Universidad de Alcalá

Escuela Politécnica Superior

GRADO EN INGENIERÍA ELECTRÓNICA DE COMUNICACIONES

Trabajo Fin de Grado

Actualización de herramientas para diseño e implementación de controladores digitales con Matlab2018

Autor: JAVIER ROBLEDO ARÉVALO

Tutor: FELIPE ESPINOSA ZAPATA

Cotutor: CARLOS SANTOS PÉREZ

2019

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

Grado en Ingeniería Electrónica de Comunicaciones

Trabajo Fin de Grado

Actualización de herramientas para diseño e implementación de controladores digitales con Matlab2018

Autor: Javier Robledo Arévalo

Tutor: Felipe Espinosa Zapata

Cotutor: Carlos Santos Pérez

TRIBUNAL:

Presidente: Jose Manuel Rodríguez Ascariz

Vocal : Cristina Losada Gutiérrez

Vocal 2: Felipe Espinosa Zapata

FECHA: 24 de mayo de 2019

Quiero dedicar este trabajo a mis padres, Miguel Ángel y Carmen,

y mi hermano, Víctor,

porque sin su ayuda nada de esto hubiera sido posible.

Se lo agradezco enormemente a mis compañeros

de grado por su apoyo;

y a Felipe Espinosa y Carlos Santos por su ayuda.

Índice general

Índice de figuras

I. INTRODUCCIÓN GENERAL

1. Resumen

En este Trabajo Fin de Grado (TFG), se trata de actualizar las herramientas de diseño asistido por computador, basado en Matlab, para el diseño e implementación de controladores digitales. Actualmente se está trabajando, en asignaturas de grado relativas a control de procesos, con herramientas de Matlab 2012 sobre Ubuntu 12.04. Lo que suele generar problemas con las versiones nativas de sistema operativo y aplicaciones de cálculo de los alumnos, razón por la cual se recurre a máquinas virtuales que reproducen las herramientas de laboratorio.

Por lo tanto, se propone en este trabajo una actualización de software de herramientas de Matlab 2018b, sobre Windows 10 y Ubuntu 18.04.

Abstract

In this End of Degree Project (TFG), the aim is to update the computer-aided design tools, based on Matlab, for the design and implementation of digital controllers. Currently, work is being done on degree subjects related to process control, with Matlab 2012 tools on Ubuntu 12.04. What usually generates problems with the native versions of the operating system and student calculation applications, which is why we resort to virtual machines that reproduce the laboratory tools.

Therefore, a software update of Matlab 2018b tools on Windows 10 and Ubuntu 18.04 is proposed in this paper.

Palabras clave: Herramientas CACSD, Matlab 2018b, Windows 10 Ubuntu 18.04.

2. Motivación y objetivos

La motivación de este trabajo nace de la incomodidad que puede llegar a suponer trabajar con herramientas desactualizadas en prácticas de laboratorio, tanto para los estudiantes como para el personal del laboratorio. Bajo esta premisa, a continuación, se detallan los pasos realizados desde una versión anterior de las herramientas que permiten el diseño y la implementación de controladores digitales hasta llegar a versiones actuales a fecha de expedición de este mismo documento.

Los objetivos, por lo tanto, son los de actualizar las herramientas que entran en juego para el diseño y la implementación remota de controladores digitales con Matlab 2018b, aplicados al robot P3-DX que se muestra en la siguiente imagen:

Figura 1. Pioneer P3-DX mobile robot

II. DESARROLLO

Se inicia el estudio sobre Windows 10, realizando un ensayo sobre comunicación por socket entre nodos con este mismo sistema operativo, y más tarde sobre diferentes versiones de Linux (Ubuntu10.04 y Ubuntu18.04) con servidores creados mediante la generación de código con diferentes targets: Generic Real Time proporcionado por Matlab, y RTAI (RealTime Application Interface), que nace de aplicar una extensión software a Linux que permite a las aplicaciones funcionar a tiempo real.

Una vez comprobada la funcionalidad de la comunicación para todos los casos anteriores se procederá a actuar sobre el robot mostrado en la figura 1 a modo de ejemplo de interacción con hardware externo de ejecutables generados con Matlab/Simulink. Dicho robot está soportado por el sistema operativo Linux (Ubuntu 12.04), soporte software que se pretende mantener, pero la comunicación se realizará desde un PC que trabaja sobre Windows 10. Se comienza con un control remoto para seguimiento lineal de velocidades implementando un servosistema, y a continuación se procede a un estudio más complejo que trata sobre el control remoto para seguimiento no lineal de trayectorias basado en Lyapunov con ayuda de las cámaras Kinect disponibles en el entorno de laboratorios del Departamento de Electrónica.

Después se detallan los ensayos realizados cuando el cliente está soportado por Linux en su versión más reciente: Ubuntu18.04. Generando código desde este mismo sistema operativo con el software Matlab2018b, y recreando los estudios finales realizados cuando el cliente estaba soportado en Windows.

3. Aplicaciones de control en tiempo real con Windows 10 y Matlab 2018b

3.1. Introducción

Durante este capítulo se trabaja con la premisa de que el cliente en las comunicaciones es un ordenador con el sistema operativo Windows 10. Habrá distintos servidores en función de los estudios que se van a realizar: Ubuntu10.04, Ubuntu12.04, Ubuntu18.04 y Windows 10.

Antes de seguir con las explicaciones del trabajo que aquí se presenta, es necesario poner en contexto por qué se utiliza el sistema operativo de Linux para ejecuciones en tiempo real y no Windows o cualquier otro. Todo se resume en la flexibilidad que tiene cada sistema operativo y cuanta personalización puede asumir cada uno de ellos.

A nivel usuario, Windows es una buena opción debido a su sencillez a la hora de instalar programas, compatibilidad entre software, etc. Es el sistema operativo más utilizado en el mundo como se puede ver en la figura 2el cual muestra un gráfico de los sistemas operativos más utilizados globalmente durante el año 2017, y eso hace que no existan apenas problemas cuando se quieren realizar tareas que no requieran de unas características de operación muy específicas como puede ser el que se estudia en este documento: la ejecución a tiempo real.

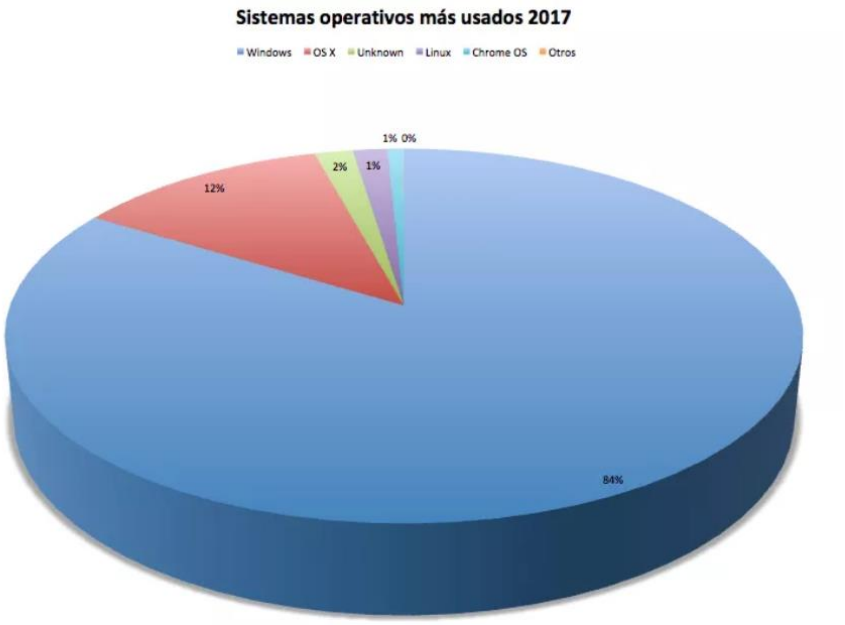


Figura 2. Gráfico de sistemas operativos más usados [Fireosoft, 2017]

Aparentemente, apostar por Windows supone tener sólo ventajas, pero como se ha comentado, todo depende de las especificaciones que se requieran. En cuanto a las ejecuciones a tiempo real Windows no es una apuesta segura, ya que no es un sistema operativo que permita modificar los valores de prioridad de las tareas que se ejecutan y esto hace que no se tenga una frecuencia de ejecución estable. El tipo de características que tenga el PC que va a realizar las operaciones influye mucho, ya que si se tiene un ordenador con un procesador lento los tiempos de ejecución serán mayores, en cambio, si se trabaja con un PC de gama alta, los periodos de muestreo pueden ser menores y será más fácil realizar tareas a tiempo real debido a que la máquina puede soportar tal carga de trabajo y atender tareas que emergen en plena ejecución de manera muy rápida. Para los estudios que alberga este documento se ha utilizado un PC que actúa como cliente con el sistema operativo Windows 10 y con las siguientes características hardware:

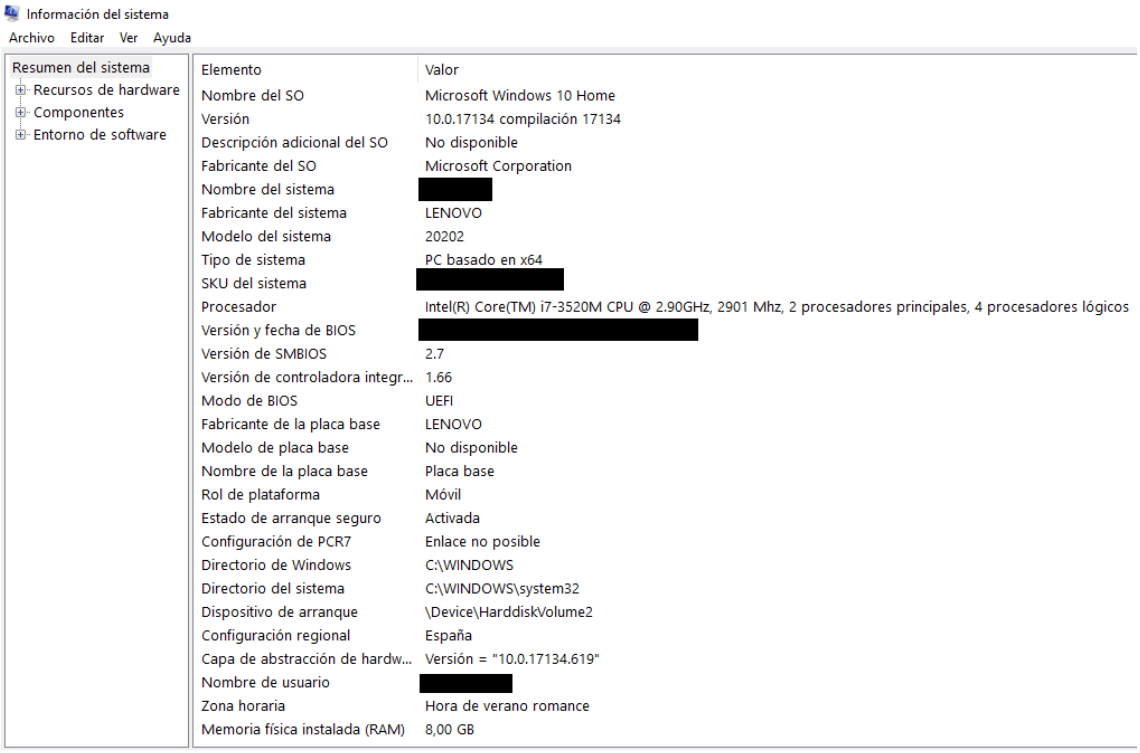


Figura 3. Características hardware del centro remoto con Windows10

Por ejemplo, al lanzar una aplicación de duración total de 10 segundos, con un periodo de muestreo de 100 milisegundos, generará un total de 100 muestras. El resultado de una aplicación con Windows es que se realizarán las 100 muestras, pero el periodo de muestreo no es fijo durante la ejecución ya que, si surge una tarea no deseada en el momento que se está ejecutando la aplicación, este sistema operativo atiende la operación, como puede ser que llegue un correo electrónico, o una alerta del antivirus, etc. Por lo tanto, se tiene un sistema operativo que realiza el número de muestras determinadas pero el periodo de muestreo tiene variaciones que no lo hacen fiable. Más adelante, en este capítulo, se exponen gráficas que demuestran las desviaciones que se tienen en distintos estudios realizados.

Por lo tanto, se tiene que Windows es un sistema operativo extendido, fácil de usar, intuitivo y compatible con la mayoría de software, pero que cuando se requiere de precisión y fiabilidad a la hora de ejecutar aplicaciones en tiempo real, este no es capaz de garantizar periodos estables.

En cambio, Linux es un sistema operativo abierto, es decir, que se pueden modificar parámetros que en Windows son imposibles de acceder, como puede ser la frecuencia de oscilación del procesador, la prioridad de atención a las tareas, etc.

3.2. Aplicación en tiempo real sin acceso a hardware externo

Como introducción a la ejecución en tiempo real sobre Windows se va a realizar un pequeño ensayo en el que no existe interacción con hardware externo y se comparan los resultados obtenidos en simulación y ejecución en tiempo real de una aplicación diseñada con Simulink.

El ejemplo es el mismo para ambos casos, un seno de amplitud 1, que es amplificado a la salida con factor 3. La ejecución está impuesta para que duré un total de 15 segundos y con un periodo de muestreo de 50ms, es decir 300 muestras. La configuración del entorno de Simulink es muy importante, y esta se puede consultar en el Anexo I de este documento.

Por lo tanto, se tiene como arquitectura de bloques el escalón y un fichero para almacenar los datos. Dicha formación se observa en la siguiente figura:

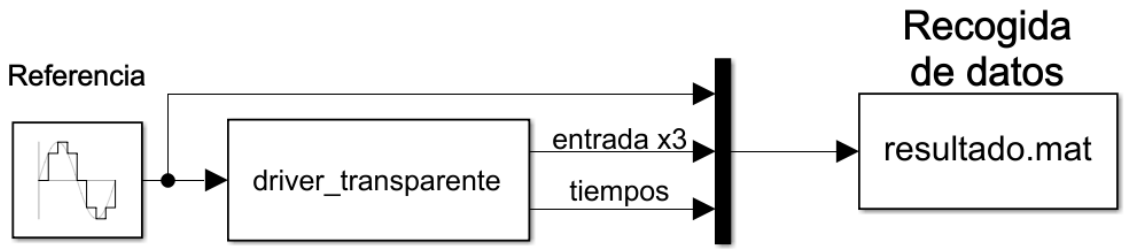


Figura 4. Arquitectura de bloques en Simulink para comparar simulación y ejecución en tiempo real de una misma aplicación.

Como se ha comentado en el punto 3.1, Windows presenta inestabilidad a la hora de realizar tareas en tiempo real, por lo que de una ejecución a otra varían los resultados, aunque se trate de la misma aplicación, ya que se pueden tener tareas paralelas ejecutándose en momentos diferentes. La figura 5 muestra cómo de ocupada estaba la CPU a la hora de lanzar la aplicación destinada para la ejecución en tiempo real de este ensayo.

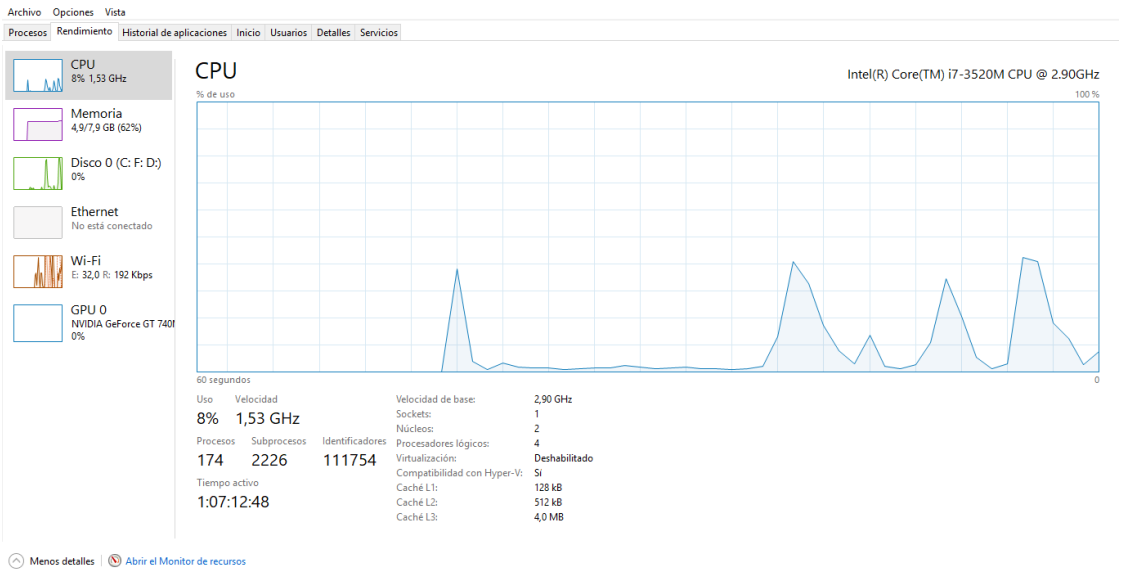


Figura 5. Rendimiento de CPU a la hora de ejecutar aplicación en tiempo real

3.2.1. Comparación de aplicación en simulación y tiempo real

Se ha realizado una simulación del bloque correspondiente a la figura 4, y más tarde se construyó un ejecutable a partir de Simulink. Los resultados obtenidos son totalmente idénticos, es decir, se tiene como conclusión que cuando no existe interacción con hardware externo Windows 10 sí cumple los periodos de muestreo y proporciona resultados idénticos a los obtenidos en simulación. En la figura 6 que se presenta a continuación, se muestran los resultados obtenidos a la salida del escalón, tanto para el camino realizado por simulación como para el obtenido a través de una ejecución en tiempo real:

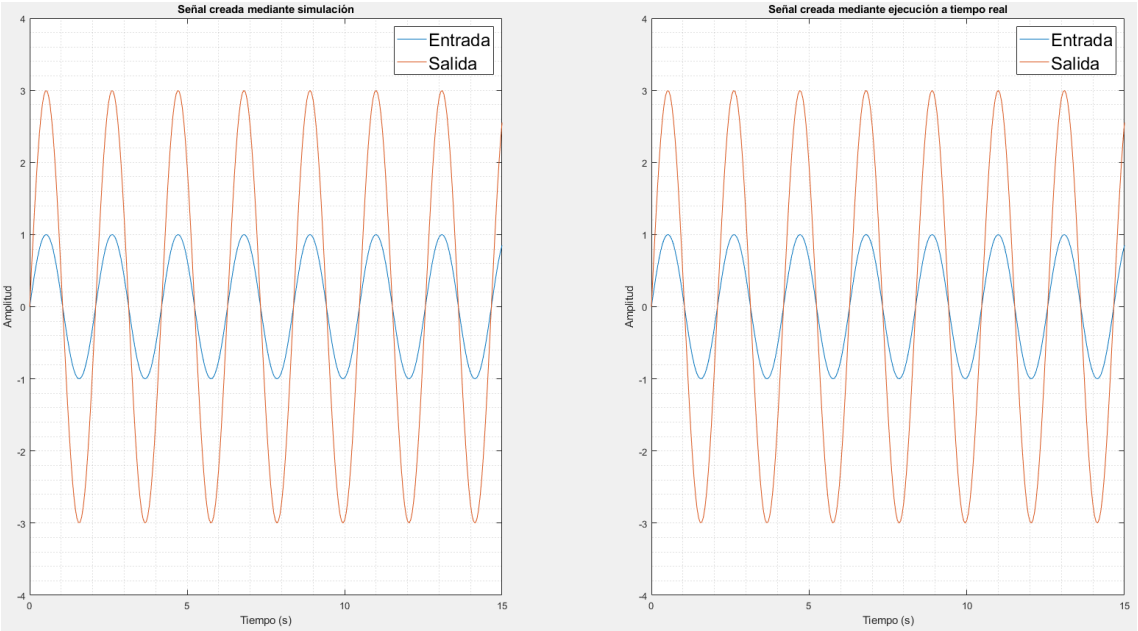


Figura 6. Resultados obtenidos de simulación y ejecución en tiempo real

Como prueba de la temporización de la ejecución total del ejecutable y el tiempo existente entre cada una de las muestras realizadas, se muestra en la siguiente figura una representación del tiempo que ha transcurrido entre cada muestra y la siguiente, así como un dato estadístico que indica la duración total real de la aplicación, la media del periodo de muestreo obtenido y la desviación típica resultante respecto al valor ideal, que se había establecido en 50ms:

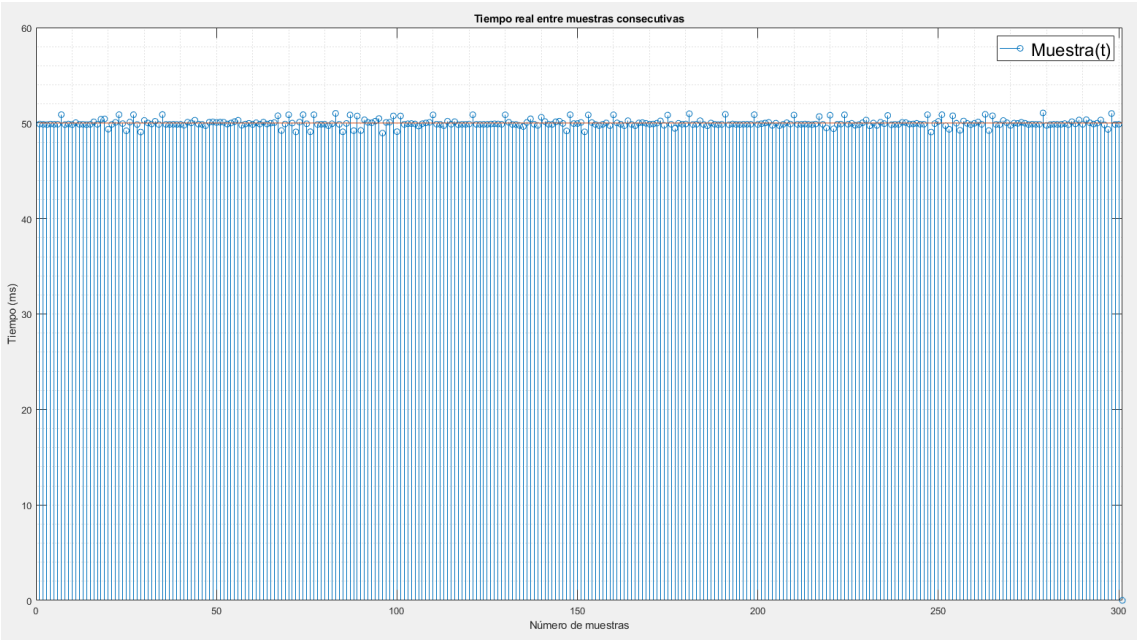


Figura 7. Tiempo entre muestras consecutivas en la ejecución en tiempo real

```
Desviación típica: 2.908369ms
Media: 49.833548ms
La ejecucion ha durado 14.999898 segundos en vez de 15 segundos
```

Se valora positivamente el resultado obtenido, ya que el periodo de muestreo no se desvía del valor ideal establecido y el tiempo de ejecución total real es el indicado en la creación de la aplicación, frente al segundo que se tarda en obtener los mismos resultados mediante la simulación.

3.3. Aplicación en tiempo real con acceso a hardware externo

En este apartado se parte de la base de la generación de código para su ejecución en tiempo real como se ha explicado en el punto 3.1, y se incorpora la interacción con hardware externo al PC en uso. En este caso, se han creado distintos escenarios para la comunicación por socket entre un PC cliente ejecutado en Windows10 y servidores instalados en Windows10, Ubuntu10.04, Ubuntu12.04 y Ubuntu18.04. Tanto las aplicaciones del cliente como las de los servidores son generadas, teniendo en cuenta los correspondientes targets, para su ejecución en tiempo real.

A modo de introducción, para la comunicación por socket un servidor crea una puerta de enlace asociada a su IP y un puerto específico. Más tarde identifica esa puerta de enlace y queda a la espera de que reciba un mensaje mediante una función bloqueante. Cuando un dato es recibido, es tratado, se envía un dato a la dirección IP cliente que contactó en primer lugar y el servidor vuelve al estado de espera de un nuevo dato. Por otro lado, desde el punto de vista del cliente, se le indica la dirección IP del servidor y el mismo puerto que se estableció en la aplicación del servidor. Una vez creada la puerta de enlace con el servidor se envía la petición y el cliente pasa a un estado de espera de datos procesados provenientes del servidor. Una vez recibidos se espera al siguiente periodo de muestreo y se repite el proceso. Este proceso de envío y recepción en bucle se repite hasta que llegue el final de la ejecución y así poder cerrar la puerta de enlace utilizada para comunicarse con el servidor. En la siguiente figura se muestra un flujograma que explica de manera intuitiva cómo funciona esta lógica de sockets para una comunicación UDP (comunicación no orientada a la conexión):

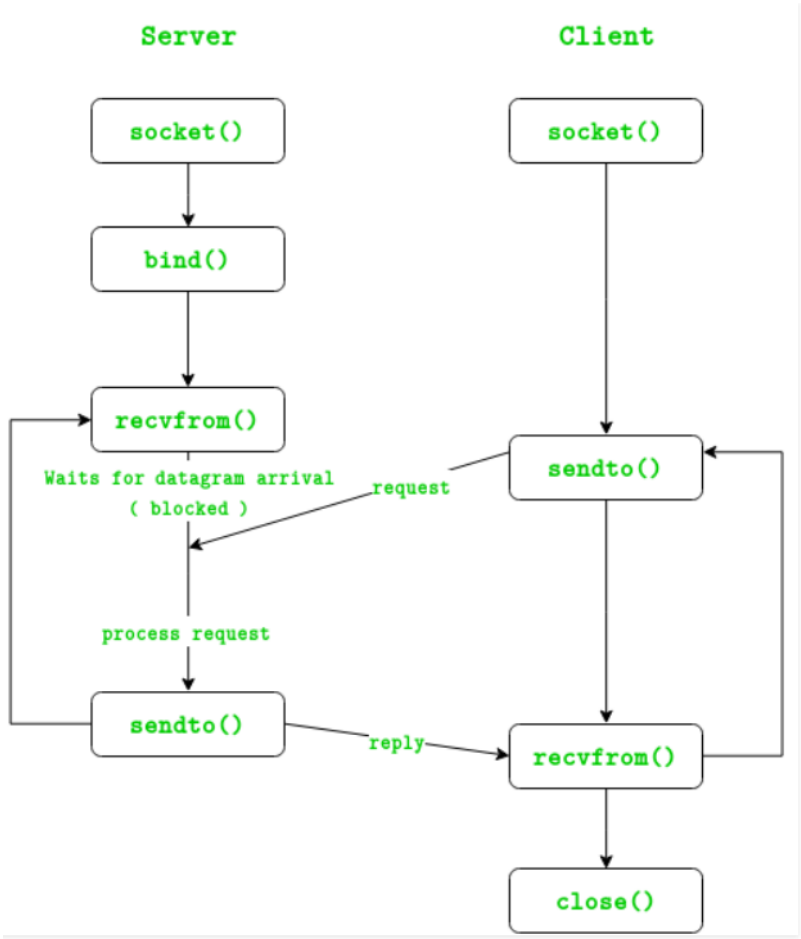


Figura 8. Flujograma de la lógica de sockets [Geeksforgeeks, 2018]

Para la comunicación entre los distintos dispositivos es necesario un driver Simulink que realice las funciones anteriormente descritas, como pueden ser la creación del socket, la espera de datos, el envío de unos nuevos, etc. Estos drivers están instanciados en una s-function de Matlab-Simulink programada en C. Estos archivos de código fuente están reflejados en el Anexo II de la memoria, incluyendo los pertenecientes a este apartado de interacción básica por socket entre dispositivos y también los códigos generados para el control remoto de varios P3-DX.

Hay dos tipos diferenciados de implementación de la s-function:

Implementación de s-function con grt.tlc (Generic Real-Time) como target de Simulink.

