Universidad de Alcalá
Escuela Politécnica Superior

GRADO EN INGENIERÍA ELECTRÓNICA DE COMUNICACIONES

Trabajo Fin de Grado

INTEGRACIÓN DE HERRAMIENTAS PARA DOCENCIA SEMI-PRESENCIAL DE CONTROL DE ROBOT P3-DX

Autor: Fernando Martínez Ardid

Tutor/es: Felipe Espinosa Zapata

2021

UNIVERSIDAD DE ALCALÁ
Escuela Politécnica Superior

GRADO EN INGENIERIA ELECTRÓNICA DE COMUNICACIONES

Trabajo Fin de Grado

INTEGRACIÓN DE HERRAMIENTAS PARA DOCENCIA SEMI-PRESENCIAL DE CONTROL DE ROBOT P3-DX

Autor: Fernando Martínez Ardid

Tutor/es: Felipe Espinosa Zapata

TRIBUNAL:

Presidente: José Manuel Villadangos

Vocal 1º: Ana Jiménez Martín

Vocal 2º: Felipe Espinosa Zapata

FECHA: septiembre 2021

Dedicado a Andrés, Yolanda y Sete.

Dedicado a mi familia y amigos.

Gracias a Felipe por enseñarme la parte de la electrónica que me gusta.

ÍNDICE

ÍNDICE DE FIGURAS

I. RESUMEN

I.1 RESUMEN

La limitación de recursos de laboratorio en escuelas de ingeniería, así como la demanda de una mayor disponibilidad y flexibilidad de acceso a los mismos, para un amplio número de estudiantes, está potenciando la modalidad semipresencial en esta rama de estudios universitarios.

En este Trabajo de Fin de Grado se hace una propuesta de adaptación de prácticas presenciales de control de robots P3-DX, utilizados en diferentes asignaturas y grados de la EPS (UAH), para que se pueda realizar la mayor parte de estas de manera remota, monitorizando (audio y video) la respuesta del robot a la actividad desarrollada por el estudiante.

I.2 PALABRAS CLAVE

Formación semipresencial, laboratorio de control, robot P3-DX, control remoto, supervisión remota.

I.3 ABSTRACT

The limitation of laboratory resources in engineering schools, as well as the demand for greater availability and flexibility of access to them, for many students, is promoting the blended modality in this branch of university studies.

In this Final Degree Project, a proposal is made for the adaptation of face-to-face practices of control of P·-DX robots, used in different subjects and degrees of the EPS (UAH), so that most of them can be carried out in a remote, monitoring (audio and video) the response of the robot to the activity developed by the student.

I.4 KEY WORDS

Blended training, control laboratory, P3-DX robot, remote control, remote supervision.

II. MEMORIA

II.1. INTRODUCCIÓN. MOTIVACIÓN.

La pandemia derivada del Covid-19 ha desembocado en una crisis mundial que afecta absolutamente a todo lo que se denominaba normalidad. Durante pocos meses todas las personas han tenido que renovarse y adaptarse a una situación cotidiana excepcional. De igual manera, en el ámbito de la docencia, se ha requerido un cambio de metodologías: clases online, exámenes online, realización de prácticas de manera remota, etc.

Mas allá de la situación de emergencia sanitaria, las metodologías de enseñanza presencial han evidenciado carencias importantes especialmente en enseñanzas universitarias con una importante componente experimental: restricciones horarias de acceso a laboratorio, limitación de recursos para la formación práctica, dificultad de compartir espacios y recursos con otros centros de formación, etc.

Las limitaciones comentadas de la formación clásica en ingeniería se han visto, en parte, compensadas por la proliferación de herramientas software de simulación y emulación de procesos físicos en los que se trabaja con modelos de la realidad. Aun reconociendo su contribución, el estudiante de ingeniería ha de demostrar las habilidades y competencias adquiridas sobre demostradores reales y, por tanto, parte de los resultados de aprendizaje se han de evaluar tras los ensayos con prototipos reales.

Por una u otra razón, la formación semipresencial está en el centro de atención de la mayoría de los proyectos de innovación docente, como ha quedado patente en el XIII Encuentro de Innovación en Docencia Universitaria (EIDU) organizado por la UAH y celebrado los días 1 y 2 de junio de 2021.

La motivación principal de este TFG es la adaptación de prácticas de sistemas electrónicos de control de robots, introduciendo una importante componente semipresencial manteniendo las competencias y resultados de aprendizaje marcadas en las guías docentes de las asignaturas implicadas, como es el caso de Ingeniería de Control Electrónico en el Grado de Ingeniería Electrónica y Automática Industrial o Control Industrial en el Grado en Ingeniería Electrónica de Comunicaciones.

Con este TFG se pretende cubrir parte de los objetivos fijados en el proyecto de innovación docente EASE-I: Metodología y herramientas de apoyo a la docencia en Electrónica (Ver Anexo I), que forma parte de la propuesta global reflejada en la figura 1. Entre los objetivos concretos están:

- Desarrollar acciones de innovación y renovación de metodologías educativas facilitando la realización remota de prácticas sobre prototipos reales de laboratorio integrando conocimientos de varias disciplinas.

- Contribuir a una mejora en la calidad de la docencia en ingeniería en la UAH, de forma que se aumenten las horas de disponibilidad de la instrumentación y prototipos de ensayo para los estudiantes de forma remota, para la mejora de la adquisición de las competencias previstas en su titulación.

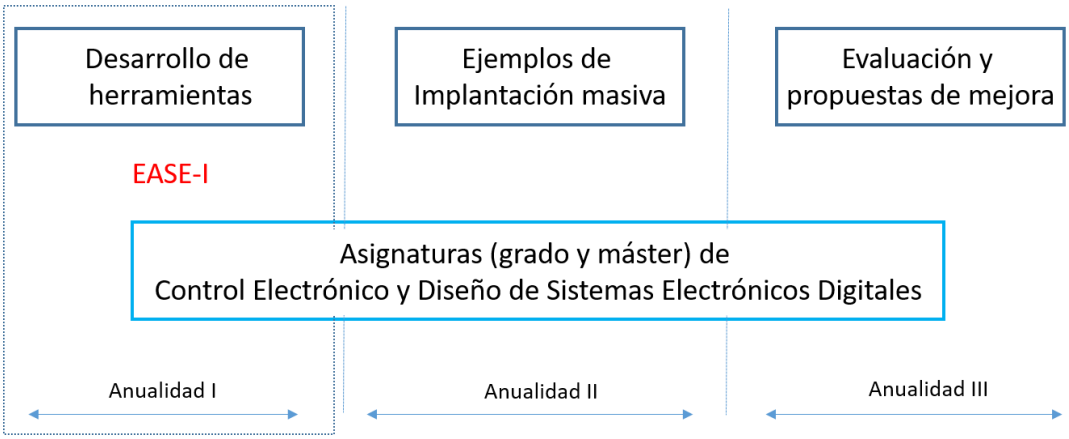


Figura 1: Propuesta global EASE y específica para la primera anualidad EASE-I

En este contexto, el TFG se centra en la asignatura de Control Industrial de GIEC. En la figura 2 se muestran elementos propios de este laboratorio con el que los estudiantes realizan sus prácticas.

Figura 2: Componentes de prácticas presenciales con robots P3-DX en asignaturas de control electrónico.

Para la consecución de los citados objetivos, se plantean las siguientes fases:

1. Identificar qué parte de las prácticas son susceptibles de adaptación a una ejecución remota y cómo complementarla con sesiones presenciales.

2. Analizar el software y el hardware requerido en cada práctica para realizar la propuesta de ejecución remota accediendo a prototipos reales.

3. Introducir elementos de supervisión que permitan tener una realimentación de la práctica realizada.

II.2. PLATAFORMA DE PARTIDA

La ingeniería de control forma parte del currículo de diferentes titulaciones de grado y máster. A modo de ejemplo, la Escuela Politécnica Superior (UAH) oferta: Grado en Ingeniería Electrónica de Comunicaciones, Grado en Ingeniería Electrónica y Automática Industrial, Máster U. en Ingeniería Industrial, Máster U. en Ingeniería Electrónica. En todas ellas el/la estudiante cuenta con asignaturas que requieren el acceso de, al menos, dos horas semanales de prácticas de control electrónico.

Una vez se han explicado y analizado los fundamentos teóricos en el aula, se requiere la demostración práctica, con prototipos reales, en laboratorio. Los laboratorios en la EPS, generalmente, son espacios con 12 puestos en los que el/la estudiante realiza su actividad de forma individual o en pareja. La posibilidad de extender el horario de acceso a estos espacios es una demanda recurrente entre el estudiantado.

Si nos centramos en un laboratorio de control electrónico, las diferentes etapas formativas se muestran en la figura 3. En rojo se indican aquellas etapas que requieren interacción con prototipos electrónicos o robóticos, mientras que en verde se muestran aquellas que requieren procesamiento y/o simulación con herramientas software disponibles en el laboratorio y, según el tipo de licencia requerida, también en el PC del/la estudiante.

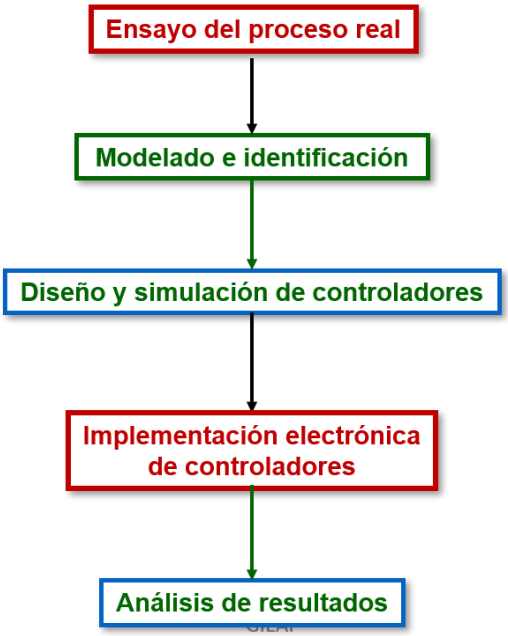


Figura 3: Etapas para la realización de prácticas de laboratorio de las asignaturas de Control Electrónico

En la figura 4 se muestra el caso de un puesto de trabajo de la asignatura Control Industrial (GIEC) donde el prototipo de ensayo es un robot P3-DX, controlado remotamente desde un PC de sobremesa, ambos conectados a la misma red WLAN.

Tanto en la fase de ensayo en lazo abierto como en la de implementación electrónica de controladores (figura 3) el/la estudiante interacciona periódicamente con el robot, ejecutando los algoritmos diseñados en el PC de sobremesa habilitado pare estas tareas.

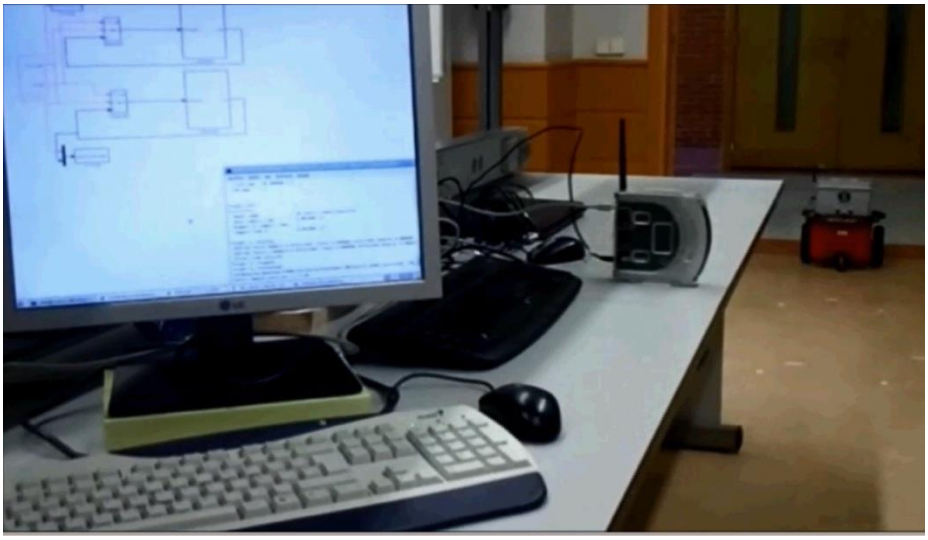


Figura 4: Puesto de trabajo de la asignatura de Control Industrial

El puesto de laboratorio básicamente está compuesto por un PC, un robot y una red WLAN, las cuales se explicarán detalladamente a continuación.

II.2.1. PC REMOTO

El elemento principal de actuación, en lazo abierto y en lazo cerrado, es un PC situado en el laboratorio de control y conectado inalámbricamente con el proceso a controlar de forma remota.

El alumno genera un archivo ejecutable a partir de la simulación del sistema diseñado completo de manera individual usando MATLAB/SIMULINK. Dicho archivo ejecutable se lanza en el ordenador remoto, que envía la actuación (velocidad lineal y angular) al robot para realizar todas las pruebas pertinentes de la práctica para su posterior estudio una vez recibidas las velocidades de respuesta del robot.

Este ordenador está conectado a la misma red que el robot a través de SSH haciendo uso del programa PuTTY.

II.2.1.1. LINUX O WINDOWS

El sistema operativo que utiliza el ordenador local del laboratorio es Windows 10. Cabe la posibilidad de que el sistema que pueda tener este ordenador sea Linux en su versión UBUNTU. Esto implica que, el driver que utiliza Matlab para generar el ejecutable de interacción con el robot cambia por las distintas librerías internas que tiene dicho sistema operativo.

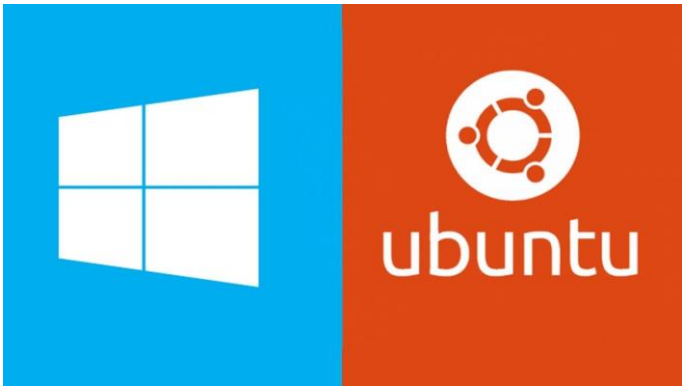


Figura 5: Linux vs Windows

En la siguiente tabla se resumen los aspectos diferenciadores de los sistemas operativos mostrados en la figura 5.

II.2.1.1.1 TABLA COMPARATIVA DE SISTEMAS OPERATIVOS: LINUX Y WINDOWS

Windows	Linux
Sistema operativo propietario (código cerrado) más común en aplicaciones	Sistema operativo libre (código abierto), poco común en general, excepto en aplicaciones de ingeniería.

domóticas y profesionales.	
Al ser un sistema operativo comercial facilita la configuración de periféricos.	La configuración de periféricos en Linux es más complicada pero facilita la personalización.
Proporciona una interfaz gráfica de usuario fácil e intuitivo.	Interfaz gráfica menos intuitiva, generalmente se trabaja a nivel de línea de comandos.
Costes de licencia por cada usuario	Sistema Operativo sin costes de licencia. Ubuntu es una de las distribuciones más extendidas.
Soporta programas habituales en gestión, incluyendo las aplicaciones de Microsoft.	Ofrece aplicaciones equivalentes a Windows y Office, pero la portabilidad con los anteriores no está garantizada.
Dentro de MATLAB, se dispone unos compiladores de código más sencillos y comunes.	Los compiladores de código dentro de MATLAB son mucho más específicos y complejos de usar.

Trabajando con Matlab/Simulink, según el sistema operativo, cambia el modo de compilación y la ejecución del fichero de control.

La solución Windows 10 para el desarrollo de aplicaciones de control con Matlab/Simulink evita que el alumno haya de recurrir a una máquina virtual compatible con Linux.

El ordenador de laboratorio, a su vez, está dotado del programa de licencia libre PuTTY [PuTTY,2021], originalmente para Windows, el cual se utiliza como un cliente SSH realizando una conexión remota y segura con el robot utilizando dicho protocolo.

En la figura 6 se puede observar el contenido del PC con las aplicaciones básicas que permiten la ejecución de control remoto del robot.

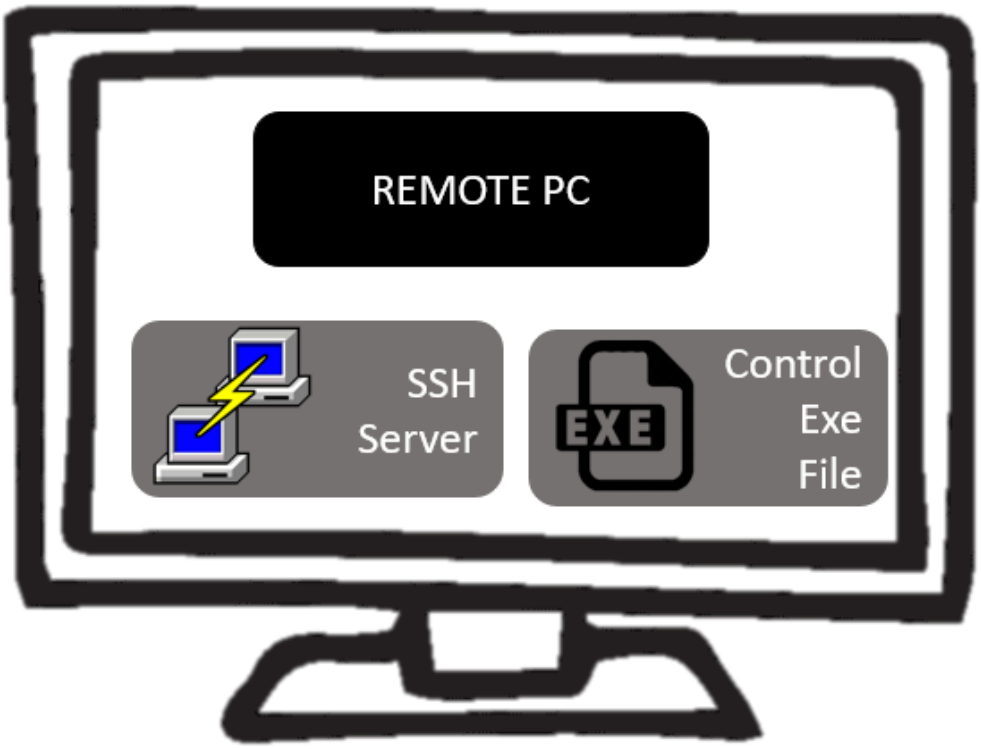


Figura 6: Contenido del PC del laboratorio utilizado en las clases presenciales de control remoto del robot.

II.2.1.2. DRIVER PC I

Para la interacción del PC con hardware externo, sea el robot o sensores externos, en MATLAB se recurre a una *s_function,* función escrita en C siguiendo la correspondiente plantilla y encapsulada en un bloque Simulink, véase el Anexo IV.

Este driver esta generado en MATLAB usando la librería WS2_32.lib con el código fuente para generar la *s-function*, La librería WS2_32.lib [Información sobre Bibliotecas DLL de Windows, 2021] es una librería que implementa la Winsock API, la cual provee una serie de funciones para implementar el protocolo TCP/IP y compatibilidad parcial con otras API's a través de sockets [Información sobre Sockets de Internet, 2021].

Winsock (Windows Socket) [Información sobre Winsocks, 2021] es una biblioteca con una serie de funciones que utiliza Windows para implementar el protocolo TCP/IP que permite la comunicación entre dos ordenadores de la misma red. Esta biblioteca incluye también soporte para envío y recepción de paquetes de datos a través de sockets (proceso o hilo que esta tanto en la máquina cliente como en la máquina servidora que sirve para que ambos programas tanto en el cliente como en el servidor lean y escriban información).

II.2.2. ROBOT

El robot utilizado es el robot P3-DX que es una base robótica de MobileRobots Inc. (ActivMedia Robotics), véase figura 7 [Robot P3-DX, 2021]. Desde el punto de vista físico el robot dispone de 2 ruedas activas y una libre. Los sensores que tiene son sonares en la parte delantera y encoders en las ruedas.

Las especificaciones del robot completas se pueden observar en el Anexo III.

Figura 7: Robot Pioneer 3-DX

II.2.2.1 SOFTWARE

En este apartado se van a comentar las distintas componentes software del robot en la configuración previa a la modificación introducida con este TFG.

II.2.2.1.1 LINUX

El sistema operativo que utiliza el robot es Linux con versión de UBUNTU 11.04, sobre el cual se ha generado el driver que se explica a continuación.

Este sistema operativo tiene que estar dotado de un servidor SSH [Instalar SSH en UBUNTU, 2021], al que se conecta el PC remoto para la activación de dicho driver.

Esta versión de UBUNTU fue lanzada en 2011, la cual ya no dota de servicio técnico y se puede considerar obsoleta.

II.2.2.1.2 DRIVER ROBOT

Este driver es generado a través de MATLAB sobre UBUNTU. Este driver abre los sockets del robot, para recibir los comandos del ejecutable remoto y actuar sobre el propio sistema de tracción diferencial, además envía las variables registradas de movimiento al PC remoto para cerrar el lazo de control

En el Anexo II se puede observar un ejemplo del código fuente y cabeceras que genera el driver para la recepción de datos en el robot y el correspondiente envío de variables al PC remoto.

II.2.2.2. HARDWARE

El hardware del sistema previo a este TFG está compuesto por una Vía Epia, un disco duro, un ethernet converter y un convertidor DC-DC.

II.2.2.2.1. VIA EPIA, DISCO DURO Y ETHERNET CONVERTER

El PC montado en el robot consta de una placa base Vía Epia [Via Epia, 2021] con procesador tipo VIA, más reducida y con menor consumo que un ordenador convencional, pero con los recursos de cálculo suficientes para gestionar los movimientos del robot.
Las características principales de la Vía Epia modelo VIA Nano-ITX NX15000G son:

- Procesador C7 1.5 GHz.
- Chipset CX700M
- 1 zócalo SODIMM x DDR2 533 con un máximo de 1 Gb de memoria.
- Sistemas compatibles como Windows xp, Windows Embedded CE, Windows Embedded Standar y Linux
- 1 Puerto LAN
- 1 ranura Mini.PCI
- 4 puertos USB
- 2 puertos RS232

Figura 8: Vía Epia Nano-ITX NX15000G

A esta Vía Epia se le conecta un disco duro [Disco duro, 2021] para el almacenamiento del Sistema Operativo y de los datos y su vez uso de un conversor de ethernet [Ethernet Converter, 2021].

Las características principales del disco duro Toshiba modelo MK8032GAX son:

- Capacidad de almacenamiento de 80 GB
- Interfaz de hardware IDE
- Firmware AD002D

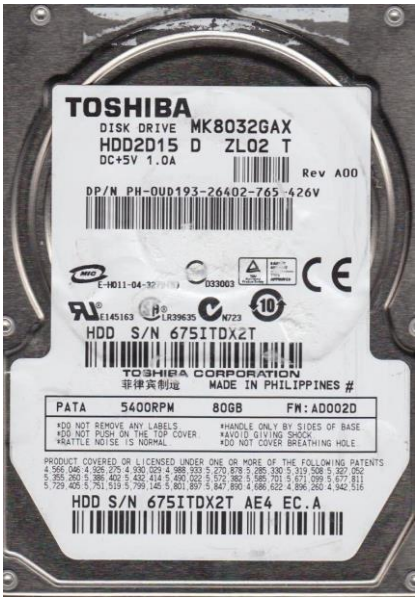


Figura 9: Disco Duro Toshiba MK8032GAX

Las características principales del ethernet converter son:

- 80211 g Wireless 125 tasas de transferencia de alta velocidad (modo Turbo g) cuando se utiliza con puntos de acceso 125 Modo de alta velocidad (Turbo g) o routers inalámbricos
- Cuatro puertos 10/100 integrados para hasta 4 dispositivos cableados simultáneas.
- Conexiones inalámbricas fácil y seguro con AOSS
- Se puede colocar en posición horizontal o vertical.

Figura 10: Ethernet Converter BUFFALO Wireless-G MIMO.

La colocación física de estos elementos hardware del sistema embarcado se puede observar en la figura 11.

Figura 11: Disposición de la Vía-Epia, Disco Duro y Ethernet Converter dentro de la caja del robot

II.2.2.2.2. CONVERTIDOR DC-DC

La Vía Epia requiere de alimentaciones de +12 VDC -12VDC y +5VDC. Para ello se hizo necesaria la incorporación de un convertidor DC/DC (figura 8) que proporcionase esas alimentaciones a partir de las baterías del Robot, tres módulos en paralelo de 12VDC [Baterias, 2021].

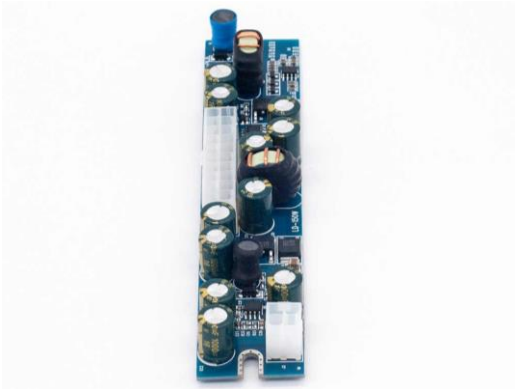


Figura 12: Convertidor DC-DC del sistema antiguo

El convertidor que tenía este sistema es un DC-DC de 12 V a ATX/ 150W RGEEK modelo RG120B [Convertidor DC-DC antiguo, 2021] cuyas características principales son:

- Salida de ±12V, +5V y 150 W.
- Interfaz de 20 + 4 pines
- 1 puerto SATA
- Potencia nominal <150W.
- Datos de corriente y voltaje:

ITEM	Output Voltage				Output Current		
	Min (V)	Avg (V)	Max (V)	Ripple Wave (mV)	Sb Current (A)	Avg (A)	Peak(A)
+3.30 V	3.14	3.3	3.47	50 MAX	0	4.0	6.0
+5.00 VSB	4.75	5	5.25	50 MAX	0.1	1.5	2.0
+5.00 V	4.75	5	5.25	50 MAX	0	4.0	6.0
+12.00 V	11.4	12	12.6	120 MAX	0	5.0	8.0
-12.00 V	-11.4	-12	-12.6	120 MAX	0	0.1	0.2

Figura 13: Datos de Tensión y Corriente del convertidor DC-DC RGEEK RG120B

II.2.3. RED WLAN

El PC, el robot y los sensores externos (cámaras) son nodos de una misma red local. Por tanto, se ha de configurar la IP de cada nodo, de manera que se establezca la conexión inalámbrica necesaria entre ellos, especialmente entre el Robot y el PC remoto.

II.2.3.1. ROUTER BUFFALO

Router configurado para generar la red local (COVE), a la que están conectados todos los nodos comentados, incluido el robot bajo estudio.

Figura 14: Router Buffalo antiguo.

Este router buffalo WHR-HP-G54-DD (figura 14) [Router antiguo, 2021] tiene las siguientes características:

- Algoritmo de seguridad WPA2.
- Intervalo de frecuencia 2.412 – 2.472 GHz.
- Consumo energético de 3.4 W.
- Dimensiones: 28 x 130 144 mm.
- Fuente de alimentación 100 – 240 V, 50-60 Hz.

II.3. PLATAFORMA MODIFICADA

Todo el sistema explicado anteriormente, va a ser modificado en parte, para adaptarse, tanto a sistemas hardware más eficientes, como a las

actualizaciones recientes de software. Se busca modernizar en parte el sistema de la manera que se va a explicar a continuación.

II.3.1. PC REMOTO

El PC local (laboratorio) está conectado a una doble red. Red local (WLAN) de la que el robot es también un nodo, y red externa para permitir el acceso remoto del estudiante trabajando en modo semipresencial. Las principales actualizaciones en este PC se explican en los siguientes apartados.

II.3.1.1. WINDOWS 10

El sistema operativo desde el que vamos a trabajar es Windows 10. Este ordenador es el encargado de ejecutar el archivo .exe generado por el alumno en MATLAB/SIMULINK, para mandarle las consignas de velocidad lineal y velocidad angular al robot, bien en lazo abierto bien en lazo cerrado.

Figura 15: Logo del Sistema Operativo Windows 10

Este PC, haciendo uso del programa PuTTY, se conecta mediante SSH (Secure SHell) [Información sobre Secure Shell (SSH), 2021] con un servidor creado en el robot. En primer lugar, se ejecuta el driver (ejecutable) ubicado en el robot, para abrir los sockets esperando a recibir las consignas desde el PC remoto y devolviendo los registros del movimiento.

Tras la ejecución, en el ordenador quedan registradas las variables programadas en el proyecto de control y generadas en tiempo real, tanto en el PC como en el robot. Terminada la ejecución, se pueden analizar dichas variables, por ejemplo, para comparar resultados de simulación y de ensayo experimental.

II.3.1.2. DRIVER PC II

Para generar el archivo ejecutable con Matlab/Simulink para Windows partimos de cómo se generaba el driver en el sistema anterior, haciendo uso de las mismas librerías nombradas en el apartado *Driver PC I dentro de PC remoto en el apartado de Plataforma de partida.*

Se continúa haciendo uso de la librería WS2_32.lib y del Winsock para generar el ejecutable, a través del driver en el PC remoto.

Se puede observar el código fuente del driver actualizado en el anexo IV.

II.3.2. ROBOT

La plataforma robótica del P3-DX mostrada en la figura 16 se sigue manteniendo.

Las actualizaciones llevadas a cabo en este TFG tienen que ver con la arquitectura hardware y software embarcada.

Figura 16: Robot Pioneer 3-DX

II.3.2.1. HARDWARE

II.3.2.1.1. MINIPC

El elemento principal de procesamiento e interfaz hardware con la plataforma básica del robot es un miniPC *ubicado tal y como se muestra en la figura 17.*

Figura 17: Sistema físico completo con miniPC en la plataforma robótica P3-DX

Se trata de un Intel NUC, modelo BXNUC8i5INHX, *ver figuras 18 y 19* [MiniPC, 2021] reduciendo peso y volumen, a la vez que mejorando las prestaciones de la Vía Epia.

Las especificaciones técnicas detalladas del miniPC se encuentran en la referencia bibliográfica.

La conexión entre en miniPC y la plataforma del P3-DX se realiza mediante un cable serie modelo US232R-100-BULK (figura 19) [Cable serie, 2021].

Figura 18: MiniPC Intel-NUC Perfil

Figura 19: MiniPC Intel-NUC desde atrás.

Figura 20: Cable serie US232R-100-BULK

Las características principales del BXNUC8i5INHX se indican a continuación:

- Procesador Intel Core i5-8265U
- Contiene dos slots para hasta 16 GB de memoria RAM DDR4 donde se ha añadido en este caso un módulo de 8GB.
- Almacenamiento de 1 TB
- Salida de video HDMI 2.0
- SDXC card slot.
- Sistema operativo preinstalado Windows 10, 64 bits.
- Dimensiones de 117 x 112 x 51 mm.

II.3.2.1.2. CONVERTIDOR DC-DC

La fuente de alimentación del hardware propio y embarcado del robot son las baterías del propio robot, tres módulos de 12 V y 9 A/h conectados en paralelo.

Para cumplir con las especificaciones de alimentación del miniPC, 19 VDC, se necesita un convertidor DC-DC elevador.

Figura 21: Convertidor DC-DC del sistema nuevo

En el sistema se va a utilizar un convertidor elevador DC-DC de la marca TECNOIOT [Convertidor DC-DC nuevo, 2021] recibiendo una tensión de entrada de 12 V y ajustando los valores de salida a 19 V y 12 A.

Las características del convertidor son las siguientes:

- Módulo de aumento no aislado (BOOST).
- Voltaje de entrada: DC 8.5V-50V.
- Corriente de entrada: 15A máximo.
- Corriente en reposo: 10mA.

- Voltaje de salida: 10-60V.
- Rango Contante: 0,2-12A
- Temperatura: -40 a +85º
- Frecuencia de funcionamiento: 150 KHz
- Eficiencia de conversión: Hasta un 96%
- Protección contra sobre corriente: Si
- Protección de polaridad inversa de entrada: No.
- Dimensiones: 67mm x 48mm x 28mm
- Peso: 60g.

II.3.2.2. SOFTWARE

II.3.2.2.1. LINUX

Dentro del miniPC el sistema operativo instalado es Linux, versión UBUNTU 20.04 LTS. En la carpeta target está ubicado ejecutable *robot* del que hablaremos próximamente.

Figura 22: Logo Sistema Operativo que se utilizará dentro del miniPC.

II.3.2.2.2. EJECUTABLE ROBOT

El ejecutable *robot* esta generado a partir de un proyecto Matlab/Simulink (versión para Linux) haciendo uso de una conexión UART (Puerto Serie). Dentro del miniPC no es estrictamente necesario que se encuentra MATLAB/SIMULINK instalado, pues el ejecutable puede generarse en otra ubicación de MATLAB con el mismo sistema operativo que el de miniPC.

El proyecto simulink incluye el driver *Comunica_dr_cam_w10*, con una parte común para todos los miniPCs/robots y una parte específica relacionada con la IP propia.

El código fuente del driver utilizado para generar el driver del robot (ficheros *robot_dr.c y robot_dr.h*) se describe en el Anexo V.

II.3.3. RED WLAN

La red local de conexión entre las distintas partes del sistema (robots, PC de control, cámaras, etc.).

