Universidad de Alcalá
Escuela Politécnica Superior

Máster Universitario en Ingeniería Electrónica

Trabajo Fin de Máster

Análisis y evaluación de herramientas para el diseño, implementación y ejecución en tiempo real de controladores digitales utilizando Matlab/Simulink 2021

Autor: Víctor Sevillano Gamarra

Tutor: Cristina Losada Gutiérrez

2022

UNIVERSIDAD DE ALCALÁ

ESCUELA POLITÉCNICA SUPERIOR

Máster Universitario en Ingeniería Electrónica

Trabajo Fin de Máster

Análisis y evaluación de herramientas para el diseño, implementación y ejecución en tiempo real de controladores digitales utilizando Matlab/Simulink 2021

Autor: Víctor Sevillano Gamarra

Tutor: Cristina Losada Gutiérrez

Tribunal:

Presidente: José Luis Martín Sánchez

Vocal 1º: Julio Pastor Mendoza

Vocal 2º: Francisco Javier Rodríguez Sánchez

Fecha de depósito: 7 de septiembre de 2022

Lyapunov Based control

Resumen

Este trabajo tiene como objetivo analizar y evaluar la posibilidad de realizar la ejecución en tiempo real de un controlador conectado de forma remota a un robot móvil empleando las herramientas de diseño de sistemas de control asistido por ordenador MATLAB/Simulink 2021a en un PC con un sistema operativo Windows. Para ello se han estudiado diferentes alternativas, y se desarrolla una solución que contempla el diseño de un controlador mediante esta herramienta para un caso real, un *driver* para la conexión con el *hardware* a controlar a través de la red Ethernet y la ejecución de toda la solución sobre un sistema real, en un PC Windows. Los resultados obtenidos han permitido validar la solución desarrollada. Finalmente se ha analizado la solución y se ha parametrizado el alcance de la misma para el uso de este procedimiento en casos similares.

Palabras clave: MATLAB-Simulink, UDP, Robot Movil, PX-D3,control trayectoria, espacio de estados, comunicación, tiempo-real, Windows.

Abstract

The objective of this work is to analyze and evaluate the possibility of performing the real-time execution of a controller remotely connected to a mobile robot using MATLAB/Simulink 2021a computer-aided control system design tools on a PC with a Windows operating system. For this purpose, there has been studied different alternatives, developing a solution that contemplates the design of a controller in Matlab/Simulink for a real case, a driver for the connection with the hardware to be controlled through the Ethernet network and the execution of the whole solution on a real system, on a Windows PC. The obtained results allows to validate the solution. Finally, that solution has been analyzed and its scope has been parameterized for the use of this procedure in similar cases.

Translated with www.DeepL.com/Translator (free version)

Keywords: MATLAB-Simulink, UDP, mobile robot, PX-D3,trajectory control, Space state, communication, real-time, Windows.

Índice general

Índice de figuras

Índice de tablas

Lista de acrónimos

CPU	Unidad Central de Procesamiento.
EPS	Escuela Politécnica Superior.
IP	*Internet Protocol*.
PC	Ordenador Personal.
TFM	Trabajo Fin de Máster.
UAH	Universidad de Alcalá.
UDP	*User Datagram Protocol*.

Capítulo 1

Introducción

1.1 Presentación

Hoy en día, la electrónica de control está presente tanto dentro del desarrollo industrial, como cada vez más en la vida cotidiana. Desde el estudio de la tracción de un automóvil , hasta los algoritmos de gestión de potencia de la red eléctrica, la ingeniería de control está experimentando un auge sin precedentes. Es por este motivo, que el desarrollo de algoritmos de control resulta una práctica muy importante tanto dentro del desarrollo académico de los titulados en ingeniería, como en el ámbito de la investigación.

Existe una gran diversidad de tipos de algoritmos de control, pero de forma general se puede resumir su desarrollo en las siguientes fases:

1. **Especificación del control**: requisitos de tiempos de respuesta,trayectoria, medidas disponibles, sobreoscilación máxima u otras decisiones de diseño del conjunto planta controlador.

2. **Modelado de la planta a controlar**: estudio del comportamiento físico del sistema que se desea controlar. Para ello será necesario excitar el sistema con diversos inputs para conocer el tipo de respuesta [1], en base al conocimiento de los fenómenos físicos que lo rigen.

3. **Diseño de la estructura del algoritmo de control**: basándose en las distintas tipologías, como estructuras de lazo cerrado, control de histéresis, o la téctica empleada como control en espacio de estados, uso de lógica difusa u otras técnicas

4. **Cálculo de los parámetros del controlador**: cálculo de parámetros de ajuste de los controladores que permitan controlar el sistema con la especificación del control dada y basándose en los límites y modelado del sistema.

5. **Simulación del sistema controlado**: desarrollar un algoritmo de prueba que permita verificar el controlador ante el modelo teórico del sistema de forma virtual.

6. **Implementación del código a ejecutar en el *hardware* del controlador**: construir o codificar el algoritmo de control sobre el *hardware* de realizará el control en el sistema real según el algoritmo diseñado y probado en las anteriores fases.

7. **Verificación empírica de los resultados**: si el modelado del sistema es fidedigno al sistema real, en esta fase se deberían obtener resultados similares a los de la fase de simulación.

[1] Lineal, no lineal para sistemas continuos o lógica si se trata de un control discreto

En la figura 1.1 se muestran gráficamente las diferentes fases del diseño de sistemas de control electrónico, que han sido previamente explicadas.

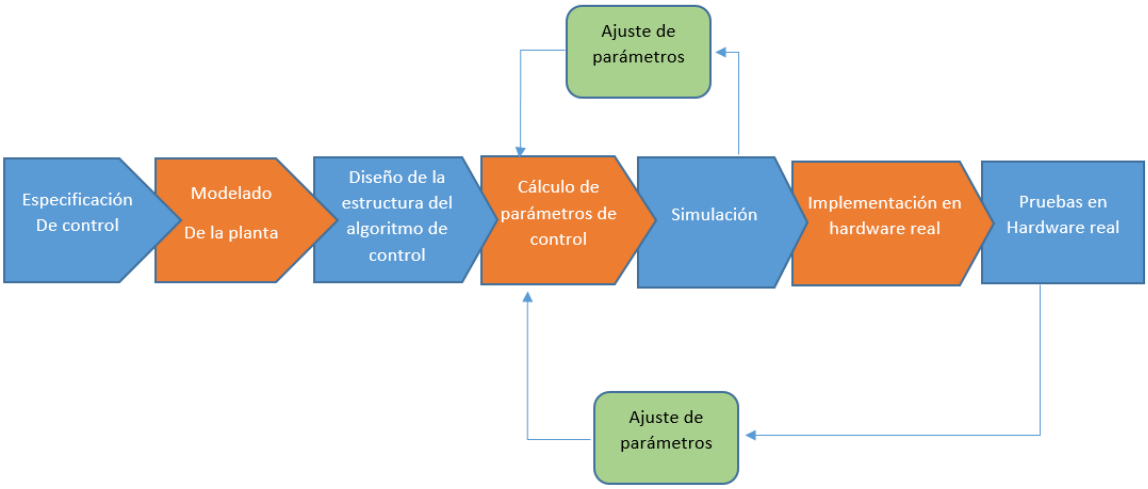


Figura 1.1: Fases en el desarrollo de un algoritmo de control.

Como se puede intuir, este proceso puede llegar a ser complejo ya que muchos factores entran en juego. Es por ello que integrar estas fases y sus correcciones puede ser laborioso y en ocasiones poco intuitivo. Por lo tanto, en lo que respecta al ámbito docente, se buscan herramientas que sean capaces de integrar al máximo este proceso y hacerlo visual e intuitivo, pues ~~es~~ esta cualidad es la que va a permitir una mejor adquisición de conceptos a la hora de comprender los fenómenos físicos de un sistema y la teoría de control aplicada al mismo.

Por ello, en este Trabajo Fin de Máster (TFM) se han estudiado herramientas que permitan este tipo de desarrollo a través de software de programación visual [1], y con Implementación en plataformas, que en la actualidad sean accesibles para cualquier estudiante de Ingeniería electrónica en la Universidad de Alcalá (UAH).

Esto plantea una serie de retos cuya naturaleza se desarrolla a lo largo de este trabajo. Para comenzar, en el siguiente apartado se presentan los objetivos del TFM y el diagrama funcional del sistema desarrollado.

1.2 Descripción funcional

1.2.1 Caso de Estudio

El principal objetivo de este TFM es realizar el análisis y evaluación de una alternativa que permita diseñar y ejecutar controladores en tiempo real, con sistema operativo basado en el sistema operativo Windows 10, como se ilustra en el diagrama funcional de la figura 1.2. Para la evaluación experimental, se realiza el control no lineal (basado en Lyapunov) de la trayectoria de un robot móvil. De esta forma, se busca poder llevar a cabo las diferentes etapas del diseño de sistemas electrónicos de control desarrollar controladores con las herramientas *software* y *hardware* disponibles para estudiantes de grado y máster en la UAH, lo que permitirá su utilización en la docencia de asignaturas de Control Electrónico.

Para alcanzar el objetivo, se ha valorado el uso de herramientas de diseño de sistemas de control asistido por ordenador de programación basadas en modelos como MATLAB/Simulink, las cuales son capaces de automatizar los procesos de generación de código, simulación y ejecución en el mismo entorno. Así pues, se centra el esfuerzo de los usuarios en el diseño de controladores, objeto de estudio de las asignaturas de teoría de control.

Además del análisis y evaluación de las herramientas *software*, el robot que se desea controlar, es un robot móvil, el cual deberá recibir unas señales de acción y enviar variables de salida. Estas señales, se han de recibir y enviar mediante una comunicación inalámbrica basada en el protocolo de comunicación UDP desde un Ordenador Personal (PC) en el que se ejecuta el sistema de control de trayectoria. Es por ello que durante el TFM también ha sido necesario implementar estos protocolos de comunicación entre el controlador (que como se ha comentado, se ejecuta en el PC con Windows 10) y el Robot.

En este contexto, en el presente TFM se ha desarrollado tanto una solución para la ejecución de controladores en tiempo real (o próximo al mismo) en MATLAB/Simulink, como las comunicaciones y la interfaz con el robot y un proceso de verificación paralelo.

Este TFM supone una actualización de a versiones más actuales de MATLAB del trabajo desarrollado por el trabajo de fin de grado por la UAH de Javier Robledo Arévalo [2], durante el cual se expone la generación de controles para la misma infraestructura mediante MATLAB/Simulink y su ejecución con versiones antiguas de MATLAB.

1.2.2 Diagrama funcional

A continuación, en la figura 1.2 se muestra el diagrama funcional del sistema que se desea implementar:

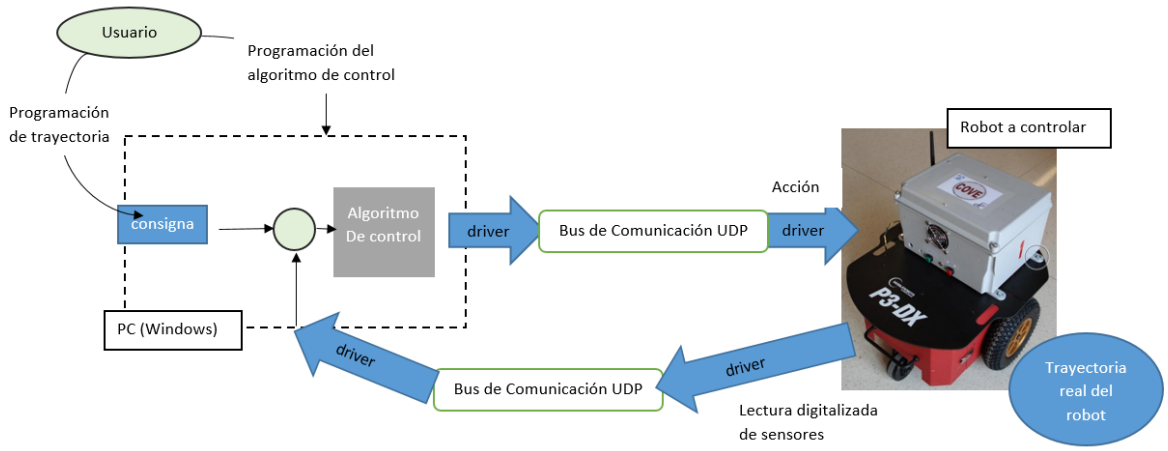


Figura 1.2: Diagrama funcional del montaje.

De esta forma, todo el *hardware* de control debe ejecutarse en el PC, basándose en las herramientas proporcionadas por MATLAB/Simulink y los drivers de la comunicación UDP con el robot, como se ha explicado hasta el momento.

Por otro lado, se debe indicar que el robot solo proporciona lecturas basándose en sus encoders, leyendo el número de vueltas por segundo de cada rueda. Este sistema de medida presenta un problema de precisión debido a los deslizamientos de ruedas, ya que este sistema no contabiliza recorrido si la rueda

asociada no está girando y es el caso de los deslizamientos cuando el robot lleva una velocidad distinta a la de sus ruedas. En ocasiones, debido a esta desviación, la medida de velocidad obtenida por los encoders del robot puede ser insuficiente para obtener cierta precisión en un controlador. Por esa razón, para aminorar estos problemas, junto con el robot, existen una serie de cámaras ubicadas a lo largo el pasillo de laboratorios del Departamento de Electrónica,en la Escuela Politécnica Superior (EPS) de la UAH, que se disponen como en la ilustración 1.3;

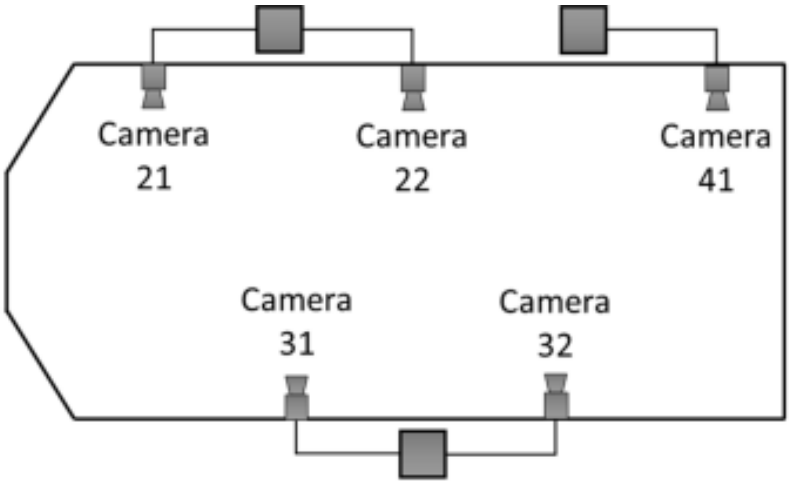


Figura 1.3: Disposición de cámaras en el pasillo del departamento.

Estas cámaras tienen como objetivo monitorizar la trayectoria del robot durante su funcionamiento, añadiendo información extra de la medida. Por además ello ha sido necesario proporcionar una interfaz de comunicación con estas cámaras y el PC del controlador.

1.2.3 Organización y fasesdel trabajo

El presente trabajo se ha dividido en las siguientes fases:

1. **estudio teórico:** en este apartado se presentan las distintas alternativas *software* para el desarrollo de este TFM, así como la bibliografía en la que se basa ylos conceptos de ejecución y programación necesarios para el posterior desarrollo 2.

2. **desarrollo:** en este apartado se presenta el diseño tanto del controlador,como de los drivers y la solución aportada 3.

3. **resultados:** en este apartado se analizan los distintos resultados obtenidos tras ejecutar los distintos test sobre la solución 4.

4. **resultados:** finalmente se exponen las conclusiones y posibles implementaciones futuras 5.

Capítulo 2

Estudio teórico

2.1 Introducción

En este capítulo, se exponen los fundamentos teóricos que han llevado al planteamiento de la solución, así como los trabajos más relevantes en los que este estudio teórico se fundamenta.

2.2 Estado del Arte

Este trabajo, cuenta con numerosos títulos y artículos de bibliografía en los que se ha bsado el análisis y al solución del mismo. Sin embargo, de entre todos los títulos y trabajos, es preciso realizar una especial mención a dos de ellos.

- Trabajo de fin de grado por la UAH de Javier Robledo Arévalo [2]
- *Remote Control of a Robotic Unit: A Case Study for Control Engineering Formation* [3]

En ambos estudios, se plantean casos de estudio relacionados con el robot P3-DX y su control con una aproximación de MATLAB/Simulink.

En el primer caso [2] se plantea un escenario muy similar al de este trabajo, pues se requiere una solución para comunicar paquetes de datos con el sistema de cámaras y del robot P3-DX con un PC para la ejecución de un controlador en tiempo real. Para ello se plantea una Solución basada en MATLAB/-Simulink y en el *coder* de Simulink en su versión de 2018.