Este target es el apropiado para la generación de código en Windows o sobre Ubuntu18.04. En ambos casos la implementación de la s-function es muy parecida, salvo que en Windows es necesario hacer referencia a una dll (dynamic-link library) que deberá estar incluida en el directorio donde se esté trabajando, siendo dicha librería

"WS2_32.lib", lo que permite la compilación de las librerías de comunicación de Windows: Winsock2.h. Una vez aclarado esta pequeña diferencia, a continuación, se muestran las instrucciones necesarias para compilar el código y poder implementar la s-function en Matlab-Simulink sin problemas.

Para el caso de trabajar sobre el sistema operativo de Windows10, es necesario tener instalado un compilador compatible con software de Matlab. En el Anexo III se detalla la información necesaria para la descarga, instalación y configuración de uno de los compiladores que Matlab soporta.

En el directorio de trabajo de Matlab debe estar tanto el fichero fuente de la s-function (MySourceFile.c) como el enlace dinámico 'WS2_32.LIB', y se ejecuta la siguiente sentencia:

```
mex MySourceFile.c -lWS2_32
```

A continuación, debe mostrarse un mensaje por la consola de Matlab que confirme el éxito en la compilación del código y se generará el archivo con extensión .mexw64 y el mismo nombre que tiene el archivo fuente. En el ejemplo anterior se debe generar un archivo llamado MySourceFile.mexw64. Este paso es necesario para poder implementar la s-function de manera correcta y además sirve como depurador de código, ya que si existen errores sintácticos en el código este comando los reportará.

En el caso de que se quiera volver a generar una compilación de código no es necesario borrar el archivo generado anteriormente, ya que la nueva ejecución del comando borrará al anterior.

Para el caso de trabajar con el target grt.tlc pero sobre el sistema operativo de Linux, como es el caso de Ubuntu18.04, no es necesario instalar ningún compilador adicional, ya que el que viene por defecto con este SO es soportado por Matlab. En este documento se ha trabajado con la versión 7.3.0 de gcc.

Del mismo modo que en Windows, el usuario se debe encontrar en el directorio Matlab donde esté el archivo fuente de interés. Una vez allí se ejecuta la siguiente instrucción:

```
mex -compatibleArrayDims MySourceFile.c
```

siendo de nuevo 'MySourceFile.c' el archivo que incluye el código fuente a compilar. De manera análoga a Windows, la consola de Matlab debe reportar un mensaje de éxito en la operación MEX y generará un archivo de extensión .mexa64, por lo que para el ejemplo anterior se obtiene MySourceFile.mexa64.

Si se requiere la inclusión de archivos de cabeceras con extensión .h, se deben incluir seguidos de las instrucciones anteriormente descritas. A continuación, se muestra la sintaxis necesaria para crear el archivo .mexw64 si se trabaja sobre el sistema operativo Windows, o .mexa64 en el caso de Linux, cuando el usuario posee un fichero .c con la necesidad de la inclusión de archivos .h:

```
mex MySourceFile.c -Iheader1 -Iheader2
```

Siendo “header1.h” y “header2.h” los archivos cabecera a incluir.

Finalmente, a modo de ejemplo, para la compilación de un código programado en C, incluido en el archivo “centro_remoto.c”, con referencias a un archivo de cabecera que recibe el nombre de “funciones_cremoto.h”, y trabajando sobre Windows10 con comunicaciones por socket, se debe escribir en Matlab lo siguiente:

```
mex centro_remoto.c -Ifunciones_cremoto.h -lWS2_32
```

Implementación de s-function con rtai.tlc como target de Simulink.

Como se explica en la introducción de este documento, RTAI es la extensión que algunas versiones de Ubuntu utilizan como interfaz para la aplicación en tiempo real y garantiza el cumplimiento de los periodos de muestreo. Al ser esta una extensión que se incluye al sistema operativo de Linux se deben realizar acciones diferentes a las citadas en el punto anterior. Estos pasos que se muestran a continuación han sido realizados cuando se trabaja con RTAI, y por lo tanto en los estudios que afectan a Ubuntu10.04 y Ubuntu12.04.

La compilación del código para la implementación de la s-function mediante el target rtai.tlc está detallada en [Salazar, 2008]

Una vez compilado el código fuente e implementada la s-function dentro del bloque de Simulink y configurado el panel del Code Generator tal y como se explica en el Anexo I, ya sólo es necesario comenzar la construcción del ejecutable, utilizando el atajo Ctrl + B cuando el entorno de Simulink esté abierto. En la figura 9 se muestra un ejemplo del modelo utilizado en Simulink para el apartado 3.3.1. Para el resto de los apartados únicamente es necesario cambiar el nombre del driver al que se referencia en la s-function y, si el usuario lo quiere, también modificar el nombre del bloque de salida que almacena los datos.

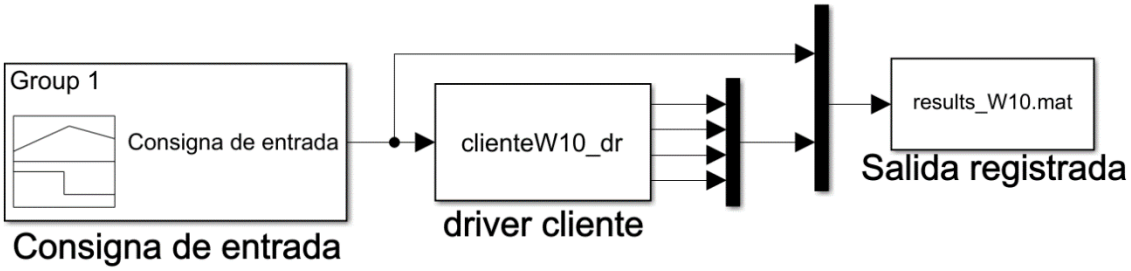


Figura 9. Modelo de Simulink con driver para validar comunicación

A continuación, se describen 4 escenarios de interacción con hardware externo, en el que el cliente es soportado por Windows10 en el mismo dispositivo desde donde se realizó el estudio del apartado 3.2, y los servidores se implementan en distintos dispositivos con sistemas operativos diferentes. La lógica implementada es simple, ya que se pretende validar la comunicación, por lo que se trata del envío de una señal por parte del cliente hacia el servidor, y una vez que el servidor reciba el dato, este debe devolverla multiplicada por 3 hacia el cliente.

3.3.1. Drivers cliente y servidor interaccionando en Windows10

Para este ejemplo de comunicación entre dispositivos que están soportados ambos por Windows10, ha sido necesario implementar cambios en los respectivos drivers de comunicación para garantizar la compatibilidad dentro del SO con el que se trabaja. Dichos cambios se ven reflejados en el Anexo II, donde se encuentran los drivers utilizados para el cliente y el servidor, denominados clienteW10_dr.c y servidorW10_dr.c respectivamente. Para la realización de este experimento, se deben crear sendos ejecutables tal y como se ha mostrado anteriormente y ser ejecutados cada uno en su dispositivo. Primero se debe ejecutar la aplicación en el servidor, para que este quede a la espera de datos provenientes del cliente, y una vez preparado el escenario ya se podrá ejecutar la aplicación en el lado del cliente.

Como se ha comentado en la introducción del apartado 3.3, se trata de validar la comunicación entre dispositivos con ejecución en tiempo real, donde el cliente mandará una consigna y el servidor la devolverá multiplicada por 3, por lo tanto, dichas señales de salida y entrada del cliente han sido registradas para su posterior exposición. También se han obtenidos datos respecto a la conexión, recogiendo el número de paquetes UDP enviados y recibidos y los perdidos en la transmisión. Por último, el periodo de muestreo elegido para estos ensayos es de Ts=50ms, y se han registrado unas variables temporales que marcan la duración total real de la ejecución, así como los tiempos transcurridos entre muestras.

A continuación, se muestran los resultados graficados que se han obtenido al realizar dicha comunicación dentro de la misma red local mediante enlace alámbrico, por lo que asegura una mayor fiabilidad a la hora de enviar y recibir los paquetes y por ende se tendrá un mejor resultado. La siguiente imagen corresponde con la representación de las consignas, tanto la enviada por el cliente como la que recibe del servidor:

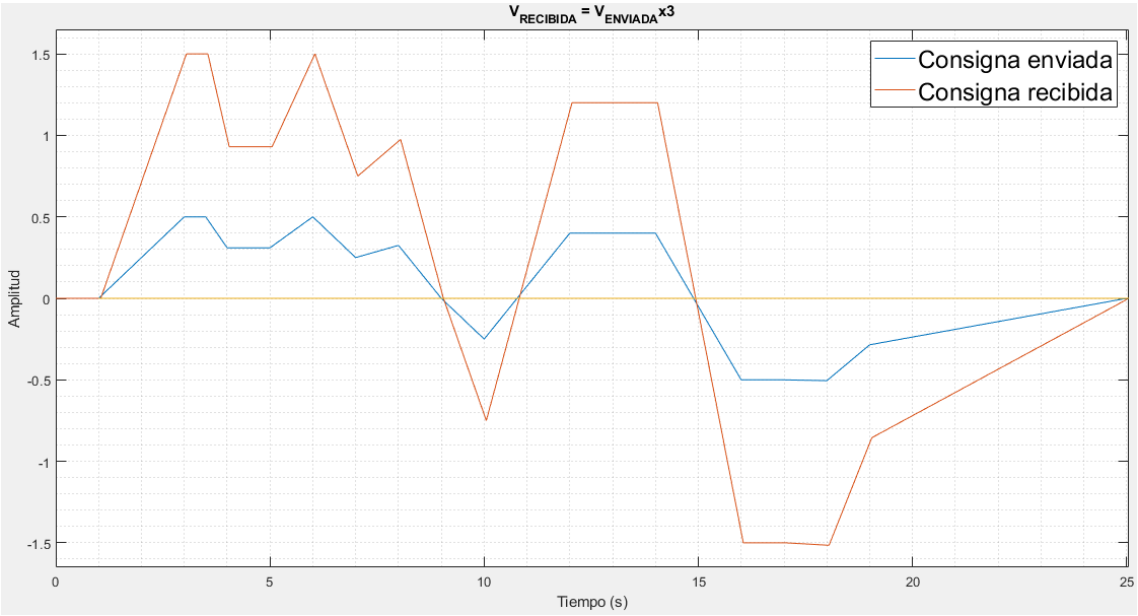


Figura 10. Señales registradas en el ejemplo de comunicación W10-W10

En la figura 11 se muestra una gráfica que ilustra lo relacionado con la conexión existente, mostrando la cantidad de paquetes totales enviados por el cliente, los recibidos por el mismo, la cantidad de paquetes perdidos y cuándo han sido perdidos.

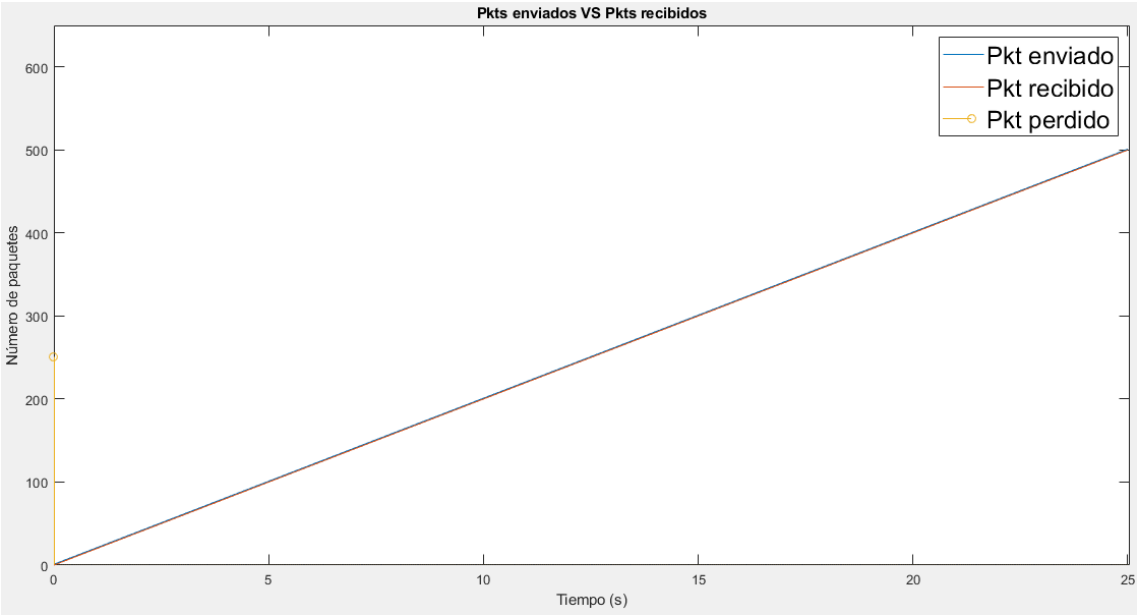


Figura 11. Paquetes enviados, recibidos y perdidos en la comunicación W10-W10

Como última imagen de este experimento se muestra lo más relevante a la ejecución en tiempo real, y es el tiempo que ha existido entre cada una de las muestras con la siguiente, creando una recta horizontal sobre el valor 50ms, para indicar el valor ideal con el que se trabajaba y poder comparar los valores de periodo de muestreo real que se ha obtenido a la hora de realizar este ensayo. También se adjunta información adicional, como puede ser el tiempo total de ejecución en la aplicación cliente, la media del periodo de muestreo, la desviación típica que existe respecto al valor ideal (50ms) y la cantidad de paquetes totales perdidos en la comunicación.

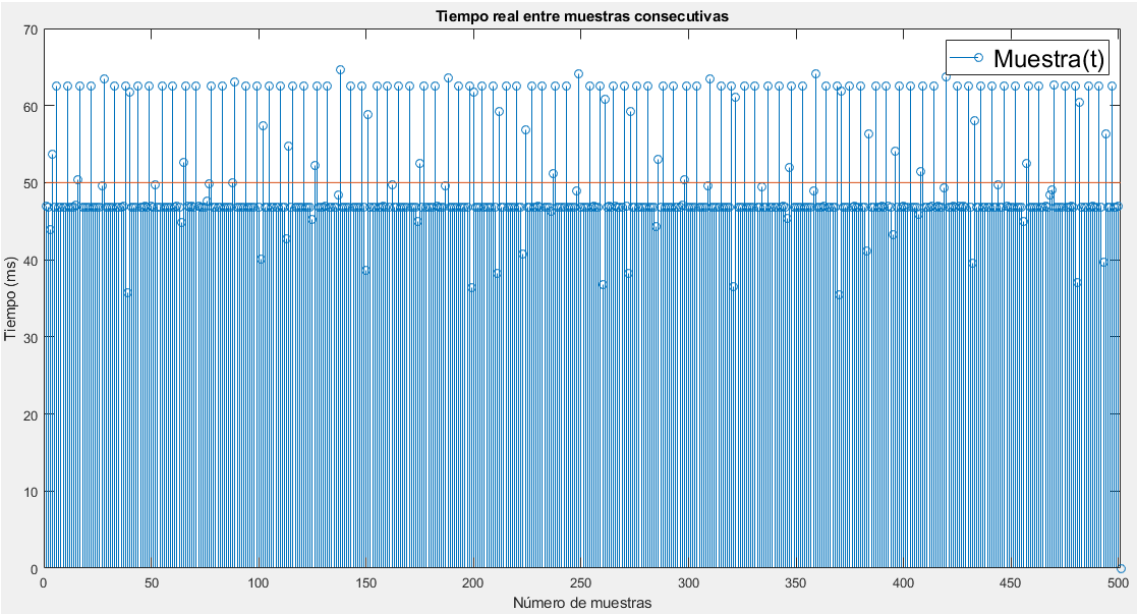


Figura 12. Tiempo entre muestras en el ejemplo de comunicación W10-W10

```
Desviación típica: 6.908380ms
Media: 49.881762ms
La ejecucion ha durado 24.990763 segundos

Se han perdido 1 paquetes
```

Este ensayo ha sido realizado con el cliente descrito en el apartado 3.1, con un rendimiento de la CPU como el que se muestra a continuación. Dicha imagen resulta interesante, ya que se trabaja sobre un sistema operativo que no garantiza el cumplimiento de los tiempos en lo que a periodo de muestreo se refiere, ya que Windows no admite modificaciones sobre su software a bajo nivel y no se ha utilizado una interfaz de aplicaciones en tiempo real como es el caso de trabajar sobre las versiones Linux en los casos 3.3.2 y 3.3.3.

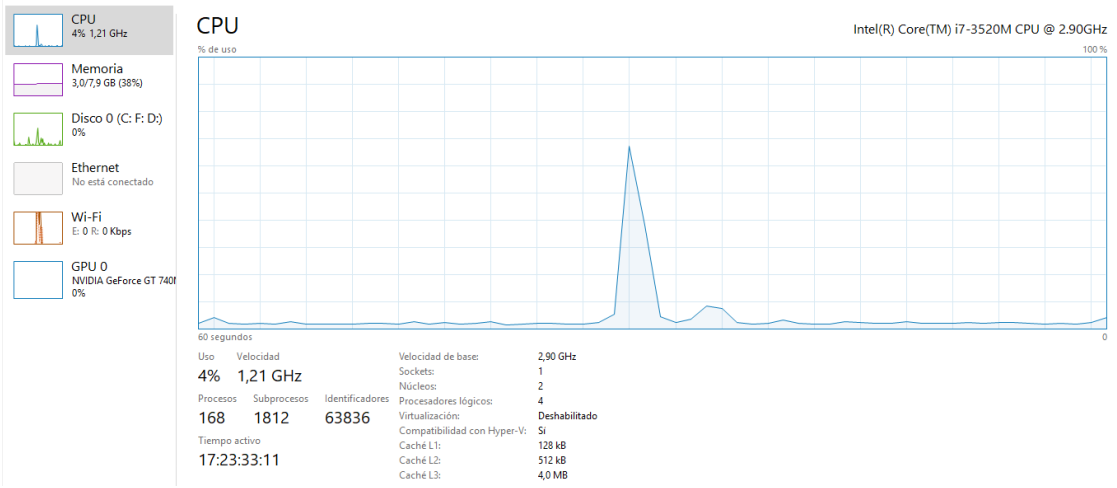


Figura 13. Rendimiento de la CPU cliente en la comunicación W10-W10

3.3.2. Driver cliente en Windows10 y servidor en Ubuntu10.04

gg

Se ha realizado un ensayo de comunicación alámbrica entre Windows10 y una versión de Linux correspondiente a Ubuntu10.04 para validar la compatibilidad que debe existir entre ambos sistemas operativos para conseguir la actualización de las herramientas para las posteriores pruebas de diseño e implementación de controladores digitales desde Matlab 2018. Para ello, se tiene como driver en la aplicación del cliente un archivo codificado en lenguaje C, de manera similar al visto en el apartado 3.3.1, cuya única modificación respecto al ahora empleado se refleja en la dirección IP a la que se va a realizar la conexión, siendo antes la IP del servidor que se soportaba sobre Windows10, y ahora hace referencia a un PC servidor dentro de la red privada con conexión cableada con el sistema operativo Ubuntu10.04 en funcionamiento.

Las aplicaciones, tanto por el lado cliente como por el lado servidor, han sido creadas en sus respectivos dispositivos. Para el ejecutable en Windows10, que va a funcionar como cliente, se han seguido los pasos indicados en la introducción del apartado 3.3, del mismo modo que se creó para el estudio realizado en el apartado 3.3.1. En cambio, para la aplicación soportada en el servidor, es decir, en Ubuntu10.04, ha sido necesaria una compilación de código y construcción del modelo de Simulink diferente a la vista hasta ahora. Dicha ruta de creación es la que se sigue en la explicada en [Salazar, 2011] donde, ayudado por la interfaz para aplicaciones en tiempo real (RTAI) se explica cómo crear un ejecutable que permita al usuario realizar estudios con aplicaciones que se ejecuten en tiempo real.

Una vez que se han creado ambos ejecutables, cada uno en su plataforma de lanzamiento, se comienza por ejecutar la aplicación del lado servidor, que quedará a la espera de datos provenientes del cliente. Ahora es momento de lanzar la aplicación cliente y esperar a que termine la ejecución.

Los resultados obtenidos se muestran a continuación, comenzando por la representación de las consignas, tanto la enviada como la obtenida:

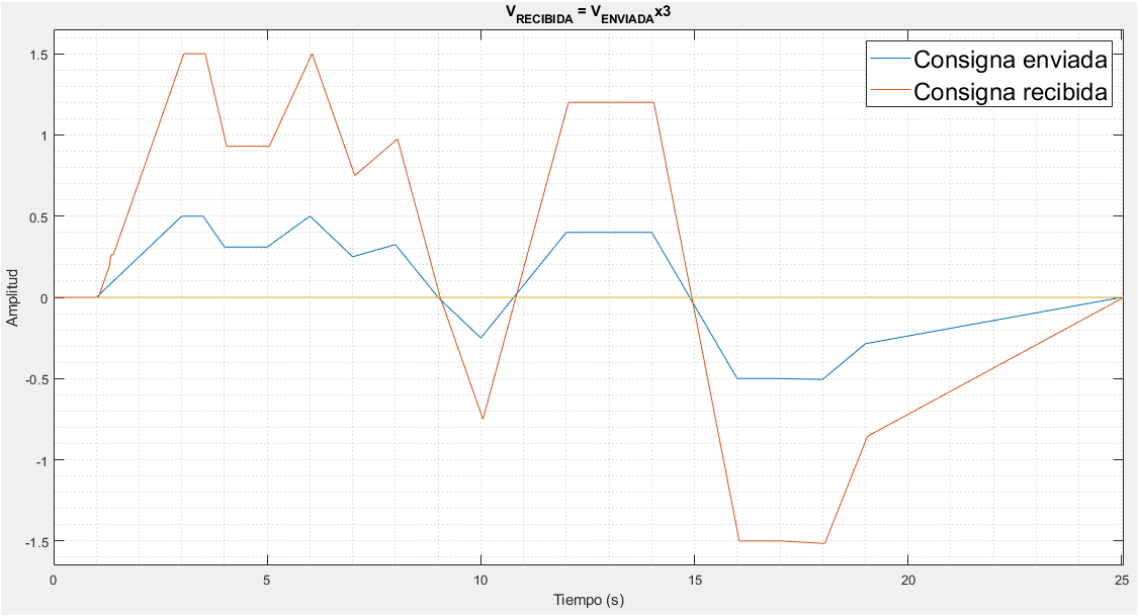


Figura 14. Consignas en la aplicación de comunicación W10-U10.04

La conexión entre ambos dispositivos ha sido alámbrica y dentro de la misma red local, por lo que se asegura una mayor fiabilidad a la hora de compartir datos de un PC a otro y haciendo que el número de paquetes perdidos sea muy bajo. Este dato se puede observar en la figura 15:

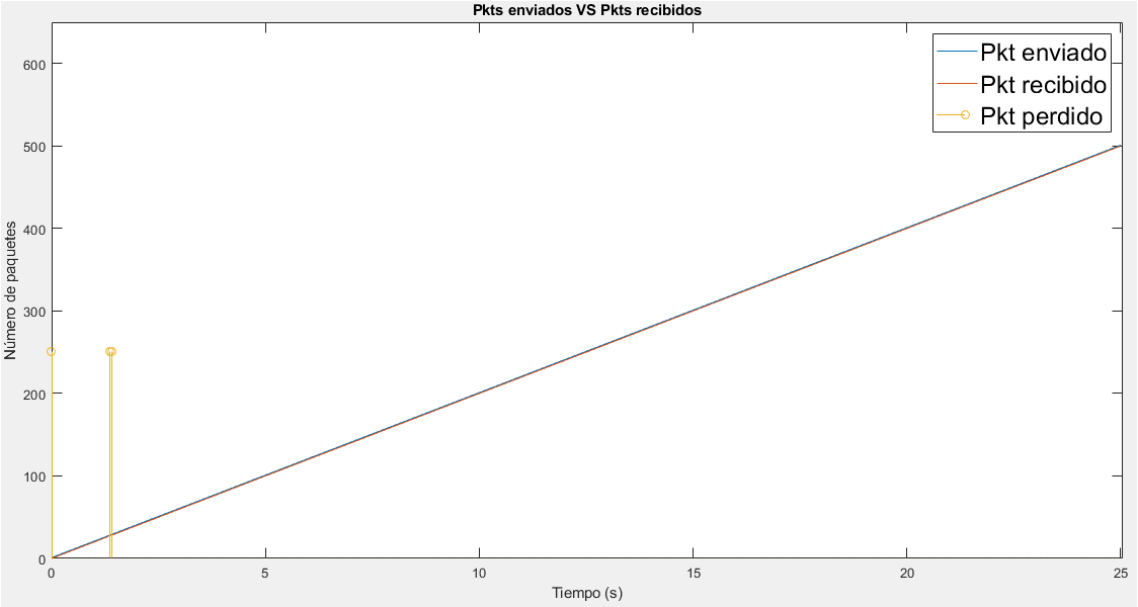


Figura 15. Paquetes enviados, recibidos y perdidos en el ejemplo de comunicación W10-U10.04

Por último, se han vuelto a obtener datos referentes al periodo de muestreo y su variación a lo largo de la ejecución de la aplicación, dando como resultado lo que se muestra en la siguiente gráfica y teniendo unos valores que rondan el valor ideal de muestreo del lado cliente, siendo este de 50ms:

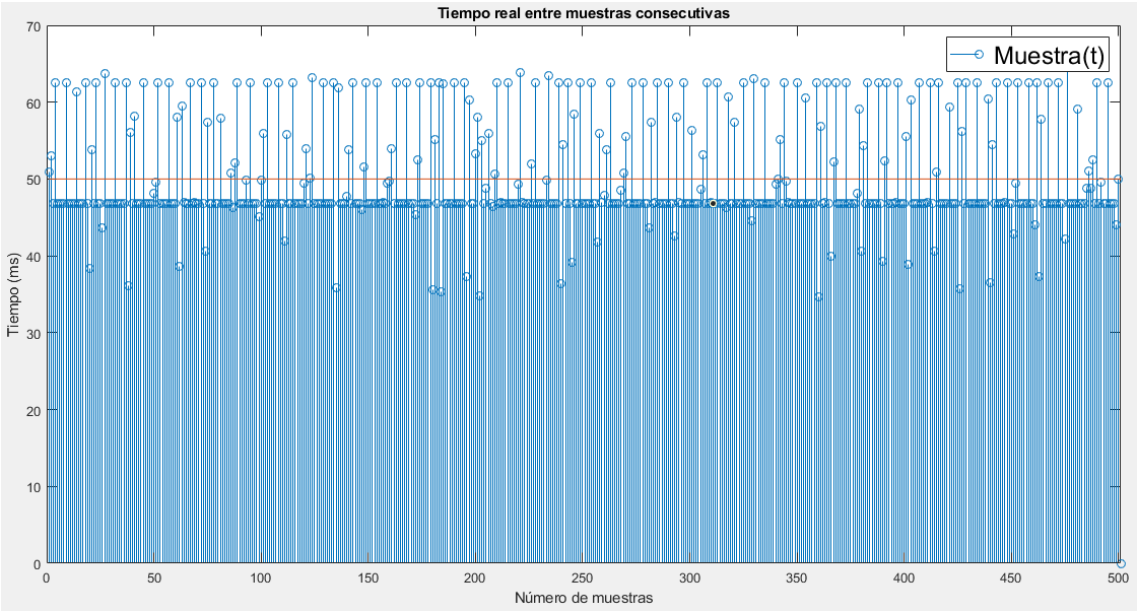


Figura 16. Tiempo entre muestras en la comunicación W10-U10.04

```
Desviación típica: 6.933710ms
Media: 49.900198ms
La ejecucion ha durado 24.999999 segundos

Se han perdido 3 paquetes
```

A continuación, para cerrar este estudio, se muestra el rendimiento que se tenía en el lado del cliente a la hora de estar ejecutando la aplicación:

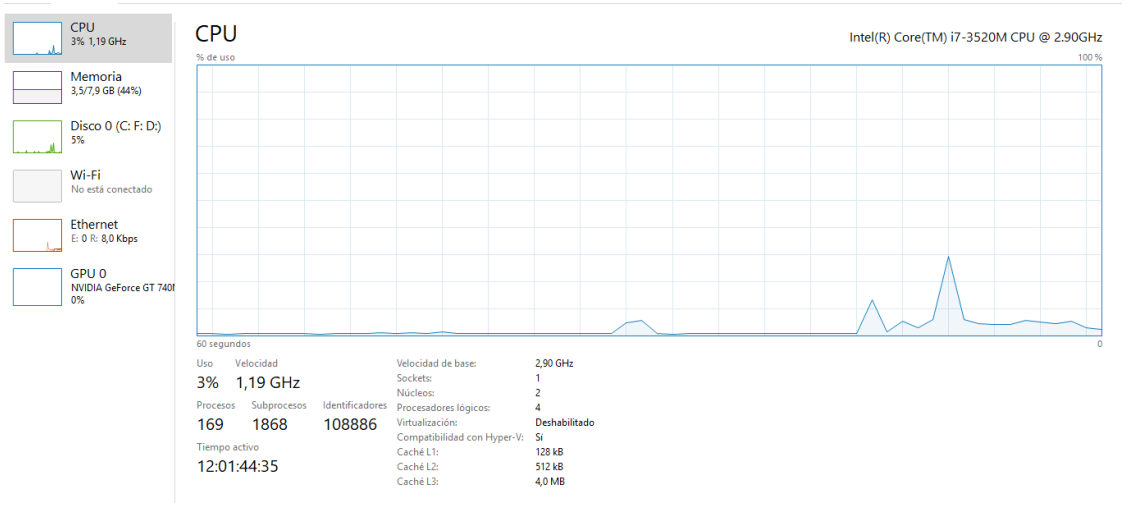


Figura 17. Rendimiento de la CPU cliente en la comunicación W10-U10.04

Como conclusión de este ensayo sobre Ubuntu10.04 se obtiene que la comunicación entre un PC soportado por Windows10 y un dispositivo que tenga instalado el sistema operativo Ubuntu10.04, es posible con un periodo de muestreo de 50ms, ya que el tiempo de ejecución real es el marcado idealmente con anterioridad, con una media de periodos de muestreos muy próximo al ideal y una desviación típica relativamente pequeña, algo superior en el caso W10-U10.04.

