UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

ESCUELA DE INGENIERÍA EN MECATRÓNICA

TRABAJO DE GRADO PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN MECATRÓNICA

TEMA:

"ESTIMACIÓN DE POSICIÓN Y VELOCIDAD EN DATOS DE POSICIONAMIENTO GLOBAL"

AUTOR: VÍCTOR STALIN LÓPEZ TAPIA

DIRECTOR: CARLOS XAVIER ROSERO CHANDI

IBARRA-ECUADOR

FEBRERO 2020

UNIVERSIDAD TÉCNICA DEL NORTE

BIBLIOTECA UNIVERSITARIA

AUTORIZACIÓN DE USO Y PUBLICACIÓN

A FAVOR DE LA UNIVERSIDAD TÉCNICA DEL NORTE

1. IDENTIFICACIÓN DE LA OBRA

En cumplimiento del Art. 144 de la Ley de Educación Superior, hago la entrega del presente trabajo a la Universidad Técnica del Norte para que sea publicado en el Repositorio Digital Institucional, para lo cual pongo a disposición la siguiente información:

DATOS DEL AUTOR			
CÉDULA DE IDENTIDAD:	100447918 – 2		
APELLIDOS Y NOMBRES:	LÓPEZ TAPIA VÍCTOR STALIN		
DIRECCIÓN:	*San Clemente*		
EMAIL:	vslopez@utn.edu.ec - stalin170296@gmail.com		
TELÉFONO FIJO:	–	**TELÉFONO MÓVIL:**	0968754336
DATOS DE LA OBRA			
TÍTULO:	"ESTIMACIÓN DE POSICIÓN Y VELOCIDAD EN DATOS DE POSICIONAMIENTO GLOBAL"		
AUTOR:	VÍCTOR STALIN LÓPEZ TAPIA		
FECHA (AAAA-MM-DD):	2020-02-06		
SÓLO PARA TRABAJOS DE GRADO			
PROGRAMA:	PREGRADO		
TÍTULO POR EL QUE OPTA:	INGENIERO EN MECATRÓNICA		
ASESOR/DIRECTOR:	CARLOS XAVIER ROSERO C.		

2. CONSTANCIAS

El autor manifiesta que la obra objeto de la presente autorización es original y se la desarrolló sin violar derechos de autor de terceros, por lo tanto la obra es original, y que es el titular de los derechos patrimoniales, por lo que asume la responsabilidad sobre el contenido de la misma y saldrá en defensa de la Universidad en caso de reclamación por parte de terceros.

Ibarra, a los 06 días del mes de Febrero de 2020.

Víctor Stalin López Tapia
C.I.: 100447918 – 2

UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

CERTIFICACIÓN

En calidad de director del trabajo de grado "ESTIMACIÓN DE POSICIÓN Y VELOCIDAD EN DATOS DE POSICIONAMIENTO GLOBAL", presentado por el egresado VÍCTOR STALIN LÓPEZ TAPIA, para optar por el título de Ingeniero en Mecatrónica, certifico que el mencionado proyecto fue realizado bajo mi dirección.

Ibarra, Enero de 2020

Carlos Xavier Rosero
DIRECTOR DE TESIS

Agradecimiento

A mis padres y a mi hermano por guiarme y apoyarme a lo largo de toda mi vida y en especial en mi formación profecional.

A mis compañeros de equipo por las experiencias compartidas en el transcurso de la carrera.

A mi director de trabajo de grado Xavier Rosero guiarme con sus conocimientos para lograr culminar este proceso.

Stalin López

Dedicatoria

Dedico este trabajo a mi hermano Lenin, a mi madre Eulalia y a mi padre Baldomero, que me brindaron su apoyo incondicional para culminar mi formación profesional.

Stalin López

Resumen

El sistema de posicionamiento global (GPS, Global Position System) tiene varias aplicaciones en la vida moderna como asistencia en carreteras, comercio móvil o manejo de vehículos no tripulados. Sin embargo, los dispositivos receptores GPS actualizan la información de posicionamiento con una frecuencia máxima de 1 Hz. El presente trabajo ofrece una aplicación que permite estimar la posición y la velocidad en los datos de posicionamiento con frecuencias superiores a 1 Hz. Para el desarrollo de la aplicación, es necesario identificar un modelo dinámico, adquirir datos del GPS, procesar los datos, identificar las matrices necesarias para el algoritmo y aplicar el filtro de Kalman. Para la adquisición de los datos se emplea un teléfono inteligente. Los datos se toman sobre un vehículo que circula las calles de la ciudad de Ibarra, en una trayectoria rectilínea y una curva. El modelo dinámico consiste en las ecuaciones de movimiento rectilíneo y el algoritmo de estimación aplicado es el filtro de Kalman para modelos lineales. La aplicación se desarrolla en lenguaje Python y se realizan pruebas a diferentes frecuencias de estimación sobre las trayectorias grabadas. Los resultados de las pruebas muestran que al aumentar la frecuencia, los datos estimados convergen con los datos reales luego de algunas iteraciones.

Abstract

The Global Positioning System (GPS) has several applications in modern life such as roadside assistance, mobile commerce or unmanned vehicle handling. However, GPS receiving devices update the positioning information with a maximum frequency of 1 Hz. This work offers an application that allows you to estimate the position and speed in the positioning data with frequencies greater than 1 Hz. For development of the application, it is necessary to identify a dynamic model, acquire GPS data, process data, identify the necessary matrices for the algorithm and apply the Kalman filter. A smartphone is used to acquire the data. The data are taken on a vehicle that circulates the streets of Ibarra city, in a straight path and curve one. The dynamic model consists of rectilinear motion equationes and the estimation algorithm applied is the Kalman filter for linear models. The application is developed in Python language and tests are carried out at different frequencies of estimation on the recorded trajectories. The test results show that as the frequency increases, the estimated data converges with the actual data after some iterations.

Índice general

Índice de figuras

Índice de cuadros

Introducción

Este trabajo de grado ha sido realizado con el *Grupo de Investigación en Sistemas Inteligentes de la Universidad Técnica del Norte (GISI-UTN).*

Problema

Los sistemas de seguimiento por radiolocalización han adquirido creciente importancia en el campo de las comunicaciones inalámbricas. En particular, el Sistema de Posicionamiento Global (GPS) basado en satélites, iniciado por el departamento de defensa de Estados Unidos (EE.UU), en 1978 revolucionó la tecnología de rastreo de localización a medida que aumentó su uso comercial. Ofrecido de forma gratuita y accesible en todo el mundo, el GPS se está convirtiendo rápidamente en una herramienta universal a medida que disminuye el costo de integrar la tecnología en vehículos, maquinaria, computadoras y teléfonos celulares [1].

Varios factores limitan la precisión del GPS. El vapor de agua y otras partículas en la atmósfera pueden disminuir la velocidad de la señal, lo que provoca un retraso de propagación. Los errores que ocurren cuando una señal rebota en un edificio o terreno antes de llegar a la antena del receptor, también pueden reducir la precisión. Así mismo, el ruido del receptor puede ocasionar errores menores [1], [2].

Los problemas citados provocan que las lecturas de posición y velocidad sobre todo en ambientes cerrados como bosques o ciudades, no sean precisas. Así, surge el problema de buscar

un mecanismo para estimar (predecir) valores más precisos de datos de navegación al tratar la información errónea con filtros y/o algoritmos predictivos [3], [4], [5].

En la actualidad existen aplicaciones para acceder a los datos de módulos receptores GPS como en [6], pero dichas aplicaciones simplemente muestran las coordenadas que envía el GPS, mas no permiten manipular los datos.

Alcance

En el presente proyecto se desarrollará una aplicación que tomará datos de posicionamiento obtenidos de un dispositivo receptor GPS, o de un repositorio digital en caso de ser necesario. Con base en esa información se estimará la posición y velocidad de un objeto móvil y se corregirá el error en los datos de posicionamiento.

Finalmente, luego de desarrollar la aplicación mediante software matemático, se validará el funcionamiento de esta por medio de nueva información.

Objetivos

Objetivo general

Estimar la posición y velocidad en datos de posicionamiento global.

Objetivos específicos

- Proponer soluciones de estimación de posición y velocidad mediante métodos matemáticos obtenidos de la literatura.
- Implementar un método matemático en el desarrollo de una aplicación para software libre.
- Validar el funcionamiento de la aplicación a través de datos reales.

Justificación

Hoy en día el GPS tiene una amplia gama de aplicaciones que incluyen seguimiento de paquetes, comercio móvil, respuesta de emergencia, exploración, topografía, cumplimiento de la ley, recreación, seguimiento de vida silvestre, búsqueda y rescate, asistencia en carretera, recuperación de vehículos robados, procesamiento de datos satelitales y administración de recursos [1], entre otras tales como [7], [8], [9]. Mismas que requieren un monitoreo continuo del posicionamiento de los objetos, debido a que la pérdida de la ubicación en lapsos de tiempo extendidos puede provocar desempeños bastante deficientes del trabajo realizado.

El GPS se usa frecuentemente para la navegación automotriz [10]. La medición de GPS de los vehículos puede recopilarse y analizarse para detectar o controlar el tráfico urbano en tiempo real. Desarrollar una aplicación que permita estimar el posicionamiento de un objeto móvil podría evitar embotellamientos o accidentes de tránsito en las vías [11].

Los vehículos terrestres no tripulados o vehículos aereos no tripulados se pueden aplicar a muchas áreas para ayudar a los humanos en diversas tareas. El desarrollo de una aplicación para software matemático libre podría facilitar el desarrollo de trabajos futuros en los que sea necesario acceder y manipular los datos de un dispositivo receptor GPS [12].

Capítulo 1

Revisión Literaria

En este capítulo se hace una descripción del sistema de posicionamiento global (GPS, Global Positioning System), y también los métodos que se usan para realizar la estimación de posición y velocidad de los datos del GPS.

1.1. Sistema de posicionamiento global (GPS)

Es un sistema que tiene como objetivo la determinación de las coordenadas espaciales de puntos respecto de un sistema de referencia mundial. Los satélites que conforman el GPS están ubicados en 6 órbitas con inclinación de 55 grados respecto al plano del Ecuador y con una distribución aproximadamente uniforme; con 4 satélites en cada órbita, dando un total de 24 satélites activos y a estos se suman 3 satélites de reserva. En la Figura 1.1 se muestra una representación de la constelación de satélites que conforman el GPS, misma que permite que en cualquier punto del planeta existan entre 5 y 11 satélites observables [13]. Los receptores GPS pueden capturar datos de posicionamiento con una frecuencia máxima de 1 Hz, es decir una muestra por segundo [14].

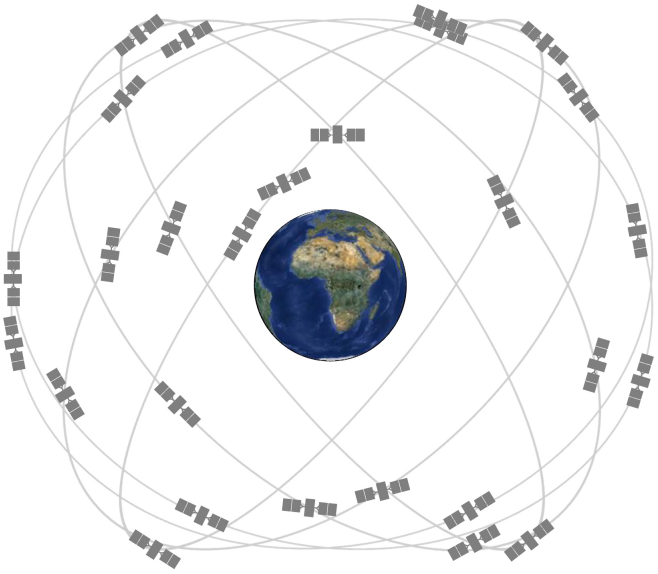

Figura 1.1: Constelación de satélites del sistema de posicionamiento global [15]

1.2. Métodos de estimación de posición y velocidad

Existen varios métodos matemáticos para estimar la posición y la velocidad de un objeto móvil partiendo de datos obtenidos por el GPS, como los que se detallan a continuación.

