] Gleason, S., & Gebre-Egziabher, D. (Eds.). (2009). *GNSS applications and methods*. Artech House.

[8] Li, Q., & Yao, C. (2003, January 4). *Real-time concepts for embedded systems*.(pp. 5) CRC Press.

[9] Ralston, A., Reilly, E. D., & Hemmendinger, D. (1993, January 2). *Encyclopedia of computer science* (pp. 713-714). New York: Van Nostrand Reinhold. - ACM Digital Library. Retrieved January 28, 2015, from http://dl.acm.org/ralston.cfm

[10] Waite, W. M., & Goos, G. (1984). *Compiler construction* (pp. 1). New York: Springer-Verlag.

[11] Greene, B. (2004). Agile methods applied to embedded firmware development. *Agile Development Conference, 2004*. IEEE.

[12] (2012). NMEA Standards. Retrieved January 30, 2015, from http://www.nmea.org/content/nmea_standards/nmea_0183_v_410.asp.

[13] (2014). GNSS, Satellite Navigation and Data Format (NMEA). Retrieved January 30, 2015, from http://www.challenge.toradex.com/projects/10115-globo-daq/updates/10129-gnss-module-gnss-satellite-navigation-and-data-format-nmea-relevant-information.

[14] (2011). The Nog: GPS Error Budget. Retrieved January 30, 2015, from http://thenogspot.blogspot.com/p/gps-error-budget.html.

[15] (2012). Trimble - GPS Tutorial - Error Correction. Retrieved January 30, 2015, from http://www.trimble.com/gps_tutorial/howgps-error2.aspx.

[16] (2014). Scrum Theory - Scrum Guides. Retrieved January 30, 2015, from http://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-ES.pdf.

[17] J. Sanz Subirana, J. M. Juan Zornoza and M. Hernández-Pajares (2013). GNSS Data Processing, Vol. I: Fundamentals and Algorithms, ESA TM-23 / ESA Publications. http://www.esa.int/About_Us/ESA_Publications/ESA_TM-23_GNSS_DATA_PROCESSING.

[18] (2014). GPS: An Introduction to Satellite Navigation, with an interactive Worldwide Laboratory using Smartphones - Coursera, Stanford University. Stanford, California, USA. Retrieved October 8, 2015, from https://www.coursera.org/course/gpslab. https://d396qusza40orc.cloudfront.net/gpslab/lecture_slides/Stanford_GPS_MOOC_Module%202%20%28Part%201%29.pdf

[19] (2012). Klobuchar Ionospheric Model - Navipedia. Retrieved October 11, 2015, from http://www.navipedia.net/index.php/Klobuchar_Ionospheric_Model.

[20] LEE Jiyun, AMT Tetra Tech. (2010). Long Term Monitoring of Ionospheric Anomalies to Support the Local Area Augmentation System. Retrieved from http://waas.stanford.edu/papers/Lee_IONGNSS_2010_LongTermIonoMonitoring_C5_paper.pdf.

[21] Blewitt, G. (1997). Basics of the GPS technique: observation equations (pp 3, 7, 27). Geodetic applications of GPS, 10-54. Retrieved October 11, 2015, from http://www.nbmg.unr.edu/staff/pdfs/Blewitt%20Basics%20of%20GPS.pdf

[22] Gurtner, W., & Estey, L. (2007). RINEX-The Receiver Independent Exchange Format-Version 3.00. Astronomical Institute, University of Bern and UNAVCO, Bolulder, Colorado.

[23] (2015). RINEX: The Receiver Independent Exchange Format - IGS. Retrieved October 12, 2015, from https://igscb.jpl.nasa.gov/igscb/data/format/rinex211.txt.

[24] (2015). NV08C-RTK NMEA Protocol Specification (pp. 4-7) - NVS Technologies AG. Retrieved October 7, 2015, from http://nvs-gnss.com/support/documentation/item/download/79.html.

[25] (2011). automake: Cross-Compilation - Gnu. Retrieved October 8, 2015, from https://www.gnu.org/software/automake/manual/html_node/Cross_002dCompilation.html.

[26] (2007). Introduction to cross-compiling for Linux - Landley.net. Retrieved October 8, 2015, from http://landley.net/writing/docs/cross-compiling.html.