3.3.3. Driver cliente en Windows10 y servidor en Ubuntu12.04 sobre P3-DX

Del mismo modo que en el ensayo anterior, se ha trabajado con un servidor instaurado sobre el sistema operativo de Linux, más concretamente en la versión Ubuntu12.04. La creación del ejecutable que se es lanzado en el cliente ha sido realizada de igual manera que estudios anteriores, y para la aplicación del lado servidor ha sido necesario seguir los mismos pasos vistos en el ensayo 3.3.2, ya que se aplica el uso de la interfaz RTAI sobre Ubuntu.

El título de este apartado hace referencia a que el servidor se encuentra sobre el dispositivo P3-DX, que alberga el sistema operativo Ubuntu12.04, por lo que este ensayo lleva consigo una comunicación inalámbrica, aunque dentro de la misma red. Este hecho hace la pérdida de paquetes aumente significativamente, ya que el protocolo que se emplea, UDP, no ayuda a la garantía de que los datos lleguen correctamente al destino.

Esto hace que la consigna recibida sea algo peor que las vistas en ensayos anteriores, la pérdida de paquetes consecutivos hace que el servidor no actualice el valor recibido desde el cliente y no se modifique el valor de entrada, quedando en un estado permanente donde se mantiene el mismo valor de salida, aunque el de entrada debiera ser otro. En la siguiente figura 17 se observa la degradación de la respuesta obtenida en el cliente:

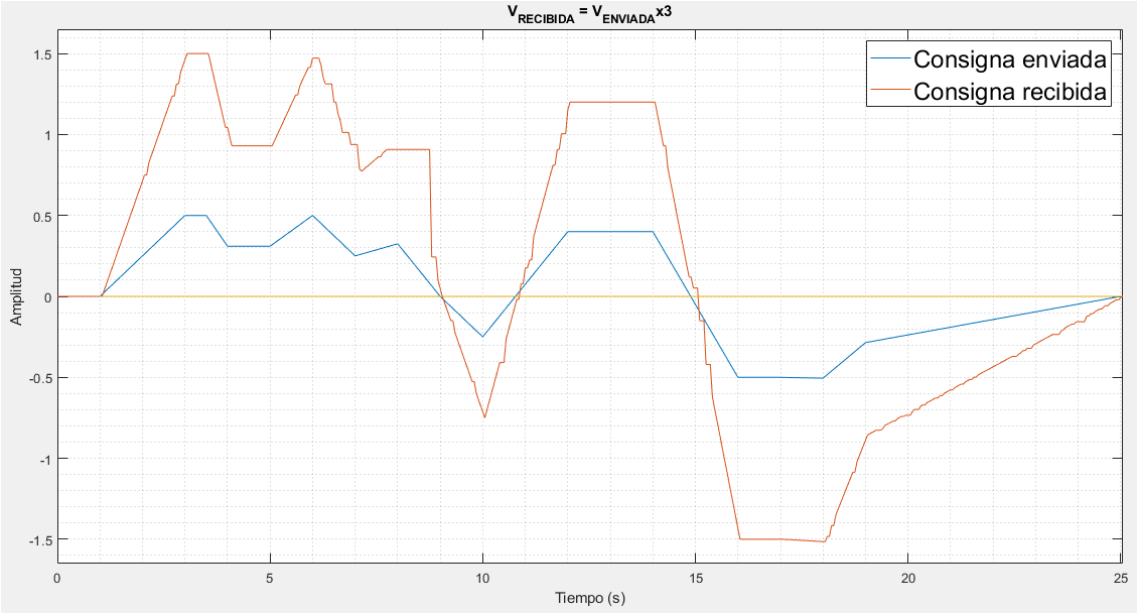


Figura 18. Consignas en el ejemplo de comunicación W10 (PC remoto cliente)-U12.04 (P3-DX servidor)

A continuación, se muestra un zoom de la figura anterior donde se puede observar con más detalle lo descrito en la introducción de este ensayo. En el margen de tiempo que comprenden los segundos del 7 al 9, se observa cómo hay un margen de tiempo donde el valor recibido en el cliente no cambia. Este periodo coincide con la pérdida masiva de paquetes durante el tiempo que dura la señal al mismo nivel mientras que la consigna de entrada sí está cambiando. En la figura 20 se reflejan la cantidad de paquetes perdidos y se puede observar cómo la franja temporal de mayor pérdida de paquetes coincide con la degradación de la respuesta mostrada en la figura 19.

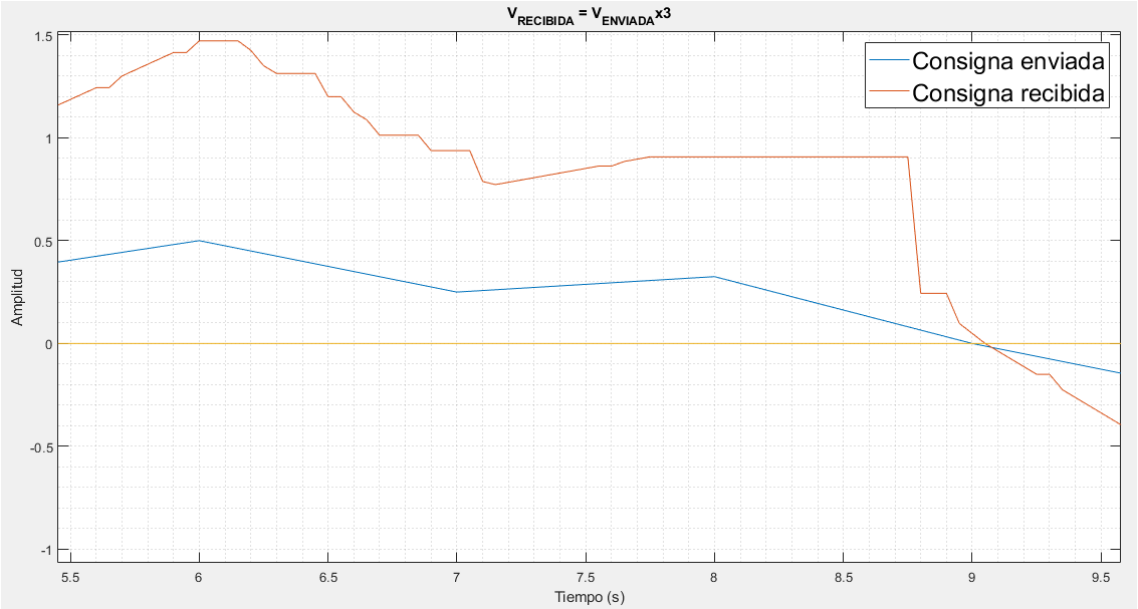


Figura 19. Zoom de las consignas en la comunicación W10 (PC remoto cliente)-U12.04 (P3-DX servidor)

Del ensayo, cabe concluir que el canal de comunicación inalámbrico contribuye a degradar la respuesta, en este caso en lazo abierto, entre el nodo de control y el nodo asociado al proceso controlado.

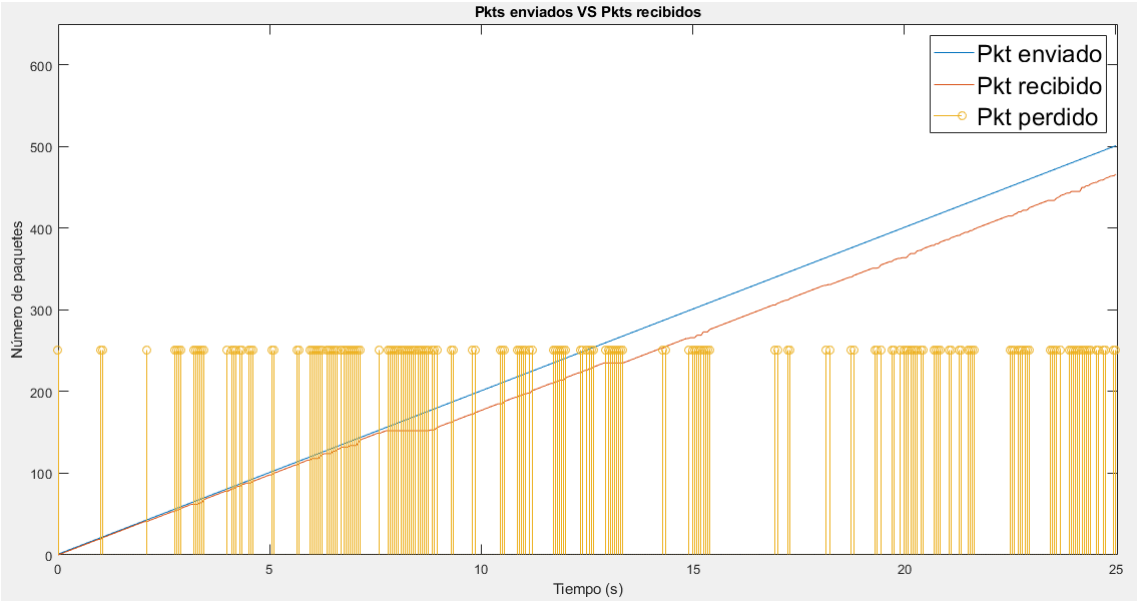


Figura 20. Paquetes enviados, recibidos y perdidos en la comunicación W10 (PC remoto cliente)-U12.04 (P3-DX servidor)

Para este estudio la media de periodos de muestreo también se cumple, aunque la desviación es mayor que en el caso de comunicación cableada (ver figura 16). En la figura 21 muestra el registro de los tiempos que han existido entre una muestra y la consecutiva.

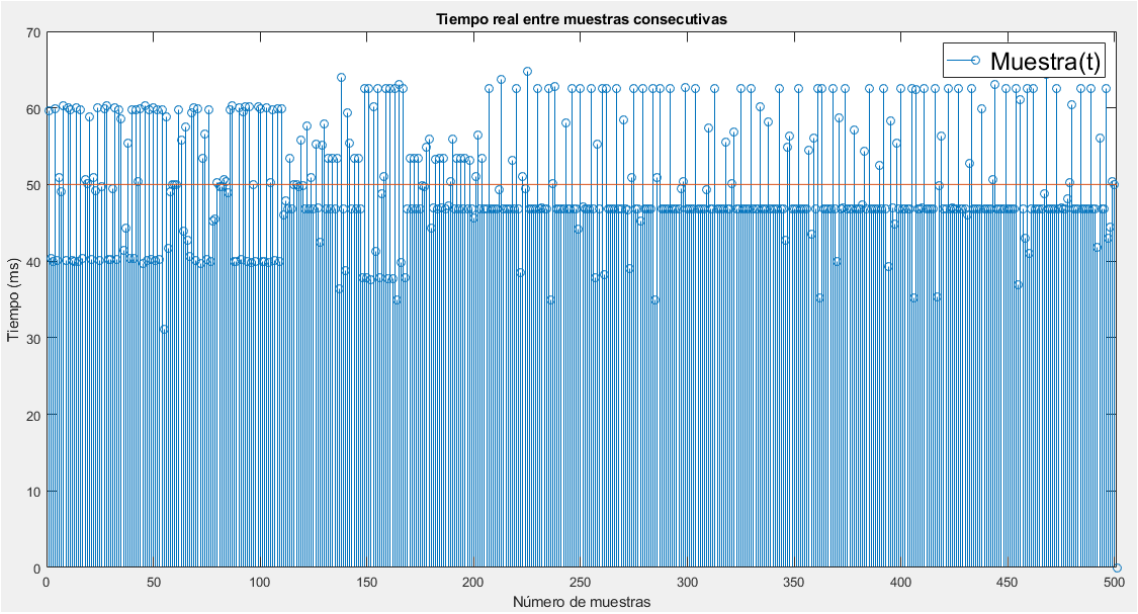


Figura 21. Tiempo entre muestras en la comunicación W10 (PC remoto cliente)-U12.04 (P3-DX servidor)

```
Desviación típica: 7.554650ms
Media: 49.880631ms
La ejecucion ha durado 24.990196 segundos

Se han perdido 188 paquetes
```

El número de paquetes perdidos asciende hasta un total de 188 para este ensayo, es decir, de las 500 muestras que ha realizado el cliente sólo han llegado al servidor y han vuelto correctamente el 62% del total. Estos valores pueden variar de un estudio a otro, ya que dependen totalmente del estado de la red en el momento en el que se está lanzando la aplicación. Por otro lado, en cuanto a tiempos de ejecución se refiere, la dinámica se mantiene teniendo una media de periodos de muestreo muy próxima a la ideal, con una pequeña desviación típica esperada y un tiempo real de ejecución total muy próximo al establecido a priori.

La tasa de ocupación y rendimiento de la CPU del cliente en el momento de estar ejecutando la aplicación es la que se muestra en la siguiente figura:

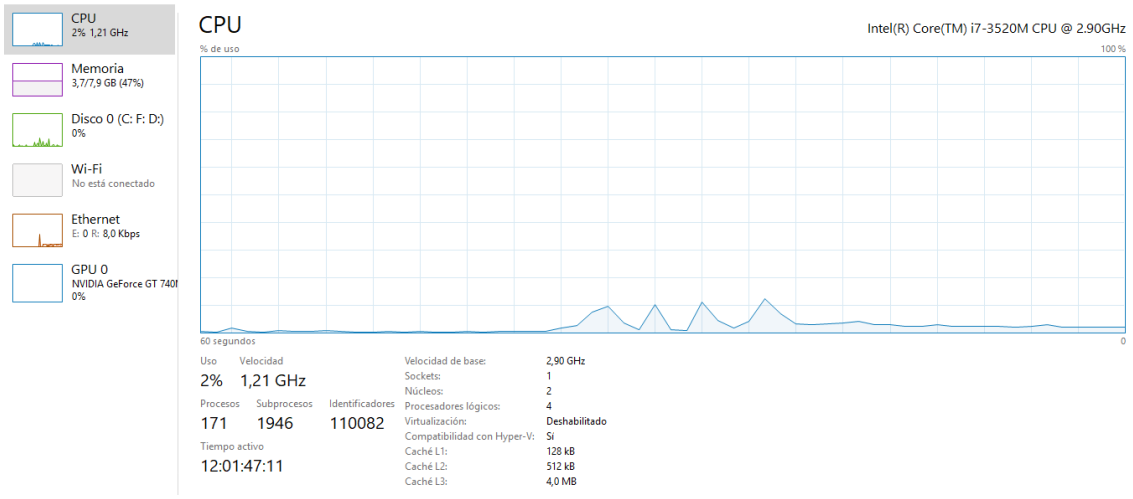


Figura 22. Rendimiento de la CPU cliente en la comunicación W10-U12 (P3-DX)

3.3.4. Driver cliente en Windows10 y servidor en Ubuntu18.04

Este ensayo trata sobre la comunicación cableada, de igual modo que en el apartado 3.3.2, entre un cliente instalado en Windows10, y un servidor soportado sobre la versión más reciente de Ubuntu hasta la fecha de hoy: Ubuntu18.04, cuya aplicación ha sido creada con Matlab2018b y sin necesidad de empleo del target RTAI. La creación de ambos ejecutables ha sido realizada siguiendo los pasos que se describen en la introducción de este apartado 3.3.

En la figura que se muestra a continuación, se observa que la consigna recibida en el cliente es correcta y no tiene estados de retención del valor de amplitud y que los pasos por 0 coinciden en ambas señales:

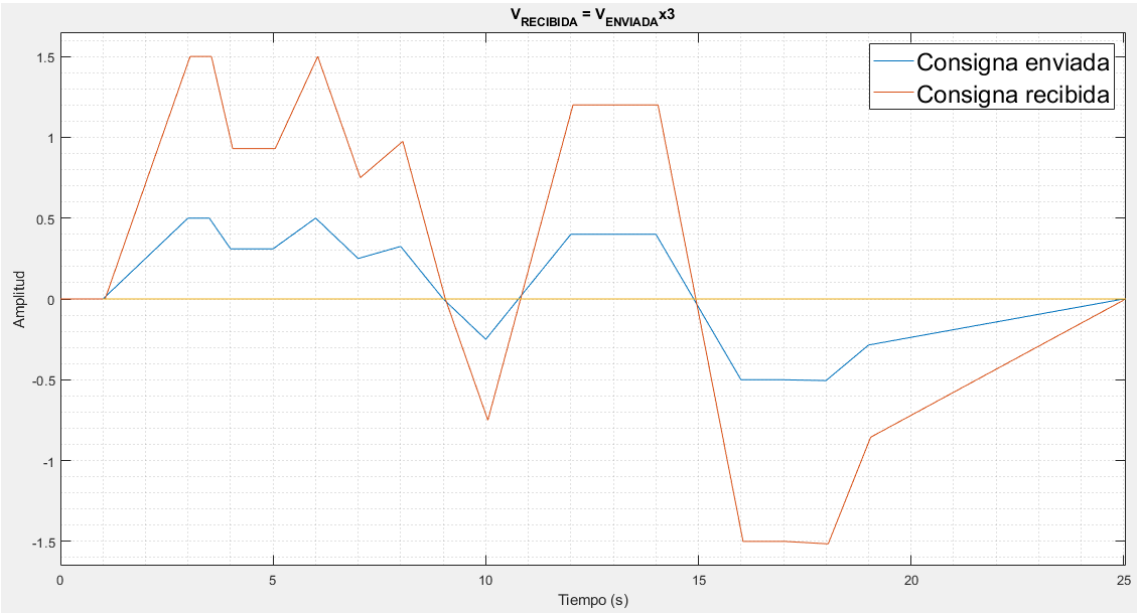


Figura 23. Consignas en la comunicación W10-U18

Por otro lado, como cabía esperar en una comunicación alámbrica, disminuye la tasa de paquetes perdidos, haciendo así que el ensayo obtenga mejores resultados. En la figura 24 se ve reflejado que durante la ejecución de la aplicación no se pierde ningún paquete.

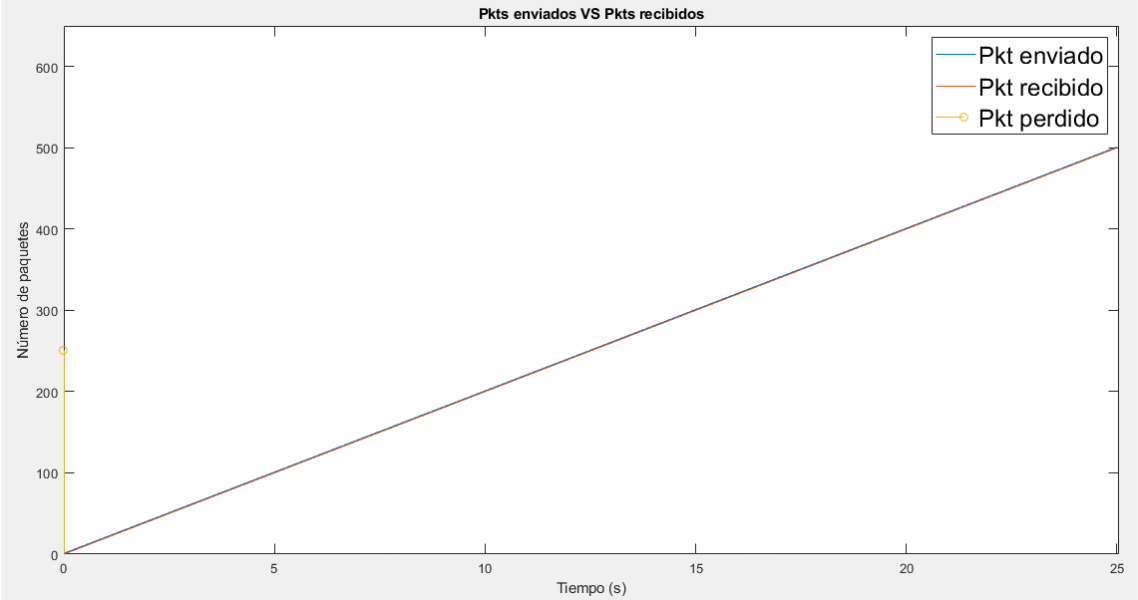


Figura 24. Paquetes enviados, recibidos y perdidos en la comunicación W10-U18

Finalmente, para concluir la valoración de este ensayo, se muestra una figura donde aparecen los tiempos que han existido entre cada una de las muestras y poder ver la variación del periodo de muestreo a lo largo de la ejecución de la aplicación. Se obtiene un valor medio de periodo muy próximo al ideal, 50ms, y con un tiempo de ejecución total real bastante próximo al establecido, 25 segundos. Por otro lado, el total de paquetes perdidos en esta ejecución ha sido de tan solo 1.

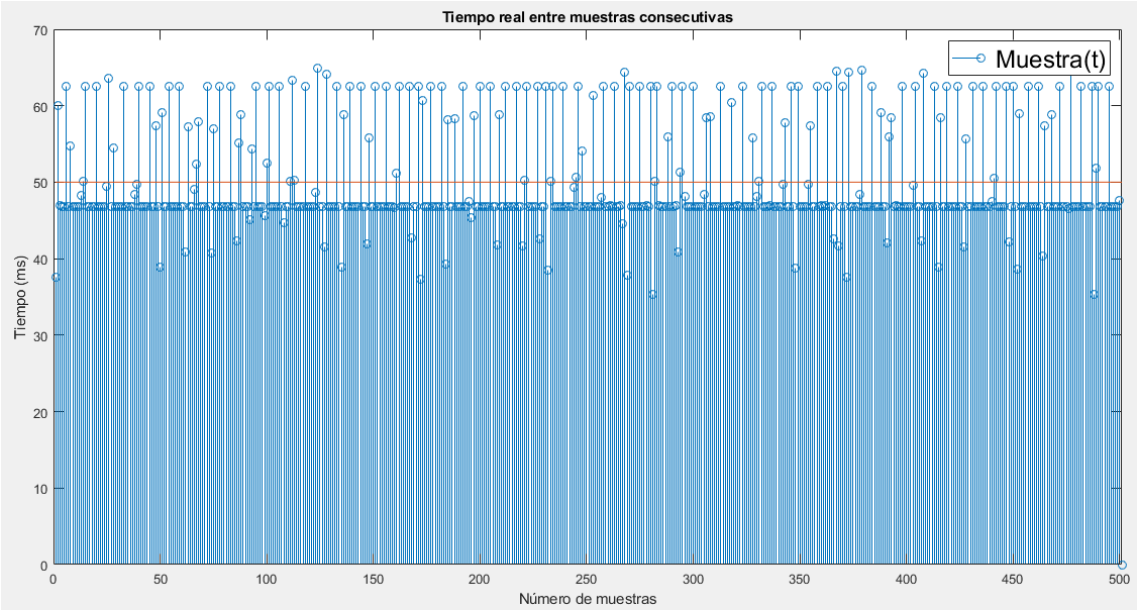


Figura 25. Tiempo entre muestras en la comunicación W10-U18

```
Desviación típica: 6.974836ms
Media: 49.875501ms
La ejecucion ha durado 24.987626 segundos

Se han perdido 1 paquetes
```

El rendimiento de la CPU donde se ejecutaba la aplicación cliente es la siguiente:

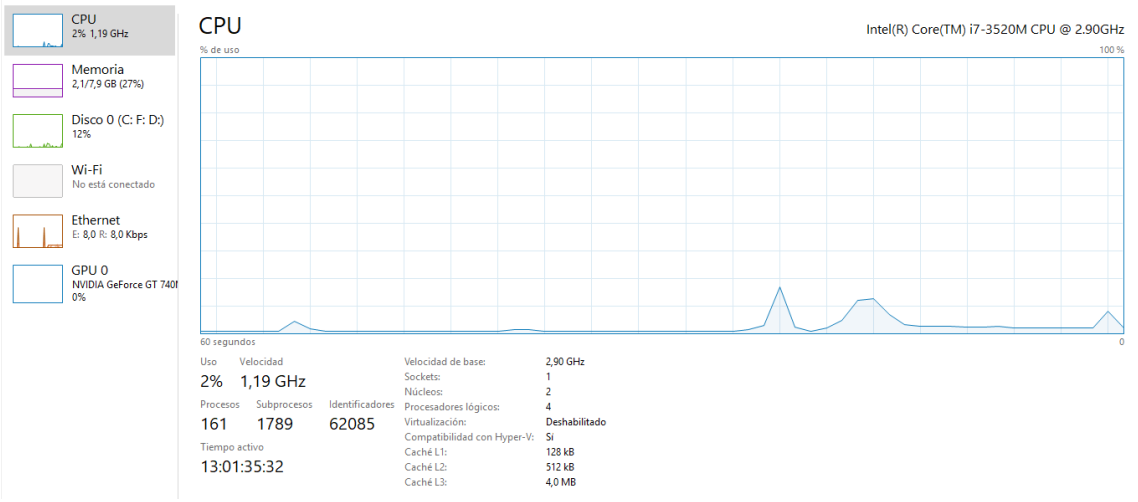


Figura 26. Rendimiento de la CPU cliente en la comunicación W10-U18

3.4. Ejemplo de seguimiento no lineal de trayectorias para robot P3-DX

Partiendo de la base adquirida en el apartado 3.3 sobre comunicación entre distintos dispositivos, ahora da comienzo el estudio final sobre el robot P3-DX con las herramientas actualizadas. En el robot se sigue trabajando con Ubuntu 12.04, pero el lado cliente ha sido creado y ejecutado sobre Windows 10 con Matlab 2018b, en el mismo PC donde se han realizado los estudios anteriores y cuyas características hardware se muestran en la figura 3. Para poner en funcionamiento al robot, es decir, a la escucha de que este reciba el primer dato desde el centro remoto y pueda comenzar a operar, se ha creado un ejecutable que está incorporado en su memoria denominado: “cove5_2019”. Para todos los ejemplos que se proponen de aquí en adelante en este documento es necesario lanzar este ejecutable en el robot. Los pasos a seguir para lanzar dicha aplicación desde el centro remoto son:

- Encender la unidad COVE 4, tanto el robot como la CPU que incorpora.
- Abrir un terminal desde un centro remoto conectado en la misma red que el robot

- Acceder al robot mediante el comando: 'sudo ssh cove5'; o en caso de no tener la etiqueta de la IP descrita: 'sudo ssh 192.168.11.61'.
- Una vez dentro del robot, dirigirse al directorio donde se encuentran los ejecutables: 'cd /home/cove5/target/'
- Mediante el comando 'ls -l' se muestra un listado de todos los ejecutables que incorpora esta unidad. Ya estaría listo para ejecutar el que interesa en este trabajo mediante la instrucción: './cove5_2019 -v -f XX" (donde XX es un número que indica el tiempo de ejecución total que se desea realizar en segundos).