1.2.1. Redes neuronales dinámicas

El método se basa en inteligencia artificial (IA), que permite integrar datos del GPS con datos de un sistema de navegación inercial (INS), con el fin de estimar la posición y velocidad de un vehículo en movimiento. Para ello, se diseña una red neuronal de retardo de entrada como se muestra en la Figura 1.2. Luego se establece un modelo de error en las direcciones norte, este y vertical, que es utilizado cuando el GPS no proporciona datos. Finalmente se realizan pruebas con datos reales tomados durante 1200 segundos y se verifica el desempeño del modelo. Con este método se consigue que los errores sean mínimos a lo largo de la trayectoria como se puede ver en la Figura 1.3 [16].

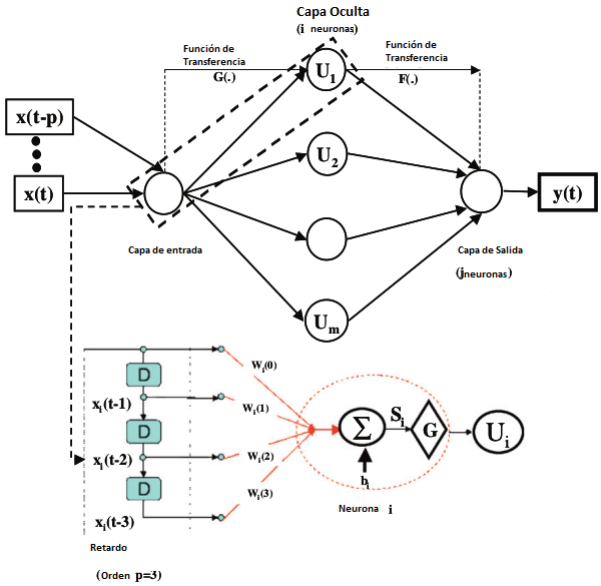

Figura 1.2: Arquitectura de red neuronal con retardo de entrada y cálculos de neuronas individuales [16]

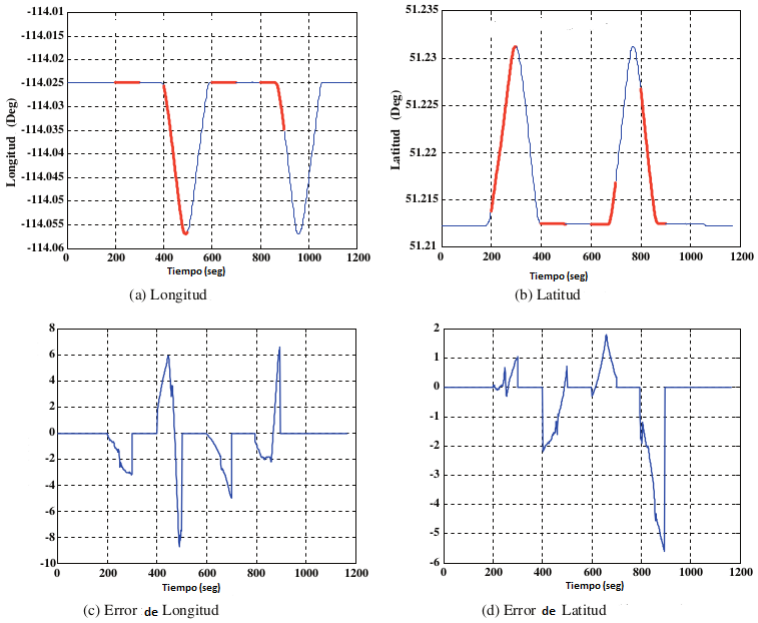

Figura 1.3: Distribución del error de posición del modelo de Redes Neuronales Dinámicas [16]

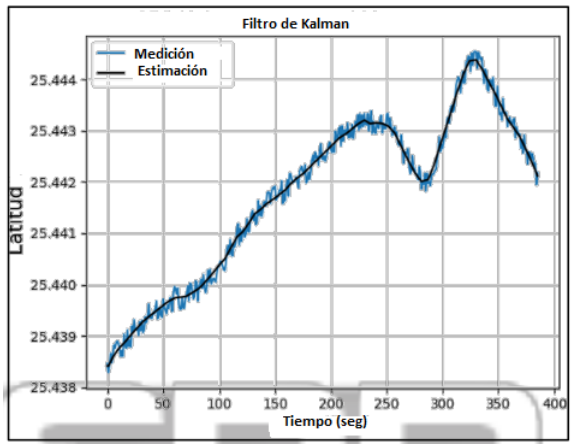

Figura 1.4: Estimación de latitud con el filtro de Kalman [17]

1.2.2. Filtro de Kalman para modelos lineales (KF, Kalman Filter)

Se aplica el filtro de Kalman a los datos obtenidos por el receptor GPS de un teléfono inteligente. Primero se recolectan los datos del GPS haciendo uso de una aplicación que permite guardar los datos en un archivo con extensión .gpx. Los datos se graban mientras se circula en una bicicleta por las calles de la ciudad. El modelo

$$\begin{cases} X_k = A \cdot X_{k-1} \\ z_k = H \cdot X_k + R, \end{cases} \tag{1.1}$$

toma en cuenta el ruido que se tiene en las mediciones. Por último se implementa el algoritmo en Python y en las Figuras 1.4 y 1.5 se observa como se comporta el filtro cuando se estima unicamente la posición [17].

1.2.3. Filtro de Kalman para modelos no lineales

Cuando se tiene sistemas dinámicos no lineales se puede aplicar el filtro de Kalman Extendido o el filtro de Kalman Unscented, que serán descritos a continuación.

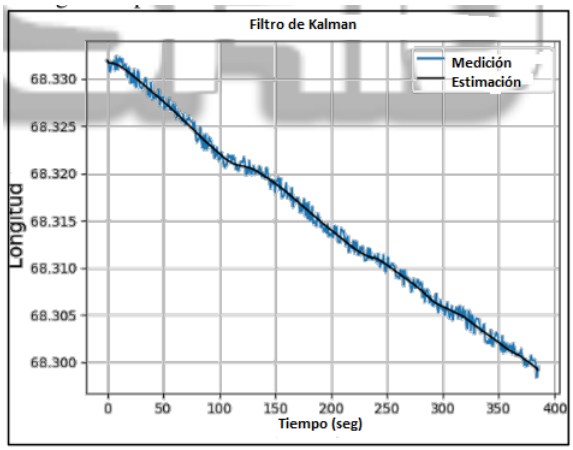

Figura 1.5: Estimación de longitud con el filtro de Kalman [17]

1.2.3.1. Filtro extendido (EKF, Extended Kalman Filter)

Con el objetivo de corregir los errores de posicionamiento obtenidos por un receptor GPS, [18] hace uso del filtro de Kalman extendido y una variación del mismo conocida como filtro de Kalman suavizado. Con ayuda de un teléfono inteligente se graban dos trayectorias, una recta y una con secciones curvas. Los datos obtenidos en un archivo KML se leen a través del software Matlab para luego ser procesados. Para determinar R, se deja el receptor GPS sobre un punto fijo y se toman varias mediciones, asumiendo que el punto real es la media de las mediciones tomadas, y con ayuda de métodos matemáticos se calcula Q. Finalmente se aplican los algoritmos respectivos y se demuestra que el filtro de Kalman suavizado es mejor que el EKF, como indican las Figuras 1.6 y 1.7.

1.2.3.2. Filtro unscented (UFK, Unscented Kalman Filter)

Para estimar la posición y la velocidad en un receptor GPS, en [19] se hace uso de un modelo no lineal, donde se representan tres posiciones y tres velocidades, que corresponden a las direcciones este, norte y altitud. Luego de establecer un modelo se aplica el filtro de Kalman extendido y el filtro de Kalman unscented para posteriormente comparar el desempeño de cada

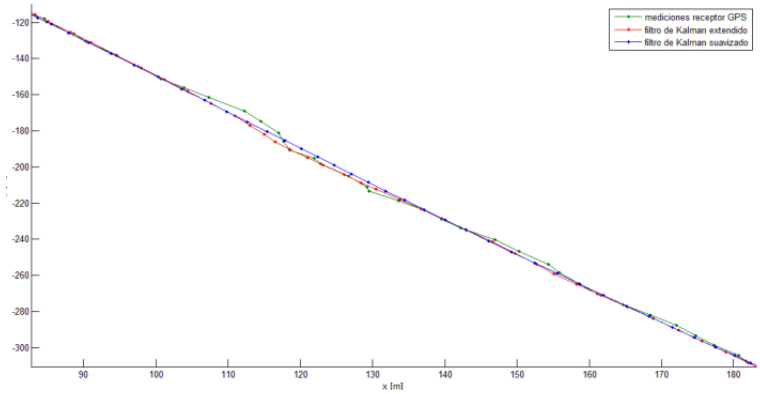

Figura 1.6: Estimación de la trayectoria recta con el filtro de Kalman extendido [18]

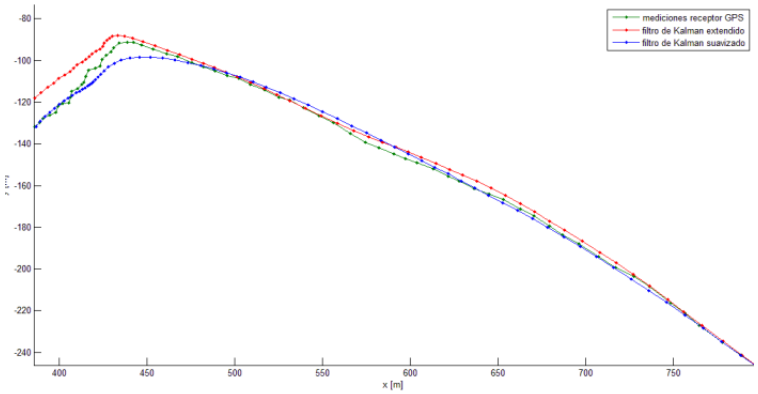

Figura 1.7: Estimación de la trayectoria con secciones curvas con el filtro de Kalman extendido [18]

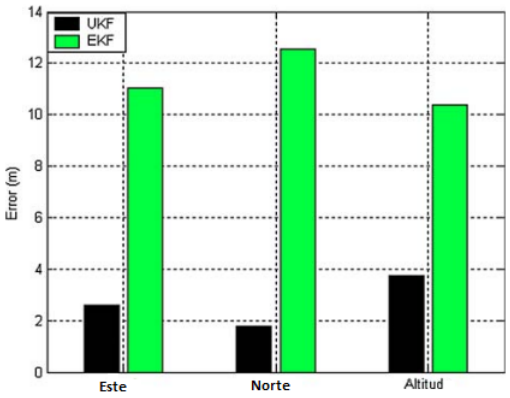

Figura 1.8: Error en la estimación con el filtro de Kalman Uncented [19]

uno. La simulación se la realiza en un computador con procesador pentium 4, el código para la simulación se desarrolla en el software Matlab y para obtener los datos se usa la herramienta SatNav disponible para Matlab. Se hace la simulación de la trayectoria de un vehículo durante 450 segundos, donde cada 50 segundos se cambia el tipo de movimiento a velocidad constante, aceleración constante, aceleración variable y movimiento circular. Finalmente, se analizan las simulaciones, obteniendo como resultados que el UKF presenta menor error que el EKF en la estimación de la posición, como se muestra en la Figura 1.8.

1.3. Propuesta

Considerando que los datos provenientes del GPS se actualizan con una frecuencia máxima de 1 Hz y que las mediciones no son exactas, es decir, contienen errores, se propone aplicar el filtro de Kalman para modelos lineales sobre los datos obtenidos por un receptor GPS, con la finalidad de estimar la posición y la velocidad con frecuencias mayores a 1 Hz.

Capítulo 2

Metodología

En este capítulo se hace una brebe descripción del funcionamiento del filtro de Kalman, también se describen los pasos necesarios para aplicar el filtro y se presenta un ejemplo numérico.