II.3.3.1. ROUTER NUEVO

Se ha sustituido el Router Buffalo por el ARCHER C6 [Router Nuevo, 2021] con la misma funcionalidad que el anterior, pero con mejores prestaciones.

Figura 23: Router para el sistema nuevo

Las características de este router son:

- Admite el estándar 802.11ac.
- Conexiones simultáneas de 2.4GHz 300 Mbps y 5GHz 867 Mbps para 1200 Mbps de ancho de banda total disponible.
- 4 antenas externas y una antena interna proporcionan conexiones inalámbricas estables y una cobertura óptima.
- Fácil administración de la red al alcance de su mano con TP-Link Tether.
- MU-MIMO logra el doble de eficiencia al comunicarse con hasta 2 dispositivos a la vez.
- La tecnología Beamforming ofrece una cobertura inalámbrica más amplia.
- Admite el modo de punto de acceso para crear un nuevo punto de acceso Wi-Fi.

II.4. ACCESO REMOTO AL PUESTO DE LABORATORIO

La metodología de asignaturas impartidas en modalidad semipresencial incluye la posibilidad de que el estudiante pueda realizar prácticas sin tener que acudir presencialmente al laboratorio. Con esta idea, en este TFG se ha desarrollado un entorno que facilite el acceso remoto a los recursos de un laboratorio de control.

Una vez descritos los elementos físicos disponibles en el laboratorio se necesita contar con una herramienta de forma que el alumno, desde su domicilio o desde su puesto de trabajo, pueda acceder a los recursos del laboratorio de una manera ordenada.

Para facilitar esto proponemos varias alternativas: ANYDESK, software libre para uso particular pero no institucional; y mediante red privada virtual (Virtual Private Network -VPN-) de la UAH.

II.4.1. ANYDESK

Existe una amplia cantidad de programas que permiten al usuario conectar dos ordenadores a través de internet, haciendo uso de una clave individual que cada ordenador tiene asignada. Accediendo con esa clave el usuario puede controlar el ordenador de manera remota.

la figura 10 se puede ver la sencillez de la conexión entre dos ordenadores a través del software ANYDESK.

En este proyecto, el programa que vamos a utilizar es ANYDESK por los siguientes motivos:

- Permite cifrar la clave de acceso, esto quiere decir que, la conexión con el PC remoto se puede activar en cualquier momento únicamente teniendo el código del PC y la clave, la cual generara el administrador del sistema.

- Facilita la transferencia de archivos entre ambos ordenadores, lo cual simplifica el intercambio de los datos que el PC remoto reciba del robot.

- Tiene versiones para distintos sistemas operativos, y la propia instalación detecta el sistema utilizado en el acceso a la página oficial del programa, con lo que la instalación es muy sencilla.

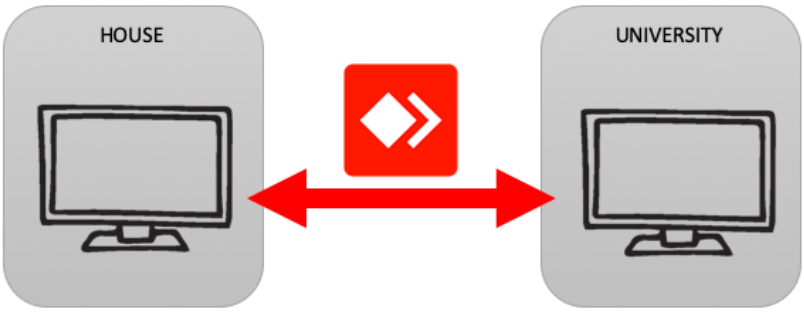


Figura 24: Conexión remota a través de ANYDESK

Además, ANYDESK presenta otra utilidad de interés. Mediante una ventana emergente en el PC local (figura 24), el administrador de este gestiona la conexión, de forma que puede admitir o no una solicitud de acceso, e incluso desconectar una conexión en curso. La ventana tiene un chat, para la interacción entre usuario remoto y administrador local. La clave de acceso del usuario remoto puede ser modificada en cualquier momento por el administrador.

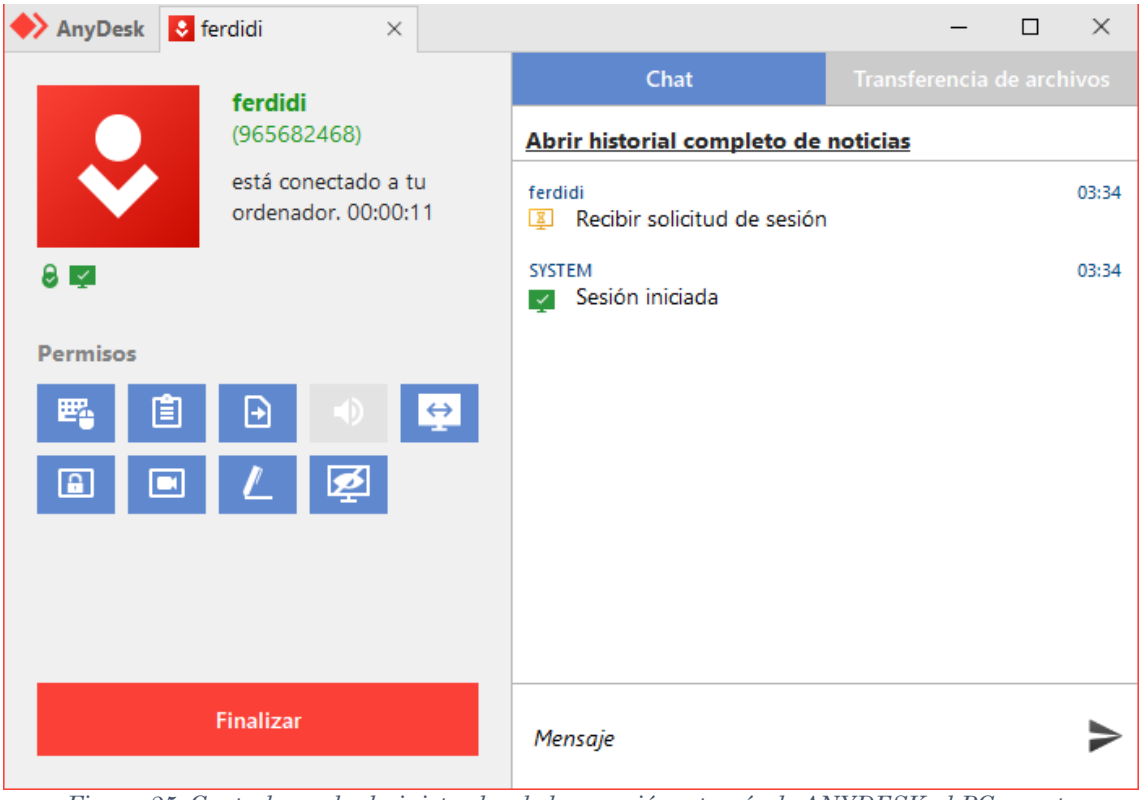


Figura 25: Control por el administrador de la conexión a través de ANYDESK al PC remoto

En el caso de la aplicación abordada en el TFG, el profesor de laboratorio, como administrador, gestiona la disponibilidad del PC del laboratorio para acceso remoto con usuarios ANYDESK.

En resumen, aunque la conexión pueda ser independiente de cada alumno, en el momento que ocurra algún problema o alguna conexión simultanea

se podrá tanto expulsar como eliminar una conexión. (El tema de la gestión de conexiones se tratará en apartados posteriores).

En el anexo VI se pude observar una guía de instalación del software ANYDESK tanto para Windows como para MacOS.

II.4.2. AUDIO Y VIDEO DE MANERA REMOTA

Para que el alumno compruebe de manera semipresencial que el robot responde, se le va a proporcionar una cámara Kinect para XBOX 360 modelo LPF-00057 [Cámara Kinect, 2021] para poder recibir tanto audio como video del robot (figura 26).

Figura 26: Cámara Kinect para XBOX360

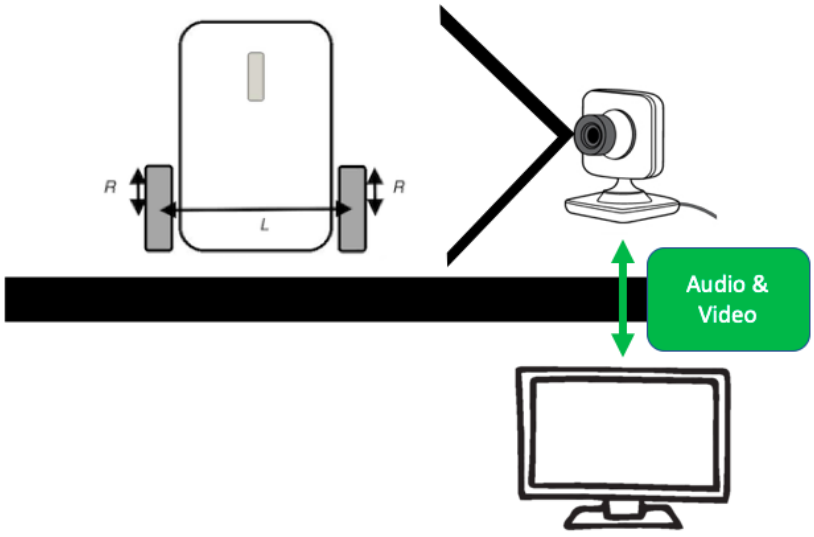


Figura 27: Disposición de la cámara dentro del laboratorio remoto

Esta cámara necesita un adaptador especifico ya que de fábrica viene con un conector especifico de XBOX, en este caso es un ESUMIC modelo VDF-XBOX-ADPT-EU [Adaptador Kinect para PC, 2021].

Figura 28: Adaptador para PC de la cámara Kinect

En el anexo VII se puede observar cómo realizar el acceso dentro del ordenador remoto a la cámara Kinect.

II.4.3. VPN

Se puede barajar otra opción de conexión con el PC remoto haciendo uso de la VPN de la UAH, como es habitual en algunas asignaturas que utilizan herramientas software de licencia limitada instalada en PCs concretos de un laboratorio.

Para ello hay programas, como FortiClient, con licencia UAH, cuyo tutorial indica el proceso de instalación y uso. Estando conectado a la VPN se tiene acceso a licencias completas de programas, es decir, es como si se estuviera conectado directamente a la red de la universidad en el PC que el usuario decida.

Establecida la conexión VPN en el PC remoto, se puede acceder al PC de laboratorio con IP fija, mediante conexión a Escritorio Remoto. El 'Escritorio Remoto' es una aplicación Windows que permite una conexión segura a través de internet a un PC local e IP fija, pasando el usuario remoto a tener el control de teclado y ratón del puesto local.

Una vez accedido al PC local, en este caso el PC del laboratorio, para entrar en la red local COVE, utilizamos la herramienta PuTTY ya descrita anteriormente. De esta forma se tiene acceso a cada uno de los nodos de la red, robots y cámaras.

Dentro de la bibliografía [Conexión VPN site-to-site, 2021] de este Trabajo de Fin de Grado se puede observar la forma correcta para conectarse a un entorno de escritorio remoto a través de VPN site-to-site en 2 sencillos pasos.

En la figura 23 se puede observar un pequeño esquema de cómo se realizaría la conexión a un escritorio remoto a través de una VPN site-to-site.

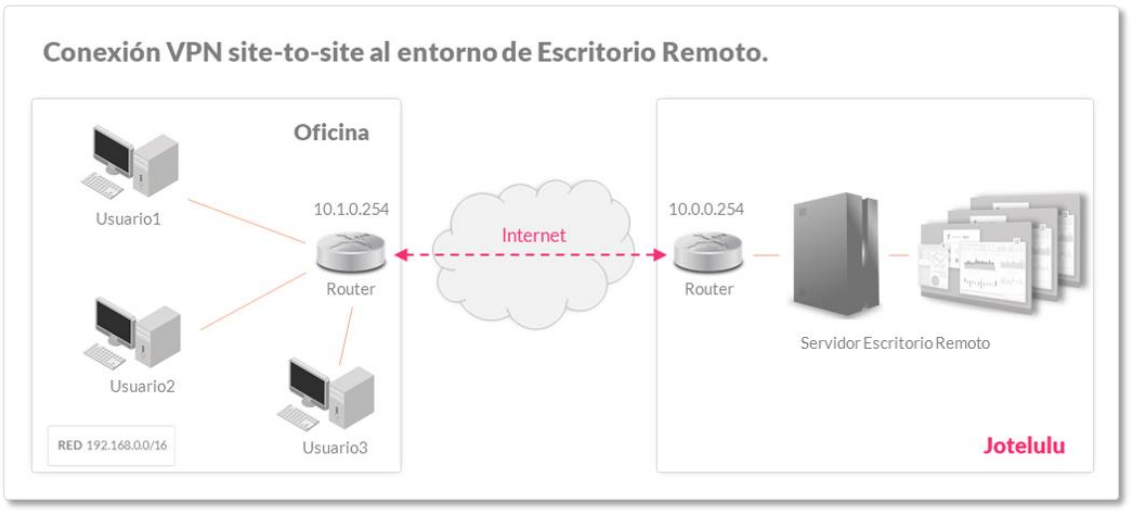


Figura 29: Conexión a escritorio remoto a través del uso de una VPN

II.4.4. PUESTA A PUNTO DEL SISTEMA

El estudiante tiene que comprobar el correcto funcionamiento con una práctica demo, a partir de ahí, podrá experimentar con su propuesta.

En el siguiente diagrama se van a indicar los pasos a seguir dentro de todo el sistema para el funcionamiento del sistema:

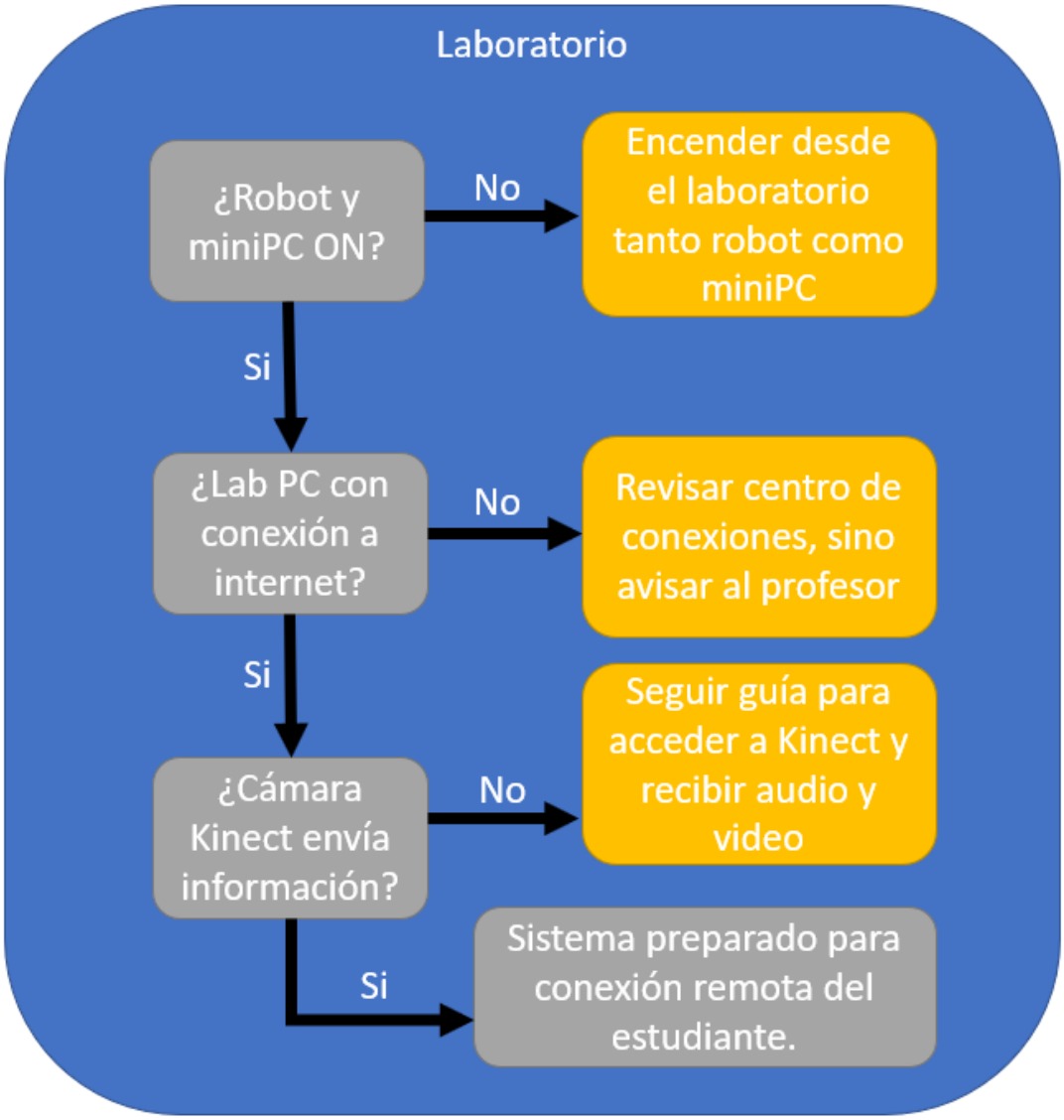


Figura 30: Diagrama de Flujo para el uso del PC de Laboratorio

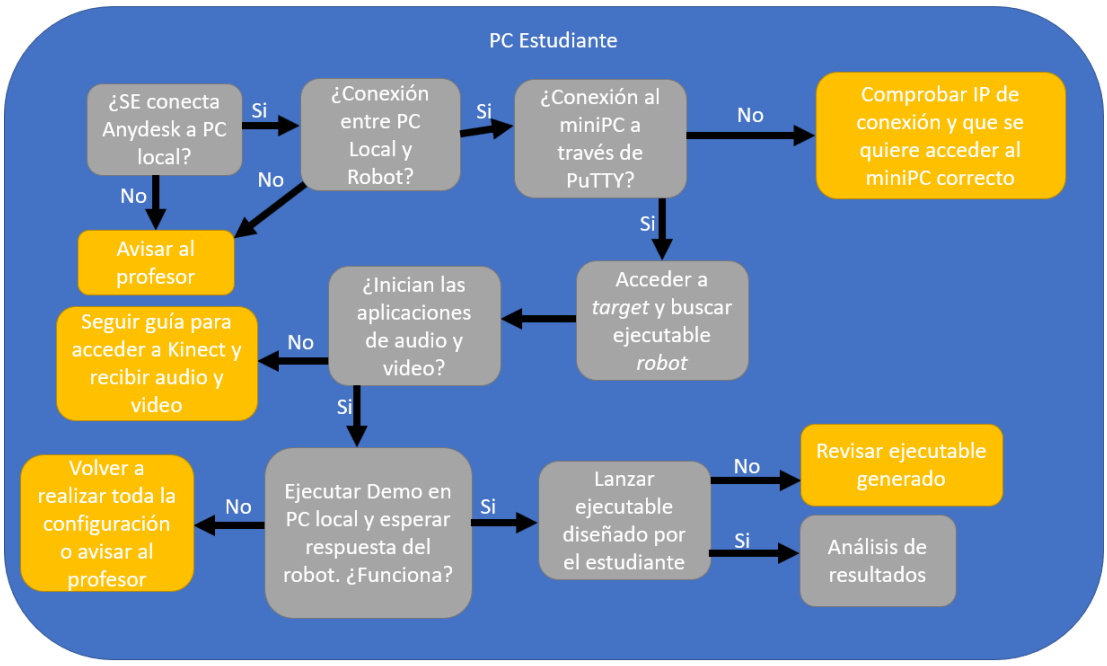


Figura 31: Diagrama de Flujo para el uso del sistema a través del PC del estudiante

1) LABORATORIO:

   a) Encender el robot y el miniPC.

   b) Comprobar que el PC de laboratorio está conectado a internet.

   c) Comprobar que la cámara Kinect está conectada, con audio y video activos.

2) PC ESTUDIANTE:

   a) Realizar la conexión al PC local del laboratorio a través de ANYDESK.

   b) Comprobar la conexión entre PC local y Robot con un ping a través de los símbolos del sistema.

   c) Con la aplicación PuTTY, conectarse a través de SSH con el miniPC.

   d) Dentro del miniPC, haciendo uso de comandos Linux, entrar en la carpeta *target* y ejecutar el programa *robot*.

   e) Iniciar las aplicaciones de audio y video de la cámara.

   f) Ejecutar el programa *Demo* y comprobar la respuesta del robot:

      (1) con la cámara, mientras dura la ejecución,
      (2) analizando las variables registradas, una vez terminada la ejecución.

g) Si el funcionamiento es correcto, lanzar el ejecutable diseñado por el estudiante. En caso de error, consultar al profesor.

h) Si la ejecución es correcta, comprobación y análisis de resultados.

II.4.5. HERRAMIENTA DE GESTIÓN DE TIEMPOS DE TRABAJO Y ASIGNACIÓN DE RECURSOS DEL LABORATORIO

Para la ejecución del sistema de una manera automática y ordenada, se necesitaría una herramienta que gestione las conexiones y sobre todo las horas a las que se puede disponer del puesto en el laboratorio.

Una solución es aprovechar la herramienta de BlackBoard de la Universidad de Alcalá. Se ha de generar una tabla horaria a la que los alumnos deberán apuntarse, evitando solapes, y en coordinación con el profesor de la asignatura.

El estudiante podrá utilizar las herramientas de diseño como MATLAB/SIMULINK disponibles en su propio PC o en el del laboratorio.

En la figura 24 se va muestra un ejemplo para solucionar la gestión de tiempos.

Horario	Lunes	Martes	Miércoles	Jueves	Viernes
8:00				Alumno 5	
9:00		Alumno 9			
10:00	Alumno 1				
11:00					Alumno 8
12:00			Alumno 2		
13:00					
14:00	DESCANSO				
15:00					
16:00					
17:00		Alumno 10		Alumno 4	
18:00	Alumno 3				
19:00					Alumno 6
20:00		Alumno 7			

Figura 32: Ejemplo de tabla para la gestión del espacio de trabajo en el laboratorio remoto

II.5. RESUMEN DE PRÁCTICA SEMIPRESENCIAL

El objetivo del TFG es diseñar un entorno de trabajo para que el alumno tenga acceso a los recursos disponibles en laboratorios de la EPS (UAH), concretamente en el de Control Electrónico del Departamento de Electrónica.

En el caso de realizar prácticas con el robot P3-DX, el alumno podrá trabajar, con las herramientas de diseño y simulación, bien con MATLAB/SIMULINK en su propio PC o en el del laboratorio. El ensayo de soluciones de control sobre el propio robot se hará en dos fases:

- De forma no presencial, on-line, con el robot operativo, pero sin estar en contacto con el suelo. El estudiante se conecta vía internet al PC de laboratorio, y después a la red local COVE de la cual forma parte tanto el PC como el robot. De esta forma se puede comprobar la viabilidad del diseño, corregir posibles fallos y realizar propuestas de mejora. Téngase en cuenta que además de los registros, el estudiante puede ver y escuchar de forma remota el comportamiento del robot en el ensayo.

- De forma presencial, con el robot listo para realizar las trayectorias programadas en lazo cerrado y en el entorno disponible en el laboratorio y zona anexa. La dedicación a esta fase se reducirá notablemente si se consigue una solución depurada en la fase anterior.

Haciendo alusión a todo lo explicado en apartados anteriores, en la *figura 25* se observa el funcionamiento del sistema antiguo. El alumno realiza su diseño y simulación en su propio PC, genera el ejecutable y a través de una memoria USB (pendrive) lo instala en el PC de laboratorio conectado de forma inalámbrica al robot bajo estudio.

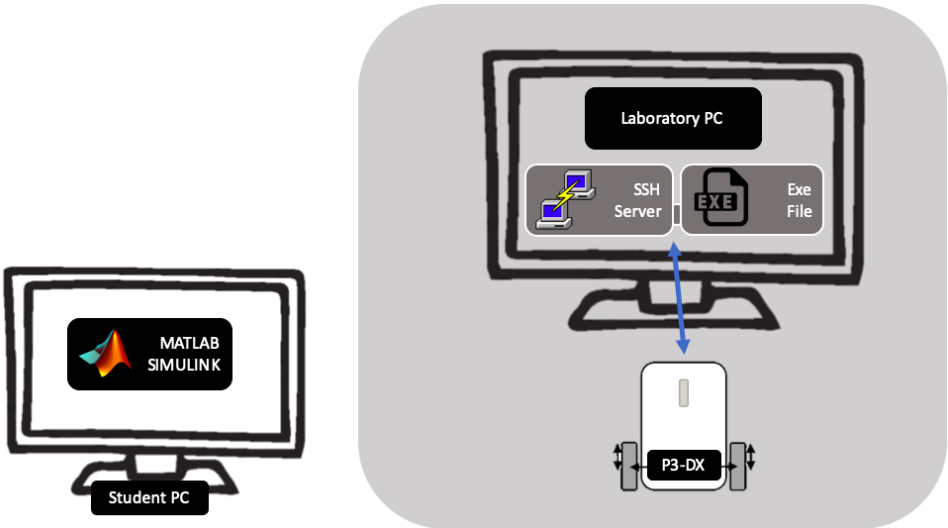


Figura 33: Esquema del funcionamiento del sistema antiguo.

En la *figura 26* se pueden ver las mejoras introducidas con este TFG. Haciendo uso de internet el estudiante se conecta (por ejemplo, con herramienta ANYDESK), al PC local del laboratorio. Este PC forma parte de la red inalámbrica a la que está conectado el robot y sensores externos. Además, al mismo PC se conecta una cámara Kinect que facilita el seguimiento visual y sonoro del ensayo realizado con el robot operativo, pero con las ruedas motrices al aire...

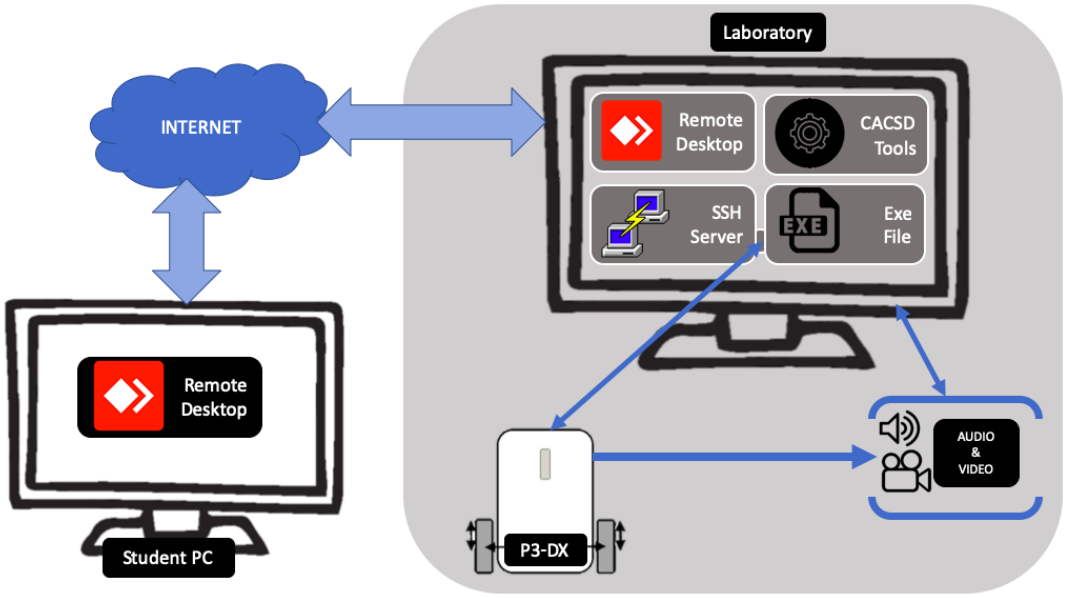


Figura 34: Esquema del funcionamiento del sistema nuevo.

II.6. CONCLUSIONES

Parte de la innovación en la formación universitaria pasa por la modalidad semipresencial, especialmente para la realización de prácticas en laboratorio. Con ello se consigue optimizar la utilización de los recursos materiales (ordenadores, equipos específicos de asignaturas, demostradores reales, etc.). Para conseguir este objetivo se requiere, acondicionar los puestos de trabajo y realizar una gestión adecuada de permisos y tiempos, garantizando una ejecución segura de las prácticas.

Este Trabajo de Fin de Grado presenta una solución semipresencial para la realización de prácticas de control electrónico en la EPS, adaptando los equipos existentes e incorporando elementos hardware y software para disponer de un demostrador real.

Como trabajo futuro se plantea la necesidad de contar con una herramienta de gestión de uso del puesto o puestos similares a los presentados en este TFG. Garantizando aspectos como: asignación ordenada de franjas temporales, la solución de posibles conflictos o solapes, la identificación de los usuarios, etc. Por otra parte, extender la disponibilidad de los puestos de

trabajo exige el mantenimiento y supervisión adecuado de los mismos por parte del departamento correspondiente

II.7. REFERENCIAS BIBLIOGRÁFICAS

[PuTTY, 2021] "Download PuTTY: latest release (0.76)." https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html (accessed Sep. 11, 2021).

[Via Epia, 2021] "Via Nano-ITX EPIA NX15000G - Ibertrónica." https://ibertronica.es/via-nano-itx-epia-nx15000g (accessed Sep. 11, 2021).

[Disco duro, 2021] "MK8032GAX, HDD2D15 D ZL02 T, Toshiba 80GB IDE 2.5 Disco Duro : Amazon.es: Informática." https://www.amazon.es/MK8032GAX-HDD2D15-ZL02-Toshiba-Disco/dp/B00MBN9OVA (accessed Sep. 11, 2021).

[Ethernet Converter, 2021] "BUFFALO Wireless-G MIMO Performance Ethernet Converter - Accesorio de Red (Color Blanco, IEEE802.11b/IEEE802.11g, Inalámbrico, Ethernet, 125, 100, 3-40 °C) : Amazon.es: Informática." https://www.amazon.es/Buffalo-Wireless-G-Performance-Ethernet-Converter/dp/B000BNDEZY (accessed Sep. 11, 2021).

[Baterias, 2021] "Batería 12 Voltios 9 Amperios Energivm MVH1290F2 - Baterias para todo Reguero Baterias." https://www.reguerobaterias.es/p90037558_bateria-12-voltios-9-amperios-energivm-mvh1290f2.html?gclid=CjwKCAjwhOyJBhA4EiwAEcJdcbUNyvtoPRdAtMYdwg7F0NFw1dHL-a0-ncPLn0jcTXJU_ugNqywKJxoCPC0QAvD_BwE (accessed Sep. 11, 2021).

[Convertidor DC-DC antiguo, 2021] "RGEEK fuente de alimentación para ordenador, fuente de alimentación de 12V y 150W, DC, ATX, Pico, ATX, DC DC, PSU de 24 Pines, MINI ITX, DC a ATX|atx psu|atx pc poweratx pc power supply - AliExpress." https://es.aliexpress.com/i/32748520366.html (accessed Sep. 11, 2021).

[Router antiguo, 2021] "Buffalo WHR-HP-G54-DD - Router (100 Mbit/s, 125 Mbit/s, WPA2, Negro, Plata, 3,4W) : Amazon.es: Electrónica." https://www.amazon.es/Buffalo-WHR-HP-G54-DD-Router-Negro-Plata/dp/B000ZKCP1W (accessed Sep. 11, 2021).

[MiniPc, 2021] "Mini PC Intel® NUC 8 Mainstream-G (NUC8i5INH)." https://www.intel.es/content/www/es/es/products/sku/189233/intel-nuc-8-mainstreamg-mini-pc-nuc8i5inh/specifications.html (accessed Sep. 11, 2021).

[Cable Serie, 2021] "US232R-100-BULK FTDI - Módulo: cableado integrado | RS232,USB; D-Sub 9pin,USB A; 1m; US232R-100-BLK | TME - Elektroniikka komponentit." https://www.tme.eu/es/details/us232r-100-blk/modulos-usb/ftdi/us232r-100-bulk/?brutto=1&currency=EUR&gclid=CjwKCAjwhOyJBhA4EiwAEcJdcSQaMYstN88tnkAltCQ5Y3Yb7Nn-Vki32S73zh8pbX-HXG6iaIfhRBoCQfwQAvD_BwE (accessed Sep. 11, 2021).

[Convertidor DC-DC Nuevo, 2021] "Convertidor de aumento de 15A, fuente de alimentación de corriente constante, controlador LED de 400 50V a 10 60V, módulo de aumento de cargador de voltaje de DC 8,5 W|Accesorios y piezas de reemplazo| - AliExpress." https://es.aliexpress.com/item/33034601692.html?src=google&albch=shopping&acnt=439-079-4345&slnk=&plac=&mtctp=&albbt=Google_7_shopping&gclsrc=aw.ds&albagn=888888&ds_e_adid=438858099979&ds_e_matchtype=&ds_e_device=c&ds_e_network=u&ds_e_product_group_id=1212517463975&ds_e_product_id=es33034601692&ds_e_product_merchant_id=109204739&ds_e_product_country=ES&ds_e_product_language=es&ds_e_product_channel=online&ds_e_product_store_id=&ds_url_v=2&ds_dest_url=https%3A%2F%2Fs.click.aliexpress.com%2Fdeep_link.htm%3Faff_short_key%3DUneMJZVf&albcp=10191226958&albag=102259630456&isSmbAutoCall=false&needSmbHouyi=false&gclid=Cj0KCQjwpf2IBhDkARIsAGVo0D2KCJ_RQKsCUyXoEahHz8z76kLZIT8vaqa8PfwQNtRtrEBYaFg9-d8aAhYzEALw_wcB&aff_fcid=153c986c89cc479795c155f951469345-1629503137059-06010-UneMJZVf&aff_fsk=UneMJZVf&aff_platform=aaf&sk=UneMJZVf&aff_trace_key=153c986c89cc479795c155f951469345-1629503137059-06010-UneMJZVf&terminal_id=5c24aa42f12b4eecb439fdef177636c5 (accessed Sep. 11, 2021).