En ese trabajo se planteaba una solución que pasaba por implementar el *driver* de comunicaciones con el sistema mediante el uso de una *S-Function* de Simulink de forma programática en código C, haciendo uso de las funciones de *Socket* de Windows (ver figura 2.1).

De esta forma se obtiene un *driver* que puede interactuar con el modelo. Sin embargo esta solución solo era funcional una vez que se generaba el ejecutable, no en tiempo de simulación. Para utilizar esta funcionalidad, se hace necesario la generación de código ejecutable en tiempo real del controlador más los bloques que generan las referencias.

Sin embargo, en las últimas versiones de Simulink, el *coder* ya no es funcional para la mayor parte de los bloques de generación de señal por lo que se hace necesario buscar una solución alternativa a este problema [4].

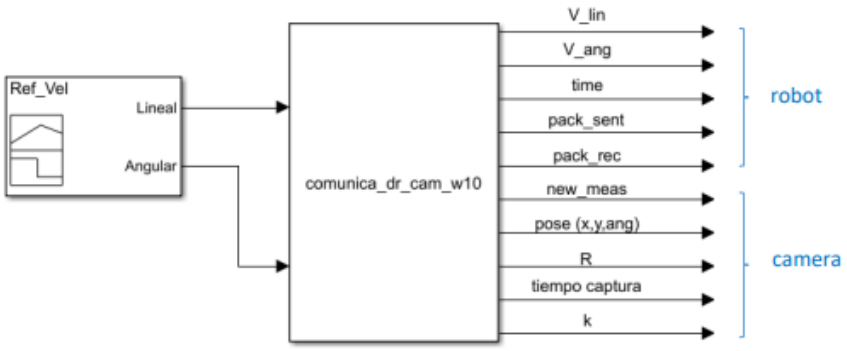


Figura 2.1: Bloque driver utilizado para generar código ejecutable en tiempo real de controladores del sistema del robot P3-DX y cámaras para versiones anteriores a MATLAB/Simulink 2021a

El siguiente trabajo expuesto [3] en este apartado, plantea el problema de modelado físico del robot P3-DX. En este título, se ha basado el análisis de la infraestructura del laboratorio. Por otro lado, también el apartado de modelado del robot y tras ello el diseño del control de velocidades y trayectorias planteado.

2.3 Alternativas para la programación basada en modelos

Los beneficios y versatilidad de la programación basada en modelos [1], como ya se ha estudiado son claros:

- Integración del proceso de diseño e implementación de controladores
- Aceleración de los procesos de desarrollo de controladores
- Facilidad a la hora de localizar errores
- Dificultad de aprendizaje del lenguaje cuasi nula

Sin embargo, existen en mercado diversas alternativas que soportan este tipo de programación. Sus herramientas integradas, así como su alcance difieren entre los distintos *softwares* en el mercado. De esta forma también lo hace la estructura esquemática que utilizan y la funcionalidad de los distintos bloques y aspectos como el tipo de distribución de *software* (principalmente como código Propietario o código libre). Existen también diversos *softwares* de diseño y simulación basada en modelos, específicos para topologías de sistemas concretas entre los cuales podríamos nombrar como ejemplo PSIM para sistemas electrónicos de Potencia o FluidSIM para simulación de sistemas hidráulicos. No obstante, debido al enfoque específico de estos *software*, solo se analizarán alternativas con un enfoque más general, de forma que sea extrapolable al mayor número de casuísticas de control. Tras una investigación bibliográfica, se pueden destacar las siguientes alternativas, las cuales podrían ser utilizada para conseguir el objetivo de este trabajo.

2.3.1 Scilab Xcos

Scilab [5] se presenta como un *software* de código abierto GLP (General public license) [6] , desarrollado por el grupo ESI. Este *software* integra una herramienta principal dedicada a la programación de cálculo numérico de alto nivel de código, similar a otras herramientas extendidas como MATLAB u Octave. Esta parte puede resultar interesante a la hora del cálculo de los parámetros de controladores para el diseño

de un algoritmo de control, pero por otro lado este programa cuenta con una interfaz de diseño basado en modelos denominada X-cos [7] mediante la cual se pueden diseñar, implementar y simular algoritmos de control y sistemas de diversas tipologías físicas.

Figura 2.2: Pantallazo de una simulación en en el entorno Xcos dentro de Scilab

Por otro lado, tanto Scilab como Xcos cuentan con un gestor de librerías y Toolboxes [8] desarrolladas por Scilab llamado ATOM. Estas toolboxes añaden librerías de código para la programación numérica así como diversos bloques con funcionalidades específicas ya implementadas y documentadas en APIs. El catálogo de Toolboxes desarrolladas es extenso y es posible crear Toolboxes y añadirlas a Scilab de forma pública para todos los usuarios de Scilab, o privada, solo uso local. Para El ejemplo del control del robot, sería posible utilizar Las toolboxes de tiempo real [8], toolboxes que puedan facilitar la programación del control o alguna toolbox de sockets para las comunicaciones con el *hardware* a controlar. Scilab cuenta con una herramienta externa encargada de generar código C a partir de un modelo de Scilab como de Xcos y usarlo en el *hardware* Target definido.

Pese a que esta alternativa es muy completa, gratuita y realmente compatible con otras similares muy extendidas, esta no cuenta con una documentación tan exhaustiva para gran parte de las funcionalidades. Por otro lado, para modelos muy extensos la interfaz de Xcos puede resultar demasiado minimalista ya que no cuenta con paletas de debuggeo ni inspección del modelo.

2.3.2 LabView

Entre las alternativas más comunes, también suele presentarse LabVIEW como un *software* extendido disponible para usos de programación gráfica. LabVIEW es un entorno diseñado exclusiva y específicamente para el desarrollo de programación gráfica por National Instruments [9] con licencia de Código propietario [6].

A diferencia de otros de los *software* de programación gráfica, LabVIEW no cuenta con una interfaz adicional de codificación en lenguaje estructurado textual. Esto es porque LabVIEW es un lenguaje orientado a tareas de instrumentación, control y validación basándose en las herramientas *hardware* de National Instruments y terceros, lo cual puede plantear una desventaja frente a otras alternativas a la hora de integrar las fases de diseño, cálculo y simulación en el mismo entorno. Pese a este aspecto y citando la página web del desarrollador, LabVIEW puede interaccionar con otros lenguajes de programación como MATLAB [9, 10]. Entre las tareas que se pueden desarrollar con LabVIEW está el modelado de sistemas dinámicos, la implementación del control, simulación de sistemas y la conexión con *hardware* externo

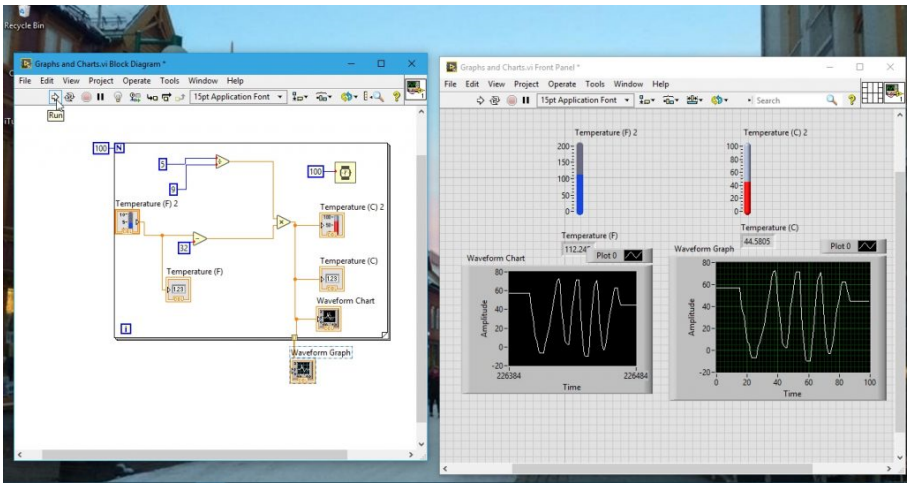


Figura 2.3: Pantallazo de una simulación en LabVIEW

desde el PC de forma eficiente y muy optimizada para la adquisición de datos. Sin embargo, este *software* es de pago y no cuenta con una interfaz de cálculo lo cual puede llegar a ser un problema a la hora de integrar todo el proceso de diseño e implementación del controlador de una planta.

2.3.3 MATLAB/Simulink

MATLAB es un *software* de programación matemática y cálculo de código propietario [6, 11] desarrollado por *MathWorks*, y ampliamente extendido entre el mundo académico y profesional. Esto es debido principalmente a su capacidad de manejo de datos, su optimización en el cálculo matricial, así como la leve dificultad operativa del lenguaje propio de MATLAB, lo hacen ideal para el desarrollo de cálculos complejos de forma rápida e intuitiva [12].

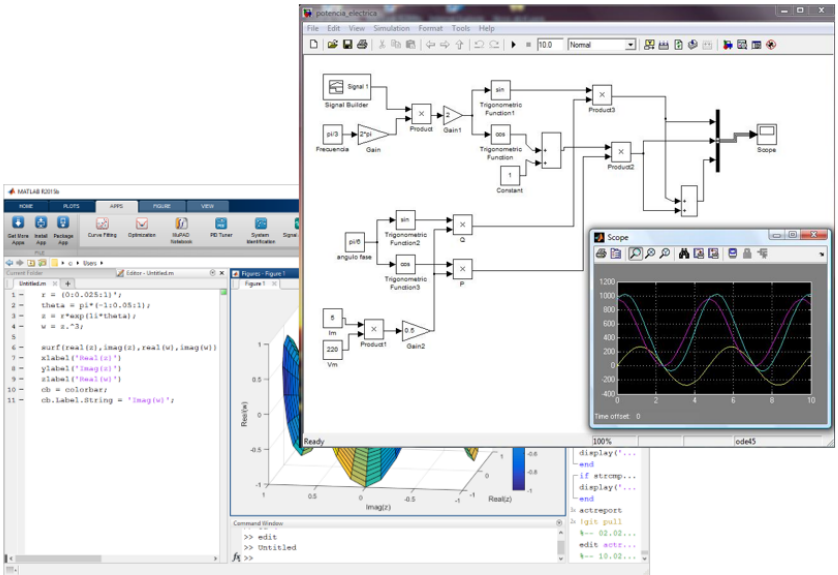


Figura 2.4: Simulación de un modelo en la perspectiva de Simulink y editor de código MATLAB

Así como Scilab incluye la interfaz XCos, MATLAB incluye Simulink 2.4 [13], una interfaz dedicada específicamente a la programación gráfica basada en modelos. Esta interfaz es totalmente compatible

con los parámetros y cálculos realizados en MATLAB y permite la simulación basada en el modelo matemático de sistemas dinámicos, el diseño y verificación de algoritmos de control (Pruebas MIL, SIL and HIL [14]), además de la generación de código C y ejecutables de los controladores u otros modelos trazados para distintas plataformas y *hardware* de *target*. Tanto MATLAB como Simulink cuentan con una gran variedad de toolboxes y librerías que implementan diversos bloques y funciones desarrolladas por MathWorks, entre ellas las dedicadas a las comunicaciones con *hardware* externo, y herramientas de generación y compilación de código C, con extensa documentación y APIs. Estas toolboxes son de código cerrado, algunas de ellas gratuitas y otras de pago [12]. Pese a que esta alternativa no resulta la más económica, si es muy completa sencilla, y cuenta con un amplio uso ya dentro de los estudios de Ingeniería electrónica cursados en la UAH y otros centros [11].

2.3.4 OpenModelica

OpenModelica, es un *software* desarrollado por Open Source Modelica Consortium (OSMC) [15] como Public-License [6] desarrollado de forma colaborativa Y concebido para la simulación de sistemas dinámicos de cualquier naturaleza física. Al igual que sucede con LabVIEW, OpenModelica no cuenta con una interfaz de cálculo, pese a ser compatible con otros lenguajes de programación extendidos como Python.

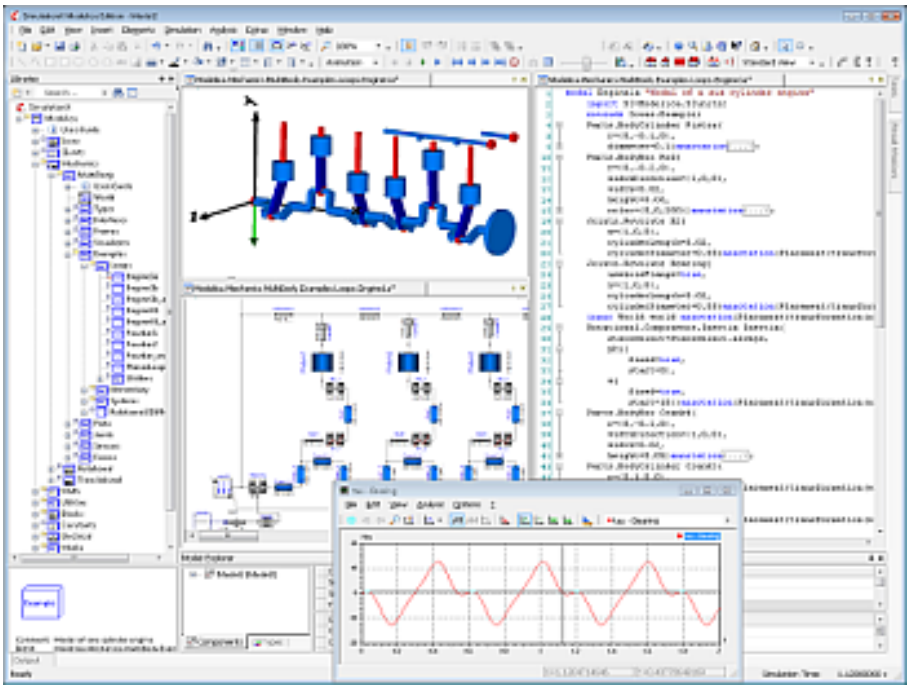


Figura 2.5: Simulación de un modelo mecánico en OpenModelica. Se observa la interfaz de diseño,la simulación así como la interfaz textual de definición de modelos

OpenModelica cuenta con una interfaz de diseño del modelo como en los *softwares* anteriormente presentados y otra de definición del modelo 2.5 de los componentes o bloques [16]. De esta forma cada bloque o componente cuenta con una definición interna del modelo matemático de las ecuaciones del componente en código propio de modélica. De esta forma para modificar algún parámetro o comportamiento de algún componente, será preciso modificar esta, lo cual comparándolo con otras alternativas puede resultar poco intuitivo y alejado del concepto de programación basada en modelos.

OpenModelica es un *software* muy eficiente a la hora de modelar y simular sistemas, así como implementar y verificar controladores simulados. Sin embargo, carece de herramientas propias para generación

de código, comportamiento en tiempo real o comunicación con *hardware* externo [17] (existen pluggins desarrollados por terceros para este fin sin certificación y complejos).

2.3.5 Comparativa

Tras este análisis se obtienen que de entre todas las opciones disponibles,se ha utilizado el *software* de MATLAB/Simulink, ya que pese a ser la herramienta menos económica, es de las cuatro alternativas la más completa. En ella se incluye tanto Simulink como el editor y terminal de código MATLAB lo cual supone una gran ventaja frente a otras alternativas, ya que esto permite desarrollar cálculos dentro el mismo entorno con formatos y funciones compatibles entre MATLAB y Simulink.

Por otro lado, MATLAB es un entorno muy extendido dentro de toda la comunidad académica e industrial y la licencia está financiada por la Universidad de Alcalá de Henares [11].

2.4 Aspectos teóricos de la ejecución en tiempo real

Uno de los requisitos para la implementación de este trabajo, es obtener un método de ejecución de un controlador digital de lazo cerrado (figura 2.6 que se ejecute sobre un sistema con *Windows* como sistema operativo en MATLAB/Simulink, de forma que el periodo de ejecución sea capaz de mantenerse dentro de una tolerancia admisible, ya que de lo contrario el control puede comprometerse como se explica a lo largo del siguiente apartado.