Primero se ha realizado un servosistema en el centro remoto para el seguimiento lineal de velocidades al introducir al robot unas consignas establecidas a través de un driver creado expresamente para el sistema operativo Windows 10. Más tarde, se procede al estudio del control remoto para seguimiento no lineal de trayectorias basado en Lyapunov.

Ambos estudios se han realizado en la zona oeste de la primera planta de la Escuela Politécnica Superior de la Universidad de Alcalá como se muestra en la figura 27 En el estudio del servosistema se tiene por origen el laboratorio OL3 y como destino el laboratorio OL2, y en el segundo estudio se trabaja en el pasillo que comunica los laboratorios.

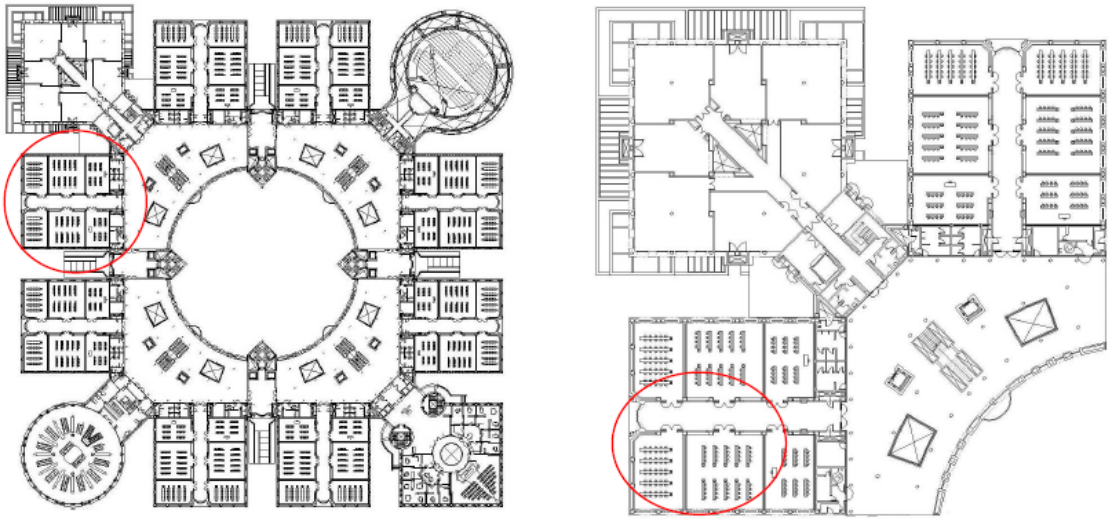


Figura 27. Plano de la primera planta del edificio politécnico UAH

3.4.1. Control remoto para seguimiento lineal de velocidades

El objetivo es comunicar dos dispositivos a partir de drivers creados con Simulink y ejecutados en tiempo real, uno de ellos en Ubuntu12.04 (servidor/robot) y el otro soportado sobre Windows10 (cliente). El driver cliente es el que alberga modificaciones respecto a las versiones utilizadas años atrás, por lo que únicamente se ve este reflejado en el Anexo II.

El ensayo consiste en ejecutar en el PC remoto (cliente) un servosistema [Ogata, 1996], de forma que permita comparar los resultados de simulación (servo aplicado a modelo en variables de estado del robot) y los de ejecución en tiempo real (servo aplicado a robot real comunicado de forma inalámbrica), simultáneamente, ver figura 28.

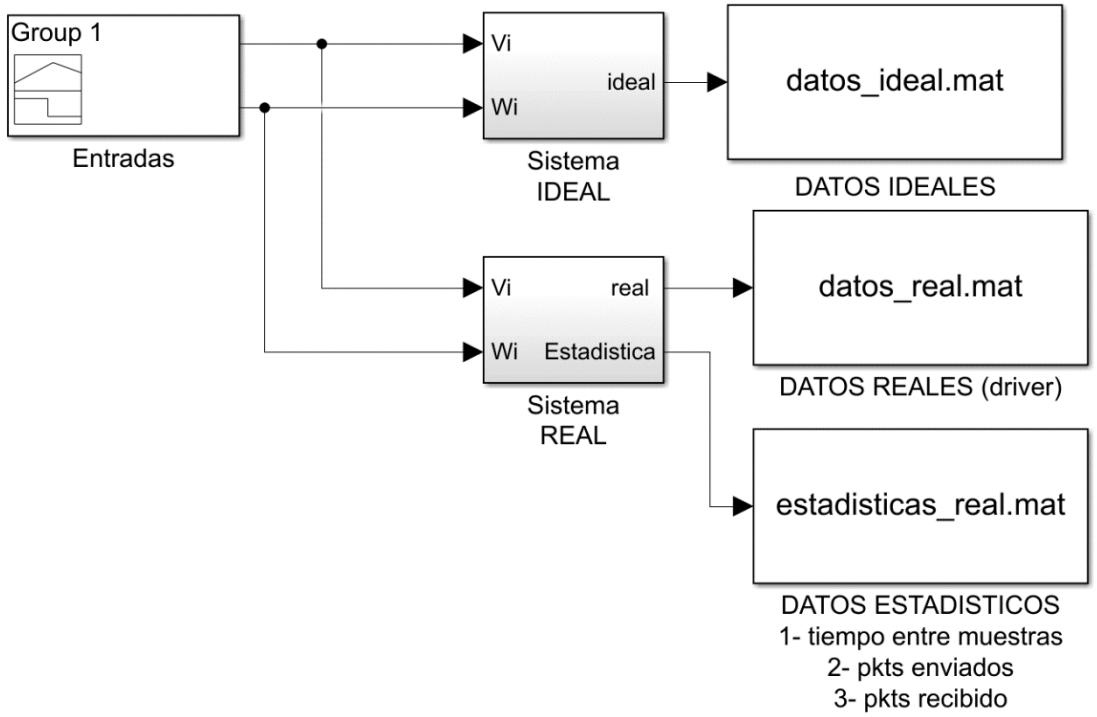


Figura 28. Modelo de Simulink para el estudio del servosistema

Las referencias de velocidad lineal y angular a controlar se diseñan para conseguir una trayectoria no lineal. Estas referencias se aplican al servosistema ejecutado en el centro remoto (PC cliente). El servosistema genera las señales de actuación sobre la planta, modelada en simulación (formando parte del sistema ideal en la figura 28 y robot real (formando parte del sistema real en la figura 28) en experimentación. Las consignas se envían al robot en paquetes UDP de 12 Bytes de tamaño en el bloque definido como "Sistema REAL" y cuya estructura se muestra en la figura 29. Con las señales de actuación de velocidad lineal y angular recibidas, el robot establece las velocidades de giro de los motores, lee los encoders y devuelve al cliente la información de velocidad lineal y angular obtenida nuevamente en paquetes UDP de 12 Bytes. Por otro lado, en el bloque "Sistema IDEAL", se tiene una planta que modela el comportamiento del robot en variables de estado, las cuales han sido obtenidas por identificación del modelo del robot. Dicho modelo que incluye la planta ideal se muestra en la figura 30.

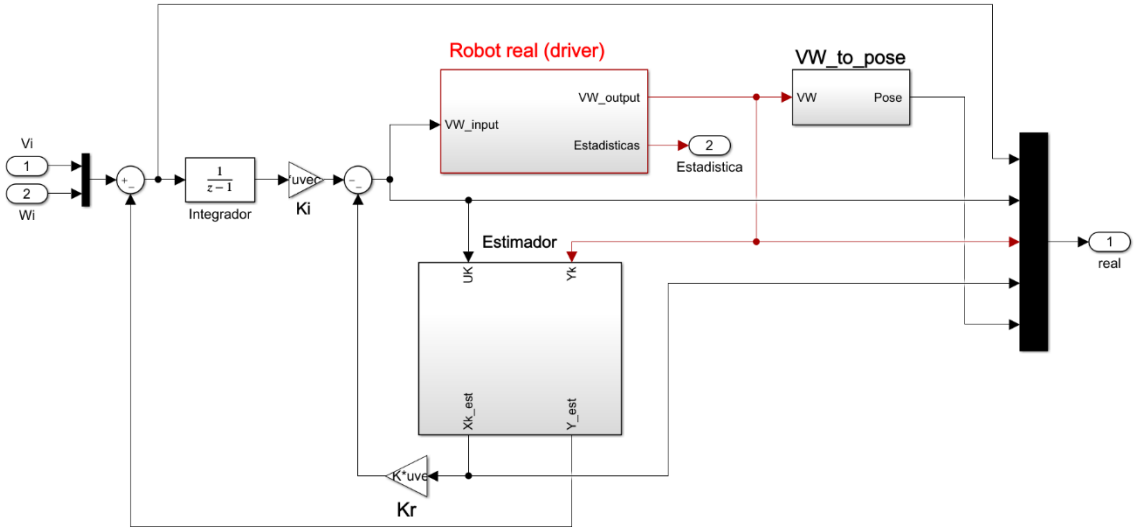


Figura 29. Estructura del "Sistema REAL"

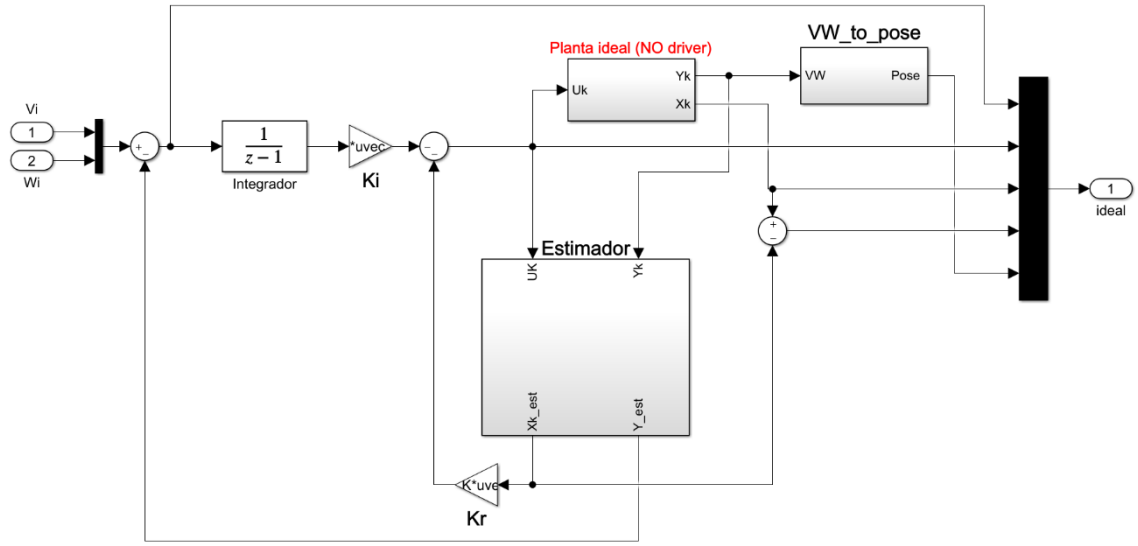


Figura 30. Estructura del "Sistema IDEAL"

La implementación del driver se realiza mediante una s-function como se muestra en la figura 31. A parte de las consignas recibidas por parte del robot, también se han obtenido datos estadísticos, como el tiempo total de ejecución, el tiempo entre cada una de las muestras, y los paquetes enviados y recibidos.

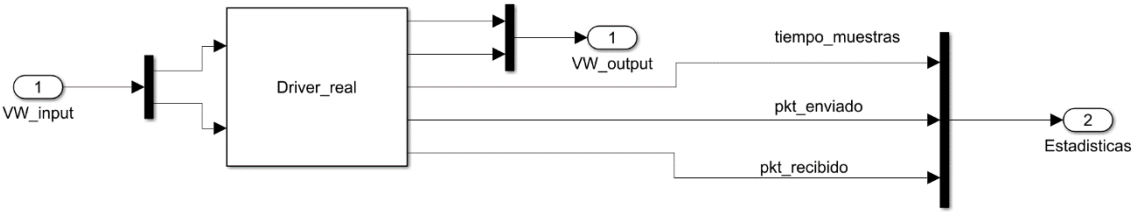


Figura 31. Planta real. Driver implementado sobre una s-function

Una vez obtenido el ejecutable mediante Windows10, con la versión 2018b del software de Matlab, se lanza la aplicación que inicia el robot y este queda a la espera de que se ejecute el cliente y comience a mandar datos de velocidades. Acto seguido se inicia la aplicación cliente creada y se espera el tiempo de ejecución designado, en este caso 25 segundos.

A continuación, se muestran varias gráficas sobre el comportamiento que se ha obtenido , al igual que en el estudio del apartado 3.3.3. Las figuras representan diversas señales de interés, como son la relación entre las consignas ideales y las que realmente actúan sobre el robot, la comparación entre la trayectoria inicialmente establecida y la realmente realizada, o incluso la representación de los tiempos que existen entre una muestra y la siguiente, etc.

La figura 32 muestra la salida de la planta, tanto la obtenida por la planta ideal (modelo en VVEE) , como las registradas en el robot. Se observa cómo la desviación entre las consignas ideales y las obtenidas por el robot son pequeñas, quedando así validado el estudio.

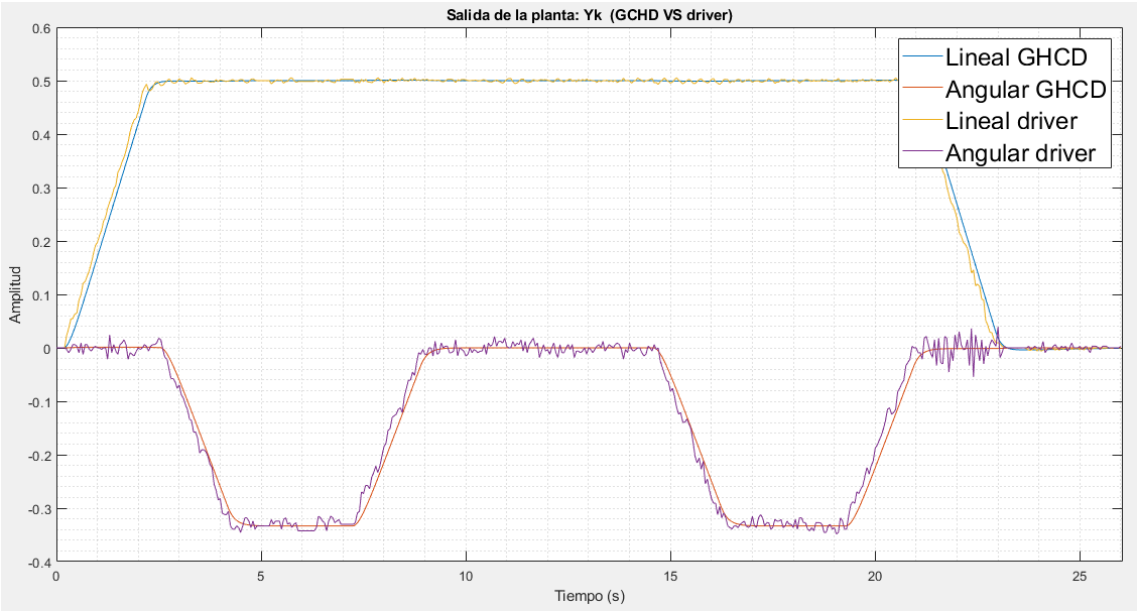


Figura 32. Salidas de la planta: ideal VS real

En la siguiente figura se muestran las trayectorias obtenidas, tanto la ideal que se obtiene del modelo matemático como la proporcionada por la odometría (encoders) del robot:

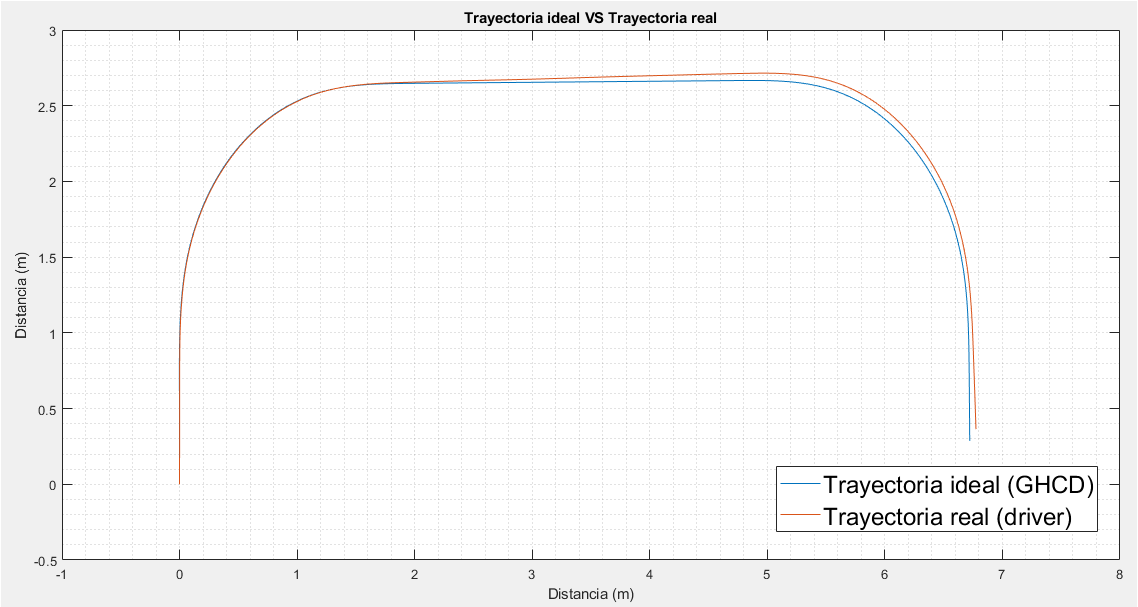


Figura 33. Trayectoria: ideal VS real

La comunicación de este estudio ha sido inalámbrica, por lo que la tasa de paquetes perdidos aumenta respecto a una conexión por cable. En la figura 34 se muestra la gráfica con dicha tasa, junto con una representación de los paquetes enviados por parte del centro remoto y los recibidos desde el robot:

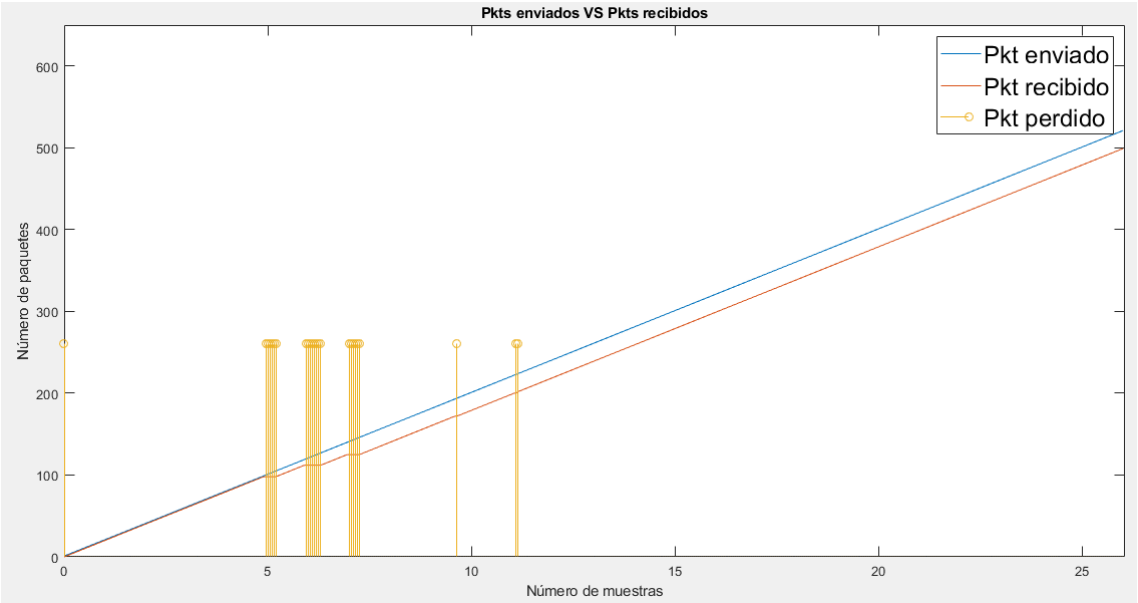


Figura 34. Paquetes enviados, recibidos y perdidos en la comunicación Windows10-P3DX

Finalmente, esta última gráfica muestra el tiempo existente entre muestras. Vuelve a existir la compensación que realiza el sistema operativo para que la media final sea la más aproximada posible al periodo de muestreo establecido a priori.

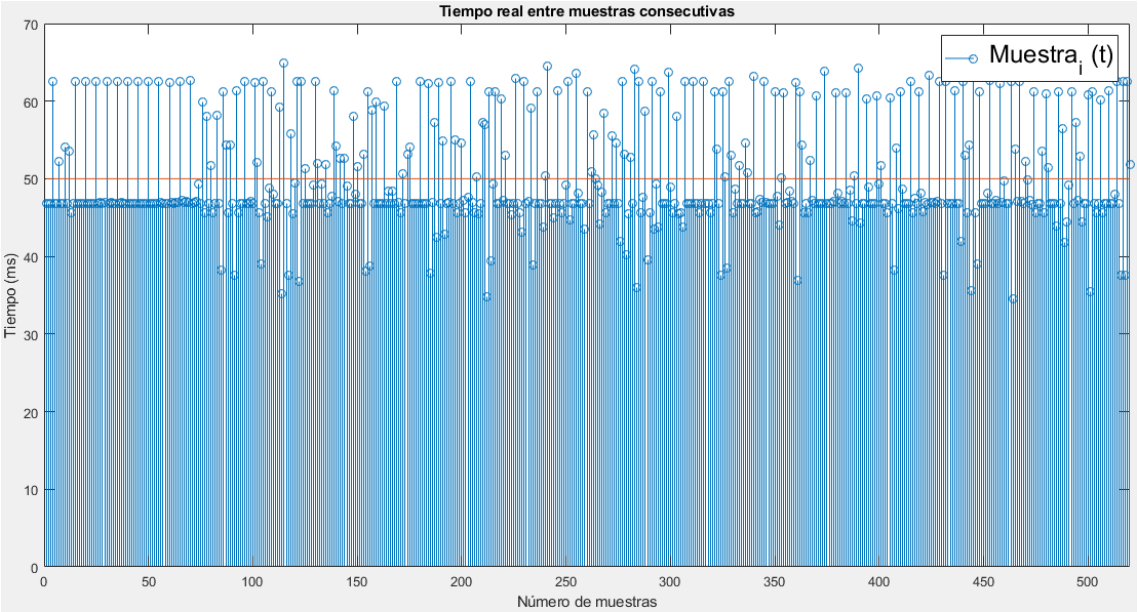


Figura 35. Tiempo entre muestras consecutivas

```
Desviación típica: 7.077705ms
Media: 49.907653ms
La ejecucion ha durado 26.001887 segundos en vez 26 segundos
```

3.4.2. Control remoto para seguimiento no lineal de trayectorias

El seguimiento no lineal de trayectorias de un robot requiere realimentar la pose de este, lo que supone contar con la componente cinemática del robot y, con ello, las no linealidades asociadas. Lyapunov desarrolló una teoría para el estudio de estabilidad de sistemas no lineales y el diseño de controladores estables para procesos no lineales [Slotine, 1991], [Antsaklis, 2007], [Malisoff, 2009].

El modelo 2D de la cinemática de un robot está basado en el modelo uniciclo de la figura 36 y responde a las ecuaciones:

$$\dot{x} = v\,cos\theta$$

$$\dot{y} = v\,sin\theta$$

$$\dot{\theta} = w$$

Siendo [x, y, θ] las coordenadas cartesianas que definen la pose del robot, y [v, w] las velocidades aplicadas.

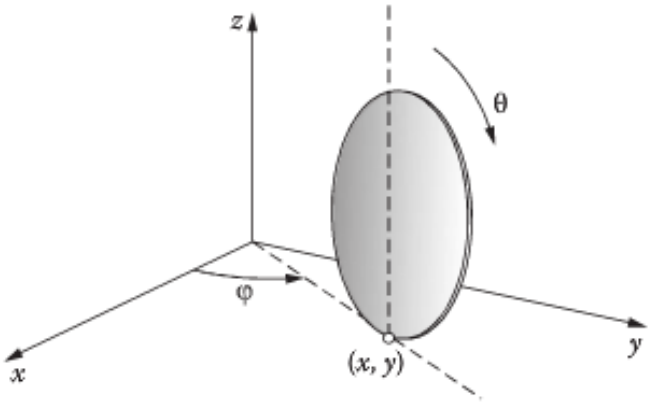


Figura 36. Modelo de una rueda rodando en vertical sobre un plano

Dado un sistema no lineal, descrito por la dinámica del estado $x \epsilon \mathbb{R}^n$, $y = f(x)$, el método directo de Lyapunov establece, básicamente, que el sistema es estable en el origen x=0, si existe una función escalar del estado x: V(x) tal que cumple :

1.- La función de Lyapunov es positiva: $V(x) > 0$,

2.- La derivada de la función de Lyapunov es negativa o nula $\dot{V}(x) \leq 0$.

Con este soporte de estudio de estabilidad, hay diferentes referencias bibliográficas para el diseño de sistemas de seguimiento (tracking) de trayectorias no lineales. En este apartado del TFG se ha implementado una solución basada en la propuesta por Amoozgar [Amoozgar, 2012], pero teniendo en cuenta no solo la cinemática del robot, sino su dinámica y servosistema incorporado.

Las variables de interés utilizadas en el estudio se muestran en la figura 37, donde el robot con coordenadas [x, y, θ], ha de seguir a la referencia [x_r, y_r, θ_r] cambiante con el tiempo. La función de Lyapunov utilizada es:

$$V = \frac{1}{2}d^2 + 1 - cos(\theta_r - \theta) = \frac{1}{2}d^2 + 1 - cos(e_\theta)$$

Y la ley de control no lineal basado en Lyapunov (Lyapunov Based Controller) para ambas velocidades:

$$v = v_{ls} \cos(\alpha) + v_r \, cos(e_\theta)$$

$$w = \dot{\theta}_{md} + v_{md}[\, K_w(v_{ls} \sin(\alpha) + v_r \sin(e_\theta)) + d \sin(\alpha)\,]$$

Siendo

$$v_{ls} = K_v \, d$$

$$\theta_{md} = atan2\left(\frac{v_{ls}\sin(\alpha - e_\theta)}{V_r + V_{ls}\cos(\alpha - e_\theta)}\right) + \theta_r$$

$$v_{md} = \sqrt{{v_r}^2 + (K_v d)^2 + 2 v_r K_v d \cos(\alpha - e_\theta)}$$

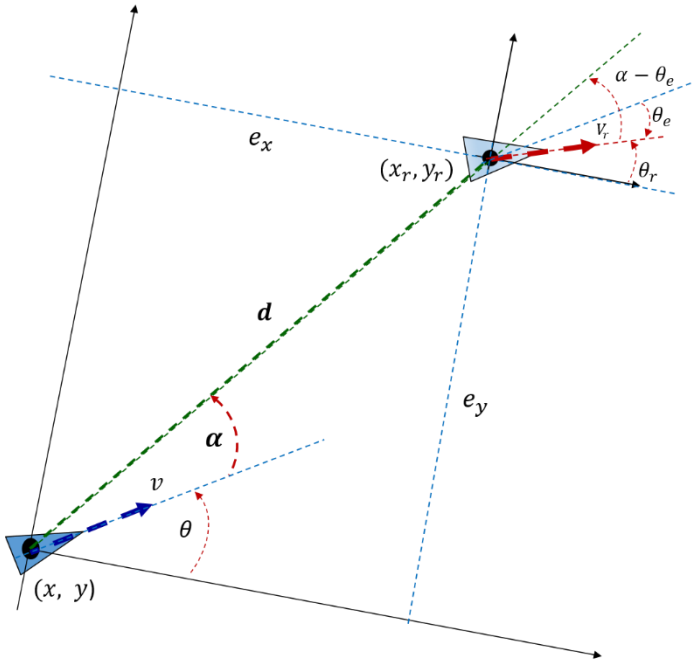


Figura 37. Variables implicadas en el seguimiento de trayectorias no lineales [Amoozgar, 2012]

A continuación, se muestran los diagramas de bloques de los proyectos realizados en Simulink previos a la generación de código y ejecución en tiempo real; así como la comparación entre los resultados simulados y obtenidos experimentalmente. En el primer estudio se trata de un control sin realimentación de los datos captados por la cámara, pero sí han sido recogidos para una mejor comprensión. Como segunda solución al estudio se tiene un modelo que realimenta en el control de trayectoria dichos datos captados por la cámara Kinect 2.0 de Microsoft, la cual se muestra en la siguiente figura 38:

Figura 38. Cámara Kinect 2.0 utilizada para registro de la pose del robot

En la figura 39 se muestra el diseño de la primera solución, en la que para el control LBC se realimenta la pose obtenida por odometría a partir de los encoders del propio robot.