[Router Nuevo, 2021] "Archer C6 | Router Gigabit Inalámbrico MU-MIMO de Banda Dual AC1200 | TP-Link Iberia." https://www.tp-link.com/es/home-networking/wifi-router/archer-c6/ (accessed Sep. 11, 2021).

[Camara Kinect, 2021] "Microsoft - Kinect Sensor + Juego Kinect Adventures [Reedición] (Xbox 360) : Amazon.es: Videojuegos." https://www.amazon.es/Microsoft-Kinect-Sensor-Adventures-Reedición/dp/B009SJAIP6/ref=sr_1_1?dchild=1&keywords=kinect+xbox+360&qid=1631315672&sr=8-1 (accessed Sep. 11, 2021).

[Adaptador Kinect para PC] "Generic - Adaptador con enchufe europeo para sensor Kinect de Microsoft Xbox 360 (adaptador de CA, conector USB) : Amazon.es: Videojuegos."

https://www.amazon.es/dp/B008OAVS3Q/ref=sspa_dk_hqp_detail_aax_0?psc=1&spLa=ZW5jcnlwdGVkUXVhbGlmaWVyPUFCWE9CN05UQzE4TU4mZW5jcnlwdGVkSWQ9QTA0MjMzNTIzM09STUpKMUU0N0ZSJmVuY3J5cHRlZEFkSWQ9QTA5NjYxNTRRTU0xWks1MTdETFQmd2lkZ2V0TmFtZT1zcF9ocXBfc2hhcmVkJmFjdGlvbj1jbGlja1JlZGlyZWN0JmRvTm90TG9nQ2xpY2s9dHJ1ZQ== (accessed Sep. 11, 2021).

[Conexión VPN site-to-site, 2021] "Cómo conectarse al entorno de Escritorio Remoto por VPN site-to-site." https://jotelulu.com/soporte/tutoriales/conectar-entorno-escritorio-remoto-por-vpn-site-to-site (accessed Sep. 11, 2021).

[Robot P3-DX, 2021] "P i o n e e r 3-D X," Accessed: Sep. 15, 2021. [Online]. Available: www.mobilerobots.com.

[Instalar SSH en UBUNTU, 2021] "How to Enable SSH on Ubuntu 20.04 | Linuxize." https://linuxize.com/post/how-to-enable-ssh-on-ubuntu-20-04/ (accessed Sep. 15, 2021).

[Información sobre Bibliotecas DLL de Windows, 2021] "Anexo:Bibliotecas DLL de Windows - Wikipedia, la enciclopedia libre." https://es.wikipedia.org/wiki/Anexo:Bibliotecas_DLL_de_Windows#Win32_API_(o_WinAPI32) (accessed Sep. 15, 2021).

[Información sobre Winsocks, 2021] "Winsock - Wikipedia, la enciclopedia libre." https://es.wikipedia.org/wiki/Winsock (accessed Sep. 15, 2021).

[Información sobre Sockets de Internet, 2021] "Socket de Internet - Wikipedia, la enciclopedia libre." https://es.wikipedia.org/wiki/Socket_de_Internet (accessed Sep. 15, 2021).

[Información sobre Secure Shell (SSH), 2021] "Secure Shell - Wikipedia, la enciclopedia libre." https://es.wikipedia.org/wiki/Secure_Shell (accessed Sep. 15, 2021).

III. ANEXOS

III.1. ANEXO I: Memoria Proyecto EASE UAH

Universidad de Alcalá

VICERRECTORADO DE ESTRATEGIA Y PLANIFICACIÓN
Colegio de San Ildefonso
Plaza de San Diego s/n
28801 Alcalá de Henares (Madrid)

ANEXO II: Memoria Complementaria

TÍTULO DEL PROYECTO

Metodología y herramientas de apoyo a la docencia en Electrónica EASE-I

LÍNEAS DE ACTUACIÓN

Línea 5 (Anexo I).- Herramientas para la mejora de la calidad de la docencia

Diseño de laboratorios remotos que contribuyan a una mejora de la calidad de la docencia y de la experiencia educativa de los estudiantes. Se propone reorientar tecnologías, recursos y herramientas tradicionalmente aplicadas de forma presencial para que el estudiante pueda desarrollar las mismas competencias de forma remota, personalizando y adaptando su formación a las condiciones espacio-temporales disponibles.

La metodología, planificación y supervisión propuestas, además de facilitar el acceso de los estudiantes a los recursos materiales limitados de un laboratorio de diseño de sistemas electrónicos digitales y de control electrónico, ha de incentivar la motivación por la comprensión y aplicación de conceptos teórico-prácticos previamente adquiridos.

Para la supervisión, discusión y evaluación de los resultados se contará con la herramienta BlackBoard Collaborate que facilita la interacción online síncrona profesor-alumno.

A tenor de esta experiencia, se evaluará la viabilidad de generación de Suplementos Especiales al Título, con propuesta de prácticas más avanzadas orientadas al autoaprendizaje y al desarrollo de ideas propias de los estudiantes.

INTRODUCCIÓN/JUSTIFICACIÓN DEL PROYECTO

El promover una docencia universitaria de calidad está entre las prioridades de los responsables educativos, desde organismos transnacionales (Objetivo 4° de la Agenda 2030) a locales (Vicerrectorado de Planificación y Estrategia de la UAH). El proyecto de innovación docente EASE-I se enmarca en el campo de la ingeniería y concretamente en el área de tecnología electrónica.

La situación de excepcionalidad sanitaria reciente ha venido a acelerar procesos de enseñanza-aprendizaje que, a un ritmo más lento, veníamos promoviendo los distintos profesores participantes en este proyecto de innovación docente. En esta línea, EASE-I plantea soluciones a un problema transversal en la formación de un graduado y máster en ingeniería: ¿cómo mejorar la adquisición de las competencias prácticas previstas, disponiendo de los mismos recursos instrumentales necesarios pero, aumentando la disponibilidad de la instrumentación y prototipos de laboratorio de forma remota?

Antecedentes:

En el ámbito europeo existen universidades con laboratorios remotos aplicados a temas de control electrónico.

Un ejemplo es el de la universidad suiza EPFL, donde se dispone de 22 estaciones para experimentos online, 10 servo drivers, 10 sistemas de control de temperatura y un sistema para el control de un péndulo invertido.

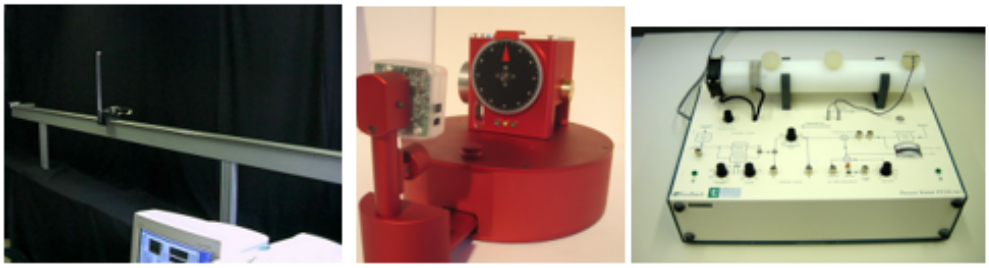


Figura 1. Equipamiento para enseñanza remota de sistemas de control. Universidad EPFL.

También en el Instituto Tecnológico de Karlsruhe han desarrollado el "KUKA Robot Learning Lab", un laboratorio de robótica accesible a distancia que permite a los estudiantes acceder a los robots del laboratorio de forma remota y ejecutar sus propios proyectos en los robots. Los entornos virtuales de simulación robóticos no reflejan fielmente el comportamiento del mundo real, especialmente cuando se manipulan objetos o se emulan sensores. Para los estudiantes, poder manejar el laboratorio más horas es una forma única de aprender los fundamentos de la robótica y tener más experiencia con el hardware robótico. En la parte izquierda de la figura 2 se muestra el laboratorio con diez robots desplegados en serie, cada uno equipado con un robot manipulador con capacidades de detección de fuerza/par y visión artificial - sensor 3D y una cámara web para la visualización remota del experimento. Además del video de la cámara web, los alumnos también obtienen registros y datos recogidos durante los experimentos. El objetivo es proporcionar la misma experiencia de desarrollo y acceso a los datos como si los usuarios se sentaran justo al lado de los robots. El laboratorio interviene en la impartición de cursos masivos abiertos en línea (MOOC) con cientos de participantes. En la parte derecha de la figura 2 se muestra el uso en la plataforma MOOC Udacity.

Figura 2. Herramientas para enseñanza remota de sistemas de robótica. Instituto Tecnológico de Karlsruhe.

Propuesta general

La propuesta se centra en una metodología de enseñanza de materias con alta componente práctica, habilitando herramientas que permita la ejecución remota de las mismas, permitiendo un incremento en la disponibilidad de los limitados recursos del laboratorio.

Para ello se propone la integración de herramientas que permitan la ejecución y supervisión remota de ensayos interdisciplinares, así como la evaluación y realimentación necesaria profesor-estudiante. En definitiva, se persigue una finalidad ambiciosa que difícilmente puede encajar en el marco de un año. Por esta razón, en la figura 3 se esboza la propuesta global (3 años), acorde al plan de trabajo previsto para el Grupo de Innovación Docente de Excelencia, concretando la actividad a realizar en el primer año.

Las disciplinas de Diseño de Sistemas Electrónicos Digitales y Control Electrónico, con una importante componente práctica, son comunes a varias titulaciones de ingeniería, como los casos implantados en la Escuela Politécnica Superior que se citan a continuación y en cuya docencia participan los miembros del equipo del proyecto:

- Grado en Ingeniería Electrónica y Automática Industrial
- Grado en Ingeniería Electrónica de Comunicaciones
- Máter Universitario en Ingeniería Electrónica
- Máster Universitario en Ingeniería Industrial
- Programa (grado y máster) en Ingeniería de Telecomunicación

Tanto la propuesta metodológica como el amplio abanico de titulaciones y asignaturas tendrá un importante impacto en la formación de ingenieros (grado y máster), justifican la propuesta del proyecto EASE-I dentro del marco general del grupo de innovación docente de Excelencia EASE.

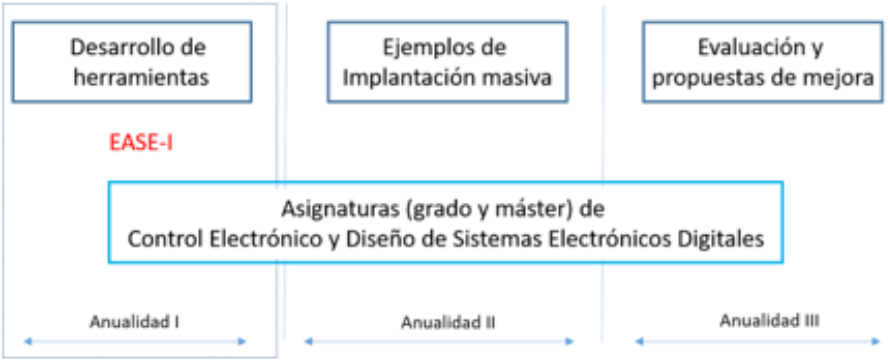


Figura 3. Propuesta global EASE y específica para la primera anualidad EASE-I

OBJETIVOS

Objetivo general:
Diseño de herramientas para realización de prácticas de laboratorio de forma remota, contribuyendo a la mejora de la calidad docente de estudiantes de ingeniería, en grado y máster.

Objetivos particulares:

- Promover la Educación de Calidad, objetivo cuarto de la Agenda 2030 para el desarrollo sostenible de Naciones Unidas.

- Impulsar la línea prioritaria V de Innovación Docente de la UAH: Herramientas para la mejora de la calidad de la docencia (Anexo I de la convocatoria).

- Desarrollar acciones de innovación y renovación de metodologías educativas facilitando la realización remota de prácticas sobre prototipos reales de laboratorio integrando conocimientos de varias disciplinas.

- Contribuir a una mejora en la calidad de la docencia en ingeniería en la UAH, de forma que se aumenten las horas de disponibilidad de la instrumentación y prototipos de ensayo para los estudiantes de forma remota, para la mejora de la adquisición de las competencias previstas en su titulación.

- Facilitar el desarrollo de las titulaciones de grado y máster en ingeniería en la EPS, si bien la metodología propuesta es transversal, el proyecto EASE-I se centra en el Grado en Ingeniería Electrónica y Automática Industrial (GIEAI) y en el Máster Universitario en Ingeniería Electrónica (MUIE).

- Detallar su evaluación práctica en una asignatura en particular, describiendo el caso de uso, y planteando aquí ya algunos de los problemas a los que se quiere dar solución.

ACCIONES A DESARROLLAR

Teniendo en cuenta el planteamiento global EASE indicado en la figura 3, en este proyecto de innovación docente se proponen las siguientes acciones.

A.1. Identificación de prácticas de laboratorio de asignaturas de Control Electrónico y Diseño de Sistemas Electrónicos Digitales susceptibles de ejecución remota.
Como punto de partida se elegirá una práctica de asignaturas de grado o de máster de cada temática (Diseño de Sistemas Electrónicos Digitales y Control Electrónico), 2 en total.

A.2. Dotación e integración de herramientas necesarias para ejecución remota de ensayos.
Elección de componentes electrónicos (hardware y software) y de recursos telemáticos necesarios para el acceso a los puestos de ensayo. Planificación de franjas temporales en los que cada alumno pueda llevar a cabo la ejecución remota de la práctica y franjas dedicadas al mantenimiento del laboratorio.

A.3. Recursos de supervisión de ensayo remoto
Elección, ubicación y configuración de cámaras que permitan al estudiante y al profesor tener una realimentación de la correcta ejecución de la prueba experimental. El estudiante se conectará de forma remota a uno de los PCs del laboratorio conectados en red con el/losprototipo/s de ensayo. El profesor configurará las sesiones de ensayo para que en estos PCs se registre la información asociada a la práctica: identificando del usuario, programas ejecutados y resultados obtenidos.

A.4. Integración de herramientas para la interacción profesor-estudiante.
Integración de las herramientas anteriores con otras disponibles en el aula virtual de la UAH, como Collaborate, que garanticen la interacción estudiante-profesor tanto para la resolución de eventuales problemas durante la ejecución como para la evaluación de los resultados obtenidos y propuestas de mejora.

A.5. Casos de uso particulares
Como paso previo a la implantación masiva y evaluación de una muestra suficientemente amplia, que permita extraer conclusiones sobre los pros y contras de la estrategia metodológica docente implícita en EASE, se pondrá en práctica un caso de uso de las diferentes asignaturas indicadas en la acción A.1.

A modo de ejemplo se describe un caso práctico.

En la asignatura Diseño de Sistemas de Control Electrónico, obligatoria en el Máster Universitario en Ingeniería Electrónica, los fundamentos teóricos se aplican sobre un robot P3-

DX: identificación de planta, diseño de controladores y ejecución de seguimiento de trayectorias no lineales (figura 4).

El proyecto EASE-I permitirá el diseño de prácticas (al menos 2/3 del total) que puedan realizarse con los robots ubicados en plataformas, de modo que se puedan ejecutar aplicaciones de forma remota cuyo resultado sea el giro de las ruedas motrices sin que se desplace el robot. Tareas como identificación y modelado del robot, ensayo de algoritmos de control y colaboración entre robots puedes probarse con los robots ubicados en dichas plataformas, tal y como se muestra en la parte izquierda de la figura 4, pero con el robot sin contacto con el suelo. De esta forma, se limitan las horas de presencialidad destinadas a la experimentación con desplazamiento de los robots (parte derecha de la figura 4) aprovechando la realimentación de los sensores de posición ubicados en el pasillo de los laboratorios OL1-OL6 de la EPS.

Figura 4. Prácticas presenciales con robots P3-DX. Asignatura de Diseño de Control Electrónico. Máster Universitario en Ingeniería Electrónica.

DIFUSIÓN

En relación a la difusión de los resultados y trabajo realizado en el proyecto de innovación docente, se plantean las siguientes:

Presentación de los resultados en el Encuentro de Innovación Docente UAH EIDU que se celebra anualmente.

Jornadas de difusión (EPS y UAH) para atracción de nuevos estudiantes.

Participación en congreso nacional o publicación en español de las acciones y metodologías educativas planteadas en el proyecto.

Publicación en revistas relevantes de ámbito internacional (JCR) en temas de innovación educativa (IEEE Trans. on Education, Educational Research, Education and Training, etc).

DESCRIPCIÓN/JUSTIFICACIÓN DEL EQUIPO DEL PROYECTO

El equipo del proyecto EASE-I está formado por los profesores del UAH-GI20-154, con mención de excelencia. Por una parte C. Mataix, JL. Lázaro, I. Bravo, A. Gardel, con amplia experiencia en la docencia de asignaturas de Diseño de Sistemas Electrónicos Digitales; y por otra D. Pizarro y F. Espinosa, con amplia experiencia en la docencia de asignaturas de Control Electrónico.

Además, se contará con la colaboración de Carlos Santos, investigador del grupo Geiser (UAH) y que desarrollo parte de las pruebas experimentales de su tesis doctoral (Control self-triggered para seguimiento de trayectorias no lineales de robots con adaptación al retardo del canal) con los robots P3-DX indicados.