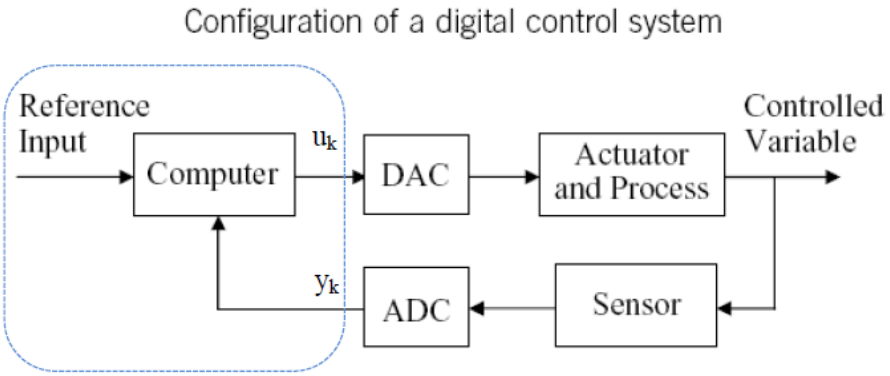


Figura 2.6: Esquema genérico de un sistema de control digital muestreado

Es aquí donde entra en juego el concepto de sistema de tiempo real. Un sistema de tiempo real se define como todo sistema computacional capaz de responder a un estímulo dentro de un plazo determinado, en este caso el plazo estará definido por la frecuencia de muestreo Ts

2.4.1 Efectos de la variación del periodo

En cualquier sistema dinámico continuo, la respuesta a una entrada que evoluciona en el tiempo puede escribirse en función de sus derivadas temporales 2.1.

$$p[q_t = \sigma_t | q_{t-1} = \sigma_{t-1}] \tag{2.1}$$

De la misma forma, si se trata de un sistema discretizado como es el caso de un controlador digital, la acción entregada por el controlador en el tiempo, es función del tiempo de muestreo 2.3, por lo que

si el tempo de muestreo varía pero el sistema discretizado no lo hace, la respuesta temporal variará en función del tiempo de respuesta

$$\frac{S(x,t)}{E(x,t)} = F(x, \frac{dx}{dt}, ... \frac{d^n x}{dt^n})] \tag{2.2}$$

Sin embargo en el dominio discreto el sistema responde según su Ratio de muestreo (Ts) y su modelo discretizado. En controlador en la condición de periodo variable, ejecuta el mismo sistema discretizado del controlador pero con distinta frecuencia de muestreo

$$S(x[k]) = F(E(x[k]....x[k-n]), x[k]....x[k-n], Ts)] \tag{2.3}$$

Desde el plano continuo, el modelo continuo de la planta no cambia ya que este no depende de la frecuencia de muestreo sino de las leyes físicas que rigen la planta. Sin embargo, el modelo continuo equivalente del controlador sí lo hace ya que es función del tiempo de muestreo. Por ello el sistema continuo equivalente global cambia si varía el tiempo de muestreo.

Este efecto indeseado supone un problema, ya que cuando se diseña n controlador, este se discretiza a una frecuencia de muestreo constante TS (en este caso de 25 ms) por lo que todo el comportamiento de la planta estará subordinado al tiempo de muestreo real.

Es por ello que un sistema controlado estable podría incluso llegar a ser inestable si se diera una frecuencia de muestreo que incumpliera alguna de las condiciones de estabilidad del sistema de controlador más planta de *Lyaunov* (sección 3.2.3).

2.4.2 Gestión de tiempos en Windows

Cuando se tratan problemas de tiempo real, en la mayor parte de los escenarios, la solución comienza por el uso de sistemas operativos brinden características de tiempo real. Estos sistemas operativos se encargan de organizar la ejecución de sus distintos procesos de forma que sea posible **cumplir plazos de ejecución marcados para cada tarea**.

En sistemas de esa topología como *FreeRTOS* [18], las tareas se ejecutan mediante ejecutivos cíclicos, prioridad de tareas y otros mecanismos de organización de tareas que pueden correr sobre la Unidad Central de Procesamiento (CPU) del sistema con el fin último de cumplir requisitos temporales de las mismas. En este caso sería lo ideal, ya que el sistema operativo podría forzar la ejecución del resto de tareas en instantes concretos para cumplir con la especificación de tiempo de muestreo constante.

Sin embargo, en este caso, la solución debe ejecutarse sobre una plataforma *Windows* [19], sistema operativo que presenta tareas concurrentes y cuyo fin a la hora de organizarlas no es el cumplimiento de plazos temporales definidos con exactitud, sino optimizar la organización de tareas para obtener **los mínimos tiempos de ejecución de cada tarea**. De esta forma pueden llegar a priorizarse tareas sin un especial interés temporal frente a otras que si lo tienen, simplemente por el hecho de que priorizarlas reduce el tiempo de ejecución total de todas las otras tareas.

Por esta razón a la hora de ejecutar un controlador en *Windows*, que el tiempo de muestreo de la comunicación se mantenga constante,va a depender del número de procesos activos en cada momento y la prioridad que tenga cada uno. Por ello la ejecución de la tarea de comunicaciones podrá tener prioridad sobre la de control de forma que puede comprometerse el tiempo de muestreo constante.

Pese a ello, Desde Windows 10 [20], las siguientes versiones de Windows implementan una librería de gestión de *deadlines* y gestión de eventos en tiempo real a bajo nivel. Sin embargo la experimentación de estas librerías no es el objetivo principal de este trabajo además de la complejidad que este uso supone.

2.5 Aspectos teóricos de la comunicación PC-Robot

2.5.1 Introducción

Según el diagrama funcional que se adjunta en la descripción funcional de este trabajo, el robot y el PC donde se ha de ejecutar el controlador se encuentran dentro de la misma red de comunicación de área local WLAN haciendo uso de protocolo Wi-Fi. Es por ello que para establecer un bus de comunicación entre el proceso del controlador, será necesario hacer uso del concepto de *socket* de comunicación para este tipo de redes. Un *socket*, es un objeto *software* cuya función es el manejo de *hardware* de la máquina encargado de establecer una conexión con un punto de acceso remoto dentro de la misma red. Una vez creado el *socket*, este será responsable de configurar el *hardware*, realizar y aceptar las solicitudes de conexión con el punto remoto específico, enviar y recibir las tramas de datos Ethernet de bajo nivel y filtrarlas a las variables del proceso.

Por lo tanto, los *sockets* requieren comunicarse de forma directa con el *hardware* mediante sus *drivers*. Es por ello que, en máquinas con Sistema operativo, los *sockets* están integrados en estos ya que es el sistema operativo el que se comunica con el *hardware*. Tanto el robot como el PC en el que se desea realizar la comunicación, llevan instalado un sistema operativo, en el caso del PC que es el caso que se pretende implementar de tipo Windows, por lo que para manejar sus *sockets*, será necesario el uso de la librería de *sockets* para Windows [21] , o algún elemento *software* que ya incluya de forma implícita el manejo de esa librería [22].

Como ya se comenta en detalle en el apartado de estado del arte 2.2, esta solución ya no es útil para las nuevas versiones ya que el coder de Simulink que requería ya no es funcional para bloques generadores de señal en las versiones de MATLAB/Simulink a partir de la v2021a.

No obstante, si ha sido necesario el uso de las librerías de *socket* a la hora de crear un entorno simulado en otro PC para verificar el comportamiento de tiempo real y las comunicaciones del modelo con un punto remoto externo.

2.5.2 Parámetros de la comunicación por *socket*

2.5.2.1 Parámetros de direccionamiento

Para establecer una conexión de Internet entre dos equipos, existen dos niveles de direccionamiento [23].

- *Internet Protocol* (IP) es un número de 32 bits, que identifica un equipo conectado a una red de Internet. Existen diversos niveles de IP dependiendo del alcance. Como todos los equipos tanto el PC como el robot y las cámaras están en la misma red local, la dirección IP (IPv4 asignada por la red WLAN) es única dentro de la red y será el identificador único a cada equipo.
- Puerto es otro nivel de direccionamiento. Cada equipo cuenta con su IP asignada por su red local, y a su vez cada equipo cuenta con diversos puertos. Por cada puerto se puede establecer un canal de comunicación distinto con otros equipos 2.7.

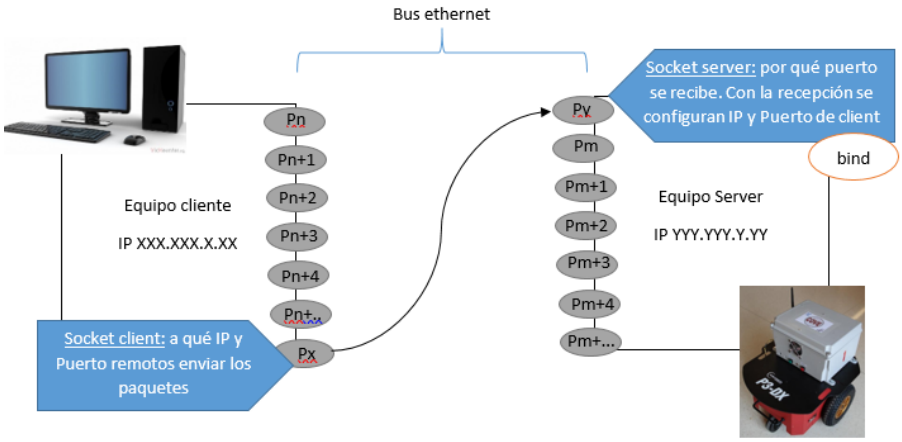


Figura 2.7: Explicación gráfica del concepto de puertos, IP y proceso básico de identificación entre los dos equipos a comunicar.

2.5.2.2 Protocolo *UDP*

El protocolo UDP [24] , es un protocolo de transmisión de datos mediante comunicaciones de *socket* de Internet, es decir el direccionamiento para abrir un canal de comunicación mediante UDP entre equipos, se realiza mediante el uso de direcciones IP y Puertos. El protocolo UDP no precisa de una conexión para enviar y recibir datos ya que la cabecera de cada dato contiene la información de la dirección de IP y Puerto del emisor.

En este protocolo a diferencia de en *TCP (Transmision Control Protocol)*, no es necesaria la confirmación de envío o lectura, por lo que si un dato no llega su destinatario, el envío puede continuar sin confirmación.

Es este concepto el que lo hace este protocolo factible para esta aplicación, ya que en caso de pérdida de datos, el controlador sigue enviando datos sin bloquearse, lo cual añadiría latencias indeseadas que se traducirían en retrasos y no sería posible la constancia en el tiempo de muestreo.

2.5.2.3 Roles en la comunicación

En la comunicación entre el PC que ejecuta el controlador, y el sistema del robot y las cámaras existen dos roles fundamentales en la comunicación [25].

- *Client* es el encargado de enviar una request para obtener algún dato del server
- *Server* envía la respuesta a esa request del client.

En el caso presente, el *Client* sería el controlador, es decir el PC que ejecute el controlador el que envíe la acción como *request* al Robot de forma que este se encargue de responder con la lectura de los sensores.

Igual que en el caso del *software* en el robot, las cámaras implementan un *software* de comunicación *Client* qu envía paquetes de datos al servidor, (el controlador) cuando existe una *request* enviada inicialmente por este.

2.5.3 Flujograma del uso de sockets

De esta forma se obtienen dos flujogramas distintos: el del cliente y el servidor.Por lo tanto los flujogramas de cada parte serán como los presentados 2.8

Como se puede observar,la recepción de un dato no es condición necesaria para el envío de otro nuevo dato. Igualmente el modelo de comunicación de las cámaras también se correspondería con el simple envío de paquete de datos de pose cuando esta tenga disponibilidad de los mismos.

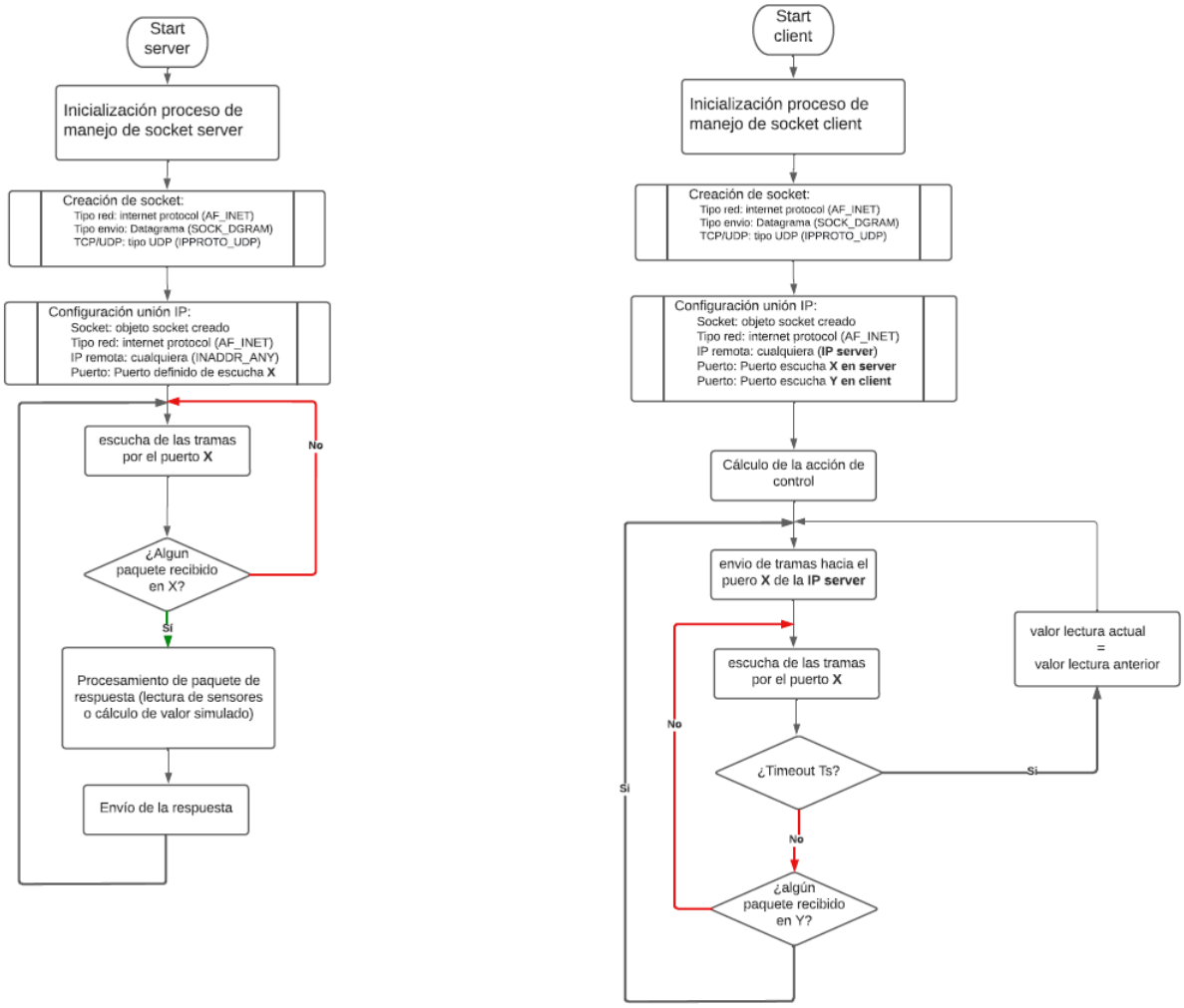


Figura 2.8: Algoritmo de envío y recepción de datos del robot o la cámara con el PC

Capítulo 3

Desarrollo

3.1 Introducción

En este capítulo se incluirá la descripción del desarrollo del trabajo, para lo cual se han necesitado diversas fases las cuales se pueden clasificar a grandes rasgos como las siguientes:

- Diseño, cálculo y verificación de los lazos de control
- Pruebas de comunicación simuladas
- Verificación de la comunicación con *hardware* real y lazo de control

3.2 Calculo e implementación del algoritmo de control utilizado

3.2.1 Modelado de la planta

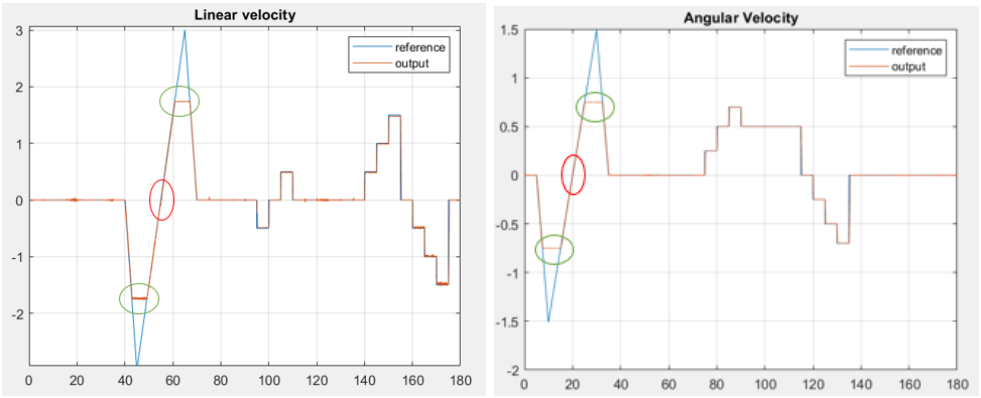
Como en todo problema de control, el primer paso para diseñar el control es modelar la planta, determinar las ecuaciones de su comportamiento físico y sus parámetros. Para ello, es necesario hacer una distinción entre dos tramos de funcionamiento dentro del control de la planta [26]:

- Tramos de funcionamiento lineal
- Tramos de funcionamiento no lineal

3.2.1.1 Modelado de tramos de funcionamiento no lineal de la planta

Para modelar los límites y comportamiento no lineal del robot, es necesario analizar los límites operativos de este:

- Retrasos, debidos a la comunicación ejecución y retardos en los sensores
- Zonas muertas, debida a las zonas de fricción estática cercana a la velocidad nula
- Zonas de saturación, debidas al par y velocidad máximos del robot



(a) Velocidad lineal. (b) Velocidad angular.