2.1. Generalidades del filtro de Kalman

El filtro de Kalman es esencialmente un conjunto de ecuaciones matemáticas que implementan un estimador de tipo predictor-corrector que es óptimo en el sentido de que minimiza la covarianza de error estimada [20]. Tiene como objetivo estimar el estado $X_k \in \mathbb{R}^n$ de un proceso controlado en tiempo discreto representado por

$$X_k = A \cdot X_{k-1} + B \cdot u_{k-1} + w_{k-1}, \tag{2.1}$$

con una medición $Y_k \in \mathbb{R}^m$

$$Y_k = H \cdot X_k + v_k, \tag{2.2}$$

donde, la matriz A de dimensión nxn relaciona el estado en el instante previo $k-1$ con el estado en el momento k. La matriz H de dimensión mxn relaciona el estado con la medición Y_k [21], [22]. Las variables aleatorias w_k y v_k representan el error del proceso y de la medición

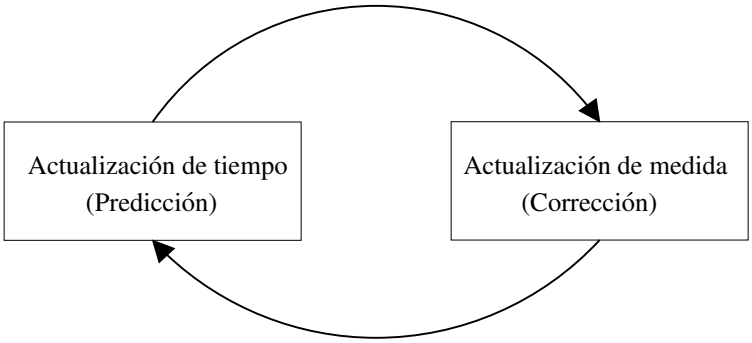

Figura 2.1: Diagrama de funcionamiento del filtro de Kalman

respectivamente, y están representadas por

$$\begin{cases} p(w) \approx N(0,Q) \\ p(v) \approx N(0,R) \end{cases} \tag{2.3}$$

El algoritmo del filtro de Kalman consiste en dos etapas: la primera se encarga de realizar una predicción del estado actual, basándose en los valores del estado anterior y el modelo dinámico del sistema. La segunda realiza una correción de dicha predicción, teniendo en cuenta las mediciones de los sensores. Como resultado de este proceso, se tiene una estimación óptima del vector de estados, cuyos valores servirán para realimentar el sistema. La Figura 2.1 muestra de una manera simplificada el funcionamiento del filtro de Kalman.

2.1.1. Ganancia de Kalman

La ganancia de Kalman K es una ponderación cuyo valor oscila entre cero y uno. Su función es dar mayor importancia al valor medido o al valor estimado. Para esto compara los errores de cada uno. Representándola de una forma sencilla se tiene

$$K = \frac{E_e}{E_e + E_m}, \tag{2.4}$$

donde E_e representa el error en la estimación y E_m en el error en la medición [23]. Si E_e es menor que E_m significa que la estimación tiene un valor más aproximado al real y en ese caso teniendo en cuenta la ecuación 2.4 la ganancia de Kalman tiende a cero. Por otro lado si E_e es mayor que E_m significaría que la medición es muy próxima al valor real y entonces, por la ecuación 2.4 la ganancia de Kalman tiende a uno.

2.1.2. Estimación del estado actual

Para entender como se realiza una estimación óptima con el filtro, una manera sencilla de representar el proceso es la siguiente

$$Estimation = Prediction + K \cdot [Medida - Prediction], \tag{2.5}$$

donde, simplemente se necesita conocer la ganancia de Kalman, tener una predicción del estado y tener los datos de las mediciones en ese instante. El valor estimado sería más cercano al valor predicho o al valor medido dependiendo de la ganancia de Kalman. Si la ganancia de Kalman tiene un valor cercano a uno la estimación tendría un valor muy cercano al valor medido y en el caso contrario en el que la ganancia de Kalman tenga un valor cercano a cero la estimación tendría un valor muy cercano al valor de la predicción.

2.1.3. Error en la estimación

El error en la estimación E_{ea} indica que tan cercanos están los valores medidos con los valores estimados

$$E_{ea} = (1 - K)E_p \tag{2.6}$$

donde, K es la ganancia de Kalman y E_p es el error en la predicción. Si la ganancia de kalman tiene un valor cercano a uno en la siguiente iteración el valor de la estimación se acercará

rápidamente al valor de la medición y si la ganancia de Kalman tiende a cero la estimación se acercará lentamente al valor de las mediciones [23].

2.1.4. Modelo matemático

Las ecuaciones que componen el filtro de Kalman se pueden dividir en dos grupos: las que se encargan de realizar la predicción (estimación a priori)

$$X_{kp} = A \cdot X_{k-1} + B \cdot u_{k-1} + w_{k-1}, y \tag{2.7}$$

$$P_{kp} = A \cdot P_{k-1} \cdot A^T + Q_k \tag{2.8}$$

y las que hacen la correción de los datos (estimación a posteriori)

$$K = \frac{P_{kp} \cdot H^T}{H \cdot P_{kp} \cdot H^T + R}, \tag{2.9}$$

$$X_k = X_{kp} + K \cdot [Y - H \cdot X_{kp}], \tag{2.10}$$

$$P_k = (I - K \cdot H) \cdot P_{kp}, \tag{2.11}$$

donde, X corresponden al vector de estados y los subindices k y $k-1$ representan el instante actual y el instante anterior respectivamente. A es la matriz de transición, u es el vector de entradas, B es la matriz de entrada w representa el error del proceso. P es la matriz de covarianza del error, Q es la matriz de covarianza de ruido en el proceso y el subindice kp indica que son las estimaciones a priori (predicción). K representa la ganancia de Kalman, H es la matriz de mediciones y H^T es la matriz transpuesta de H, R la matriz de covarianza del ruido en las mediciones, Y es el vector de mediciones e I es una matriz identidad.

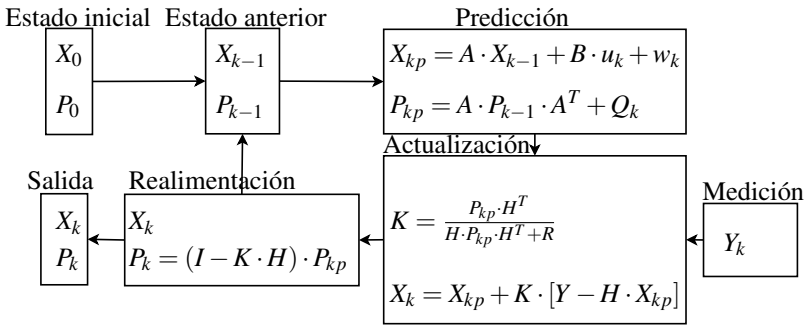

Figura 2.2: Esquema del filtro de Kalman

Para explicar el funcionamiento, se parte desde un estado inicial X_0, luego se predice el vector de estados X en el instante k, con base en el valor del estado anterior X_{k-1} y la dinámica del proceso. Después se actualiza el vector de estados con ayuda de una medición nueva y la ganancia de Kalman. Obteniendo así una estimación óptima del vector de estados X en el instante k. Finalmente X_k se realimenta en el sistema para continuar con el ciclo. En la Figura 2.2 se muestra el diagrama de funcionamiento [24].

2.2. Algoritmo propuesto

El algoritmo de estimación se muestra en el diagrama de flujo de la Figura 2.3. Se inicia con la adquisición de los datos de posicionamiento de un receptor GPS que luego se procesan antes de ser aplicados en el algoritmo. Las matrices del filtro de Kalman se inicializan con base en el modelo dinámico que debe ser identificado previamente y finalmente se aplican las ecuaciones del filtro para obtener estimaciones óptimas.

En las siguientes subsecciones se explica de manera detallada la aplicación de este algoritmo y la obtención del modelo dinámico.

2.2.1. Modelo dinámico

El modelo elegido únicamente trabaja con los datos de la posición y velocidad en los ejes X e Y, despreciando el eje z (altitud). Se considera el vehículo como una partícula que se mueve a

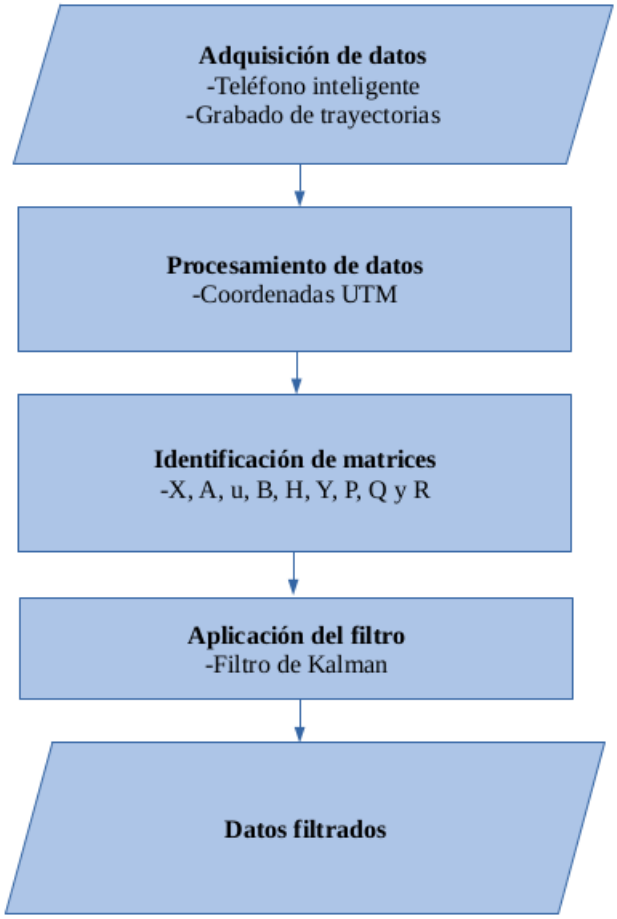

Figura 2.3: Diagrama de flujo

lo largo de una trayectoria recta en el plano xy, donde

$$X_k = X_{k-1} + V_x \Delta T + \frac{1}{2} a x_k \Delta T^2 \tag{2.12}$$

$$Y_k = Y_{k-1} + V_y \Delta T + \frac{1}{2} a y_k \Delta T^2 \tag{2.13}$$

describen la posición del vehículo en el eje X e Y respectivamente, la velocidad que lleva el vehículo está descrita por

$$V x_k = V x_{k-1} + a x_k \Delta T \tag{2.14}$$

$$V y_k = V y_{k-1} + a y_k \Delta T \tag{2.15}$$

Representando el modelo en espacio de estados en tiempo discreto se tiene

$$X_k = A \cdot X_{k-1} + B \cdot u_k, \tag{2.16}$$

$$Y_k = C \cdot X_k, \tag{2.17}$$

donde, X es el vector de estados, A es la matriz de estados, B la matriz de entrada, u es el vector de entradas, Y es el vector de salida y C es la matriz de salida. El módulo math [25] permitirá realizar las operaciones matemáticas necesarias para aplicar el filtro.

2.2.2. Adquisición de datos

Para adquirir los datos se necesitan dos partes importantes, que son: el dispositivo receptor y el objeto en movimiento. El receptor GPS usado para obtener los datos de posicionamiento consiste en un teléfono celular con sistema operativo android, haciendo uso de la aplicación GPSLogger disponible en PlayStore. Mientras que, para grabar los datos de la trayectoria real se lo hace manualmente con ayuda de la herramienta de google earth, donde se toma el punto

Figura 2.4: Trayectoria rectilinea seguida por el receptor GPS (Av. Atahualpa).

de inicio y el punto final de la trayectoria y se traza la ruta. Los datos se toman en un vehículo en movimiento sobre la trayectoria rectilínea de la Figura 2.4 ubicada en la Avenida Atahualpa y sobre la trayectoria con sección curva de la Figura 2.5 ubicada en el Redondel de Ajaví de la ciudad de Ibarra. Luego de grabar los datos se guardan en un archivo con extención .csv para posteriormente ser tratados y finalmente usados en el algoritmo. La librería Pandas [26] permitirá manipular de forma sencilla los datos obtenidos por el receptor GPS en formato csv. La línea de color verde mostrada en las Figuras 2.4 y 2.5 indica la ruta real por la cual se desplazó el vehículo, mientras que los puntos indican los datos que fueron tomados por el dispositivo con una frecuencia de 1 Hz (los datos se actualizan cada 1 segundo) para la trayectoria recta y 0.5 Hz (los datos se actualizan cada 2 segundos) para la trayectoria curva.