III.2. ANEXO II: Driver PC I, Código Fuente y Cabecera

CODIGO FUENTE

```
#define S_FUNCTION_NAME  robot_dr
#define S_FUNCTION_LEVEL 2
#define _BSD_SOURCE

#include "simstruc.h"
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>
#include <signal.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <stddef.h>
#include <string.h>
#include <strings.h>
#include <math.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <termios.h>
#include <sys/socket.h>
#include <netinet/tcp.h>
#include <netdb.h>
#include <limits.h>
#include <pthread.h>
#include <stdint.h>
#include <errno.h>
#include <semaphore.h>

#include "robot_dr.h"

#if defined(RT)
#warning  DEFINED !!!
#elif MATLAB_MEX_FILE
#warning MATLAB_MEX_FILE DEFINED !!!
#endif

//VARIABLES SELF-TRIGGERED

//double tiempo_actualizacion=5;
//double tiempo_ultima_actualizacion=0;
//double actualiza=0;
float v_lin_local;
float v_rot_local;
float recibido=0;
float v_consigna=0;
float o_consigna=0;

// Robot data
typedef struct
{
  //RT_TASK *task;

  pthread_t thread; //int thread;
  sem_t *sem;       //SEM *sem;
}p3dx_data_t;

p3dx_data_t p3dx_data;

 //Protocolo de parada
 int parada=0;

//COMUNICACIONES
 int envio=1;
 int numero_envio=0;
 int recibe= 0;
 int errorbind=0;
 pthread_t receive;
 void * receive_function(void * ptr);
 server_t rc_server;

// // LASER
// pthread_t laser;
// int pid_laser;


// SERIAL PORT (added CLG)
struct termios tty;
int serial_port;

// //FUNCTIONS
// void * server_thread_laser(void * ptr);


//////////////////////////////////////////////////////////////

/* Function: mdlInitializeSizes
================================================
 * Abstract:
 *   Setup sizes of the various vectors.
 */
static void mdlInitializeSizes(SimStruct *S)
{
   ssSetNumSFcnParams(S, 1);
   if (ssGetNumSFcnParams(S) !=
ssGetSFcnParamsCount(S)) {
      return; /* Parameter mismatch will be reported
by Simulink */
   }

   if (!ssSetNumInputPorts(S, 0)) return;


   if (!ssSetNumOutputPorts(S,2)) return;

   ssSetOutputPortWidth(S, 0,
DYNAMICALLY_SIZED); //Linear velocity
odometry
   ssSetOutputPortWidth(S, 1,
DYNAMICALLY_SIZED); //Angular velocity
odometry


   ssSetNumSampleTimes(S, 1);

   /* Take care when specifying exception free code -
see sfuntmpl_doc.c */

   // SS_OPTION_PLACE_ASAP: this is typically
used by devices connecting to hardware.
  ssSetOptions(S, SS_OPTION_PLACE_ASAP);

}


/* Function: mdlInitializeSampleTimes
================================================
 * Abstract:
 *   Specifiy that we inherit our sample time from
the driving block.
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{

   ssSetSampleTime(S, 0, Ts);
```

```
    ssSetOffsetTime(S, 0, 0.0);

}

#define MDL_START  /* Change to #undef to
remove function */
#if defined(MDL_START)
  /* Function: mdlStart
=====================================
==============
   * Abstract:
   *    This function is called once at start of model
execution. If you
   *    have states that should be initialized once, this
is the place
   *    to do it.
   */
  static void mdlStart(SimStruct *S)
  {
#if defined(RT)
    int temp;
    int i, cove;
    char DIR_UDP_COVE[20];
    int PORT_UDP_COVE;
    char robot[10];

    // read parameter with robot name
    mxGetString((ssGetSFcnParam(S, 0)),robot,6);
    cove = atoi(&robot[4]);
    printf("cove %d \n", cove);

    switch(cove)
    {
       case 0:
          PORT_UDP_COVE = PORT_UDP_COVE0;
          strcpy(DIR_UDP_COVE,
DIR_UDP_COVE0);
          break;
       case 1:
          PORT_UDP_COVE = PORT_UDP_COVE1;
          strcpy(DIR_UDP_COVE,
DIR_UDP_COVE1);
          break;
       case 2:
          PORT_UDP_COVE = PORT_UDP_COVE2;
          strcpy(DIR_UDP_COVE,
DIR_UDP_COVE2);
          break;
       case 3:
          PORT_UDP_COVE = PORT_UDP_COVE3;
          strcpy(DIR_UDP_COVE,
DIR_UDP_COVE3);
          break;
       case 4:
          PORT_UDP_COVE = PORT_UDP_COVE4;
          strcpy(DIR_UDP_COVE,
DIR_UDP_COVE4);
          break;
       case 5:
          PORT_UDP_COVE = PORT_UDP_COVE5;
          strcpy(DIR_UDP_COVE,
DIR_UDP_COVE5);
          break;
       default: // cove9 para pruebas en red local
          PORT_UDP_COVE = PORT_UDP_COVE9;
          strcpy(DIR_UDP_COVE,
DIR_UDP_COVE9);
          break;
    }
    printf("\nDir.IP: %s, PORT: %d\n",
DIR_UDP_COVE, PORT_UDP_COVE );

   /* create binary semaphore with semaphore
available and queue tasks on FIFO basis */
     //p3dx_data.sem =
rt_typed_sem_init(nam2num("SEM_P2OS"), 1,
BIN_SEM | FIFO_Q );

    p3dx_data.sem = malloc(sizeof(sem_t));
    sem_init(p3dx_data.sem,0,1);

   // Control when serial port is opened
   temp = open("/dev/ttyUSB0", O_RDWR);
   //temp = rt_spopen(RTCOM1, 115200, 8, 1,
RT_SP_PARITY_NONE,
RT_SP_NO_HAND_SHAKE,
RT_SP_FIFO_SIZE_4);

   if (temp < 0) {
            printf("RT Serial 1 open error
(%d)\n",temp);
   }
   else{
       printf("RT Serial 1 opened\n");
       serial_port = temp;
   }

   tty.c_cflag &= ~PARENB; // Clear parity bit,
disabling parity (most common)
   tty.c_cflag &= ~CSTOPB; // Clear stop field, only
one stop bit used in communication (most common)
   tty.c_cflag |= CS8; // 8 bits per byte (most
common)
   //tty.c_cflag &= ~CRTSCTS; // Disable RTS/CTS
hardware flow control (most common)
   tty.c_cc[VTIME] = 2;// wait for up to 200 ms (2
deciseconds)
   tty.c_cc[VMIN] = 0;
   cfsetispeed(&tty, B115200);
   if (tcsetattr(serial_port, TCSANOW, &tty) != 0) {
     printf("Error %i from tcsetattr: %s\n", errno,
strerror(errno));
   }

#ifdef DEBUG_IO
   // Control when serial port is opened
   temp = open("/dev/ttyUSB0", O_RDWR);
   //temp = rt_spopen(RTCOM2, 115200, 8, 1,
RT_SP_PARITY_NONE,
RT_SP_NO_HAND_SHAKE,
RT_SP_FIFO_SIZE_4);

   if (temp < 0) {
            printf("RT Serial 2 open error
(%d)\n",temp);
   }
   else{
       printf("RT Serial 2 opened\n");
       serial_port = temp;
   }
#endif
    // SOCKET SEVERS

    //REMOTE CENTER SERVER
    bzero((char
*)&rc_server.dir_serv,sizeof(rc_server.dir_serv));
    rc_server.dir_serv.sin_family=AF_INET;

rc_server.dir_serv.sin_addr.s_addr=inet_addr(DIR_
UDP_COVE);

rc_server.dir_serv.sin_port=htons(PORT_UDP_COV
E0);
```

```
if((rc_server.sockfd=socket(AF_INET,SOCK_DGRAM,0))<0)
                printf("cliente:no puedo abrir socket");

   if(bind(rc_server.sockfd,(struct sockaddr *)&rc_server.dir_serv,sizeof(rc_server.dir_serv))<0)
   {
      printf("server rc: no se puede asociar la dir local\n");
      errorbind=1;
   }
   else
   {
      printf("socket abierto\n");
   }
   // Create a linux thread
   //p3dx_data.thread = rt_thread_create(P2OS_Main, NULL, 10000);
   pthread_create(&(p3dx_data.thread), NULL, P2OS_Main, NULL);

   // Bloqueo en espera del centro remoto

recvfrom(rc_server.sockfd,(char*)&rc_server.receive_data,sizeof(comm_data_t),0,(struct sockaddr *)&rc_server.dir_cli,&rc_server.long_dir_cli);  // NO_RT!

//    receive = rt_thread_create(receive_function, NULL, 10000);
   pthread_create(&receive, NULL, receive_function,NULL);


#endif

  }
#endif /*  MDL_START */


static void mdlOutputs(SimStruct *S, int_T tid)
{


  real_T *y0    = ssGetOutputPortRealSignal(S,0);
           /* Velocidad Lineal */
  real_T *y1    = ssGetOutputPortRealSignal(S,1);
           /* Velocidad Angular */


  #if defined(RT)

  double sonar_enable;
  int i=0;
  int n=0;


      sem_wait(p3dx_data.sem);
      //rt_sem_wait(p3dx_data.sem); // LOCK
      // 2nd: read values from robot
      v_lin_local = (real_T)p2os_data_read.vel.vlin; // m/s
      v_rot_local = (real_T)p2os_data_read.vel.vrot; // rad/s

           sem_post(p3dx_data.sem);
  //     rt_sem_signal(p3dx_data.sem); // UNLOCK


  if(recibe==1) //ACTUALIZACIÓN
  {

      //printf(" Recepcion Numero %d Velocidad lineal %f velocidad angular %f \n",rc_server.receive_data.data_number,rc_server.receive_data.lineal,rc_server.receive_data.angular);
      v_consigna=rc_server.receive_data.lineal;
      o_consigna=rc_server.receive_data.angular;


      if(v_consigna > V_SAT)
         v_consigna = V_SAT;
      else if(v_consigna < -V_SAT)
         v_consigna = -V_SAT;

      if(o_consigna > O_SAT)
         o_consigna = O_SAT;
      else if(o_consigna < -O_SAT)
         o_consigna = -O_SAT;

      if(sonar_enable > 0.0)
         sonar_enable = 1.0;
      else
         sonar_enable = 0.0;



      numero_envio=numero_envio+1;
      rc_server.send_data.lineal=v_lin_local;
      rc_server.send_data.angular=v_rot_local;

rc_server.send_data.data_number=numero_envio;
      n=sendto(rc_server.sockfd,(char *)&rc_server.send_data,sizeof(comm_data_t),0,(struct sockaddr *)&rc_server.dir_cli,sizeof(rc_server.dir_cli));
      if(n<0)
        perror("dg_cli:error en el sendto\n");
      recibe=0;
      printf(" Envio Numero %d Velocidad lineal %f velocidad angular %f \n",rc_server.send_data.data_number,rc_server.send_data.lineal,rc_server.send_data.angular);
   }
        // 1st: write commands
//      rt_sem_wait(p3dx_data.sem); // LOCK
      sem_wait(p3dx_data.sem);

      p2os_cmd_write.lin_speed = v_consigna;  // m/s
      p2os_cmd_write.rot_speed = o_consigna;  // rad/s
      p2os_cmd_write.sonar_enable = sonar_enable; // 1 / 0

     // rt_sem_signal(p3dx_data.sem); // UNLOCK
     sem_post(p3dx_data.sem);


      *y0 = (real_T)v_lin_local;
      *y1 = (real_T)v_rot_local;


      #else // NOT RT MODE

      // Write in output port of S-Function
      *y0 = (real_T)0.0;
      *y1 = (real_T)0.0;
      #endif


}
```

```
/* Function: mdlTerminate
=====================================
=========
 * Abstract:
 *    No termination needed, but we are required to
have this routine.
 */
static void mdlTerminate(SimStruct *S)
{
#if defined(RT)
  int i;

  //P2OS_Terminate(); // Send signal to P2OS Driver

  // clean task

  //rt_task_delete(p3dx_data.task);

  P2OS_MainQuit();

/*  rt_spclose(RTCOM1);*/
  close(serial_port);

// rt_sem_delete(p3dx_data.sem);
 sem_destroy(p3dx_data.sem);


#endif

  return;
}


void * receive_function(void * ptr)
{

    int n=0;
    while(1)
    {

        n =
recvfrom(rc_server.sockfd,(char*)&rc_server.receive
_data,sizeof(comm_data_t),0,(struct sockaddr
*)&rc_server.dir_cli,&rc_server.long_dir_cli);  //
NO_RT!
        recibe=1;
        // LOCK

        memcpy(&rc_server.storage_data[3],
&rc_server.storage_data[2], sizeof(comm_data_t));
        memcpy(&rc_server.storage_data[2],
&rc_server.storage_data[1], sizeof(comm_data_t));
        memcpy(&rc_server.storage_data[1],
&rc_server.storage_data[0], sizeof(comm_data_t));
        memcpy(&rc_server.storage_data[0],
&rc_server.receive_data, sizeof(comm_data_t));

        // UNLOCK

        if(n<0)
           printf("dg_cli: error en el recvfrom
PREDECESOR");   // NO_RT!

        printf("DATO RECIBIDO Numero %d
Velocidad lineal %f velocidad angular %f
\n",rc_server.receive_data.data_number,rc_server.r
eceive_data.lineal,rc_server.receive_data.angular);

    }

}
```

```
////////////////////////////////////////////////////

////////////////////////////////////////////////////
////////////////////////////////////////////////////
// Private Defines

#define PLAYER_NUM_ROBOT_TYPES 1
#define PACKET_LEN 256

/* Command numbers */
#define SYNC0 0
#define SYNC1 1
#define SYNC2 2

#define PULSE 0
#define OPEN 1
#define CLOSE 2
#define ENABLE 4
#define SETA 5
#define SETV 6
#define SETO 7
#define VEL 11
#define RVEL 21
#define SETRA 23
#define SONAR 28


#define STOP 29
#define VEL2 32


#define BUMP_STALL 44
#define JOYDRIVE 47
#define ROTKP 82
#define ROTKV 83

#define ROTKI 84
#define TRANSKP 85
#define TRANSKV 86
#define TRANSKI 87

#define P2OS_CYCLETIME_USEC 20000
#define P2OS_CYCLETIME_NSEC
(P2OS_CYCLETIME_USEC*1000)


#define PLAYER_POWER_MASK_VOLTS 1
#define PLAYER_POWER_MASK_WATTS 2
#define PLAYER_POWER_MASK_JOULES 4
#define PLAYER_POWER_MASK_PERCENT 8
#define P2OS_NOMINAL_VOLTAGE 12.0

#define MOTOR_MAX_LIN_SPEED 1000 // mm/s
#define MOTOR_MAX_ROT_SPEED 180 // deg/s

/* Argument types */
#define ARGINT   0x3B // Positive int (LSB, MSB)
#define ARGNINT  0x1B // Negative int (LSB, MSB)

#define PLAYER_ERR_ERR 0
#define PLAYER_ERR_WARN 1
#define PLAYER_ERR_MSG 2

#define PLAYER_WARN(msg)
ErrorPrint(PLAYER_ERR_WARN, 0, __FILE__,
__LINE__, "warning : " msg "\n")
#define PLAYER_WARN1(msg, a)
ErrorPrint(PLAYER_ERR_WARN, 0, __FILE__,
__LINE__, "warning : " msg "\n", a)
#define PLAYER_WARN2(msg, a, b)
ErrorPrint(PLAYER_ERR_WARN, 0, __FILE__,
__LINE__, "warning : " msg "\n", a, b)
```

```
// Convert degrees to radians
#ifndef DTOR
#define DTOR(d) ((d) * (M_PI) / 180.0)
#endif

// Convert radians to degrees
#ifndef RTOD
#define RTOD(r) (((double)(r) * 180.0) / M_PI)
#endif


////////////////////////////////////////////////////////////////
// Private Types

typedef struct
{
  unsigned char packet[PACKET_LEN];
  unsigned char size;
  //double timestamp;
} P2OSPacket_t;

typedef struct
{
  int AdjustNumSamplesFlag; //
  int AlignAngle; //
  int AlignSpeed; //
  double AngleConvFactor; //
  int AngleIncrement; //
  int CenterAwayCost; //
  const char* Class;
  int ClearOnFail; //
  int CollisionRange; //
  double DiffConvFactor; //
  double DiscardThreshold; //
  double DistConvFactor; //
  double DistanceWt; //
  int DrivingRotAccel; //
  int DrivingRotDecel; //
  int DrivingRotVelMax; //
  int DrivingTransAccel; //
  int DrivingTransDecel; //
  int DrivingTransNegVelMax; //
  int DrivingTransVelMax; //
  int EmergencyMaxTransDecel; //
  int FailedTh; //
  int FailedX; //
  int FailedY; //
  int FastSpeed; //
  int FrontBumpers; //
  int FrontClearance; //
  int FrontPaddingAtFastSpeed; //
  int FrontPaddingAtSlowSpeed; //
  int FuseAllSensors; //
  int GoalAngleTol; //
  int GoalDistanceTol; //
  int GoalOccupiedFailDistance; //
  int GoalRotAccel; //
  int GoalRotDecel; //
  int GoalRotSpeed; //
  int GoalSpeed; //
  double GoalSwitchTime; //
  int GoalTransAccel; //
  int GoalTransDecel; //
  int GoalUseEncoder; //
  int GridRes; //
  double GyroScaler; //
  int HasMoveCommand; //
  int HeadingRotAccel; //
  int HeadingRotDecel; //
  int HeadingRotSpeed; //
  double HeadingWt; //
  int Holonomic; //
  int IRNum; //
  int IRUnit; //
  int IdleTimeTriggerTh; //
  int IdleTimeTriggerX; //
  int IdleTimeTriggerY; //
  double KDegPerDeg; //
  double KDegPerMm; //
  double KMmPerMm; //
  int LaserFlipped; //
  const char* LaserIgnore;
  const char* LaserPort;
  int LaserPossessed; //
  int LaserPowerControlled; //
  int LaserTh; //
  int LaserX; //
  int LaserY; //
  int LocalPathFailDistance; //
  const char* Map;
  double MarkOldPathFactor; //
  int MaxRVelocity; //
  int MaxRotSpeed; //
  int MaxSpeed; //
  int MaxVelocity; //
  int MinNumSamples; //
  int NewTableSensingIR; //
  int NforLinVelIncrements; //
  int NforRotVelIncrements; //
  int NumFrontBumpers; //
  int NumRearBumpers; //
  int NumSamples; //
  int NumSamplesAngleFactor; //
  int NumSplinePoints; //
  double ObsThreshold; //
  double OccThreshold; //
  int OneWayCost; //
  double PassThreshold; //
  double PeakFactor; //
  int PeakStdTh; //
  int PeakStdX; //
  int PeakStdY; //
  int PeturbTh; //
  int PeturbX; //
  int PeturbY; //
  int PlanEverytime; //
  double PlanFreeSpace; //
  double PlanRes; //
  int Qtt; //
  int Qxx; //
  int Qyy; //
  double RangeConvFactor; //
  int RearBumpers; //
  int RecoverOnFail; //
  int ReflectorMatchDist; //
  int ReflectorMaxAngle; //
  int ReflectorMaxRange; //
  int ReflectorVariance; //
  int RequestEncoderPackets; //
  int RequestIOPackets; //
  int Resistance; //
  int RobotDiagonal; //
  int RobotLength; //
  int RobotLengthFront; //
  int RobotLengthRear; //
  int RobotRadius; //
  int RobotWidth; //
  int RotAccel; //
  int RotDecel; //
  int RotVelMax; //
  int SecsToFail; //
  double SensorBelief; //
  int SettableAccsDecs; //
```

```
  int SettableVelMaxes; //                          0,
  int SideClearanceAtFastSpeed; //                  0,
  int SideClearanceAtSlowSpeed; //                  0,
  int SlowSpeed; //                                 0,
  int SmoothWindow; //                              0,
  int SonarAngleRes; //                             0,
  int SonarAperture; //                             0,
  double SonarBetaMax; //                           0,
  double SonarBetaMin; //                           0,
  int SonarIncidenceLimit; //                       0,
  double SonarLambdaF; //                           0,
  double SonarLambdaR; //                           0,
  int SonarMaxRange; //                             0,
  int SonarMinLineSize; //                          0,
  int SonarNum; //                                  0,
  int SonarRangeRes; //                             0,
  int SonarSigma; //                                0,
  int SplineDegree; //                              0,
  int StallRecoverDuration; //                      0,
  int StallRecoverRotation; //                      0,
  int StallRecoverSpeed; //                         1.626,
  int StdTh; //                                     1,
  int StdX; //                                      0,
  int StdY; //                                      0,
  const char* Subclass;                            0,
  int SuperMaxTransDecel; //                        0,
  int SwitchToBaudRate; //                          1,
  int TableSensingIR; //                            0,
  int TransAccel; //                                0,
  int TransDecel; //                                0,
  int TransVelMax; //                               0,
  int Triangulate; //                               0,
  double TriangulateScoreThreshold; //              0,
  int TriggerAngle; //                              0,
  int TriggerDistance; //                           0,
  int TriggerTime; //                               0,
  int TriggerTimeEnabled; //                        "",
  int UseCollisionRangeForPlanning; //              "COM3",
  int UseEStop; //                                  0,
  int UseLaser; //                                  1,
  int UseReflectorCenters; //                       0,
  int UseSonar; //                                  18,
  int Vel2Divisor; //                               0,
  int VelConvFactor; //                             0,
  double VelocityWt; //                             "0",
} RobotParams_t;                                   0,
                                                   500,
RobotParams_t p3dx_sh_params =                     0,
{                                                  0,
  0,                                               2200,
  0,                                               0,
  0,                                               0,
  0.001534,                                        0,
  0,                                               0,
  0,                                               5,
  "Pioneer",                                       5,
  0,                                               0,
  0,                                               0,
  0.00606, // NOTE it was changed (BUG: Player &   0,
Stage should change it too!);                      0,
  0,                                               0,
  1,                                               0,
  0,                                               0,
  0,                                               0,
  0,                                               0,
  0,                                               0,
  0,                                               0,
  0,                                               0,
  0,                                               0,
  0,                                               0,
  0,                                               0,
  0,                                               0,
  0,                                               0,
```

```
0,
0,
0,
0,
1,
0,
0,
0,
0,
0,
0,
0,
0,
120,
511,
210,
301,
250,
425,
0,
0,
0,
0,
0,
1,
1,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
16,
0,
0,
0,
0,
0,
0,
0,
0,
0,
"p3dx-sh",
0,
38400,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
20,
1,
0
};
```

```
//////////////////////////////////////////////////////////////////
// Private Variables

char ignore_checksum;
int param_idx; // index of our robot's data in the
parameter table

int motor_max_trans_accel = MAX_ACCEL;
int motor_max_trans_decel = -MAX_ACCEL;
int motor_max_rot_accel = MAX_ACCEL;
int motor_max_rot_decel = -MAX_ACCEL;

P2OSPacket_t tx_packet;
P2OSPacket_t rx_packet;

RobotParams_t
PlayerRobotParams[PLAYER_NUM_ROBOT_TYPE
S];

volatile int p2os_run = 1;

p2os_data_t p2os_data_read;
p2os_cmd_t p2os_cmd_write;


//////////////////////////////////////////////////////////////////
// Private Functions

int P2OS_MainSetup(void);
void P2OS_MainProcess(void);
void P2OS_MainQuit(void);

int  P2OSPacket_CalcChkSum(P2OSPacket_t *
pkt);
void P2OSPacket_Print(P2OSPacket_t * pkt);
void P2OSPacket_PrintHex(P2OSPacket_t * pkt);
int  P2OSPacket_Build(P2OSPacket_t * pkt,
unsigned char *data, unsigned char datasize );
int  P2OSPacket_Send(P2OSPacket_t * pkt);
int  P2OSPacket_Receive_timeout(P2OSPacket_t *
pkt, char ignore_checksum , int timeout);
int  P2OSPacket_Check(P2OSPacket_t * pkt, int
ignore_checksum);
int  P2OS_SendReceive(P2OSPacket_t * tx_pkt,
P2OSPacket_t * rx_pkt, char publish_data);

void P2OS_ToggleSonarPower(unsigned char val);

//////////////////////////////////////////////////////////////////

void P2OS_Terminate(void)
{/*
  rt_sem_wait(sem_p2os_exit);
  p2os_run = 0;
  rt_sem_signal(sem_p2os_exit);
*/
}

void * P2OS_Main(void *devicep)
{
   int temp;

   /*p3dx_data.task =
rt_task_init_schmod(nam2num("ARCOS_DRV"), 0,
0, 0, SCHED_FIFO, 0xFF);
   mlockall(MCL_CURRENT | MCL_FUTURE);*/
   // makes task hard real time (default: soft)
   // uncomment the next line when developing :
develop in soft real time mode
```

```
/*#ifdef RT_HARD
   rt_make_hard_real_time();
#endif*/

/*   rt_spclear_rx(RTCOM1);*/
/*   rt_spclear_tx(RTCOM1);*/

  if(P2OS_MainSetup())
    goto exit_task;

  P2OS_MainProcess();

exit_task:



    // makes task soft real time
  //  rt_make_soft_real_time();



    return 0;

//  pthread_exit(NULL);

}

int P2OS_MainSetup()
{
  int i;

  int bauds = DEFAULT_BAUD;
  int numbauds = sizeof bauds / sizeof(int);
  int currbaud = 0;

  struct termios term;
  unsigned char command;
  unsigned char command_s[5];

  int flags=0;
  char sent_close = 0;

  char name[20], type[20], subtype[20];
  int cnt;
  int num_sync_attempts;

  enum
  {
    NO_SYNC,
    AFTER_FIRST_SYNC,
    AFTER_SECOND_SYNC,
    READY
  } psos_state;

  printf("P2OS_MainSetup()\n");

  psos_state = NO_SYNC;

  PlayerRobotParams[0] =  p3dx_sh_params;

  //rt_sem_wait(p3dx_data.sem);
  sem_wait(p3dx_data.sem);
  p2os_cmd_write.lin_speed = 0;
  p2os_cmd_write.rot_speed = 0;
  //rt_sem_signal(p3dx_data.sem);
  sem_post(p3dx_data.sem);

  // Sync:

  num_sync_attempts = 0;

  while(psos_state != READY)
  {

    if(p2os_run==0)
      return 1;

    switch(psos_state)
    {
      case NO_SYNC:

        //printf("P2OS_MainSetup: Before
NO_SYNC\n");
        command = SYNC0;
        P2OSPacket_Build(&tx_packet, &command, 1);
        P2OSPacket_Send(&tx_packet);

        //printf("TX SYNC0\n");

       // usleep(P2OS_CYCLETIME_USEC);
//rt_sleep(nano2count(P2OS_CYCLETIME_NSEC));

        break;
      case AFTER_FIRST_SYNC:

        command = SYNC1;
        P2OSPacket_Build(&tx_packet, &command, 1);
        P2OSPacket_Send(&tx_packet);
        break;
      case AFTER_SECOND_SYNC:
        command = SYNC2;
        P2OSPacket_Build(&tx_packet, &command, 1);
        P2OSPacket_Send(&tx_packet);
        break;
      default:
        break;
    }

    //usleep(P2OS_CYCLETIME_USEC);
//rt_sleep(nano2count(P2OS_CYCLETIME_NSEC));

    if(P2OSPacket_Receive_timeout(&rx_packet,
ignore_checksum, 200E6)) // Wait 200 mS
    {
      if((psos_state == NO_SYNC) &&
(num_sync_attempts < 20))
      {
        num_sync_attempts++;

        continue;
      }
      else
      {
        printf("Robot doesn't work: %d
attemps\n",num_sync_attempts);
        return(1);

      }
    }

    switch(rx_packet.packet[3])
    {
      case SYNC0:
        psos_state = AFTER_FIRST_SYNC;
        break;
      case SYNC1:
        psos_state = AFTER_SECOND_SYNC;
        break;
      case SYNC2:
        psos_state = READY;

        break;
      default:
```

```
      // maybe P2OS is still running from last time.
let's try to CLOSE
      // and reconnect
      if(!sent_close)
      {
        command = CLOSE;
        P2OSPacket_Build(&tx_packet, &command,
1);
        P2OSPacket_Send(&tx_packet);

        sent_close = 1;

//rt_sleep(nano2count(2*P2OS_CYCLETIME_NSEC
));
        usleep(2*P2OS_CYCLETIME_USEC);
      }
      break;
    }

    usleep(P2OS_CYCLETIME_USEC);

//rt_sleep(nano2count(P2OS_CYCLETIME_NSEC));

  } // while(psos_state != READY)

  if(psos_state != READY)
  {
    #ifdef DEBUG_ARCOS
    printf("Couldn't synchronize with P2OS\n");
    #endif
    return(1);
  }

  cnt = 4;
  cnt += snprintf(name, sizeof(name), "%s",
&rx_packet.packet[cnt]);
  cnt++;
  cnt += snprintf(type, sizeof(type), "%s",
&rx_packet.packet[cnt]);
  cnt++;
  cnt += snprintf(subtype, sizeof(subtype), "%s",
&rx_packet.packet[cnt]);
  cnt++;

  command = OPEN;
  P2OSPacket_Build(&tx_packet, &command, 1);
  P2OSPacket_Send(&tx_packet);

//rt_sleep(nano2count(P2OS_CYCLETIME_NSEC));
  usleep(P2OS_CYCLETIME_USEC);

  command = PULSE;
  P2OSPacket_Build(&tx_packet, &command, 1);
  P2OSPacket_Send(&tx_packet);

//rt_sleep(nano2count(P2OS_CYCLETIME_NSEC));
  usleep(P2OS_CYCLETIME_USEC);

  #ifdef DEBUG_ARCOS
  printf("Done.\n   Connected to %s, a %s %s\n",
name, type, subtype);
  #endif

  // now, based on robot type, find the right set of
parameters
  for(i=0;i<PLAYER_NUM_ROBOT_TYPES;i++)
  {
    if(!strcasecmp(PlayerRobotParams[i].Class,type)
&&
!strcasecmp(PlayerRobotParams[i].Subclass,subtype
))
    {
      param_idx = i;
      break;
    }
  }
  if(i == PLAYER_NUM_ROBOT_TYPES)
  {
    #ifdef DEBUG_ARCOS
    fputs("P2OS: Warning: couldn't find parameters
for this robot; "
          "using defaults\n",stderr);
    #endif
    param_idx = 0;
  }
  // first, receive a packet so we know we're
connected.

  // turn off the sonars at first
  P2OS_ToggleSonarPower(0);

  // Set max accel/decel limits
  if(motor_max_trans_accel > 0)
  {
    command_s[0] = SETA;
    command_s[1] = ARGINT;
    command_s[2] = motor_max_trans_accel &
0x00FF;
    command_s[3] = (motor_max_trans_accel &
0xFF00) >> 8;
    P2OSPacket_Build(&tx_packet, command_s, 4);
    P2OS_SendReceive(&tx_packet, &rx_packet,0);
  }

  if(motor_max_trans_decel < 0)
  {
    command_s[0] = SETA;

    command_s[1] = ARGNINT;
    command_s[2] = abs(motor_max_trans_decel) &
0x00FF;
    command_s[3] = (abs(motor_max_trans_decel) &
0xFF00) >> 8;
    P2OSPacket_Build(&tx_packet, command_s, 4);
    P2OS_SendReceive(&tx_packet,&rx_packet,0);
  }
  if(motor_max_rot_accel > 0)
  {
    command_s[0] = SETRA;
    command_s[1] = ARGINT;
    command_s[2] = motor_max_rot_accel & 0x00FF;
    command_s[3] = (motor_max_rot_accel & 0xFF00)
>> 8;
    P2OSPacket_Build(&tx_packet, command_s, 4);
    P2OS_SendReceive(&tx_packet,&rx_packet,0);
  }
  if(motor_max_rot_decel < 0)
  {
    command_s[0] = SETRA;
    command_s[1] = ARGNINT;
    command_s[2] = abs(motor_max_rot_decel) &
0x00FF;
    command_s[3] = (abs(motor_max_rot_decel) &
0xFF00) >> 8;
    P2OSPacket_Build(&tx_packet, command_s, 4);
    P2OS_SendReceive(&tx_packet,&rx_packet,0);
  }

/* FUTURE USE

  // if requested, change PID settings
  P2OSPacket pid_packet;
  unsigned char pid_command[4];
  if(charrot_kp >= 0)
```

```
  {
    pid_command[0] = ROTKP;
    pid_command[1] = ARGINT;
    pid_command[2] = charrot_kp & 0x00FF;
    pid_command[3] = (charrot_kp & 0xFF00) >> 8;
    pid_packet.Build(pid_command, 4);
    charSendReceive(&pid_packet);
  }
  if(charrot_kv >= 0)
  {
    pid_command[0] = ROTKV;
    pid_command[1] = ARGINT;
    pid_command[2] = charrot_kv & 0x00FF;
    pid_command[3] = (charrot_kv & 0xFF00) >> 8;
    pid_packet.Build(pid_command, 4);
    charSendReceive(&pid_packet);
  }
  if(charrot_ki >= 0)
  {
    pid_command[0] = ROTKI;
    pid_command[1] = ARGINT;
    pid_command[2] = charrot_ki & 0x00FF;
    pid_command[3] = (charrot_ki & 0xFF00) >> 8;
    pid_packet.Build(pid_command, 4);
    charSendReceive(&pid_packet);
  }
  if(chartrans_kp >= 0)
  {
    pid_command[0] = TRANSKP;
    pid_command[1] = ARGINT;
    pid_command[2] = chartrans_kp & 0x00FF;
    pid_command[3] = (chartrans_kp & 0xFF00) >> 8;
    pid_packet.Build(pid_command, 4);
    charSendReceive(&pid_packet);
  }
  if(chartrans_kv >= 0)
  {
    pid_command[0] = TRANSKV;
    pid_command[1] = ARGINT;
    pid_command[2] = chartrans_kv & 0x00FF;
    pid_command[3] = (chartrans_kv & 0xFF00) >> 8;
    pid_packet.Build(pid_command, 4);
    charSendReceive(&pid_packet);
  }
  if(chartrans_ki >= 0)
  {
    pid_command[0] = TRANSKI;
    pid_command[1] = ARGINT;
    pid_command[2] = chartrans_ki & 0x00FF;
    pid_command[3] = (chartrans_ki & 0xFF00) >> 8;
    pid_packet.Build(pid_command, 4);
    charSendReceive(&pid_packet);
  }
*/

  printf("Robot READY\n");

  return(0);
}


void P2OS_MainQuit()
{
  unsigned char command;
  int count;

  for(count = 0; count < 3; count++){
    usleep(P2OS_CYCLETIME_USEC);
//rt_sleep(nano2count(P2OS_CYCLETIME_NSEC));
    command = STOP;
    P2OSPacket_Build(&tx_packet, &command, 1);
    P2OSPacket_Send(&tx_packet);
  }

  for(count = 0; count < 3; count++){
    usleep(P2OS_CYCLETIME_USEC);
//rt_sleep(nano2count(P2OS_CYCLETIME_NSEC));
    command = CLOSE;
    P2OSPacket_Build(&tx_packet, &command, 1);
    P2OSPacket_Send(&tx_packet);
  }

  usleep(P2OS_CYCLETIME_USEC);
//rt_sleep(nano2count(P2OS_CYCLETIME_NSEC));

  printf("P2OS stopped\n");
}

void SIP_ParseStandard(unsigned char *buffer,
p2os_data_t * p2os_data)
{
  int status;
  int cnt = 0;
  unsigned char i;
  double v_left, v_right;
  unsigned char numSonars;
  unsigned char sonarIndex;

  status = buffer[cnt]; // TYPE
  cnt += sizeof(unsigned char);

  /*
   * Remember P2OS uses little endian:
   * for a 2 byte short (called integer on P2OS)
   * byte0 is low byte, byte1 is high byte
   * The following code is host-machine endian
independant
   * Also we must or (|) bytes together instead of
casting to a
   * short * since on ARM architectures short * must
be even byte aligned!
   * You can get away with this on a i386 since
shorts * can be
   * odd byte aligned. But on ARM, the last bit of the
pointer will be ignored!
   * The or'ing will work on either arch.
   */

  // XPOS

  cnt += sizeof(short);

  // YPOS

  cnt += sizeof(short);

  // THPOS

  cnt += sizeof(short);

  // L VEL
  v_left = (short)(buffer[cnt] | (buffer[cnt+1] << 8));

  cnt += sizeof(short);

  // R VEL
  v_right = (short)(buffer[cnt] | (buffer[cnt+1] << 8));

  p2os_data->vel.vlin = ((v_left + v_right ) / 2) / 1e3;

  p2os_data->vel.vrot =  -((v_left - v_right) /
(2.0/PlayerRobotParams[param_idx].DiffConvFactor
));
```

```
  cnt += sizeof(short);

  // BATTERY
  p2os_data->power.volts =
((double)buffer[cnt])/10.0;  // Convert to volts

  cnt += sizeof(unsigned char);

  // STALL LEFT
  p2os_data->stall = buffer[cnt] & 0x01;

  cnt += sizeof(unsigned char);

  // STALL RIGHT
  p2os_data->stall |= (buffer[cnt] & 0x01)<<1;

  //sip_packet->frontbumpers = buffer[cnt] >> 1;
  cnt += sizeof(unsigned char);

  // CONTROL
  cnt += sizeof(short);

  // FLAGS
  cnt += sizeof(short);

  // COMPASS
  cnt += sizeof(unsigned char);

  // SONAR COUNT
  numSonars=buffer[cnt];
  cnt+=sizeof(unsigned char);

  if(numSonars>0)
  {
   if(numSonars > MAX_NUM_SONAR) // Saturate
number of sonars
     numSonars = MAX_NUM_SONAR;

    for(i=0;i<(numSonars-1);i++)
    {
     sonarIndex=buffer[cnt];

     p2os_data->sonar.ranges[sonarIndex]=(unsigned
int)
     ((((unsigned short)buffer[cnt+1] | ((unsigned
short)buffer[cnt+2] << 8)) )
     *
PlayerRobotParams[param_idx].RangeConvFactor);

     cnt+=sizeof(unsigned char)+sizeof(unsigned
short);
    }
  }

  cnt += sizeof(short);

  //analog = buffer[cnt];
  cnt += sizeof(unsigned char);

  //digin = buffer[cnt];
  cnt += sizeof(unsigned char);

  //digout = buffer[cnt];
  cnt += sizeof(unsigned char);

}


/* send the packe t, then receive and parse an SIP */
int P2OS_SendReceive(P2OSPacket_t * tx_pkt,
P2OSPacket_t * rx_pkt, char publish_data)
{

    if(tx_pkt)
      P2OSPacket_Send(tx_pkt);

    /* receive a packet */
    if(P2OSPacket_Receive_timeout(rx_pkt,
ignore_checksum,100E6)) // 100 ms
    {

/*// HABILITAR ESTO PARA CONOCER LA
FRECUENCIA DE RX*/
/*#ifdef DEBUG_IO*/
/*      rt_spset_mcr(RTCOM2, RT_SP_RTS, 1); //
ERRORES*/
/*      rt_sleep(nano2count(5E6)); // 5 mS*/
/*      rt_spset_mcr(RTCOM2, RT_SP_RTS, 0);*/
/*#endif*/

      return 1;
    }

   //   printf("K");

/*// HABILITAR ESTO PARA CONOCER LA
FRECUENCIA DE RX*/
/*#ifdef DEBUG_IO*/
/*      rt_spset_mcr(RTCOM2, RT_SP_DTR, 1); //
DATOS CORRECTOS*/
/*      rt_sleep(nano2count(5E6)); // 5 mS*/
/*      rt_spset_mcr(RTCOM2, RT_SP_DTR, 0);*/
/*#endif*/

    if(rx_pkt->packet[0] == 0xFA && rx_pkt-
>packet[1] == 0xFB &&
       (rx_pkt->packet[3] == 0x30 || rx_pkt-
>packet[3] == 0x31 ||
       rx_pkt->packet[3] == 0x32 || rx_pkt->packet[3]
== 0x33 ||
       rx_pkt->packet[3] == 0x34))
    {

      /* It is a server packet, so process it */
//      rt_sem_wait(p3dx_data.sem);
      sem_wait(p3dx_data.sem);
      SIP_ParseStandard(&rx_pkt-
>packet[3],&p2os_data_read);
      //rt_sem_signal(p3dx_data.sem);
      sem_post(p3dx_data.sem);


    }
    else if(rx_pkt->packet[0] == 0xFA && rx_pkt-
>packet[1] == 0xFB &&
           (rx_pkt->packet[3] == 0x20))
    {
     //CONFIGpac: do nothing
    }
     else
    {
     #ifdef DEBUG_ARCOS
     puts("Unknown packet: ");
     P2OSPacket_PrintHex(rx_pkt);
     #endif
    }

  return(0);
}



/* toggle sonars on/off, according to val */
void P2OS_ToggleSonarPower(unsigned char val)
{
```

```
  unsigned char command[4];

  command[0] = SONAR;
  command[1] = ARGINT;
  command[2] = val;
  command[3] = 0;
  P2OSPacket_Build(&tx_packet, command, 4);
  P2OS_SendReceive(&tx_packet,&rx_packet,0);
}

/* toggle motors on/off, according to val */
void P2OS_ToggleMotorPower(unsigned char val)
{
  unsigned char command[4];

  command[0] = ENABLE;
  command[1] = ARGINT;
  command[2] = val;
  command[3] = 0;
  P2OSPacket_Build(&tx_packet, command, 4);
  P2OS_SendReceive(&tx_packet,&rx_packet,0);
}


void P2OS_SendPulse (void)
{
  unsigned char command;
  command = PULSE;
  P2OSPacket_Build(&tx_packet,&command, 1);
  P2OS_SendReceive(&tx_packet,&rx_packet,0);
}

// Linear speed (m/s)
void P2OS_SendVEL(double vel)
{
  unsigned char command[4];
  unsigned short abs_vel;

  // Convert to mm/s
  short vel_int = (short) (vel*1e3);

  command[0] = VEL;

  if(vel_int >= 0)
    command[1] = ARGINT;
  else
    command[1] = ARGNINT;

  abs_vel = (unsigned short)abs(vel_int);

  if(abs_vel < MOTOR_MAX_LIN_SPEED)
  {
    command[2] = abs_vel & 0x00FF;
    command[3] = (abs_vel & 0xFF00) >> 8;
  }
  else
  {
    #ifdef DEBUG_ARCOS
    puts("Speed demand threshholded!\n");
    #endif
    command[2] = MOTOR_MAX_LIN_SPEED &
0x00FF;
    command[3] = (MOTOR_MAX_LIN_SPEED &
0xFF00) >> 8;
  }

  P2OSPacket_Build(&tx_packet,command, 4);
  P2OS_SendReceive(&tx_packet,&rx_packet,0);
}

// rot speed (rad/s)
void P2OS_SendROT(double rot)
{
  unsigned char command[4];
  unsigned short abs_rot;
  short rot_deg;

  rot_deg = (short)RTOD(rot); // Convert Radians to
Degress

  command[0] = RVEL;

  if(rot_deg >= 0)
    command[1] = ARGINT;
  else
    command[1] = ARGNINT;

  abs_rot = (unsigned short)abs(rot_deg);

  if(abs_rot < MOTOR_MAX_ROT_SPEED)
  {
    command[2] = abs_rot & 0x00FF;
    command[3] = (abs_rot & 0xFF00) >> 8;
  }
  else
  {
    #ifdef DEBUG_ARCOS
    puts("Turn rate demand threshholded!");
    #endif
    command[2] = MOTOR_MAX_ROT_SPEED &
0x00FF;
    command[3] = (MOTOR_MAX_ROT_SPEED &
0xFF00) >> 8;
  }

  P2OSPacket_Build(&tx_packet,command, 4);
  P2OS_SendReceive(&tx_packet,&rx_packet,0);
}

void P2OS_MainProcess(void)
{
  double lin_speed; // m/s
  double rot_speed; // rad/seg
  int sonar_enable = 0; // Default off
  int change_sonar = 0;
  int a = 0;

  P2OS_ToggleMotorPower(1); // Default: motors
enabled

    while(p2os_run) // Loop until exit
    {
      // Check exit

     if(p2os_cmd_write.sonar_enable ^
sonar_enable){
                sonar_enable ^= 1;
                change_sonar = 1;
      }

     if(change_sonar){
              P2OS_ToggleSonarPower(sonar_enable);
              change_sonar = 0;
     }

     //P2OS_SendPulse();

     // Update Linear speed
     //rt_sem_wait(p3dx_data.sem);
     sem_wait(p3dx_data.sem);
     lin_speed = p2os_cmd_write.lin_speed;
     //rt_sem_signal(p3dx_data.sem);
     sem_post(p3dx_data.sem);
```

```
    P2OS_SendVEL(lin_speed);  // R/W

    // Update rot speed
    //rt_sem_wait(p3dx_data.sem);
    sem_wait(p3dx_data.sem);
    rot_speed = p2os_cmd_write.rot_speed;
    //rt_sem_signal(p3dx_data.sem);
    sem_post(p3dx_data.sem);

    P2OS_SendROT(rot_speed);   // R/W

  } // while
}


void P2OSPacket_Print(P2OSPacket_t * pkt) {
  int i;
  if (pkt->packet) {
    printf("\"");
    for(i=0;i < pkt->size;i++) {
      printf("%u ", pkt->packet[i]);
    }
    puts("\"");
  }
}

void P2OSPacket_PrintHex(P2OSPacket_t * pkt) {
  int i;
  if (pkt->packet) {
    printf("\"");
    for(i=0;i < pkt->size ;i++) {
      printf("0x%.2x ", pkt->packet[i]);
    }
    puts("\"");
  }
}

int P2OSPacket_Check(P2OSPacket_t * pkt, int
ignore_checksum ) {

  unsigned short recv_chksum;
  unsigned short pkg_chksum;

  recv_chksum = (unsigned
short)(P2OSPacket_CalcChkSum(pkt) & 0xffff);
  pkg_chksum = ((unsigned short)(pkt->packet[pkt-
>size-2]) << 8) | pkt->packet[pkt->size-1];

  if ( ignore_checksum )
  {
    return(1);
  }
  else
  {
    return recv_chksum == pkg_chksum;
  }
}

int P2OSPacket_CalcChkSum(P2OSPacket_t * pkt)
{
  unsigned char *buffer = &pkt->packet[3];
  int c = 0;
  int n;

  n = pkt->size - 5;

  while (n > 1) {
    c+= (*(buffer)<<8) | *(buffer+1);
    c = c & 0xffff;
    n -= 2;
    buffer += 2;
  }

  if (n>0) c = c^ (int)*(buffer++);

  return(c);
}


int P2OSPacket_Receive_timeout(P2OSPacket_t *
pkt, char ignore_checksum, int timeout )
{
  unsigned char prefix[3];
  int cnt,temp;

  memset(pkt->packet,0,sizeof(pkt->packet));

  do
  {
    memset(prefix,0,sizeof(prefix));

    while(1)
    {
      cnt = 0;
      temp = read(serial_port, (void *)&prefix[2], 1);
      //temp = rt_spread_timed(RTCOM1, (void
*)&prefix[2], 1, nano2count(timeout));

      //if ((temp == SEM_TIMOUT) || (temp < 0))
      if (temp < 0)
      {
            // Si no se recibe nada: salir
            //printf("T1 ");
        return(1);
      }

      if (prefix[0]==0xFA && prefix[1]==0xFB)
        break;

      prefix[0]=prefix[1];
      prefix[1]=prefix[2];

    } // while(1)

    pkt->size = prefix[2]+3;
    memcpy( pkt->packet, prefix, 3);

    cnt = 0;
    while( cnt!=prefix[2] )
    {
          temp = read(serial_port, (void *)&pkt-
>packet[3+cnt], prefix[2]-cnt);
      //temp = rt_spread_timed(RTCOM1, (void
*)&pkt->packet[3+cnt], prefix[2]-cnt,
nano2count(timeout));

/*      if ((temp == SEM_TIMOUT) || (temp < 0))*/
          if (temp < 0)
      {
        return(1);
      }

      cnt += prefix[2]-cnt;

    } // while(...)

  } while
(!P2OSPacket_Check(pkt,ignore_checksum));

  // printf("SYNC OK\n");

  return(0);
}
```

```
int P2OSPacket_Build(P2OSPacket_t * pkt,
unsigned char *data, unsigned char datasize ) {
  unsigned short chksum;

  pkt->size = datasize + 5;

  /* header */
  pkt->packet[0]=0xFA;
  pkt->packet[1]=0xFB;

  if ( pkt->size > 198 ) {
    #ifdef DEBUG_ARCOS
    puts("Packet to P2OS can't be larger than 200
bytes");
    #endif
    return(1);
  }
  pkt->packet[2] = datasize + 2;

  memcpy( &pkt->packet[3], data, datasize );

  chksum = (unsigned short)
(P2OSPacket_CalcChkSum(pkt) & 0xffff);
  pkt->packet[3+datasize] = chksum >> 8;
  pkt->packet[3+datasize+1] = chksum & 0xFF;

  if (!P2OSPacket_Check(pkt,0)) {
    #ifdef DEBUG_ARCOS
    puts("DAMN");
    #endif
    return(1);
  }
   pkt->size = datasize + 5;
  return(0);
}

int P2OSPacket_Send(P2OSPacket_t * pkt)
{
  int cnt=0,temp;
  temp = write(serial_port, (void *)pkt->packet, pkt-
>size );

  //temp = rt_spwrite_timed(RTCOM1, (void *)pkt-
>packet, pkt->size, DELAY_FOREVER);

  if (temp < 0)
  {
    #ifdef DEBUG_ARCOS
    perror("Send");
    #endif
    return(1);
  }

  return(0);
}


#ifdef  MATLAB_MEX_FILE    /* Is this file being
compiled as a MEX-file? */
#include "simulink.c"      /* MEX-file interface
mechanism */
#else
#include "cg_sfun.h"       /* Code generation
registration function */
#endif
```

CABECERA

```
#ifndef _P2OSDRV_H
#define _P2OSDRV_H


#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>
#include <signal.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <stddef.h>
#include <string.h>
#include <strings.h>
#include <math.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <termios.h>
#include <sys/socket.h>
#include <netinet/tcp.h>
#include <netdb.h>
#include <limits.h>
#include <pthread.h>
#include <stdint.h>

//#include <rtai_lxrt.h>
//#include <rtai_sem.h>


// Llamada de RT Serial (debug IO)
//#define DEBUG_IO

//#define RT_HARD // Enable Hard Real Time Task

//#define DEBUG_ARCOS

// #define DEBUG_ARCOS // Enable to DEBUG
#define RTCOM1  0 // RT Serial

#define DEFAULT_BAUD B115200
#define MAX_NUM_SONAR 8

#ifndef Ts
#define Ts 0.01
#endif

#ifndef V_SAT
#define V_SAT  1.0
#endif

#ifndef O_SAT
#define O_SAT  3.0
#endif

#ifndef M_PI
   #define M_PI 3.14159265358979323846
#endif


// Configuración de aceleración máxima
#define MAX_ACCEL 2000

#define FILTER_CONSTANT (0.5)

#define ENABLE_FILTER
```

```
//Server Remote Center
#define PORT_UDP_COVE06500
#define DIR_UDP_COVE0   "192.168.11.11"

#define PORT_UDP_COVE16500
#define DIR_UDP_COVE1   "192.168.11.21"

#define PORT_UDP_COVE26500
#define DIR_UDP_COVE2   "192.168.11.31"

#define PORT_UDP_COVE36500
#define DIR_UDP_COVE3   "192.168.11.41"

#define PORT_UDP_COVE4  6500
#define DIR_UDP_COVE4   "192.168.11.51"

#define PORT_UDP_COVE5  6500
#define DIR_UDP_COVE5   "192.168.11.61"

#define PORT_UDP_COVE9  6500
#define DIR_UDP_COVE9   "192.168.1.102"


typedef struct
{
  int data_number;
  float lineal;
  float angular;
}comm_data_t;


//SERVER STRUCT
typedef struct
{
  int sockfd;
  struct sockaddr_in dir_cli;
  int long_dir_cli;
  struct sockaddr_in dir_serv;
  comm_data_t send_data;
  comm_data_t receive_data;
  comm_data_t storage_data[4];
}server_t;



// Readings from a robot's sonars.
typedef struct sonar_data
{
  unsigned int ranges[MAX_NUM_SONAR];     // The
range readings [mm]
} sonar_data_t;

// The power interface data in this format.
typedef struct power_data
{
  float volts; // Battery voltage [V]
} power_data_t;

typedef struct vel2d
{
  double vlin; // m/s
  //double vlin_acc; // m/s
  double vrot; // rad/s
  //double vrot_acc; // rad/s
} vel2d_t;

typedef struct p2os_data
{
  vel2d_t vel; //  velocities [m/s,rad/s]
  char stall;  // Are the motors stalled?
  sonar_data_t sonar;
  power_data_t power;
} p2os_data_t;

typedef struct p2os_cmd
{
  double lin_speed; // m/s
  double rot_speed; // rad/seg
  int sonar_enable;
} p2os_cmd_t;

 // Variable to read data and write commands
extern p2os_data_t p2os_data_read;
extern p2os_cmd_t p2os_cmd_write;

// Mutex to protect variable access
//extern pthread_mutex_t p2os_mutex;
//extern SEM *sem_p2os;


/**
 Main Program

 Interface to P3-DX robot
 (Non-returning function)
*/
void * P2OS_Main(void *devicep);

/// This function tell to interface that must finish
void P2OS_Terminate(void);

#endif
```

III.3. ANEXO III: Datasheet Robot Pioneer 3-DX

adept mobilerobots

Pioneer 3-DX

Pioneer 3-DX is a small lightweight two-wheel two-motor differential drive robot ideal for indoor laboratory or classroom use. The robot comes complete with front SONAR, one battery, wheel encoders, a microcontroller with ARCOS firmware, and the Pioneer SDK advanced mobile robotics software development package.