[27] (2014). Intel® Edison Product Brief - Intel. Retrieved October 7, 2015, from http://download.intel.com/support/edison/sb/edison_pb_331179001.pdf.

[28] (2011). About | Yocto Project. Retrieved October 7, 2015, from https://www.yoctoproject.org/about.

[29] (2015). NV08C-CSM. Datasheet - NVS Technologies AG. Retrieved October 7, 2015, from http://www.nvs-gnss.com/products/receivers/item/download/77.html.

[30] (2012). NMEA Standard 0183 - NMEA. Retrieved October 7, 2015, from https://www.nmea.org/content/nmea_standards/nmea_0183_v_410.asp.

[31] (2014). BINR Protocol Specification - NVS Technologies AG. Retrieved October 7, 2015, from http://www.nvs-gnss.com/support/documentation/item/download/39.html.

[32] (2015). IoT - Intel® Galileo Board Get Started Guide | Intel ... Retrieved October 7, 2015, from https://software.intel.com/en-us/iot/library/galileo-getting-started.

[33] (2015). Intel® Galileo Firmware Updater Tool User Guide. Retrieved October 7, 2015, from https://downloadmirror.intel.com/24748/eng/IntelGalileoFirmwareUpdaterUserGuide-1.0.4.pdf.

[34] (2015). IoT - Installing drivers and updating firmware for Arduino on ... Retrieved October 7, 2015, from https://software.intel.com/en-us/installing-drivers-and-updating-firmware-for-arduino-windows.

[35] (2015). Build and Software User Guide - Intel. Retrieved October 7, 2015, from https://downloadmirror.intel.com/23962/eng/Quark_BSP_BuildandSWUserGuide_329687_007.pdf.

BIBLIOGRAFÍA

Blewitt, G. (1997). Basics of the GPS Technique: Observation Equations. In B. Johnson, *Geodetic Applications of GPS* (pp. 10-54). Sweden: Nordic Geodetic Commission.

Gleason, S., & Gebre-Egziabher, D. (2009). *GNSS Aplications and Methods.* Artech House.

Greene, B. (2004). Agile methods applied to embedded firmware development. *Agile Development Conference, 2004* (pp. 71-77). Salt Lake: IEEE.

Hofmann Wellenhof, B., Lichtenegger, H., & Wasle, E. (2008). *GNSS – Global Navigation Satellite Systems: GPS, GLONASS, Galileo, and more.* (1 ed.). Vienna, Austria: Springer-Verlag Wien.

Sanz Subirana, J., Juan Zornoza, J. M., & Hernández-Pajares, M. (2013). *GNSS Data Processing Volume I: Fundamentals and Algorithms* (Vol. 1). Neetherlands: ESA Communications.

Schwaber, K., & Sutherland, J. (2013, July). *ScrumGuides.org.* Retrieved from http://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-US.pdf

ANEXOS

A - GUÍA DE DESPLIEGUE DE LAS HERRAMIENTAS PARA LA CREACIÓN DE IMAGEN YOCTO LINUX PARA INTEL® GALILEO Y INTEL® EDISON.

Antes de hablar sobre el proceso de construcción de la imagen en sí, es necesario hablar primero sobre el proyecto Yocto Linux para contextualizar varios de los conceptos utilizados en la descripción de las tareas de esta breve guía de usuario. La información escrita aquí en su mayoría ha sido extraída y resumida a partir de la documentación oficial del proyecto Yocto Linux.
http://www.yoctoproject.org/docs/1.8/ref-manual/ref-manual.html
http://www.yoctoproject.org/docs/1.8/dev-manual/dev-manual.html
http://www.yoctoproject.org/docs/1.8/bitbake-user-manual/bitbake-user-manual.html

YOCTO LINUX

El proyecto Yocto Linux es un proyecto open-source enfocado a desarrollo de una distribución GNU Linux para plataformas basadas en diferentes arquitecturas embebidas como ARM, MIPS, PowePC, x86 y x86-64. Yocto Linux cuenta con un sistema para la construcción del software conocido como OpenEmbedded build system, este a su vez usa a BitBake para la construcción de las imágenes en base a los metadatos que los desarrolladores y la comunidad han almacenado en los repositorios de openembedded.