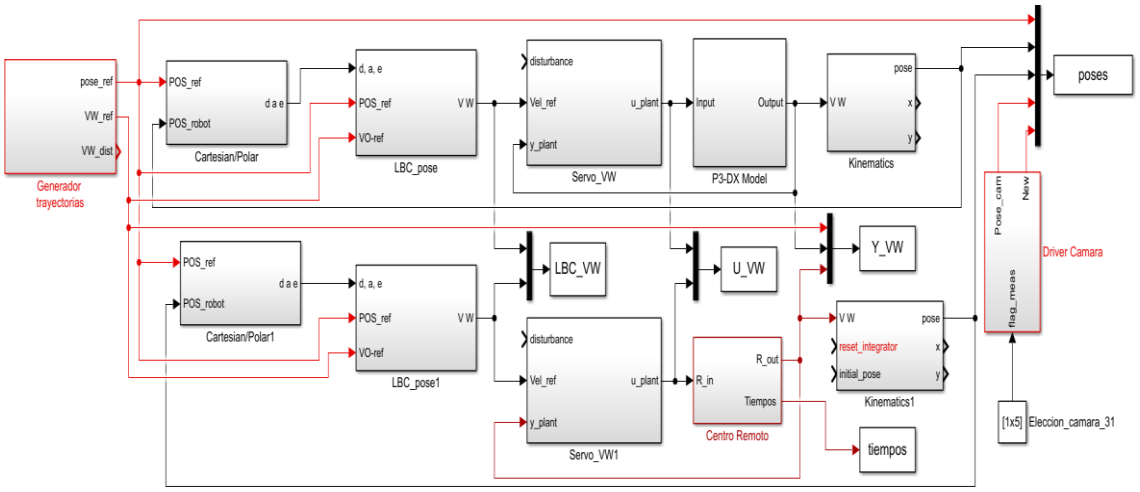


Figura 39. Modelo de solución LBC sin realimentación de cámara

En este caso, la realimentación de la pose se obtiene a partir de la odometría del robot. Téngase en cuenta que la pose es estimada, pues los encoders proporcionan medida directa de la velocidad angular de cada rueda. Con todo, se ha registrado la pose real del robot mediante una cámara Kinect ubicada en la zona de ensayo.

En la figura 40, se compara la trayectoria resultante de la planta simulada y la registrada por la odometría del robot, siendo ambas comparadas con la trayectoria de referencia:

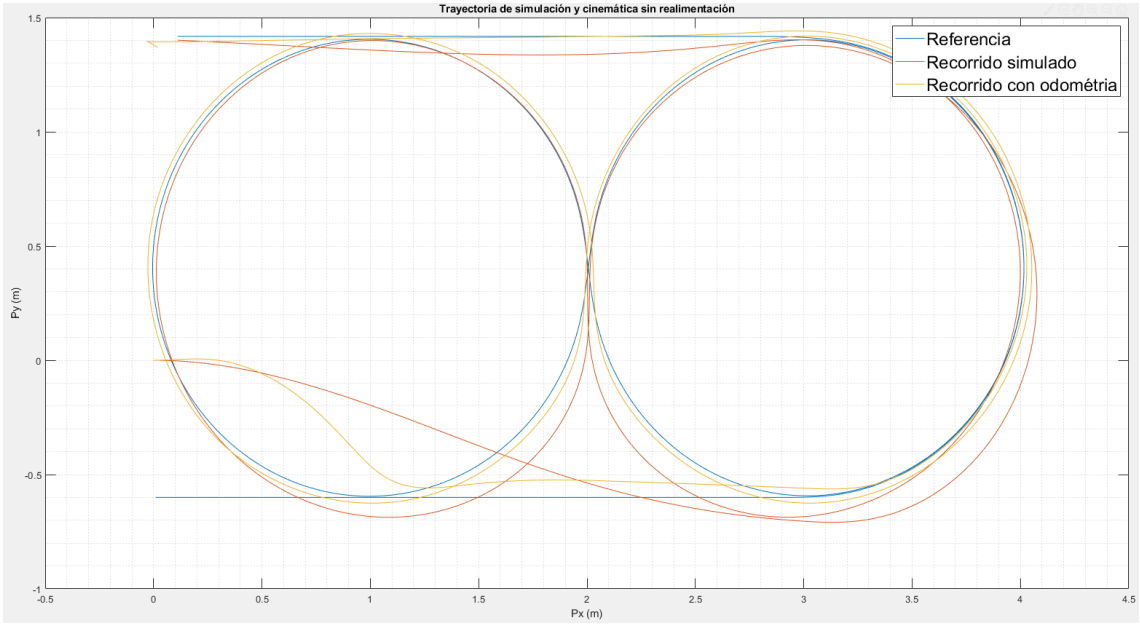


Figura 40. Trayectoria simulada y cinemática sin realimentación

Por otro lado, en la figura 41, se muestra la trayectoria de referencia, la obtenida por la realimentación odométrica y la realmente realizada y captada por la cámara.

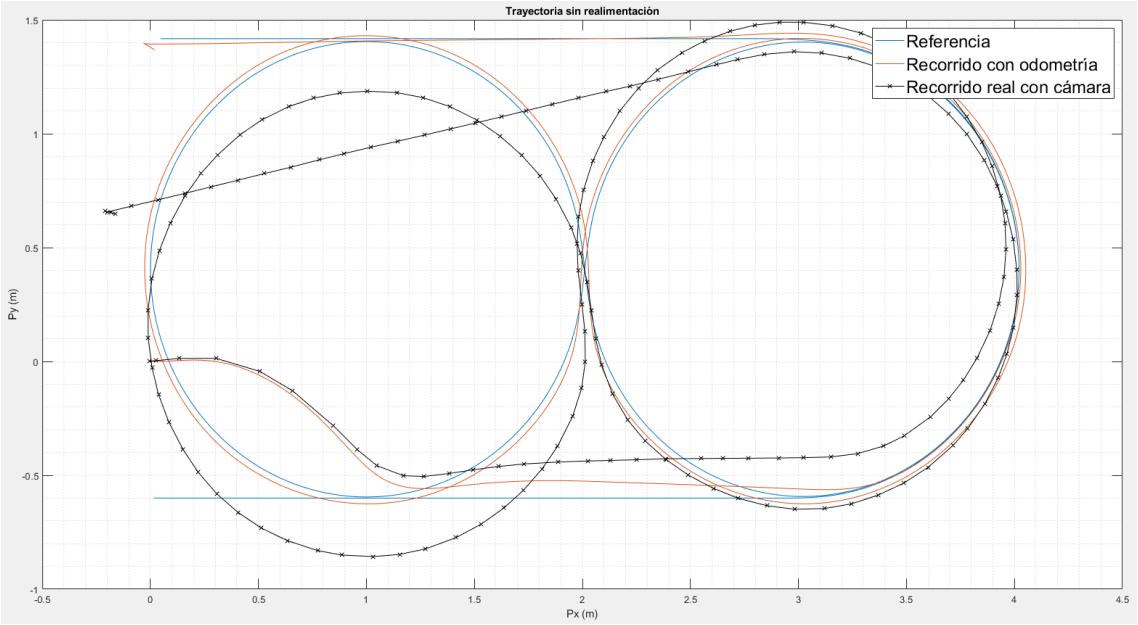


Figura 41. Comparación de trayectorias sin realimentación de la cámara

Se observa cómo el recorrido trazado con línea roja y negra no coinciden, siendo la segunda más próxima a la realmente realizada por el robot. Dado que no hay realimentación de la cámara sino de la obtenida indirectamente (modelo cinemático) a partir de la información de velocidad lineal y angular proporcionada por la odometría. y cometiendo un error notable. Es un ejemplo claro de que esta solución no es válida para el seguimiento de trayectorias no lineales.

En la siguiente figura se muestra la diferencia existente entre la velocidad lineal y angular resultante tras el control LBC realizado en simulación y la realmente obtenida en el ensayo real:

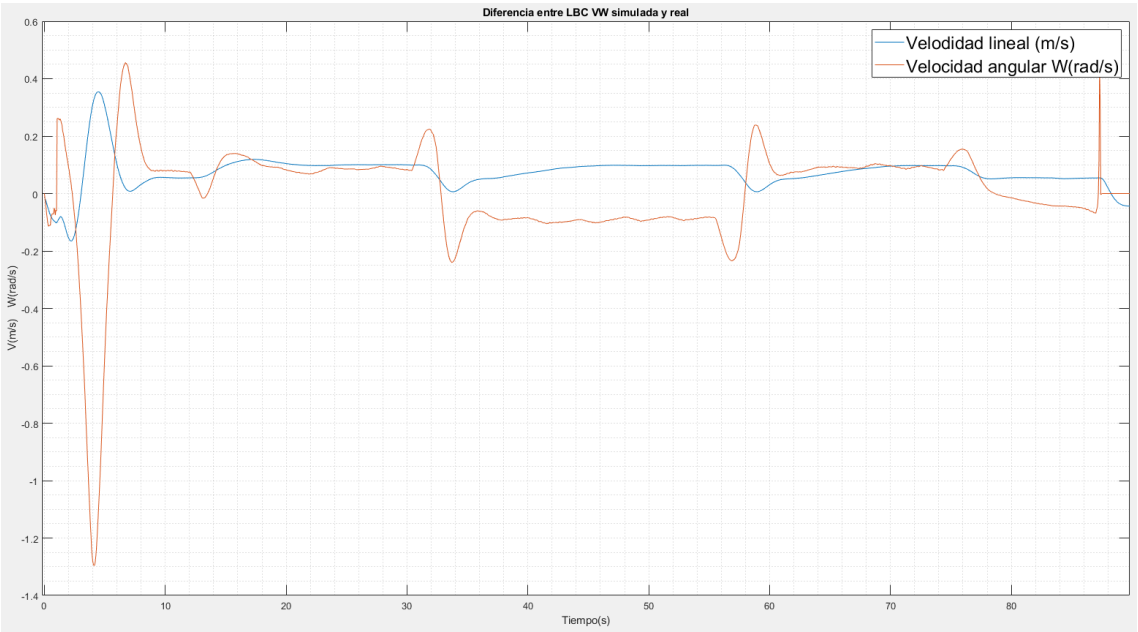


Figura 42. Diferencia tras control LBC en simulación e implementación

La figura 43 representa la diferencia resultante entre la entrada a la planta simulada y la entrada a la planta real (consignas de entrada al robot):

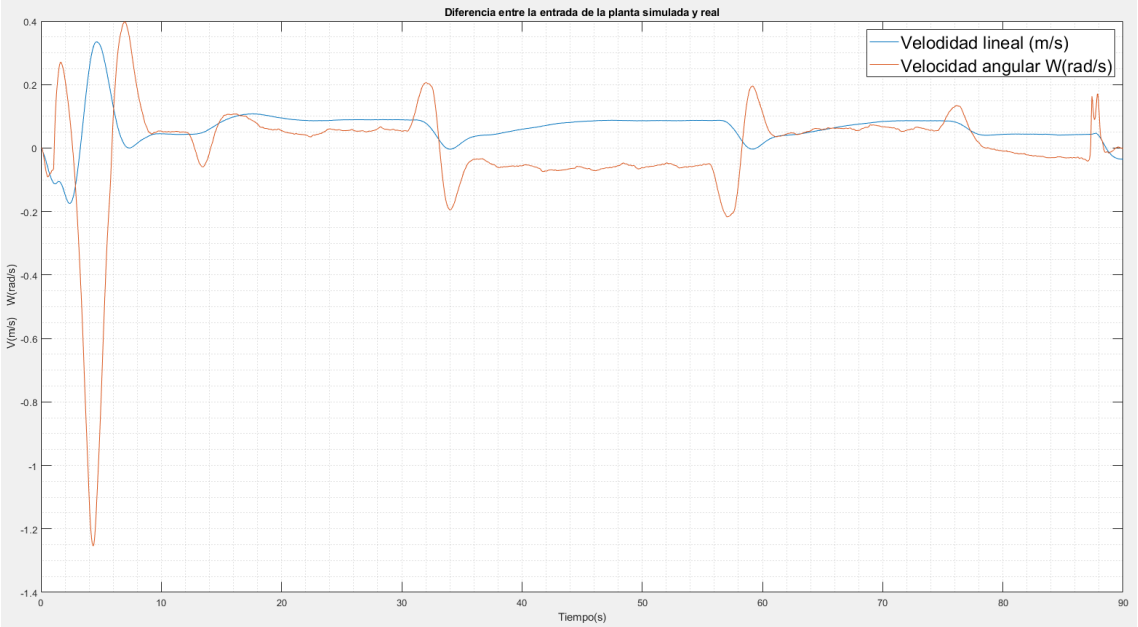


Figura 43. Diferencia de consignas en la entrada a la planta simulada y real

En las siguientes figuras se compara el error en posición (x, y) y en orientación (θ) resultantes de la ejecución simulada (figura 44) y experimental (figura 45):

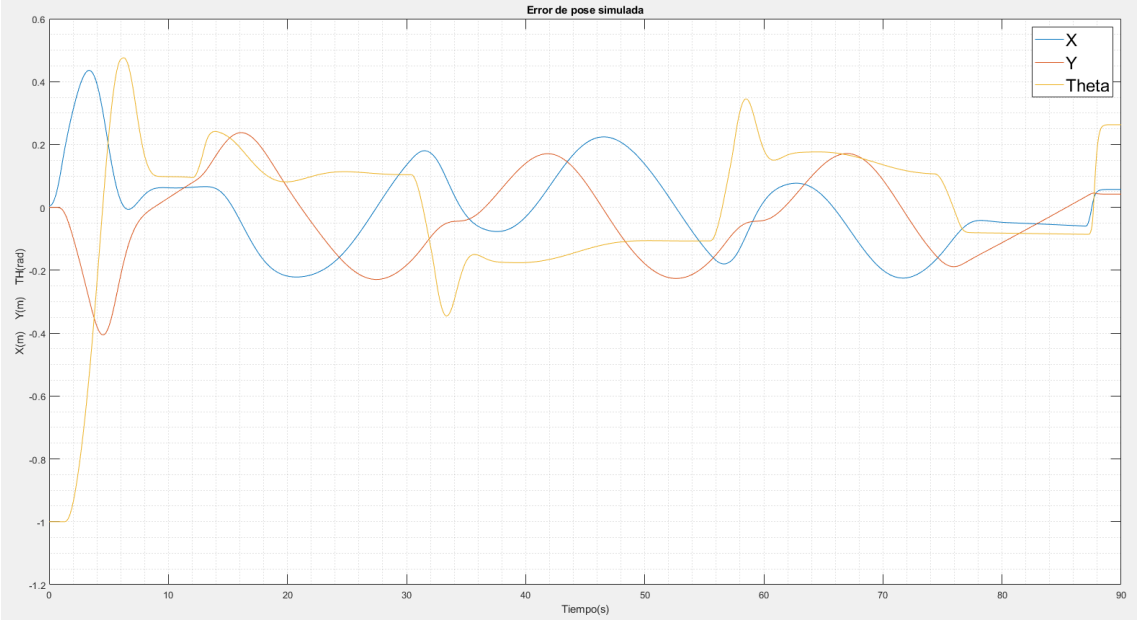


Figura 44. Error de pose en simulación

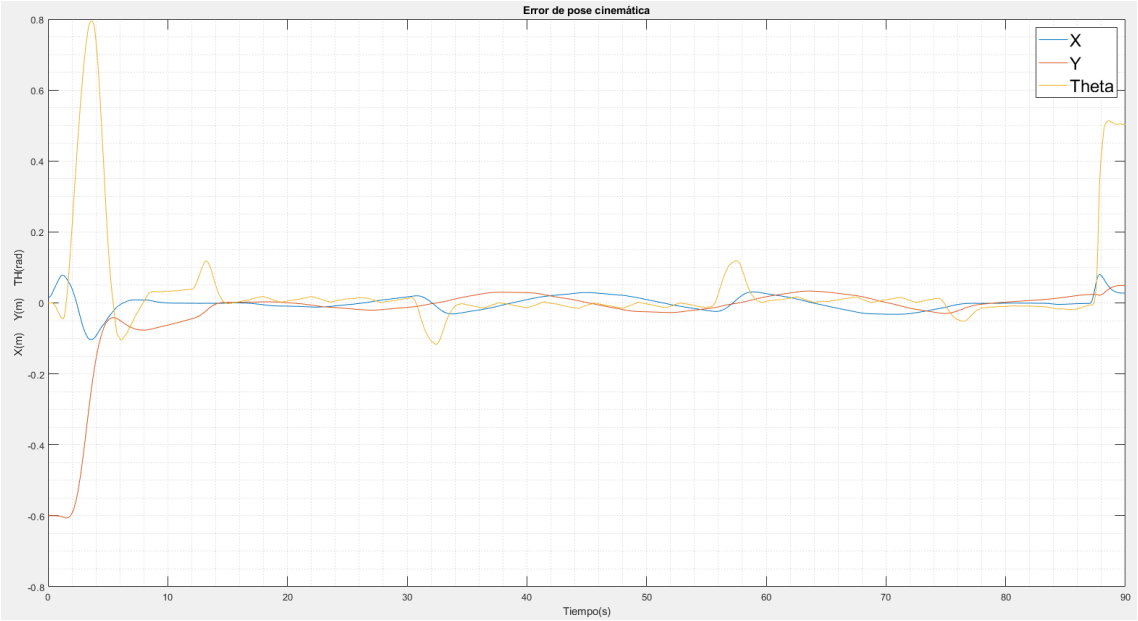


Figura 45. Error en pose de modelo cinemático

Finalmente, para cerrar este estudio, se muestran a continuación datos relacionados con la tasa de pérdida de paquetes UDP en la conexión, el tiempo de ejecución total real del ensayo y el tiempo entre muestras. En la figura 46 se observa la citada tasa de pérdida de paquetes, donde se muestra el número total de paquetes UDP enviados al robot y los devueltos por este, a su vez se muestra con una línea vertical de color amarillo el momento exacto en el que se produjo la pérdida.

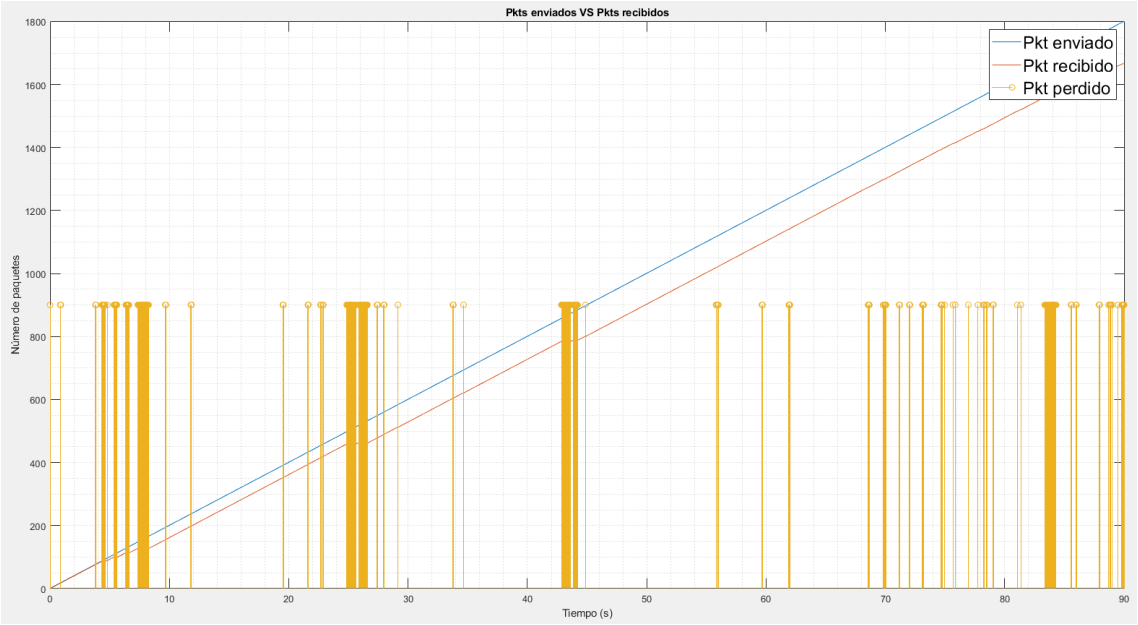


Figura 46. Paquetes enviados, recibidos y perdidos en el ensayo LBC sin realimentación

Por último, en la figura 47 se muestra la gráfica que detalla la información correspondiente al periodo de muestreo:

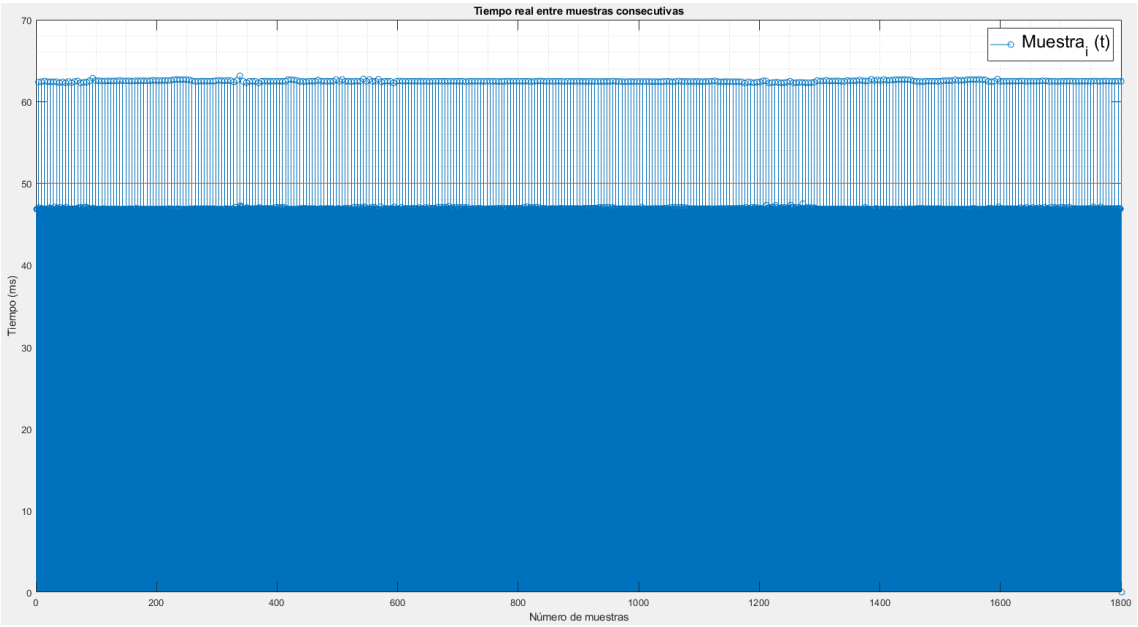


Figura 47. Tiempo entre muestras en el ensayo LBC sin realimentación

```
Se han perdido 198 paquetes
Desviación típica: 6.369950ms
Media: 49.972580ms
La ejecucion ha durado 90.000617 segundos en vez 90 segundos
```

Se muestra cómo la ejecución ha durado el tiempo establecido a priori (90 segundos), que la media del periodo de muestreo es muy próximo al ideal (50ms), con una pequeña desviación típica respecto a ese valor medio, y con una pérdida total de 198 paquetes, los cuales representan el 11% de los paquetes enviados durante el ensayo, siendo este un dato asumible, contando que la comunicación es vía inalámbrica con el receptor en movimiento y con un protocolo de comunicación que no está orientado a la conexión.

La segunda solución LBC propuesta se basa en la realimentación de los datos obtenidos por la cámara para mejorar el control. En la siguiente figura se muestra el modelo empleado en Simulink para la generación de código:

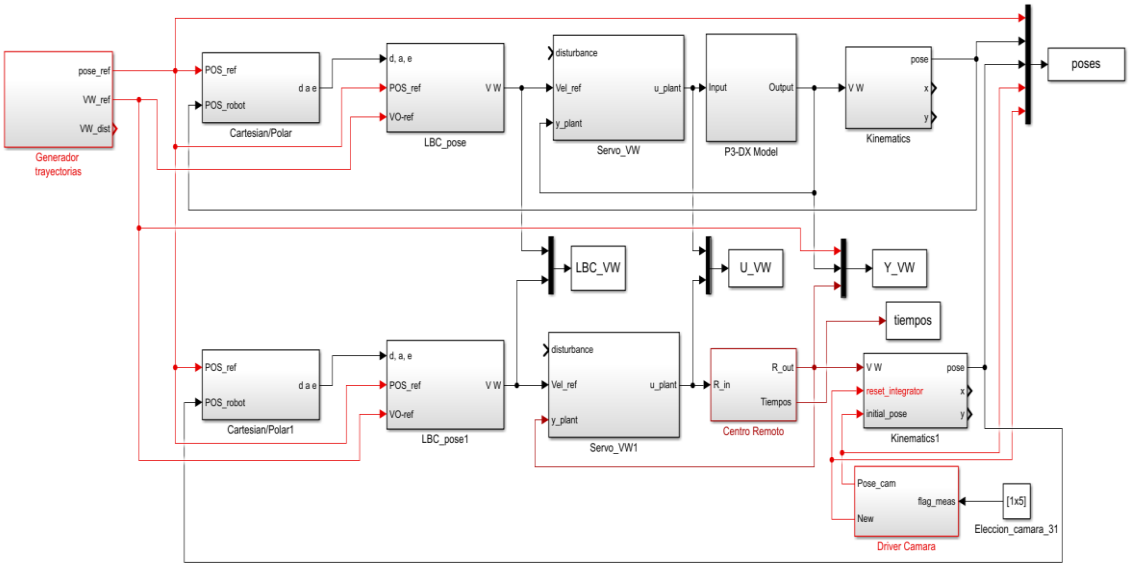


Figura 48. Modelo de solución LBC con realimentación de cámara

En este caso, la realimentación de la pose se obtiene a partir de los datos que recoge la cámara Kinect anteriormente comentada. Dado que el periodo de adquisición de la pose con la cámara es entre 4 y 6 veces superior a la del periodo de control, se sigue manteniendo realimentación básica (Ts=50ms) a partir de la odometría como en el caso anterior, pero cuando la cámara registra una nueva pose se incorpora al lazo de control, actualizando así los integradores del módulo “kinematics1” (ver figura 48).

En la figura 49 se muestra la trayectoria de referencia, la obtenida a la salida del modelo cinemático (por lo tanto, realizada por el robot), y la obtenida por la realimentación que provoca la visión artificial.