Pioneer research robots are the world's most popular intelligent mobile robots for education and research. Their versatility, reliability and durability have made them the preferred platform for advanced intelligent robotics. Pioneers are pre-assembled, customizable, upgradeable, and rugged enough to last through years of laboratory and classroom use.

Product Features and Benefits

- **Easy to Use** - Comes assembled and integrated with its accessory packages.
- **Reliable** - Construction is durable and rugged. Easily handles the small gaps, minor bumping, jarring, or other obstacles that hinder other robotic platforms. Some Pioneer robots have been in service for over 15 years.
- **Pioneer Software Development Kit** - All Adept MobileRobots platforms include Pioneer SDK, a complete set of robotics applications and libraries that accelerate the development of robotics projects. Pioneer SDK is backed by our product support team.
- **Customizable** - Easily accessorize by choosing from dozens of supported and tested accessories that integrate with the robotic platform. Additional help is available for future upgrades or added accessories.
- **Reference Platform** - Pioneer robots are a standard in intelligent mobile platforms. Search your preferred robotics journal or conference listings to find many examples of Pioneer platforms in research applications.
- **Technical Support** - Pioneer software and hardware comes fully documented with additional help available through our product support team.

Specifications

Construction
- Body: 1.6 mm aluminum (powder-coated)
- Tires: Foam-filled rubber

Operation
- Robot Weight: 9 kg
- Operating Payload: 17 kg

Differential Drive Movement
- Turn Radius: 0 cm
- Swing Radius: 26.7 cm
- Max. Forward/Backward Speed: 1.2 m/s
- Rotation Speed: 300°/s
- Max. Traversable Step: 2.5 cm
- Max. Traversable Gap: 5 cm
- Max. Traversable Grade: 25%
- Traversable Terrain: Indoor, wheelchair accessible

Power
- Run Time: 8-10 hours w/3 batteries (with no accessories)
- Charge Time: 12 hours (standard) or 2.4 hrs (optional high-capacity charger)
- Available Power Supplies:
  - 5 V @ 1.5 A switched
  - 12 V @ 2.5 A switched

Batteries
- Supports up to 3 at a time
- Voltage: 12 V
- Capacity: 7.2 Ah (each)
- Chemistry: lead acid
- Hot-swappable Batteries: Yes

Available Recharge Options:
- Direct plug-in
- Docking station
- Powercube (3-battery charging bay)

* Batteries are accessible through hinged latched access panel for hot-swapping (continuous operation)

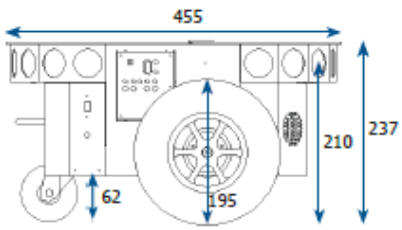
Microcontroller I/O
- System Serial
- 32 digital inputs
- 8 digital outputs
- 7 analog inputs
- 3 serial expansion ports

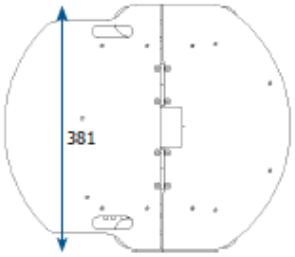
*Some ports may not be available if certain accessories are included with the robot

User Control Panel
- MIDI programmable piezo buzzer
- Main power indicator
- Battery charge indicator
- 2 AUX power switches
- System reset
- Motor enable pushbutton

Pioneer 3-DX

Dimensions (mm)





Core Software - included with all research platforms

ARIA provides a framework for controlling and receiving data from all MobileRobots platforms, as well as most accessories. Includes open source infrastructures and utilities useful for writing robot control software, support for network sockets, and an extensible framework for client-server network programming.

MobileSim open-source simulator which includes all MobileRobots platforms and many accessories.

MobileEyes graphical user interface client for remote operation and monitoring of the robot.

Mapper 3-Basic tool for creating and editing map files for use with ARIA, MobileSim, and navigation software.

SONARNL provides sonar-based approximate localization and navigation.

Accessory Support Software - bundled with purchase of robotic accessory

ARNL enables robust, laser-based autonomous localization and navigation.

Robotic Arm Support Pioneer arms are packaged with integrated software support.

Speech Recognition and Synthesis Library: Easy-to-use C++ development library for speech recognition based on the open source Sphinx2 system. Speech synthesis (text-to-speech) based on Cepstral synthesizer.

ACTS Color Tracking System: Software application which reads images from a camera and tracks the positions and sizes of multiple color regions. Information can be incorporated into your own software via ARIA.

Optional Industrial Grade Internally Mounted Computers

Mamba EBX-37 (Dual Core 2.26 GHz - 2-8 GB RAM)
6 X USB2.0 Ports
2 X PC/104+ Slots
4 X RS-232 Serial Ports
2 X 10/100/1000 Ethernet Ports
Onboard Audio & Video
Solid State Drive
Optional Wireless Ethernet

Optional Accessories:

- Laser-range finders
- Mono- and stereo-vision cameras
- Rear SONAR
- Wireless serial to Ethernet for remote operation
- Robotic arms and grippers
- Gyroscope
- Segmented bumper arrays
- Speakers and microphones
- Joystick
- Many more...

Include our integrated & supported accessories with your Pioneer 3-DX.

Here are some popular configurations to choose from:

Mapping & Vision

Gripping & Manipulation

Audio & Speech

More Information:

See our website www.mobilerobots.com for a full range of supported accessories or contact our sales department to discuss your application.

Adept Technology, Inc. 10 Columbia Drive, Amherst, NH 03031
Tel: 603-881-7960 Email: sales@mobilerobots.com
www.mobilerobots.com

09366-P3DX Rev. A

III.4. ANEXO IV: Tools for communication and execution of experiments with the robot in Windows 10.

Index

1. Creating and configuring a model for real-time execution in Simulink

1.1 Simulink model configuration

In order to generate the executable, it is necessary to correctly configure the Simulink model as it is explained below.

Solver

As it is shown in Figure 1, the Solver configuration must be the following:

- **Type**: Fixed-Step
- **Solver**: discrete (no continuous states)
- **Fixed-step size**: Ts (the chosen sample time)

Additionally, both "***Tasking and sample time options***" at the bottom of in Figure 1 must be selected.

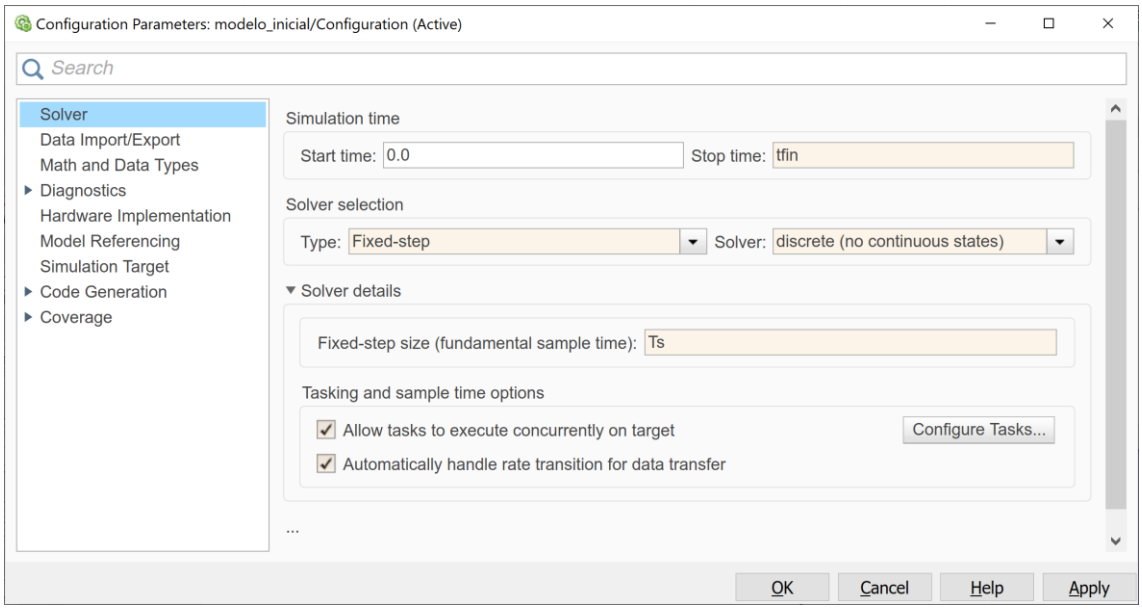


Figure 1. "Solver" configuration in Simulink

Hardware implementation

The configuration within the Hardware implementation window is shown in Figure 2.

- **Hardware board**: Determine by Code Generation system target file
- **Device vendor**: Intel (or AMD, depending on the PC hardware characteristics)
- **Device Type**: x86-64 (Windows 64)

Generally, all these parameters are correctly configured by default, but it is important checking if there are correct before generating an executable.

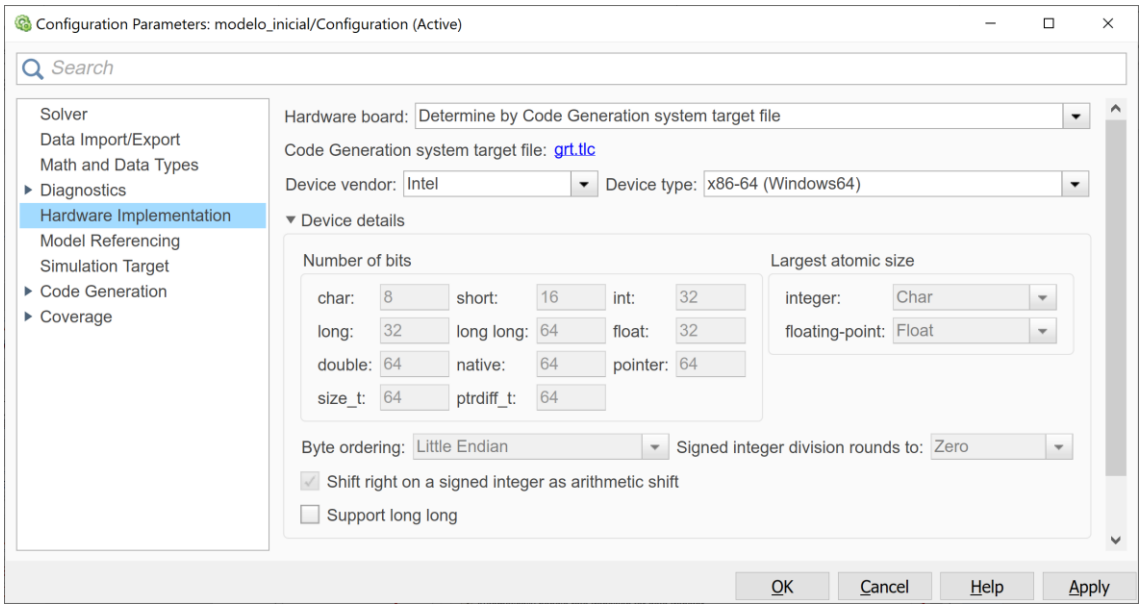


Figure 2. "Hardware Implementation" configuration in Simulink

Code generation

First of all, if the "*Code Generation*" option does not appear, it is necessary to install the Matlab module "*Simulink Coder*", following the instructions in ***Appendix 1.*** . Moreover, it can be also necessary installing the Mingw64 compiler, following the instructions in the Appendix 2.

Then, the code generation configuration must be set as shown in Figure 3:

- **Toolchain**: MinGW64 | gmake (64 bit Windows)
- **Build configuration**: Faster Runs
- **Select objective**: Execution efficiency

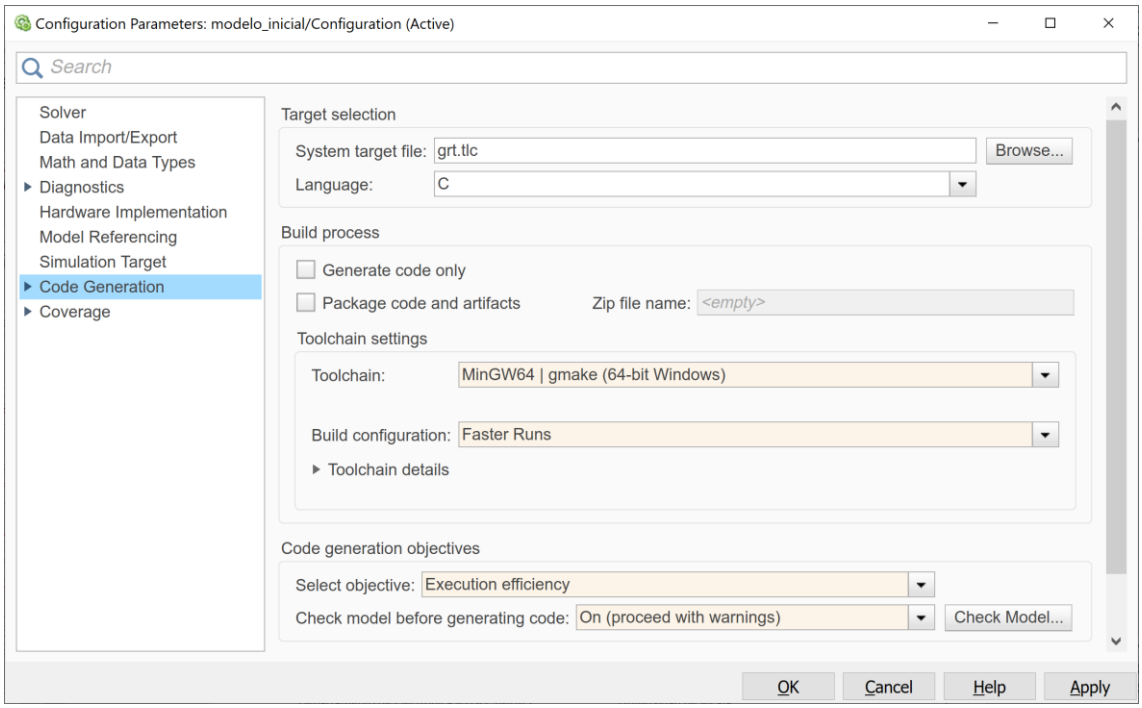


Figure 3. "Code Generation" configuration in Simulink

Interface

In addition, it must be set the standard math library version to C99 (ISO). This is done in the configuration menu, under the "Interface" section. Specifically, accessing to "Advanced parameters" by clicking on the three points that appear at the bottom of the window "Interface" (Figure 4).

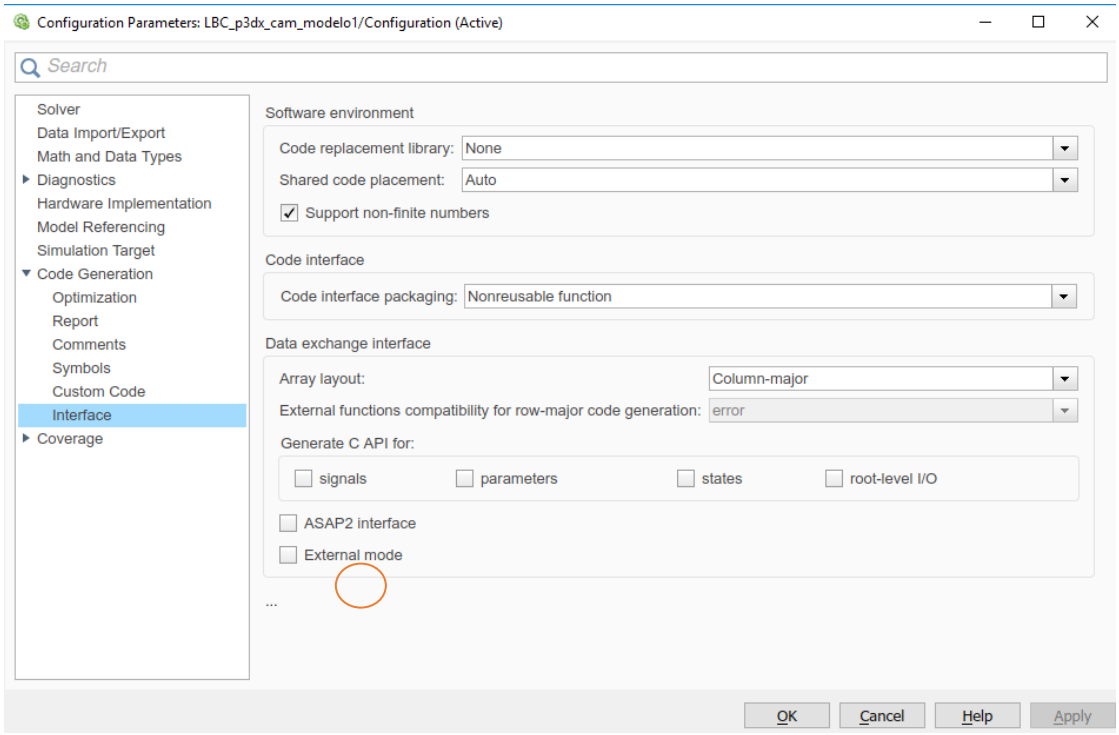


Figure 4. Access to the configuration of the advanced parameters, from the menu Code Generation/Interface

After doing this, new options appear. In the first one: "Standard math library", the option "C99 (ISO)" must be chosen.

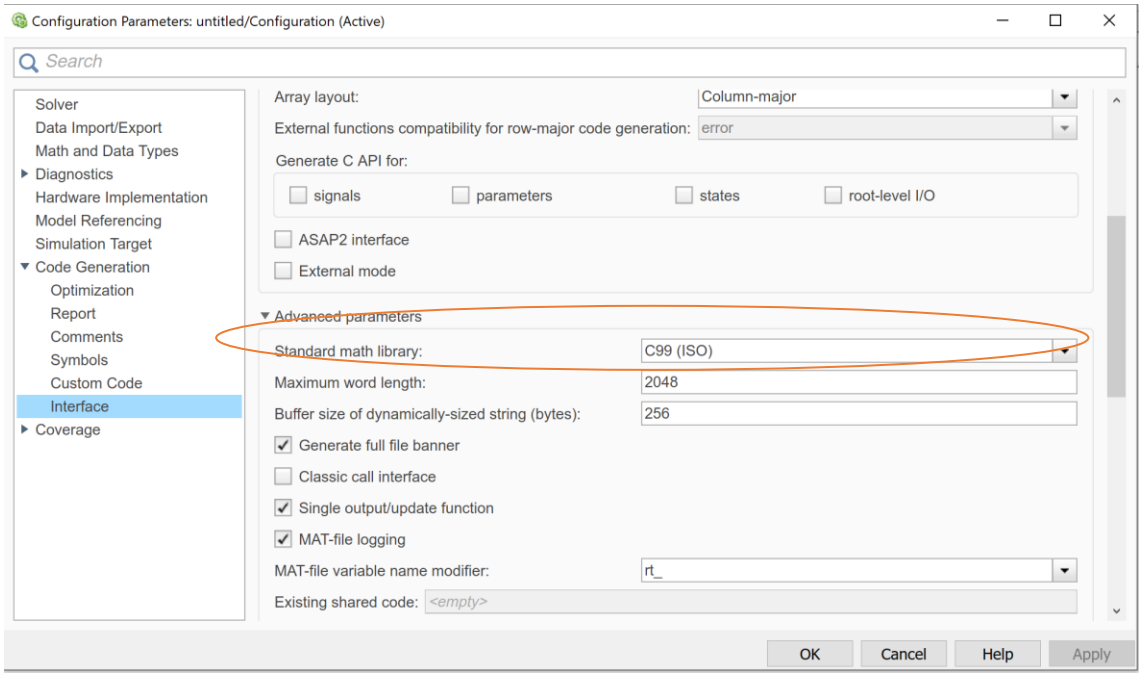


Figure 5. Standard math library configuration, in the Advanced parameters within the "Interface" window.

1.1. S-function comunica_dr_cam_w10 for the access to the robots and the camera

The available s-function comunica_dr_cam_w10 allows communication with the robots and cameras in Simulink. To compile the s-function code, the following files are required in the Matlab working folder:

- ***comunica_dr_cam_w10.c***: source code file
- ***centro_remoto.h***: header file with the main definitions
- ***WS2_32.lib***: library for socket communications.

To generate the s-function with the external hardware interface driver, the source file (*.c) with the designed code has to be compiled. This allows to obtain the corresponding *.mexw64 file using the following command[1]:

```
mex comunica_dr_cam_w10.c -lWS2_32 -Icentro_remoto
```

This *.mexw64 file is used by the s-function block in Simulink, that performs the PC-robot interface tasks. Then, the s-function of the User-Defined Functions library can be incorporated into the Simulink model. (see Figure 6)

[1] This process requires the MinGW-w64 compiler. If it is not installed, it can be downloaded and installed following the steps in the ***Appendix 2. How to install the MinGW compiler in Matlab***.

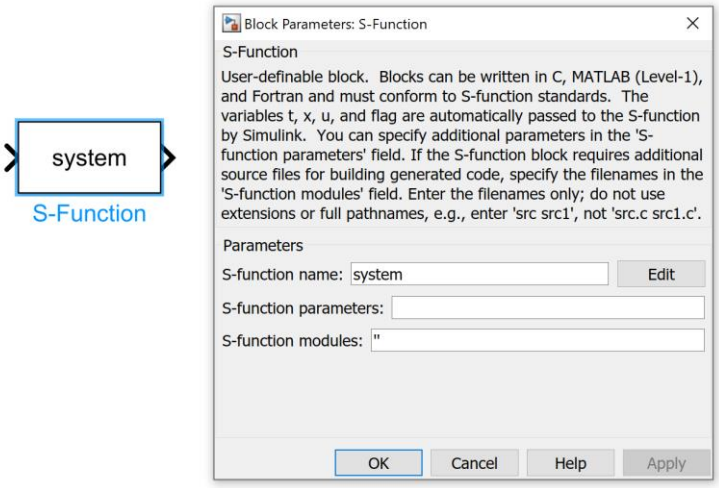


Figure 6. s-function Simulink block for configurating the external hardware interfaz driver between the PC and the robot.

In our case, the driver receives two inputs and provides 10 outputs, as shown in Figure 8. The first 5 correspond to the robot, while the last 5 are associated with the camera measurements. The most important outputs are detailed below.

The s-function ***comunica_dr_cam_w10*** has 3 input parameters:

- the **sample time** (Ts),
- the **robot name** (a string with the format 'coveX' between single quotes, where X is the robot number)
- the **camera number** (21, 22, 31, 32 or 41) according to the diagram in Figure 15.

Figure 7 shows an example where the s-funtion has been configured to work with the robot cove0, and the camera 31.

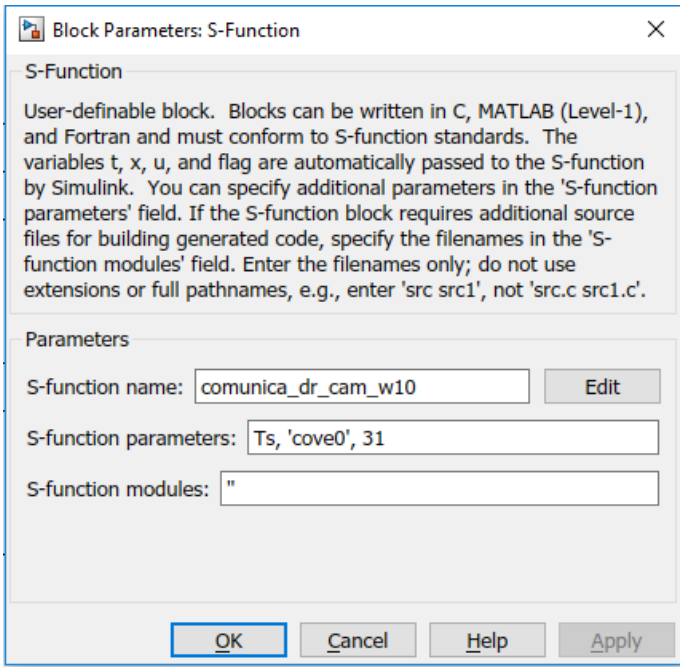


Figure 7. Example of the configuration of the s-function comunica_dr_cam_w10 for the robot cove0 and the camera 31.

After compiling the s-function, it must be included in the Simulink model. Figure 8 shows the s-function inputs and outputs. In this figure, the inputs correspond to the desired linear and angular velocity values. Regarding the outputs, the s-function provides 10 outputs, the first 5 correspond to the robot, while the last 5 are associated with the camera measurements. The most important ones are detailed below:

- *V_lin, V_ang*: linear and angular velocities measured by the odometric sensors onboard the robot.
- *New_meas*: flag that is activated when a new measurement is available.
- *Pose*: pose $(x, y, \theta)^T$ measured by the camera, with (x, y) in m and θ in rad.
- *R*: measurement error covariance matrix. Since it is a symmetric matrix, only 6 values are needed:

$$[R_{11}, R_{12}, R_{21}, R_{22}, R_{23}, R_{33}] \rightarrow R = \begin{pmatrix} R_{11} & R_{12} & R_{13} \\ R_{12} & R_{22} & R_{23} \\ R_{13} & R_{23} & R_{33} \end{pmatrix}$$

- *k*: sample number.

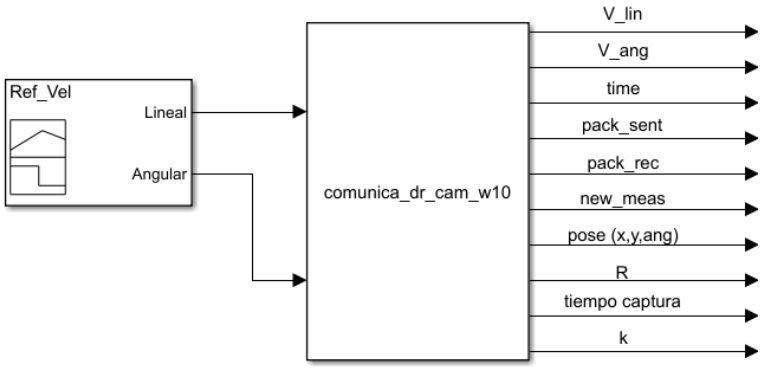


Figure 8. Diagram of the s-function comunica_dr_cam_w10 for the communication with the robot and one of the cameras.

Connection to the robots

Checking connection

To check the remote connection between the PC and the robot, it can be used the *ping* command to the robot IP or hostname "coveX" (see Table 1) from the Command Prompt (Figure 9).

cove0	192.168.11.11	**cove2**	192.168.11.31	**cove4**	192.168.11.51
cove1	192.168.11.21	**cove3**	192.168.11.41	**cove5**	192.168.11.61

Table 1. Hostnames and IP addresses assigned to each robot.

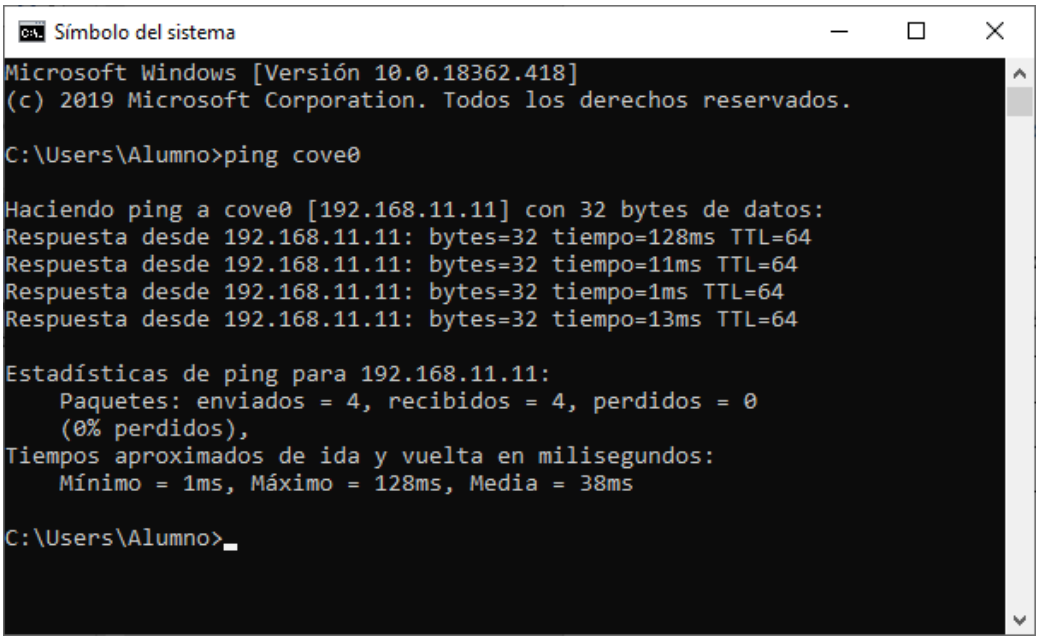


Figure 9. Response to ping command for cove0 in the Command Prompt.

SSH connection through PuTTY

To access to the robots from the laboratory PC through ssh, it is used the tool *PuTTY*[2], *see* Figure 10.