Figura 3.1: Referencias del ensayo realizado sobre el robot para modelar tramos de saturación (verde) y zonas muertas (rojo)

Para determinar estas zonas, se cuenta con un ensayo realizado sobre el robot con muestras de distintas trayectorias de referencia sobre el robot.

Como se expone en este ensayo (figura 3.1), el robot cuenta con unas zonas cercanas a velocidad lineal y angular cercanas a la velocidad 0, en las que la respuesta del robot no sigue la referencia debido a las zonas muertas debidas a la fricción estática. A continuación se observan zonas donde la respuesta no alcanza el valor de referencia debido a que se alcanza el límite máximo de la acción del robot.

Por último se puede estimar el retraso total en 0.2 segundos con un muestreo de 0.025 ms, lo que equivale a 8 ciclos de latencia (este fenómeno se observa entre el cambio ed la referencia y el comienzo de variación de la respuesta).

3.2.1.2 Modelado de tramos de funcionamiento lineal de la planta

Con un breve estudio sobre la cinemática [3] del robot se llega al diagrama de bloques de diagrama de bloques y ecuaciones continuas coorreladas, por lo que se modelará como un sistema en variables de estados 3.1.

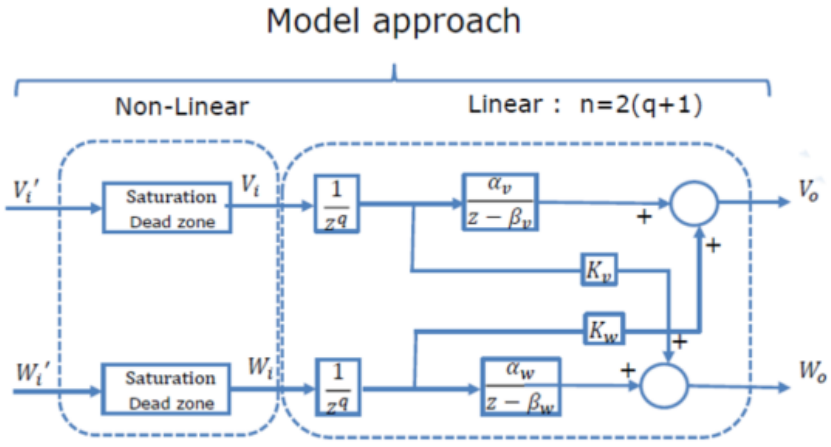
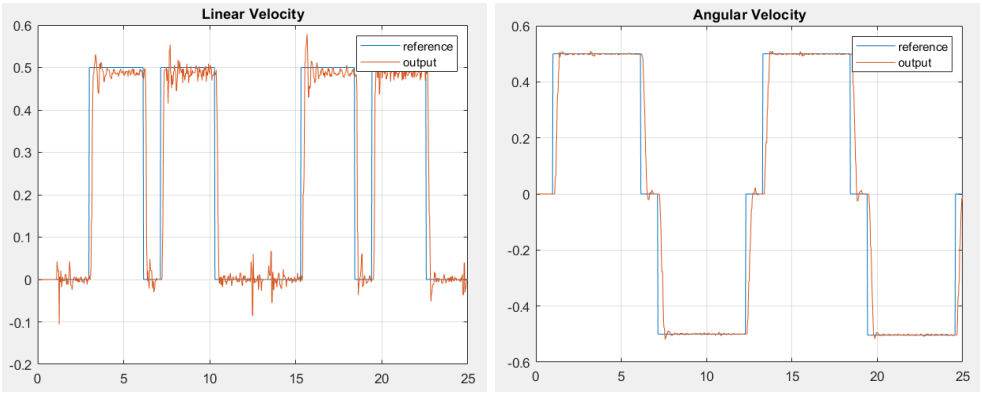


Figura 3.2: Diagrama de bloques del robot, planta a controlar



(a) Velocidad lineal. (b) Velocidad angular.

Figura 3.3: Referencias de pulso del ensayo de velocidad angular y lineal con la respuesta del robot.

$$\begin{cases} X[k+1] = FX[k] + GU[k] \\ Y[k] = CX[k] + DU[k] \end{cases} \tag{3.1}$$

de esta forma la matriz característica del sistema F, debe representar de forma implícita el retraso añadido, en este caso en forma de n variables de estado, así como las variables intrínsecas de la dinámica de las velocidades lineal y angular del robot.

De esta forma se obtiene un sistema de orden 10, 8 retrasos y 2 ecuaciones de estado de la dinámica del robot[1].

Finalmente, dado este análisis del modelo físico, es el momento de parametrizarlo y para ello se cuenta con otro ensayo de excitación de escalón de la velocidad lineal y angular 3.3.

Tras este ensayo y con las ecuaciones planteadas, para parametrizar la solución, se puede aplicar mínimos cuadrados aproximando la solución a una respuesta de orden uno con 8 ciclos de retraso. En este ensayo no hay zonas de saturación, pero si retardos. Para este cálculo, MATLAB cuenta con una función que implementa dicho ajuste de parámetros *idss*() integrada en la *toolbox System Identification Toolbox* [27] . De esta forma se obtienen las matrices de sistema. Para simular el control con el modelo se utiliza el esquema de la figura 3.4.

3.2.2 Control en espacio de estados

Una vez parametrizado el modelo, es hora de realizar el control de la planta. Para ello será necesario establecer una estrategia de control para la velocidad lineal y angular, que como ya se ha visto evolucionan de forma lineal. Existen técnicas de control basadas en la prealimentación de consigna o en control basado en Lyapunov.

Sin embargo en este caso, lo más sencillo es realizar un control en bucle cerrado, que integre el error de posición de las variables de salida (velocidad lineal y angular), de forma que en régimen permanente el error entre la consigna de velocidades y la salida, sea nulo (error de posición nulo de las velocidades) [26].

Para ello será necesario calcular el error en cada instante k de la velocidad tanto lineal como angular, o lo que es equivalente, Generar acción de la planta a través de un sistema de polos igual a 1, cuyas variables de estado sean los errores de posición de la salida respecto de la consigna (matriz de integradores).

[1]Estos ensayos se han realizado con el *software* de comunicación *UDP* anterior, es posible que con la nueva solución este parámetro cambie la latencia de la red

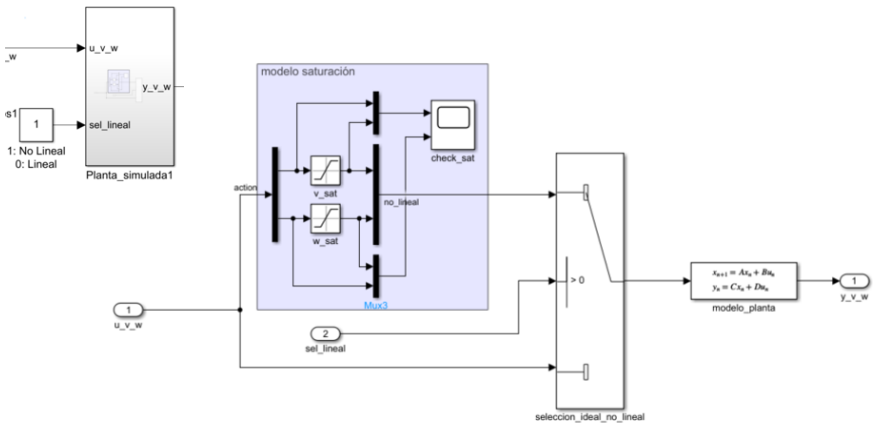


Figura 3.4: Implementación en Simulink del modelo en espacio de estados así como ls zonas de saturación no lineales para simulación del modelo.

de esta forma, para obtener todos los polos como unidad, la matriz F_r del controlador ha de ser la matriz identidad I que multiplique el error y la G_r la matriz que selecciona del errror, las variables a controlar (w y u). De esta forma se obtendría un sistema serie entre el sistema en espacio de estados conformado por regulador y el sistema de la planta 3.2.

$$\begin{pmatrix} X_s[k+1] \\ X_R[k+1] \end{pmatrix} = \begin{pmatrix} F_s & 0 \\ -G_R C_x & F_R \end{pmatrix} \begin{pmatrix} X_s[k] \\ X_R[k] \end{pmatrix} + \begin{pmatrix} G_s \\ 0 \end{pmatrix} u[k] + \begin{pmatrix} 0 \\ G_R \end{pmatrix} r[k]. \tag{3.2}$$

Como se observa el sistema cuenta con unas variables de estado del regulador y de la planta así como la acción de la planta, de esta forma se plantean dos grados de libertad sobre los cuales se puede actuar par a encontrar el sistema deseado. Así pues si se deduce una ley de control L_i tal que permita obtener un sistema con 3.3 una F_eq equivalente determinada por esta ley de control mediante realimentación de las variables de estado de la planta y la entrada de acción de la planta.

$$X_i[k+1] = F_i X_i[k] + G_i u_i[k] + G_x r_i[k] = (F_i - G_i L_i) X_i[k] + G_x r_i[k]; \tag{3.3}$$

De esta manera, encontrar una Ley L_i tal que haga de los valores propios de todo el sistema serie bastaría con resolver el cálculo de la ecuación para L_i 3.4.

$$|Iz - F_i + G_i L_i| = 0 \tag{3.4}$$

MATLAB cuenta con una función, para resolver esa ecuación par los autovalores deseados, en este caso se han utilizado un vector de polos que aseguran que la acción no satura con una referencia como la que se va a usar, pese al compromiso con el tiempo de respuesta [2].

De esta forma se implementan en Simulink las leyes partiendo de las variables de estado estimadas por el observador 3.5 3.2.4.

[2]Será preciso tener en cuenta el compromiso de velocidad de respuesta con el problema de control en zona linea, así como el filtrado de ruido, ya que sistemas rápidos implican mayor ancho de banda por lo tanto respuestas más sometidas al efecto de ruido de medida y de modelo

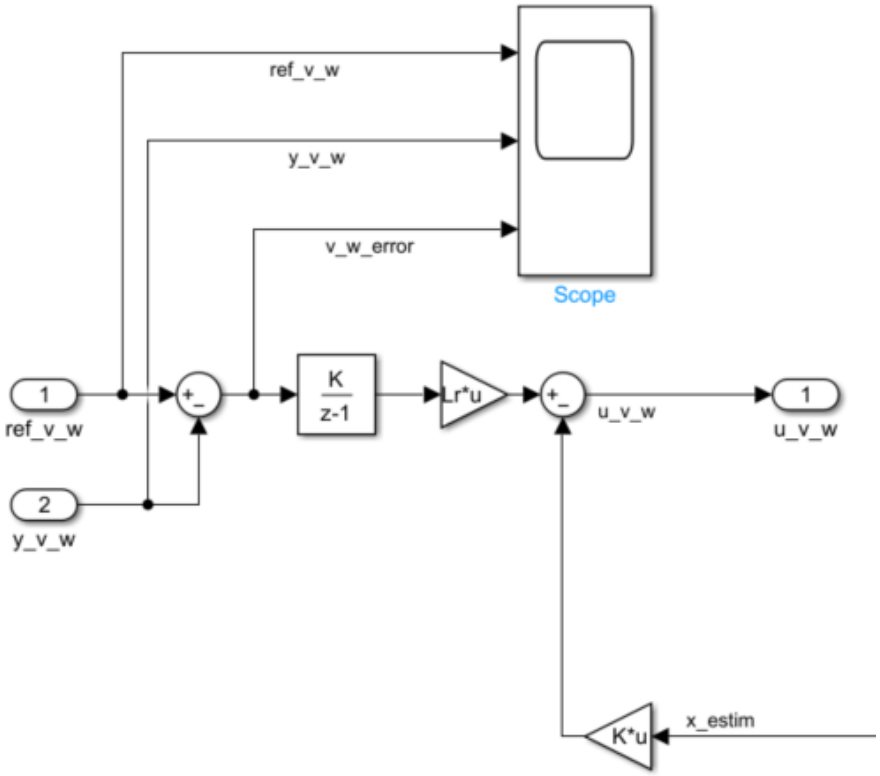


Figura 3.5: Implementación en Simulink de las reglas de control calculadas

3.2.3 Control basado en Lyapunov

Tras el diseño del servocontrol, se obtiene un controlador de la velocidad del robot, tanto lineal como angular pero no es posible controlar la pose del robot respecto a una referencia. A diferencia de la dinámica que muestran las velocidades del robot, la cinemática de este no son ecuaciones lineales y para realizar un control de trayectoria variable es preciso realizar un control no lineal [3]

En este caso se trata el control de una trayectoria en dos dimensiones cuya pose se puede describir como $p = [xy\theta]$ donde cada una de las variables son las representadas en la figura,donde θ es el giro proporcionado a cada instante por la velocidad angular ω 3.6.

Por lo tanto para realizar este control no lineal se realiza mediante un LBC (*Lyapunov Based control*). Este control consiste en plantear una función de Lyapunov $V(x)$ de tal forma que para el sistema global con reglas de control aplicadas $y = f(x)$, presente el punto de equilibrio estable en el valor consigna y se cumplan en $V(x)$ los requisitos de estabilidad de Lyapunov:

- $V(\bar{0}) = \bar{0}$
- $V(x) > \bar{0}$ for x $\forall D - \bar{0}$ o solución Definida Positiva
- $\dot{V}(x) \leq \bar{0}$ for x $\forall D$ o solución Semi-Definida Negativa

de esta forma el sistema $y = f(x)$ 3.7 con velocidad lineal 3.5 y velocidad angular 3.6 se propone la función de Lyapunov 3.7 basada en la bibliografía [28].

$$v = v_l s * cos(\alpha) + v_r * cos(\epsilon_\theta). \tag{3.5}$$

[3] una linearización del problema no tendría el suficiente alcance. Se suponen trayectorias lejanas a un punto e equilibrio

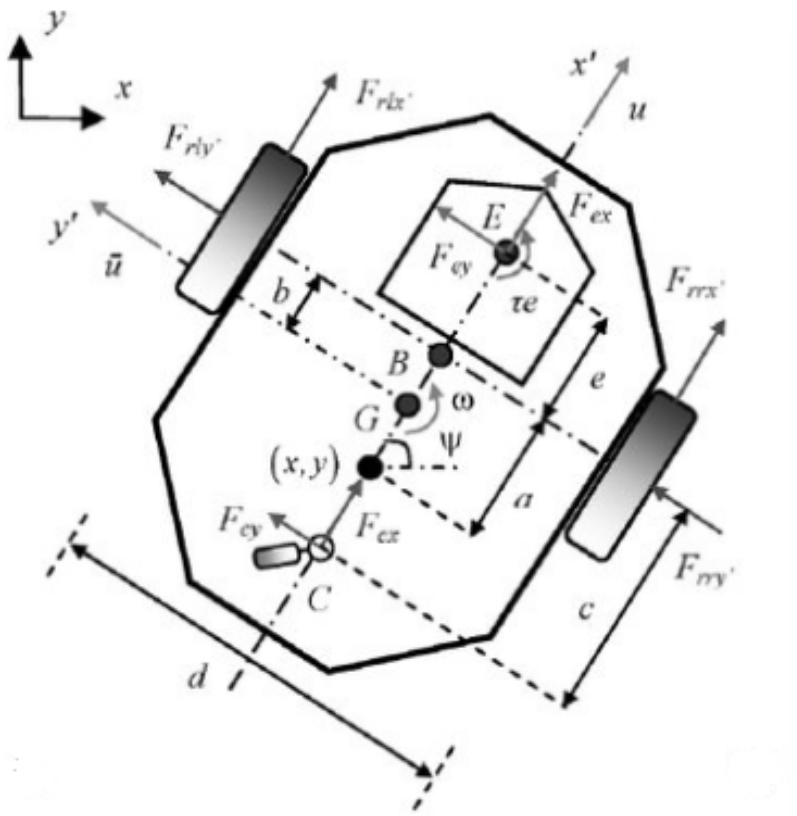


Figura 3.6: Diagrama de ejes de velocidad lineal y angular del robot PIONEER-3DX

$$w = \dot{\theta}_m d + v_m d[K_w(v_l ssin(\alpha) + v_r sin(\epsilon_\theta) + dsin(\alpha))] \tag{3.6}$$

$$V = frac12d^2 + 1 - cos(\epsilon_\theta) \tag{3.7}$$

Como se observa, la función es nula en el origen y siempre positiva para cualquier valor, en cuanto a la derivada depende del valor de $v_l s\ \theta_m d$ and $v_m d$. Para cumplir la condición de derivada semi definida negativa para todos los valores, se aplican las siguientes reglas de control [28] 3.7.