2.2.3. Procesamiento de datos

Los datos que se usan del archivo obtenido por la aplicación son: la posición y la velocidad. Sin embargo la posición está dada en coordenadas geográficas (latitud y longitud), por lo que es

Figura 2.5: Trayectoria con sección curva seguida por el receptor GPS (Redondel de Ajaví).

necesario transformarlas a coordenadas cartesianas, haciendo uso del sistema de coordenadas UTM (Universal Transverse Mercator), que es un sistema de proyección cartográfico basado en cuadrículas con el cual se pueden referenciar puntos sobre la superficie terrestre como se puede ver en la Figura 2.6 [27]. La Figura 2.7 muestra los datos medidos y la ruta real en coordenadas UTM de la trayectoria rectilínea. La librería utm [28], disponible para python, permite transformar coordenadas geográficas en el sistema geodésico mundial de 1984 (WGS84) al sistema de coordenadas universal transversal de Mercator (UTM). Librería Matplotlib [29] permitiráe crear graficas de los datos obtenidos.

2.2.4. Matrices del filtro de Kalman

El modelo del filtro de Kalman está descrito por las ecuaciones 2.1 y 2.17. Para identificar cada una de las matrices primero hay que conocer el vector de estados X. En este caso se desea estimar la posición y la velocidad, por ello la posición en el eje X y la posición en el eje Y corresponden a las variables de estado X_1 y X_2 respectivamente y las velocidades en X e Y

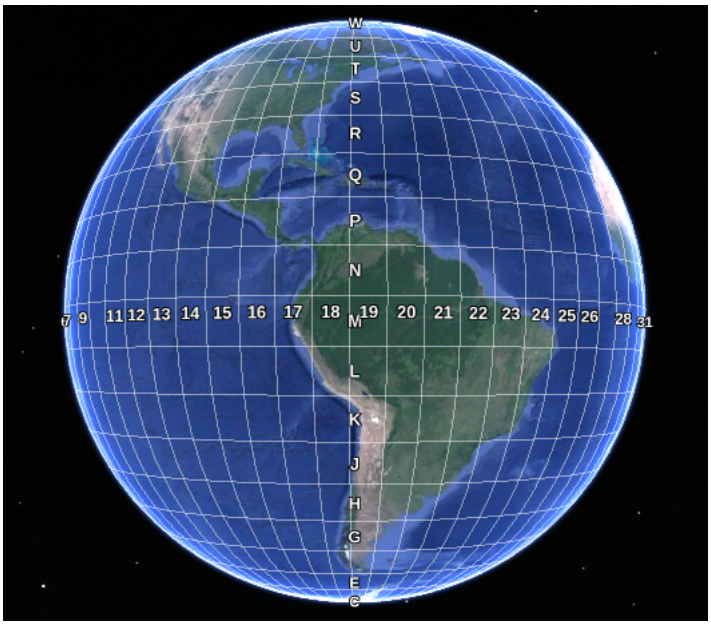

Figura 2.6: Zonas en coordenadas UTM

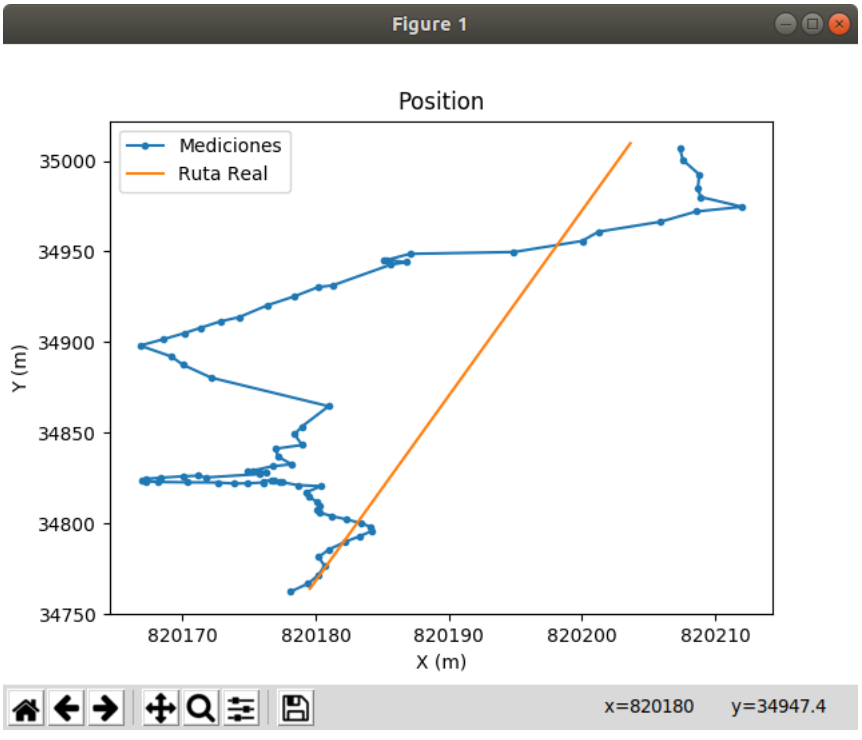

Figura 2.7: Trayectoria representada en coordenadas UTM

corresponden a X_3 y X_4, quedando el vector de estados

$$X = \begin{vmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{vmatrix} \tag{2.18}$$

y la matriz A(matriz de transición)

$$A = \begin{vmatrix} 1\,0 & \Delta T & 0 \\ 0\,1 & 0 & \Delta T \\ 0\,0 & 1 & 0 \\ 0\,0 & 0 & 1 \end{vmatrix} \tag{2.19}$$

Se toma en cuenta como entradas del sistema las aceleraciones en el eje X e Y, dando lugar al vector de entradas u

$$u = \begin{vmatrix} ax\,ay \end{vmatrix} \tag{2.20}$$

con su respectiva matriz de entrada B.

$$B = \begin{vmatrix} \frac{\Delta T^2}{2} & 0 \\ 0 & \frac{\Delta T^2}{2} \\ \Delta T & 0 \\ 0 & \Delta T \end{vmatrix} \tag{2.21}$$

La matriz de mediciones H

$$C = H = \begin{vmatrix} 1\,0\,0\,0 \\ 0\,1\,0\,0 \\ 0\,0\,1\,0 \\ 0\,0\,0\,1 \end{vmatrix} \tag{2.22}$$

corresponde a la matriz C del modelo en espacio de estados, que luego de ser multiplicada con el vector de estados da como resultado el vector de mediciones Y.

$$Y = \begin{vmatrix} X_1 \\ X_2 \\ 0 \\ 0 \end{vmatrix} \tag{2.23}$$

La matriz de covarianza del error inicial P se coloca de manera aleatoria y con un valor muy cercano a cero, debido a que el algoritmo ajustará este valor a uno adecuado en cada iteración [18].

$$P = \begin{vmatrix} 0{,}1 & 0 & 0 & 0 \\ 0 & 0{,}1 & 0 & 0 \\ 0 & 0 & 0{,}1 & 0 \\ 0 & 0 & 0 & 0{,}1 \end{vmatrix} \tag{2.24}$$

Para asignar el valor de Q, que representa la covarianza de ruido en el proceso, se hace una busqueda heurística y se selecciona un valor muy cerano a cero.

$$Q = 0{,}00001 \tag{2.25}$$

Para identificar la matriz R, que representa la covarianza del ruido en las mediciones, se toma en cuenta el valor de la exactitud que brinda la aplicación GPSLogger, que indica la desviación estándar, y sabiendo que la covarianza es igual al cuadrado de la desviación estándar se consigue la matriz diagonal

$$R = \begin{vmatrix} Exa^2 & 0 & 0 & 0 \\ 0 & Exa^2 & 0 & 0 \\ 0 & 0 & Exa^2 & 0 \\ 0 & 0 & 0 & Exa^2 \end{vmatrix} \tag{2.26}$$

Donde Exa representa la exactitud de la medición en ese instante de tiempo, lo que significa que por cada iteración el valor de R cambiaría. Con ayuda de la librería Numpy [30] se crearán

Tabla 2.1: Datos del ejemplo numérico

$Pos(m)$	$v(m/s)$	$a(m/s^2)$	$med(m)$	E_m
0	4	2		
5	4	0	3	4
9	2	-2	10	4
10	2	0	8	4
12	2	0	12	4
14	0	-2	11	4
13	3	3	15	4
17.5	4	1	16.5	4
22	0	-4	20	4
20	0	0	17	4
20	0	0	21	4
20	1	1	20	4

y manipularan de forma sencilla las matrices de N dimensiones.

2.3. Ejemplo numérico

Para entender el comportamiento del filtro de Kalman se propone un ejemplo sencillo, donde se pretende estimar la posición de un vehículo en movimiento sobre el eje X. En la Tabla 2.1 se tiene a que muestra la aceleración, v es la velocidad que lleva el vehículo, *med* es la medición que se tiene de la posición y E_m es el error que se tiene en las mediciones. Se considera ΔT igual a un segundo y aplicando la ecuación de movimiento rectilíneo uniformemente acelerado

$$X = X_{k-1} + v_{k-1}\Delta T + \frac{1}{2}a\Delta T^2, \tag{2.27}$$

se obtiene la posición que debería tener el vehículo idealmente *Pos*. Para calcular la ganancia de Kalman K, se aplica la ecuación 2.4, asumiendo que el error inicial en la estimación E_e es de 3 metros

$$K = \frac{E_e}{E_e + E_m} = \frac{3}{3+4} = 0{,}43 \tag{2.28}$$

Tabla 2.2: Valores de las variables del filtro de Kalman

$Pos(m)$	$v(m/s)$	$a(m/s^2)$	$med(m)$	E_m	E_{ea}	K	$Estimation$
0	4	2			3		
5	4	0	3	4	1.71	0.43	4.14
9	2	-2	10	4	1.20	0.30	9.30
10	2	0	8	4	0.92	0.23	9.54
12	2	0	12	4	0.75	0.19	12.00
14	0	-2	11	4	0.63	0.16	13.53
13	3	3	15	4	0.55	0.14	13.27
17.5	4	1	16.5	4	0.48	0.12	17.38
22	0	-4	20	4	0.43	0.11	21.79
20	0	0	17	4	0.39	0.10	19.71
20	0	0	21	4	0.35	0.09	20.09
20	1	1	20	4	0.32	0.08	20.00

luego, se realiza la estimación de la posición con ayuda de la ecuación 2.5

$$Estimation = Prediction + K \cdot [Medida - Prediction] = 5 + 0{,}43(3-5) = 4{,}14 \tag{2.29}$$

a continuación se calcula el error en la estimación con la ecuación 2.6

$$E_{ea} = (1-K)E_e = (1-0{,}43)(3) = 1{,}71 \tag{2.30}$$

luego, se vuelve a repetir el proceso varias veces y se obtiene como resultado la Tabla 2.2, donde se observa que las estimaciones se aproximan al valor real luego de algunas iteraciones.

Capítulo 3

Resultados

En este capítulo se describe la aplicación desarrollada y se realizan pruebas sobre la misma, para luego validar su funcionamiento.

3.1. Aplicación

En el entorno de desarrollo integrado (IDE, Integrated Development Environment) Pycharm se realiza la programación del algoritmo. El código se presenta en el anexo A.1.

Para hacer uso de la aplicación es necesario tener instalado Python en su version 2.7, el IDE Pycharm y descargar los archivos del enlace adjunto[1]. Primero se abre Pycharm y se crea un proyecto, luego se asigna un nombre al proyecto y en la sección de intérprete se selecciona Python 2.7, tal y como se muestra en la Figura 3.1.

Luego se mueve los archivos descargados a la dirección en la que se encuentra el proyecto creado como en la Figura 3.2. El archivo con extensión *.py* es el código de la aplicación, mientras que los archivos con extensión *.csv* son los datos de las trayectorias grabadas por el receptor GPS y los puntos de la trayectoria real que siguió el vehículo. Para realizar pruebas sobre otra trayectoria se deben reemplazar las columnas de "lat", "lon", "accuracy" y "speed", por otros datos de latitud, longitud, exactitud y velocidad respectivamente como se indica en la Tabla 3.1.