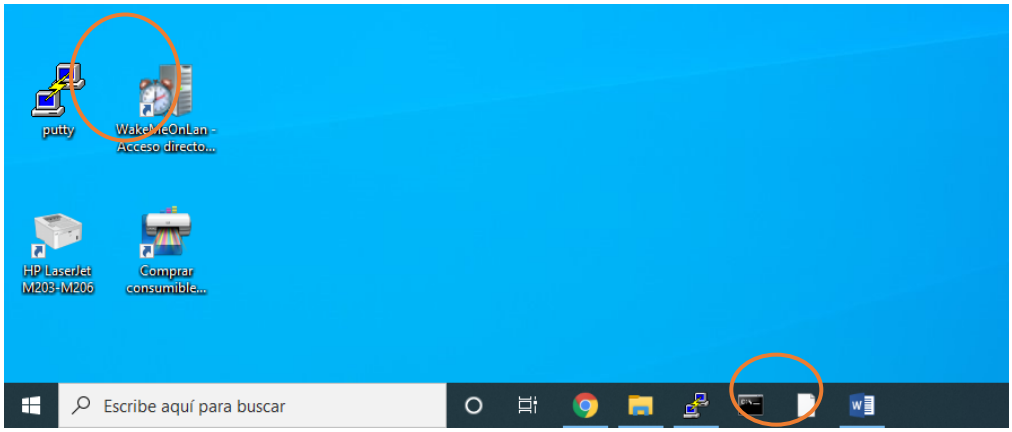


Figure 10. Desktop and PuTTY shorcuts in the lab.

The connections for the available robots are already configured on the laboratory PC (Figure 11). So it is enough to make double-click on the robot to which you it is desired to connect. If the connections were not saved, it is sufficient to enter the IP address (or the robot hostname: coveX) in the field available for this purpose, and then press the "Open" button to open the connection. It is also possible to include the user name before the IP, separated by the character '@' (Figure 12).

In all the robots, the information for logging in is the following: **Usuario**: root; **Password**: 123456

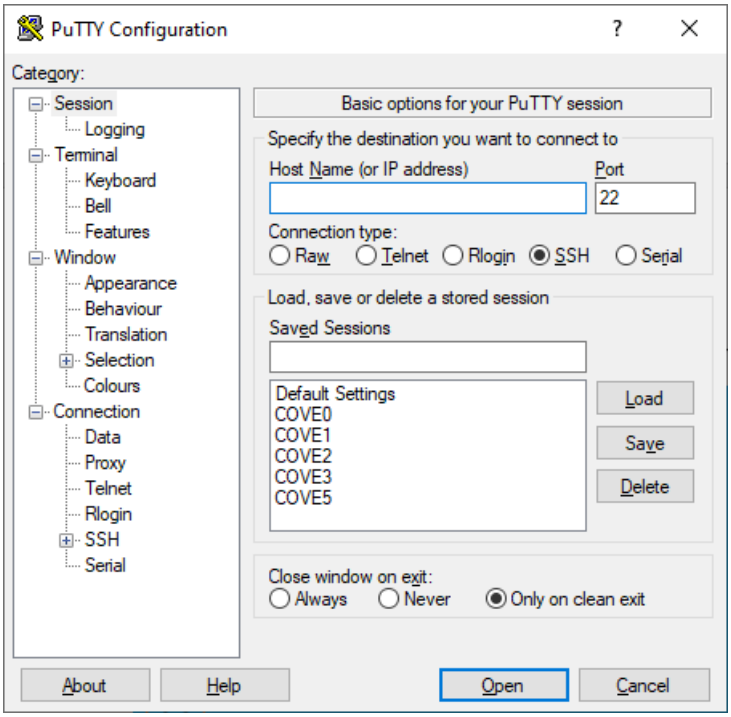


Figure 11. PuTTY main window, included the Saved Sessions for the robots.

[2] Available for its download in https://www.PuTTY.org/

Figure 12 shows an example including the configuration for the connection to the robot cove0 (whose IP address is 192.168.11.11), whereas Figure 13 presents the login with the user name “root”.

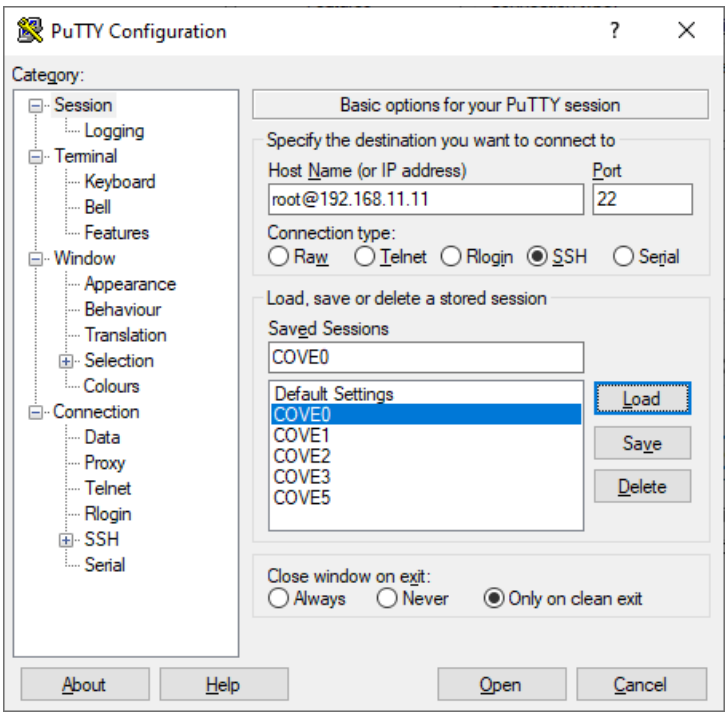


Figure 12. PuTTY configuration window with the information for the connection the cove0 robot for the user “root”.

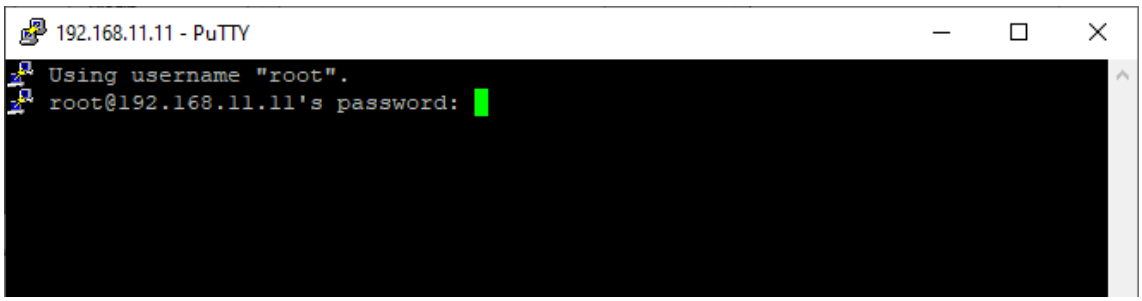


Figure 13. Robot response window.

After login, the system responds with a connection to the robot via remote terminal as shown in Figure 14.

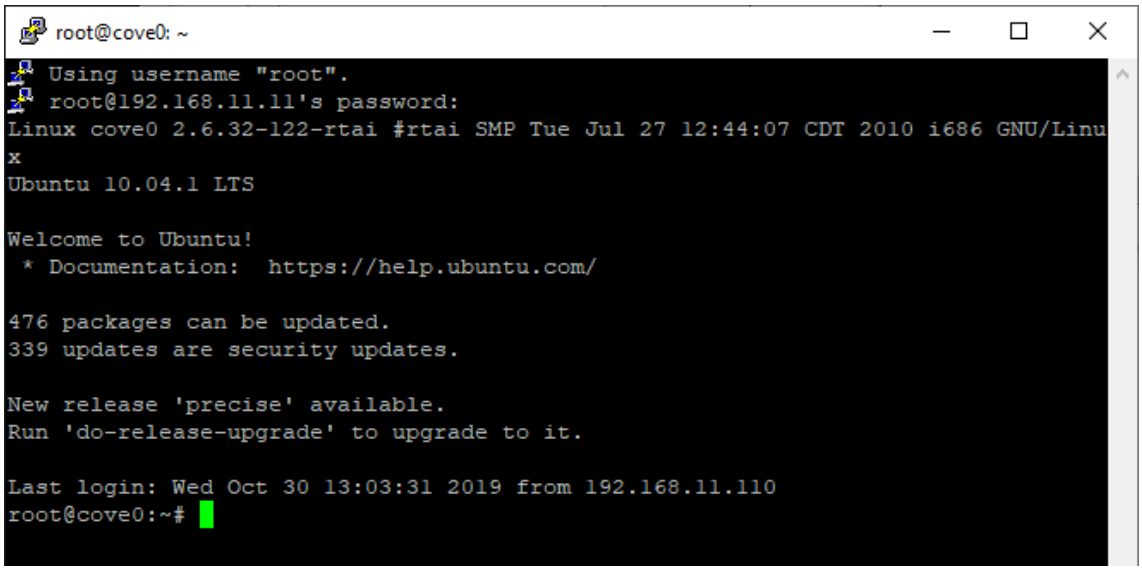


Figure 14. Robot response window when requesting remote Access through PuTTY.

1. Execution of experiments with the robots

Once connected to the robot, it can be accessed through the PuTTY terminal. To launch the executable, the following steps must be followed:

1.1 In the remote terminal (robot through ssh):

1. Access the folder where the executable is located:

```
# cd /home/coveN/target
```

2. Launch the executable, which remains on hold until it receives commands from the PC.

```
# ./robotMUIE -v -f  t_execution
```

   - *t_execution* is the execution time in seconds.
   - Only in cove 5, the following command must be executed previously:

```
# ./TEST_Robot_con_servo –v –f 1
```

1.1. In the PC

After launching the executable ***robotMUIE*** in the robot, the .exe generated from Simulink must be run on the local PC. After the configured time (t_execution), the results are stored in one, or several, ".mat" files, that can be loaded in Matlab.

1. Communication with the cameras

In the laboratory corridor, there are 5 cameras available, controlled by 3 mini PCs, with the distribution shown in Figure 15.

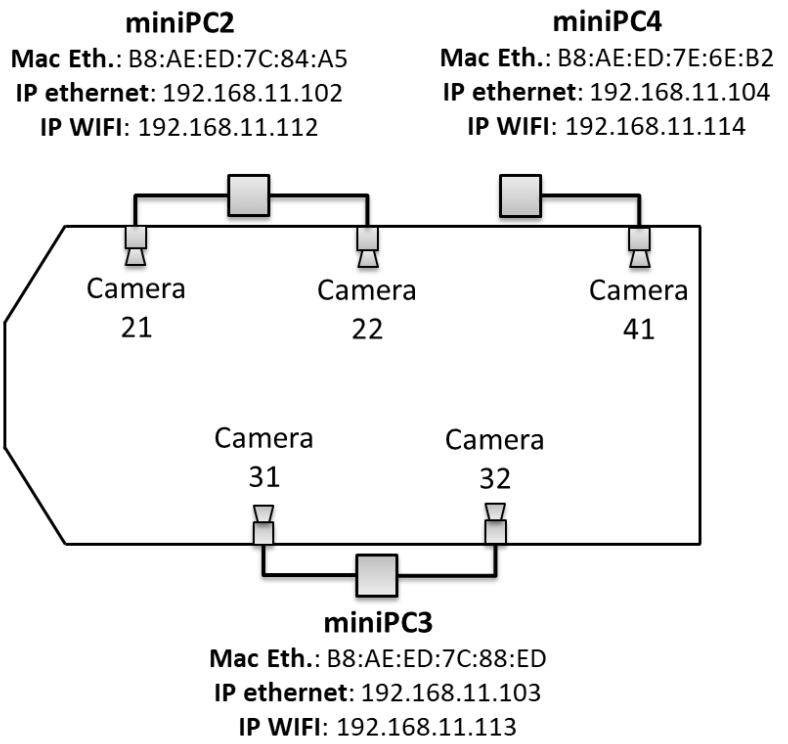


Figure 15. Diagram with the distribution of cameras and mini PCs in the labs LO1-L06 corridor.

1.1 Access to the cameras

The first step to access the Kinect cameras is to boot up the corresponding mini PC via "Wake on Lan". Since in Windows 10 this option does not exist natively, it can be done using the program " *WakeMeOnLan*[3]", whose shortcut can be found on the desktop in the lab. PC (Figure 16).

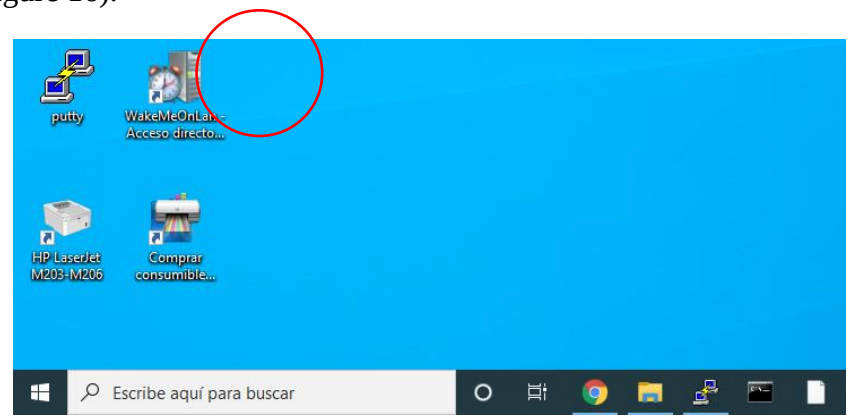


Figure 16. Desktop and shortcut to WakeMeOnLan

Once the WakeMeOnLan program is started, a window like the one shown in Figure 21 appears, in which the MAC addresses of the mini-PCs are configured. To boot up a mini PC, it is enough to select it from the list and press F8.

[3] Available for downloading in: https://www.nirsoft.net/utils/wake_on_lan.html

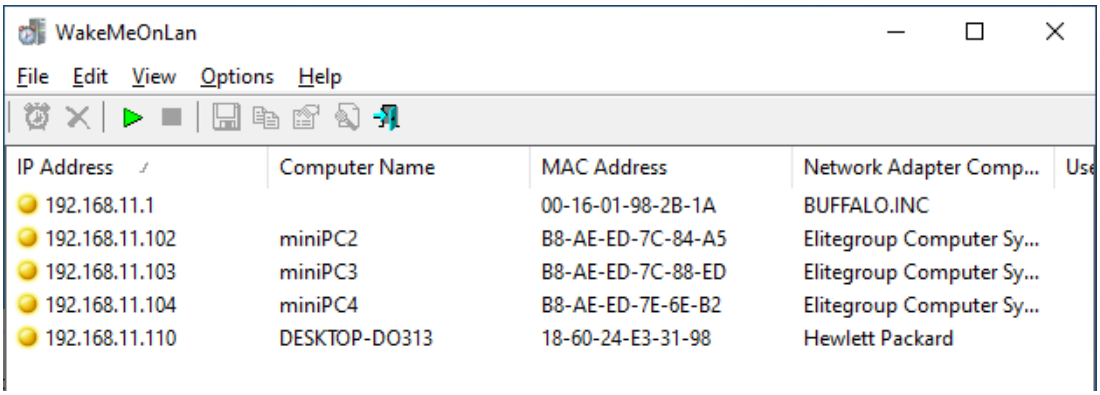


Figure 17. Main window of the WakeMeOnLan programm.

As with the robots, it is possible to check whether a mini PC is accessible by running the ping command to the IP address or using the hostname (minipcX, where X is the number of mini PC). The relationship between the robot and cameras hostnames and IP addresses can be configured in the system file "hosts", within the directory: “C:\Windows\System32\drivers\etc”. Currently, this file includes the elements shown in Figure 18, but there can be added new ones by editing this hosts file.

```
192.168.11.11    cove0
192.168.11.21    cove1
192.168.11.31    cove2
192.168.11.41    cove3
192.168.11.51    cove4
192.168.11.61    cove5
```

(a)

```
192.168.11.102  minipc2
192.168.11.103  minipc3
192.168.11.104  minipc4
```

(b)

Figure 18. Lists of devices includes in the Windows 10 hosts file in the lab. PC (a) Robots, (b) miniPCs

Once the mini-PC has been booted, it can be accessed using the PuTTY tool, just as with the robots. This can be done by entering the IP address or host name in the main program window (Figure 19). The user and password for access to the mini PCs is: **alcor/alcor.**

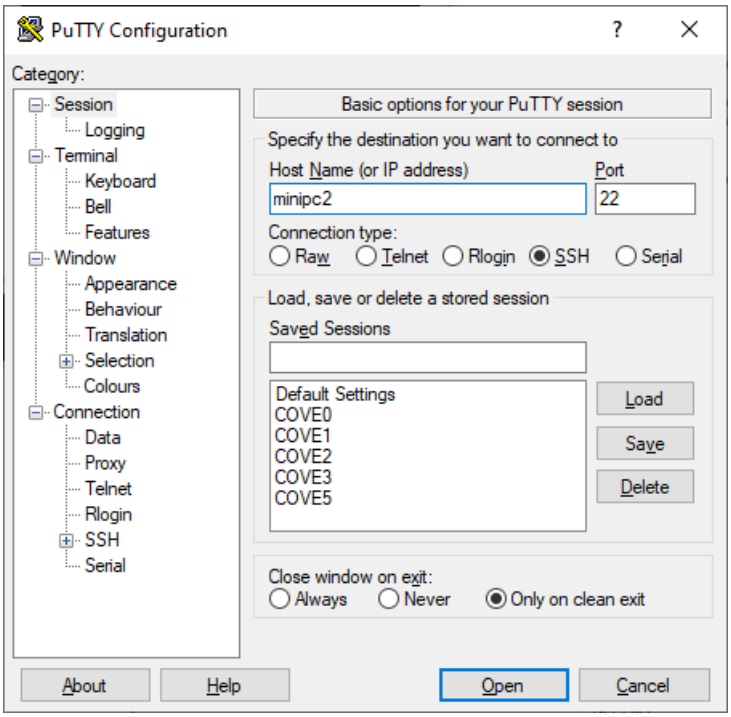


Figure 19. Example of configuration for accessing to minipc2 in the PuTTY main window.

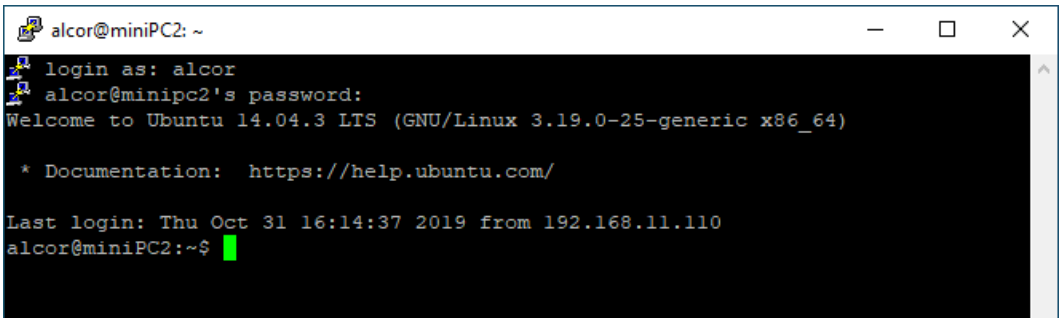


Figure 20. Example of the robot remote terminal.

1.1. Access to the cameras through the remote terminal

Once connected to the mini PC, the following steps must be followed in the remote terminal:

1. Access to the folder where the executable is located:

```
$ cd /home/alcor/workspace/miguel
```

2. There are two different alternatives:

   a. Launch the executable "***camnet2***" directly from the command Prompt. In this case, the camera number and the execution time must be specified (as shown in Figure 22). For example, the following command allows using the camera 21 for 120 seconds.

```
$ DISPLAY=:0.0 ./camnet2 –ncam 21 –f 120
```

   b. Modify and run the "***camnet2_launcher***" script, which performs all the necessary steps for the camera configuration, and the

software execution. The editor "vim[4]"can be opened with the following command:

```
$ vim camnet2_launcher
```

This causes the contents of the script to appear (Figure 21) on which the runtime must be adjusted.
To enter text insertion mode, press the "i" key. Once the edition is finished, to save and/or exit, first press ESC to finish the edition, and then:

- *:q* to exit without saving the changes.
- *:wq* to sabe changes and exit.

After modifying the script, it can be launched using the following command:

```
$ ./camnet2_launcher ncamara (Ej.: $ ./camnet2_launcher 21)
```

When running camnet2 or camnet2_launcher, the result of the execution is displayed on the terminal, showing information about the detected robots. Each robot has a tag (tagID) as shown in Table 2

Robot	tagID	Robot	tagID
cove0	9	cove3	39
cove1	19	cove5	49
cove2	29		

Table 2. TagID asigned to each robot.

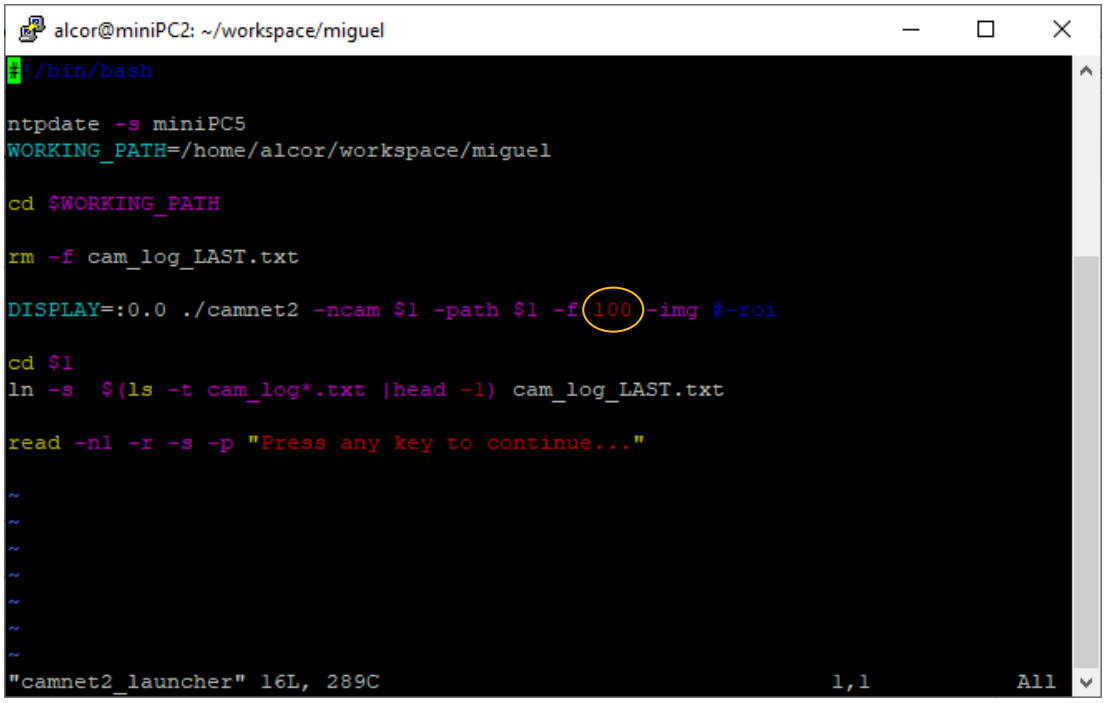


Figure 21. Script camnet2_launcher in the editor Vim

[4]More information about the Vim editor can be found in: https://openwebinars.net/blog/vim-manual-de-uso-basico/.

The following figures show two examples in which the result of the execution of the script for reading the cameras can be seen, in case no robot is detected (Figure 22), and having detected the robot cove0 (Figure 23)

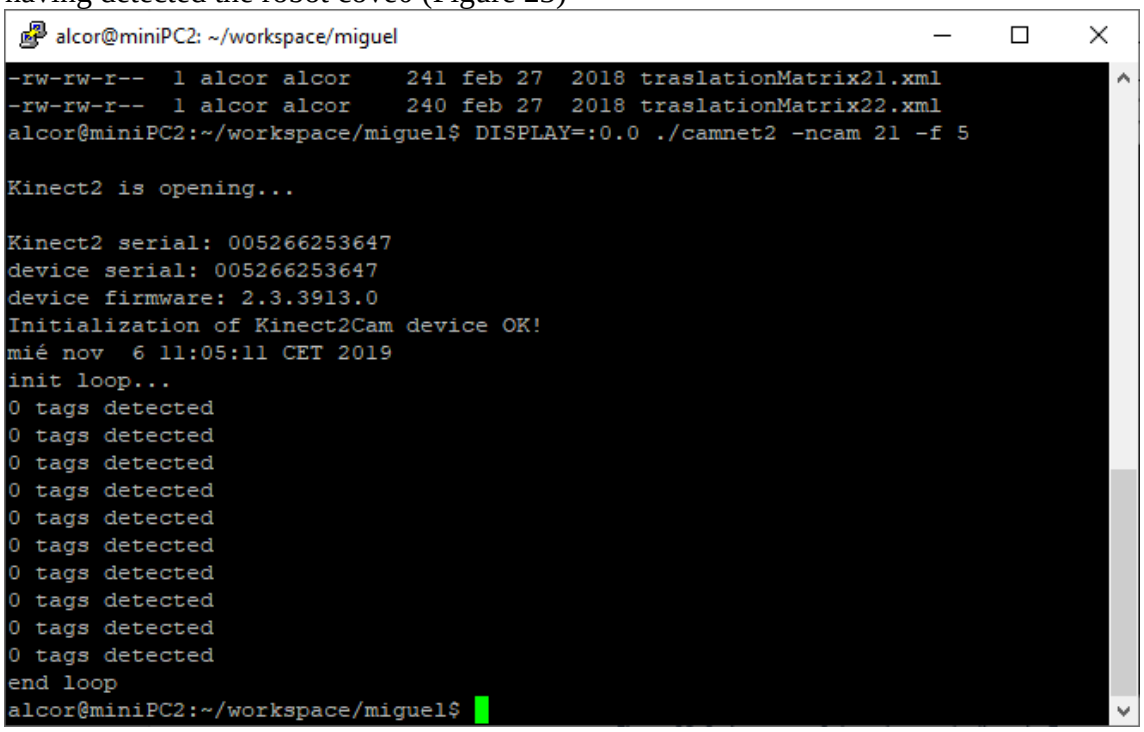


Figure 22. Result after running Camnet2 without detecting any robot.

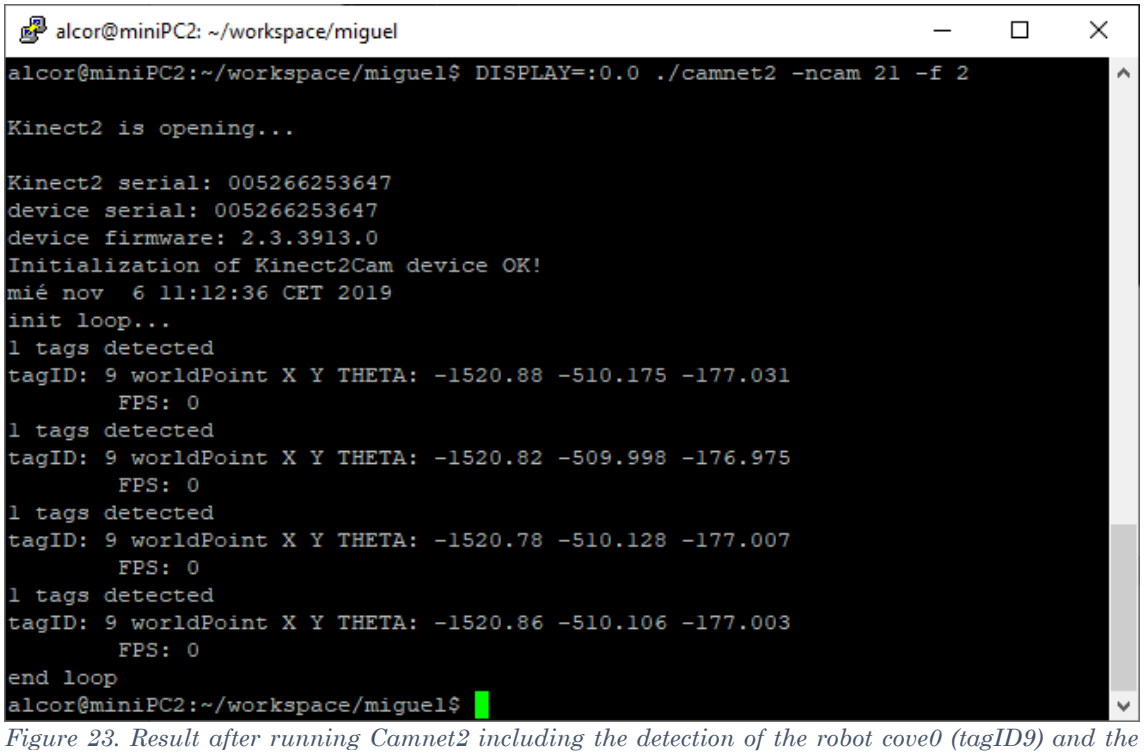


Figure 23. Result after running Camnet2 including the detection of the robot cove0 (tagID9) and the measured pose (X, Y are in mm and θ in degrees).

1 Execution of experiments with the cameras

The camera driver can be attached both only to obtain pose information, or to include that information into the control loop. In the first case, it is enough to save the data obtained from the camera in a file. In case of wanting to compare this pose with the reference pose or the robot output one, it should be taken into account the initial pose of the robot in the world coordinate system. Furthermore, to incorporate the pose information from the cameras to the control algorithm, it is necessary to incorporate a block with the kinematics of the robot, to obtain the pose (x, y, θ) from the robot's linear an angular speed. In the Figure 24, it is shown an example of a system in which it has incorporated the driver that includes the communication with the robot and the camera.

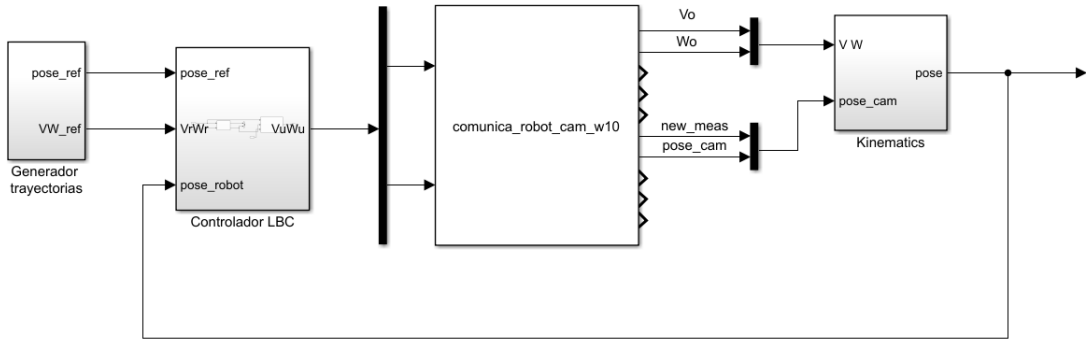


Figure 24. Example of Simulink model that includes the driver comunica_robot_cam_w10 for communication with the robot and the camera.

The subsystem "Kinematics" has to include different blocks that model the kinematics of the robot, according to the following equations:

$$\theta_k = \theta_{k-1} + \omega_k \cdot T_s$$

$$v_{x_k} = v_k \cdot \cos(\theta_k) \rightarrow x_k = x_{k-1} + v_{x_k} \cdot T_s$$

$$v_{y_k} = v_k \cdot \text{sen}(\theta_k) \rightarrow y_k = y_{k-1} + v_{y_k} \cdot T_s$$

There are different alternatives to model these equations. One of them is the use of integrating blocks, and blocks ***Fcn*** with mathematical functions. There must be two different kinematic blocks: one for the generation of trajectories from the linear and angular reference speed, and another one used at the output, for the incorporation of information from the cameras in the control loop.

1.1 Kinematics block configuration for path generation

The input kinematics block (Figure 25) models the above mentioned equations to obtain the reference pose, from linear and angular velocity. To compare this pose to with the output, it is necessary to set the initial values in the discrete-time integrators, in the box reserved for it (Initial Condition in Figure 26).

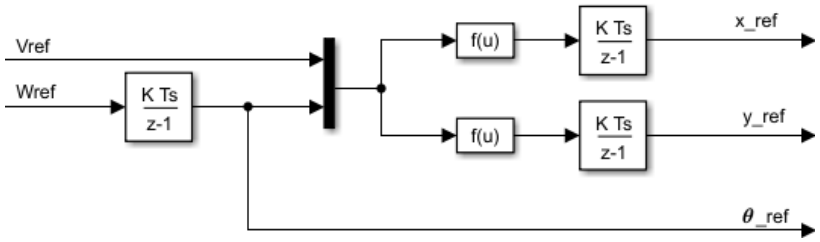


Figure 25. Input Kinematics block, to obtain the reference path from linear and angular velocity.

