Facultad
de
Ciencias

Fusion de sensores con redes neuronales para aplicaciones de realidad virtual

(sensor fusión with neural networks for augmented reality aplications)

Trabajo de Fin de Grado
para acceder al

GRADO EN INGENIERIA INFORMATICA

Autor: Sergio Urquijo Alvarez

Director: Eugenio Villar Bonet

julio - 2021

Contenido

Resumen

En este trabajo se pretende explorar la posibilidad de usar redes neuronales para realizar la fusión sensorica de los datos crudos de una imu y así calcular la posición de un objeto en el espacio. Partiendo de un sistema de posicionamiento basado en cámaras y balizas luminosas en el entorno, ya desarrollado por otros alumnos [2][5] ,mi objetivo con este trabajo es explorar la posibilidad de usar redes neuronales junto con una unidad de medida inercial para mejorar la precisión de dicho sistema, reducir su gitter o aumentar su tasa de refresco.

Palabras clave: UMI, MPU, inercial, realidad virtual, realidad aumentada, aprendizaje autónomo, inteligencia artificial, posicionamiento, sensórica, fusión de sensores, kalman, red neuronal.

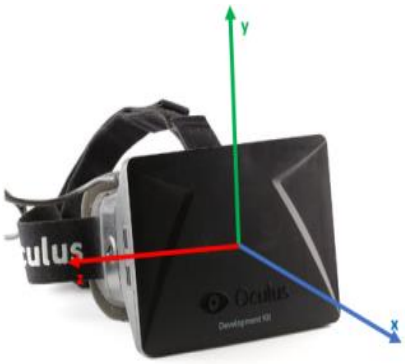
Abstract

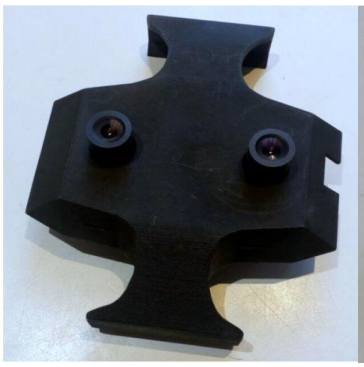
This paper will explore the possibility of using neural networks to perform the sensor fusion of an IMU`s raw data to calculate the position of an object in space. Starting from an already developed positioning system based on cameras and light beacons on the walls [2][5] i will be exploring the possibility of using neural networks together with an inertial measurement unit to improve precision of the system, reduce the gitter and or increase the refresh rate of the result.

Keywords: IMU, MPU, inercial measurement unit, motion processing unit, augmented reality, virtual reality, machine learning, artificial intelligence, AI, positioning, sensor fusión, kalman, neural network.

Introducción

Disponemos de un sistema de posicionamiento a través de cámaras y balizas en la pared ya desarrollado por alumnos de teisa [5] basado en una patente del grupo GIM[2].






Este sistema de posicionamiento está pensado para localizar unas gafas de realidad virtual[1] en un espacio interior y correr en hardware embebido de bajos recursos. Se desarrolló en un primer momento para la plataforma Odroid XU3 octa y ha sido portado[2] en este trabajo a una rasberryPI 4b. Su último desarrollo fue llevado a cabo por un alumno en su trabajo de fin de grado de informática [5]. En este trabajo se consiguió que el sistema funcionase en una prueba de concepto de forma correcta bajo una buena iluminación y haciendo movimientos suaves. Actualmente presenta las siguientes limitaciones:

1. El sistema produce datos de posición a una frecuencia de 30hz limitado por los fotogramas por segundo de las cámaras.
2. El sistema puede “perderse” con movimientos rápidos. Aquellos movimientos que en el tiempo entre fotogramas de la cámara causan que la posición nueva de una baliza en el FOV de la cámara ocupe la posición vieja de otra baliza diferente.
3. Debido a la baja resolución de las cámaras (720p), se produce un pequeño ruido en la posición calculada.

[1] Oculus rift SDK 1.0

[2] Solo Recopilación, instalar la librería openCV v3.0.2 (tiene su miga) y cambiar un tipo de dato interno a double64 para aprovechar la nueva arquitectura

4. Actualmente solo funciona con las gafas para las que fue desarrollado: Esto es porque el código internamente depende de las llamadas a la librería de las gafas para que estas usen su sensor interno y retornen la orientación.

Como se explicará en el apartado “Estado del arte” la solución a la mayoría de estos problemas es fusionar este sistema (baja precisión, baja frecuencia, sin drift) con una unidad de medición inercial MPU (Motion processing unit). De esta forma aportaríamos las siguientes soluciones a los problemas anteriores:

1. Entre cada dato y dato correcto del sistema de cámaras, interpolaríamos datos computados con la imu. Puesto que solo tenemos un tiempo de 1/30segundos, y sabemos que durante un periodo pequeño el resultado es fiable (Gómez, oct 2018, Página 58), podemos recoger datos de posicion a frecuencia mas alta con la imu.
2. Si tenemos datos de posición de la imu a frecuencia más alta el movimiento necesario para “perderse” será mucho más violento.
3. De nuevo, la precisión de los acelerómetros durante ese periodo de tiempo sería mejor o al menos con menor gitter.
4. Como veremos en el desarrollo de trabajo mas adelante, ya no dependeríamos de la imu interna de las oculus sino que podemos sacar la orientación con el sensor que incorporemos (mpu-9255 en este caso)

Explicación del hardware:

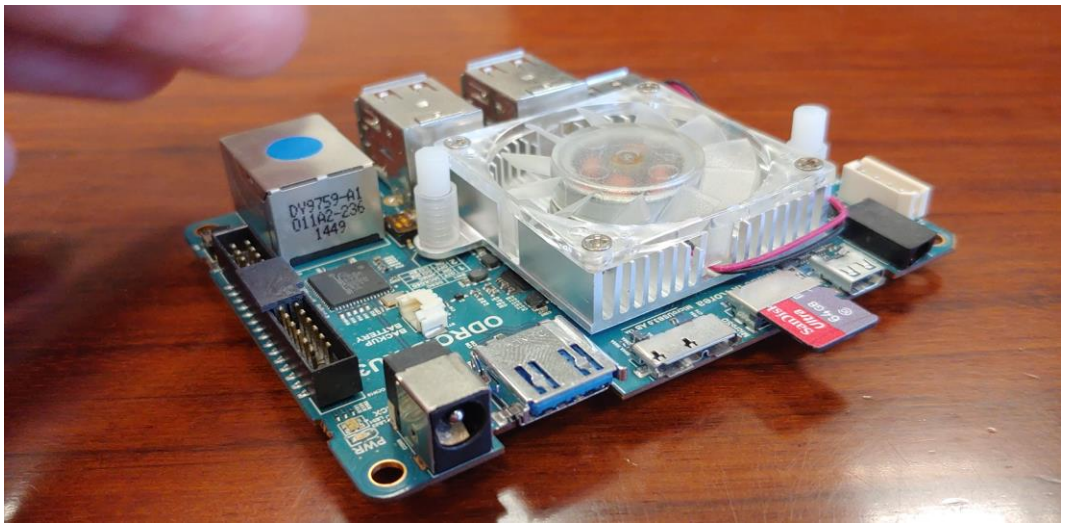
Rasberry pi:

Computador embebido con Linux debían de 64bits

Existen varias versiones, pero en este caso tenemos la versión 4b de 4gb:

- 4gb de ram LPDDR4-3200 SDRAM (existe de 8gb)
- Procesador de 4 nucleos con arquitectura arm64 cortex a-72 (arm v8) @1.5ghz
- GPU mali, etehrnet, usb, wifi 2.4ghz y 5ghz, bluetooth 5, hdmi, usb-c…

Odroid XU3:



- CPU: oct acore exynos 5422 (4nucleos cortex a7+4 cortex a15) (32bits)
- GPU: arm mali T628
- RAM: 2gb LPDDR3

MPU-9255 (Iven Sense, 2014)

Es un sensor de 9 dof (degrees of fredom).

1. Aceleración eje x: double en unidades adimensionales equivalentes a m/s2
2. Aceleración eje y: double en unidades adimensionales equivalentes a m/s2
3. Aceleración eje z: double en unidades adimensionales equivalentes a m/s2
4. Giroscopio eje x: double en unidades adimensionales equivalentes a radianes/s
5. Giroscopio eje y: double en unidades adimensionales equivalentes a radianes/s

6. Giroscopio eje z: double en unidades adimensionales equivalentes a radianes/s
7. Magnetómetro eje x: double en unidades adimensionales equivalentes a teslas
8. Magnetómetro eje y: double en unidades adimensionales equivalentes a teslas
9. Magnetómetro eje z: double en unidades adimensionales equivalentes a teslas

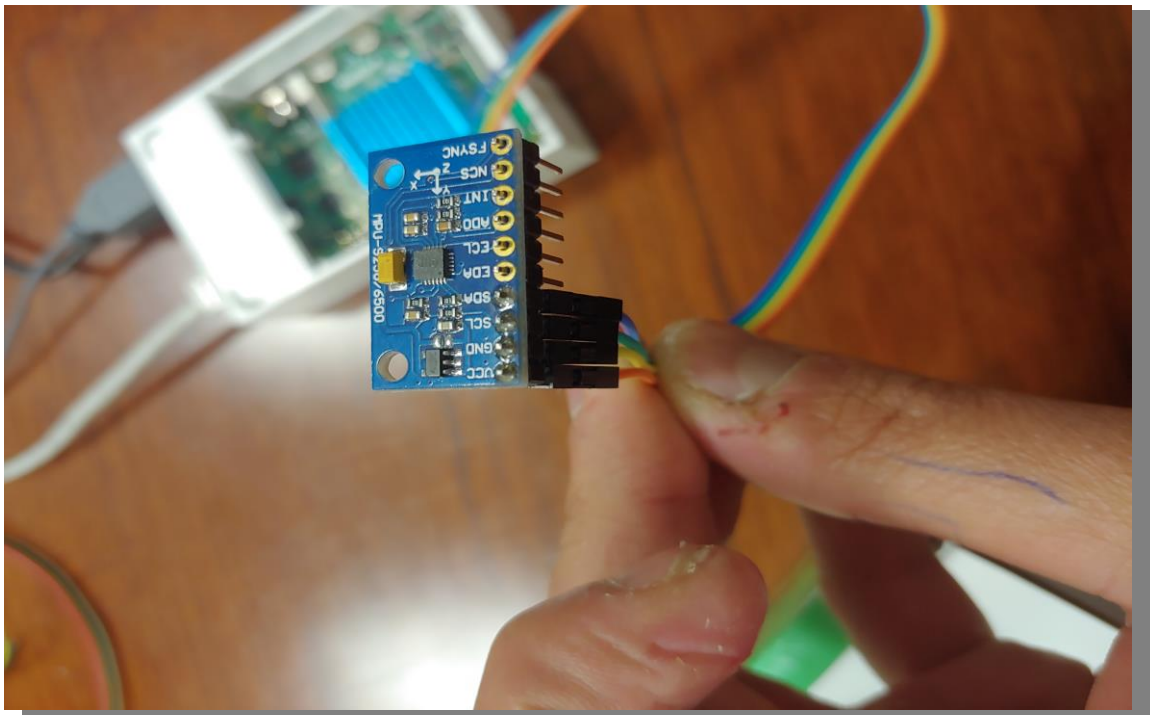
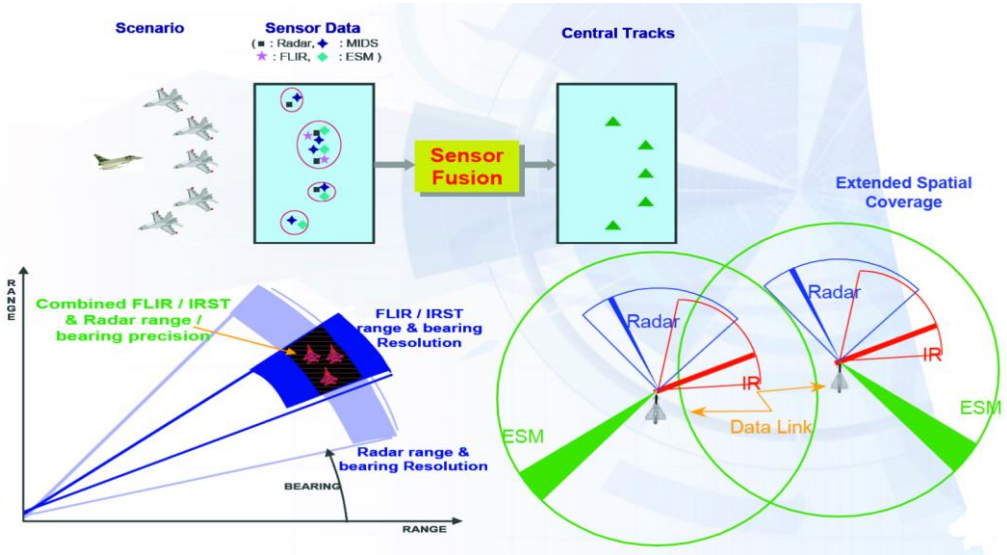


Ilustración 1: El empaquetado smd del centro es el sensor mpu9255. En la imagen se ve en una placa breakout junto con componentes pasivos como regulador de voltaje y capacitores necesarios para la señar i2c. Este sensor soporta bus i2c y spi. En la imagen esta conectado con la resberry pi a través de i2c y alimentado con la propia rasberry.

Fusión sensórica ¿Qué es?

La fusión de sensores es el proceso de combinar datos de diferentes fuentes o sensores para producir datos mejorados. Los logros de la fusión de sensores son robustez, cobertura espacial y temporal ampliada, mayor confianza, ambigüedad e incertidumbre reducidas, y resolución mejorada. (Elmenreich, oct 2002) [1]



Muchas veces se entiende también como el proceso completo que recoge los datos crudos de un sensor y los procesa para obtener la medición de la magnitud física que interese. Es decir, que muchas veces también se incluye dentro del proceso, el calibrado de sensores, arreglos de clipping, linealizacion...