[1] `https://github.com/stalin170296/Position-and-Velocity-Estimation-from-GPS-Data`

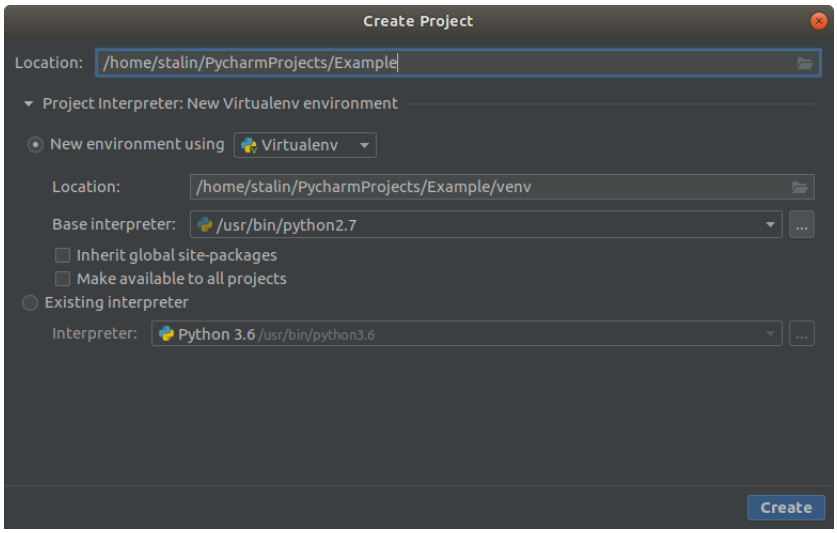

Figura 3.1: Ventana para crear un proyecto en Pycharm

Tabla 3.1: Datos necesarios para el algoritmo

lat	*lon*	*Exactitud*	*Velocidad*
0.316315	-78.123551	27	6.5
0.316258	-78.123549	21	6.75
0.316186	-78.123538	21	7
0.316116	-78.123539	20	7.5
0.316074	-78.123537	18	6
0.316025	-78.123509	17	6.5
0.316002	-78.12354	17	5.5
0.315951	-78.123564	15	5.25
0.315901	-78.123606	14	5
...	...	...	...

Después, en Pycharm, se instalan las librerías necesarias, que son: pandas [26], math [25], numpy [30], matplotlib [29] y utm [28] como se muestra en la Figura 3.3.

La frecuencia de estimación se representa por la variable *Niter*, que puede ser reemplazada por cualquier valor entero positivo. La variable *time* indica el intervalo de tiempo entre cada actualización de la información.

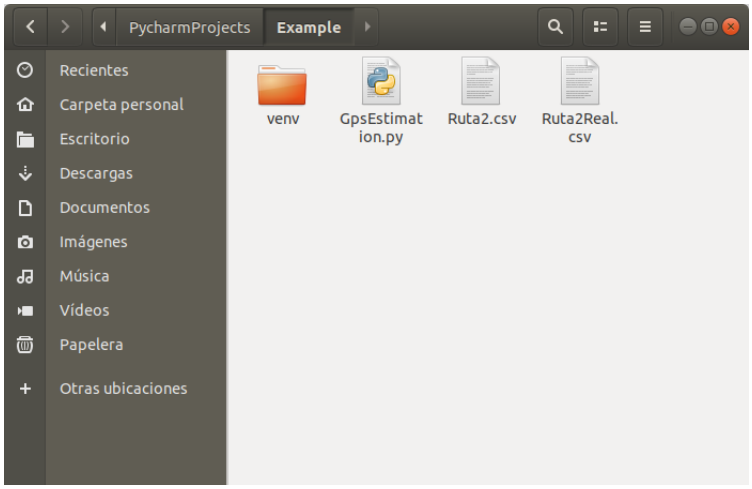

Figura 3.2: Dirección donde se copian los archivos descargados

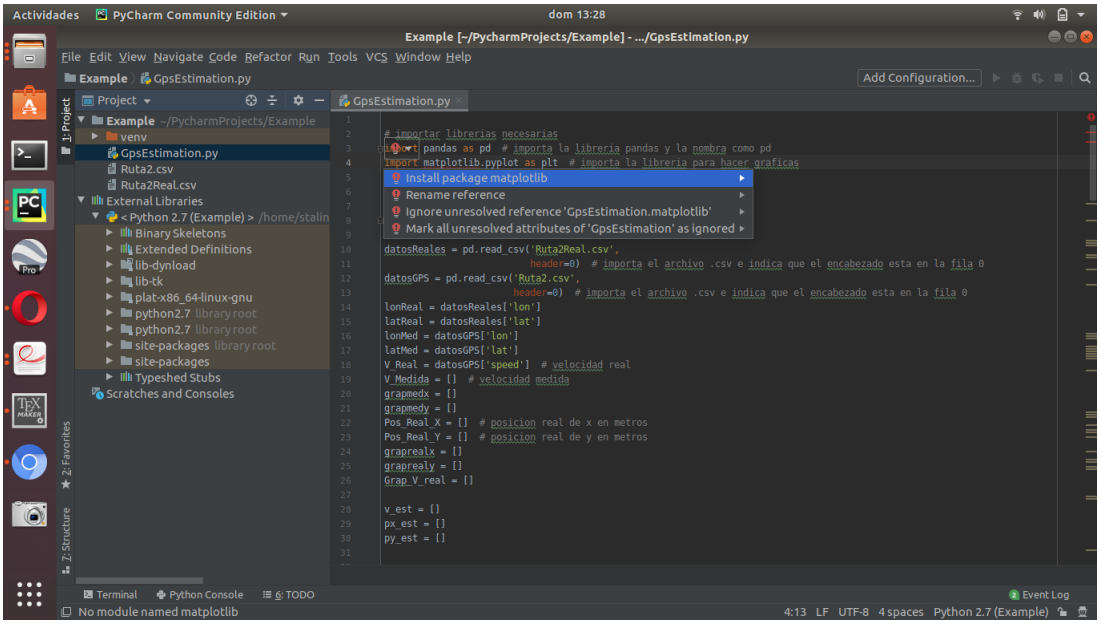

Figura 3.3: Instalación de librerías en Pycharm

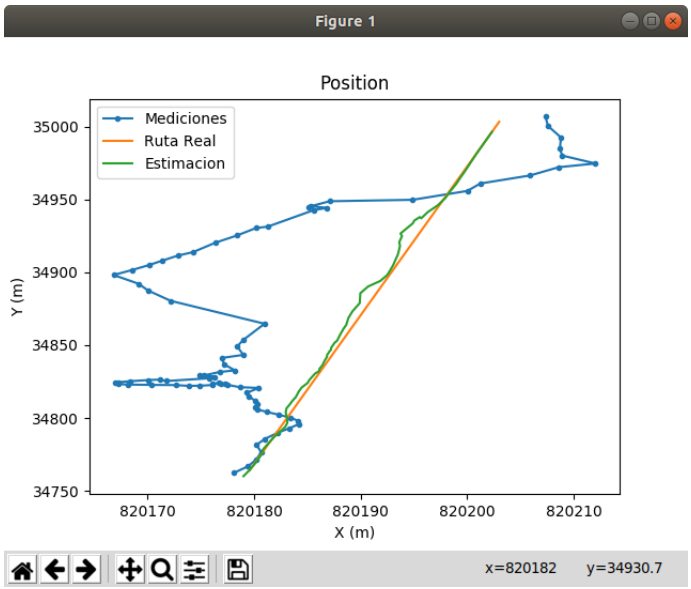

Figura 3.4: Estimación de la posición con frecuencia de 1 Hz

3.2. Pruebas y resultados

En esta sección se realizan estimaciones a diferentes frecuencias sobre la trayectoria rectilínea y no rectilínea.

3.2.1. Estimación sobre trayectoria rectilínea

Los datos se tomaron a lo largo de una trayectoria recta con una frecuencia de 1 Hz. Primero se aplica el algoritmo únicamente a los datos de cada observación (Frecuencia de 1Hz), para verificar el comportamiento del algoritmo, obteniendo los resultados de las Figuras 3.4 y 3.5, donde, la línea de color azul muestan los datos medidos, la de color naranja la trayectoria real y la de color verde muestra las estimaciones realizadas. Luego, cambiando el valor de $Niter = 1$ por $Niter = 20$ se realiza una estimación de la posición y la velocidad con una frecuencia de 20 Hz. En la Figura 3.6 se muestran los resultados de la posición y en la Figura 3.7 de la velocidad. Donde se puede ver que la gráfica de la velocidad real y la estimada se sobreponen. En las Figuras 3.8 y 3.9 se observa las estimaciones de la posición y velocidad respectivamente con una frecuencia de 100 Hz. Donde, al igual que en el caso anterior la velocidad real y la estimada

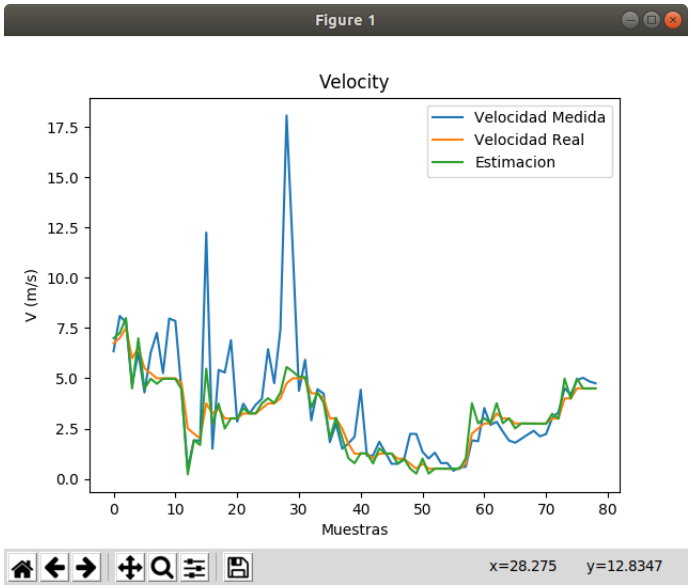

Figura 3.5: Estimación de la velocidad con frecuencia de 1 Hz

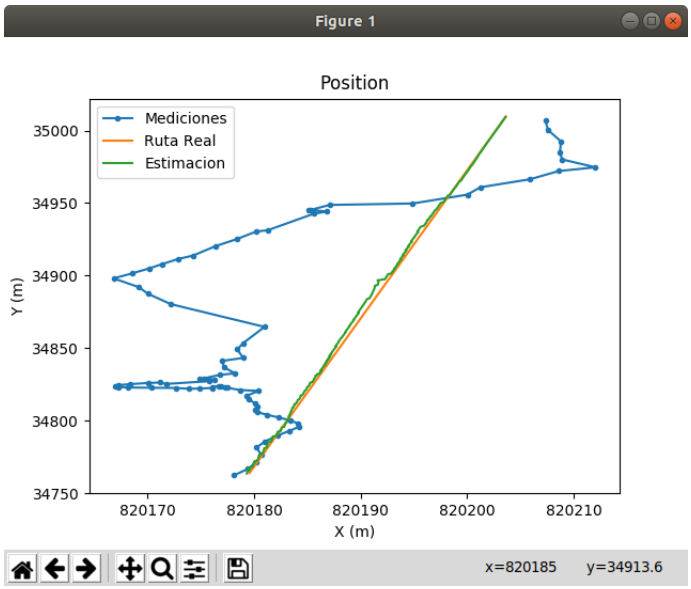

Figura 3.6: Estimación de la posición con frecuencia de 20 Hz

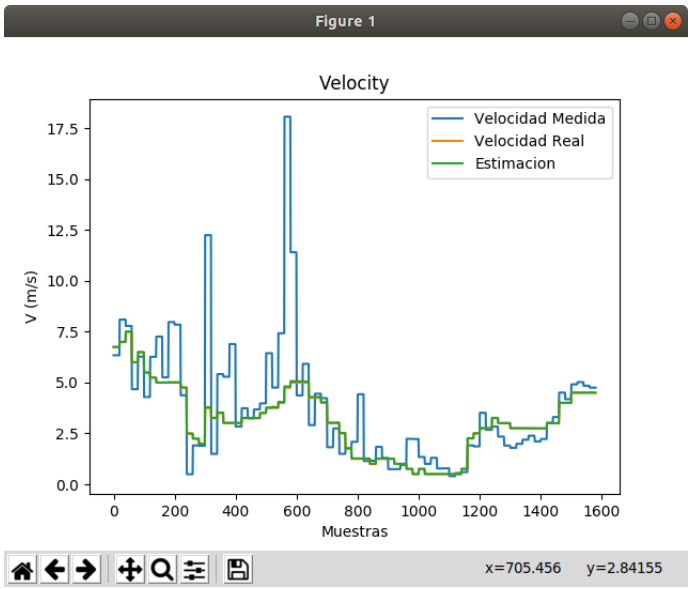

Figura 3.7: Estimación de la velocidad con frecuencia de 20 Hz

se sobreponen.