BITBAKE

Bitbake es conocido como un motor de ejecución de tareas genéricas (generic task execution engine), este se deriva de Portage, que es el sistema de gestión de paquetes utilizado por la distribución Gentoo Linux. BitBake permite ejecutar tareas de python sobre el shell de una manera eficiente y paralelizada, al tiempo que lidia con tareas complejas y dependencias fuertemente interrelacionadas. Bitbake funciona de una manera similar a GNU Make y es usado más comúnmente para construir paquetes. Pero además, Bitbake posee una serie de características entre las que destacan:

- Ejecuta tareas de acuerdo a los metadatos proporcionados en la construcción de las tareas.

- Incluye librerías para hacer fetch (referenciar, localizar y descargar) al momento de la adquisición del código fuente desde diferentes lugares como archivos locales, sistemas de control de versiones o sitios web.
- Cada receta contiene toda la información acerca de la unidad (dependencias, localización de archivos fuente, checksums, descripciones, etc.)
- Incluye abstracción cliente/servidor que puede ser usada desde CUI o ser utilizada como un servicio sobre XML-RPC.

Conceptos relevantes relacionados con BitBake y el proyecto Yocto Linux:

Recetas

Las recetas contienen información descriptiva acerca de:

- Paquetes (autor, fuente, licencia, etc)
- Versión de la receta.
- Dependencias existentes.
- Ubicación del código fuente y como hacer "fetch".
- Buscar y aplicar parches si es necesario.
- Cómo compilar y configurar el código fuente.
- Ubicación sobre la máquina target para instalar los paquetes creados.

Clases

Las clases contienen información que es útil a la hora compartir de compartir información entre los archivos de metadatos. En el directorio classes del directorio raíz de Bitbake viene un archivo especial de metadatos llamado `base.bbclass`, este archivo es especial ya que desde él siempre son incluidas todas las recetas y las clases de manera automática. Una clase puede ofrecer definiciones por defecto de funciones como: fetching, unpacking, configuring, compiling, installing, packing, etc.

Layers (Capas)

Las capas permiten mantener las cosas de manera modular es decir, permite personalizar configuraciones que pueden ser objetivo para alguna máquina en específico pero sin intervenir con

configuraciones ajenas a esa máquina. Estos cambios sobre las recetas y las clases se elaboran a través de recetas tipo append (`.bbappend`).

Append Files

Permiten extender y sobreescribir información existente dentro de una receta de una manera similar a los archivos `.patch`.

Poky

El término *poky* puede significar varias cosas, pero usualmente hace referencia a la distribución por defecto de Yocto Linux.

A continuación se procede a dar paso a la descripción del proceso de creación y configuración de una imagen personalizada de Yocto Linux para la Galileo[17]:

1. Preparación del equipo host.

Es necesario contar con un equipo host con buena capacidad de cómputo puesto que la ejecución del constructor de la imagen BitBake consume la máxima capacidad de procesamiento del equipo para la ejecución de todas las tareas en paralelo entre las cuales se encuentran tareas demandantes de recursos como la compilación. También se debe tener en cuenta que el equipo host debe tener instalada alguna una distribución con amplia difusión como lo es Ubuntu, Fedora u OpenSuSe.[18]

2. Instalación de las dependencias.

El equipo host debe contar con una serie de dependencias necesarias en el proceso de construcción de la imagen de Yocto Linux.

- build-essential: Paquete encargado de la instalación de otros paquetes esenciales de C/C++.
- gcc-multilib: Compilador de C.
- vim-common: versión enriquecida del editor de textos vi.
- uuid-dev: universally unique id library, encabezados y librerías estáticas.
- wget: aplicación para la descarga de archivos desde la web utilizando protocolos http y ftp.

[17] (2015). Downloading - Intel. Retrieved October 14, 2015, from https://downloadmirror.intel.com/23197/eng/Quark_BSP_BuildGuide_329687_009.pdf

[18] (2014). Yocto Project Reference Manual 1.8. Retrieved October 14, 2015, from http://www.yoctoproject.org/docs/current/ref-manual/ref-manual.html#detailed-supported-distros.