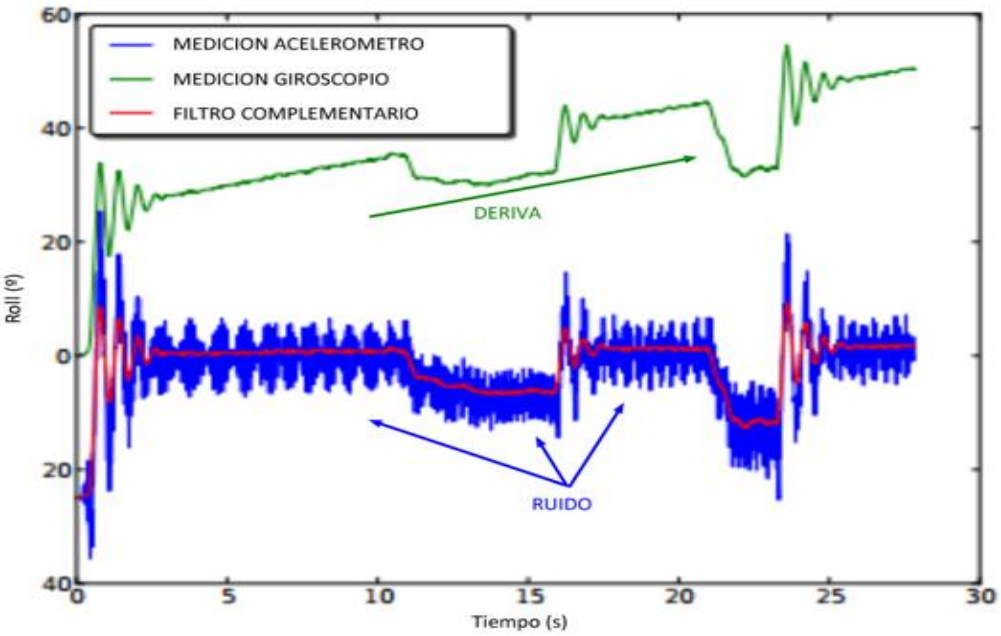


Ilustración 2: Giroscopio señal limpia pero con deriva, acelerómetro señal susceptible a vibraciones pero sin deriva.

En el ejemplo de la imagen, se ve como juntando con un filtro complementario las señales del giroscopio y la del acelerómetro conseguimos un resultado sin ruido y sin deriva.

Filtro complementario:

$$\theta = A \cdot (\theta_{prev} + \theta_{gyro}) + B \cdot \theta_{accel}$$

Donde teta sería, en el caso de la **Ilustración 2:** Giroscopio señal limpia pero con deriva, acelerómetro señal susceptible a vibraciones pero sin deriva.**,** la señal "roll" y A y B son valores de ponderación que suman 1 (se le suele dar menor valor ponderado a la señal del giróscopo, es decir A<B)

Motivación

Si quieres hacer que una maquina controle un sistema a través de unos actuadores lo primero es hacerla capaz de "sentir/medir" la magnitud física que trata de controlar (al menos para sistemas de control de ciclo cerrado como un PID
Posicionar objetos con la precisión que otorgaría hacerlo con sistemas inerciales tiene muchísimas implicaciones especialmente para el mundo de la robótica, automoción y los sistemas de control en general.

El sistema de posicionamiento por cámaras [5] del que disponemos utiliza muy pocos recursos de CPU. Según mis pruebas en la rasberry Pi 4, nunca se supera del 30% de tiempo de CPU. Estas pruebas se han realizado en el peor de los casos puesto que para el desarrollo del proyecto estoy utilizando un sistema operativo Debian completo con entorno gráfico en una rasberry Pi 4 de 4gb. Es decir, que parte de ese 30% de utilización de CPU ni siquiera se debe al proceso en cuestión. Además, hay que tener en cuenta que existen sistemas embebidos con rendimiento un orden de magnitud superior a la rasberry pi4 con sus 4 núcleos Cortex-A72. La Odroid con la que se desarrolló el sistema es mucho más potente (8 núcleos exynos de Samsung) a pesar de ser más vieja.
La motivación de este proyecto viene de la mano de que incluso si el sistema de posicionamiento por cámaras no es perfecto, tenemos la CPU ociosa la mayor parte del tiempo para tratar de explorar métodos alternativos de mejorar el rendimiento.
Uno de estos métodos es utilizar un sistema de posicionamiento puramente inercial para hacer los cálculos intercalados entre cada uno de los cálculos del sistema de cámaras [5]
En este proyecto se pretende hacer algo equivalente, pero en lugar de usar algoritmos convencionales, usar machine learning y redes neuronales con la esperanza de que no solo funcione, sino que quizá lo haga mejor.

Estado del arte

A día de hoy, el cálculo de la orientación en tres ejes de un cuerpo está muy logrado. Con un sensor genérico como el mpu6050 de invensense, configuraciones estándar (para un drone en concreto registro fs_sel=1, afs_sel = 1, simple rate = 350hz) [6] y el procesamiento por hardware interno que incorpora dicho sensor (dmp, el cual hace un

filtro kalman por hw) tenemos sensibilidades sobre 0.015º, precisiones de +-1º, y frecuencia de refresco real de 350hz. Para la gran mayoría de las aplicaciones, incluso aquellas que requieren de un control súper fino (precisión) y rápido (refresco) como puede ser la auto nivelación de un drone es más que suficiente y estamos hablando de un sensor muy genérico con configuraciones generales.

El problema lo tenemos con el cálculo de la posición en el espacio. Si el cálculo de la orientación podemos hacerlo con a través de métodos "convencionales", y disponemos de los valores de aceleración en cada uno de los ejes[3], la teoría nos dice que deberíamos ser capaces de simplemente usar las ecuaciones del movimiento, es decir, integrar la aceleración con respecto al tiempo para obtener la velocidad y de nuevo integrar esta velocidad con respecto al tiempo para obtener la variación de posición delta-p y con eso sumándolo a la posición inicial deberíamos ser capaces de computar la ubicación en el espacio de un objeto. Pues bueno, esto es cierto… hasta que no lo es. Como se ha comprobado en otros trabajos [3] este sistema funciona durante máximo de 2segundos. A partir de ese tiempo, el cálculo de la doble integral está integrando error y el resultado de la posición se desvía a infinito.

Para entender el problema del "drift/deriva" imaginemos que tenemos un ascensor que se mueve en un raíl totalmente aplomado, es decir, limitamos el movimiento a una sola dimensión, una recta. Este ascensor tiene un acelerómetro que nos dará valores de aceleración positiva o negativa (dependiendo de la dirección del movimiento). Obviado la aceleración de la gravedad si el sensor no está calibrado, dejamos el ascensor en reposo (posición = 0, velocidad = 0, aceleración = 0) y hacemos 10 lecturas del sensor obtenemos los siguientes resultados:

13.45678	13.45680
13.45676	13.45682
13.45680	13.45680
13.45677	13.45679
13.45674	13.45684

Como vemos, los resultados son un número diferente de cero. Además, no es del todo constante, sino que tiene ruido. La media de este valor es 13.45679.

[3] Cuidado, los valores de aceleración que proporciona el sensor están expresados en un sistema de referencia arbitrario pero fijo al sensor. Para pasar esos valores a un sistema de referencia relativo a un csp fijo a la habitación se requiere de un cambio de bases.

Es decir, el sensor no está calibrado y a pesar de que debería estar mostrando cero m/s2, nos muestra 13.45679m/s2. Ahora para calibrarlo lo único que tiene que hacer nuestro software es saber que cualquier valor que se lea de dicho sensor hay que restarle 13.45679 para obtener el valor real de aceleración. Una vez hacemos este ajuste y repetimos el experimento vemos los siguientes resultados:

-0.00001 0.00005
0.00002 -0.00003
-0.00001 0.00005
0.00002 -0.00003
0.00001 -0.00001

Vemos como aun restándole la media de 10 lecturas anteriores, la media cuando el sensor está en reposo sigue siendo diferente de cero. Bueno esto es porque el sensor no es un sensor ideal sino uno real. Uno de los motivos es que la temperatura afecta al sensor, pero incluso si no lo hiciese, en las primeras 10 lecturas que hicimos para calibrar conseguimos la media de 10 valores, pero si tras esos 10 valores seguimos recolectando otros 90 es prácticamente imposible que la media de los primeros 10 valores coincida con la media de los 100 totales. En resumen, es imposible en la práctica calibrar un sensor a la perfección y siempre nos va a dar un valor muy cercano al real, pero con un pequeño offset.

Ahora si tratamos de calcular la posición del ascensor (sigue en reposo) con este sensor calibrado, al de un tiempo ese pequeñísimo error en la aceleración empieza a computar un valor de velocidad diferente de cero. Al de 200 ciclos tenemos que la velocidad es 0.002m/s y la posición por tanto ya no es cero. Si esperamos 2segundo o más la posición tiende a derivarse a infinito o cerca.

Aunque sea imposible conseguir un sistema de posicionamiento puramente inercial sin deriva, podemos tratar de reducir el error y mantenerlo bajo unos máximos durante el mayor tiempo posible.

Una solución lógica es usar mejores sensores:

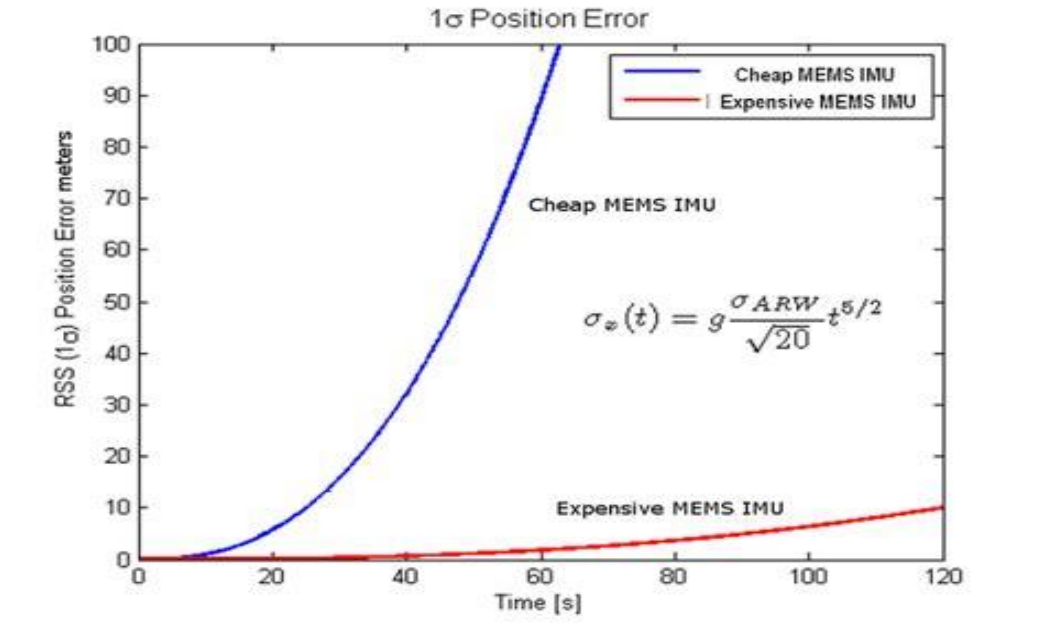


Ilustración 3: Comparativa drift en mems caro rojo y barato en azul. Imagen tomada de la página 36 de la tesis de (DEBITETTO, 2011)[8]

Otra solución es el uso de un conglomerado de varios imu´s para asi mejorar la precisión y hacer las veces de uno de mayor calidad:

"[…] In situations of pure inertial navigation (no external aiding), multiple IMUs (MIMU) can be used to improve the performance of a single unit" (ARIEL LAREY, ELIEL AKNIN, & ITZIK KLEIN, 2016) [7]

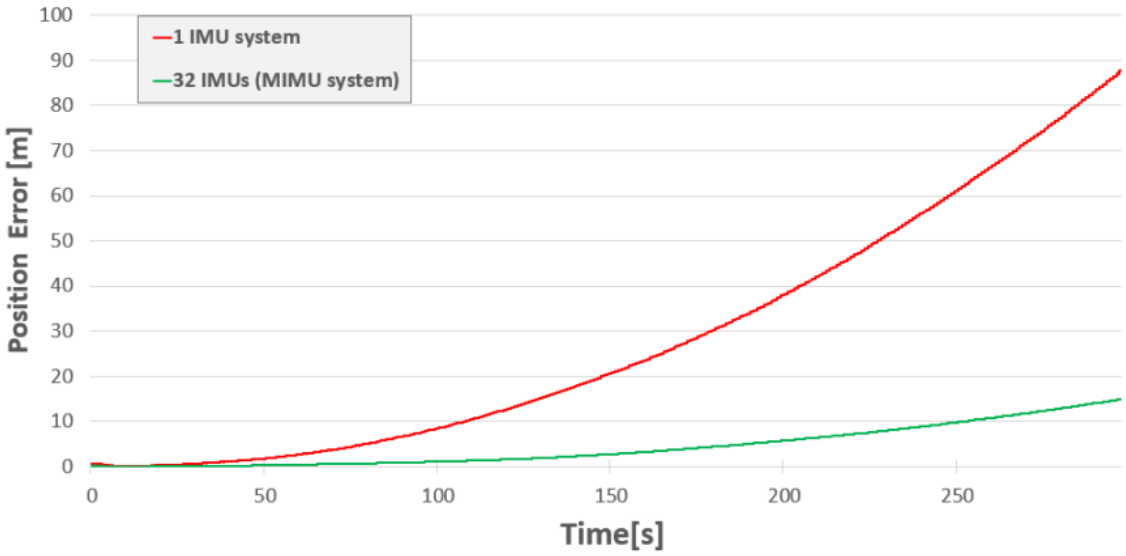


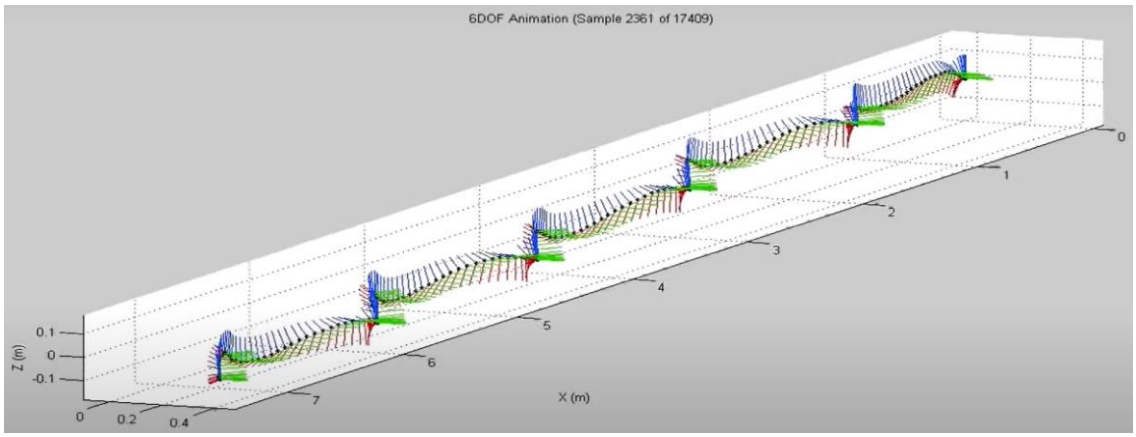
Ilustración 4: (ARIEL LAREY, ELIEL AKNIN, & ITZIK KLEIN, 2016)[7] (Multiple Inertial Measurement Units—An Empirical Study, página 5**)**

En el caso de este trabajo, trataremos de ver si cabe la posibilidad de una tercera solución que consista en el uso de redes neuronales para interpretar los datos crudos del sensor.