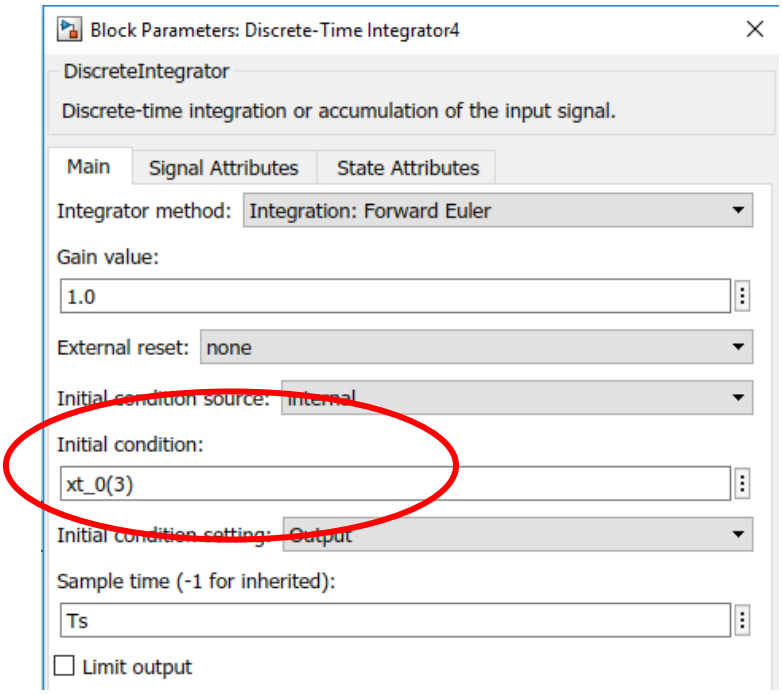


Figure 26. Configuration of an integrator in the input kinematics subsystem, with internal initial condition set to xt_0(3).

These values can be obtained by accessing the mini PC connected to the camera chosen by ssh, as explained in section 0, and running camnet2 for a short time. For example, to run camnet2 for camera 31, for 2 seconds, we must write the following line in the terminal.

```
$ DISPLAY=:0.0 ./camnet2 –ncam 31 –f 2
```

As it has been seen previously, in case of detecting a robot, this command shows in the screen information about the pose of the robot, corresponding to their coordinates (x, y) in millimeters, and the angle (θ) in degrees, so it is necessary to make a change of units to introduce the initial values of pose in meters and radians.

1.1. Kinematics block configuration for incorporating the camera measurements to the control loop.

Figure 27 shows an example of the output kinematics block. In this subsystem, the discrete-time Integrator blocks, must be configured so that the external reset (*External reset*) is activated by rising edge, and the initial condition (Initial condition source) is also

external, as shown in Figure 28. Thus, each time there is a new measurement from the camera (*new_meas* is activated), the integrator is restarted with that value.

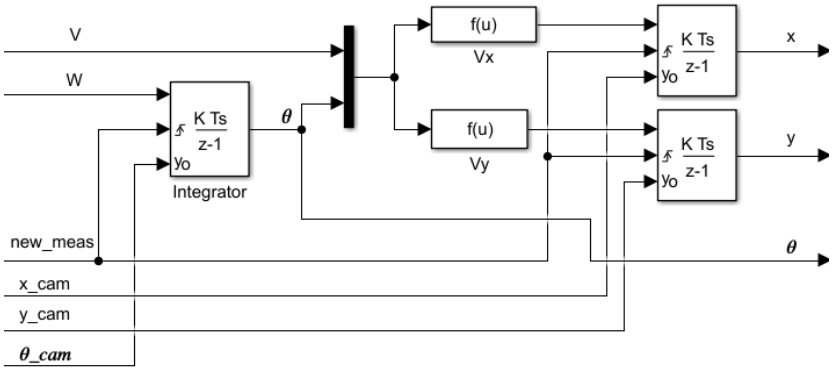


Figure 27. Example of the output kinematic subsystem for the P3DX robot.

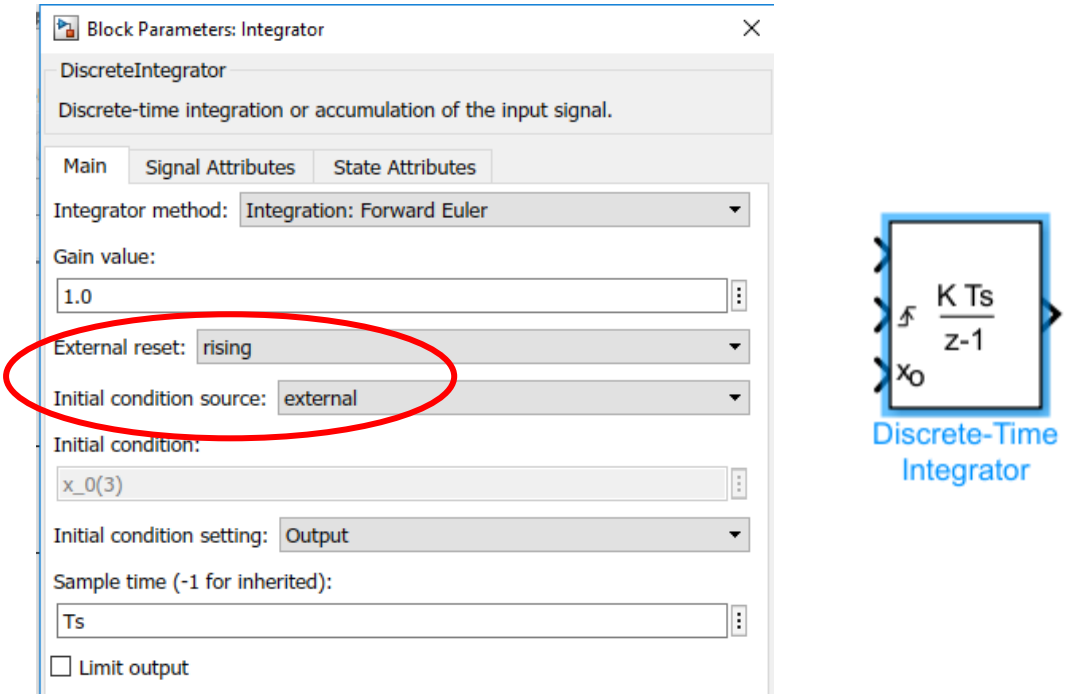


Figure 28. Configuration of the integrator block with external reset and initial condition.

Appendix 1. How to install the Simulink Coder Module in Matlab

To generate code from Simulink models, the "Simulink Coder" tool must be installed. In case it is not installed, it can be downloaded from the Matlab menu, in the Apps tab, selecting "Get more apps" (Figure A1)

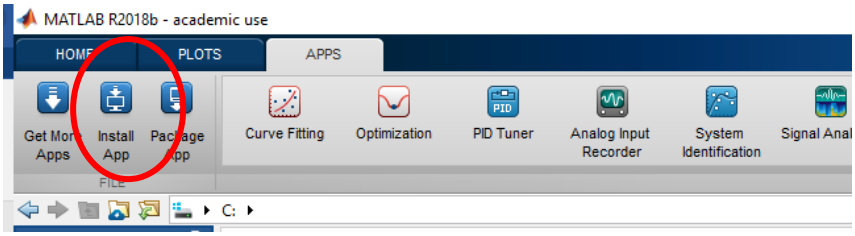


Figure A1. Tab APPs in Matlab

Once the application browser is open (Figure A2) the next step is to find and install the Simulink Coder, following the instructions on screen. After finishing the installation, Matlab must be restarted before using it.

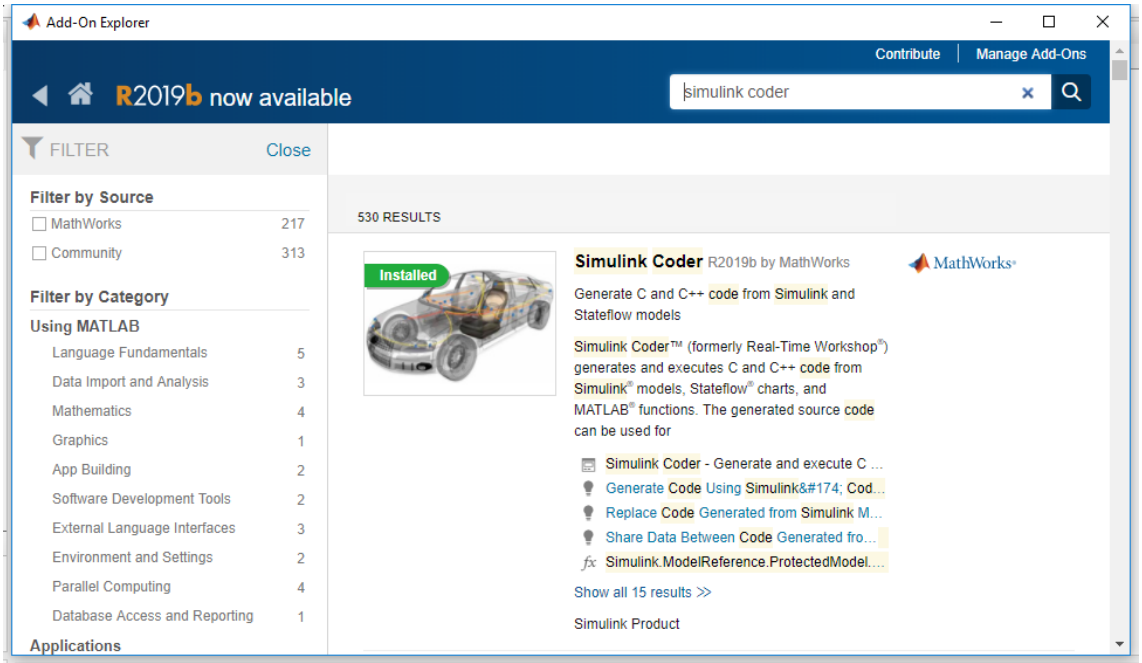


Figure A2. Matlab apps explorer

Appendix 2. How to install the MinGW compiler in Matlab

To compile the s-functions written in C/C++, it is necessary to have previously installed the MinGW-w64 compiler in Matlab. In case it is not installed, it can be downloaded from the Matlab menu, in the Apps tab, selecting "Get more apps" (Figure A3)

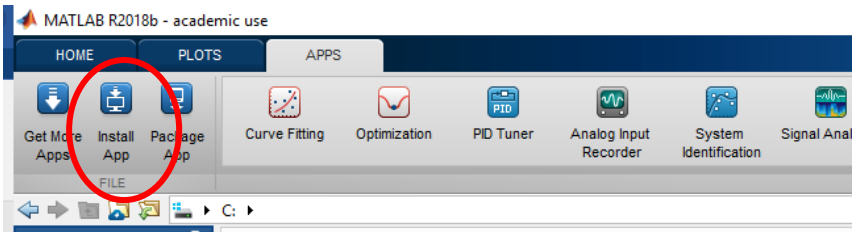


Figure A3. Tab APPs in Matlab

Once the application browser is open (Figure A4) the next step is to find and install the C/C++ compiler for Matlab MinGW-w64, following the instructions on screen. After finishing the installation, Matlab must be restarted before using the compiler.

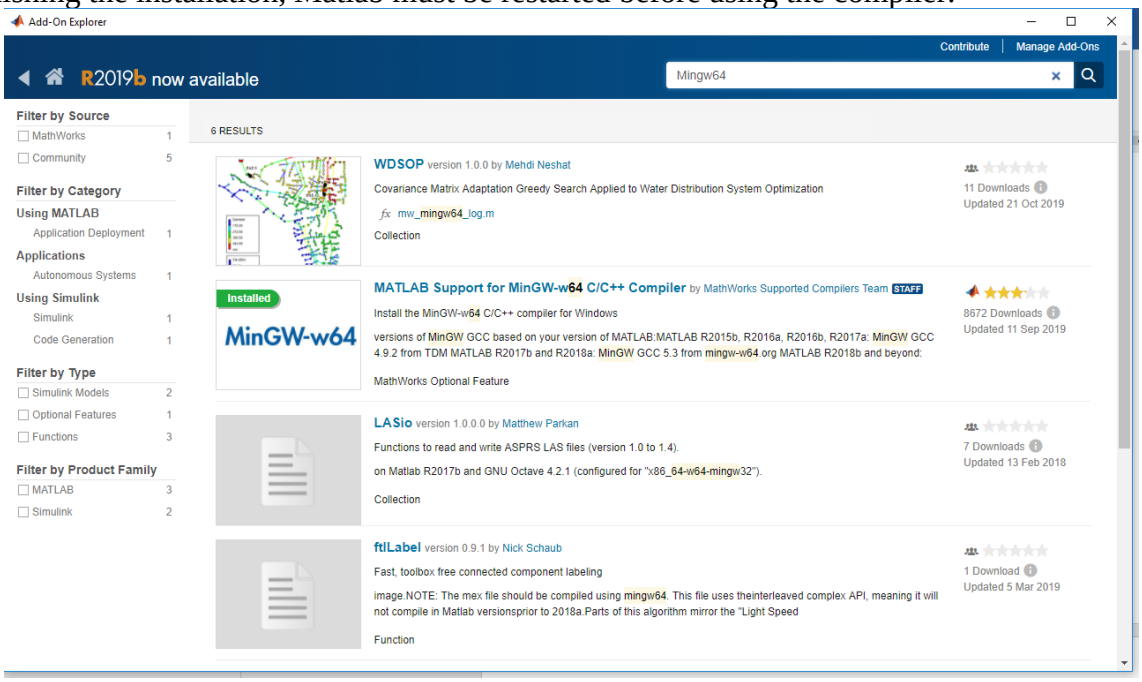


Figure A4. Matlab Apps explorer

To check if the compiler is correctly installed, the command *mex –setup* should be executed. Getting the answer shown in Figure A5 if everything is correct.

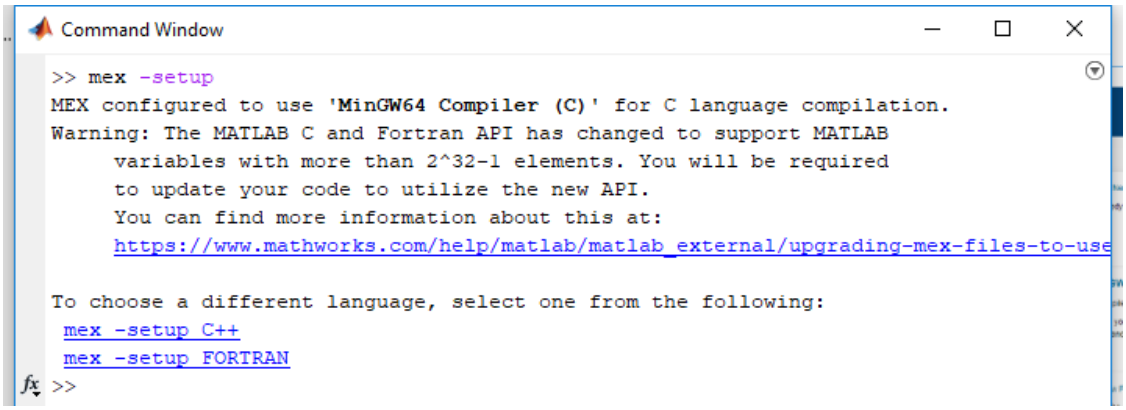


Figure A5. Information about the C/C++ compiler installed and configured in Matlab.

Appendix 3. Summary of the steps for communication and execution of experiments with the robot (and cameras) in Windows 10.

1. Create and configure the Simulink model

Tab	Configuration
Solver	**Type**: Fixed-Step **Solver**: discrete (no continuous states) **Fixed-step size**: Ts (the chosen sample time) **Tasking and sample time options**: the following options must be selected: *Allow tasks to execute concurrently on target* *Automatically handle rate transition for data transfer*
Hardware implementation	**Hardware board**: Determine by Code Generation system target file **Device vendor**: Intel (or AMD, depending on the PC hardware characteristics) **Device Type**: x86-64 (Windows 64)
Code generation[5]	**Toolchain**: MinGW64 | gmake (64 bit Windows) **Build configuration**: Faster Runs **Select objective**: Execution efficiency
Interface	Accessing to "Advanced parameters" by clicking on the three points that appear at the bottom of the window "Interface". After doing this, new options appear. In the first one: "Standard math library", the option "C99 (ISO)" must be chosen.

2. Compile the driver comunica_dr_cam_w10

Required files (there must be in the working folder):

- ***comunica_dr_cam_w10.c***: source code file
- ***centro_remoto.h***: header file with the main definitions
- ***WS2_32.lib***: library for socket communications.

Command for compiling the driver (it creates comunica_dr_cam_w10.mexw64)

```
mex comunica_dr_cam_w10.c -lWS2_32 -Icentro_remoto
```

3. Include in Simulink the s-function for the driver and configure it

The s-function ***comunica_dr_cam_w10*** has 3 ***input parameters***:

[5] If the "*Code Generation*" option does not appear, it is necessary to install the Matlab module "*Simulink Coder*". It can be also necessary installing the Mingw64 compiler.

- the **sample time** (Ts),
- the **robot name** (with the format 'coveX' between single quotes, where X is the robot number)
- the **camera number** (21, 22, 31, 32 or 41) according to the diagram in Figure 15.

In case of using the cameras, **configure the initial values for the reference and the robot**[6]

4. Complete the model and build it

With ctrl+B in Simulink, or using the command rtwbuild('model_name');

5. Connection to the robots and execution of experiments

Once connected to the robot, it can be accessed through the PuTTY terminal. To launch the executable, the following steps must be followed:

In the remote terminal (robot through ssh)	***In the PC***
1. Access the folder where the executable is located: # cd /home/coveN/target 2. Launch the executable, which remains waiting until it receives commands from the PC (t_exec is the execution time in seconds) # ./robotMUIE -v -f t_exec	After the executable ***robotMUIE*** in the robot, the .exe file generated from Simulink must be run on the local PC. The results are stored in one, or several, ".mat" files, that can be loaded in Matlab.

6. Communication with the cameras and execution of experiments

In the PC

1. Configure the Simulink model for using the cameras. **Do not forget to configure the initial values for the reference and the robot** in the kinematics subsystems**.** If needed, compile the driver and build the executable.
2. Access the robot remote terminal using the PuTTY tool. The user and password is: root/123456
3. Boot up the camera miniPC via "Wake on Lan" (if needed)

[6] The initial pose of the robot can be obtained by accessing the mini PC connected to the camera chosen by ssh, and running camnet2 for a short time. In case of detecting a robot, this command shows in the screen information about the pose of the robot, corresponding to their coordinates (x, y) in millimeters, and the angle (θ) in degrees, so it is necessary to make a change of units to introduce the initial values of pose in meters and radians.

4. Access the camera's miniPC using the PuTTY tool. The user and password is: alcor/alcor.

In the remote terminal (robot through ssh)

5. Access the folder where the executable is located:

```
# cd /home/coveN/target
```

6. Launch the executable, which remains waiting until it receives commands from the PC (t_exec is the execution time in seconds)

```
# ./robotMUIE -v -f  t_exec
```

In the remote terminal (camera's miniPC through ssh)

7. Access to the folder where the executable is located:

```
$ cd /home/alcor/workspace/miguel
```

8. Launch the camera executable. There are two different alternatives:
   a. Launch the executable "***camnet2***" directly from the command Prompt (the camera number and the execution time must be specified). For example, using camera 21 for 120 seconds.

```
$ DISPLAY=:0.0 ./camnet2 –ncam 21 –f 120
```

   b. Modify and run the "***camnet2_launcher***" script. It can be launched using the following command:

```
$ ./camnet2_launcher ncamara (Ej.: $ ./camnet2_launcher 21)
```

In the PC

9. After the executable robotMUIE in the robot, the .exe file generated from Simulink must be run on the local PC. The results are stored in one, or several, ".mat" files, that can be loaded in Matlab.

II.5. ANEXO V: Ejecutable para el robot, Código fuente y Cabeceras

ROBOT_DR.C

```
#define S_FUNCTION_NAME  robot_dr
#define S_FUNCTION_LEVEL 2

#define _BSD_SOURCE

#include "simstruc.h"
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>
#include <signal.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <stddef.h>
#include <string.h>
#include <strings.h>
#include <math.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <termios.h>
#include <sys/socket.h>
#include <netinet/tcp.h>
#include <netdb.h>
#include <limits.h>
#include <pthread.h>
#include <stdint.h>
#include <errno.h>
#include <semaphore.h>

#include "robot_dr.h"

#if defined(RT)
#warning  DEFINED !!!
#elif MATLAB_MEX_FILE
#warning MATLAB_MEX_FILE DEFINED !!!
#endif

//VARIABLES SELF-TRIGGERED

float v_lin_local;
float v_rot_local;
float recibido=0;
float v_consigna=0;
float o_consigna=0;

// Robot data
typedef struct
{
  pthread_t thread;
  sem_t *sem;
}p3dx_data_t;

p3dx_data_t p3dx_data;

 //Protocolo de parada
 int parada=0;

//COMUNICACIONES
 int envio=1;
 int numero_envio=0;
 int recibe= 0;
 int errorbind=0;
 pthread_t receive;
 void * receive_function(void * ptr);
 server_t rc_server;

// SERIAL PORT (added CLG)
struct termios tty;
int serial_port;

////////////////////////////////////////////////////////

/* Function: mdlInitializeSizes
=============================

 * Abstract:
 *   Setup sizes of the various vectors.
 */
static void mdlInitializeSizes(SimStruct *S)
{
   ssSetNumSFcnParams(S, 1);
```

```
    if (ssGetNumSFcnParams(S) !=
ssGetSFcnParamsCount(S)) {

        return; /* Parameter mismatch will be reported by
Simulink */

    }


    if (!ssSetNumInputPorts(S, 0)) return;


    if (!ssSetNumOutputPorts(S,2)) return;


    ssSetOutputPortWidth(S, 0, DYNAMICALLY_SIZED);
//Linear velocity odometry

    ssSetOutputPortWidth(S, 1, DYNAMICALLY_SIZED);
//Angular velocity odometry



    ssSetNumSampleTimes(S, 1);


    /* Take care when specifying exception free code - see
sfuntmpl_doc.c */


    // SS_OPTION_PLACE_ASAP: this is typically used by
devices connecting to hardware.
   ssSetOptions(S, SS_OPTION_PLACE_ASAP);


}




/* Function: mdlInitializeSampleTimes
=============================================

 * Abstract:

 *    Specify that we inherit our sample time from the
driving block.

 */

static void mdlInitializeSampleTimes(SimStruct *S)

{


    ssSetSampleTime(S, 0, Ts);

    ssSetOffsetTime(S, 0, 0.0);


}


#define MDL_START  /* Change to #undef to remove
function */
```

```
#if defined(MDL_START)

  /* Function: mdlStart
=============================================
==========

  * Abstract:

  *    This function is called once at start of model execution.
If you

  *    have states that should be initialized once, this is the
place

  *    to do it.

  */

 static void mdlStart(SimStruct *S)

 {

#if defined(RT)

   int temp;

   int i, cove;

   char DIR_UDP_COVE[20];

   int PORT_UDP_COVE;

   char robot[10];


   // read parameter with robot name

   mxGetString((ssGetSFcnParam(S, 0)),robot,6);

   cove = atoi(&robot[4]);


   switch(cove)

   {

     case 0:

       PORT_UDP_COVE = PORT_UDP_COVE0;

       strcpy(DIR_UDP_COVE, DIR_UDP_COVE0);

       break;

     case 1:

       PORT_UDP_COVE = PORT_UDP_COVE1;

       strcpy(DIR_UDP_COVE, DIR_UDP_COVE1);

       break;

     case 2:

       PORT_UDP_COVE = PORT_UDP_COVE2;

       strcpy(DIR_UDP_COVE, DIR_UDP_COVE2);

       break;

     case 3:

       PORT_UDP_COVE = PORT_UDP_COVE3;

       strcpy(DIR_UDP_COVE, DIR_UDP_COVE3);

       break;

     case 4:
```

```
        PORT_UDP_COVE = PORT_UDP_COVE4;
        strcpy(DIR_UDP_COVE, DIR_UDP_COVE4);
        break;
      case 5:
        PORT_UDP_COVE = PORT_UDP_COVE5;
        strcpy(DIR_UDP_COVE, DIR_UDP_COVE5);
        break;
  }
  printf("\nDir.IP: %s, PORT: %d\n", DIR_UDP_COVE, PORT_UDP_COVE );


  /* create binary semaphore with semaphore available and queue tasks on FIFO basis */
  //p3dx_data.sem = rt_typed_sem_init(nam2num("SEM_P2OS"), 1, BIN_SEM | FIFO_Q );

  p3dx_data.sem = malloc(sizeof(sem_t));
  sem_init(p3dx_data.sem,0,1);

  // Control when serial port is opened
  temp = open("/dev/ttyUSB0", O_RDWR);
  //temp = rt_spopen(RTCOM1, 115200, 8, 1, RT_SP_PARITY_NONE, RT_SP_NO_HAND_SHAKE, RT_SP_FIFO_SIZE_4);

  if (temp < 0) {
          printf("RT Serial 1 open error (%d)\n",temp);
  }
  else{
     printf("RT Serial 1 opened id: %d\n",temp);
     serial_port = temp;
  }

  tty.c_cflag &= ~PARENB;  // Clear parity bit, disabling parity (most common)
  tty.c_cflag &= ~CSTOPB;  // Clear stop field, only one stop bit used in communication (most common)
  tty.c_cflag &= ~CSIZE;   // Clear all bits that set the data size
  tty.c_cflag |= CS8;      // 8 bits per byte (most common)
  tty.c_cflag &= ~CRTSCTS; // Disable RTS/CTS hardware flow control (most common)
  tty.c_cflag |= CREAD | CLOCAL; // Turn on READ & ignore ctrl lines (CLOCAL = 1)
  tty.c_cc[VTIME] = 2;     // wait for up to 200 ms (2 deciseconds)
  tty.c_cc[VMIN] = 0;
  cfsetispeed(&tty, B115200);
  if (tcsetattr(serial_port, TCSANOW, &tty) != 0) {
    printf("Error %i from tcsetattr: %s\n", errno, strerror(errno));
  }

#ifdef DEBUG_IO
  // Control when serial port is opened
  temp = open("/dev/ttyUSB0", O_RDWR);
  //temp = rt_spopen(RTCOM2, 115200, 8, 1, RT_SP_PARITY_NONE, RT_SP_NO_HAND_SHAKE, RT_SP_FIFO_SIZE_4);

  if (temp < 0) {
          printf("RT Serial 2 open error (%d)\n",temp);
  }
  else{
     printf("RT Serial 2 opened\n");
     serial_port = temp;
  }
#endif
  // SOCKET SEVERS

  //REMOTE CENTER SERVER
  bzero((char *)&rc_server.dir_serv,sizeof(rc_server.dir_serv));
  rc_server.dir_serv.sin_family=AF_INET;

rc_server.dir_serv.sin_addr.s_addr=inet_addr(DIR_UDP_COVE);
  rc_server.dir_serv.sin_port=htons(PORT_UDP_COVE0);

if((rc_server.sockfd=socket(AF_INET,SOCK_DGRAM,0))<0)
                printf("cliente:no puedo abrir socket");

  if(bind(rc_server.sockfd,(struct sockaddr *)&rc_server.dir_serv,sizeof(rc_server.dir_serv))<0)
  {
     printf("server rc: no se puede asociar la dir local\n");
     errorbind=1;
```

```
  }
  else
  {
     printf("socket abierto\n");
  }
  // Create a linux thread
  //p3dx_data.thread = rt_thread_create(P2OS_Main,
NULL, 10000);
  pthread_create(&(p3dx_data.thread), NULL,
P2OS_Main, NULL);

  // Bloqueo en espera del centro remoto

recvfrom(rc_server.sockfd,(char*)&rc_server.receive_data,si
zeof(comm_data_t),0,(struct sockaddr
*)&rc_server.dir_cli,&rc_server.long_dir_cli);  // NO_RT!

//    receive = rt_thread_create(receive_function, NULL,
10000);
  pthread_create(&receive, NULL,
receive_function,NULL);

#endif

 }
#endif /*  MDL_START */

static void mdlOutputs(SimStruct *S, int_T tid)
{

  real_T *y0   = ssGetOutputPortRealSignal(S,0);      /*
Velocidad Lineal */
  real_T *y1   = ssGetOutputPortRealSignal(S,1);      /*
Velocidad Angular */

  #if defined(RT)

  double sonar_enable;
  int i=0;
  int n=0;

     sem_wait(p3dx_data.sem);
     //rt_sem_wait(p3dx_data.sem); // LOCK
     // 2nd: read values from robot
     v_lin_local = (real_T)p2os_data_read.vel.vlin;     // m/s
     v_rot_local = (real_T)p2os_data_read.vel.vrot;     //
rad/s

          sem_post(p3dx_data.sem);
//     rt_sem_signal(p3dx_data.sem); // UNLOCK

 if(recibe==1) //ACTUALIZACIÓN
 {

     //printf(" Recepcion Numero %d Velocidad lineal %f
velocidad angular %f
\n",rc_server.receive_data.data_number,rc_server.receive_
data.lineal,rc_server.receive_data.angular);
     v_consigna=rc_server.receive_data.lineal;
     o_consigna=rc_server.receive_data.angular;

     if(v_consigna > V_SAT)
        v_consigna = V_SAT;
     else if(v_consigna < -V_SAT)
        v_consigna = -V_SAT;

     if(o_consigna > O_SAT)
       o_consigna = O_SAT;
     else if(o_consigna < -O_SAT)
        o_consigna = -O_SAT;

     if(sonar_enable > 0.0)
       sonar_enable = 1.0;
     else
        sonar_enable = 0.0;

     numero_envio=numero_envio+1;
     rc_server.send_data.lineal=v_lin_local;
     rc_server.send_data.angular=v_rot_local;
```

```
      rc_server.send_data.data_number=numero_envio;

      n=sendto(rc_server.sockfd,(char
*)&rc_server.send_data,sizeof(comm_data_t),0,(struct
sockaddr *)&rc_server.dir_cli,sizeof(rc_server.dir_cli));

      if(n<0)

        perror("dg_cli:error en el sendto\n");

      recibe=0;

      printf(" Envio Numero %d Velocidad lineal %f
velocidad angular %f
\n",rc_server.send_data.data_number,rc_server.send_data.l
ineal,rc_server.send_data.angular);

   }

        // 1st: write commands

//      rt_sem_wait(p3dx_data.sem); // LOCK

      sem_wait(p3dx_data.sem);


      p2os_cmd_write.lin_speed = v_consigna;  // m/s

      p2os_cmd_write.rot_speed = o_consigna;  // rad/s

      p2os_cmd_write.sonar_enable = sonar_enable;  // 1 / 0


     // rt_sem_signal(p3dx_data.sem); // UNLOCK

      sem_post(p3dx_data.sem);



      *y0 = (real_T)v_lin_local;

      *y1 = (real_T)v_rot_local;



      #else // NOT RT MODE


      // Write in output port of S-Function

      *y0 = (real_T)0.0;

      *y1 = (real_T)0.0;

      #endif



}



/* Function: mdlTerminate
=====================================================
====

 * Abstract:

 *    No termination needed, but we are required to have
this routine.

 */

static void mdlTerminate(SimStruct *S)

{

#if defined(RT)

//   int i;


  //P2OS_Terminate(); // Send signal to P2OS Driver


  // clean task


  //rt_task_delete(p3dx_data.task);


  P2OS_MainQuit();


/*  rt_spclose(RTCOM1);*/

  close(serial_port);


// rt_sem_delete(p3dx_data.sem);

  sem_destroy(p3dx_data.sem);



#endif


  return;

}



void * receive_function(void * ptr)

{


   int n=0;

   while(1)

   {



      n =
recvfrom(rc_server.sockfd,(char*)&rc_server.receive_data,si
zeof(comm_data_t),0,(struct sockaddr
*)&rc_server.dir_cli,&rc_server.long_dir_cli);  // NO_RT!

      recibe=1;

      // LOCK
```

```
      memcpy(&rc_server.storage_data[3],
&rc_server.storage_data[2], sizeof(comm_data_t));

      memcpy(&rc_server.storage_data[2],
&rc_server.storage_data[1], sizeof(comm_data_t));

      memcpy(&rc_server.storage_data[1],
&rc_server.storage_data[0], sizeof(comm_data_t));

      memcpy(&rc_server.storage_data[0],
&rc_server.receive_data, sizeof(comm_data_t));


      // UNLOCK


      if(n<0)

         printf("dg_cli: error en el recvfrom PREDECESOR");
// NO_RT!


      printf("DATO RECIBIDO Numero %d Velocidad lineal
%f velocidad angular %f
\n",rc_server.receive_data.data_number,rc_server.receive_
data.lineal,rc_server.receive_data.angular);


   }

}
////////////////////////////////////////////////


////////////////////////////////////////////////

////////////////////////////////////////////////

// Private Defines


#define PLAYER_NUM_ROBOT_TYPES 1

#define PACKET_LEN 256


/* Command numbers */

#define SYNC0 0

#define SYNC1 1

#define SYNC2 2


#define PULSE 0

#define OPEN 1

#define CLOSE 2

#define ENABLE 4

#define SETA 5

#define SETV 6

#define SETO 7

#define VEL 11

#define RVEL 21

#define SETRA 23

#define SONAR 28


#define STOP 29

#define VEL2 32


#define BUMP_STALL 44

#define JOYDRIVE 47

#define ROTKP 82

#define ROTKV 83


#define ROTKI 84

#define TRANSKP 85

#define TRANSKV 86

#define TRANSKI 87


#define P2OS_CYCLETIME_USEC 20000

#define P2OS_CYCLETIME_NSEC
(P2OS_CYCLETIME_USEC*1000)


#define PLAYER_POWER_MASK_VOLTS 1

#define PLAYER_POWER_MASK_WATTS 2

#define PLAYER_POWER_MASK_JOULES 4

#define PLAYER_POWER_MASK_PERCENT 8

#define P2OS_NOMINAL_VOLTAGE 12.0


#define MOTOR_MAX_LIN_SPEED 1000 // mm/s

#define MOTOR_MAX_ROT_SPEED 180 // deg/s


/* Argument types */

#define ARGINT  0x3B // Positive int (LSB, MSB)

#define ARGNINT  0x1B // Negative int (LSB, MSB)


#define PLAYER_ERR_ERR 0

#define PLAYER_ERR_WARN 1

#define PLAYER_ERR_MSG 2
```

```
#define PLAYER_WARN(msg) \
ErrorPrint(PLAYER_ERR_WARN, 0, __FILE__, __LINE__, \
"warning : " msg "\n")

#define PLAYER_WARN1(msg, a) \
ErrorPrint(PLAYER_ERR_WARN, 0, __FILE__, __LINE__, \
"warning : " msg "\n", a)

#define PLAYER_WARN2(msg, a, b) \
ErrorPrint(PLAYER_ERR_WARN, 0, __FILE__, __LINE__, \
"warning : " msg "\n", a, b)


// Convert degrees to radians
#ifndef DTOR
#define DTOR(d) ((d) * (M_PI) / 180.0)
#endif

// Convert radians to degrees
#ifndef RTOD
#define RTOD(r) (((double)(r) * 180.0) / M_PI)
#endif



////////////////////////////////////////////////////////////////

// Private Types

typedef struct
{
  unsigned char packet[PACKET_LEN];
  unsigned char size;
  //double timestamp;
} P2OSPacket_t;

typedef struct
{
  int AdjustNumSamplesFlag; //
  int AlignAngle; //
  int AlignSpeed; //
  double AngleConvFactor; //
  int AngleIncrement; //
  int CenterAwayCost; //
  const char* Class;
  int ClearOnFail; //
  int CollisionRange; //
  double DiffConvFactor; //
  double DiscardThreshold; //
  double DistConvFactor; //
  double DistanceWt; //
  int DrivingRotAccel; //
  int DrivingRotDecel; //
  int DrivingRotVelMax; //
  int DrivingTransAccel; //
  int DrivingTransDecel; //
  int DrivingTransNegVelMax; //
  int DrivingTransVelMax; //
  int EmergencyMaxTransDecel; //
  int FailedTh; //
  int FailedX; //
  int FailedY; //
  int FastSpeed; //
  int FrontBumpers; //
  int FrontClearance; //
  int FrontPaddingAtFastSpeed; //
  int FrontPaddingAtSlowSpeed; //
  int FuseAllSensors; //
  int GoalAngleTol; //
  int GoalDistanceTol; //
  int GoalOccupiedFailDistance; //
  int GoalRotAccel; //
  int GoalRotDecel; //
  int GoalRotSpeed; //
  int GoalSpeed; //
  double GoalSwitchTime; //
  int GoalTransAccel; //
  int GoalTransDecel; //
  int GoalUseEncoder; //
  int GridRes; //
  double GyroScaler; //
  int HasMoveCommand; //
  int HeadingRotAccel; //
  int HeadingRotDecel; //
  int HeadingRotSpeed; //
  double HeadingWt; //
  int Holonomic; //
  int IRNum; //
  int IRUnit; //
  int IdleTimeTriggerTh; //
```