De esta forma los valores de velocidad linear y angular se calcularán mediante las siguientes reglas propuestas 3.8que cumplen con la condición de la derivada de Lyapunov.

$$\begin{cases} v_{ls} = K_v d \\ \theta_{md} = atan2\left(fracv_l ssin(\alpha - \epsilon_\theta)V_r + V_l scos(\alpha - \epsilon_\theta)\right) + \theta_r \\ v_{md} = \sqrt{v_r^2 + (K_v d)^2 + 2v_r K_v dcos(\alpha - \epsilon_\theta)} \end{cases} \tag{3.8}$$

Seleccionando Kv y Kw mayores a 0 se puede demostrar con una transformación algebraica simple, que satisface el criterio $\dot{V}(x) \leq \bar{0}$ for x $\forall D$. Estas ecuaciones se implementan en el modelo como una función de MATLAB.

Por otro lado para estimar la pose existen dos métodos:

- Algoritmo de odometría, integración de las velocidades de lectura de los odómetros

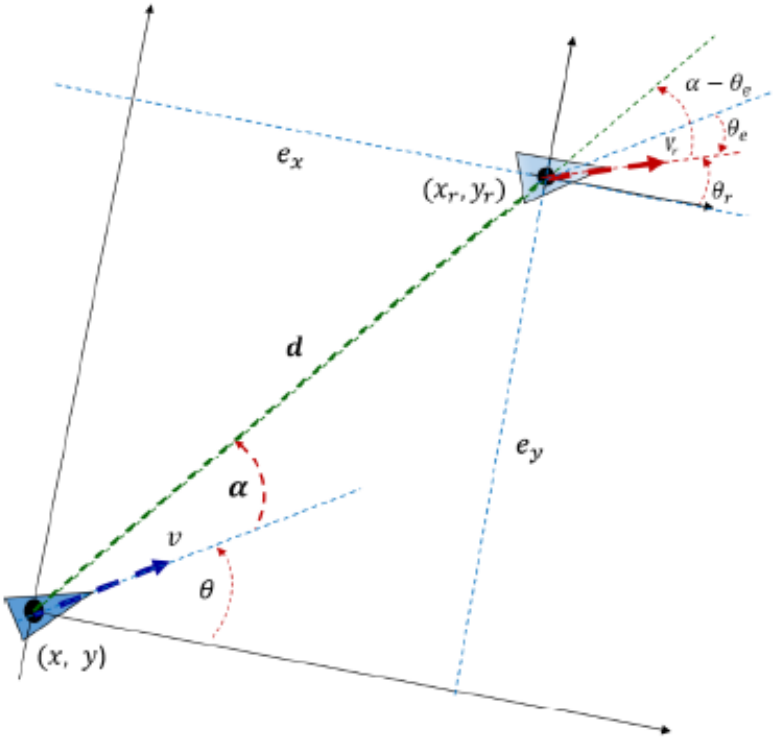


Figura 3.7: Esquema de las variables implicadas en el control no lineal LBC

- Lectura de pose de las cámaras presentes en el pasillo del laboratorio de experimentación.

El algoritmo odométrico no calcula la pose integrando las velocidades de las lecturas de los odómetros obteniendo la pose del robot en cada instante de muestreo.Sin embargo en la práctica aparecen deslizamientos [29] que son imperceptibles por los odómetros del robot, ya que en esas situaciones las vueltas contadas por los odómetros de las ruedas, no se corresponden con la velocidad real del móvil al no ser nula la velocidad de contacto de la rueda con el suelo.

Sin embargo las cámaras tienen un periodo de refresco mucho mayor que el período de muestreo [4]. Por ello se utiliza el algoritmo odométrico en los intervalos hasta obtener medida de las cámaras, momento en el cual se actualiza el valor de la pose al a valor de la cámara, reduciendo la incertidumbre en la pose dada por deslizamientos en las ruedas y otros fenómenos no observables por el sistema de odometría.

Así pues la estimación de pose en Simulink se implementa en un subsistema encargado de ello.Por otro lado las variables de entrada de error de pose vienen dada como

$e_x = \begin{bmatrix} e_d & e_\alpha & e_\theta \end{bmatrix}$ Sin embargo tanto la cámara como el algoritmo odométrico trabajan en coordenadas cartesianas lo cual hace necesario el uso de un bloque que traslade estas coordenadas a polares o al menos el error de pose (figura 3.8).

3.2.4 Observador Estimador

Como se ha adelantado anteriormente, las lecturas de salida del robot son velocidad lineal y angular. Esto imposibilita el acceso directo a las variables de estado del sistema. Sin embargo como se explicó en

[4] medida experimental con *software* de comunicación antiguo

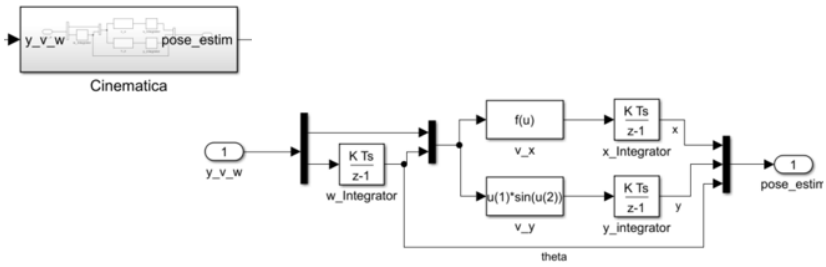


Figura 3.8: Diagrama de bloques del cálculo de pose sin cámaras

el apartado de la implementación del control lineal, es preciso disponer de las variables de estado para poder realizar el control realimentado.

Es aquí cuando entra el concepto de Observador. Un observador es un elemento dentro de un algoritmo de control, el cual emula el comportamiento de la planta y calcula sus variables de estado a cada periodo de muestreo. Esto lo hace aplicando las mismas acciones $u(t)$ que a la planta sobre el modelo de la misma a cada instante de muestreo

Sin embargo, esta acción es insuficiente ya que existen perturbaciones, fallos de parametrización del modelo entre otros efectos como el ruido, que llevan a que exista una diferencia entre las variables de estado emuladas y las variables de estado reales en la planta. Es por ello que para solucionar este hecho, se plantea un problema de control cuya premisa es conseguir que el error de las variables de salida del sistema emulado sean las mismas que las del sistema real.

Para solucionar este problema se aplica un procedimiento similar al del control de sistema lineal por espacio de estados mediante integradores. De este modo La regla de control se aplicarían como en el esquema 3.9. Y procediendo del mismo modo se llega a la ecuación de cálculo de las leyes de control L_i*,para lo cual se utiliza la función *place*() que resuelve esta ecuación para los valores propios deseados que implementa [30].

$$L_a = place(F, G, [p_1, p_2....p_n]) \tag{3.9}$$

De esta forma la implementación del Estimador en Simulink queda como la de la figura 3.10 e integrada en el control lineal como 3.11

3.2.5 Controlador Global del sistema

De esta forma, integrando todo el proceso anterior se obtiene el diagrama de bloques global del control. El bloque de la planta será en el caso simulado, el modelo de la planta, en el caso real el *driver* con el robot y la cámara.

Se observa el bucle interno de control en espacio de estados con el control basado en integradores, toma como referencia las consignas provenientes del control externo que constituye el control basado en Lyapunov. Para este se incluyen los bloques del algoritmo odométrico con la entrada de corrección de cámara, la cual solo será necesaria en el caso de ejecución con planta real.

Finalmente el generador de trayectorias está englobado dentro del bloque de generador de referencias 3.14, que será el encargado de producir las coordenadas de cada punto de la trayectoria de referencia a cada instante de muestreo como consigna del control externo.

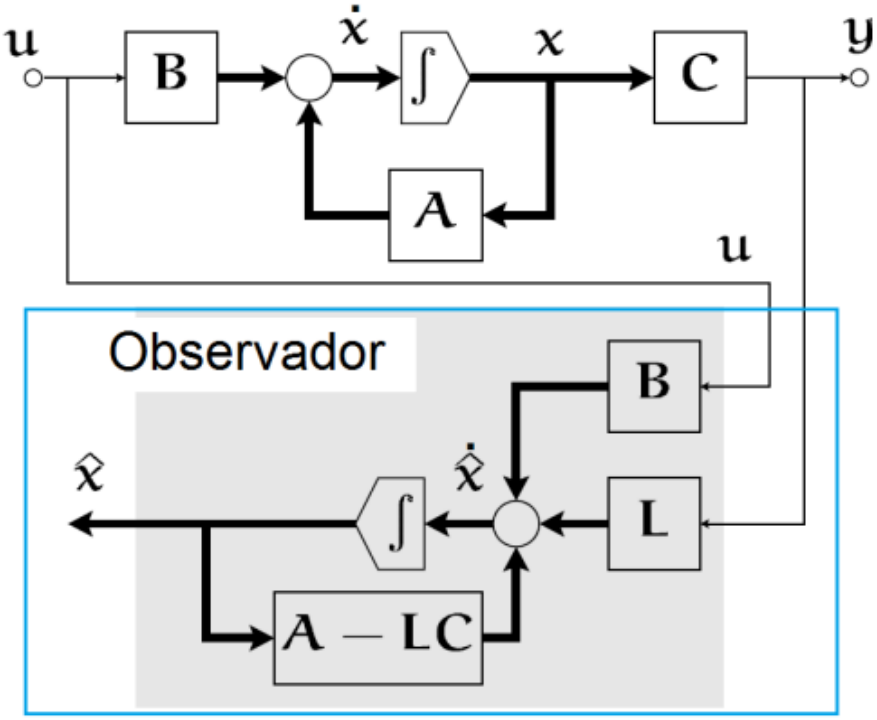


Figura 3.9: Diagrama de bloques estimador teórico, fase de estimación y de corrección según ley de control calculada

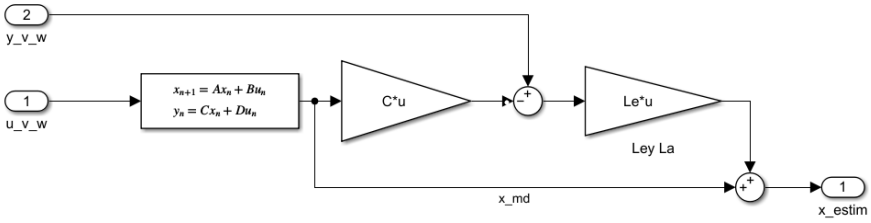


Figura 3.10: Diagrama de bloques estimador con el modelo del sistema y la ley de predicción calculada

3.3 Implementación del driver de comunicación on cámaras y robot

3.3.1 Revisión de soluciones anteriores

Según se expone en el apartado de estado del arte, previamente a este trabajo, existía una solución planteada para un problema similar. La solución pasaba por el uso de una S-function de Simulink que implementaba la creación y gestión de sockets en código C dentro del modelo. A continuación, se utilizaba esa S-Function como driver entre el modelo del controlador y el Robot. Por último, se generaba el código del controlador, así como de la S-Function y del generador de señales

Este método era funcional Hasta la versión de MATLAB 2021a en la cual la generación de código ya no incluye el bloque de generación de referencias.

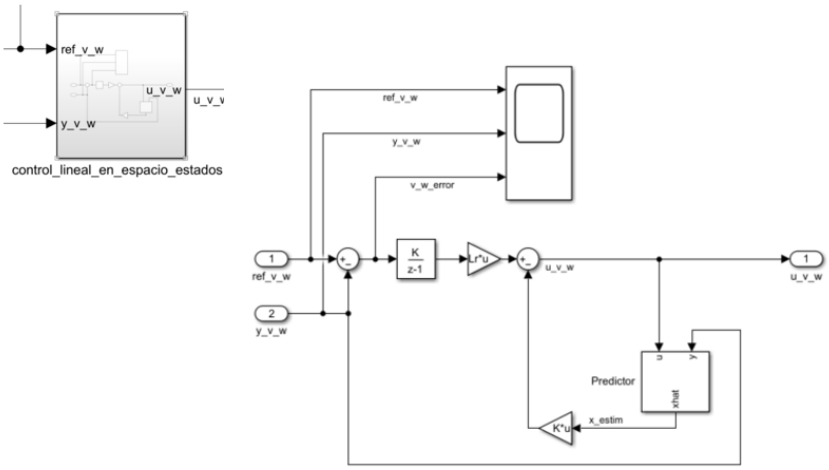


Figura 3.11: Control lineal con predictor de estado añadido

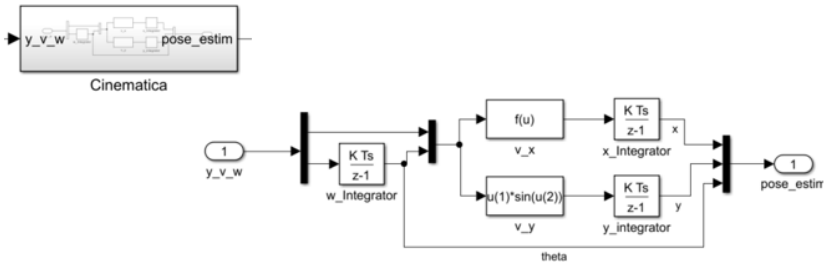


Figura 3.12: Diagrama de bloques del cálculo de pose sin cámaras

3.3.2 Implementación de solución

Así pues Otra posible solución que se plantea es la ejecución del controlador desde Simulink, sin necesidad de generar un ejecutable del modelo del control, haciendo uso de *Simulation-pacing* en las opciones de ejecución de Simulink. Esta herramienta permite equiparar el tiempo de simulación con tiempo real, con la equivalencia haciendo los tiempos en la simulación equivalentes a los reales 3.15, manteniendo en caso de no existir bloqueo, el mismo Ts real que el configurado para simulación. También es posible acelerarlos o decelerarlos según el caso de uso.

Sin embargo, tras esta solución subyace otro problema ya que la *S-Function* del *driver* no es ejecutable en simulación. Es por ello que ha sido necesario hacer uso de otro elemento que gestione los *sockets* y pueda usarse en Simulación. Aquí es donde aparecen los bloques de comunicaciones de socket UDP de Simulink.

3.3.3 Bloques de sockets UDP de Simulink

La comunicación UDP envía la información como paquetes de datos. Cada paquete de datos contiene una o más variables con una estructura y tamaño concretos. Así pues el tipo de dato que se envía ha de ser el mismo tipo dato que se recibe, por lo que ha de contener las mismas variables y del mismo tamaño, por lo que hay que asegurar que el paquete sea del mismo tipo tanto en el envío como en la recepción.

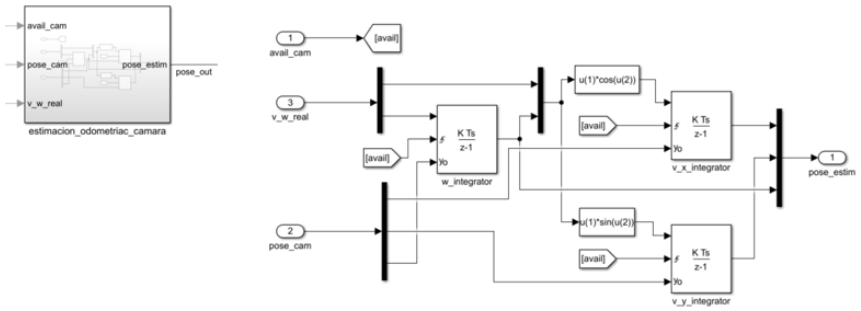


Figura 3.13: Diagrama de bloques del cálculo de pose con cámaras

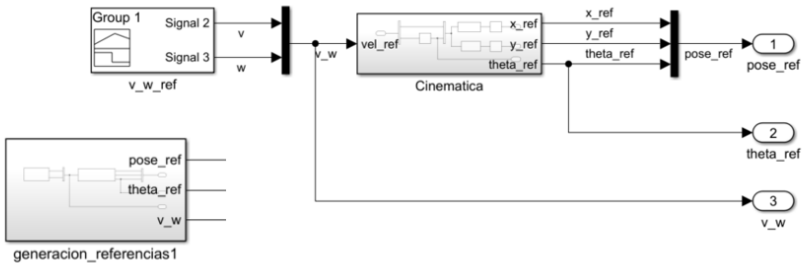


Figura 3.14: Diagrama de bloques del interior del subsistema de driver de cámaras y robots, implementado con los bloques UDP de Simulink

Según el código implementado en la comunicación del robot y en las cámaras, los tipos de los distintos paquetes de datos son de un tipo definido tanto para el caso de las cámaras como para el de los robots.