La solución a este problema es la fusión sensórica. Podríamos usar tanto sensores de acelerómetros (con alta frecuencia y sensibilidad pero con drift) junto con datos de GPS (baja frecuencia, baja precisión pero sin el problema de la deriva) para juntarlos y obtener un resultado preciso sin drift.. Esto no siempre es posible; por ejemplo, en interiores.
Para el caso concreto de unas gafas de realidad virtual, a día de hoy, las soluciones que se están aportando en la industria son el uso de acelerómetros junto con sistemas que vienen a sustituir el GPS en interiores. Algunos ejemplos de estos sistemas son:

- **Gafas HTC Vive (Facebook):** Uso de visión por computador para reconocer el entorno. Diversas cámaras en las gafas
- **Playstation VR 2.0 (Sony):** Un sistema bastante viejo llamado Kinect que se compone de cámaras colocadas en el televisor que enfocan a las gafas
- **Rift v3(Oculus):** Sistema de balizas colocadas en las esquinas de la habitación que triangulan a través de ondas de radio.

Como vemos, a día de hoy no existe un sistema de medición de la posición puramente inercial para localizar un objeto a largo plazo.
Lo más cercano a esto es un sistema desarrollado por (Madgwick, 2010) [4] en su Ph.D. para la universidad de bristol y despues comercializado como MGIMU.



En este sistema se usa un “truco” muy interesante para trakear la posicion de los pies de un athleta mientras corre. Como Madgwick describe en su informe, puesto que la doble integración comienza a producir error en menos tiempo que el periodo entre pisadas, lo que el sistema hace es intuir correctamente que siempre y cuando los acelerómetros detectan que el pié esta en contacto con el suelo[4], se puede asumir que las aceleraciones y velocidades son cero y por tanto inicializar las variables en el código a cero cada vez que el pie se posa.

¿Por qué con redes neuronales?

Simplicidad

En primer lugar, nos ahorraríamos la complejidad de implementar un filtro kalman. Además, el uso de una red neuronal supondría que nos ahorraríamos el proceso de calibración del sensor y linealización de la señal. Estos dos cálculos estarían implícitos en el entrenamiento de la red.

Rendimiento y efectividad

Argumento en contra: Explicación del algoritmo tradicional: Primero para cada uno de los 9 valores numéricos que nos proporciona el sensor hay que hacer una suma de un número real para hacer el calibrado y después una multiplicación por una constante para

[4] Para detectar cuando el pie está en contacto con el suelo se mira que las aceleraciones son constantes, pero no necesariamente cero.

que dichos valores representen unas unidades a poder ser en el sistema internacional (Aceleración en m/s2, giroscopio en rad/s, y magnetómetro en $kg\ s^{-2}\ A^{-1}$ = Teslas). Después se tiene que hacer un cómputo [5]para que el vector aceleración este expresado con el sistema de referencia fijo a la habitación y, por último, la doble integración de la aceleración con respecto al tiempo para conseguir un delta de posición y sumárselo a la posición inicial.

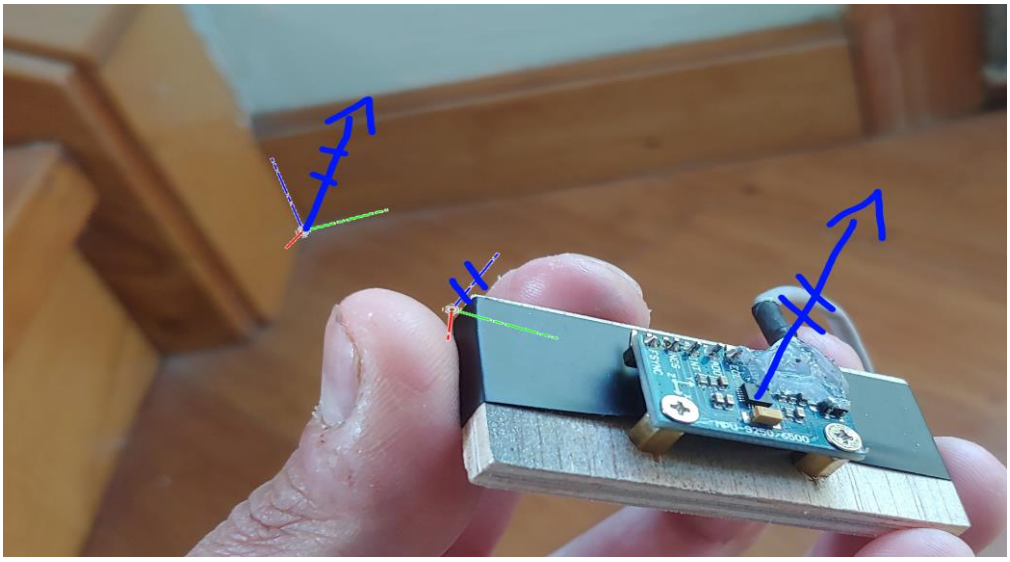


Ilustración 5: En azul, una aceleración en el eje z según el sistema de referencia local al sensor (lo que el dato crudo significa) no equivale a una aceleración puramente en el eje z de la habitación puesto que está inclinado.

Como vemos, el algoritmo que llamaremos tradicional es un cálculo analítico "perfecto". El motivo de que tengamos el problema de la deriva no es por el algoritmo sino por la naturaleza imperfecta del sensor.

Entonces si una red neuronal, en su proceso de entrenamiento lo que hace no es más que ajustar sus pesos y biases para que hagan el cálculo equivalente ¿Por qué iba a funcionar mejor la red que el algoritmo tradicional teniendo en cuenta que la red no va a dar resultados exactos? ¿Solo podemos ir a peor?

Imaginemos que queremos hacer una puerta lógica xor, una suma, una raíz o una multiplicación. Una red neuronal se puede entrenar para que lo haga, pero siempre va a tener una rendimiento igual o peor a cualquier algoritmo exacto.

[5] Este cómputo requiere de conocer la orientación que a su vez se consigue con magnetómetro, acelerómetro y giróscopo, pero no entro a explicarlo.

Además ¿Es posible que una red haga ese cálculo concreto? Sabemos que con esos datos tiene que ser posible hacer el cálculo, pero puede que una red neuronal, al no ser un modelo de cálculo Turing completo [10] [11] no sea capaz de hacer las operaciones necesarias para obtener el resultado. Independientemente de si es turing completo o no, una red neuronal es un circuito combinatorio compuesto de sumatorios de productos, una integral no es mas que un sumatorio y cualquier otra funcion matemática por complicada que sea (un logaritmo seno…) se puede calcular como una serie infinita de sumandos (taylor series). Todo indica que debería ser posible.

Argumento a favor: **¿Podría rendir mejor que hacer el cálculo tradicional?** Esto es especulación puesto que, primero no hemos visto todavía los resultados y segundo que incluso si los resultados son mejores que el método tradicional, la red es una caja negra en la que no vemos realmente lo que está haciendo por dentro.
Una red neuronal al final es una máquina de buscar patrones. En la sección "Objetivos; Arquitectura propuesta" se puede ver como los valores de entrada de la red no son solo el dato en el instante i-1 para calcular la posición en i sino que se pretende introducir como entrada los n valores anteriores. Si tan solo le diésemos un dato, teniendo en cuenta que el sensor es bastante ruidoso sería prácticamente imposible que funcione. En cambio, aportando a la red el barrido de n datos anteriores la idea es que esta sea capaz de encontrar patrones en el ruido, así como dependencias entre el error de cada dato crudo con el resto de datos.
Cabe la posibilidad incluso, de que este sistema se use para tomar datos de tres sensores diferentes (como por ejemplo para las dos manos y la cabeza) y generar las posiciones de tres dispositivos.

Puede que la red sea capaz de encontrar patrones de comportamiento que relacionan los movimientos de la cabeza con los de las manos. Y que cuando con una mano se hace un

movimiento brusco para dar un espadazo a un enemigo, la cabeza sufra un movimiento brusco como reacción de la inercia del brazo. Puede que aprenda que cuando la posición de los pies llega a un límite (por ejemplo, la pared de la habitación) lo más probable es que la cabeza cambie de dirección y no se dé contra la pared. Puede que aprenda los patrones de movimiento del cuerpo humano y que reconozca la disposición inminente de un salto de la persona. Y por esto es que incluso si los sensores tienen error, puede que la red intuya la trayectoria de los movimientos y tenga una mejor precisión que el algoritmo convencional. Es por esto que en este trabajo se pretende explorar si al menos una red es capaz de funcionar, aunque sea mal, pero como prueba de concepto de que podría haber trabajo por explorar.

Uso de recursos

Otra ventaja de usar redes neuronales en lugar del método analítico de la doble integral es que no utilizaría tantos recursos de CPU. La red neuronal internamente tendría que hacer unos cálculos equivalentes al algoritmo analítico, pero con una gran diferencia. En el caso del método analítico tenemos que ejecutarlo en un proceso secuencial en CPU mientras que para hacer inferencia con la red neuronal podemos hacer paralización masiva con la GPU del sistema.

Objetivos

El objetivo de este trabajo es explorar la posibilidad de utilizar un algoritmo de aprendizaje automático como son las redes neuronales (convolucionales o no) para hacer el cálculo de fusión sensórica de una imu.

Arquitectura propuesta:

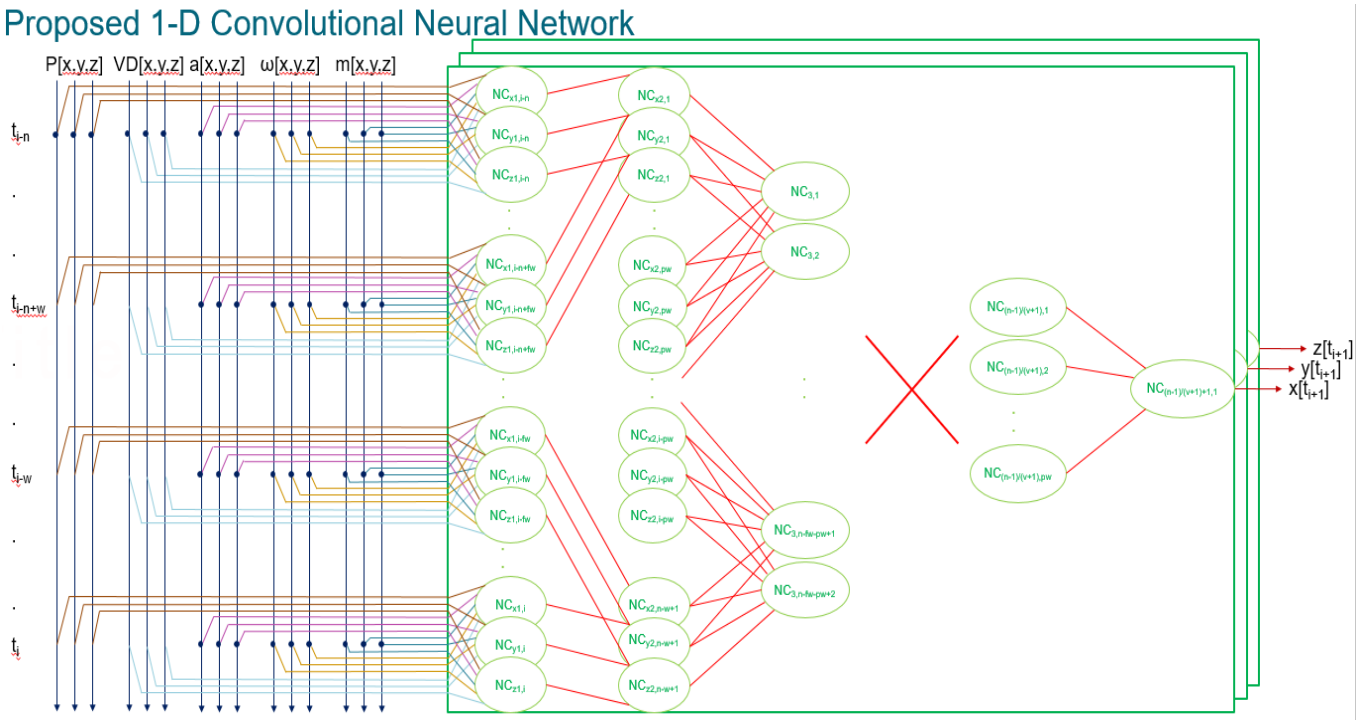


Ilustración 6: Arquitectura propuesta al comienzo del proyecto.

En donde P[x,y,z] es el vector posición expresado con sus tres componentes, VD[x,y,z] es la orientación, a[], w[] y m[] son los datos crudos del sensor; aceleraciones, giroscopio y magnetómetro.

Desarrollo del trabajo

Lo primero ha sido portar e instalar el sistema de posicionamiento, que usaremos para etiquetar los datos, basado en cámaras y balizas a una rasberry pi 4b. La motivación para este cambio de hardware es que la odroid, pese a tener una potencia de cpu equivalente o superior, es bastante vieja y no disponía del soporte necesario para instalar posteriormente las librerías de tensorflow y keras necesarias asi como las librerías de oculus.

La portabilidad del sistema ha consistido en recompilar el programa en la rasberry, instalar opencv 3.0.2 y reinstalar las librerías de las gafas oculus. Además, para el conexionado del sensor imu a la rasberry, se ha tenido que activar el bus i2c en los puertos GPIO y configurarlo a una velocidad menor de 400mhz puesto que los cables no son trenzados y tienen una longitud de aproximadamente 1m y producían errores probablemente fruto de la capacitancia entre pares de cables.