3.2.2. Estimación sobre trayectoria con sección curva

Se aplica el filtro sobre la trayectoria curva para observar el desempeño del mismo. Las Figuras 3.10 y 3.11 representan la estimación de la posición y velocidad respectivamente, con una frecuencia de 1 Hz. Luego, se aumenta la frecuencia a 20 Hz, obteniendo como los resultados mostrados en las Figuras 3.12 y 3.13. Las Figuras 3.14 y 3.15 muestran la estimación de la posición y la velocidad respectivamente, con una frecuencia de 100 Hz.

3.3. Validación del algoritmo

Para validar el algoritmo sobre la trayectoria rectilínea, se ingresan nuevos datos a la trayectoria, con la diferencia que las mediciones y los datos reales tienen el mismo valor, como se muestra en las Figuras 3.16 y 3.17. Por otra parte, para validar el funcionamiento del algoritmo sobre la trayectoria curva, se divide la trayectoria curva en varias secciones rectilíneas y se apli-

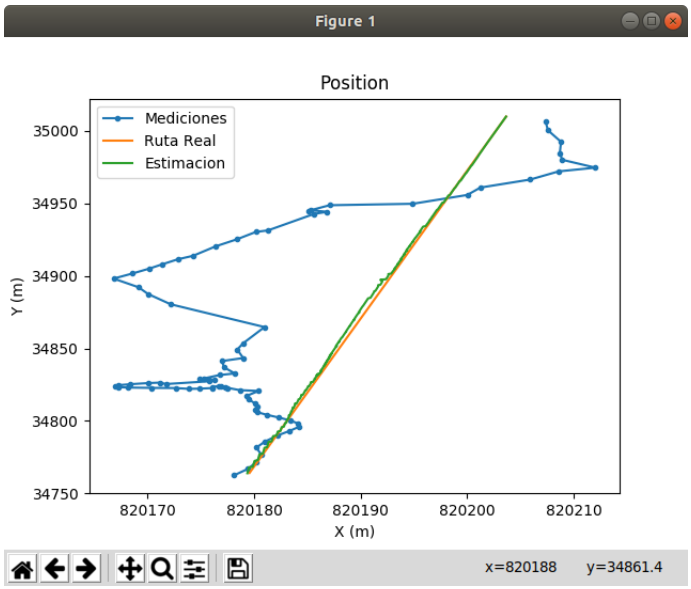

Figura 3.8: Estimación de la posición con frecuencia de 100 Hz

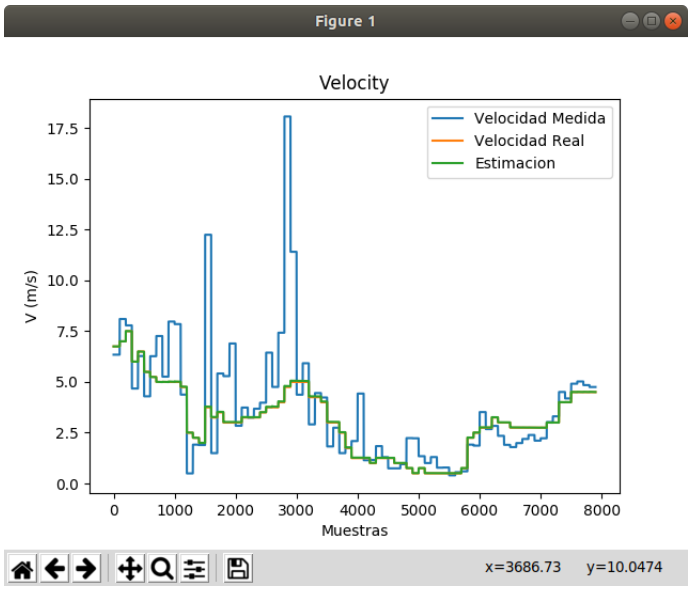

Figura 3.9: Estimación de la velocidad con frecuencia de 100 Hz

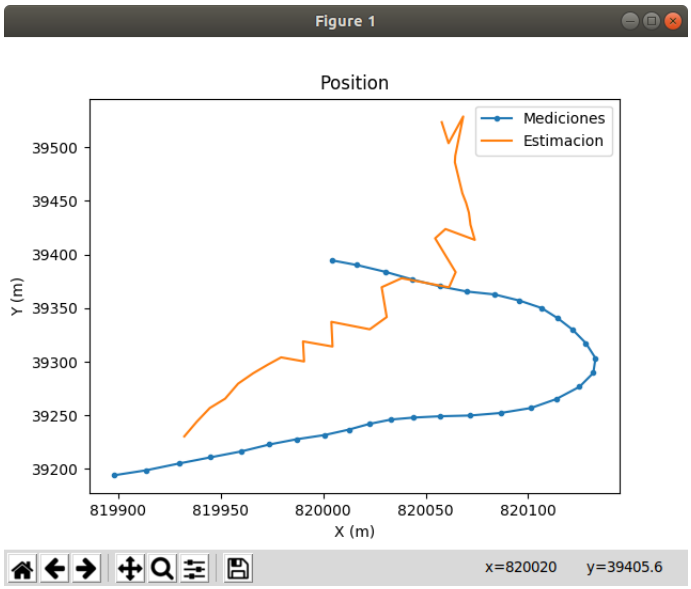

Figura 3.10: Estimación de la posición sobre una trayectoria curva con frecuencia de 1 Hz

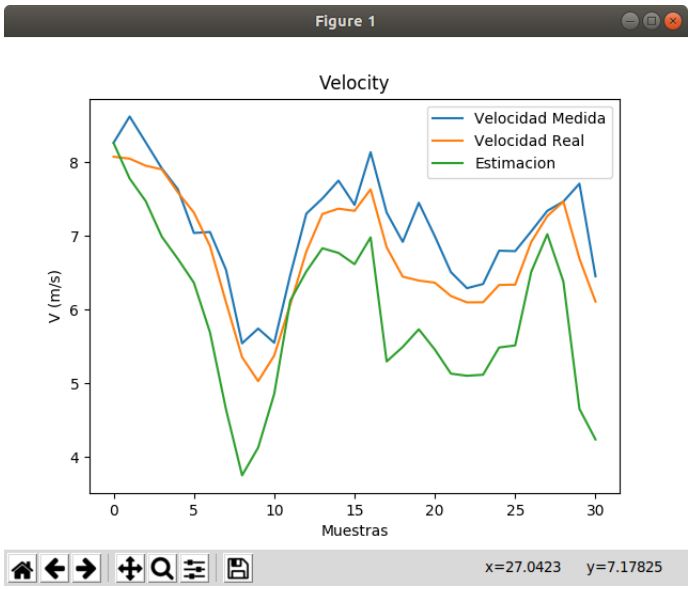

Figura 3.11: Estimación de la velocidad sobre una trayectoria curva con frecuencia de 1 Hz

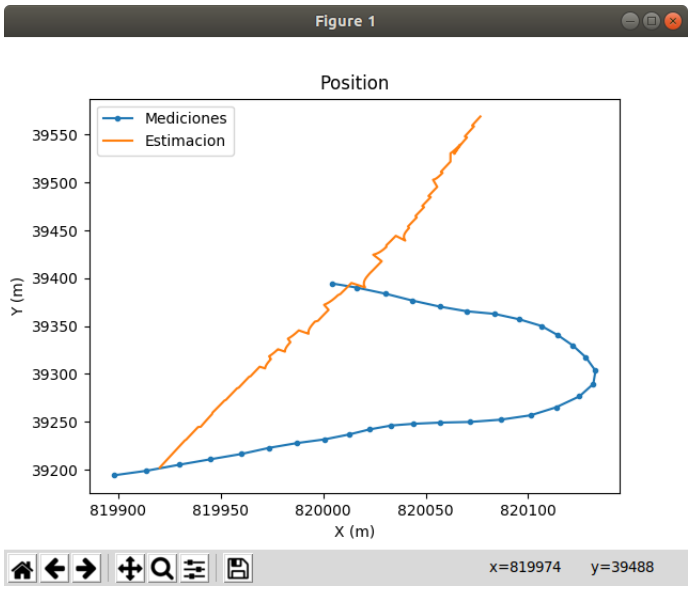

Figura 3.12: Estimación de la posición sobre una trayectoria curva con frecuencia de 20 Hz

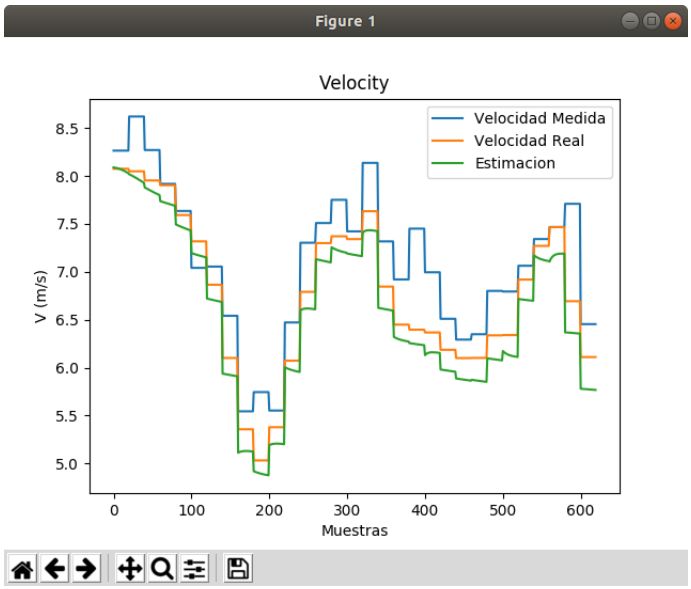

Figura 3.13: Estimación de la velocidad sobre una trayectoria curva con frecuencia de 20 Hz

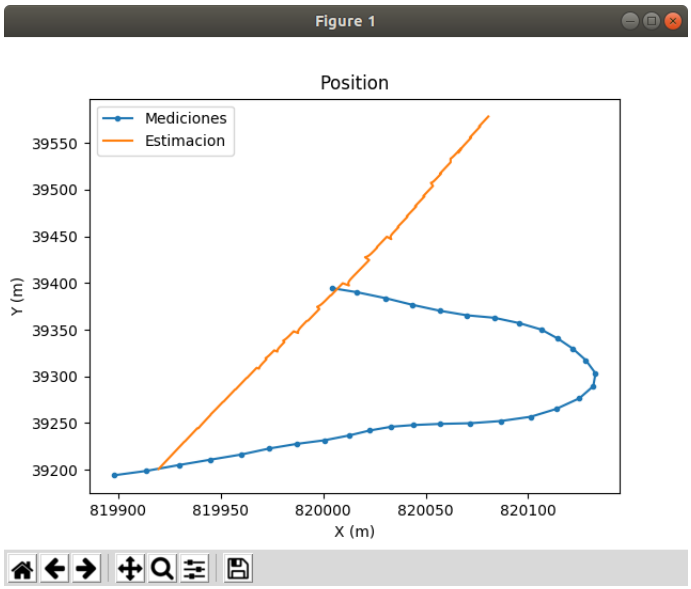

Figura 3.14: Estimación de la posición sobre una trayectoria curva con frecuencia de 100 Hz

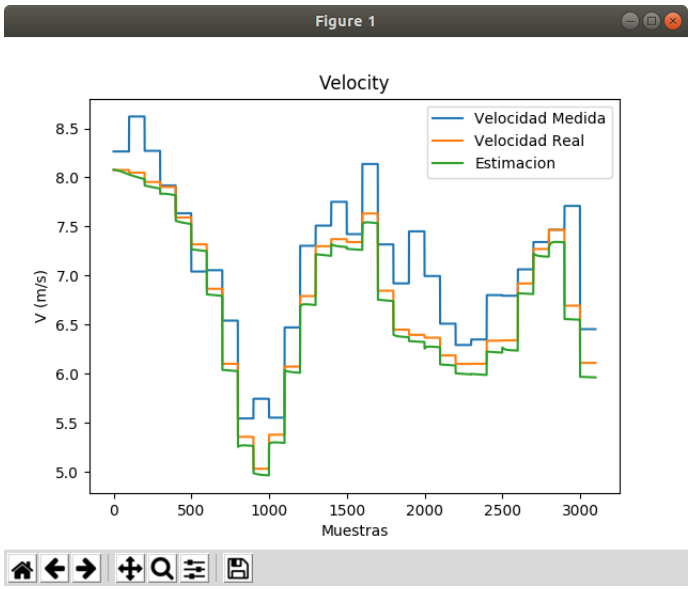

Figura 3.15: Estimación de la velocidad sobre una trayectoria curva con frecuencia de 100 Hz