Como se puede apreciar, son tipos de datos basados en estructuras, tipo de dato no incluido de forma implícita en Simulink. Sin embargo se incluyen dos bloques que cumplen una función similar a la del tipo *struct* en C: aglutinar distintas variables con distinto tipo de dato en una sola variable, un paquete. De esta forma el bloque *Pack* creará un paquete que será el input de datos a enviar en cada período de muestreo por el bloque *Send UDP*.

Del mismo modo, Simulink al no contar con un tipo de estructura, tampoco cuenta con la manera de indexar las variables dentro de una estructura. Es aquí cuando entra en juego el bloque *Unpack*. Su función es obtener del paquete de datos de recepción cada una de las variables que contiene, dado el tipo y orden de estas. Es por ello que estos bloques se usarán de la siguiente manera en tras los bloques de recepción de los paquetes recibidos desde el robot y desde las cámaras.

3.3.4 Integración de la solución de comunicación

Una vez validado el funcionamiento para una cámara y un robot, es necesario adaptar el modelo para que este sea capaz de comunicarse con cualquier cámara o robot seleccionado de los disponibles en el laboratorio. El modelo de la figura 3.19 integra todo este proceso de envío de datos y *request* de datos a la cámara cada ciclo y recepción para una cámara y robot concretos.

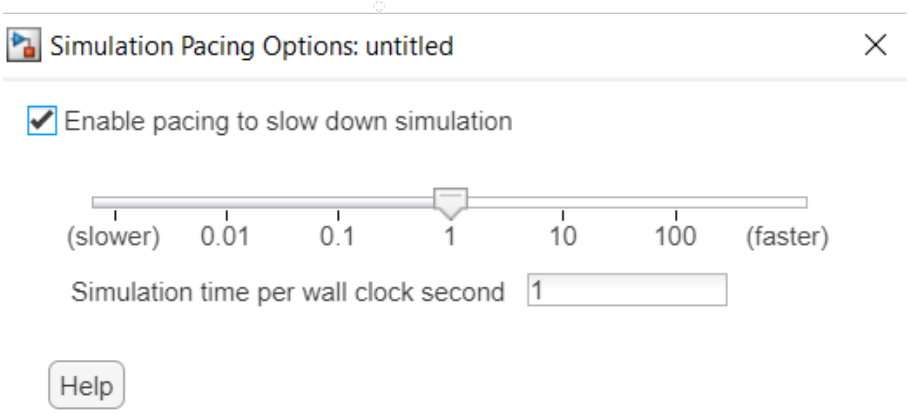


Figura 3.15: Diagrama de bloques del interior del subsistema de driver de cámaras y robots, implementado con los bloques UDP de Simulink

```
typedef struct {
  double cam_id;     //id_cam
  double data[5][10];//matrix data
  double t_sec;
  double t_usec;
}comm_recv_camara5;
```

(a) tipo de paquete de cámara

```
// Estructuras de datos
typedef struct
{
  int data_number;  // numero de pkt
  float lineal;   // Velocidad lineal
  float angular;  // Velocidad angular
}comm_data_t;
```

(b) tipo de paquete de robot

Figura 3.16: tipos datos enviados por la cámara y el robot como paquetes por cada request del controlador

3.4 Conclusiones

A lo largo de las distintas fases del desarrollo de este proyecto, se ha podido implementar, ejecutar y validar un controlador sobre la plataforma de MATLAB/Simulink, con una solución a las comunicaciones con el *hardware* real, así como una aproximación teórica a la ejecución en tiempo real.

Finalmente tras el planteamiento teórico, es conveniente validar si los requisitos temporales y de control son funcionales. Para este fin se han planteado etapas de validación en el modelo, sobre un Software simulado en un puesto remoto y en *Hardware* real haciendo uso del *driver* desarrollado.

Por ello queda realizar un análisis de los resultados obtenidos en las diversas pruebas así como los límites de error para poder establecer los límites operativos de esta solución.

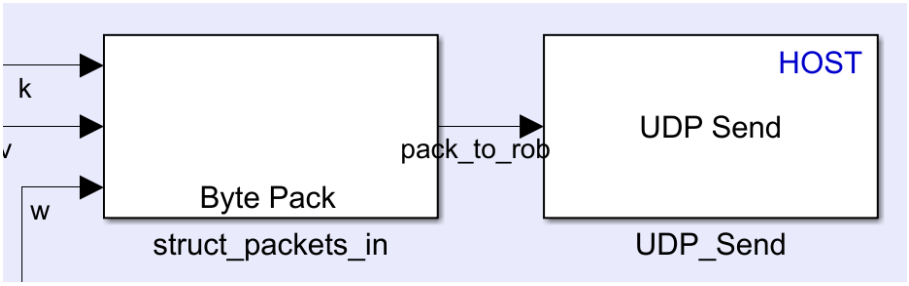


Figura 3.17: Diagrama de generación del paquete equivalente mediante el bloque *Pack* a un tipo e estructura de dato de paquete enviado al robot mediante UDP de Simulink

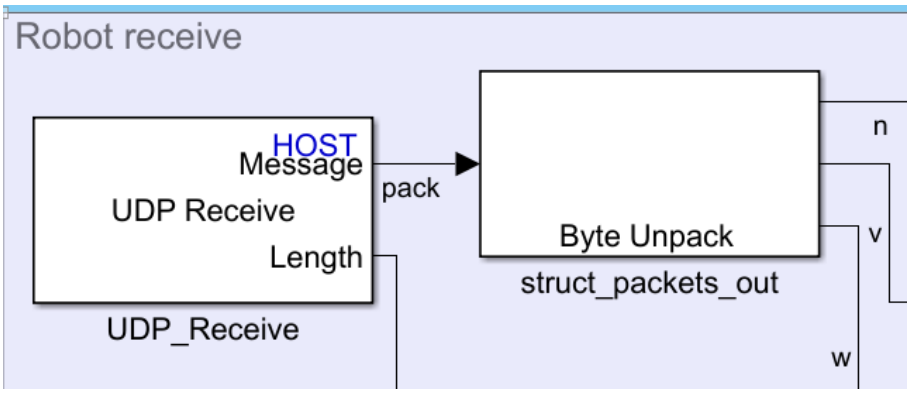


Figura 3.18: Diagrama del indexado de los datos del paquete equivalente de recepción mediante *Unpack* a un tipo de dato numérico en ese caso un entero de 32 bits y dos decimales tipo double

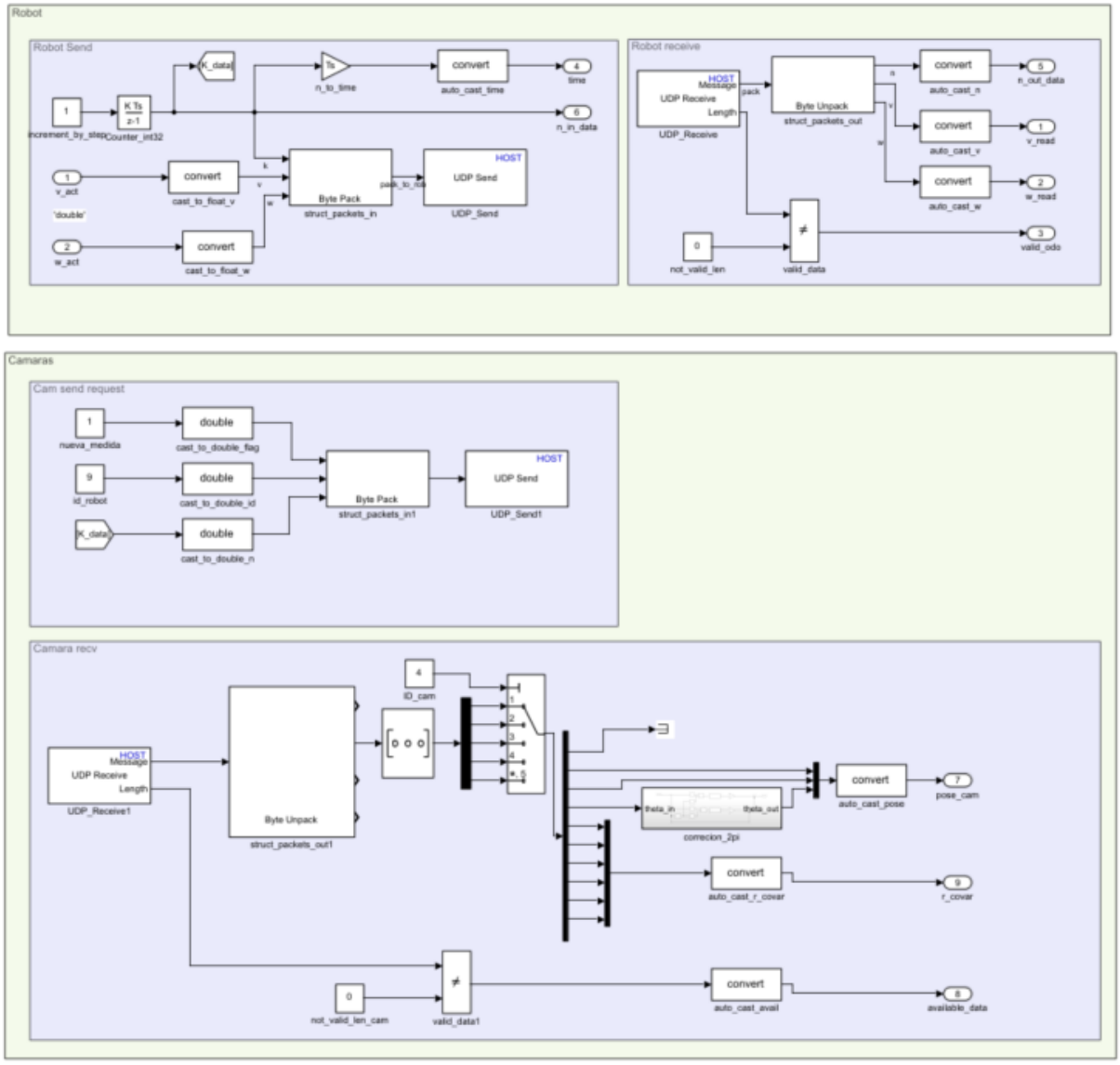


Figura 3.19: Diagrama de bloques del interior del subsistema de *driver* de cámaras y robots, implementado con los bloques UDP de Simulink

Capítulo 4

Resultados

4.1 Introducción

A lo largo de este capítulo se exponen los resultados más relevantes de este trabajo, de forma analítica para poder así validar los requisitos temporales y de control planteados, así como poder establecer unos límites operativos, del objeto de desarrollo de este proyecto.

4.2 Entorno experimental

4.2.1 Introducción

Llegado este punto, se ha desarrollado una posible solución al problema de forma que se cumplan los requisitos iniciales de este proyecto. Para ello, las pruebas de validación deberán mantener como en cualquier otro proyecto, una trazabilidad con los objetivos iniciales planteados. De esta forma se ha realizado la validación de la solución en distintas fases presentadas a continuación.

4.2.2 Test de controlador

En este apartado, se comprueba el funcionamiento del controlador, es decir, que alcanza la referencia de consigna, que la acción no se satura y que las salidas tiendan a alcanzar las referencias

En primer lugar, todas estas comprobaciones se llevan a cabo en el modelo simulado es decir, utilizando el modelo de la planta y no la planta real. Para este fin se utiliza el esquema de la figura 4.1.

4.2.3 Test de componente de comunicación

En este apartado se valida si los bloques de comunicación por sí solos cumplen la estabilidad en el periodo de envío y recepción con la frecuencia de muestreo, y realizan el envío en el formato de paquete de dato específico.

Estos tests tienen como función, validar los requisitos generales de la comunicación. Para este fin se envía una señal de rampa y se compara con la respuesta como en la figura 4.2.

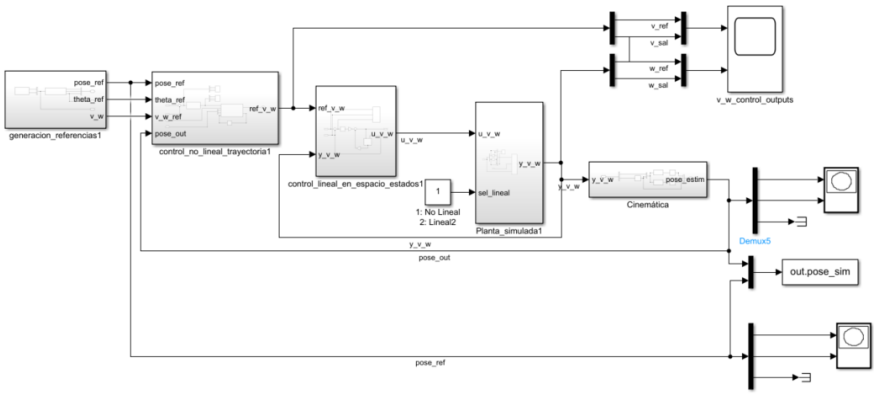


Figura 4.1: Diagrama de bloques del controlador con todas las etapas expuestas en el apartado 3.2, con los dos lazos de control lineal interno y no lineal externo y el generador de trayectorias

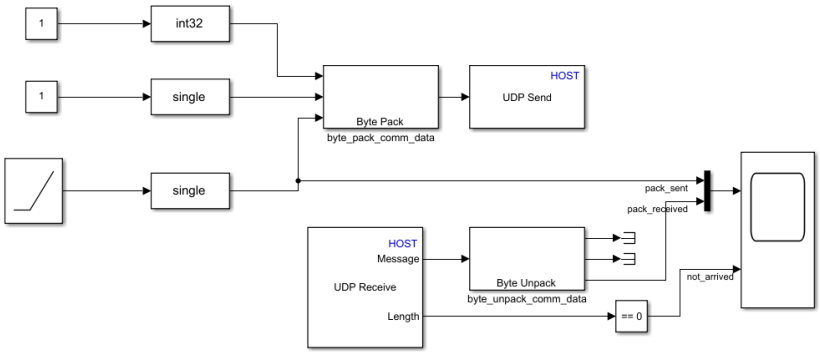


Figura 4.2: Diagrama de modelo para prueba de comunicación de los bloques de comunicación del driver implementado con el puesto remoto de simulación. Se observa la excitación de rampa en la entrada de velocidad lineal y la monitorización de la recepción

- Test de comunicación sin pérdida de paquetes. Se prueba la compatibilidad de los paquetes de envío y recepción, utilizando los mismos tipos de dato que software implemenetado en el robot. El dato enviado es el que se devuelve por lo que en caso ideal, el envío y la recepción deberían coincidir.

- Test con pérdida de paquetes. Se simula un retraso en el envío de cada cuatro paquetes. Al no llegar a tiempo el sistema debería rechazar los paquetes que no llegan a tiempo (dentro del periodo de muestreo) y evitando el bloqueo.

Esta última condición está derivada del control en tiempo real, ya que si un dato no llega a tiempo, este debería ser descartado

4.2.4 Test de control con comunicación con robot y cámaras

Finalmente, en este caso se realiza un test sobre la plataforma real, es decir, conjunto PC ejecutando el controlador con robot conectado al control mediante el driver desarrollado.

Es por ello que el esquema de control y medida utilizado 4.3 es similar al de simulación pero con el driver enligar del modelo de la planta. Se añaden los dos modelos de forma concurrente para realizar un estudio de los datos obtenidos.