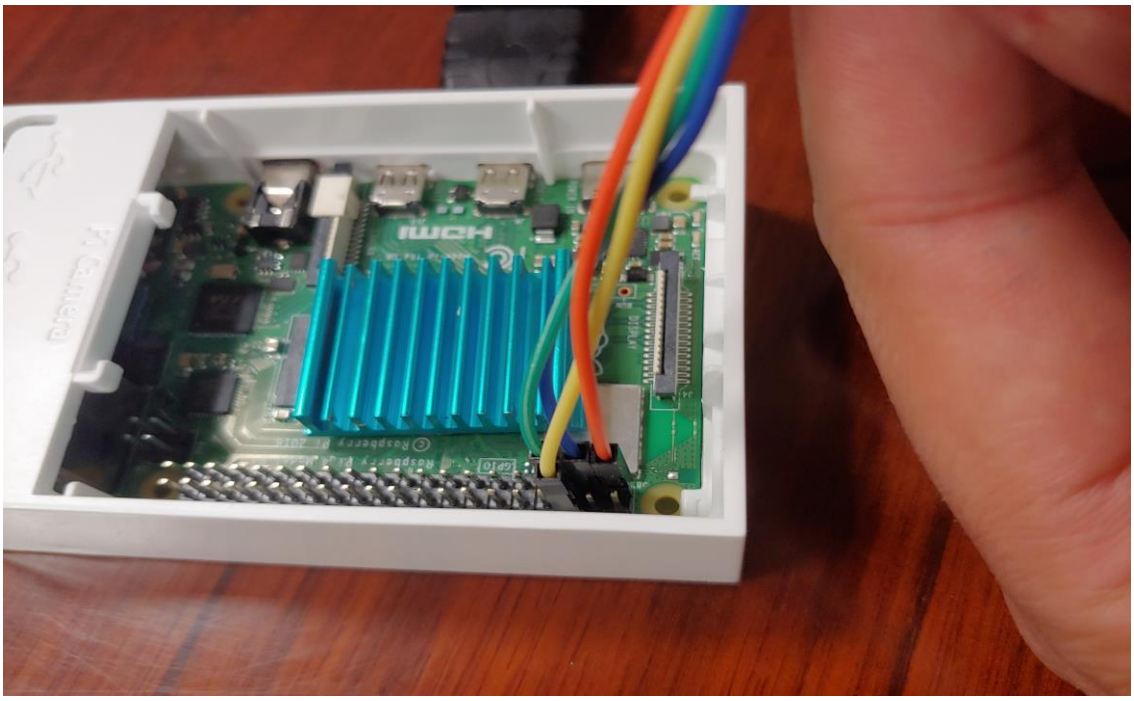
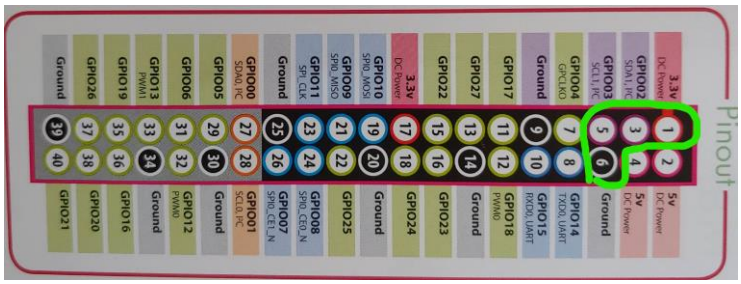
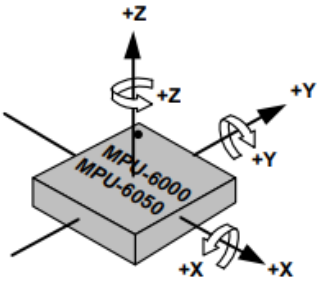


Ilustración 7: Conexión a la Rasberry con jumpers.





Recolección de los datos

Datos del sistema de cámaras

El sensor imu se ha configurado para tener un sampleo de 30hz y así coincidir con la velocidad a la que se entregan los datos de posición(etiquetas).
El sistema de posicionamiento se ha modificado para que en cada dato de posición adjunte también el epoch/timestamp en microsegundos. Este tiempo absoluto corresponde con el momento en que se tomó la fotografía de las cámaras a partir de la cual se hicieron los cálculos que dieron lugar a dicho dato de posición.
Igualmente, para el código que entrega los datos crudos del sensor se aporta también el timestamp absoluto en microsegundos.

Fichero de salida del logger_datos.cpp:

```
1624885325296693,-0.00512695,1.01099,0.0561523,0.0240391,0.12609,-0.0768032,-13.708,-8.82821,-40.2367
1624885325328951,-0.0614014,1.03113,0.0184326,0.161431,0.109335,-0.188783,-14.153,-8.65069,-40.264
1624885325361209,-0.0402832,1.01233,0.0109863,0.307201,-0.0358762,-0.333436,-14.3641,-8.65305,-40.5643
1624885325393467,0.0263672,1.02698,0.0162354,0.335127,-0.0590542,-0.276748,-14.0985,-8.94369,-40.9438
1624885325425725,-0.0319824,1.06738,-0.0384521,0.272574,0.0967688,-0.142986,-14.3205,-9.03183,-41.1082
1624885325457983,-0.0808105,1.09912,-0.11438,0.0955278,0.1152,-0.0173223,-14.7878,-9.10234,-41.3789
1624885325490241,-0.0206299,1.03967,-0.0388184,-0.0270642,-0.0928437,-0.0485986,-14.7271,-9.15875,-41.734
1624885325522499,0.00170898,1.05005,-0.0310059,0.00811905,-0.108168,-0.0845949,-14.8234,-9.637,-41.3232
1624885325554757,-0.06604,0.987671,-0.0605469,-0.0496863,-0.0458944,-0.0290237,-14.9005,-9.73085,-41.4117
1624885325587015,-0.0307617,1.01685,-0.0369873,-0.134021,-0.0833142,0.0742998,-14.8172,-9.9503,-41.4825
1624885325619273,-0.0388184,0.986206,-0.0078125,-0.15966,-0.0855207,0.0508262,-14.7507,-9.83712,-41.6783
1624885325651531,-0.0418701,1.00903,0.0045166,-0.216837,-0.0572977,0.0996633,-15.1319,-9.89094,-41.835
1624885325683789,0.0136719,0.990356,0.0662842,-0.246438,-0.0237873,0.0792778,-15.0024,-10.0784,-41.8211
1624885325716047,-0.0523682,1.00305,0.0246582,-0.193938,0.112488,0.041858,-15.1885,-9.93958,-41.5316
1624885325748305,-0.0665283,0.982056,0.0366211,-0.144464,0.094027,0.0100199,-15.0477,-9.97291,-41.3
1624885325780563,-0.0570068,0.98291,0.019165,-0.0444773,0.0577055,-0.00170813,-15.3695,-9.99958,-41.1147
1624885325812821,-0.0704346,1.00842,0.0140381,-0.0252007,0.0208374,0.00499232,-15.3374,-10.0209,-41.2449
```

Ilustración 8: timestamp, accx,accty,accz,gyrx,gyry,gyrz,magx,magy,magz

Fichero de salida del sistema de posicionamiento:

```
1624885245337839,58,8,386,-3.45585,-8.32826,1.4832
1624885245369839,58,33,395,-3.38261,-7.71876,1.11186
1624885245401839,43,29,391,-2.98877,-6.90294,0.602683
1624885245437839,17,25,381,-2.77192,-6.09394,-0.115256
1624885245469839,-13,20,374,-1.85549,-5.82433,-1.42014
1624885245501839,-40,51,380,0.0383621,-6.12741,-2.65626
1624885245537839,-36,79,363,0.0746794,-6.55862,-2.50374
1624885245569839,-26,81,384,-0.368303,-7.51087,-2.37085
```

Ilustración 9: timestamp, posicionX, posicionY, posicionZ

Ahora, estas dos fuentes de datos se tienen que correlacionar con ayuda del timestamp. El problema es que incluso aunque ambos sistemas están configurados para sacar datos a una velocidad de aproximadamente 30hz los datos poco a poco se van desfasando porque no están controlados por el mismo reloj y además el sistema de cámaras graba en estándar pal a 29.85fps(hz). La solución que he encontrado es recorrer los datos e ir eliminando una fila de los datos de la imu cuando el desfase es mayor a 1/30s
Para ejecutar los dos programas a la vez se usa un script que llama a los dos comandos (“proyecto_posicionamiento_raul\code\test_arm\main.cxx” y “logger_datos.cpp”)

Este es el resultado de los datos ya etiquetados:

aceleracion			gyroscopio			magnetometro			angulos euler			posicion		
0.050415	0.967651	0.353516	-0.0000731	0.0000296	0.00011432	-13.6203	-9.17549	-26.0336	-12.6711	0.0298529	0.319809	-111	-240	194
0.0511475	0.968994	0.355347	0.00160195	0.00030877	-0.00044404	-13.6482	-9.07289	-26.2564	-12.6424	0.0267558	0.315753	-111	-240	194
0.0510254	0.967041	0.353027	0.00048479	0.00030867	-0.00044389	-13.6706	-8.99081	-26.4346	-12.6554	0.0219293	0.307432	-111	-240	194
0.0491943	0.968018	0.352661	0.00020547	0.00030857	-0.00044375	-13.6885	-8.78078	-26.2988	-12.6544	0.0110121	0.287884	-132	-230	268
0.0507812	0.969116	0.354004	0.00048456	-0.00052902	0.00039388	-13.7029	-8.9015	-26.0509	-12.6578	0.0114363	0.289459	-111	-240	194
0.0507812	0.970215	0.351685	-0.00063225	0.00030864	-0.00016457	-13.8592	-8.8537	-25.9918	-12.6659	0.0145842	0.294374	-111	-240	194
0.0506592	0.966797	0.351685	0.00020544	0.0000294	-0.00016452	-13.6945	-8.67108	-26.0838	-12.6719	0.013247	0.292531	-111	-240	195
0.0482178	0.969482	0.355103	-0.00063212	0.0000294	-0.00016447	-13.8525	-8.66937	-25.8789	-12.699	0.0100804	0.287438	-116	-245	225
0.0511475	0.968506	0.352295	0.0007639	-0.00052897	0.00067308	-13.9789	-8.66799	-25.9934	-12.6979	0.0201559	0.305248	-110	-252	204
0.0500488	0.969116	0.352417	-0.00091133	0.0000295	-0.00016463	-13.7903	-8.95564	-26.0851	-12.705	0.0159033	0.297352	-111	-241	195
0.0491943	0.969971	0.351318	-7.35E-05	-0.00024964	-0.00044374	-13.7843	-8.89702	-26.1584	-12.7049	0.0172072	0.300248	-111	-241	195
0.0505371	0.968384	0.350586	-0.00063185	-0.00052873	0.00011473	-13.7794	-8.99449	-26.0778	-12.7143	0.014648	0.294584	-110	-252	204
0.0500488	0.970581	0.352173	-7.33E-05	0.00030893	-0.00016447	-13.9204	-8.92809	-26.1525	-12.7135	0.0215187	0.306316	-111	-241	195
0.0522461	0.967896	0.35437	-7.33E-05	-0.00052866	-0.00072274	-13.7435	-8.58622	-26.2123	-12.7052	0.0280122	0.316913	-110	-252	204
0.0509033	0.968018	0.348755	-7.33E-05	0.00058817	-0.00044335	-13.8917	-8.74585	-25.9817	-12.7052	0.0280122	0.316913	-110	-252	204
0.0509033	0.967651	0.352905	0.00048508	-0.001087	0.00039428	-13.8654	-8.72918	-26.0757	-12.7367	0.0160916	0.297815	-111	-242	195
0.0526123	0.970093	0.351562	-0.00035257	0.00030917	-0.00016417	-13.8443	-8.71585	-25.8724	-12.7263	0.00128956	0.272157	-109	-252	204
0.0500488	0.968262	0.35376	-7.33E-05	-0.00024926	-0.00016412	-13.8275	-8.84955	-25.9883	-12.7155	0.00532896	0.279469	-111	-241	195
0.0506592	0.967407	0.352905	0.00076421	3.00E-05	-0.00016406	-13.6692	-8.81214	-25.8025	-12.7122	0.00858259	0.285882	-128	-220	260
0.050293	0.967285	0.35376	-7.35E-05	0.0005883	-0.00044317	-13.6874	-9.07096	-26.0715	-12.7024	0.00704784	0.28422	-129	-222	253

Ilustración 10: vista de Excel pero el fichero sigue siendo .csv

Datos de Gacebo:

Gacebo es un simulador de drones y aeronaves no tripuladas de código abierto. Este simulador recibe las entradas pwm que recibiría la placa controladora del drone desde el receptor lo cual es equivalente a la posición de los joy-sticks del mando radiocontrol y simula (con otros parámetros) la física del drone y los movimientos, consumos de energía y trayectoria.