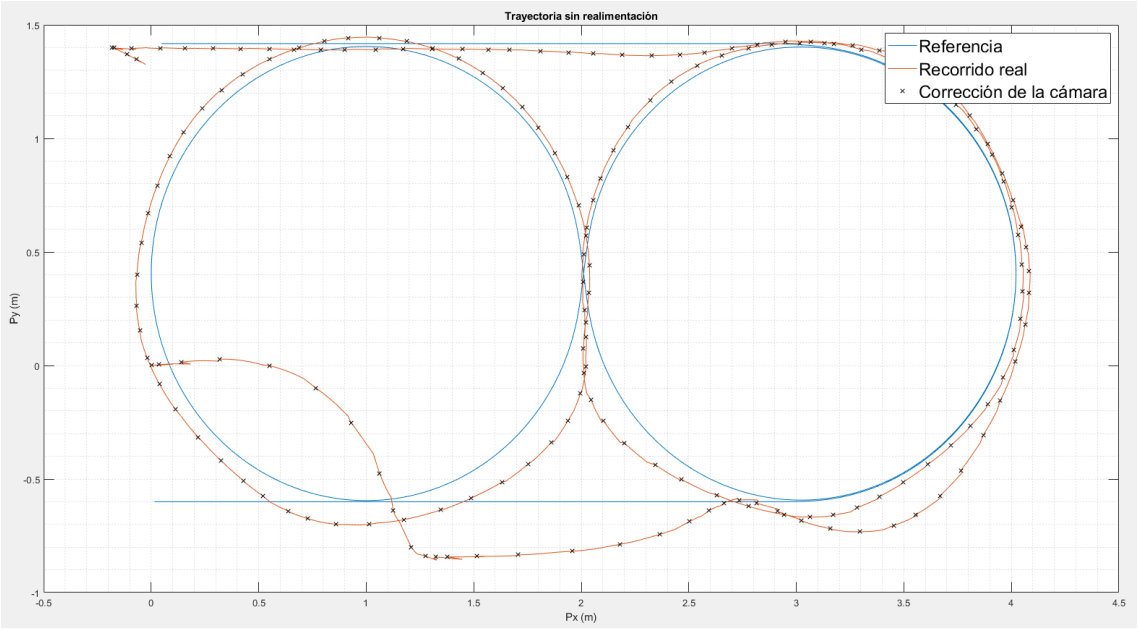


Figura 49. Comparación de trayectorias con realimentación de la cámara

Se observa cómo con esta solución la trayectoria realizada por el robot (línea roja) es mucho más parecida a la referencia (azul). Esto es debido a la realimentación que proporciona la cámara respecto a la posición real del robot. En la figura anterior se muestran con aspas negras los momentos en los que la cámara aporta información al lazo de control con nuevos datos para corregir la pose.

En la siguiente figura se muestra la diferencia existente entre la velocidad lineal y angular proporcionada por el control LBC en simulación y la recogida en el ensayo real:

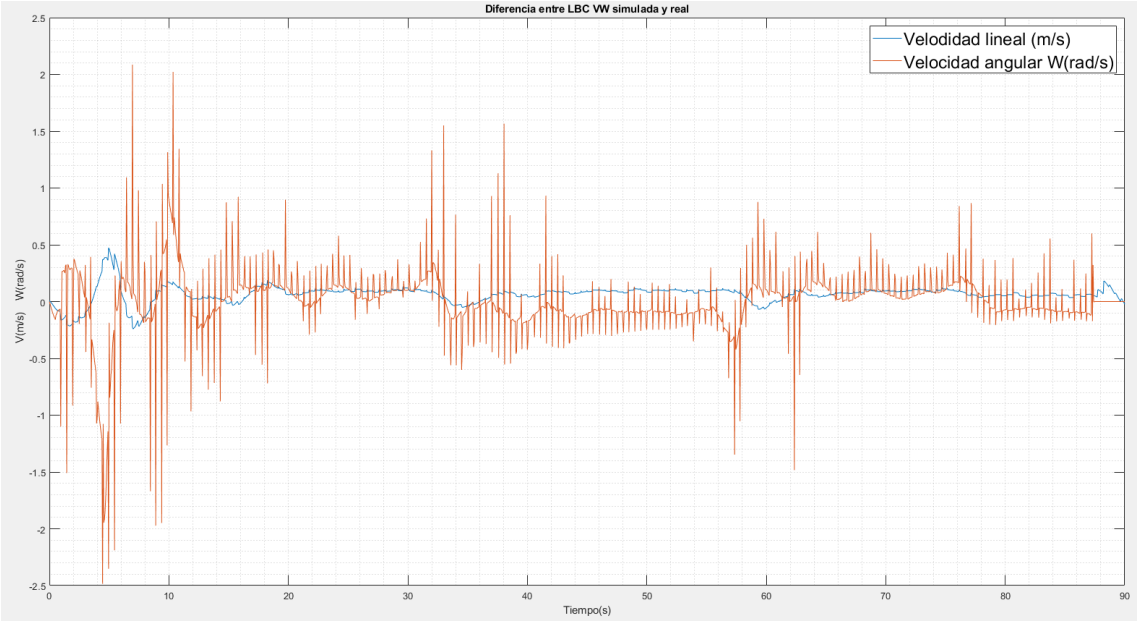


Figura 50. Diferencia tras control LBC en simulación e implementación

En la figura anterior se ve cómo la velocidad angular varía su valor constantemente de forma radical, debido a la realimentación que produce la cámara y el intento de corrección de la trayectoria. Si se compara dicha figura con su igual sin realimentación (ver figura 42), se entiende que el robot por sí solo, sin realimentación, no corrige su trayectoria y por lo tanto se tiene un trazo más lineal, pero cuando se tienen en cuenta los datos proporcionados por la visión artificial para el cálculo de trayectoria, y por lo tanto una corrección de la velocidad angular, se tiene trazo más irregular y variante debido a la nueva información proveniente de la cámara.

Ahora, se representa la diferencia existente entre las velocidades aplicadas al modelo (simulación) y al robot (implementación) para el seguimiento de la trayectoria, mostrada en la figura 49:

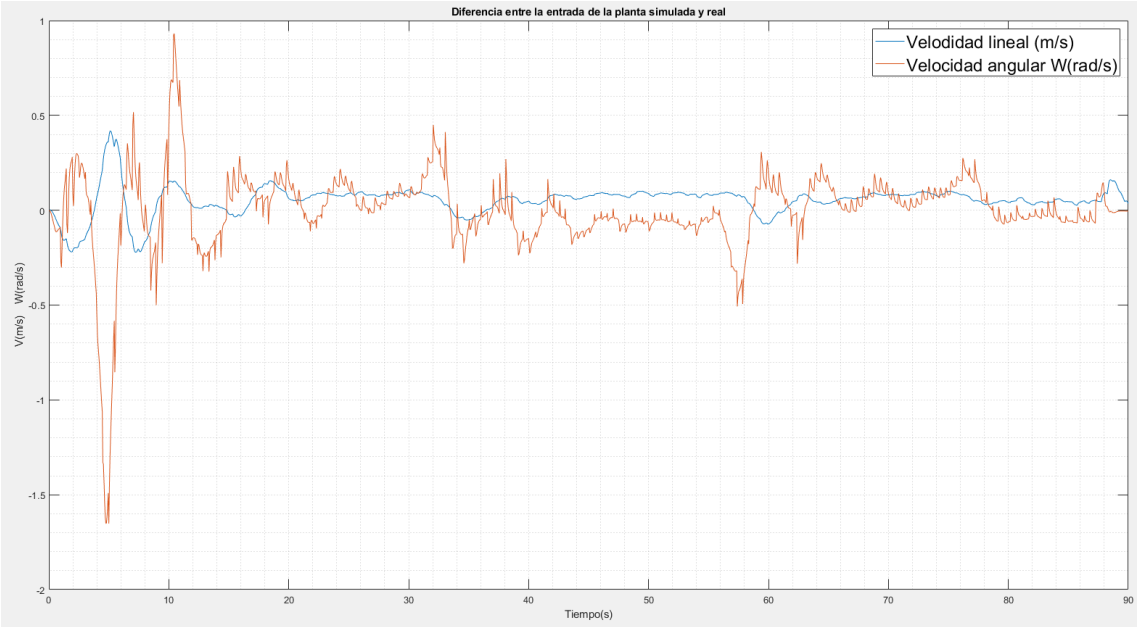


Figura 51. Diferencia de consignas en la entrada a la planta simulada y real

Finalmente, en las figuras 52 y 53, se compara el error en posición (x, y) y en orientación (θ) resultantes de la ejecución simulada y experimental.

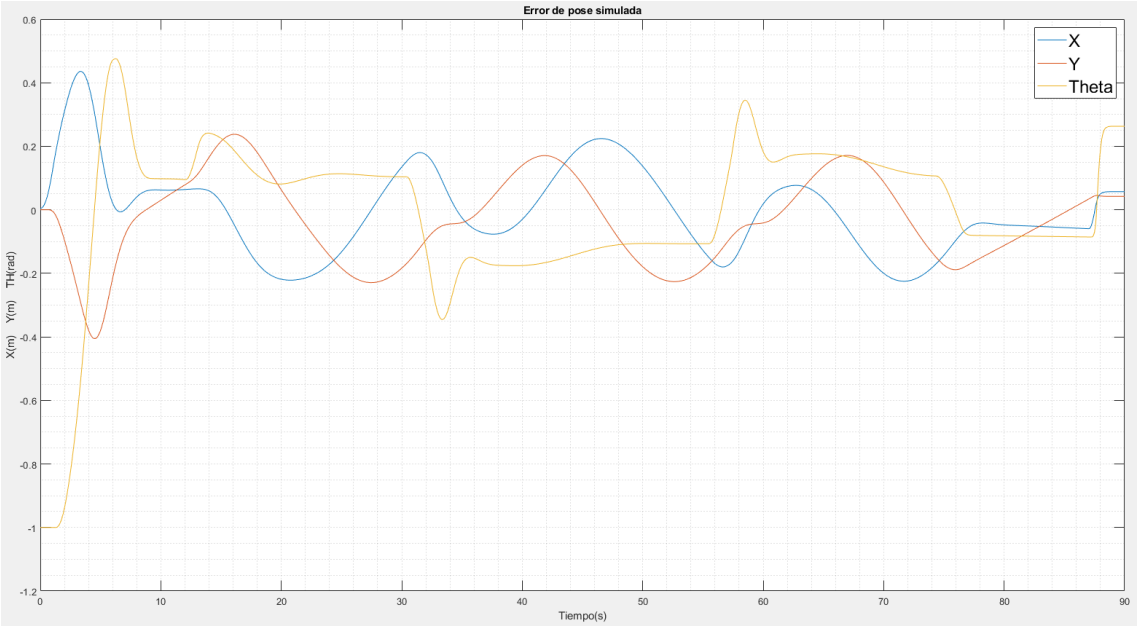


Figura 52. Error de pose en simulación

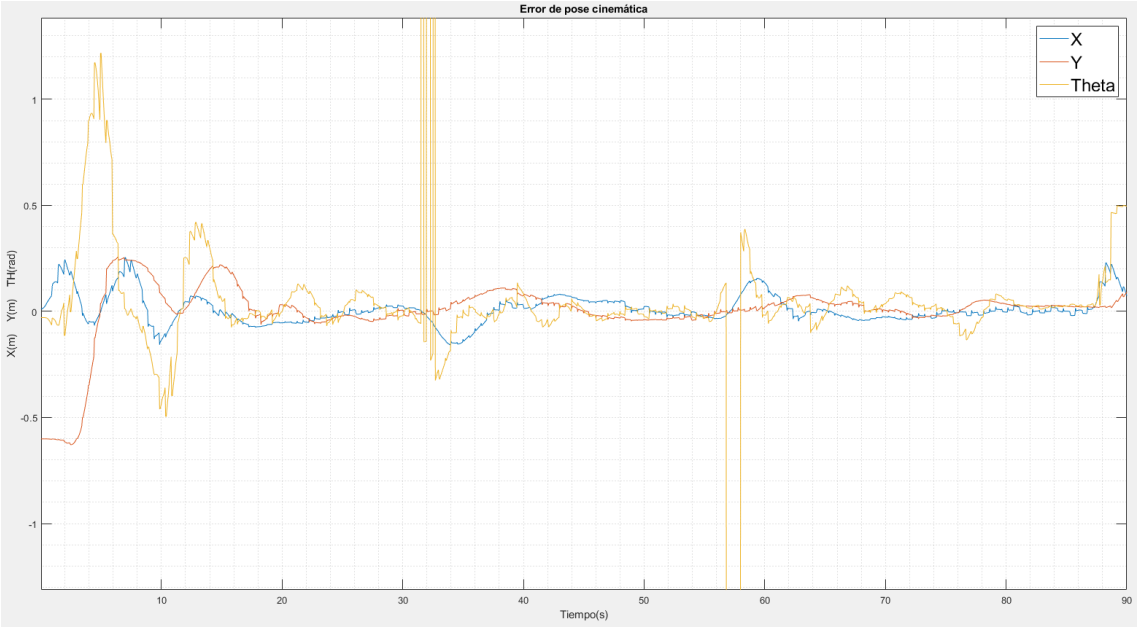


Figura 53. Error de pose en la implementación

Las altas desviaciones que se observan en el error en la pose implementada al robot son debidas a la corrección software que se realiza antes de mostrar los resultados, donde se compara el valor de orientación dada, y si el valor sobrepasa 2π (una vuelta entera) se corrige llevando dicho valor de nuevo al origen, ya que si no se realiza dicha corrección la gráfica aumentaría su valor por cada vuelta que realiza el robot.

Para cerrar el estudio, a continuación, se muestran las gráficas correspondientes a la comunicación entre el centro remoto y el robot, y la información relacionada con el tiempo de ejecución y el periodo de muestreo. En la siguiente figura se observa la tasa de paquetes perdidos en la realización del estudio, siendo de nuevo inalámbrico el canal de comunicación entre ambos nodos y empleando el protocolo UDP:

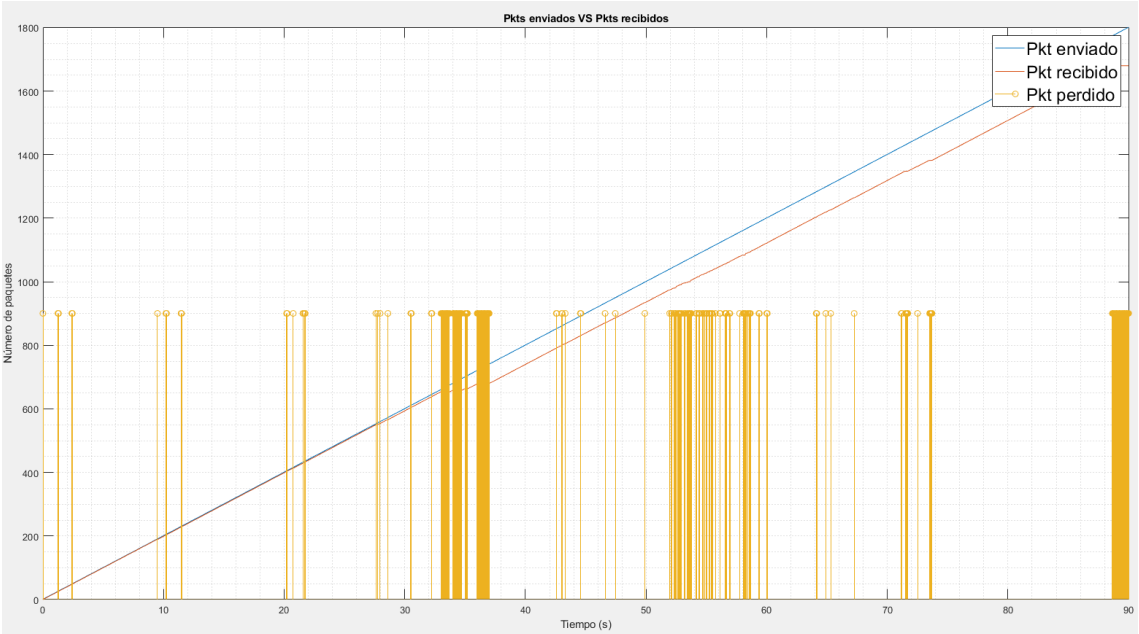


Figura 54. Paquetes enviados, recibidos y perdidos en el ensayo LBC con realimentación

En la figura 55 se muestra la representación gráfica resultante de medir los tiempos existentes entre las distintas muestras realizadas en el ensayo:

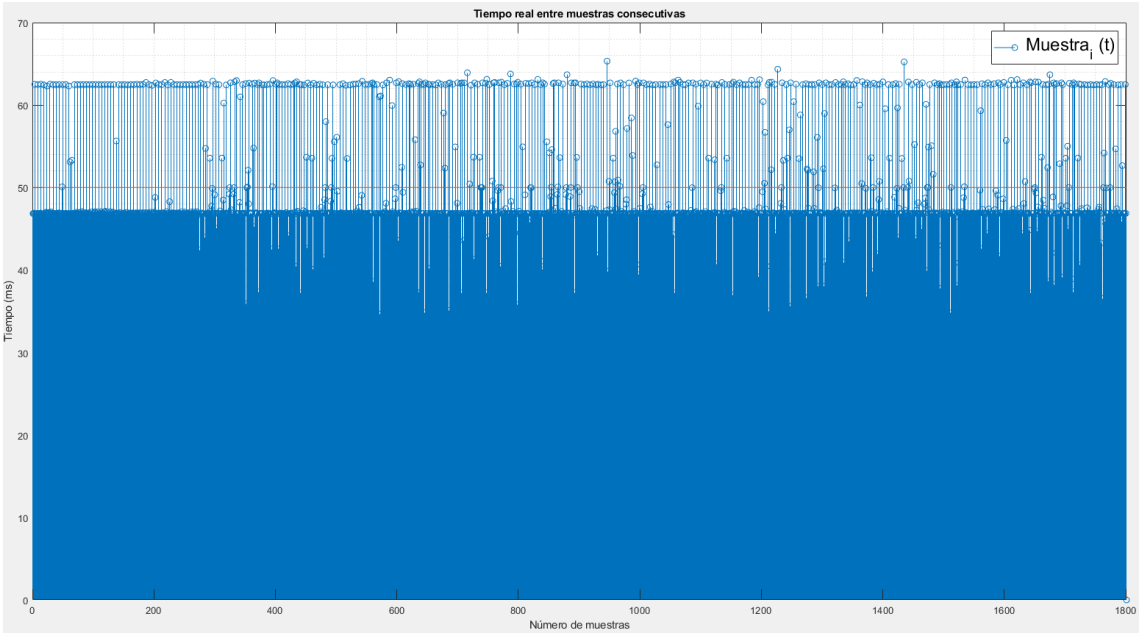


Figura 55. Tiempo entre muestras en el ensayo LBC con realimentación

```
Se han perdido 203 paquetes
Desviación típica: 6.782289ms
Media: 49.970927ms
La ejecucion ha durado 89.997640 segundos en vez 90 segundos
```

La tasa de error de paquetes perdidos es similar a la obtenida en el ensayo sin realimentación de la cámara. También se observa cómo la desviación típica del periodo

de muestreo, la media de este y el tiempo de ejecución total del estudio es más o menos parejo al estudio anterior, por lo que se tiene un resultado satisfactorio.

3.5. Resumen de trabajo con el cliente en WINDOWS 10

El objetivo principal, como bien resume el título de este documento, era la actualización de herramientas para el diseño e implementación de controladores digitales con Matlab en su versión 2018b. También se tenía en mente la posibilidad de realizar los estudios sobre el sistema operativo Windows 10, ya que es el más novedoso en lo que a tiempo se refiere y es uno de los sistemas operativos más empleados. Para valorar objetivamente si los objetivos marcados a priori han sido cumplidos, basta sólo con echar un vistazo al último ensayo realizado, donde se realiza un control remoto para seguimiento no lineal de trayectorias. Las evidencias gráficas más representativas son las figuras 49 y 55, donde en la primera de ellas se observa la trayectoria conseguida empleando Windows 10 como SO base, y empleando el software Matlab2018b para la creación de la aplicación lanzada desde el centro remoto, con un target distinto al empleado habitualmente en prácticas anteriores, siendo este el Generic Real-Time (grt.tlc). La segunda figura citada es una prueba más de lo satisfactorio que ha resultado este ensayo sobre las condiciones anteriormente descritas, ya que se muestra el tiempo que se ha tomado entre cada muestra realizada teniendo un periodo de muestreo ideal de 50ms y donde la mayoría de las muestras rondan dicho valor ideal. También un dato de interés cuando se trabaja con aplicaciones en tiempo real es obviamente el tiempo final de ejecución, donde se ha comprobado que la ejecución del ensayo ha sido la correcta en todos y cada uno de los casos realizados.

Respecto a la elección del target con el que se genera código para crear la aplicación del centro remoto: es fácil, intuitivo y práctico de usar, ya que el propio software de Matlab 2018b incorpora dicho target e incluso las herramientas de extensión que permiten compilar funciones programadas en C/C++. Ha demostrado valer para ejecuciones como mínimo con un periodo de muestreo de 50ms como se ha citado anteriormente, y de estar a la altura del target utilizado en versiones anteriores y con otros sistemas operativos (RTAI).

4. Aplicaciones de control en tiempo real con Ubuntu18.04 y Matlab2018

Ubuntu [Ubuntu, 2018] es un sistema operativo de código abierto empleado para multitud de fines, tanto académicos como profesionales. Es una distribución de Linux [Linux, 2018] basada en la arquitectura de Debian. Este SO es frecuentemente utilizado fundamentalmente por la gran capacidad de control que se tiene sobre este, ya que al ser un software open-source cada usuario puede configurarlo "a medida" y conseguir sacar el mayor rendimiento al dispositivo donde esté instalado en función de la utilidad que se le vaya a dar. Para trabajar con aplicaciones de ejecución en tiempo real, existe la extensión de Linux llamada RTAI (Real-Time Application Interface), la cual sirve para modificar el kernel del sistema operativo con el fin de garantizar el cumplimiento de un periodo de muestreo marcado con anterioridad. Esta funcionalidad se consigue modificando parámetros como la frecuencia de oscilación del procesador, anulando la hibernación o incluso permitiendo el intercambio del nivel de prioridad entre aplicaciones.

Hasta la versión Ubuntu 16 se debía emplear este SO con la extensión anteriormente citada para trabajar con aplicaciones de ejecución en tiempo real, pero a partir de Ubuntu18, Matlab, incluye un target, Generic Real-Time (grt.tlc) con el cuál es posible generar código para crear aplicaciones en tiempo real como se ha demostrado en el capítulo 3 de este mismo trabajo. Por lo que, en este apartado se explica cómo se ha realizado el camino necesario para conseguir generar aplicaciones en tiempo real cuando el centro remoto está en un PC soportado por la versión Ubuntu18.04, y con la ayuda de las herramientas que incorpora Matlab en su versión 2018b.

4.1. Control remoto para seguimiento no lineal de trayectorias

En el estudio relativo al empleo del sistema operativo de Linux, se han obviado los ensayos sobre la compatibilidad de sockets e incluso la implementación del servosistema por separado. En este apartado directamente se muestran los resultados obtenidos al realizar un control remoto para seguimiento no lineal de trayectorias cuando el centro remoto se sitúa en un ordenador soportado por el sistema operativo Ubuntu18.04.

Se han realizado dos estudios, al igual que en el apartado 3.4.2 de este documento, donde el primero corresponde a un control para seguimiento no lineal de trayectorias con realimentación odométrica, y el segundo incorpora la realimentación al control proporcionada por la cámara situada en el pasillo donde se ha realizado dicho ensayo. A continuación, se muestran las figuras más relevantes en este estudio para su

comparación respecto al estudio realizado cuando el centro remoto está soportado por Windows 10.

Los resultados finales han sido similares: la entrada a la planta, el error en las poses, la salida de la planta e incluso los tiempos existentes entre cada una de las muestras. Lo único que ha variado notablemente es la tasa de paquetes perdidos en la comunicación, de 200 paquetes perdidos cuando se trabaja con Windows 10, a un valor medio entre ambos estudios de 70 paquetes perdidos cuando se tiene el centro remoto sobre Ubuntu18.04, es decir, casi 4 veces menos.

En la figura 56 se muestra la trayectoria real obtenida en el estudio del control sin realimentación de la visión artificial, donde es comparada la trayectoria de referencia, la trayectoria recogida por los encoders del robot y la realmente realizada por el robot (vista por la cámara):

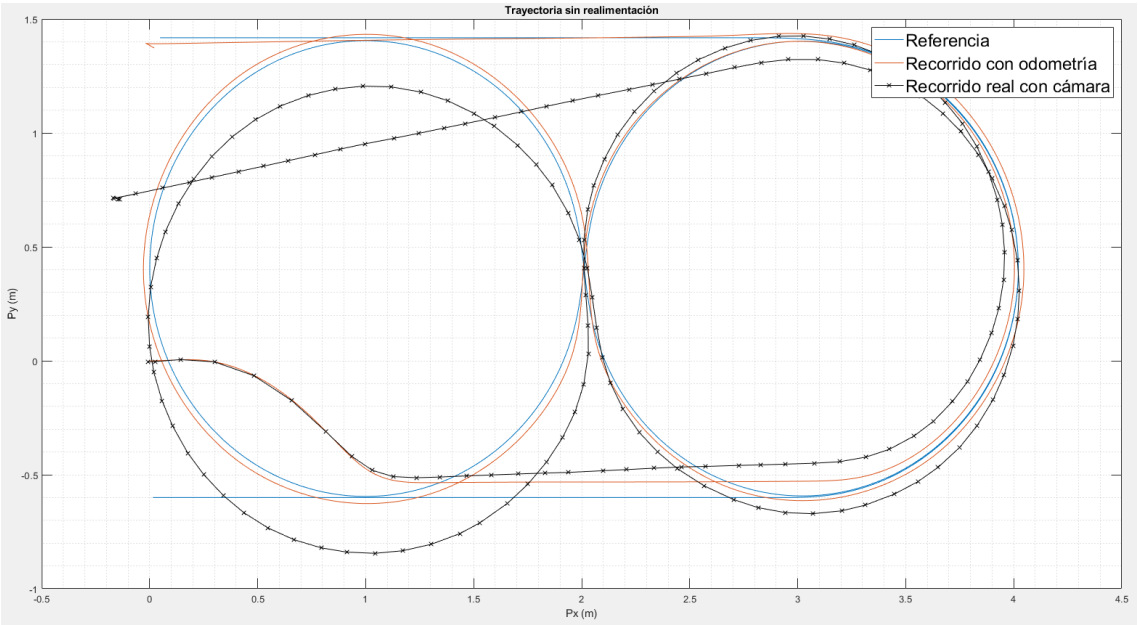


Figura 56. Comparación de trayectorias sin realimentación de la cámara

Se observa cómo la trayectoria resultante, tanto la obtenida por la odometría como la registrada por la cámara, son casi una copia exacta de la obtenida en el apartado 3.4.2 cuando se trabajaba sobre Windows 10 en el centro remoto.

En la siguiente imagen se ven reflejados los paquetes perdidos y el momento exacto del ensayo donde se han producido dichas pérdidas:

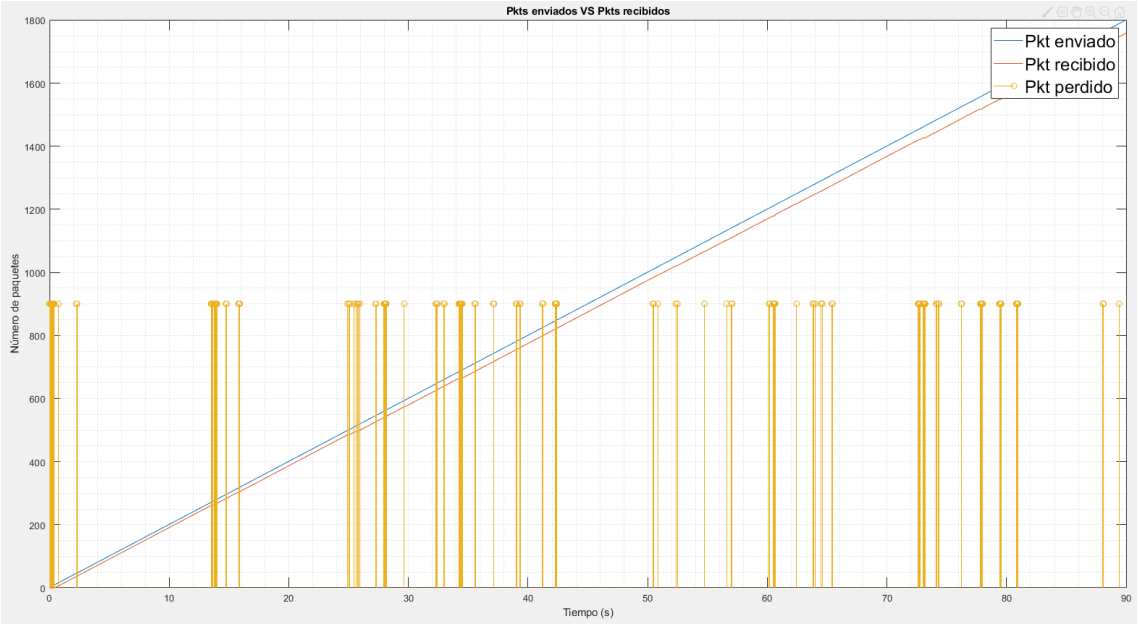


Figura 57. Paquetes enviados, recibidos y perdidos en el ensayo LBC sin realimentación

El número total de paquetes perdidos en este ensayo asciende hasta un total de 120.