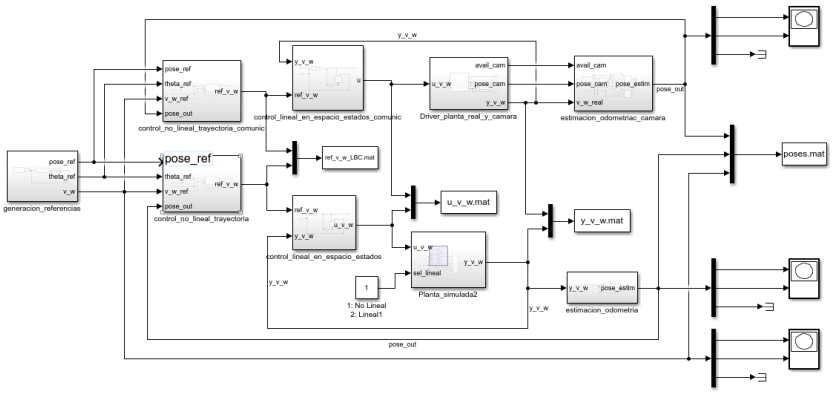


Figura 4.3: Esquema del controlador ejecutado sobre simulación abajo y sobre el driver de comunicación con la planta arriba. Dependiendo de si la comunicación es con la planta simulada o real, será necesario configurar las IPs de los bloques de envío y recepción

4.2.5 Métricas de calidad

Como ya se ha explicado, existe trazabilidad entre los tipos de ensayo y los requisitos de control y comunicación planteados. De esta forma ada ensayo deberá validar sus propios requisitos y establecer unos límites operativos sobre los mismos. Es por ello que cada ensayo requiere de una medida del error distinta.

- desviación típica
- error absoluto máximo instantáneo
- error máximo de norma cuadrática de un vector
- desviación típica de norma cuadrática de un vector

4.3 Resultados experimentales

4.3.1 Resultados del test del controlador

En este apartado se validan los requisitos de control. Aquí se verifica el seguimiento de las referencias por parte del controlador 4.4 en cuanto a velocidades y 4.5 trayectoria.

Se comprueba que las lineas de referencia son casi coincidentes. Es por ello que si el modelo de la planta es el adecuado puede considerarse como controlador válido.

4.3.2 Resultados del test de comunicación

4.3.2.1 Experimento sin pérdida

Como se observa en la figura 4.6 la recepción del dato de velocidad enviada respecto de la recibida es aparentemente similar ya que el simulador en este caso solo reenvía los paquetes recibidos.

También se observa como la pérdida de paquetes es un episodio muy puntual y cuando sucede el valor se vuelve a recuperar ya que se ha limitado el tamaño del *buffer* de recepción al último paquete para evitar la intrusión de datos antiguos que generarían retraso en las lecturas de los sensores.

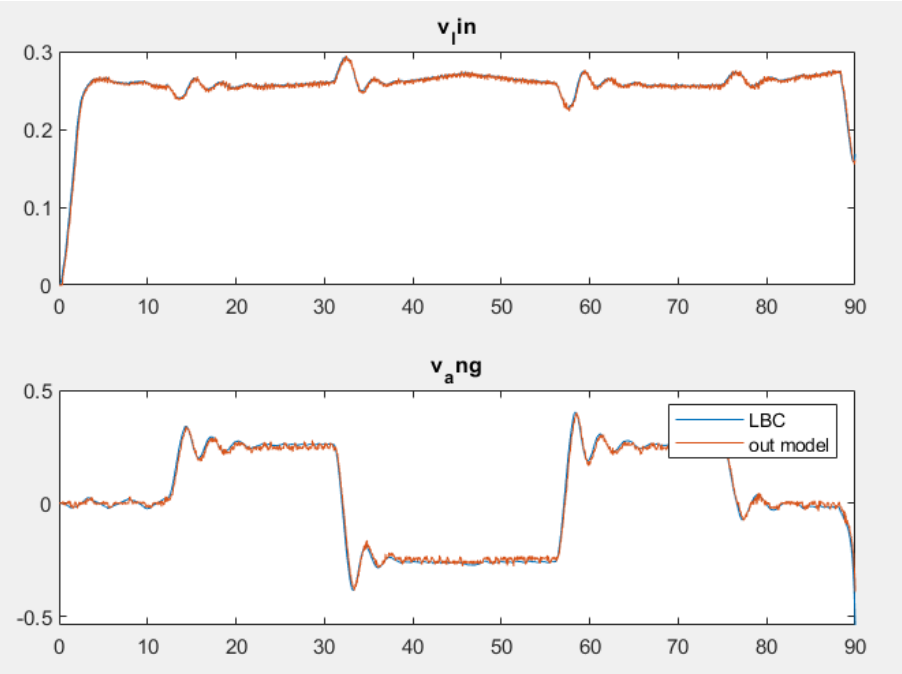


Figura 4.4: Trayectoria de referencia y trayectoria trazada de forma ideal por la planta según simulación

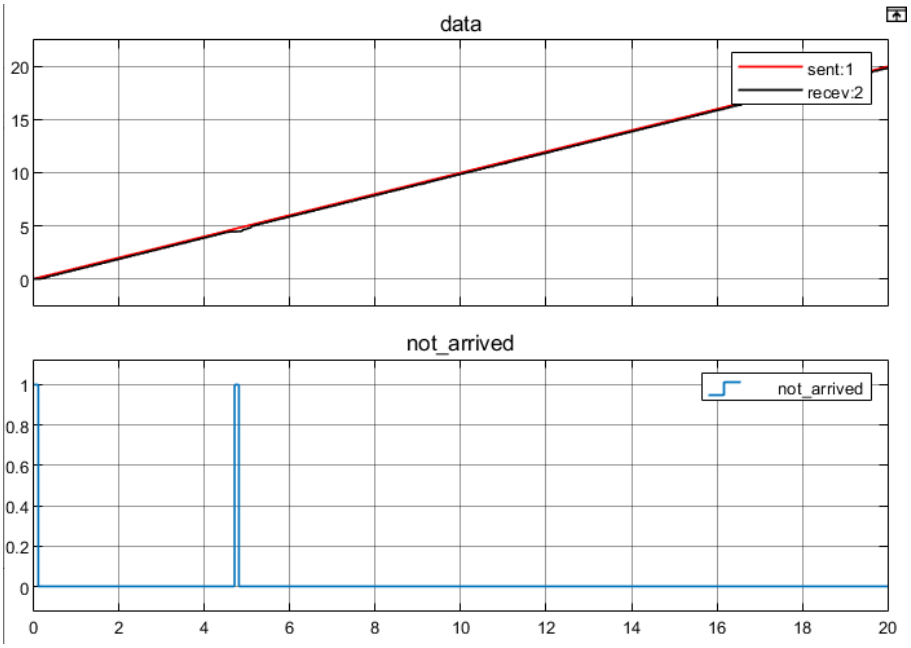


Figura 4.6: Esquema de envío y recepción de datos en el experimento sin pérdida

En cuanto a las métricas de error se obtienen los datos de la tabla 4.1.

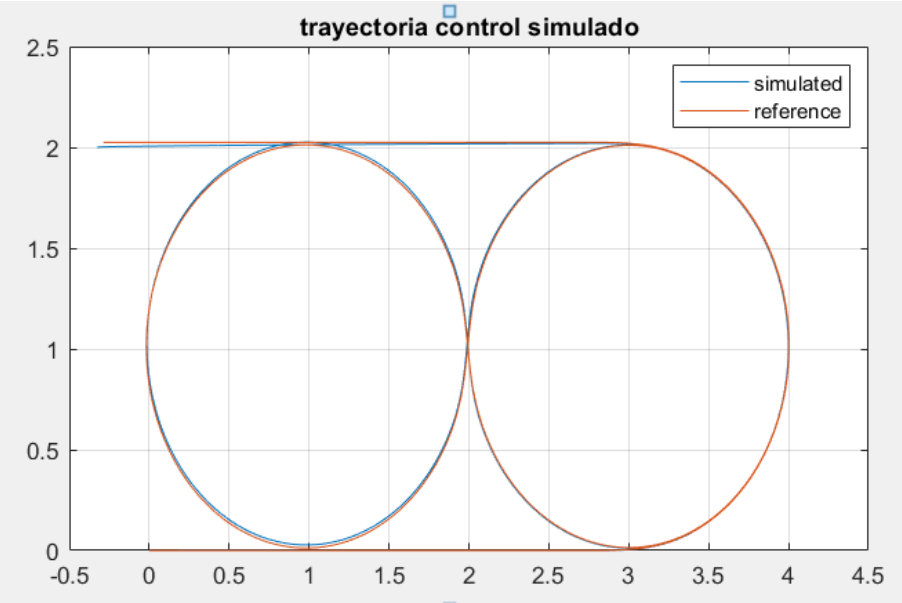


Figura 4.5: Trayectoria de referencia y trayectoria trazada de forma ideal por la planta según simulación

Tabla 4.1: Métricas de error del experimento sin pérdida.

Métrica	Valor
desviación típica	0.1465 m/s
tasa de pérdida de paquetes	0.63 %
error máximo	0.4 m/s

Es preciso remarcar que al no existir otros procesos concurrentes que bloqueen los procesos de envío y recepción de los *sockets*, la tasa de pérdida es casi nula.

Sin embargo en un entorno más ruidoso y con otros procesos concurrentes como el cálculo de los parámetros del controlador, pueden bloquear los procesos de manejo de los *sockets* y aumentar el factor de pérdida de paquetes.

4.3.2.2 Experimento con pérdida

En este apartado se impone una tasa de pérdida de paquetes del 50 % cada 10 muestras para simular el bloqueo de los procesos de los *sockets* por otros como el cálculo de variables del controlador en Simulink. De esta forma de cada 2 paquetes recibidos, uno no envía respuesta al controlador.

Como se puede observar en la figura 4.7, en este caso, pese a la pérdida de paquetes constante, el sistema mantiene el valor a su salida del último paquete válido, sin bloquear el funcionamiento mediante esperas de dato nuevo.

Sin embargo en este caso es preciso mencionar que el valor máximo de error depende de la distribución de los paquetes perdidos en el tiempo así como dela variación de la salida (figura 4.8 y tabla 4.2).

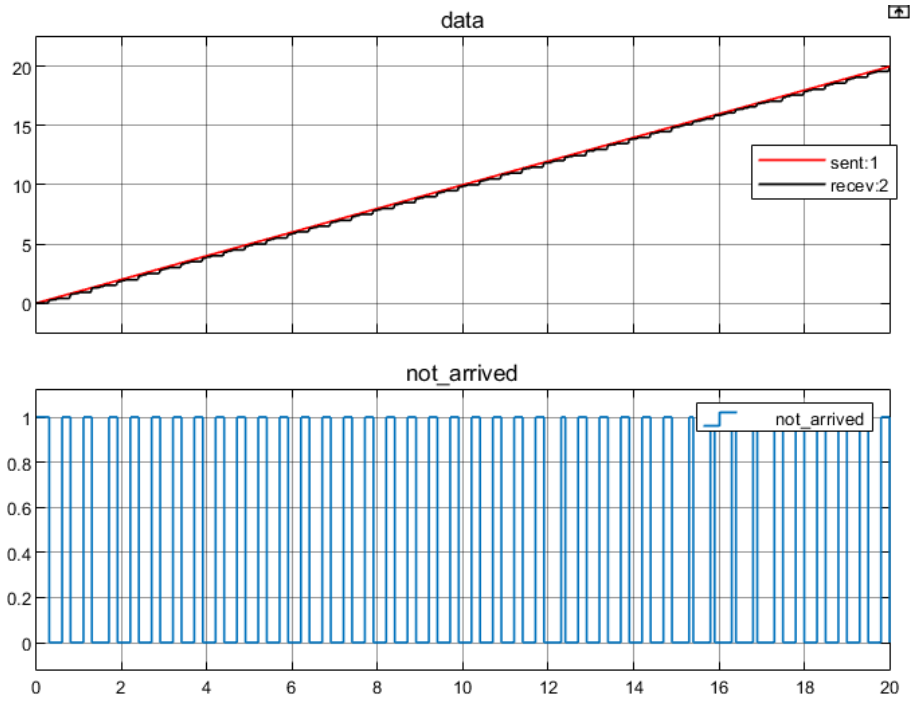
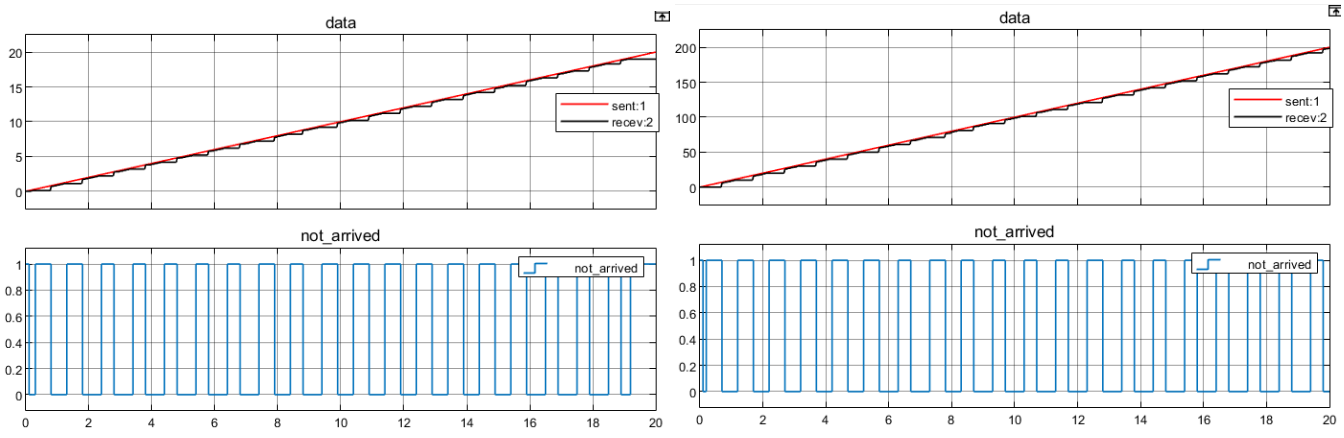
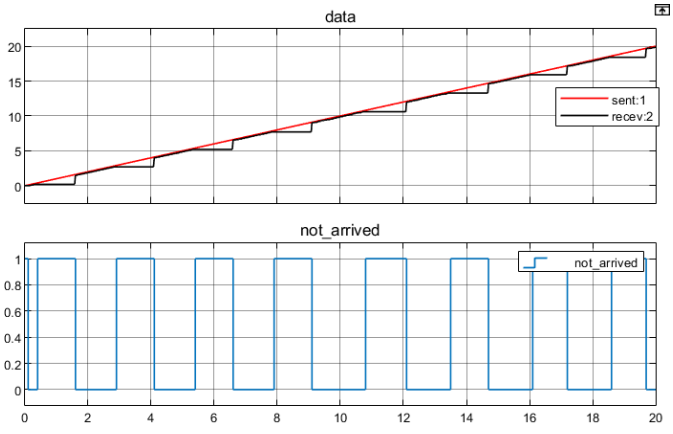


Figura 4.7: Esquema de envío y recepción de datos en el experimento sin pérdida

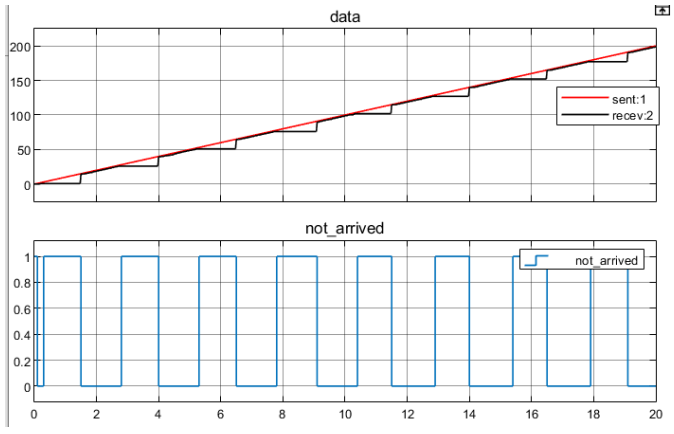


(a) pendiente 1 pérdida cada 20 paquetes.

(b) pendiente 10 pérdida cada 20 paquetes.



(c) pendiente 1 pérdida cada 50 paquetes.



(d) pendiente 10 pérdida cada 50 paquetes.

Figura 4.8: Diferentes mismo ratio de pérdida 50 % con distinta distribución temporal.

Pendiente de Rampa	Muestras sin dato	Máximo error	desviación típica
1	10	0,4	0,2
10	10	4,0	2,1
1	20	1	0,33
10	20	6,75	3,1
1	50	1,4	0,59
10	50	15	6,0

Tabla 4.2: Métricas de error del experimento con pérdida.

Por lo tanto es preciso analizar este caso que puede ser el más realista cuando se ejecuta el controlador de un robot dependiendo de la referencia y la pendiente de acción del controlador.

4.3.3 Resultados de test integrados

En este apartado se han realizado dos experimentos. Uno solo comunicando con el robot y el segundo obteniendo resultado del sistema de medición de las cámaras, corroborando así el correcto funcionamiento del *driver* ante las mismas.

4.3.3.1 Experimento con robot

En este experimento se puede extraer las gráficas del seguimiento de referencias del sistema y los parámetros de error de la tabla 4.3. Se observa como el error de posición es ínfimo respecto al de simulación si este se compara con la referencia. Es por ello que el *driver* para el control de la trayectoria puede considerarse fiable ya que introduce menor error que el propio modelo de la planta (como retrasos) o error de posición del propio lazo de control.