Esta aplicación por dentro simula una imu y se ha aprovechado esto para sacar los siguientes datos del programa mientras pilotábamos un drone con un mando conectado al ordenador:

```
### ACC(-3.336212 -0.049690 -9.222883) GYR(-0.001704 0.000033 -0.000309) QUAT(0.974022 -0.023277 -0.171104 -0.146504) : VEL(4.688697 -1.367509 -0.857035) POS(90.963378 -26.604040 -27.617779)
### ACC(-3.336212 -0.049692 -9.222927) GYR(-0.001639 0.000030 -0.000308) QUAT(0.974022 -0.023278 -0.171104 -0.146504) : VEL(4.688696 -1.367512 -0.857032) POS(90.968067 -26.605408 -27.618636)
### ACC(-3.336212 -0.049694 -9.222976) GYR(-0.001574 0.000028 -0.000308) QUAT(0.974022 -0.023279 -0.171104 -0.146505) : VEL(4.688695 -1.367514 -0.857030) POS(90.972756 -26.606775 -27.619493)
### ACC(-3.336211 -0.049696 -9.223028) GYR(-0.001509 0.000025 -0.000307) QUAT(0.974021 -0.023280 -0.171104 -0.146505) : VEL(4.688694 -1.367517 -0.857028) POS(90.977444 -26.608143 -27.620350)
### ACC(-3.336211 -0.049698 -9.223084) GYR(-0.001443 0.000022 -0.000307) QUAT(0.974021 -0.023280 -0.171104 -0.146505) : VEL(4.688693 -1.367520 -0.857026) POS(90.982133 -26.609510 -27.621207)
### ACC(-3.336211 -0.049699 -9.223144) GYR(-0.001377 0.000019 -0.000306) QUAT(0.974021 -0.023281 -0.171104 -0.146505) : VEL(4.688692 -1.367522 -0.857025) POS(90.986822 -26.610878 -27.622064)
### ACC(-3.336211 -0.049701 -9.223205) GYR(-0.001312 0.000016 -0.000306) QUAT(0.974021 -0.023282 -0.171104 -0.146506) : VEL(4.688691 -1.367525 -0.857023) POS(90.991510 -26.612245 -27.622921)
### ACC(-3.336211 -0.049702 -9.223269) GYR(-0.001246 0.000013 -0.000306) QUAT(0.974021 -0.023282 -0.171103 -0.146506) : VEL(4.688690 -1.367528 -0.857021) POS(90.996199 -26.613613 -27.623778)
### ACC(-3.336211 -0.049703 -9.223336) GYR(-0.001180 0.000010 -0.000305) QUAT(0.974021 -0.023283 -0.171103 -0.146506) : VEL(4.688689 -1.367530 -0.857019) POS(91.000888 -26.614980 -27.624635)
### ACC(-3.336211 -0.049704 -9.223403) GYR(-0.001115 0.000008 -0.000305) QUAT(0.974021 -0.023283 -0.171103 -0.146506) : VEL(4.688689 -1.367533 -0.857017) POS(91.005576 -26.616348 -27.625492)
### ACC(-3.336211 -0.049705 -9.223473) GYR(-0.001050 0.000005 -0.000304) QUAT(0.974021 -0.023284 -0.171103 -0.146507) : VEL(4.688688 -1.367536 -0.857016) POS(91.010265 -26.617716 -27.626349)
### ACC(-3.336211 -0.049706 -9.223543) GYR(-0.000985 0.000002 -0.000304) QUAT(0.974021 -0.023284 -0.171103 -0.146507) : VEL(4.688687 -1.367538 -0.857014) POS(91.014954 -26.619083 -27.627206)
```

Ilustración 11: Aceleracion, giroscopio, orientacion (quaternion), velocidad y posicion del drone.

Como vemos en estos datos nos faltaría el magnetómetro y la orientación la tenemos en forma de cuaternion en lugar de angulos euler.

Para procesar estos datos y convertirlos en formato .csv he usado este código que usa expresiones regulares:

```
import re

filepath = "C:/Users/Sergio/Desktop/beca investigacion/datos/datos gacebo/datosIMU2.txt"
infile = open(filepath, 'r')
lines = infile.readlines()

with open('C:/Users/Sergio/Desktop/beca investigacion/datos/datos gacebo/accgyrvelquatpos.csv', 'w') as f:

    for line in lines:
        match = re.search('### ACC\((-?\d+.\d+) (-?\d+.\d+) (-?\d+.\d+)\) GYR\((-?\d+.\d+) (-?\d+.\d+) (-?\d+.\d+)\)
                          QUAT\((-?\d+.\d+) (-?\d+.\d+) (-?\d+.\d+) (-?\d+.\d+)\) : VEL\((-?\d+.\d+) (-?\d+.\d+) (-?\d+.\d+)\)
                          POS\((-?\d+.\d+) (-?\d+.\d+) (-?\d+.\d+)\)', line)
        if match is not None:
            f.write(match.group(1)+','+match.group(2)+','+match.group(3)+','+match.group(4)+','+match.group(5)+','
                    +match.group(6)+','+match.group(11)+','+match.group(12)+','+match.group(13)+','+match.group(7)+','+
                    match.group(8)+','+match.group(9)+','+match.group(10)+','+match.group(14)+','+match.group(15)+','+match.group(16)+'\n')
```

El resultado de los datos procesados es el siguiente:

aceleracion			gyroscopio			velocidad			cuaternion				posicion		
-3.450243	0.164644	-9.184004	0.000713	-0.005615	0.006777	0.224292	-4.830844	0.119285	-0.726529	0.121503	0.133393	0.663023	-87.019117	-4.971029	-27.123238
-3.450278	0.164542	-9.18396	0.000783	-0.005628	0.006762	0.224462	-4.830863	0.119284	-0.72653	0.121505	0.133395	0.66302	-87.018892	-4.975859	-27.123119
-3.450313	0.16444	-9.183916	0.000854	-0.005641	0.006747	0.224633	-4.830882	0.119284	-0.726532	0.121507	0.133397	0.663017	-87.018668	-4.98069	-27.122999
-3.450348	0.164339	-9.183872	0.000926	-0.005654	0.006731	0.224803	-4.8309	0.119283	-0.726534	0.121509	0.133399	0.663015	-87.018443	-4.985521	-27.12288
-3.450383	0.164237	-9.183826	0.000998	-0.005666	0.006716	0.224972	-4.830919	0.119283	-0.726536	0.121511	0.133401	0.663012	-87.018218	-4.990352	-27.122761
-3.450419	0.164136	-9.183778	0.001072	-0.005677	0.0067	0.225142	-4.830938	0.119282	-0.726538	0.121513	0.133403	0.663009	-87.017993	-4.995183	-27.122641
-3.450454	0.164035	-9.183726	0.001146	-0.005688	0.006685	0.225312	-4.830957	0.119282	-0.72654	0.121515	0.133405	0.663006	-87.017768	-5.000014	-27.122522
-3.450489	0.163934	-9.183672	0.00122	-0.005698	0.00667	0.225481	-4.830976	0.119281	-0.726542	0.121517	0.133407	0.663003	-87.017543	-5.004845	-27.122403
-3.450524	0.163833	-9.183616	0.001296	-0.005707	0.006654	0.225651	-4.830995	0.119281	-0.726544	0.121518	0.133409	0.663	-87.017317	-5.009676	-27.122284
-3.45056	0.163732	-9.183558	0.001371	-0.005715	0.006639	0.22582	-4.831014	0.119281	-0.726546	0.12152	0.133411	0.662997	-87.017091	-5.014507	-27.122164
-3.450595	0.163631	-9.183499	0.001447	-0.005723	0.006624	0.225989	-4.831033	0.119281	-0.726548	0.121522	0.133413	0.662995	-87.016865	-5.019338	-27.122045
-3.45063	0.16353	-9.183438	0.001523	-0.00573	0.006609	0.226158	-4.831051	0.11928	-0.72655	0.121524	0.133415	0.662992	-87.016639	-5.024169	-27.121926
-3.450665	0.16343	-9.183375	0.0016	-0.005735	0.006593	0.226327	-4.83107	0.11928	-0.726551	0.121526	0.133418	0.662989	-87.016413	-5.029	-27.121806
-3.450701	0.16333	-9.183311	0.001677	-0.00574	0.006578	0.226496	-4.831089	0.11928	-0.726553	0.121527	0.13342	0.662986	-87.016187	-5.033831	-27.121687
-3.450736	0.163229	-9.183246	0.001754	-0.005744	0.006563	0.226665	-4.831108	0.11928	-0.726555	0.121529	0.133422	0.662983	-87.01596	-5.038662	-27.121568
-3.450771	0.163129	-9.183181	0.001831	-0.005747	0.006548	0.226834	-4.831127	0.11928	-0.726557	0.121531	0.133424	0.66298	-87.015733	-5.043493	-27.121449
-3.450807	0.163029	-9.183114	0.001908	-0.005749	0.006533	0.227002	-4.831146	0.11928	-0.726559	0.121533	0.133427	0.662977	-87.015506	-5.048324	-27.121329
-3.450842	0.162929	-9.183047	0.001985	-0.00575	0.006517	0.227171	-4.831165	0.11928	-0.726561	0.121534	0.133429	0.662975	-87.015279	-5.053156	-27.12121
-3.450878	0.16283	-9.182979	0.002062	-0.00575	0.006502	0.227339	-4.831184	0.11928	-0.726563	0.121536	0.133431	0.662972	-87.015052	-5.057987	-27.121091
-3.450913	0.16273	-9.182911	0.00214	-0.005749	0.006487	0.227507	-4.831203	0.11928	-0.726565	0.121537	0.133434	0.662969	-87.014825	-5.062818	-27.120971
-3.450949	0.16263	-9.182842	0.002217	-0.005746	0.006472	0.227675	-4.831222	0.11928	-0.726567	0.121539	0.133436	0.662966	-87.014597	-5.067649	-27.120852

Ilustración 12: Datos del sistema Gacebo

Diseño de las redes

Arquitectura:

Las decisiones de arquitectura de la red se toman en base a experimentos. No hay un sistema para, por ejemplo, saber si para x aplicación es mejor el optimizador Adam que el del gradiente descendiente o si va a funcionar mejor la función de activación "sigmoide" que la "tanh".

Las principales decisiones a tomar son:

1. **Función de activación de las neuronas:** relu, tanh, sigmoid…
2. **Optimizador:** gradient decent, Adam, SGD, RMSprop…
3. **Función fitness:** mean squared error, mean absolute error, cosine similarity…
4. **Número y tipo de capas:** Lambda, Dense, dropuot, flattened…
5. **Cantidad de neuronas por capa.**

La selección de estos atributos es complicada. Un problema es que el número de posibles combinaciones es demasiado como para probar todas y además no son características aisladas discretas entre sí. Esto quiere decir que incluso si solo nos restringimos a ajustar uno de los atributos dejando los otros constantes, puede que lleguemos a un valor de dicho atributo que es "el mejor" solo para la configuración fijada del resto de los atributos. Otro problema es que yo en mis experimentos lo que pruebo es que una arquitectura converge más rápido o lento que otra en los 10 primeros epochs de entrenamiento. Podría darse el caso de que la arquitectura que converge más lento en los 10 primeros epochs llegue a un mínimo menor tras 50 epochs. Al final lo que nos importa no es cuán rápido converge el entrenamiento sino cual es el mínimo valor de la función de coste escogida a la que puede llegar. Este método, a pesar de imperfecto, es el que he usado yo para seleccionar la mejor función de activación:

En mi caso use una red de 7 capas densas totalmente conexas, con 1024 neuronas cada una, la función fitness "mean squared error" y optimizador Adam con step size=0.00015. Con esa misma red neuronal probé las tres funciones de activación más comúnmente usadas que son: lineal (sin función), sigmoide, tanh, y relu. Los resultados tras 10 epochs de entrenamiento son:

- **Lineal:** Loss:0.0048 val_loss:0.012
- **Sigmoid:** Loss:0.0040 val_loss:0.0013
- **Tanh:** Loss:0.0042 val_loss:0.0015
- **Relu:** Loss:0.0039 val_loss:0.001

A partir de este momento me limité a usar la función “relu” en todas las pruebas y cambiar otras características como por ejemplo en número y tamaño de capas. De todas formas, después, en las arquitecturas finales, el mejor resultado lo obtuve combinando capas relu con sigmoide.

Entrada/salida

Color verde son entradas de la red, color rojo son salidas/resultados. En los tres ejemplos n = Cuántos datos atrás le entran a la red = 10:

- **Estrategia A:** Esta sería la estrategia propuesta: **Ilustración 6:** Arquitectura propuesta al comienzo del proyecto.
  En ella se pasa el estado inicial (posición y orientación), después una serie de datos de la imu y tras esos datos se le “pregunta” a la red cual es la nueva posición.

aceleracion			gyroscopio			magnetometro			angulos euler			posicion			
0.0411377	0.955933	0.356323	0.0504158	-0.273755	-0.111491	-13.8557	-9.49668	-24.9452	-10.6108	-4.41931	1.29667	-128	-249	278	i-10
0.0960693	0.964111	0.338867	0.0721975	-0.182998	-0.0735126	-13.8366	-9.47422	-24.968	-10.1259	-4.46162	1.35023	-133	-251	274	i-9
0.0927734	0.972534	0.315308	0.15709	-0.0176806	-0.0698823	-13.8213	-9.45625	-25.1255	-9.63457	-4.55798	1.32521	-134	-277	262	i-8
0.0552979	0.986084	0.293701	0.223832	0.025045	-0.0531271	-13.8091	-9.44188	-25.1123	-9.1592	-4.83131	1.30531	-153	-263	274	i-7
0.0450439	0.977417	0.297241	0.235281	-0.0182391	-0.0520101	-13.7993	-9.57475	-25.3801	-8.4663	-5.2522	1.45497	-157	-269	267	i-6
0.0495605	0.994751	0.285156	0.360107	-0.0659913	-0.111212	-13.9363	-9.3923	-25.4552	-8.12957	-5.81569	1.27296	-153	-252	280	i-5
0.0656738	1.01819	0.280396	0.404508	-0.165405	-0.179908	-13.9011	-9.24634	-25.5153	-7.33464	-6.58728	1.5648	-130	-267	276	i-4
0.0456543	0.928345	0.249023	0.311796	-0.216229	-0.140254	-14.1626	-9.41832	-25.8417	-6.56847	-7.27586	2.57497	-102	-256	261	i-3
0.00585938	0.981567	0.282227	0.368485	-0.352784	-0.00593342	-13.9373	-9.70028	-25.9637	-5.92493	-7.67701	3.64578	-83	-236	275	i-2
0.067749	0.956177	0.29248	0.478789	-0.461692	0.182283	-14.1915	-9.6371	-26.3397	-5.51481	-7.83561	4.24169	-80	-247	316	i-1
0.0983887	1.04968	0.337524	0.564799	-0.303077	0.401217	-13.6707	-9.29781	-26.5013	-4.73271	-7.91933	4.66257	-68	-224	308	i

Ilustración 13: Estrategia A

- **Estrategia B:** En esta estrategia no se le pregunta a la red por la posición absoluta sino por el movimiento en cada eje que se infiere de esos datos de la imu.