A continuación, se muestra la trayectoria obtenida al incluir en el control de la trayectoria los datos que capta la cámara:

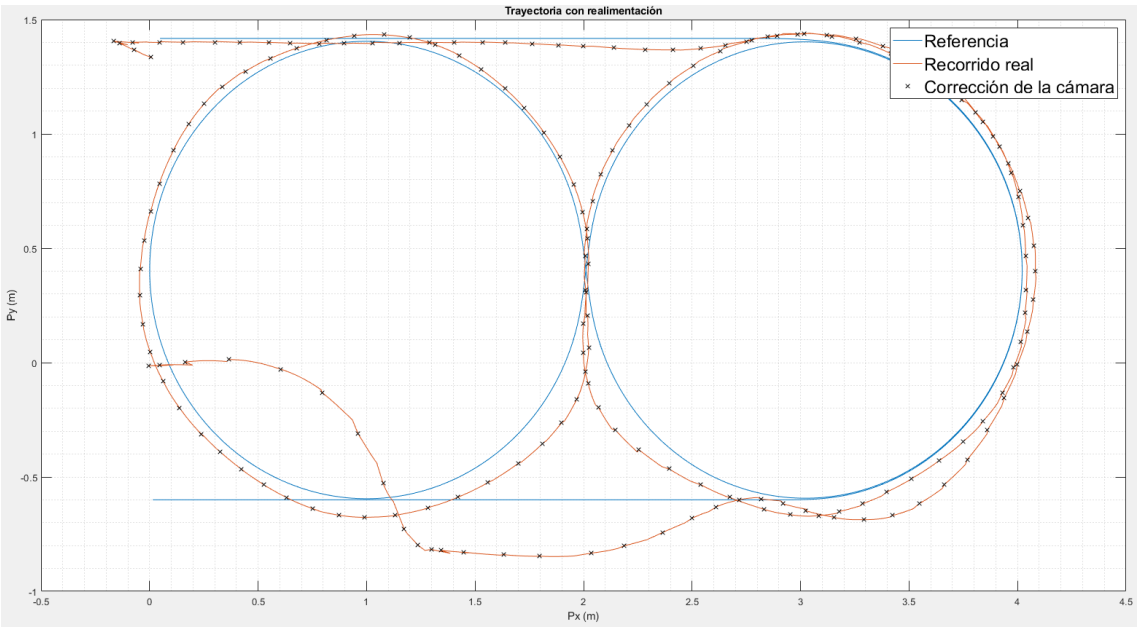


Figura 58. Comparación de trayectorias con realimentación de la cámara

La mejoría en la realización de la trayectoria es notoria cuando se introducen los datos de la cámara en el control, tal y como se muestra en la figura 58, donde la trayectoria

realizada en realidad es capaz, en mayor medida, de seguir a la trayectoria referencia marcada a priori e incluso teniendo condiciones iniciales distintas.

En la figura 59 se vuelve a mostrar una representación de la tasa de error en la comunicación, donde el número total de paquetes UDP perdidos es de 57:

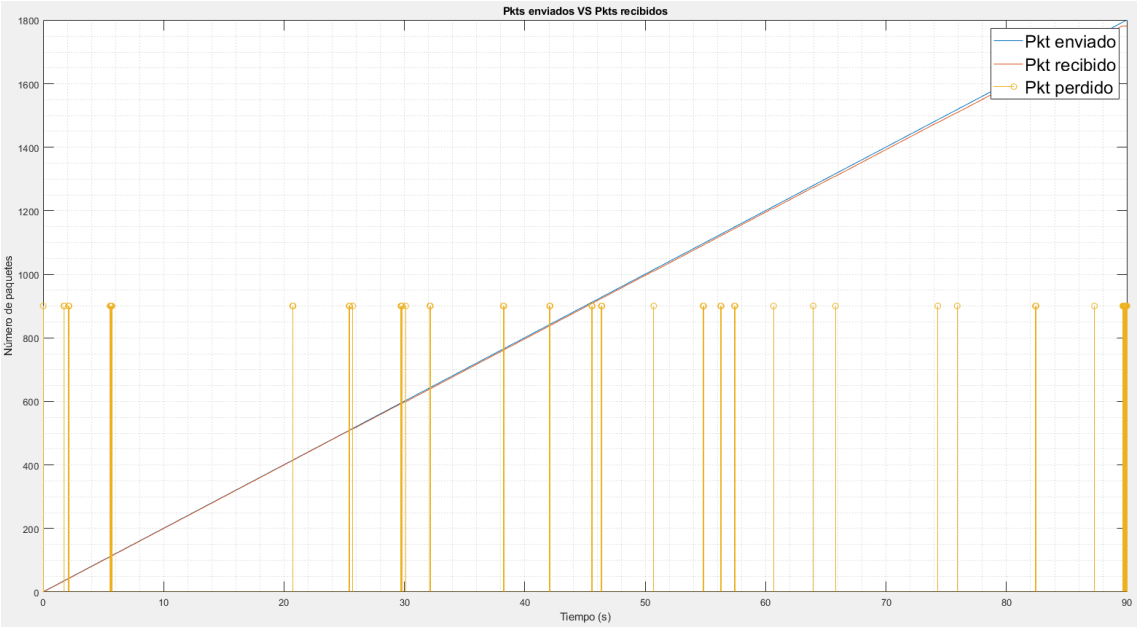


Figura 59. Paquetes enviados, recibidos y perdidos en el ensayo LBC con realimentación

4.2. Resumen de trabajo con el cliente en Ubuntu 18.04

Visto lo obtenido en el apartado anterior, se puede decir que se han alcanzado los objetivos de la actualización de las herramientas, ya que, con la última versión de Ubuntu y del software Matlab, se ha conseguido realizar correctamente los ensayos con ejecución en tiempo real. De nuevo, como en el caso del trabajo con Windows 10, la incorporación del target empleado para la generación de código (Generic Real-Time) por parte de Matlab facilita el trabajo, ya que no es necesaria la implementación de un parche que modifique el kernel del sistema operativo para crear aplicaciones de tiempo real. Si bien con la extensión de RTAI el usuario es capaz de garantizar que los tiempos de ejecución serán cumplidos, pero los resultados obtenidos con Ubuntu 18.04, sin la utilización de dicha extensión, son bastante satisfactorios tal y como se muestran las trayectorias conseguidas en las figuras 56 y 58.

III. CONCLUSIONES Y TRABAJOS FUTUROS

El objetivo principal cuando se comenzó este trabajo era la actualización de las herramientas con las que se trabajaba para el diseño y la implementación de controladores digitales. Tras el trabajo realizado, los resultados finales resultan totalmente satisfactorios, tanto para el sistema operativo Windows 10 como para Ubuntu 18.04, ambos con la versión 2018b de Matlab. Los resultados obtenidos en sendos sistemas operativos son muy similares entre sí, por lo que así se demuestra que los ensayos han sido realizados correctamente, ya que se ha trabajado desde el mismo PC en el centro remoto, sobre la misma red de conexión, con el mismo robot (P3-DX) y habiendo generado código para las aplicaciones mediante el mismo target e idéntico software.

A la hora de querer trabajar con la cámara Kinect, es muy importante tener en cuenta la red desde la cual se está trabajando en el centro remoto. El software implementado en la cámara sólo reconoce como centro remoto (independientemente del sistema operativo que se esté empleando) a aquel dispositivo que tenga como configuración de red la misma que la perteneciente al miniPC3 situado en el OL3, es decir, con los siguientes parámetros:

- Dirección IP: 192.168.11.105
- Dirección de difusión: 192.168.11.255
- Máscara de subred: 255.255.255.0
- Ruta predeterminada: 192.168.11.1
- DNS: 8.8.8.8

Como trabajo futuro está la continuidad de las actualizaciones de las herramientas de diseño, como pueden ser las versiones de Matlab que van saliendo a lo largo de los años, las versiones de los sistemas operativos con los que se han tratado en este documento, o tal vez la posibilidad de trabajar con nuevos sistemas operativos. Se tiene también como tarea futura la actualización del hardware que incorpora el robot, pasando de una arquitectura basada en Via Epia actual a tener un mini PC incorporado en el robot P3-DX. . Otra línea que explorar es el diseño de controladores (borroso, neuronal, etc) tanto para guiado de robots como para guiado cooperativo entre varias unidades.

IV. ANEXOS

5. Anexo I: Configuración para la generación de ejecutable a partir de modelo en Simulink con Matlab 2018b

La configuración de los parámetros en la creación del ejecutable es un paso vital para que la aplicación responda como se desea y se pueda trabajar correctamente. Se trata de comunicar al entorno de Matlab cómo debe crear la aplicación y qué tiempos debe intentar respetar, asegurando un tiempo final de ejecución, un periodo de muestreo ideal, el número de bits que tienen los datos a tratar, etc.

En versiones de Linux anteriores a Ubuntu18 era necesario aplicar la extensión RTAI a la base del SO para aplicar cambios tales como modificar la prioridad en las tareas que se ejecutan, la frecuencia de oscilación del procesador, anular la hibernación, y demás ajustes que aseguraban un cumplimiento de los tiempos de ejecución a un periodo de muestreo establecido. En cambio, con el uso de Windows 10 este parche no se requiere, ya que es un sistema operativo cerrado y no permite modificar su arquitectura interna. No obstante, Matlab incluye un target con el que montar el modelo y así indicar al SO que se trata de una aplicación que debe respetar los periodos de muestreo lo más estrictamente posible, dentro de las limitaciones que este sistema operativo presenta. Para la versión de Ubuntu 18.04 también es necesario seguir estos pasos de configuración del entorno Simulink para indicar que se trata de un modelo cuya finalidad es la de la ejecución en tiempo real.

Para empezar, se debe tener un modelo (fichero *.slx o *.mdl), por ejemplo ejem.slx, en Simulink abierto. En la parte superior se ve la barra de herramientas que incluyen varias opciones, en este Anexo I se va a tratar sólo la parte de configuración de parámetros, para ello hay que hacer click en el icono de configuración como se muestra en la siguiente imagen:

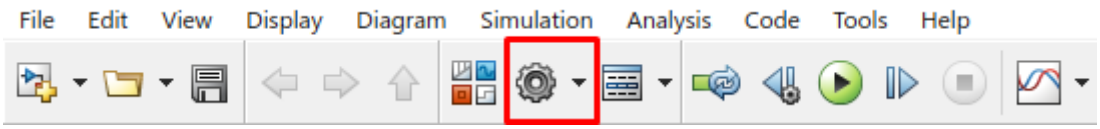


Figura 60. Anexo I: Icono de configuración en Simulink

Una vez abierta la ventana de configuración debe aparecer un entorno parecido al que se muestra en la figura 61, con varios apartados de configuración listados a la izquierda. Como primer paso nos fijamos en la opción que se marca en la lista como “Solver”:

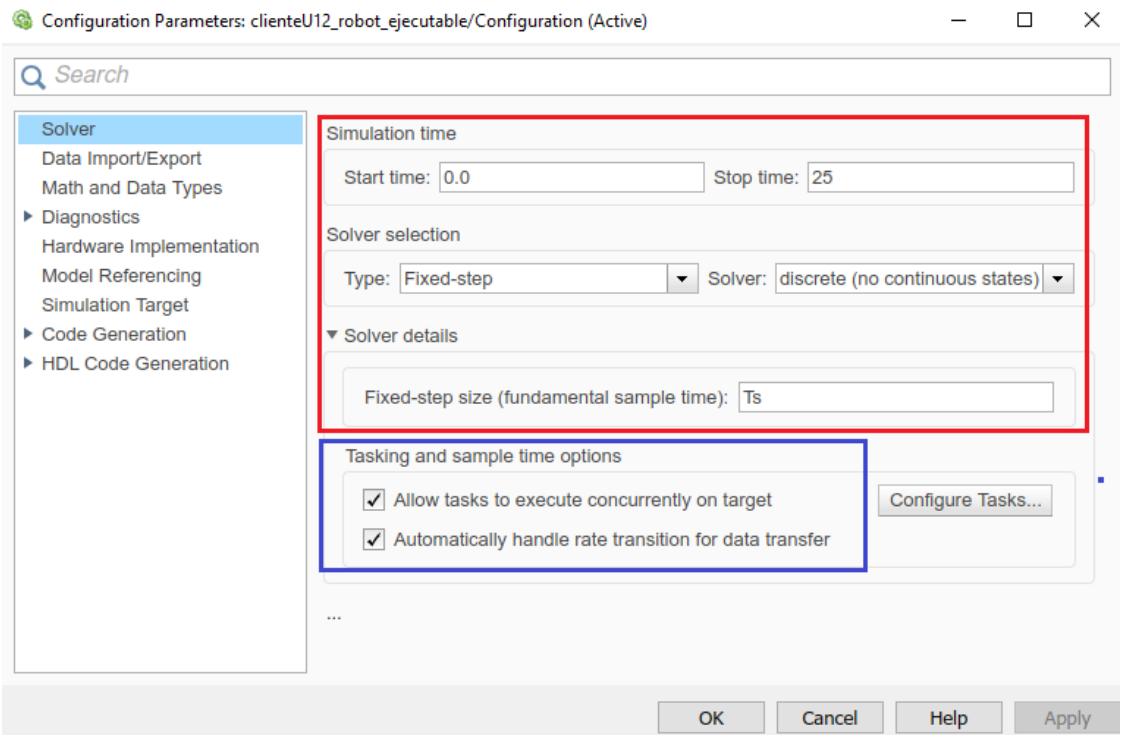


Figura 61. Anexo I: Solver

Sobre recuadro rojo se fijan parámetros relacionados con el tiempo, donde se establece un tiempo de ejecución final como Stop_time, se declara que se trata de un sistema discreto con un tiempo de ejecución fijo y con un periodo de muestreo asociado a la variable `Ts', que debe estar declarada en el WorkSpace de Matlab. Respecto a los parámetros encerrados sobre el rectángulo azul, es la parte más importante del proceso, ya que al marcar esas dos opciones se está indicando al entorno de Simulink que la finalidad de la aplicación es la ejecución en tiempo real.

La siguiente parte de la configuración trata sobre la especificación del modelo de procesador que se tiene instalado en el dispositivo. Esto evita problemas de ejecución y establece cuántos bits forman cada tipo de datos que se emplean en el modelo. Por ejemplo, en el apartado 3 de este documento se ha trabajado con un PC cliente que poseía el sistema operativo Windows, con un procesador de la marca Intel, por lo que es necesario reflejar dichos parámetros para que la longitud de los datos concuerde entre lo que Simulink interpreta y lo que realmente el dispositivo posee. En la siguiente figura se muestra cómo han sido configurados estos parámetros, atendiendo sobre el nombre “Hardware Implementation” de la lista de configuración:

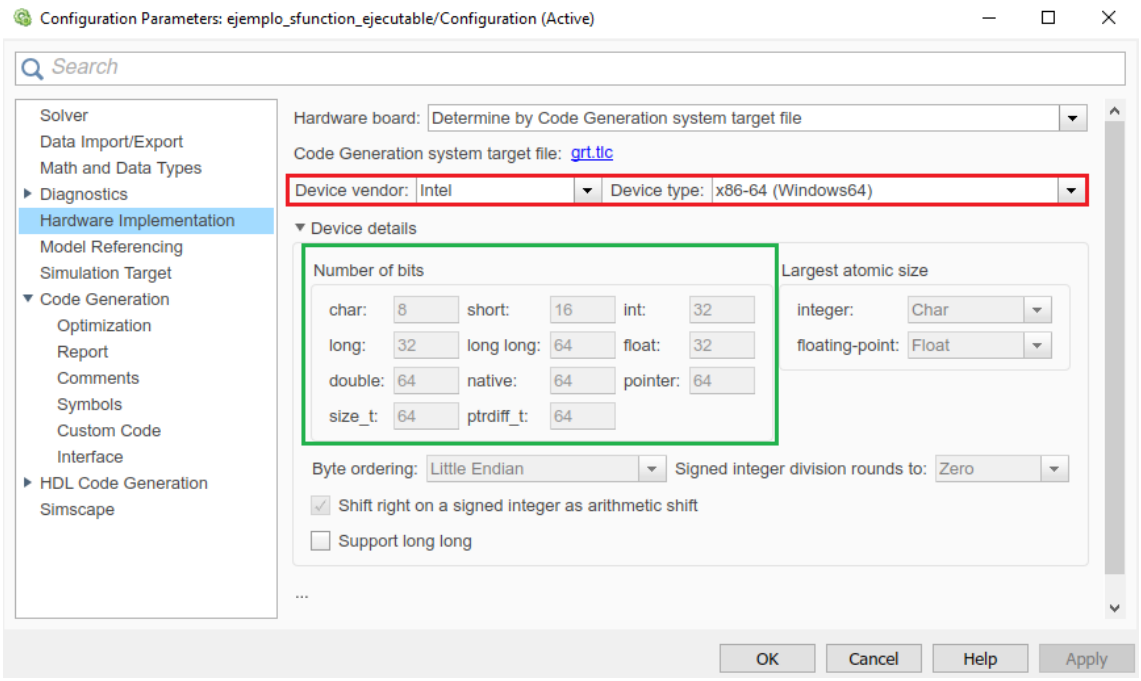


Figura 62. Anexo I: Hardware Implementation

Sobre este apartado es destacable decir que, si se está trabajando con la interacción entre Windows y Linux, es necesario que no se empleen variables de tipo long en la programación del driver, ya que la longitud de las variables varía entre los distintos sistemas operativos y la comunicación nunca llega a suceder ya que un dispositivo envía 32 bits, pero el otro espera que el dato sea de 64 bits. Como solución se puede trabajar con datos de tipo int cuando se necesite emplear un entero y datos de tipo float cuando se requieran de números en coma flotante.

Para finalizar la configuración del entorno Simulink, es necesario dirigirse sobre la opción marcada por el nombre “Code Generation”, donde se va a establecer qué compilador debe usar Matlab para crear el ejecutable de la aplicación, sobre qué target, etc. En la siguiente figura se ve una de las ventanas de interés que se deben modificar a la hora de querer crear una aplicación que atienda una ejecución a tiempo real:

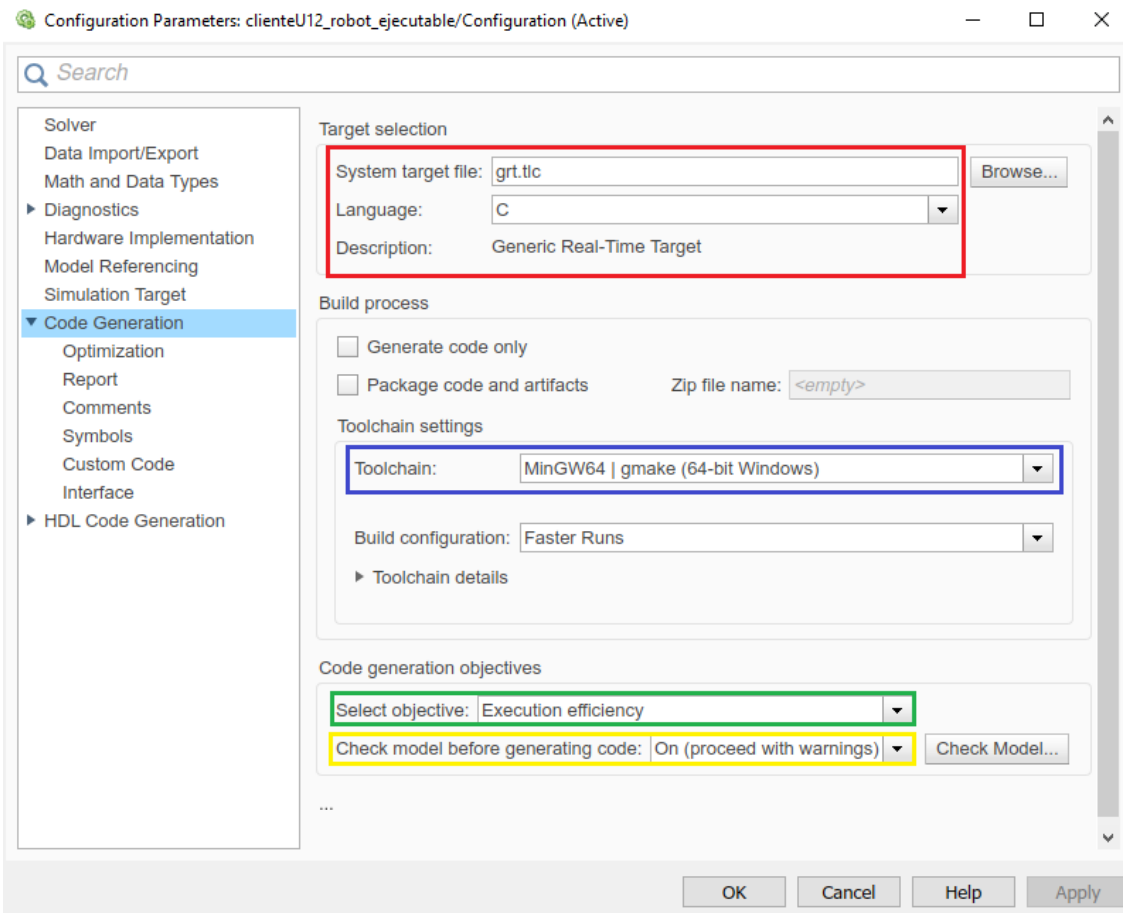


Figura 63. Anexo I: Code Generation

Sobre el recuadro marcado con color rojo se establece el target (*.tlc) para el que Simulink creará el código, en este caso: Generic Real Time (grt.tlc). El lenguaje de programación se establece en C, teniendo también la posibilidad de trabajar con C++. En la zona marcada por color azul se debe indicar el compilador a utilizar durante la creación del ejecutable, siendo este el MinGW64. Es recomendado por Matlab y el manual de instalación y configuración se encuentra en el Anexo III de este documento. De manera opcional, en el recuadro verde se elige el objetivo que se quiere tener a la hora de lanzar la aplicación, y a la hora de trabajar con ejecuciones en tiempo real se quiere la mayor eficiencia en la ejecución, por lo que es recomendable seleccionar la opción `Execution effiency'. Para la información marcada con color amarillo se debe marcar la opción `On (proceed with warnings)', que hará que antes de generar código se realice una comprobación del modelo y avisando de posibles errores.

Finalmente, como parte de la configuración del entorno, es necesario indicar al software que debe soportar valores no finitos para evitar posibles problemas de implementación y además marcar la opción de "MAT-file logging" para hacer posible la inclusión de los datos de salida en un fichero de almacenamiento como puede ser un .mat. En la siguiente figura se muestran ambas opciones a marcar dentro de la etiqueta Code_Generation/Interface:

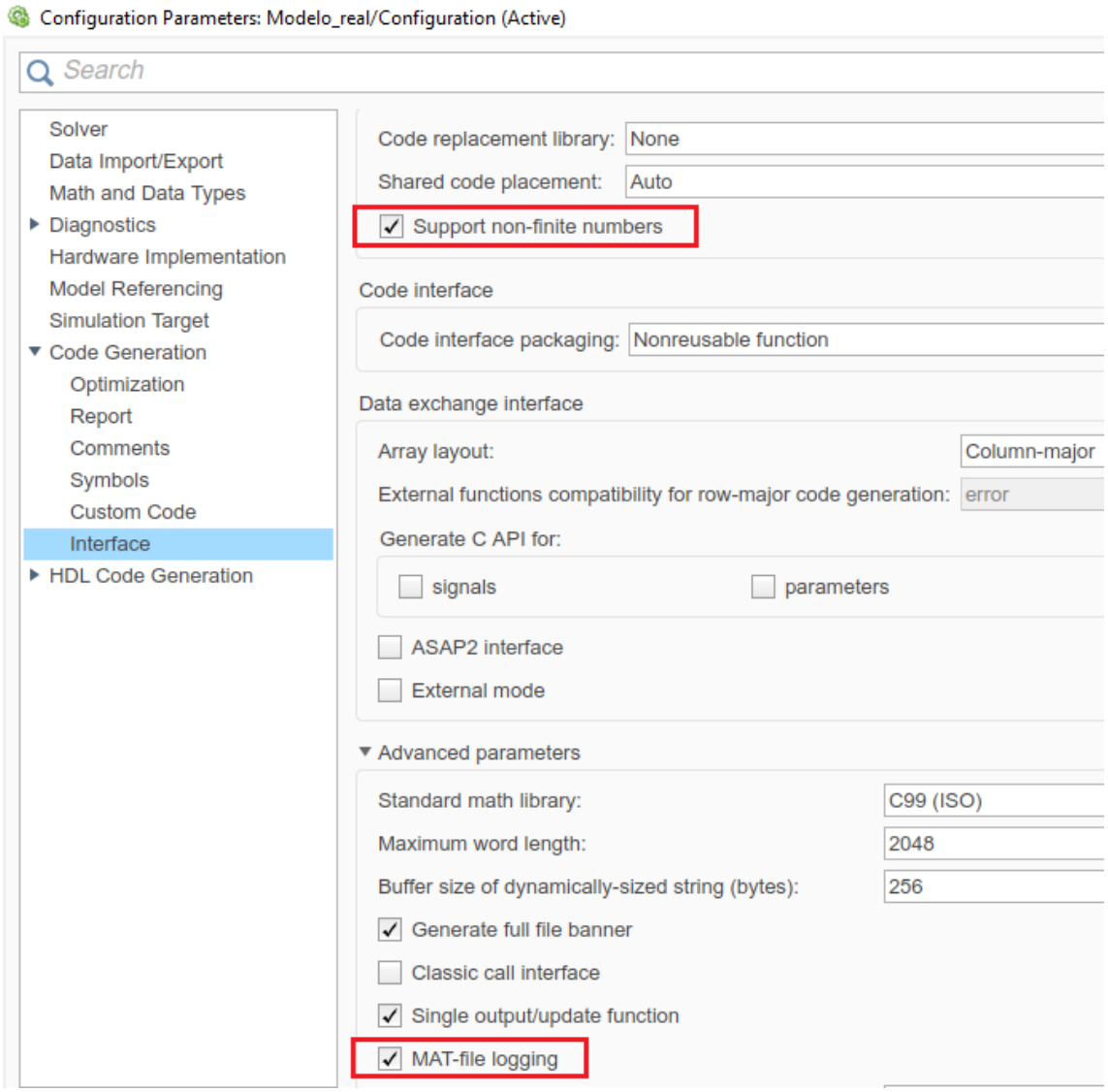


Figura 64. Anexo I: Interface

Una vez seguidos estos pasos de configuración, el modelo creado en Simulink (ejem.slx) ya estaría preparado para crear un ejecutable mediante el atajo Ctrl + B. El proceso genera dos directorios nuevos “codegen” que hacen referencia al ejecutable creado (ejem.exe en Windows), una con el nombre “slprj” y otra que recibe un nombre determinado en función del nombre que tenga el modelo de Simulink, en este caso “ejem_grt_rtw”. Para lanzar la aplicación creada se ha de ir a la carpeta donde se localizaba el modelo y hacer doble click sobre el ejecutable generado, aunque es recomendable hacer click derecho sobre el icono y clicar sobre la opción “Ejecutar como administrador”, ya que así el sistema operativo dará mayor prioridad a la aplicación.