Métrica	Valor (m)
Desviación típica con trayectoria de referencia	0.0643
Desviación típica con trayectoria de simulación	0.0113
Error máximo respecto a la referencia	0.4756
Error máximo respecto a la simulación	0.058

Tabla 4.3: Métricas de error de pose del experimento de robot, respecto a la norma entre los puntos

De esta forma en cuanto a la pose se observa un escenario muy similar a la pose simulada (figura 4.9).

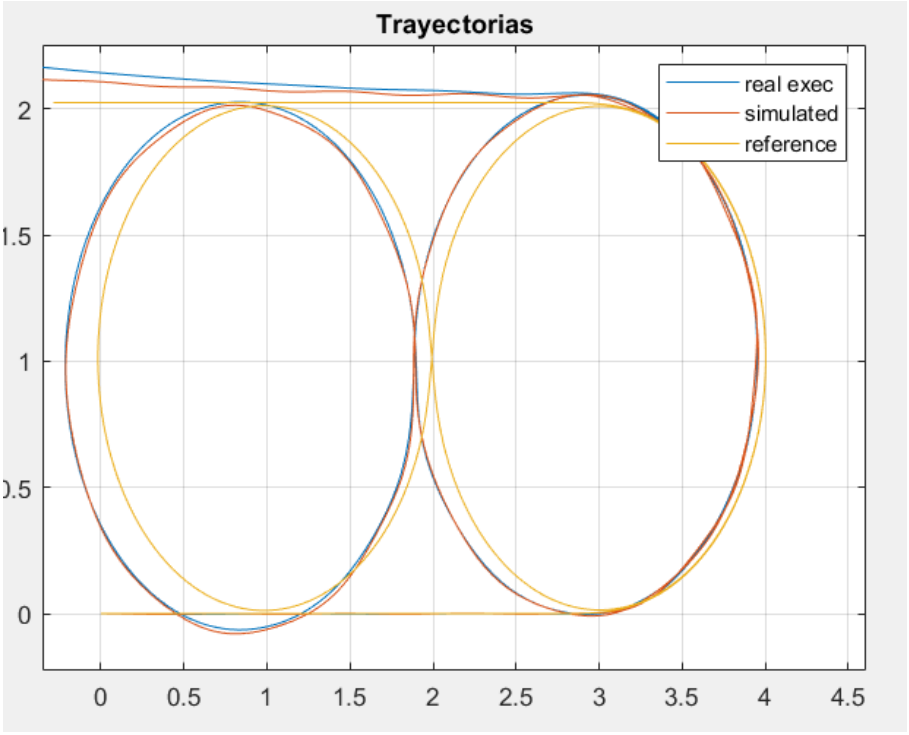


Figura 4.9: Resultados de pose de test de robot

En cuanto a la velocidad lineal y angular de la salida, se observa el seguimiento (con acoplo de ruido) de la salida de velocidad lineal y angular respecto de la referencia entregada por el controlador no lineal (figura 4.10).

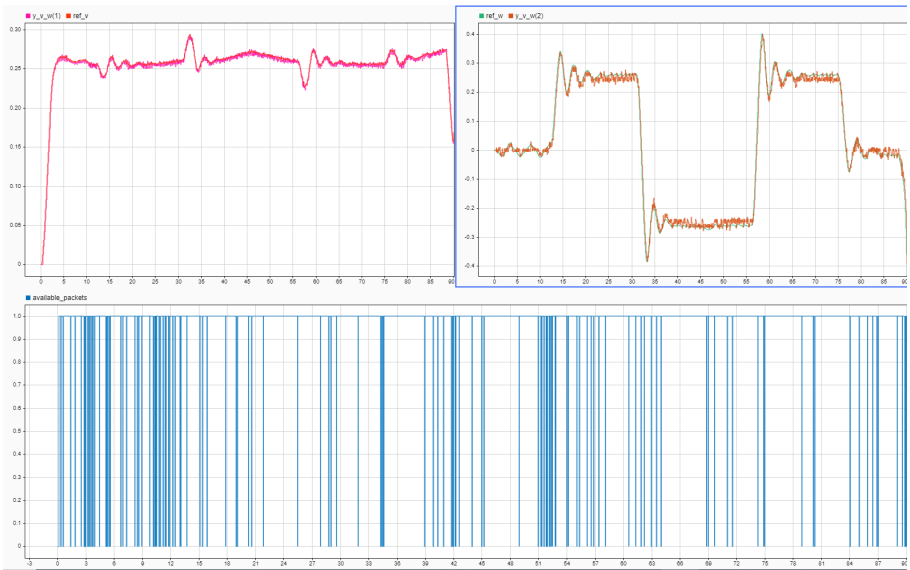


Figura 4.10: Resultados de velocidad angular y lineal de salida del robot respecto de la referencia dada por su lazo externo. Bajo paquetes de datos válidos en tiempo

Respecto a la comparativa de salida de velocidad angular y lineal del caso simulado y el real, las conclusiones son similares, como se puede ver en la figura 4.11.

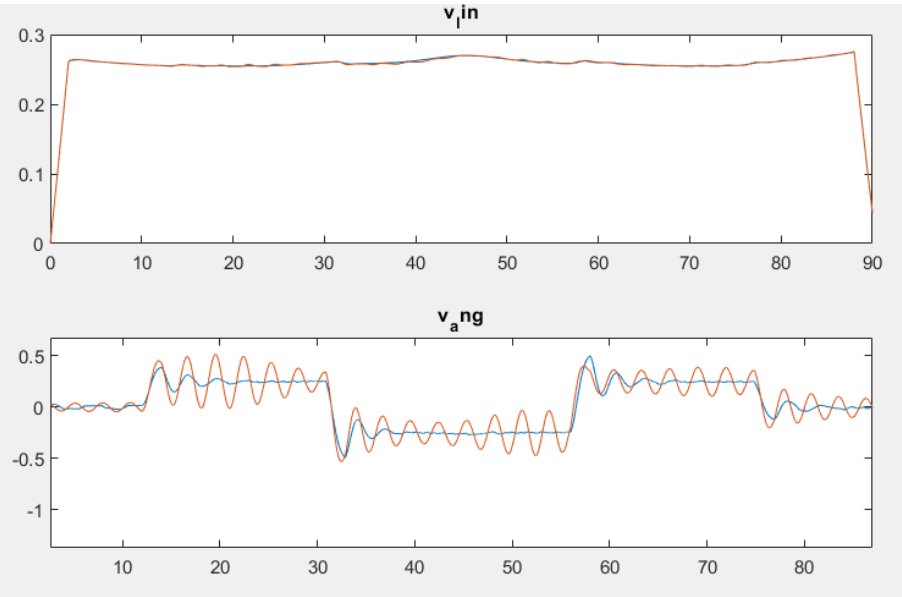


Figura 4.11: Pose real (morado, cámaras) contra las poses de simulación y del control mediante la medida de los encoders.

4.3.3.2 experimento con robot y comunicación de cámara

En este apartado se prueba la comunicación real con las cámaras mientras se ejecuta el control. De esta forma se obtiene la trayectoria real del robot evitando deslizamientos que los *encoders* del robot no pueden detectar (figura 4.12).

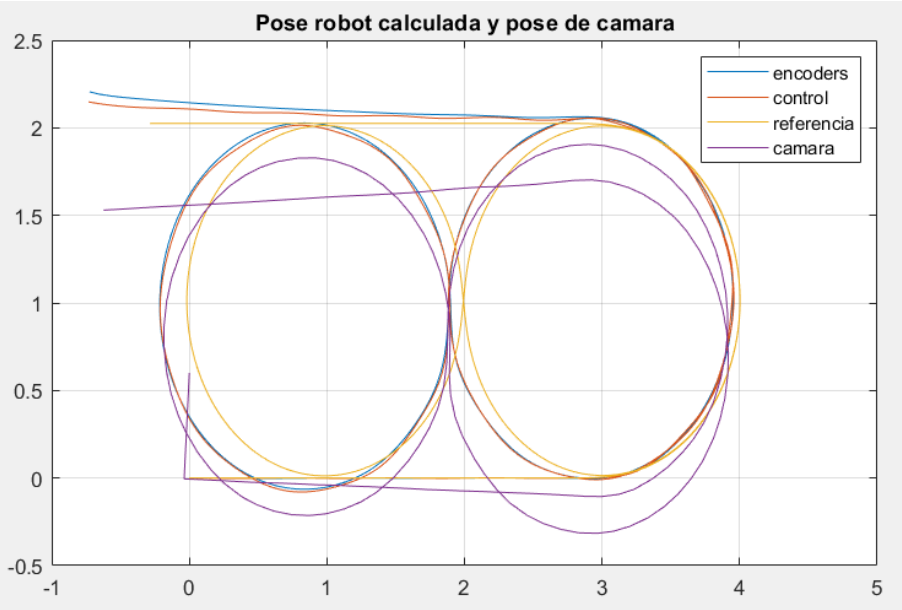


Figura 4.12: Pose real (morado, cámaras) contra las poses de simulación y del control mediante la medida de los encoders.

Como se puede observar, el cálculo basado de la pose basada en los sensores, difiere del dato obtenido de cámaras. Esto es principalmente a los deslizamientos de las ruedas. La solución a este problema pasa por implementar el control de pose basado en las cámaras cuando se obtenga información de estas 4.2.2.

Capítulo 5

Conclusiones y líneas futuras

5.1 Conclusiones

MATLAB/Simulink es una herramienta capaz de integrar todo el proceso de diseño de un controlador hasta la ejecución de este en tiempo real sobre un hardware real como se ha visto; pasando por las fases de simulación y ajuste experimental del modelo.

Finalmente se ha conseguido implementar un *driver* con el robot P3-DX, de forma que sea estable ante la perdida de paquetes de datos de la comunicación, así como de las cámaras *Kinect 2.0* que monitorizan la pose del robot.

Por otro lado se ha estudiado la ejecución de este *driver* de comunicación con el *Hardware* para un caso de control y pese a las restricciones temporales que plantea el sistema operativo Windows, la ejecución del controlador y la gestión de comunicación, llevan a una tolerancia aceptable del periodo de muestreo del controlador planteado para fines didácticos.

5.2 Lineas futuras

Hasta el momento, todas las variables monitorizadas a lo largo del tiempo, han sido referenciadas en la base de tiempos de Windows, la cual no precisa de motor de tiempo real.

Es por ello que para obtener una temporización precisa, sería recomendable obtener otra monitorización temporal de las comunicaciones, ajena al sistema operativo Windows basada en una aproximación de tiempo real. Esto permitiría estimar errores temporales más precisos y un estadística de la tolerancia temporal.

Por otro lado, puede utilizarse esta solución para implementar otros controladores distintos para el robot o replicar esta solución para otros dispositivos controlables mediante paquetes de comunicación UDP.

Finalmente, la dependencia en esta solución del entorno de Simulink es directa, ya que para ejecutar el controlador se precisa la instalación de Simulink en el *Hardware* que ejecute el controlador. Es por ello, que como linea futura de investigación, podría plantearse el uso del *Coder* integrado en Simulink, para la generación de código ejecutable (junto con la referencia del controlador) a tiempo real; de esta forma podría generarse un código ejecutable en dispositivos sin necesidad de adquirir una licencia de MATLAB-Simulink para el Hardware que ejecute el controlador.

Bibliografía

[1] "Advantages whileusing model based solutions," https://es.mathworks.com/solutions/model-based-design.html [último acceso 16/agosto/2022].

[2] J. Robledo Arévalo, "Actualización de herramientas para diseño e implementación de controladores digitales con matlab 2018," Trabajo fin de Grado, Universidad de Alcalá Escuela Politecnica Superior, 2019.

[3] C. Losada-Gutiérrez, F. Espinosa, C. Santos-Pérez, M. Marrón-Romera, and J. M. Rodríguez-Ascariz, "Remote control of a robotic unit: A case study for control engineering formation," *IEEE Transactions on Education*, vol. 63, no. 4, pp. 246–254, 2020.

[4] "Signal builder not supported by c/c++ coder," https://www.synopsys.com/blogs/software-security/5-types-of-software-licenses-you-need-to-understand/ [último acceso 22/agosto/2022].

[5] "Página oficial de scilab-xcos, que es scilab," https://www.scilab.org/about/scilab-open-source-software [último acceso 25/julio/2022].

[6] "M5 tipes of software license type," https://www.synopsys.com/blogs/software-security/5-types-of-software-licenses-you-need-to-understand/ [último acceso 22/agosto/2022].

[7] "Página oficial de xcos, posibilidades de x-cos," https://www.scilab.org/software/xcos/simulation [último acceso 29/julio/2022].

[8] "Different public toolboxes available for xcos-scilab," https://atoms.scilab.org/categories/instruments_control [último acceso 29/julio/2022].

[9] "Labview official web by nationalinstruments," https://learn.ni.com/courses/labview-core-1-lesson-1 [último acceso 6/agosto/2022].

[10] "C coder developed for labview designs," https://lavag.org/topic/20624-c-code-generator-for-labview/ [último acceso 5/agosto/2022].

[11] "Academil licenses for matlab-simulink," https://es.mathworks.com/products/academic-teaching.html [último acceso 15/agosto/2022].

[12] "Matlab official web," https://es.mathworks.com/products/matlab.html [último acceso 16/agosto/2022].

[13] "J008: Cálculo de la potencia eléctrica en simulink," https://jmirez.wordpress.com/2010/05/22/j008-calculo-de-la-potencia-electrica-en-simulink/ [último acceso 12/julio/2022].

[14] "What are mil, sil, pil, and hil, and how do they integrate with the model-based design approach," https://es.mathworks.com/matlabcentral/answers/440277-what-are-mil-sil-pil-and-hil-and-how-do-they-integrate-with-the-model-based-design-approach [último acceso 12/julio/2022].

[15] "official web openmodelica," https://www.openmodelica.org/ [último acceso 6/agosto/2022].

[16] P. Fritzon, "Introduction to object-oriented modeling and simulation with openmodelica,tutorial," Link oping, Sweden, Tech. Rep., 2006.

[17] "Blocks for communication devices such as network," https://build.openmodelica.org/Documentation/Modelica_DeviceDrivers.Blocks.Communication.html [último acceso 12/julio/2022].

[18] "Real time operative system," https://www.freertos.org/ [último acceso 25/agosto/2022].

[19] "Windows scheduler," https://docs.microsoft.com/es-es/windows/win32/taskschd/about-the-task-scheduler [último 25/agosto/2022].

[20] "Windows kernel-mode run-time biblioteca," https://docs.microsoft.com/es-es/windows-hardware/drivers/kernel/windows-kernel-mode-run-time-library [último acceso 2/septiembre/2022].

[21] "Api microsoft windows socket magament librery winsock2," https://docs.microsoft.com/en-us/windows/win32/winsock/windows-sockets-start-page-2 [último acceso 10/junio/2022].

[22] "Udp blocks simulink," https://es.mathworks.com/help/dsp/ref/udpsend.html [último acceso 30/agosto/2022].

[23] C. Ibeakanma, "What are tcp and udp ports," May 2022, https://www.makeuseof.com/what-are-tcp-and-udp-ports/ [último acceso 12/julio/2022].

[24] "Página de wikipedia, comunicación udp," https://en.wikipedia.org/wiki/User_Datagram_Protocol [último acceso 5/agosto/2022].

[25] "Página de wikipedia, comunicación server client," https://en.wikipedia.org/wiki/Client%E2%80%93server_model [último acceso 25/julio/2022].

[26] F. Espinosa and C. Losada, *Electronic Control SystemsDesign*, Universidad de Alcalá, 2020.

[27] "System identification toolbox," https://es.mathworks.com/products/sysid.html?s_tid=AO_PR_info [último acceso 30/agosto/2022].

[28] M. Amoozgar and Y. Zhang, "Trajectory tracking of wheeled mobile robots: A kinematical approach," in *Proceedings of 2012 IEEE/ASME 8th IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications*. IEEE, 2012, pp. 275–280.

[29] R. Fernández, R. Aracil, and M. Armada, "Control de tracción en robots móviles con ruedas," *Revista Iberoamericana de Automática e Informática industrial*, vol. 9, no. 4, pp. 393–405, 2012.

[30] "Explanation for obrevers in control theory," http://dea.unsj.edu.ar/control2/Clase09a-Observadores%20de%20estados.pdf [último acceso 30/agosto/2022].

Universidad de Alcalá
Escuela Politécnica Superior

ESCUELA POLITECNICA
SUPERIOR

Universidad
de Alcalá