aceleracion			gyroscopio			magnetometro			angulos euler			posicion				deltas posicion		
0.0411377	0.955933	0.356323	0.0504158	-0.273755	-0.111491	-13.8557	-9.49668	-24.9452	-10.6108	-4.41931	1.29667	-128	-249	278	i-10	-45	-13	3
0.0960693	0.964111	0.338867	0.0721975	-0.182998	-0.0735126	-13.8366	-9.47422	-24.968	-10.1259	-4.46162	1.35023	-133	-251	274	i-9	-53	-4	-42
0.0927734	0.972534	0.315308	0.15709	-0.0176806	-0.0698823	-13.8213	-9.45625	-25.1255	-9.63457	-4.55798	1.32521	-134	-277	262	i-8	-66	-53	-46
0.0552979	0.986084	0.293701	0.223832	0.025045	-0.0531271	-13.8091	-9.44188	-25.1123	-9.1592	-4.83131	1.30531	-153	-263	274	i-7	-94	-68	-22
0.0450439	0.977417	0.297241	0.235281	-0.0182391	-0.0520101	-13.7993	-9.57475	-25.3801	-8.4663	-5.2522	1.45497	-157	-269	267	i-6	-99	-98	-17
0.0495605	0.994751	0.285156	0.360107	-0.0659913	-0.111212	-13.9363	-9.3923	-25.4552	-8.12957	-5.81569	1.27296	-153	-252	280	i-5	-83	-89	-32
0.0656738	1.01819	0.280396	0.404508	-0.165405	-0.179908	-13.9011	-9.24634	-25.5153	-7.33464	-6.58728	1.5648	-130	-267	276	i-4	-60	-136	-3
0.0456543	0.928345	0.249023	0.311796	-0.216229	-0.140254	-14.1626	-9.41832	-25.8417	-6.56847	-7.27586	2.57497	-102	-256	261	i-3	-74	-256	261
0.00585938	0.981567	0.282227	0.368485	-0.352784	-0.00593342	-13.9373	-9.70028	-25.9637	-5.92493	-7.67701	3.64578	-83	-236	275	i-2	-83	-236	275
0.067749	0.956177	0.29248	0.478789	-0.461692	0.182283	-14.1915	-9.6371	-26.3397	-5.51481	-7.83561	4.24169	-80	-247	316	i-1	=M92-M83	2	38

Ilustración 14: Estrategia B

Como se ve en la imagen, la columna “deltas posicion” no es la diferencia entre el elemento i y el (i-1) sino la diferencia entre el element i y (i-10)

- **Estrategia C:** En este caso se le entrega a la red la posición i, para calcular la siguiente (i+1) junto con los n datos de la imu anteriores.
-

aceleracion			gyroscopio			magnetometro			angulos euler			posicion			
0.0411377	0.955933	0.356323	0.0504158	-0.273755	-0.111491	-13.8557	-9.49668	-24.9452	-10.6108	-4.41931	1.29667	-128	-249	278	i-10
0.0960693	0.964111	0.338867	0.0721975	-0.182998	-0.0735126	-13.8366	-9.47422	-24.968	-10.1259	-4.46162	1.35023	-133	-251	274	i-9
0.0927734	0.972534	0.315308	0.15709	-0.0176806	-0.0698823	-13.8213	-9.45625	-25.1255	-9.63457	-4.55798	1.32521	-134	-277	262	i-8
0.0552979	0.986084	0.293701	0.223832	0.025045	-0.0531271	-13.8091	-9.44188	-25.1123	-9.1592	-4.83131	1.30531	-153	-263	274	i-7
0.0450439	0.977417	0.297241	0.235281	-0.0182391	-0.0520101	-13.7993	-9.57475	-25.3801	-8.4663	-5.2522	1.45497	-157	-269	267	i-6
0.0495605	0.994751	0.285156	0.360107	-0.0659913	-0.111212	-13.9363	-9.3923	-25.4552	-8.12957	-5.81569	1.27296	-153	-252	280	i-5
0.0656738	1.01819	0.280396	0.404508	-0.165405	-0.179908	-13.9011	-9.24634	-25.5153	-7.33464	-6.58728	1.5648	-130	-267	276	i-4
0.0456543	0.928345	0.249023	0.311796	-0.216229	-0.140254	-14.1626	-9.41832	-25.8417	-6.56847	-7.27586	2.57497	-102	-256	261	i-3
0.00585938	0.981567	0.282227	0.368485	-0.352784	-0.00593342	-13.9373	-9.70028	-25.9637	-5.92493	-7.67701	3.64578	-83	-236	275	i-2
0.067749	0.956177	0.29248	0.478789	-0.461692	0.182283	-14.1915	-9.6371	-26.3397	-5.51481	-7.83561	4.24169	-80	-247	316	i-1
0.0983887	1.04968	0.337524	0.564799	-0.303077	0.401217	-13.6707	-9.29781	-26.5013	-4.73271	-7.91933	4.66257	-68	-224	308	i

Ilustración 15: Estrategia C

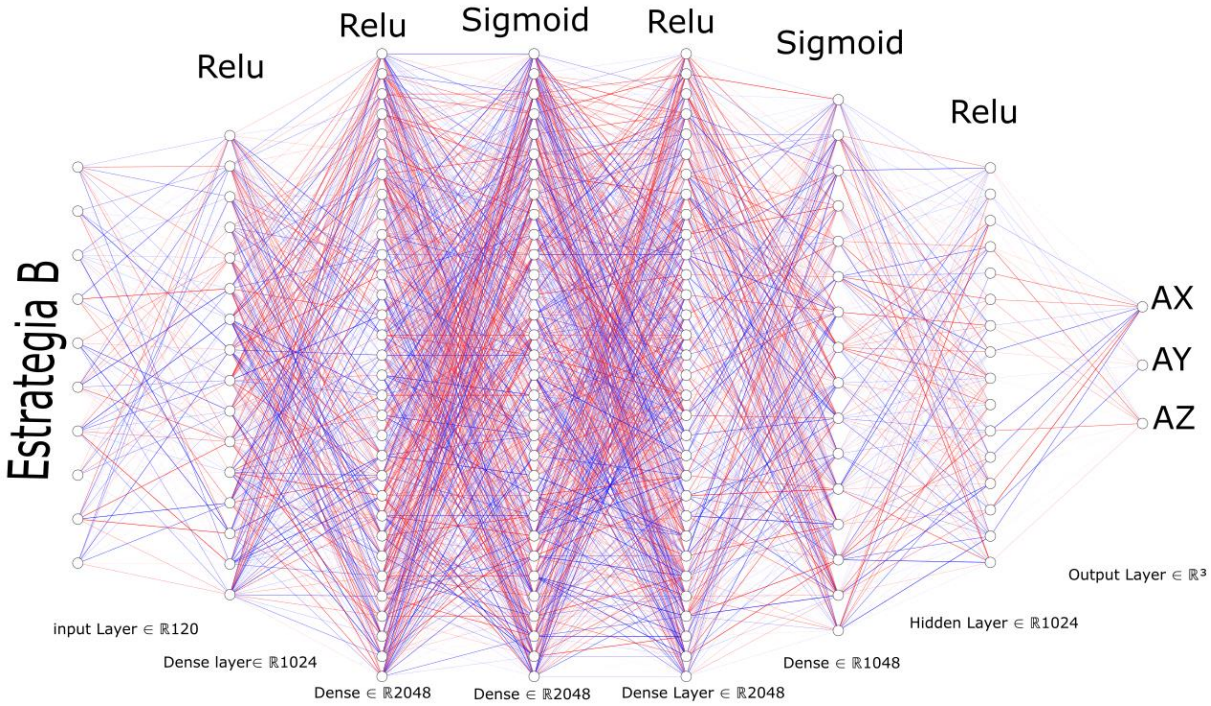
He descartado los siguientes intentos:

- **Usar los datos de Gacebo con la estrategia C:** La red convergía bien, pero puesto que los datos de Gacebo son de una frecuencia más alta (250hz) la diferencia entre la posición i-1 y la posición i-2 es poca. Lo que la red hacía vagamente es retornar como resultado de posición en i-1 el valor de entrada que recibe i-2. Puesto que estos dos valores son tan semejantes el valor de la función fitness era bueno, pero obviamente la red no estaba correlacionando los datos, simplemente escupía la entrada de nuevo.
- La estrategia A usa N*9+6 entradas mientras que la B y la C usan 12*N. Para N pequeño la diferencia no es mucha, pero si necesitamos usar N grandes, conviene usar la estrategia A. En el caso de los datos de Gacebo, puesto que son de mayor frecuencia (250hz) si tan solo ponemos N=10 la red solo estaría mirando al pasado 10*1/250=0.04 segundos. Es por esto que para los datos de Gacebo se usa la estrategia A.

Redes finales:

A partir de aquí y después de muchas pruebas, he seleccionado las tres arquitecturas que mejor resultado han dado en los 20-25 primeros epochs.
Estas son:

1. **Red numero 1º:** Esta red utiliza la **Ilustración 14:** Estrategia B con los datos reales de la imu mas las etiquetas del sistema de cámaras y balizas (**Ilustración 10:** vista de Excel pero el fichero sigue siendo .csv con N=10 datos atrás lo cual a 30hz equivale a mirar atrás 33ms, tiene 7 capas ocultas densas con tamaños:



2. **Red numero 2º:** Esta red utiliza la Ilustración **13:** Estrategia A con los datos obtenidos de **Ilustración 12:** Datos del sistema Gacebo, tiene 7 capas ocultas densas con tamaños:

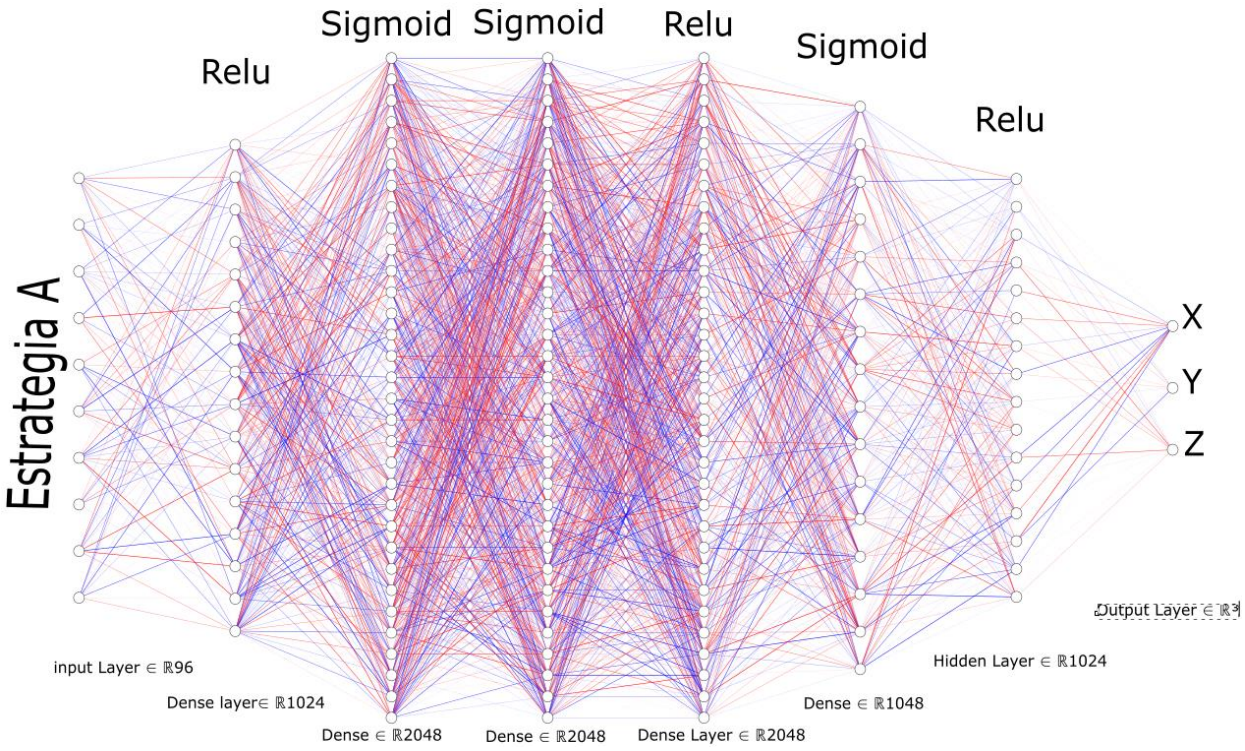


Ilustración 16: arquitectura de la red 2.

Esta arquitectura de red se ha probado con tres valores de N: 10, 20 y 30. Lo único que cambiaría de la **Ilustración 16**: arquitectura de la red 2. es la dimensión de la capa de entrada. La dimensión de la capa de entrada dim = N*numero_capos_imu+posición_i-N.shape[0]+orientación_i-N.shape[0] = N*9+3+3

Entrenamiento de las redes.

Para entrenar las redes uso las gpus en la nube de google colab y cada entrenamiento tarda aproximadamente 6h para las redes arriba descritas.
Google colab aproximadamente cada hora, si detecta inactividad, desconecta la sesión del servidor. Para solventarlo abro la consola de Chrome y ejecuto el siguiente script:

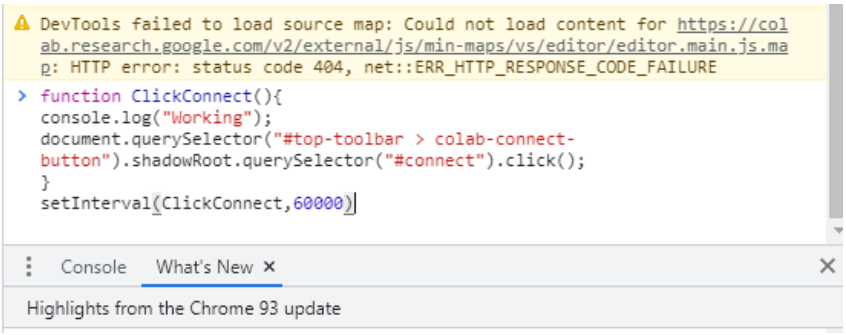


Ilustración 17: Script que clica un botón de la interfaz periódicamente

Incluso con este truco, cada 5h[6] la página pedirá un capcha. La única forma de evitar que te desconecten es despertarse y completar el capcha.

Para todos los entrenamientos y validaciones se han dividido los datos en test_set y train_set. La idea es que un 10% de los datos se dediquen para evaluar el rendimiento de la red no ha visto nunca y por tanto no ha podido "memorizar". Queremos que la red encuentre los patrones que relacionan los datos de entrada con el resultado de salida no que aprenda de memoria cada valor de salida correspondiente a cada valor de entrada.

El valor del step_size del optimizador Adam lo he ido variando en alguno de los entrenamientos. En los primeros epochs valor relativamente alto ~0.0017 y en los últimos un valor más bajo para aproximarme lo más posible al mínimo local.

[6] Realmente nose si es cada 5h. Sospecho que es aleatorio para confundir al enemigo.