- subversion: sistema de control y revisión versiones.
- git-core: sistema de control y revisión de versiones.
- autoconf: constructor de scripts automáticos de configuración.
- libtool: librería genérica de soporte a librerías dinámicas en ejecución (módulos).
- diffstat: produce gráficos de los cambios introducidos por un archivo diff.
- gawk: GNU awk, scanner de patrones y procesador de lenguaje.
- chrpath: herramienta para editar el rpath en binarios ELF.
- texinfo: sistema de documentación para información on-line e impresión de salidas.
- iasl: intel ASL compiler.
-

Ejemplo de instalación de paquetes en ubuntu:

```
$ sudo apt-get install build-essential gcc-multilib vim-common uuid-dev gawk wget
git-core subversion diffstat chrpath texinfo iasl autoconf libtool
```

3. Descarga de herramientas desde repositorio de Intel.

```
$ cd && wget
https://downloadmirror.intel.com/23197/eng/Board_Support_Package_Sources_for_Inte
l_Quark_v1.2.0.7z
```

Descomprimir el fichero:

```
$ 7z x Board_Support_Package_Sources_for_Intel_Quark_v1.2.0.7z
```

Después extraer el fichero llamado "meta-clanton_*" en el directorio home, por ejemplo:

```
$ tar -xvf meta-clanton_v1.2.0.tar.gz /home/user/galileo
```

A partir de ahora el directorio en donde fue extraído el fichero "meta-clanton_*" será conocido como el directorio raíz para Bitbake.

Entrar en $TOPDIR y ejecutar el script setup:

```
$ cd /home/user/galileo && ./setup.sh
```

Una vez ejecutado el setup, este descargara las capas de software necesarias, entre esas poky, para la construcción de la imagen de Yocto Linux.

4. A continuación se cargan las variables para la sesión shell sobre la cual se está trabajando, para esto se utiliza el comando source y el fichero con las variables; además se define un directorio destino para el despliegue de la construcción, por ejemplo "build":

```
$ source oe-init-build-env build
```

5. Ahora se procede a modificar algunos parámetros para personalizar la construcción de las imágenes, esta configuración se lleva a cabo dentro de la receta con la ruta "$TOPDIR/build/conf/bblayers.conf", dentro de este archivo se debe referenciar la ruta a las capas que se desean instalar

   ```
   $ vi bblayers.conf
   ```

```
# LAYER_CONF_VERSION is increased each time build/conf/bblayers.conf
# changes incompatibly
LCONF_VERSION = "6"

BBPATH = "${TOPDIR}"
BBFILES ?= ""

BBLAYERS ?= " \
  /home/user/galileo/meta \
  /home/user/galileo/meta-yocto \
  /home/user/galileo/meta-intel-iot-devkit \
  /home/user/galileo/meta-intel-iot-middleware \
  /home/user/galileo/meta-intel-quark \
  /home/user/galileo/meta-intel-galileo \
  /home/user/galileo/meta-netcontiki \
  /home/user/galileo/meta-openembedded/meta-networking \
  /home/user/galileo/meta-oracle-java \
  "
BBLAYERS_NON_REMOVABLE ?= " \
  /home/user/galileo/meta \
  /home/user/galileo/meta-yocto \
  "
```

Por ejemplo en esta versión del archivo bblayers.conf se incluyó la ruta al directorio correspondiente la capa de metadatos que permite instalar Java sobre la Galileo. Si se desean incluir nuevas capas de software en la construcción de la imagen de Yocto Linux, se deben descargar y referenciar dentro de este archivo.

6. Bitbake permite construir recetas aisladas para por ejemplo compilar y crear paquetes de algún software en específico o también construir toda una imagen del sistema operativo que será booteable para el dispositivo target. Siendo este último el caso, se puede personalizar la receta encargada de la construcción de la imagen, por ejemplo para la imagen de Galileo se debe modificar la receta correspondiente a la ruta:
   "$TOPDIR/meta-iot-devkit/recipes-core/images/image-full.bb"

Dentro de este archivo se pueden modificar varios parámetros, como por ejemplo el tamaño del file system:

IMAGE_ROOTFS_SIZE = "3500000"

Las unidades de este parámetro están en kiloBytes.