A continuación, se procede a realizar un sencillo ejemplo de cómo crear un ejecutable sobre Windows 10 y con el software de Matlab 2018b. Para ello, se comienza abriendo el entorno de desarrollo de Simulink y establecemos un periodo de muestreo en una variable que más tarde será cargada en Simulink, en este ejemplo se ha establecido un periodo de 5ms. Una vez cargada esta variable en el workspace de Matlab, se continúa el ejemplo abriendo el entorno de Simulink y creando un modelo sencillo. Para la realización de esta demostración se ha tomado un seno como referencia, seguido de un bloque de ganancia con valor 5, y finalmente un bloque "to file" donde se recogerán los datos obtenidos. Para la comparativa de la referencia de entrada al amplificador respecto a la de salida se ha colocado un multiplexor. El modelo creado se muestra en la siguiente figura:

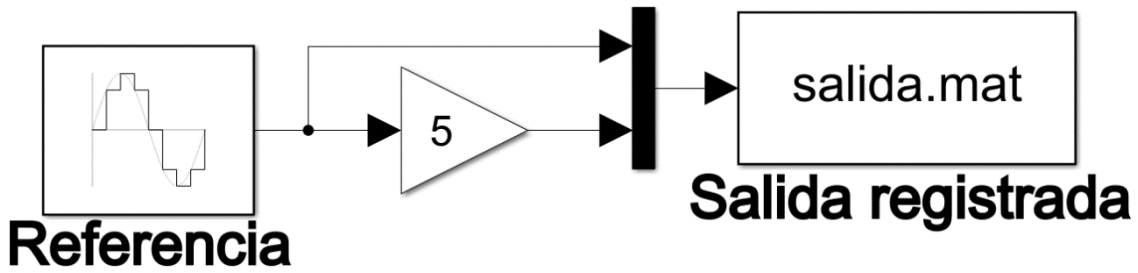


Figura 65. Anexo I: Modelo del ejemplo en Simulink

Una vez que el modelo está construido y la configuración de los bloques que lo forman realizada, se debe comenzar con la configuración del entorno de Simulink. Para ello simplemente se repiten los pasos descritos en este Anexo I. Cuando se haya configurado Simulink se construye el modelo mediante la macro Ctrl + B. Si el usuario quiere observar el proceso que va realizando Matlab en la creación del ejecutable como ocurría en versiones anteriores de Matlab sobre la consola de comandos, en la parte inferior de la ventada del modelo aparece una opción en modo de texto: "View diagnostics", que clicando sobre dicho texto da lugar a una ventana que va reportando los pasos realizados para la construcción del ejecutable. Matlab indica la correcta operación mediante el siguiente mensaje:

```
Build process completed successfully
```

Seguidos estos pasos, el usuario ya debe tener en su carpeta de trabajo el ejecutable con el nombre que tenía el modelo de Simulink, pero con extensión .exe que indica de que se trata de una aplicación. En la siguiente figura se muestran los resultados una vez ejecutada la aplicación:

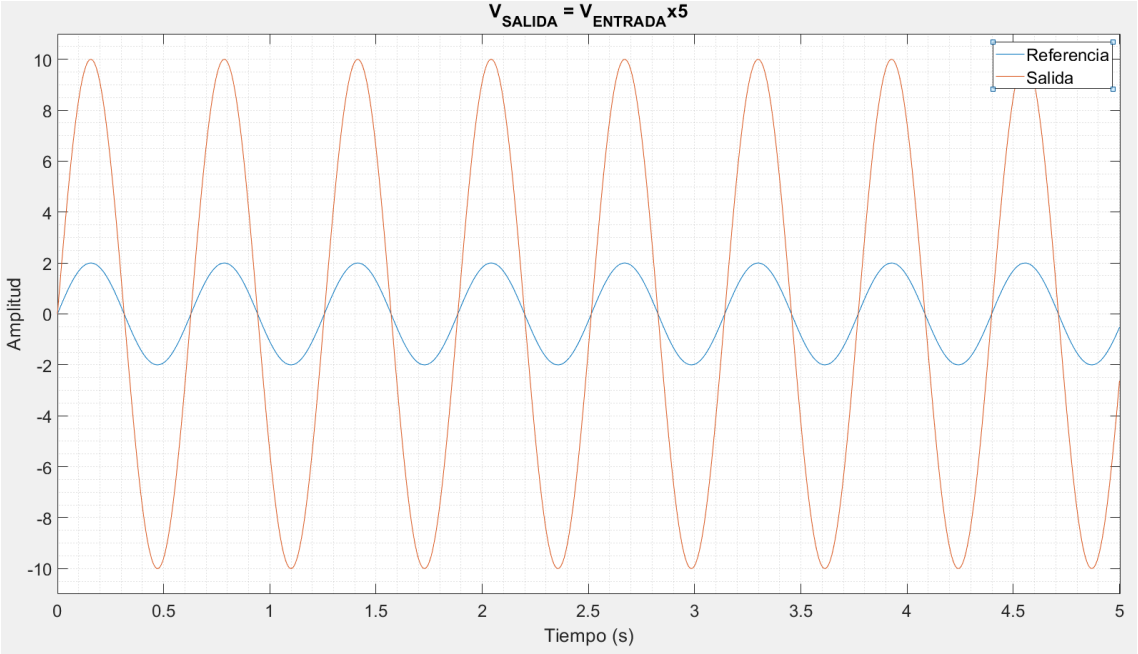


Figura 66. Anexo I: Resultado de ejecución a tiempo real

6. Anexo II: Lógica para la implementación del driver

En este anexo se describe la lógica llevada a cabo para la codificación del driver y como se ha implementado. La finalidad principal del driver en el centro remoto únicamente es la de enviar consignas al robot cada periodo de muestreo (Ts) y de almacenar las consignas devueltas por este. El código realizado en C/C++ parte de la plantilla dada por Matlab [Template s-funcion Level-2, 2019] para la creación de una s-function, donde vienen definidas unas funciones tipo que definen ciertas funcionalidades dentro de las rutinas de atención.

En la figura 66 se muestra el flujograma con la estructura del driver y la lógica que se ha seguido para su diseño.

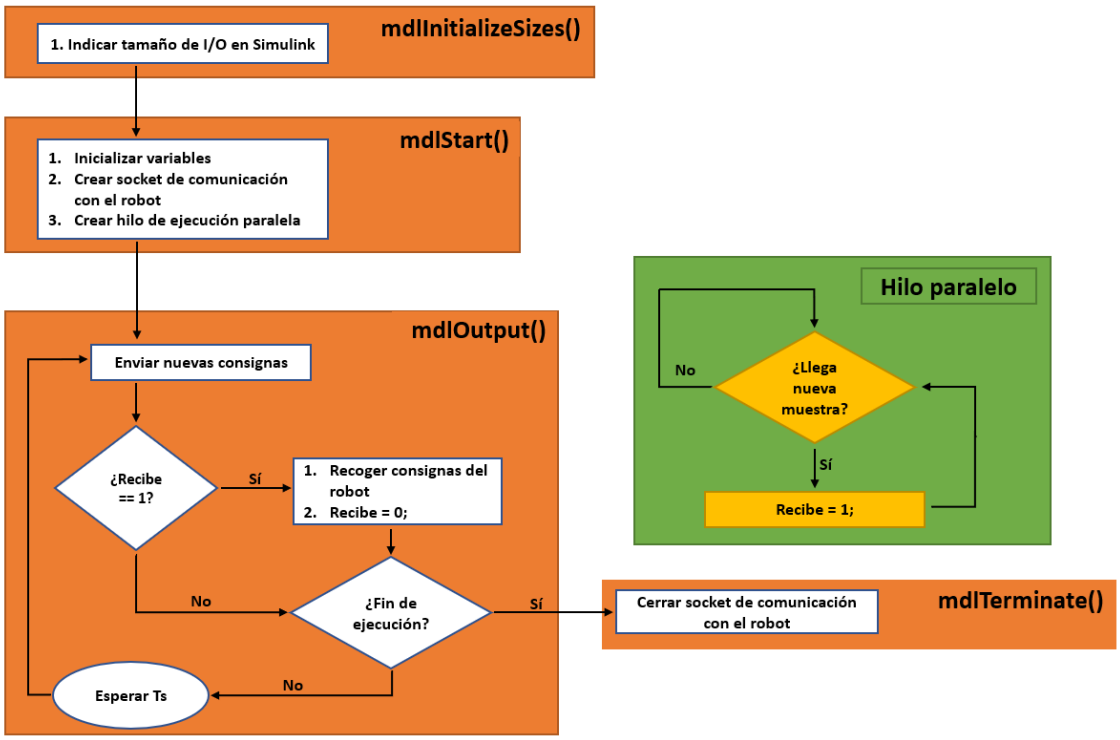


Figura 67. Anexo II: Flujograma de la codificación del driver para el centro remoto

En color naranja se indican las funciones definidas por Matlab en la plantilla y en color verde se muestra una función creada a parte dentro del mismo código fuente.

En la primera función: mdlInitializeSizes(), se comienza indicando el número de entradas y salidas que tiene la s-function. Una vez definido el tamaño del buffer de entrada y salida se ejecuta la función mdlStart(), donde se inicializan las variables a utilizar durante la ejecución del programa, se crea el socket de comunicación con el robot y se da origen a un hilo de ejecución paralelo, cuya funcionalidad será explicada más tarde. Cabe decir que esta función sólo se ejecuta una vez, al comienzo. Después se pasa a la función iterativa de la ejecución: mdlOutput(), donde se lleva a cabo la lógica descrita en la figura

anterior. Hasta que no pasa el periodo de muestreo marcado no vuelve a ejecutarse. El número de veces (N) que se ejecuta esta función depende del tiempo de ejecución total y del valor del periodo de muestreo. La ecuación que rige el número de iteraciones a realizar es:

$$N = \frac{TIEMPO_{ejecucion}}{T_s}$$

Finalmente, una vez realizadas todas las iteraciones marcadas, se realiza la última función, cuyo cometido en estos ensayos es el cierre de la conexión socket con el robot.

Respecto al hilo paralelo, es una función que se ejecuta constantemente, sin importar el valor del periodo de muestreo marcado, y que realiza la escucha mediante una función de comunicación bloqueante. Cuando recibe un dato por el socket abierto con el robot activa un flag que indica que un nuevo dato ha sido recibido, y dicho flag es valorado en la función periódica del programa principal que es el encargado de almacenar los datos recibidos.

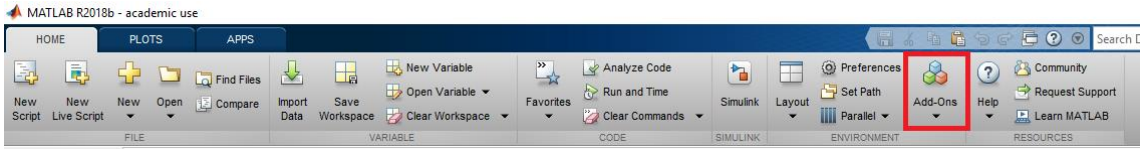
7. Anexo III: Manual de instalación y configuración del compilador para Windows: MinGW-w64

Cuando se trabaje sobre el sistema operativo Linux no es necesaria la instalación de un compilador adicional, ya que el compilador gcc viene instalado por defecto en este SO, y es soportado por Matlab. Pero a la hora de querer interactuar empleando Windows, es necesario saber qué se debe tener instalado, en el equipo que va a crear el ejecutable, el compilador MinGW. Este compilador es el que recomienda el mismo software de Matlab como se describe en el siguiente enlace: https://es.mathworks.com/matlabcentral/fileexchange/52848-matlab-support-for-mingw-w64-c-c-compiler

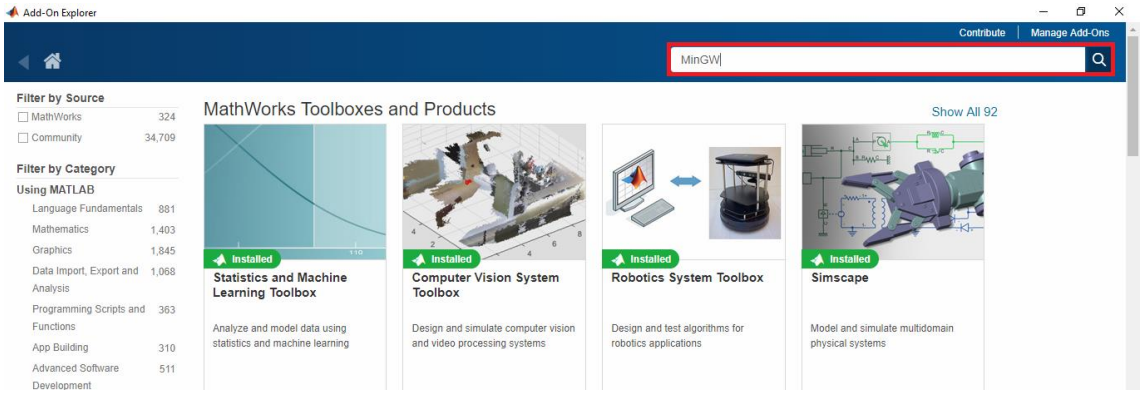
Más concretamente será necesaria la instalación de la versión 6.3 como mínimo, ya que para versiones posteriores a Matlab 2017 es lo recomendado. Para la instalación del software, se ha realizado una pauta de pasos a seguir para conseguir dicho fin. El usuario dispone de información acerca del compilador en la página web del distribuidor del software: http://mingw-w64.org. Si el usuario desea hacerse con alguna versión en concreto del compilador, puede emplear el ejecutable de instalación que se pone a disposición, bajo el nombre de "mingw-w64-install.exe", o bien siendo descargado desde: https://sourceforge.net/projects/mingw-w64/files/, aunque si no se tiene experiencia instalando software no es recomendable realizar este proceso, y completar la instalación siguiendo los pasos detallados a continuación:

Para la instalación del compilador es necesaria una conexión a Internet, ya que la obtención del Add-On se realiza mediante una descarga a través de Matlab.

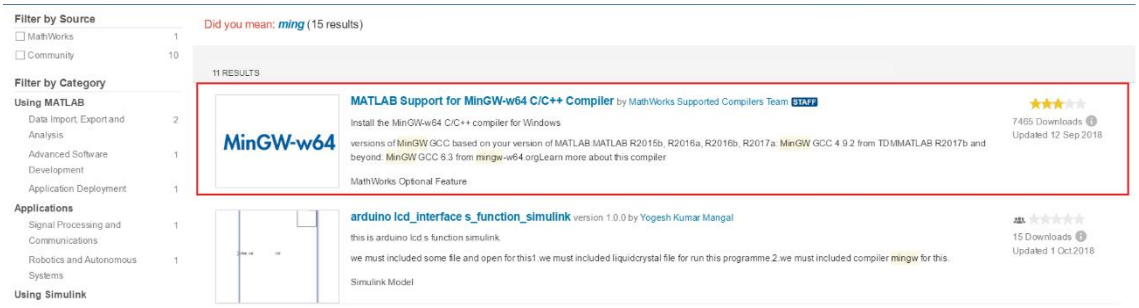
1. Abrir el entorno Matlab.
2. Dirigirse a la sección de Add-Ons, clicando en el icono que se muestra en la parte superior de la página principal de Matlab.



3. Una vez clicado sobre el icono, se abrirá la ventana correspondiente al explorador de Add-Ons, donde Matlab permite descargar e instalar extensiones propias que sirven para diferentes áreas de estudio. En este caso se debe buscar el compilador, para ello basta con introducir "MinGW" en el buscador como se muestra en a continuación:

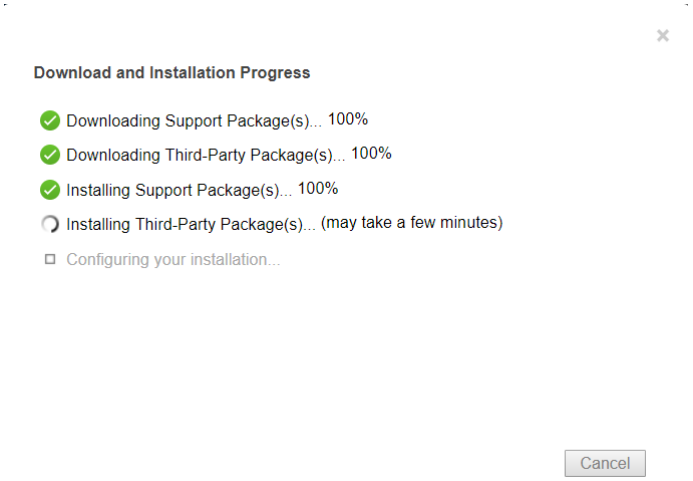


4. Una vez realizada la búsqueda, se debe clicar sobre la extensión que recibe el nombre de "Matlab Support for MinGW-w64 C/C++ Compiler". En la siguiente imagen se muestra la extensión a la que se está referenciando.

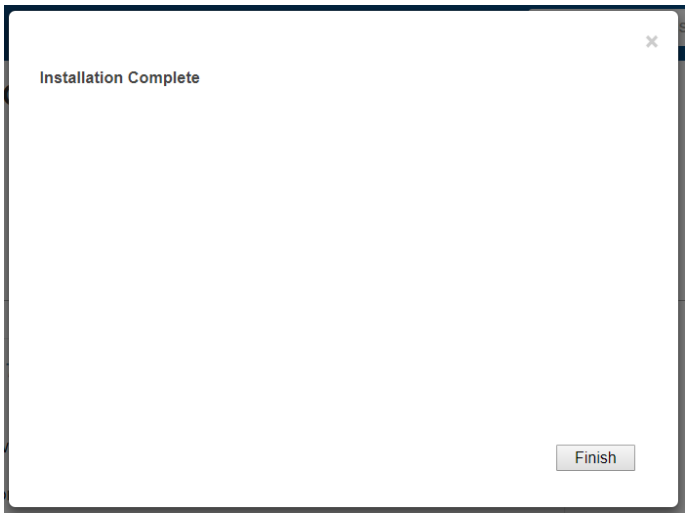


5. Tras ello, aparecerá una ventana de información sobre la extensión que se desea instalar. Únicamente es necesario clicar sobre la opción de instalar, marcada con un recuadro rojo.

6. Al hacer click sobre el botón anterior, automáticamente se iniciará la descarga y posterior instalación del compilador como extensión de Matlab.



7. Finalmente, una vez finalizada la instalación de la extensión, pinche sobre la tecla "Finish" para completar el proceso.



Para la comprobación de la correcta instalación del compilador, es necesario abrir el software Matlab e introducir en la consola de comandos lo siguiente:

```
mex -setup
```

Con este comando se indica a Matlab que debe utilizar el compilador MinGW, anteriormente instalado, para que lo utilice cuando sea necesario. Para más información acerca de este comando ver [Mex, 2018].

Una vez que Matlab ha reconocido correctamente al compilador, y este ha sido seleccionado para ser usado por defecto, al introducir el comando anterior debe alertar un mensaje por la consola de comando que indique lo siguiente, confirmando así el éxito en la instalación y configuración del compilador en el dispositivo a emplear:

```
>> mex -setup
MEX configured to use 'MinGW64 Compiler (C)' for C language compilation.
Warning: The MATLAB C and Fortran API has changed to support MATLAB
     variables with more than 2^32-1 elements. You will be required
     to update your code to utilize the new API.
     You can find more information about this at:
     https://www.mathworks.com/help/matlab/matlab_external/upgrading-mex-files-to-use-64-bit-api.html.

To choose a different language, select one from the following:
 mex -setup C++
 mex -setup FORTRAN
```

A continuación, se muestra la realización de un sencillo ejemplo, con un periodo de muestreo de 5ms, donde se compila un código fuente programado en C, que será implementado sobre una s-function de Simulink, que realiza la función de un amplificador de ganancia 5 respecto a una referencia de entrada. El ejemplo trata sobre una señal sinusoidal que atraviesa la lógica descrita y así comprobar la funcionalidad de la s-function.

Se comienza con la apertura del entorno de Matlab y designando una variable que haga referencia al periodo de muestreo que se va a imponer en la aplicación, en este ejemplo se tiene una variable de valor 5ms. También, antes de comenzar con la construcción del modelo se debe compilar el código fuente realizado, para ello, mediante el comando descrito en la introducción del apartado 3.3 de este documento ("mex") se puede compilar dicho código. Para el ejemplo que se está tratando, se tiene un código fuente programado en C, que hará la labor de recoger una entrada, multiplicar por 5 su valor, y devolverla. Cada iteración de esta lógica se debe repetir cada 5ms, ya que así ha sido establecido. Para compilar dicho archivo basta con introducir en la consola de comando o mediante la ejecución de un script, la siguiente sentencia:

```
mex ganancia_dr.c
```

siendo "ganancia_dr.c" el archivo a compilar.

Una vez compilado el código, nos dirigimos al entorno Simulink para crear el modelo. La diferencia respecto al ejemplo visto en el Anexo I es que, en vez de aplicar la amplificación a la referencia mediante un bloque de ganancia, ahora se va a realizar introduciendo la entrada a un bloque s-function que contiene la lógica programada en el archivo anteriormente compilado. El resultado del modelo es el siguiente:

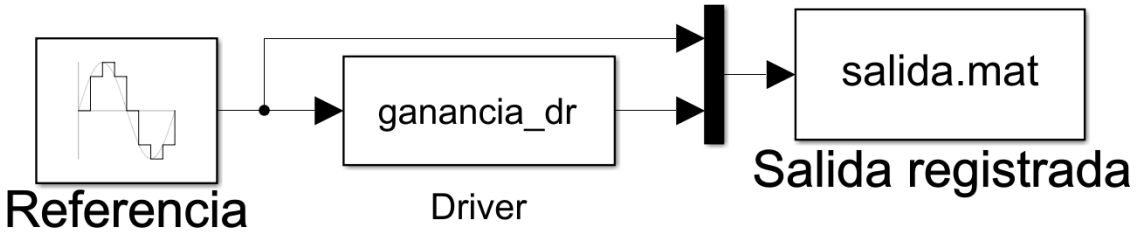


Figura 68. Anexo III: Modelo del ejemplo en Simulink

Una vez llegados a este punto, se debe configurar el entorno de Simulink para indicar que se quiere generar una aplicación que se ejecute en tiempo real, para ello se aplican las instrucciones descritas en el Anexo I.

Configurados los parámetros necesarios, se emplea el atajo Ctrl + B para construir el modelo y generar el ejecutable.

Finalmente, cuando se haya obtenido la aplicación, tan solo basta con dirigirse al directorio de trabajo y clicar dos veces sobre el icono del ejecutable. En la programación de la s-function, se pueden poner instrucciones que reporten distintos datos de interés. En este ejemplo, se ha impuesto que cuando la aplicación haya finalizado su ejecución, reporte por la pantalla de comando de la aplicación el tiempo real total que se ha estado ejecutando la aplicación, dando como resultado la siguiente expresión:

```
**starting the model**
**stopping the model**

**Simulation finished**


 El tiempo de ejecucion real ha sido de 5.012441 segundos



Pulse ENTER para salir
```

Figura 69. Anexo III: Información sobre el tiempo de ejecución en el ejemplo

El resultado obtenido se muestra en la siguiente figura, donde se observa que la señal de referencia introducida a la s-function es amplificada 5 veces tal y como se había programado anteriormente:

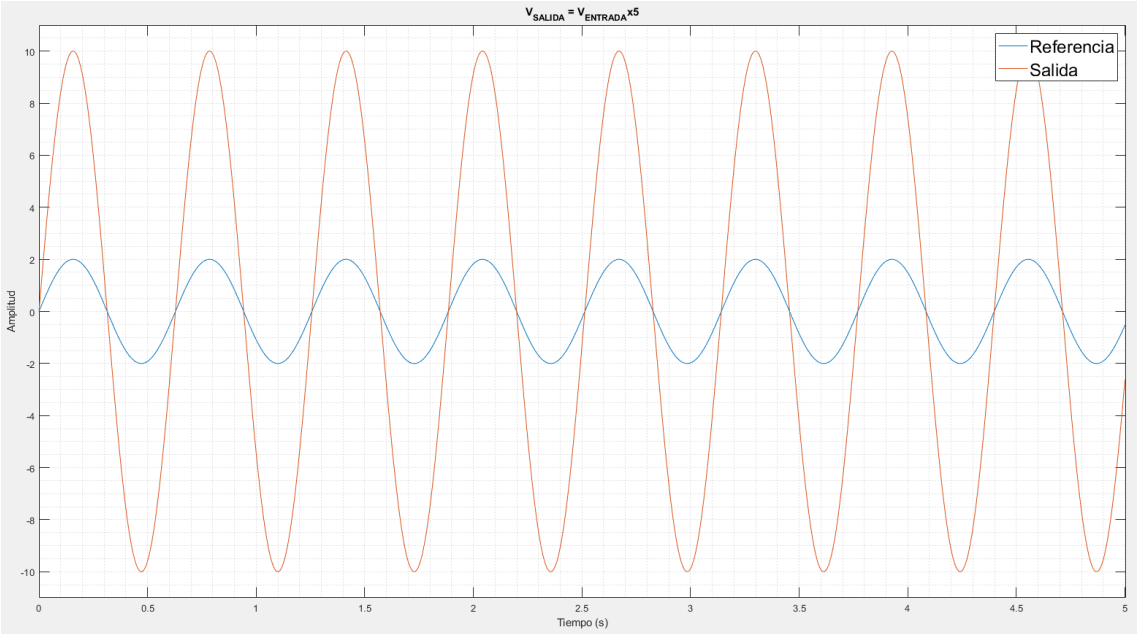


Figura 70. Anexo III: Resultado de ejecución a tiempo real

V. BIBLIOGRAFÍA

[Salazar, 2011] M. Salazar. "Integración de Matlab/Simulink/RTW y Linux/RTAI para aplicaciones de control y comunicaciones en robótica". Trabajo Fin de Carrera. Ingeniería en Electrónica. EPS. UAH. 2011.

[Santos, 2010] C. Santos. " Propuesta de control descentralizado con solapamiento para guiado en convoy de unidades P3-DX". Trabajo Fin de Carrera. Ingeniería en Electrónica. EPS. UAH. 2010.

[Arco, Alarcos, Domingo, 1998] "Programación de aplicaciones en redes de comunicaciones bajo entorno UNIX". Ed. Servicio de Publicaciones de la UAH

[Malisoff, 2009] Malisoff, M. and Mazenc, F. "Constructions of strict Lyapunov functions". ISBN 978-1-84882-534-5, DOI 10.1007/978-1-84882-535-2. Springer-Verlag, 2009

[Antsaklis, 2007] Antsaklis, P.J. and Michel, A.N. "A linear systems primer". ISBN-13:978-0-8176-4460-4. Ed. Birkhauser Boston.

[Slotine, 1991] Slotine, J.E. and Li, W. "Applied nonlinear control" ISBN: 0-13-040890-5. Prentice-Hall.

[Amoozgar, 2012] Amoozgar, M.H. and Zhang, Y.M. "Trajectory tracking of wheeled mobile robots: a kinematical approach" Proceedings of 2012 IEEE/ASME 8th IEEE/ASME Int. Conf. on Mechatronic and Embedded Systems and Applications. 2012. DOI: 10.1109/MESA.2012.6275574

[Ogata, 1996] Ogata, K. "Sistemas de Control en Tiempo Discreto" 2ª Edición. Editorial: Prentice Hall. ISBN 0-13-034281-5

[Fireosoft, 2017] https://fireosoft.com.co/blogs/sistemas-operativos-mas-usados/

[Geeksforgeeks, 2018] https://www.geeksforgeeks.org/udp-server-client-implementation-c/

[Mex, 2018] https://es.mathworks.com/help/matlab/ref/mex.html

[Ubuntu, 2018] https://www.ubuntu.com/

[Linux, 2018] https://www.linux.org/

[Mathworks, 2019] https://es.mathworks.com/

[Template s-funcion Level-2, 2019] https://es.mathworks.com/help/simulink/sfg/writing-level-2-matlab-s-functions.html

Universidad de Alcalá