Red numero 1º:

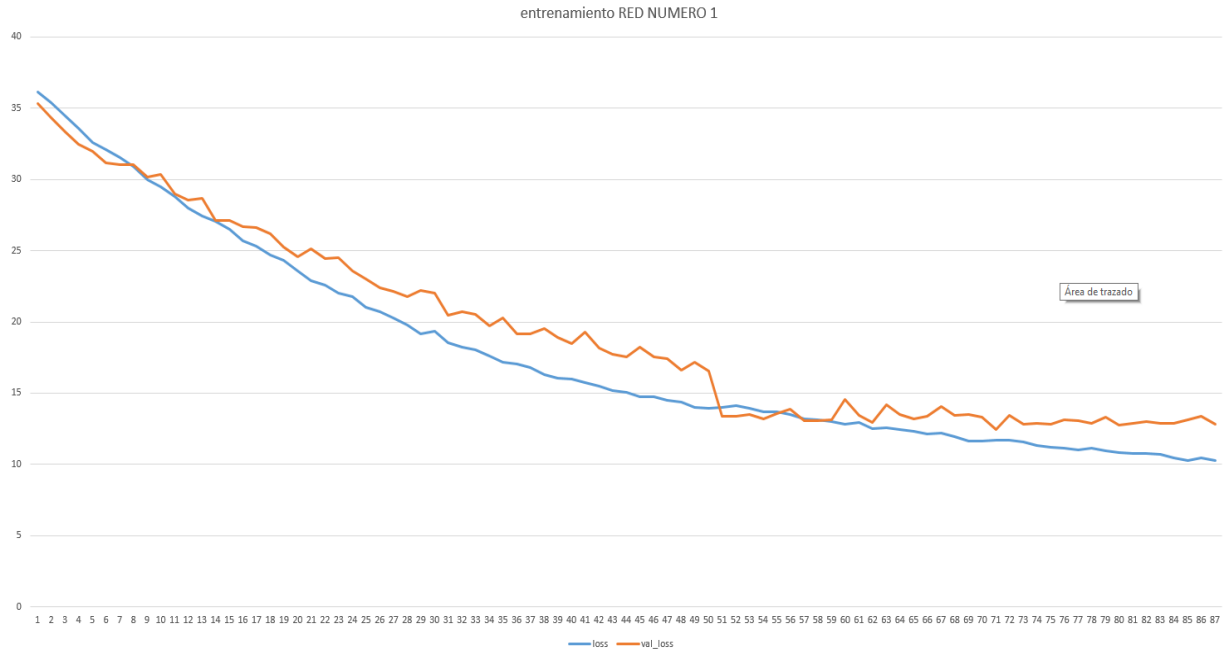


Ilustración 18: 87epochs, min loss=10.25 ,min validation_loss=12.84

Red numero 2º:

N=10

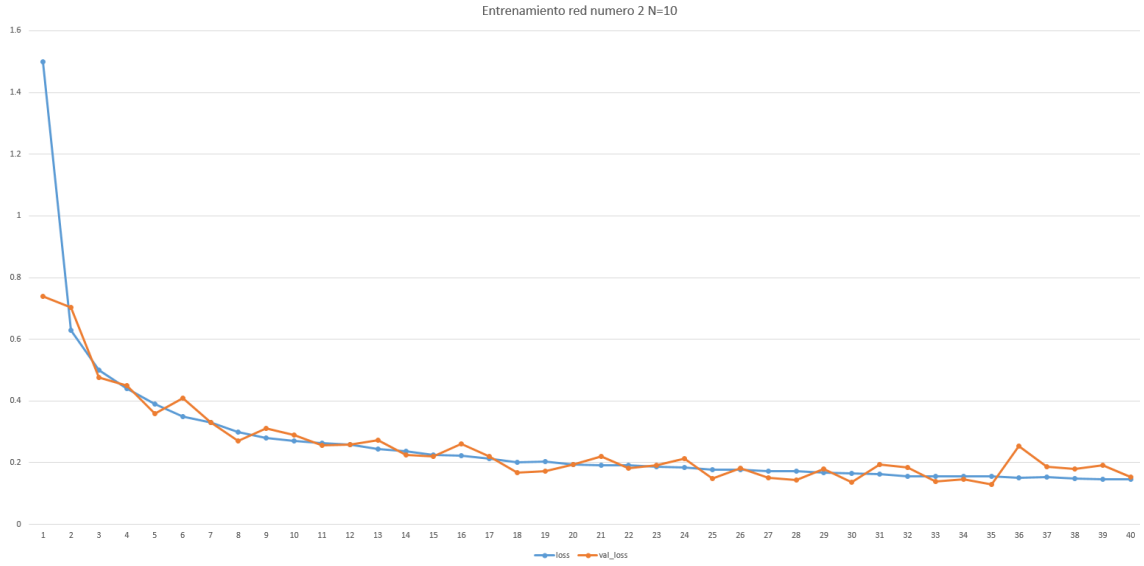


Ilustración 19: 40 epochs, min loss=0.1472 ,min validation_loss=0.154

N=20

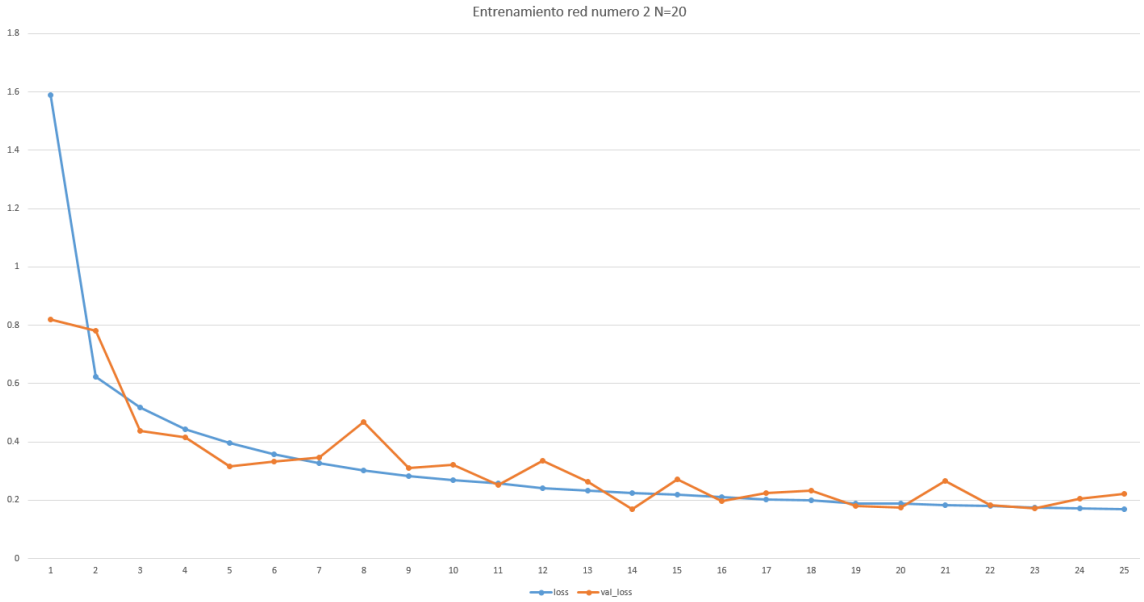


Ilustración 20: 25epochs, min loss=0.1697 ,min validation_loss=0.221

N=30

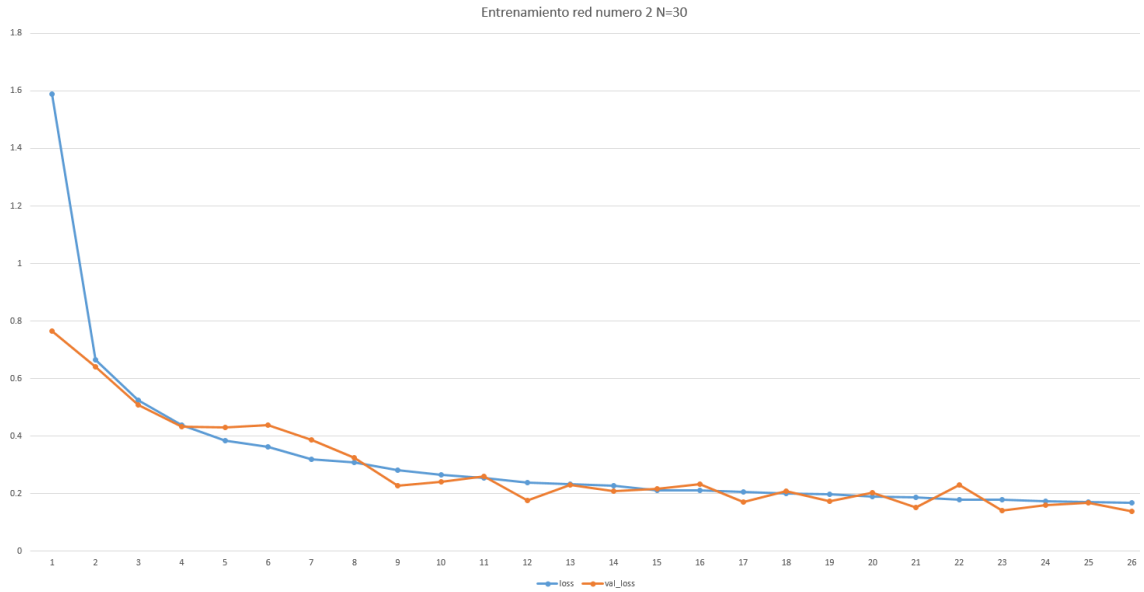
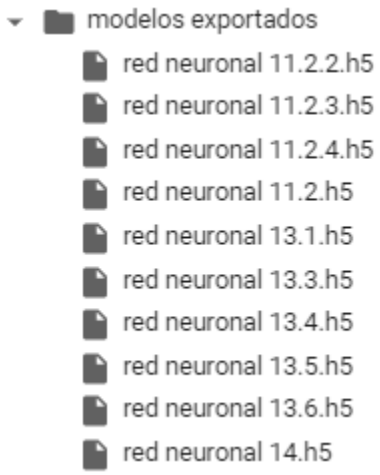


Ilustración 21: 26 epochs, min loss=0.1689 ,min validation_loss=0.1389

Inferencia y análisis de resultados

Una vez entrenadas las redes, exportamos los modelos ya entrenados en formato .h5:

```
model.save('/content/drive/MyDrive/beca investigacion/modelos exportados/red neuronal 13.4.h5')
```



Estos modelos ya contienen todo lo necesario para ser usados (ocupan aproximadamente 240MB). Se pueden usar en cualquier máquina con una consola python importando las librerías necesarias:

```
import tensorflow as tf
from tensorflow import keras
import numpy as np
import pandas as pd
from tensorflow.keras.models import load_model
```

Y cargando el modelo en una variable con:

```
model = load_model('/content/drive/MyDrive/beca investigacion/modelos exportados/red neuronal 13.6.h5')
```

Tan solo faltaría preparar los datos en forma de numpy array y pasarselos a predict():

```
predictions = model.predict(X)
```

Obviamente los datos no solo tienen que estar en el mismo formato, escala y orden sino que tienen que ser a la misma frecuencia a la que la red se entrenó. No se puede tratar de inferir con datos de otro sensor con otra calibración y escala numérica.

Resultados de la red 1º:

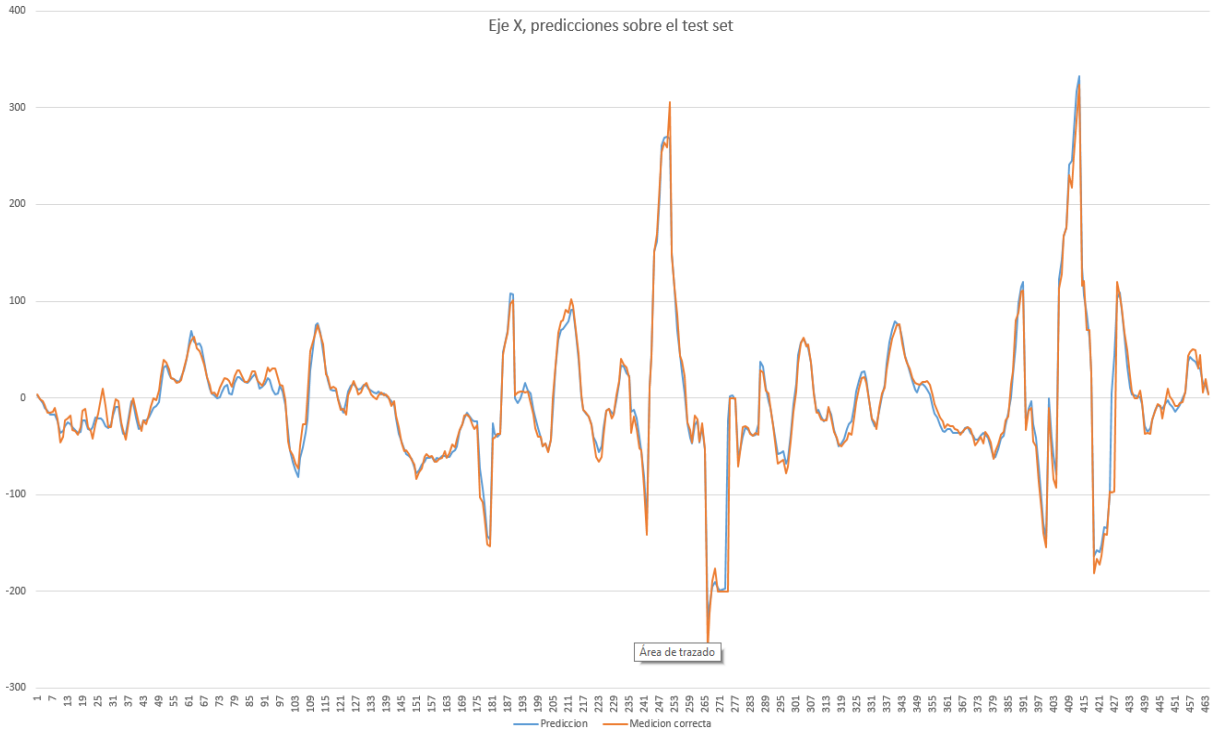


Ilustración 22

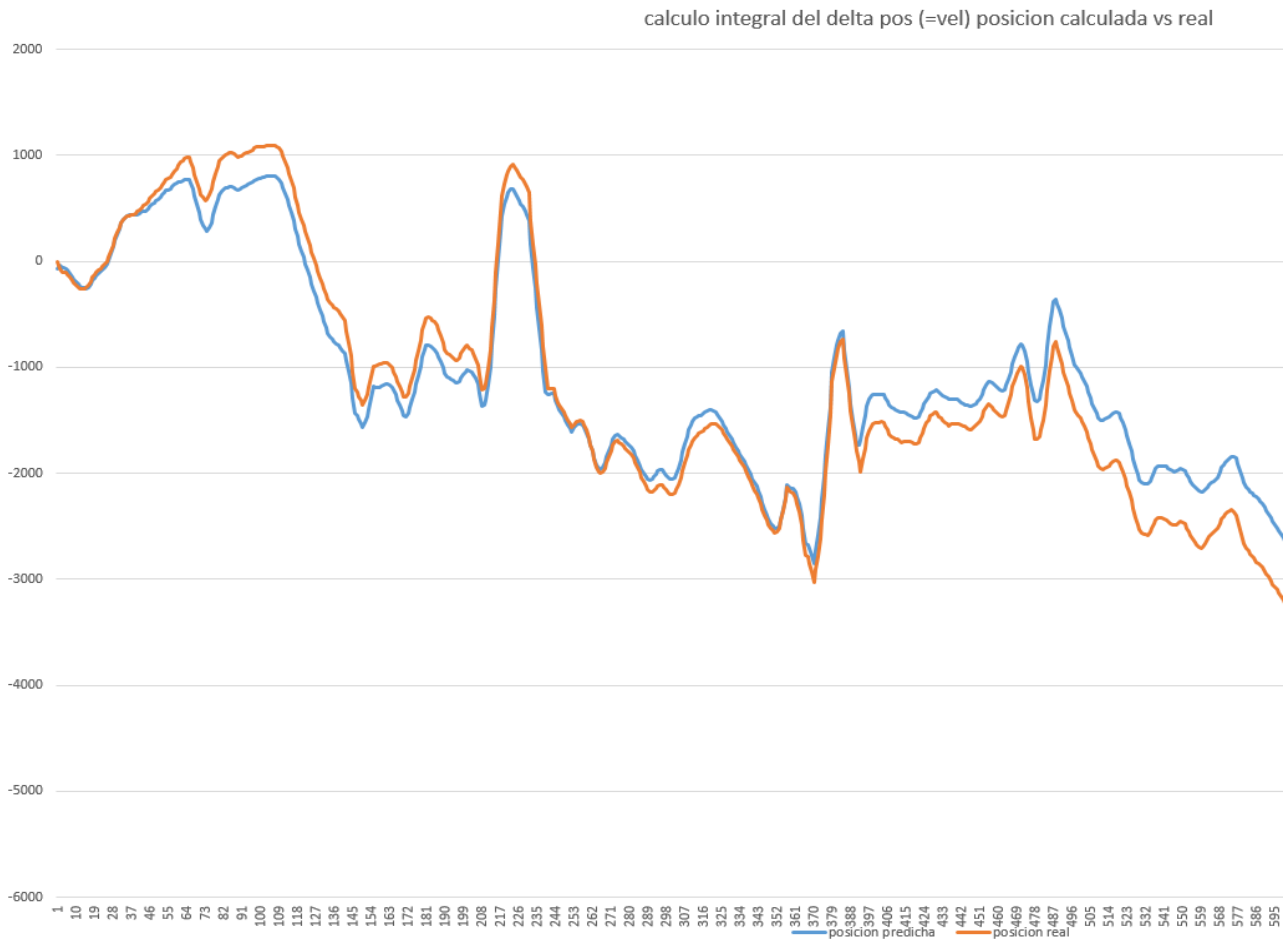
En naranja es el valor correcto y en azul la predicción.

Esto es una sección de ~500 datos, aleatoria y representativa de la calidad del total hecho sobre datos de validación.

Esta gráfica puede ser engañosa: Los resultados son muy buenos en cuanto a precision pero hay que tener en cuenta que esa gráfica solo ayuda a ilustrar el rendimiento de la red para inferencias aisladas entre sí. Cada punto que compone la linea de la gráfica es una inferencia aislada y el resultado de esta red es el delta en la posicion, en este caso del eje x. La gráfica muestra una seccion de 500 samples equivalentes a ~16segundos. Esto no quiere decir que el sistema de posicionamiento aguante 16 segundos sin deriva puesto que no representa la posicion. Quizá una representación mas adecuada sería:

Eje X		error
Valor correcto esperado	prediccion de la red	error
51	45.129913	-5.870087
78	78.86178	0.86178
108	108.45269	0.45269
113	115.555275	2.555275
123	118.32014	-4.67986
119	117.78015	-1.21985
107	111.51439	4.51439
110	101.27368	-8.72632
80	83.30193	3.30193
75	70.500656	-4.499344
63	65.396156	2.396156
41	37.33965	-3.66035
32	31.484243	-0.515757
18	12.570331	-5.429669
10	5.295026	-4.704974
5	2.1926744	-2.8073256
-2	-1.028643	0.971357
-8	-9.207453	-1.207453
-31	-30.368338	0.631662
-52	-49.687737	2.312263
-62	-51.63983	10.36017
-56	-48.880863	7.119137
-51	-43.949875	7.050125
-34	-30.573086	3.426914
-30	-29.334864	0.665136
-34	-33.301235	0.698765
-40	-36.169003	3.830997
-40	-36.70633	3.29367
-35	-34.650726	0.349274
-32	-27.975502	4.024498
-25	-20.832947	4.167053
-16	-18.373066	-2.373066
-27	-26.800001	0.199999

Otra representación interesante es hacer la integral/sumatorio movil de la gráfica Ilustración 22. Esto ya sí que nos daría la posicion real frente a la posicion predicha usando la red:



En esta gráfica se muestra la suma acumulada de los delta Posicion que veiamos en la **Ilustración 22.** Esto equivale a la integral de dicha grafica y a la vez a la integral de la velocidad que es la posicion. He ampliado la cantidad de tiempo que se muestra en la gráfica porque el rendimiento es bastante bueno. Ahora se muestran 595 datos a 30 hz equivale a 19 segudos. A partir de esos 19 segundos el error continuaba a peor.

Resultados de la red 2º:

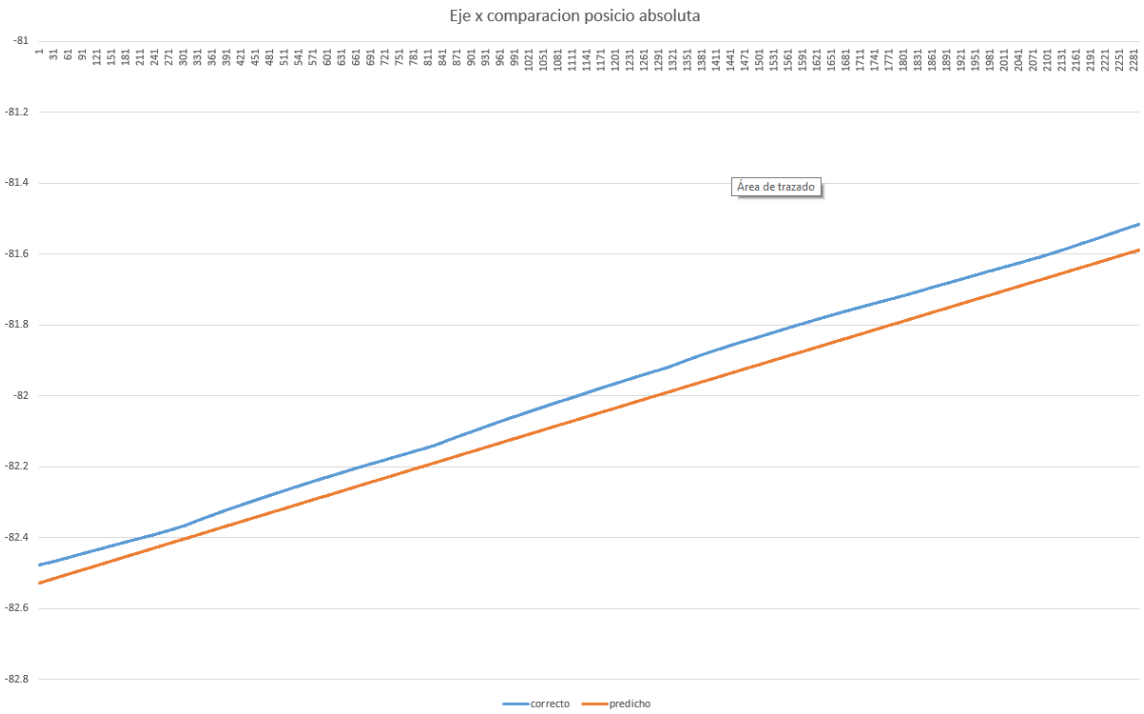
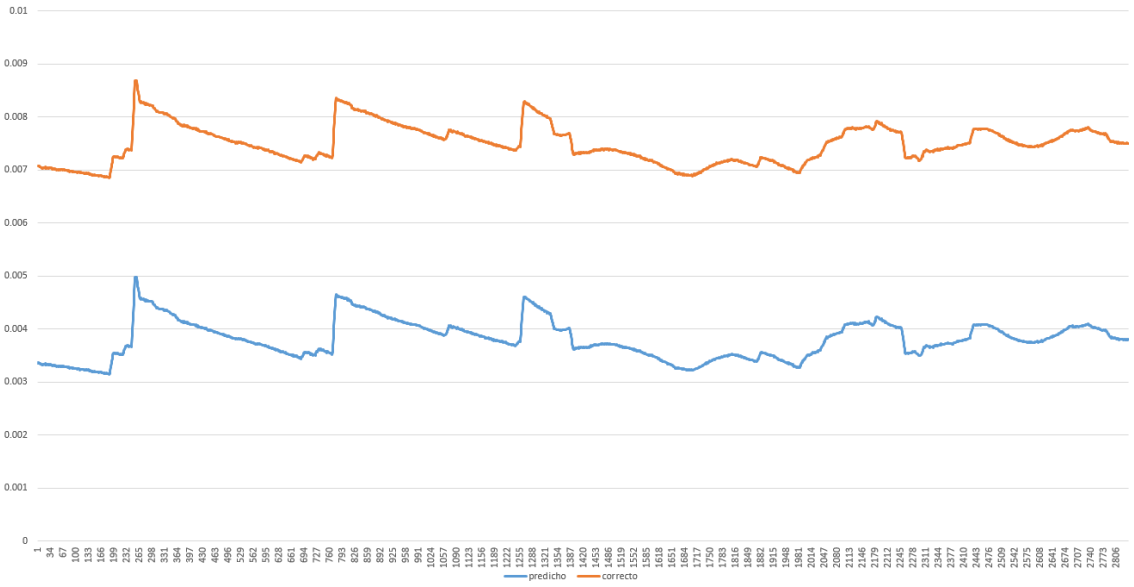


Ilustración 23: 2200 datos a 250hz son 9s en los que el drone simulado parece que se movía a velocidad constante.

En esta gráfica se muestra poca cosa primero porque casualidad que el vuelo en ese momento era bastante constante y segundo porque hay que tener en cuenta que la **Estrategia A** que usamos en la red numero 2º tiene como dato de entrada la posicion en (i-9) para cascular la posicino en i. Esto hace que la red lo tenga “facil”. En este caso es mas útil mirar cuanto y hacia que setido se desvía el calculo de la red con respecto a la posicion que le damos de entrada y compararlo con lo real:



Si por ejemplo a la red le pasamos como entrada que estamos en la posición x=7, los datos de los sensores y luego le preguntamos cual es la siguiente posición y nos dice 5. Lo que estamos ilustrando en esta gráfica entonces sería 5-7=-2.

Conclusiones

Como hemos visto, una red neuronal como la planteada Sí es capaz de cumplir las funciones del filtro kalman y demás algoritmos. El uso de recursos siempre que se ejecute en la gpu es bajo y rápido. Esto abre muchas posibilidades, puesto que no solo la implementación es más simple (subjetivo), podemos ejecutarlo en gpus, no requiere de calibración del sensor (pero sí de entrenamiento) y además tenemos la posibilidad de un mejor rendimiento. Sobre esto último decir que la arquitectura propuesta no es ni mucho menos óptima. No sabemos si el rendimiento de esta red es mejor que el del algoritmo de (Gómez, oct 2018) puesto que no se presentan datos objetivos de rendimiento. El objetivo de este trabajo era una prueba de concepto para explorar si realmente era posible y si la idea tiene futuro. En mi opinión sí lo tiene, merece la pena invertir más trabajo en seguir desarrollando y sobre todo cuantificar y comparar el rendimiento real.

Líneas futuras

En mi opinión lo más interesante a partir de ahora es la idea de que una red no solo interprete los datos de un solo sensor sino varios. Por ejemplo, manos, pies y cabeza. Y ver si es capaz de encontrar patrones de movimiento que relacionen cada parte del cuerpo. Para esto haría falta tener un laboratorio con un sistema de tracking a través de cámaras para conseguir datos de entrenamiento de alta calidad de varias partes del cuerpo:

Referencias

1. **Elmenreich, Wilfried.** *Sensor Fusion in Time-Triggered Systems.* oct 2002.

2. **Villar Bonet, Eugenio, y otros.** *Método y sistema de localización espacial mediante marcadores luminosos para cualquier ambiente.* 2014.

3. **Gómez, Alba Ruiz.** *Sistema de Posicionamiento Basado en Fusión Sensórica y Gestion de la Orientación.* oct 2018.

4. **Madgwick, Sebastian O.H.** *An efficient orientation filter for inertial and inertial/magnetic sensor arrays.* University of Bristol. April 30, 2010.

5. **Varela, Raúl Gómez.** *Sistema de posicionamiento 3D visual multipróposito, mediante marcas led.* dic 2020.

6. **InvenSense Inc.** *MPU-6000 and MPU-6050 Product Specification.* 2013.

7. **ARIEL LAREY, ELIEL AKNIN y ITZIK KLEIN.** *Multiple Inertial Measurement Units - an.* s.l. : IEEE Access, 2016.

8. **DEBITETTO, PAUL ALLEN.** *ROBUST HIERARCHICAL IMAGE-AUGMENTED NAVIGATION IN URBAN TERRAIN WITH 3D LANDMARKS.* Boston University. 2011.

9. **desconocido.** *Historia de las redes neuronales.* 2005.

10. **Siegelmann, Hava T.** *On The Computational Power Of Neural Nets.* Department of Computer Science, Rutgers University. 1992.

11. **Marinković, Javier , Pérez, Jorge y Barceló, Pablo.** *On the Turing Completeness of Modern Neural Network Architectures.* Department of Computer Science, Universidad de Chile. 2019.

12. **Iven Sense.** *MPU-9255.* 2014.