También en este archivo se pueden configurar la instalación automática de paquetes construidos a partir de capas de metadatos, por ejemplo Java:

IMAGE_INSTALL += "oracle-jse-jdk-i586"

En el caso de Java cabe mencionar que es necesario agregar el paquete a la "lista blanca" ya que Java es un software privado de terceros. Para esto se debe modificar el archivo que está en la ruta de bblayers.conf llamado local.conf, ahí al final se debe incluir la siguiente línea:

LICENSE_FLAGS_WHITELIST += "oracle_java"

7. Una vez llevadas a cabo todas las configuraciones, ahora se procede a ejecutar bitbake para construir la imagen de Yocto Linux.

```
$ bitbake image-full
```

En este momento el proceso puede tardar varias horas, cuando la construcción termine, los siguientes archivos de salida pueden ser encontrados en .../build/tmp/deploy/images/quark/. Copia estos archivos a una microSD. Renombra los archivos de acuerdo a las siguientes instrucciones:

- `image-full-quark-YYYYMMDDhhmmss.rootfs.ext3`
  (Renombrar este archivo a `image-full-quark.ext3` porque el initrams buscará este nombre because the initramfs will look for the file in that name, unless the configuration is updated) .
- `core-image-minimal-initramfs-quark-YYYYMMDDhhmmss.rootfs.cpio.gz`
  (Renombrar este archivo a `core-image-minimal-initramfs-quark.cpio.gz`)
- `bzImage--3.8-r0-quark-YYYYMMDDhhmmss.bin` (Renombrar este archivo a `bzImage.bin`)
- `grub.efi`
- `boot` (directory)

Para más información revisar el punto 6.1 del manual:
https://downloadmirror.intel.com/23197/eng/Quark_BSP_BuildGuide_329687_009.pdf
Para información sobre la compilación cruzada, revisar el punto 7 del manual.

Para la construcción de la imagen de Yocto Linux para la Edison se debe descargar el siguiente fichero: http://downloadmirror.intel.com/25028/eng/edison-src-ww25.5-15.tgz
Se deben repetir los pasos de la Galileo manera análoga para la Edison.
Si se desea instalar una imagen sobre una microSD, se deben renombrar los archivos generados dentro del directorio .../build/tmp/deploy/images/edison/
Si se desea flashear la memoria interna de la edison, se deben usar los archivos que están dentro del directorio .../build/toFlash/ utilizando el software de intel.
http://downloadmirror.intel.com/24910/eng/phoneflashtoollite_5.2.4.0_linux_x86_64.deb
https://software.intel.com/en-us/articles/flash-tool-lite-user-manual

B - SCRIPT PARA EJECUCIÓN DE STR2STR EN SEGUNDO PLANO

```
#!/bin/bash

REC_ID=PildoGal000
INPUT=serial://ttyUSB0:115200:8:o:1:off#nvs
OUTPUT=file:///gnss/data/$REC_ID%y%n%H.nvs::S=1
CONFIG_FILE=./conf/nvs08_10hz.cmd
MESSAGE="10 hz Data Logging"

STD_OUT=/var/log/str2str.stdout
STD_ERR=/var/log/str2str.stderr

exec ./bin/str2str -in "$INPUT" -out "$OUTPUT" -c "$CONFIG_FILE" -f 0 -msg
"$MESSAGE" -t 0 > "$STD_ERR" 2> "$STD_OUT" &
```

C - SCRIPT PARA EJECUCIÓN DEL SERVICIO DE STR2STR EN EL ARRANQUE DEL SISTEMA

```
#!/bin/bash
# Script de Prueba
### BEGIN INIT INFO
# Provides:          capturebinr
# Required-Start:    $remote_fs $syslog
# Required-Stop:     $remote_fs $syslog
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: Start capturebinr daemon at boot time.
# Description:       Enable service provided by capturebinr daemon, this captures raw data from nv08c-csm in binr format by using str2str program from rtklib_2.4.2.
### END INIT INFO



start() {
  if [ -f /var/run/str2str.pid ] && kill -0 $(cat /var/run/str2str.pid); then
       echo "Service already running"
       return 1
  fi

  echo "Starting capturebinr service..."
  cd /gnss/datatools
  ./capturebinr.sh >/dev/null 2>/dev/null &
  echo $! > /var/run/str2str.pid
  echo "Service started"
}


stop() {
  echo "Stopping capturebinr service... "
  PID=`cat /var/run/str2str.pid`
  kill -9 $PID
  rm -f /var/run/str2str.pid
  echo "Service stopped"
}
```

```
status() {
  systemctl status capturebinr.service
}

case "$1" in
  start)
       start
       ;;
  stop)
       stop
       ;;
  restart)
       stop
       start
       ;;
  status)
       status
       ;;
  *)
       echo "Usage: $0 {start|stop|restart|status}"
esac
```