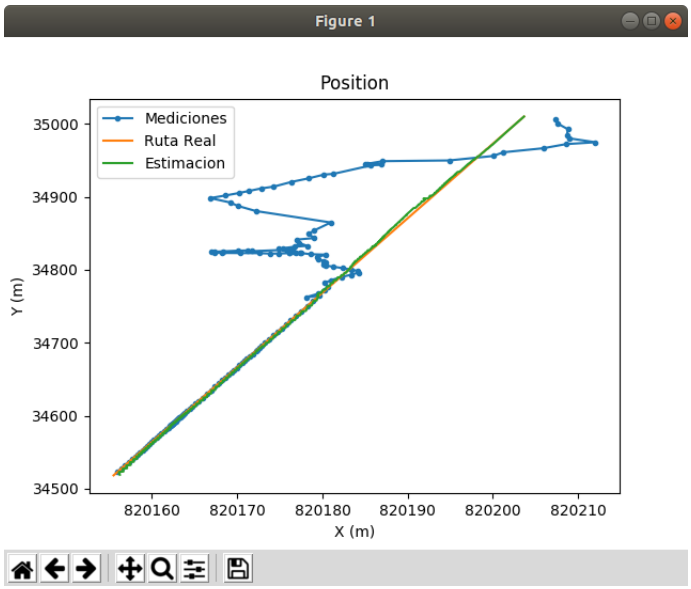

Figura 3.16: Validación de la estimación de la posición sobre la trayectoria rectilínea

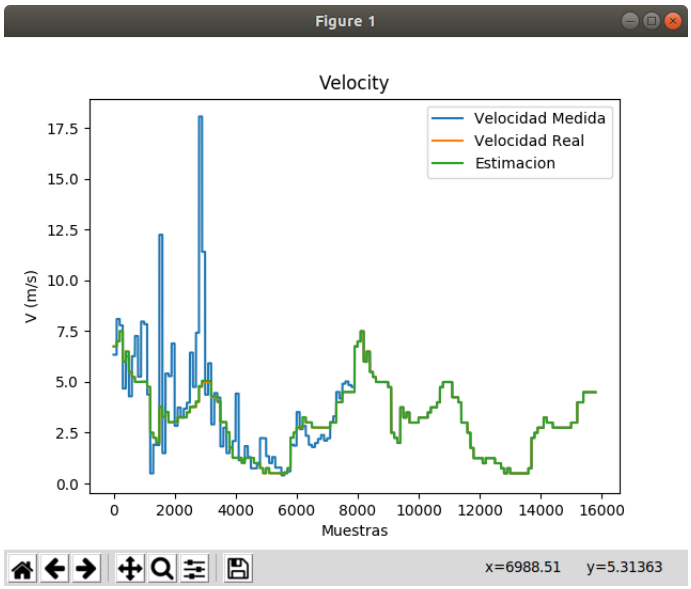

Figura 3.17: Validación de la estimación de la velocidad sobre la trayectoria rectilínea

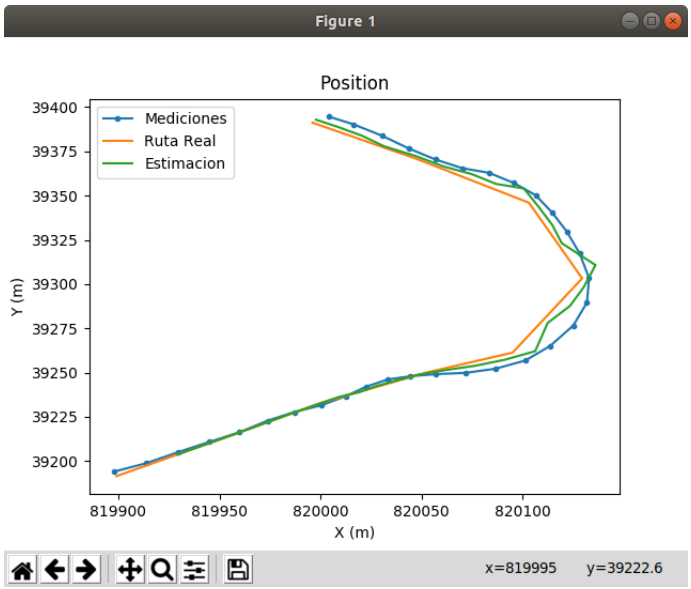

Figura 3.18: Validación de la estimación de la posición sobre la trayectoria curva

ca el algoritmo. Los resultados de la posición se observan en la Figura 3.18 y de la velocidad en la Figura 3.19

3.4. Análisis de resultados

En las Figuras 3.16 y 3.17 se observa que las gráficas de la estimación, la ruta real y los datos medidos convergen luego de algunas iteraciones, con lo que comprueba que el algoritmo ajusta automaticamente los valores de la matríz P, para realizar estimaciónes óptimas.
Se observa que, al aumentar la frecuencia, las estimaciones se acercan más a los valores reales y se alejan de las mediciónes del receptor GPS. Esto se debe a que al aumentar la frecuencia, la covarianza P se mantiene con valores mucho menores que la matriz de covarianda de ruido en las mediciones R y como consecuencia, el algoritmo aleja las estimaciones de las mediciones.
La Figura 3.18 muestra que al dividir la curva en varias secciones rectilíneas y aplicar el filtro de Kalman sobre estas, el algoritmo tiene un mejor desempeño.

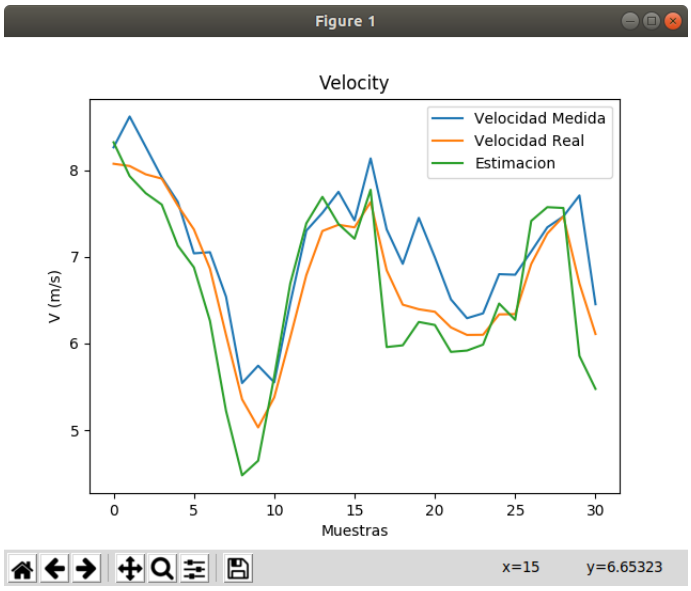

Figura 3.19: Validación de la estimación de la velocidad sobre la trayectoria curva

Conclusiones y trabajo futuro

Este capítulo muestra las conclusiones del presente proyecto y esboza algunas líneas posibles de trabajo futuro.

Conclusiones

- La aplicación desarrollada permite estimar la posición y la velocidad sobre una trayectoria rectilínea con frecuencias superiores a 1 Hz, que es la frecuencia con la que se obtienen los datos del GPS.
- Para realizar estimaciónes sobre una trayectoria curva es necesario dividir la trayectoria en varias secciones y aplicar el filtro de Kalman sobre cada una de ellas.
- Los métodos matemáticos usados para estimar la posición y velocidad en datos de posicionamiento se basan en el filtro de Kalman, tanto para modelos lineales como no lineales.
- El IDE Pycharm permite desarrollar aplicaciones en lenguaje Python de una forma rápida y sencilla.
- El método usado para validar el funcionamiento de la aplicación demuestra que, el filtro de Kalman realiza estimaciones óptimas en cada una de las iteraciones.

Trabajo futuro

En este proyecto se realiza una aplicación que permite estimar la posición y velocidad de un objeto en movimiento. Para ampliar y mejorar la aplicación se propone como trabajo futuro:

- Probar otras técnicas de estimación que permitan realizar estimaciones sobre modelos no lineales, como el filtro de Kalman Extendido.
- Integrar sensores inerciales (giroscopios y acelerómetros) con los datos de posicionamiento para mejorar la estimación de los datos en altas frecuencias.

Bibliografía

[1] R. Bajaj, S.L. Ranaweera, y D. Agrawal, "GPS: location-tracking technology", *Computer*, pp. 92-94, 2002.

[2] W. Xiang, X. Li, y M.M. Rana "Position and Velocity Estimations of Mobile Device Incorporate GNSS", *IEEE Access*, vol. 6, no. 10, pp. 31141-31147, 2018.

[3] P. Srinivas y A- Kumar, "Overview of architecture for GPS-INS integration", *Recent Developments in Control, Automation and Power Engineering (RDCAPE)*, pp. 433-438, 2017.

[4] X. Wang y M. Liang, "GPS Positioning Method Based on Kalman Filtering", *American Control Conference*, pp. 2831-2836, 2018.

[5] C. Zeng, W. Li, B. Bi y Q. Chen, "Application of extended Kalman filter based semi-codeless for tracking high dynamic GPS L2 signal", *2nd International Conference on Frontiers of Sensors Technologies (ICFST)*, pp. 284-289, Shenzhen, 2017.

[6] S. Montoya, "gidahatari", 24 Julio 2016. [En línea]. Recuperado de: http://gidahatari.com/ih-es/como-hacer-un-geolocalizador-con-una-raspberry-pi. [Accessed 27 Julio 2018].

[7] S. Desdín, A. Osa, Y. Guerrero, N. Rodríguez, M. Chávez y A. Rodríguez, "Empleo del Sistema de Posicionamiento Global (GPS) en el Manejo de Ecosistemas Agrícolas Sostenibles", *Ciencias de la Tierra y el Espacio*, pp. 69-75, 2010.

[8] Y. Li, C. Huang y J. Jiang “Research of bus arrival prediction model based on GPS and SVM”, *The 30th Chinese Control and Decision Conference (2018 CCDC)*, pp. 575-579, 2018.

[9] R. Miralles, “Implementación del Sistema de Localización y Obtención del Mapa de un Robot Móbil”, 2015.

[10] M. Spcmgenberg, V. Calmettes y J. Tourneref, “Fusion of GPS, INS and odometric data for automotive navigation”, *15th European Signal Processing Conference*, 2007.

[11] P.-J. Tseng, C.-C. Hung, Y.-H. Chuang, K. Kao, W.-H. Chen y C.-Y. Chiang, “Scaling the Real-Time Traffic Sensing with GPS Equipped Probe Vehicles”, *IEEE 79th Vehicular Technology Conference (VTC Spring*, 2014.

[12] C. Huang, Y. Liu, Y. Jia y H. Chen, “Position estimation for an unmanned ground car (UGC) by multi-sensor fusion under random loss of GPS signals”, *The 5th Annual IEEE International Conference on Cyber Technology in Automation, Control and Intelligent Systems*, pp. 1000-1005, June 2015.

[13] E. Huerta, A. Mangiaterra y G. Noguera ,“GPS Posicionamiento satelital”, *Universidad Nacinal de Rosario*, 2005.

[14] S. Mancebo ,“Análisis de precisión y eficiencia de receptores GPS bajo cobertura arbórea”, in*Universidad Politécnica de Madrid*, 2004.

[15] O. C. N. Posicionamiento ,“Sistema de Posicionamiento Global”, [En línea]. Recuperado de: https://www.gps.gov/spanish.php, 2019.

[16] A. Noureldin, A. El-shafie y M. Bayoumi, “GPS/INS integration utilizing dynamic neural networks for vehicular navigation”, *Special Issue on Intelligent Transportation Systems*,Journal , pp. 48-57 January, 2011.

[17] S. Muhammad, N. Laghari, M. Ali y M. Farrukh,“GPS Estimation using Kalman Filter”, *International Journal for Scientific Research and Development*, 2018.

[18] F. Crovo, “Análisis de trayectoria guiada por GPS utilizando métodos de estimación”, *Universidad de Concepción*, 2014

[19] D. Jwo y C. Lai, “Unscented Kalman filter with nonlinear dynamic process modeling for GPS navigation”, *GPS Solutions*,pp. 249-260, 2007

[20] G. Welch y G. Bishop, “An Introduction to the Kalman Filter”, *University of North Carolina*, 2006.

[21] M. Laaraiedh, “Implementation of Kalman Filter with Python Language”, *University of Rennes*, 2012

[22] A. Solera, “El Filtro de Kalman”, *Departamento de Investigaciones Económicas*, 2003

[23] M. Aranda, “Estudio y aplicación del Filtro de Kalman en fusión de sensores en UAVs”, *Escuela Técnica Superior de Ingeniería Universidad de Sevilla*, 2017

[24] M. Biezen, “The Kalman Filter”, [En línea]. Recuperado de: www.ilectureonline.com, 2016

[25] P. S. Fundation, “Mathematical functions”, [En línea]. Recuperado de: https://docs.python.org/2/library/math.html?highlight=math, 2019

[26] P. Community, “Python Data Analysis Library”,[En línea]. Recuperado de: https://pandas.pydata.org/, 2019

[27] S. Ibáñez, J. Gisbert y H. Moreno, “El Sistema de Coordenadas UTM”, *Escuela Técnica Superior de Ingeniería Agronómica y del Medio Natural*, 2011

[28] T. Bieniek, B. Andel “utm”, [En línea]. Recuperado de: https://github.com/Turbo87/utm, 2019

[29] J. Hunter, D. Dale, E. Firing, M. Droettboom, “Matplotlib”, [En línea]. Recuperado de: https://matplotlib.org/, 2019

[30] E. Travis, “Numpy”, [En línea]. Recuperado de: https://numpy.org/ ,2019

[31] G. Rossum, F. Drake, “El tutorial de Python”, [En línea]. Recuperado de: http://docs.python.org.ar/tutorial/, 2017

Apéndice

.A. Aplicación en Python

Programa 1: Código de la aplicación

```
# importar librerias necesarias
import pandas as pd  # importa la libreria pandas y la nombra como pd
import matplotlib.pyplot as plt  # importa la libreria para hacer graficas
import numpy as np
from numpy.linalg import inv
import math
import utm

datosReales = pd.read_csv('AjaviCurvaMediaReal2.csv',
                          header=0)  # importa el archivo .csv e indica que el encabezado esta
                              en la fila 0
datosGPS = pd.read_csv('AjaviCurvaMedia.csv',
                       header=0)  # importa el archivo .csv e indica que el encabezado esta en
                            la fila 0
lonReal = datosReales['lon']
latReal = datosReales['lat']
lonMed = datosGPS['lon']
latMed = datosGPS['lat']
V_Real = datosGPS['speed']  # velocidad real
V_Medida = []  # velocidad medida
grapmedx = []
grapmedy = []
Pos_Real_X = []  # posicion real de x en metros
Pos_Real_Y = []  # posicion real de y en metros
graprealx = []
graprealy = []
Grap_V_real = []

v_est = []
px_est = []
py_est = []


def BEARING(lat1, lon1, lat2, lon2):
    lat1 = math.radians(lat1)
    lat2 = math.radians(lat2)
```

```
    lon1 = math.radians(lon1)
    lon2 = math.radians(lon2)
    dlat = lat2 - lat1
    dlon = lon2 - lon1
    R = 6372.795477598
    a = (math.sin(dlat / 2)) ** 2 + math.cos(lat1) * math.cos(lat2) * (math.sin(dlon / 2)) **
        2
    distancia = 2 * R * math.asin(math.sqrt(a))
    distancia = distancia * 1000
    x = math.sin(dlon) * math.cos(lat2)
    y = (math.cos(lat1) * math.sin(lat2) - math.sin(lat1) * math.cos(lat2) * math.cos(dlon))
    track_angle = math.atan2(x, y)
    track_angle = math.degrees(track_angle)
    compass_bearing = (track_angle + 360) % 360
    posX = distancia * math.cos(track_angle)
    posY = distancia * math.sin(track_angle)
    return distancia


def GEOTOUTM():
    for i in np.arange(0, len(lonMed)):
        (x, y, zonanumber, zonaletter) = utm.from_latlon(latMed.ix[i], lonMed.ix[i])
        grapmedx.append(x)
        grapmedy.append(y)

    for i in np.arange(0, len(lonReal)):
        (x, y, zonanumber, zonaletter) = utm.from_latlon(latReal.ix[i], lonReal.ix[i])
        Pos_Real_X.append(x)
        Pos_Real_Y.append(y)


def GRAFICAS():
    plt.plot(grapmedx, grapmedy, '.-', label='Mediciones')
    plt.plot(Pos_Real_X, Pos_Real_Y, '-', label='Ruta Real')
    plt.plot(px_est, py_est, '-', label='Estimacion')
    plt.xlabel('X (m)')  # nombra el eje x
    plt.ylabel('Y (m)')  # nombra el eje y
    plt.title('Position')  # pone titulo a la grafica
    plt.legend()
    plt.show()

    plt.clf()
    plt.plot(V_Medida, '-', label='Velocidad Medida')
```

```
    plt.plot(Grap_V_real, '-', label='Velocidad Real')
    plt.plot(v_est, '-', label='Estimacion')
    plt.xlabel('Muestras')  # nombra el eje x
    plt.ylabel('V (m/s)')  # nombra el eje y
    plt.title('Velocity')  # pone titulo a la grafica
    plt.legend()
    plt.show()

def kf_predict(X, P, A, Q, B, U):
    X = np.dot(A, X) + np.dot(B, U)
    P = np.dot(np.dot(A, P), A.T) + Q
    return X, P

def kf_update(X, P, Y, H, R):
    V = Y - np.dot(H, X)
    S = R + np.dot(np.dot(H, P), H.T)
    K = np.dot(np.dot(P, H.T), inv(S))
    X = X + np.dot(K, V)
    P = np.dot(P, (I - np.dot(K, H)))
    return X, P

GEOTOUTM()

Niter = 2
time = 2 #tiempo de muestreo
dt = (1.0 / Niter)*time
# aceleracion en x e y
ax = 0.0
ay = 0.0
#m = (Pos_Real_Y[1] - Pos_Real_Y[0]) / (Pos_Real_X[1] - Pos_Real_X[0])  # calculo de la
    pendiente
#ang = math.atan((m))  # calculo del angulo (respuesta en radianes)
px = Pos_Real_X[0]  # posicion inicial x
py = Pos_Real_Y[0]  # posicion inicial y
# espacio de estados
A = np.array([[1.0, 0.0, dt, 0.0], [0.0, 1.0, 0.0, dt], [0.0, 0.0, 1.0, 0.0], [0.0, 0.0, 0.0,
    1.0]])
X = np.array([[Pos_Real_X[0]], [Pos_Real_Y[0]], [0.0], [0.0]])  # condiciones iniciales
B = np.array([[dt ** 2 / 2.0, 0.0], [0.0, dt ** 2 / 2.0], [dt, 0.0], [0.0, dt]])
U = np.array([[ax], [ay]])  # vector de entrada ax y ay(aceleraciones en x e y)
W = 0  # ruido en la medicion
V = 0 #ruido en el proceso
#P = 0.1   # covarianza
```

```
#Q = np.diag((0.0071, 0.0081, 0.0003, 0.0326)) # covarianza de ruido en el proceso
Q = 0.0001
P = np.diag((0.1, 0.1, 0.1, 0.1))
#Q = np.dot(B, B.T)
Z = 0  # ruido en la medicion
#R = np.eye(4)
#R = np.diag((20, 20, 20, 20))
H = np.array([[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]])
I = np.identity(4)

aux = 0
m = (Pos_Real_Y[aux+1] - Pos_Real_Y[aux]) / (Pos_Real_X[aux+1] - Pos_Real_X[aux])  # calculo
    de la pendiente
ang = math.atan((m))  # calculo del angulo (respuesta en radianes)
# calculos luego de las mediciones
print((len(lonMed)/len(lonReal)))
print(len(lonReal))
for i in np.arange(1, len(lonMed)):
    Exactitud = datosGPS['accuracy'].ix[i]
    R = np.diag(((Exactitud)**2,(Exactitud)**2,(Exactitud)**2,(Exactitud)**2))
    if (i %((len(lonMed)/len(lonReal))+1)) == 0:
        aux = aux + 1
        if aux >= (len(lonReal)-1):
            aux = aux-1
        m = (Pos_Real_Y[aux+1] - Pos_Real_Y[aux]) / (Pos_Real_X[aux+1] - Pos_Real_X[aux])  #
            calculo de la pendiente
        ang = math.atan((m))  # calculo del angulo (respuesta en radianes)
    if Pos_Real_X[aux+1] > Pos_Real_X[aux] and Pos_Real_Y[aux+1] > Pos_Real_Y[aux]: #primer
        cuadrante
        vx = V_Real[i] * math.cos(abs(ang))  # calculo la velocidad en x
        vy = V_Real[i] * math.sin(abs(ang))  # calculo la velocidad en y
        ax = (V_Real[i] - V_Real[i - 1]) * math.cos(abs(ang)) / time  # calculo la aceleracion
            en x
        ay = (V_Real[i] - V_Real[i - 1]) * math.sin(abs(ang)) / time  # calculo la aceleracion
            en y
    elif Pos_Real_X[aux+1] < Pos_Real_X[aux] and Pos_Real_Y[aux+1] > Pos_Real_Y[aux]: #segundo
        cuadrante
        vx = V_Real[i] * math.cos(math.pi-abs(ang))  # calculo la velocidad en x
        vy = V_Real[i] * math.sin(math.pi-abs(ang))  # calculo la velocidad en y
        ax = (V_Real[i] - V_Real[i - 1]) * math.cos(math.pi-abs(ang)) / time  # calculo la
            aceleracion en x
        ay = (V_Real[i] - V_Real[i - 1]) * math.sin(math.pi-abs(ang)) / time  # calculo la
            aceleracion en y
```

```
    elif Pos_Real_X[aux+1] < Pos_Real_X[aux] and Pos_Real_Y[aux+1] < Pos_Real_Y[aux]:  #
        tercer cuadrante
        vx = V_Real[i] * math.cos(abs(ang)-math.pi)  # calculo la velocidad en x
        vy = V_Real[i] * math.sin(abs(ang)-math.pi)  # calculo la velocidad en y
        ax = (V_Real[i] - V_Real[i - 1]) * math.cos(abs(ang)-math.pi) / time  # calculo la
            aceleracion en x
        ay = (V_Real[i] - V_Real[i - 1]) * math.sin(abs(ang)-math.pi) / time  # calculo la
            aceleracion en y
    elif Pos_Real_X[aux+1] > Pos_Real_X[aux] and Pos_Real_Y[aux+1] < Pos_Real_Y[aux]:  #
        tercer cuadrante
        vx = V_Real[i] * math.cos((2*math.pi)-abs(ang))  # calculo la velocidad en x
        vy = V_Real[i] * math.sin((2*math.pi)-abs(ang))  # calculo la velocidad en y
        ax = (V_Real[i] - V_Real[i - 1]) * math.cos((2*math.pi)-abs(ang)) / time  # calculo la
             aceleracion en x
        ay = (V_Real[i] - V_Real[i - 1]) * math.sin((2*math.pi)-abs(ang)) / time  # calculo la
             aceleracion en y

    d_med = BEARING(latMed[i - 1], lonMed[i - 1], latMed[i], lonMed[i])
    v_med = d_med / time

    mx = grapmedx[i]    #posicion medida
    my = grapmedy[i]    #posicion medida
    Y = np.array([[mx], [my], [0], [0]])

    U = np.array([[ax], [ay]])

    # aplicando el filtro de kalman
    for j in np.arange(0, Niter):
        Grap_V_real.append(V_Real[i])
        V_Medida.append(v_med)

        px = px + vx * dt  # posicion x real
        py = py + vy * dt  # posicion y real
        X = np.array([[px], [py], [vx], [vy]])
        graprealx.append(X[0])  # vector para graficar posicion real x
        graprealy.append(X[1])  # vector para graficar posicion real y
        (X, P) = kf_predict(X, P, A, Q, B, U)
        (X, P) = kf_update(X, P, Y, H, R)
        px_est.append(X[0])
        py_est.append((X[1]))
        v_est.append((math.sqrt(X[2]**2+X[3]**2)))

GRAFICAS()
```