int IdleTimeTriggerX; //

int IdleTimeTriggerY; //

double KDegPerDeg; //

double KDegPerMm; //

double KMmPerMm; //

int LaserFlipped; //

const char* LaserIgnore;

const char* LaserPort;

int LaserPossessed; //

int LaserPowerControlled; //

int LaserTh; //

int LaserX; //

int LaserY; //

int LocalPathFailDistance; //

const char* Map;

double MarkOldPathFactor; //

int MaxRVelocity; //

int MaxRotSpeed; //

int MaxSpeed; //

int MaxVelocity; //

int MinNumSamples; //

int NewTableSensingIR; //

int NforLinVelIncrements; //

int NforRotVelIncrements; //

int NumFrontBumpers; //

int NumRearBumpers; //

int NumSamples; //

int NumSamplesAngleFactor; //

int NumSplinePoints; //

double ObsThreshold; //

double OccThreshold; //

int OneWayCost; //

double PassThreshold; //

double PeakFactor; //

int PeakStdTh; //

int PeakStdX; //

int PeakStdY; //

int PeturbTh; //

int PeturbX; //

int PeturbY; //

int PlanEverytime; //

double PlanFreeSpace; //

double PlanRes; //

int Qtt; //

int Qxx; //

int Qyy; //

double RangeConvFactor; //

int RearBumpers; //

int RecoverOnFail; //

int ReflectorMatchDist; //

int ReflectorMaxAngle; //

int ReflectorMaxRange; //

int ReflectorVariance; //

int RequestEncoderPackets; //

int RequestIOPackets; //

int Resistance; //

int RobotDiagonal; //

int RobotLength; //

int RobotLengthFront; //

int RobotLengthRear; //

int RobotRadius; //

int RobotWidth; //

int RotAccel; //

int RotDecel; //

int RotVelMax; //

int SecsToFail; //

double SensorBelief; //

int SettableAccsDecs; //

int SettableVelMaxes; //

int SideClearanceAtFastSpeed; //

int SideClearanceAtSlowSpeed; //

int SlowSpeed; //

int SmoothWindow; //

int SonarAngleRes; //

int SonarAperture; //

double SonarBetaMax; //

double SonarBetaMin; //

int SonarIncidenceLimit; //

double SonarLambdaF; //

double SonarLambdaR; //

int SonarMaxRange; //

int SonarMinLineSize; //

```
  int SonarNum; //
  int SonarRangeRes; //
  int SonarSigma; //
  int SplineDegree; //
  int StallRecoverDuration; //
  int StallRecoverRotation; //
  int StallRecoverSpeed; //
  int StdTh; //
  int StdX; //
  int StdY; //
  const char* Subclass;
  int SuperMaxTransDecel; //
  int SwitchToBaudRate; //
  int TableSensingIR; //
  int TransAccel; //
  int TransDecel; //
  int TransVelMax; //
  int Triangulate; //
  double TriangulateScoreThreshold; //
  int TriggerAngle; //
  int TriggerDistance; //
  int TriggerTime; //
  int TriggerTimeEnabled; //
  int UseCollisionRangeForPlanning; //
  int UseEStop; //
  int UseLaser; //
  int UseReflectorCenters; //
  int UseSonar; //
  int Vel2Divisor; //
  int VelConvFactor; //
  double VelocityWt; //
} RobotParams_t;

RobotParams_t p3dx_sh_params =
{
  0,
  0,
  0,
  0.001534,
  0,
  0,
  "Pioneer",
  0,
  0,
  0.00606, // NOTE it was changed (BUG: Player & Stage should change it too!);
  0,
  1,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  1.626,
  1,
  0,
  0,
```

0,
0,
1,
0,
0,
0,
0,
0,
0,
0,
0,
0,
"",
"COM3",
0,
1,
0,
18,
0,
0,
"0",
0,
500,
0,
0,
2200,
0,
0,
0,
0,
5,
5,
0,
0,
0,
0,
0,
0,
0,
0,
0

```
0,
0,
0,
0,
0,
0,
16,
0,
0,
0,
0,
0,
0,
0,
0,
"p3dx-sh",
0,
38400,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
20,
1,
0
};

////////////////////////////////////////////////////
```

```
// Private Variables

char ignore_checksum;

int param_idx; // index of our robot's data in the parameter table

int motor_max_trans_accel = MAX_ACCEL;

int motor_max_trans_decel = -MAX_ACCEL;

int motor_max_rot_accel = MAX_ACCEL;

int motor_max_rot_decel = -MAX_ACCEL;

P2OSPacket_t tx_packet;

P2OSPacket_t rx_packet;

RobotParams_t PlayerRobotParams[PLAYER_NUM_ROBOT_TYPES];

volatile int p2os_run = 1;

p2os_data_t p2os_data_read;

p2os_cmd_t p2os_cmd_write;

////////////////////////////////////////////////////

// Private Functions

int P2OS_MainSetup(void);

void P2OS_MainProcess(void);

void P2OS_MainQuit(void);

int  P2OSPacket_CalcChkSum(P2OSPacket_t * pkt);

void P2OSPacket_Print(P2OSPacket_t * pkt);

void P2OSPacket_PrintHex(P2OSPacket_t * pkt);

int  P2OSPacket_Build(P2OSPacket_t * pkt, unsigned char *data, unsigned char datasize );

int  P2OSPacket_Send(P2OSPacket_t * pkt);

int  P2OSPacket_Receive_timeout(P2OSPacket_t * pkt, char ignore_checksum , int timeout);

int  P2OSPacket_Check(P2OSPacket_t * pkt, int ignore_checksum);

int  P2OS_SendReceive(P2OSPacket_t * tx_pkt, P2OSPacket_t * rx_pkt, char publish_data);

void P2OS_ToggleSonarPower(unsigned char val);
```

```
////////////////////////////////////////////////////////

void P2OS_Terminate(void)
{/*
  rt_sem_wait(sem_p2os_exit);
  p2os_run = 0;
  rt_sem_signal(sem_p2os_exit);
*/
}

void * P2OS_Main(void *devicep)
{
   int temp;

   /*p3dx_data.task =
rt_task_init_schmod(nam2num("ARCOS_DRV"), 0, 0, 0,
SCHED_FIFO, 0xFF);
   mlockall(MCL_CURRENT | MCL_FUTURE);*/
   // makes task hard real time (default: soft)
   // uncomment the next line when developing : develop in
soft real time mode

/*#ifdef RT_HARD
   rt_make_hard_real_time();
#endif*/

  // fflush serial port buffers
 tcflush(serial_port,TCIOFLUSH);

/*  rt_spclear_rx(RTCOM1);*/
/*  rt_spclear_tx(RTCOM1);*/

 if(P2OS_MainSetup())
   goto exit_task;

 P2OS_MainProcess();

   exit_task:

  // makes task soft real time
 //  rt_make_soft_real_time();

  return 0;

//  pthread_exit(NULL);

}

int P2OS_MainSetup()
{

 int i;

 int bauds = DEFAULT_BAUD;
 int numbauds = sizeof bauds / sizeof(int);
 int currbaud = 0;

 struct termios term;
 unsigned char command;
 unsigned char command_s[5];

 int flags=0;
 char sent_close = 0;

 char name[20], type[20], subtype[20];
 int cnt;
 int num_sync_attempts;

 enum
 {
  NO_SYNC,
  AFTER_FIRST_SYNC,
  AFTER_SECOND_SYNC,
  READY
 } psos_state;


 psos_state = NO_SYNC;
```

```
PlayerRobotParams[0] =  p3dx_sh_params;

//rt_sem_wait(p3dx_data.sem);
sem_wait(p3dx_data.sem);
p2os_cmd_write.lin_speed = 0;
p2os_cmd_write.rot_speed = 0;
//rt_sem_signal(p3dx_data.sem);
sem_post(p3dx_data.sem);

// Sync:

num_sync_attempts = 0;

while(psos_state != READY)
{

 if(p2os_run==0)
  return 1;

 switch(psos_state)
 {
  case NO_SYNC:

   command = SYNC0;
   P2OSPacket_Build(&tx_packet, &command, 1);
   P2OSPacket_Send(&tx_packet);

    usleep(P2OS_CYCLETIME_USEC);
//rt_sleep(nano2count(P2OS_CYCLETIME_NSEC));

   break;
  case AFTER_FIRST_SYNC:

   command = SYNC1;
   P2OSPacket_Build(&tx_packet, &command, 1);
   P2OSPacket_Send(&tx_packet);
   break;
  case AFTER_SECOND_SYNC:
   command = SYNC2;
   P2OSPacket_Build(&tx_packet, &command, 1);
   P2OSPacket_Send(&tx_packet);
   break;
  default:
   break;
 }

 usleep(P2OS_CYCLETIME_USEC);
//rt_sleep(nano2count(P2OS_CYCLETIME_NSEC));

 if(P2OSPacket_Receive_timeout(&rx_packet,
ignore_checksum, 200E6)) // Wait 200 mS
 {
  if((psos_state == NO_SYNC) && (num_sync_attempts <
20))
  {
   num_sync_attempts++;

   continue;
  }
  else
  {
   printf("Robot doesn't work: %d
attemps\n",num_sync_attempts);
   return(1);

  }
 }

 switch(rx_packet.packet[3])
 {
  case SYNC0:
   psos_state = AFTER_FIRST_SYNC;
   break;
  case SYNC1:
   psos_state = AFTER_SECOND_SYNC;
   break;
  case SYNC2:
   psos_state = READY;

   break;
  default:
   // maybe P2OS is still running from last time.  let's try
to CLOSE
```

```
      // and reconnect
      if(!sent_close)
      {
        command = CLOSE;
        P2OSPacket_Build(&tx_packet, &command, 1);
        P2OSPacket_Send(&tx_packet);

        sent_close = 1;

//rt_sleep(nano2count(2*P2OS_CYCLETIME_NSEC));
        usleep(2*P2OS_CYCLETIME_USEC);
      }
      break;
    }

    usleep(P2OS_CYCLETIME_USEC);
    //rt_sleep(nano2count(P2OS_CYCLETIME_NSEC));

  } // while(psos_state != READY)

  if(psos_state != READY)
  {
    #ifdef DEBUG_ARCOS
    printf("Couldn't synchronize with P2OS\n");
    #endif
    return(1);
  }

  cnt = 4;
  cnt += snprintf(name, sizeof(name), "%s",
&rx_packet.packet[cnt]);
  cnt++;
  cnt += snprintf(type, sizeof(type), "%s",
&rx_packet.packet[cnt]);
  cnt++;
  cnt += snprintf(subtype, sizeof(subtype), "%s",
&rx_packet.packet[cnt]);
  cnt++;

  command = OPEN;
  P2OSPacket_Build(&tx_packet, &command, 1);
  P2OSPacket_Send(&tx_packet);
  //rt_sleep(nano2count(P2OS_CYCLETIME_NSEC));
  usleep(P2OS_CYCLETIME_USEC);

  command = PULSE;
  P2OSPacket_Build(&tx_packet, &command, 1);
  P2OSPacket_Send(&tx_packet);
  //rt_sleep(nano2count(P2OS_CYCLETIME_NSEC));
  usleep(P2OS_CYCLETIME_USEC);

  #ifdef DEBUG_ARCOS
  printf("Done.\n   Connected to %s, a %s %s\n", name,
type, subtype);
  #endif

  // now, based on robot type, find the right set of
parameters
  for(i=0;i<PLAYER_NUM_ROBOT_TYPES;i++)
  {
    if(!strcasecmp(PlayerRobotParams[i].Class,type) &&
!strcasecmp(PlayerRobotParams[i].Subclass,subtype))
    {
      param_idx = i;
      break;
    }
  }
  if(i == PLAYER_NUM_ROBOT_TYPES)
  {
    #ifdef DEBUG_ARCOS
    fputs("P2OS: Warning: couldn't find parameters for this
robot; "
          "using defaults\n",stderr);
    #endif
    param_idx = 0;
  }
  // first, receive a packet so we know we're connected.

  // turn off the sonars at first
  P2OS_ToggleSonarPower(0);

  // Set max accel/decel limits
  if(motor_max_trans_accel > 0)
  {
    command_s[0] = SETA;
    command_s[1] = ARGINT;
```

```
  command_s[2] = motor_max_trans_accel & 0x00FF;
  command_s[3] = (motor_max_trans_accel & 0xFF00) >>
8;
  P2OSPacket_Build(&tx_packet, command_s, 4);
  P2OS_SendReceive(&tx_packet, &rx_packet,0);
 }

 if(motor_max_trans_decel < 0)
 {
  command_s[0] = SETA;

  command_s[1] = ARGNINT;
  command_s[2] = abs(motor_max_trans_decel) & 0x00FF;
  command_s[3] = (abs(motor_max_trans_decel) & 0xFF00)
>> 8;
  P2OSPacket_Build(&tx_packet, command_s, 4);
  P2OS_SendReceive(&tx_packet,&rx_packet,0);
 }
 if(motor_max_rot_accel > 0)
 {
  command_s[0] = SETRA;
  command_s[1] = ARGINT;
  command_s[2] = motor_max_rot_accel & 0x00FF;
  command_s[3] = (motor_max_rot_accel & 0xFF00) >> 8;
  P2OSPacket_Build(&tx_packet, command_s, 4);
  P2OS_SendReceive(&tx_packet,&rx_packet,0);
 }
 if(motor_max_rot_decel < 0)
 {
  command_s[0] = SETRA;
  command_s[1] = ARGNINT;
  command_s[2] = abs(motor_max_rot_decel) & 0x00FF;
  command_s[3] = (abs(motor_max_rot_decel) & 0xFF00)
>> 8;
  P2OSPacket_Build(&tx_packet, command_s, 4);
  P2OS_SendReceive(&tx_packet,&rx_packet,0);
 }

/* FUTURE USE

 // if requested, change PID settings
 P2OSPacket pid_packet;
 unsigned char pid_command[4];
 if(charrot_kp >= 0)
 {
  pid_command[0] = ROTKP;
  pid_command[1] = ARGINT;
  pid_command[2] = charrot_kp & 0x00FF;
  pid_command[3] = (charrot_kp & 0xFF00) >> 8;
  pid_packet.Build(pid_command, 4);
  charSendReceive(&pid_packet);
 }
 if(charrot_kv >= 0)
 {
  pid_command[0] = ROTKV;
  pid_command[1] = ARGINT;
  pid_command[2] = charrot_kv & 0x00FF;
  pid_command[3] = (charrot_kv & 0xFF00) >> 8;
  pid_packet.Build(pid_command, 4);
  charSendReceive(&pid_packet);
 }
 if(charrot_ki >= 0)
 {
  pid_command[0] = ROTKI;
  pid_command[1] = ARGINT;
  pid_command[2] = charrot_ki & 0x00FF;
  pid_command[3] = (charrot_ki & 0xFF00) >> 8;
  pid_packet.Build(pid_command, 4);
  charSendReceive(&pid_packet);
 }
 if(chartrans_kp >= 0)
 {
  pid_command[0] = TRANSKP;
  pid_command[1] = ARGINT;
  pid_command[2] = chartrans_kp & 0x00FF;
  pid_command[3] = (chartrans_kp & 0xFF00) >> 8;
  pid_packet.Build(pid_command, 4);
  charSendReceive(&pid_packet);
 }
 if(chartrans_kv >= 0)
 {
  pid_command[0] = TRANSKV;
  pid_command[1] = ARGINT;
```

```
    pid_command[2] = chartrans_kv & 0x00FF;
    pid_command[3] = (chartrans_kv & 0xFF00) >> 8;
    pid_packet.Build(pid_command, 4);
    charSendReceive(&pid_packet);
  }
  if(chartrans_ki >= 0)
  {
    pid_command[0] = TRANSKI;
    pid_command[1] = ARGINT;
    pid_command[2] = chartrans_ki & 0x00FF;
    pid_command[3] = (chartrans_ki & 0xFF00) >> 8;
    pid_packet.Build(pid_command, 4);
    charSendReceive(&pid_packet);
  }
*/

  printf("Robot READY\n");

  return(0);
}



void P2OS_MainQuit()
{
  unsigned char command;
  int count;

  for(count = 0; count < 3; count++){
    usleep(P2OS_CYCLETIME_USEC);
//rt_sleep(nano2count(P2OS_CYCLETIME_NSEC));
    command = STOP;
    P2OSPacket_Build(&tx_packet, &command, 1);
    P2OSPacket_Send(&tx_packet);
  }

  for(count = 0; count < 3; count++){
    usleep(P2OS_CYCLETIME_USEC);
//rt_sleep(nano2count(P2OS_CYCLETIME_NSEC));
    command = CLOSE;
    P2OSPacket_Build(&tx_packet, &command, 1);
    P2OSPacket_Send(&tx_packet);
  }
  usleep(P2OS_CYCLETIME_USEC);
//rt_sleep(nano2count(P2OS_CYCLETIME_NSEC));

  printf("P2OS stopped\n");
}


void SIP_ParseStandard(unsigned char *buffer,
p2os_data_t * p2os_data)
{
  int status;
  int cnt = 0;
  unsigned char i;
  double v_left, v_right;
  unsigned char numSonars;
  unsigned char sonarIndex;

  status = buffer[cnt]; // TYPE
  cnt += sizeof(unsigned char);

  /*
   * Remember P2OS uses little endian:
   * for a 2 byte short (called integer on P2OS)
   * byte0 is low byte, byte1 is high byte
   * The following code is host-machine endian independant
   * Also we must or (|) bytes together instead of casting to a
   * short * since on ARM architectures short * must be even
byte aligned!
   * You can get away with this on a i386 since shorts * can
be
   * odd byte aligned. But on ARM, the last bit of the pointer
will be ignored!
   * The or'ing will work on either arch.
   */

  // XPOS

  cnt += sizeof(short);

  // YPOS

  cnt += sizeof(short);
```

```
 // THPOS

cnt += sizeof(short);

// L VEL
v_left = (short)(buffer[cnt] | (buffer[cnt+1] << 8));

cnt += sizeof(short);

// R VEL
v_right = (short)(buffer[cnt] | (buffer[cnt+1] << 8));

p2os_data->vel.vlin = ((v_left + v_right ) / 2) / 1e3;

 p2os_data->vel.vrot =  -((v_left - v_right) /
(2.0/PlayerRobotParams[param_idx].DiffConvFactor));

cnt += sizeof(short);

// BATTERY
 p2os_data->power.volts = ((double)buffer[cnt])/10.0;  //
Convert to volts

cnt += sizeof(unsigned char);

// STALL LEFT
p2os_data->stall = buffer[cnt] & 0x01;

cnt += sizeof(unsigned char);

// STALL RIGHT
p2os_data->stall |= (buffer[cnt] & 0x01)<<1;

//sip_packet->frontbumpers = buffer[cnt] >> 1;
cnt += sizeof(unsigned char);

// CONTROL
cnt += sizeof(short);

// FLAGS
cnt += sizeof(short);

// COMPASS
cnt += sizeof(unsigned char);

// SONAR COUNT
numSonars=buffer[cnt];
cnt+=sizeof(unsigned char);

if(numSonars>0)
{
  if(numSonars > MAX_NUM_SONAR) // Saturate number
of sonars
    numSonars = MAX_NUM_SONAR;

  for(i=0;i<(numSonars-1);i++)
  {
    sonarIndex=buffer[cnt];

    p2os_data->sonar.ranges[sonarIndex]=(unsigned int)
    ((((unsigned short)buffer[cnt+1] | ((unsigned
short)buffer[cnt+2] << 8)) )
    * PlayerRobotParams[param_idx].RangeConvFactor);

    cnt+=sizeof(unsigned char)+sizeof(unsigned short);
  }
 }

cnt += sizeof(short);

//analog = buffer[cnt];
cnt += sizeof(unsigned char);

//digin = buffer[cnt];
cnt += sizeof(unsigned char);

//digout = buffer[cnt];
cnt += sizeof(unsigned char);

}

/* send the packe t, then receive and parse an SIP */
```

```
int P2OS_SendReceive(P2OSPacket_t * tx_pkt,
P2OSPacket_t * rx_pkt, char publish_data)

{

  if(tx_pkt)

    P2OSPacket_Send(tx_pkt);


  /* receive a packet */

  if(P2OSPacket_Receive_timeout(rx_pkt,
ignore_checksum,100E6)) // 100 ms

  {


/*// HABILITAR ESTO PARA CONOCER LA
FRECUENCIA DE RX*/

/*#ifdef DEBUG_IO*/

/*    rt_spset_mcr(RTCOM2, RT_SP_RTS, 1); //
ERRORES*/

/*    rt_sleep(nano2count(5E6)); // 5 mS*/

/*    rt_spset_mcr(RTCOM2, RT_SP_RTS, 0);*/

/*#endif*/


    return 1;

  }


 //  printf("K");


/*// HABILITAR ESTO PARA CONOCER LA
FRECUENCIA DE RX*/

/*#ifdef DEBUG_IO*/

/*    rt_spset_mcr(RTCOM2, RT_SP_DTR, 1); // DATOS
CORRECTOS*/

/*    rt_sleep(nano2count(5E6)); // 5 mS*/

/*    rt_spset_mcr(RTCOM2, RT_SP_DTR, 0);*/

/*#endif*/


  if(rx_pkt->packet[0] == 0xFA && rx_pkt->packet[1] ==
0xFB &&

    (rx_pkt->packet[3] == 0x30 || rx_pkt->packet[3] ==
0x31 ||

    rx_pkt->packet[3] == 0x32 || rx_pkt->packet[3] ==
0x33 ||

    rx_pkt->packet[3] == 0x34))

  {


    /* It is a server packet, so process it */

//    rt_sem_wait(p3dx_data.sem);

    sem_wait(p3dx_data.sem);

    SIP_ParseStandard(&rx_pkt-
>packet[3],&p2os_data_read);

    //rt_sem_signal(p3dx_data.sem);

    sem_post(p3dx_data.sem);



  }

  else if(rx_pkt->packet[0] == 0xFA && rx_pkt->packet[1]
== 0xFB &&

      (rx_pkt->packet[3] == 0x20))

  {

   //CONFIGpac: do nothing

  }

   else

  {

   #ifdef DEBUG_ARCOS

   puts("Unknown packet: ");

   P2OSPacket_PrintHex(rx_pkt);

   #endif

  }


 return(0);

}



/* toggle sonars on/off, according to val */

void P2OS_ToggleSonarPower(unsigned char val)

{

 unsigned char command[4];


 command[0] = SONAR;

 command[1] = ARGINT;

 command[2] = val;

 command[3] = 0;

 P2OSPacket_Build(&tx_packet, command, 4);

 P2OS_SendReceive(&tx_packet,&rx_packet,0);

}


/* toggle motors on/off, according to val */

void P2OS_ToggleMotorPower(unsigned char val)
```

```
{
  unsigned char command[4];

  command[0] = ENABLE;
  command[1] = ARGINT;
  command[2] = val;
  command[3] = 0;
  P2OSPacket_Build(&tx_packet, command, 4);
  P2OS_SendReceive(&tx_packet,&rx_packet,0);
}


void P2OS_SendPulse (void)
{
  unsigned char command;
  command = PULSE;
  P2OSPacket_Build(&tx_packet,&command, 1);
  P2OS_SendReceive(&tx_packet,&rx_packet,0);
}

// Linear speed (m/s)
void P2OS_SendVEL(double vel)
{
  unsigned char command[4];
  unsigned short abs_vel;

  // Convert to mm/s
  short vel_int = (short) (vel*1e3);

  command[0] = VEL;

  if(vel_int >= 0)
    command[1] = ARGINT;
  else
    command[1] = ARGNINT;

  abs_vel = (unsigned short)abs(vel_int);

  if(abs_vel < MOTOR_MAX_LIN_SPEED)
  {
    command[2] = abs_vel & 0x00FF;
    command[3] = (abs_vel & 0xFF00) >> 8;
  }
  else
  {
    #ifdef DEBUG_ARCOS
    puts("Speed demand threshholded!\n");
    #endif
    command[2] = MOTOR_MAX_LIN_SPEED & 0x00FF;
    command[3] = (MOTOR_MAX_LIN_SPEED & 0xFF00)
>> 8;
  }

  P2OSPacket_Build(&tx_packet,command, 4);
  P2OS_SendReceive(&tx_packet,&rx_packet,0);
}

// rot speed (rad/s)
void P2OS_SendROT(double rot)
{
  unsigned char command[4];
  unsigned short abs_rot;
  short rot_deg;

  rot_deg = (short)RTOD(rot); // Convert Radians to Degress

  command[0] = RVEL;

  if(rot_deg >= 0)
    command[1] = ARGINT;
  else
    command[1] = ARGNINT;

  abs_rot = (unsigned short)abs(rot_deg);

  if(abs_rot < MOTOR_MAX_ROT_SPEED)
  {
    command[2] = abs_rot & 0x00FF;
    command[3] = (abs_rot & 0xFF00) >> 8;
  }
  else
  {
```

```
    #ifdef DEBUG_ARCOS
    puts("Turn rate demand threshholded!");
    #endif
    command[2] = MOTOR_MAX_ROT_SPEED & 0x00FF;
    command[3] = (MOTOR_MAX_ROT_SPEED & 0xFF00)
>> 8;
  }

  P2OSPacket_Build(&tx_packet,command, 4);
  P2OS_SendReceive(&tx_packet,&rx_packet,0);
}

void P2OS_MainProcess(void)
{
  double lin_speed; // m/s
  double rot_speed; // rad/seg
  int sonar_enable = 0; // Default off
  int change_sonar = 0;
  int a = 0;

  P2OS_ToggleMotorPower(1); // Default: motors enabled

   while(p2os_run) // Loop until exit
   {
    // Check exit

    if(p2os_cmd_write.sonar_enable ^ sonar_enable){
                sonar_enable ^= 1;
                change_sonar = 1;
     }

    if(change_sonar){
                P2OS_ToggleSonarPower(sonar_enable);
                change_sonar = 0;
    }

    //P2OS_SendPulse();

    // Update Linear speed
    //rt_sem_wait(p3dx_data.sem);
    sem_wait(p3dx_data.sem);
    lin_speed = p2os_cmd_write.lin_speed;
    //rt_sem_signal(p3dx_data.sem);
    sem_post(p3dx_data.sem);

    P2OS_SendVEL(lin_speed);  // R/W

    // Update rot speed
    //rt_sem_wait(p3dx_data.sem);
    sem_wait(p3dx_data.sem);
    rot_speed = p2os_cmd_write.rot_speed;
    //rt_sem_signal(p3dx_data.sem);
    sem_post(p3dx_data.sem);

    P2OS_SendROT(rot_speed);   // R/W

   } // while
}


void P2OSPacket_Print(P2OSPacket_t * pkt) {
  int i;
  if (pkt->packet) {
    printf("\"");
    for(i=0;i < pkt->size;i++) {
      printf("%u ", pkt->packet[i]);
    }
    puts("\"");
  }
}

void P2OSPacket_PrintHex(P2OSPacket_t * pkt) {
  int i;
  if (pkt->packet) {
    printf("\"");
    for(i=0;i < pkt->size ;i++) {
      printf("0x%.2x ", pkt->packet[i]);
    }
    puts("\"");
  }
}
```

```
int P2OSPacket_Check(P2OSPacket_t * pkt, int
ignore_checksum ) {

  unsigned short recv_chksum;

  unsigned short pkg_chksum;


  recv_chksum = (unsigned
short)(P2OSPacket_CalcChkSum(pkt) & 0xffff);

  pkg_chksum = ((unsigned short)(pkt->packet[pkt->size-2])
<< 8) | pkt->packet[pkt->size-1];


  if ( ignore_checksum )

  {

    return(1);

  }

  else

  {

    return recv_chksum == pkg_chksum;

  }

}


int P2OSPacket_CalcChkSum(P2OSPacket_t * pkt) {

  unsigned char *buffer = &pkt->packet[3];

  int c = 0;

  int n;


  n = pkt->size - 5;


  while (n > 1) {

    c+= (*(buffer)<<8) | *(buffer+1);

    c = c & 0xffff;

    n -= 2;

    buffer += 2;

  }

  if (n>0) c = c^ (int)*(buffer++);


  return(c);

}



int P2OSPacket_Receive_timeout(P2OSPacket_t * pkt, char
ignore_checksum, int timeout )

{

  unsigned char prefix[3];

  int cnt,temp;


  memset(pkt->packet,0,sizeof(pkt->packet));


  do

  {

    memset(prefix,0,sizeof(prefix));


    while(1)

    {

      cnt = 0;

//      printf("reading serial port\n");

      temp = read(serial_port, (void *)&prefix[2], 1);


      //temp = rt_spread_timed(RTCOM1, (void *)&prefix[2],
1, nano2count(timeout));


      //if ((temp == SEM_TIMOUT) || (temp < 0))

      if (temp < 0)

      {

            // Si no se recibe nada: salir

            //printf("T1 ");

        return(1);

      }


      if (prefix[0]==0xFA && prefix[1]==0xFB)

        break;


      prefix[0]=prefix[1];

      prefix[1]=prefix[2];


    } // while(1)


    pkt->size = prefix[2]+3;

    memcpy( pkt->packet, prefix, 3);


    cnt = 0;

    while( cnt!=prefix[2] )

    {

          temp = read(serial_port, (void *)&pkt-
>packet[3+cnt], prefix[2]-cnt);
```

```
    //temp = rt_spread_timed(RTCOM1, (void *)&pkt->packet[3+cnt], prefix[2]-cnt, nano2count(timeout));

/*     if ((temp == SEM_TIMOUT) || (temp < 0))*/
            if (temp < 0)
    {
      return(1);
    }

    cnt += prefix[2]-cnt;

  } // while(...)

 } while (!P2OSPacket_Check(pkt,ignore_checksum));

 return(0);
}

int P2OSPacket_Build(P2OSPacket_t * pkt, unsigned char *data, unsigned char datasize ) {
 unsigned short chksum;

 pkt->size = datasize + 5;

 /* header */
 pkt->packet[0]=0xFA;
 pkt->packet[1]=0xFB;

 if ( pkt->size > 198 ) {
  #ifdef DEBUG_ARCOS
  puts("Packet to P2OS can't be larger than 200 bytes");
  #endif
  return(1);
 }
 pkt->packet[2] = datasize + 2;

 memcpy( &pkt->packet[3], data, datasize );

 chksum = (unsigned short) (P2OSPacket_CalcChkSum(pkt) & 0xffff);

 pkt->packet[3+datasize] = chksum >> 8;
 pkt->packet[3+datasize+1] = chksum & 0xFF;

 if (!P2OSPacket_Check(pkt,0)) {
  #ifdef DEBUG_ARCOS
  puts("DAMN");
  #endif
  return(1);
 }
 pkt->size = datasize + 5;

 return(0);
}

int P2OSPacket_Send(P2OSPacket_t * pkt)
{
 int cnt=0,temp;
 temp = write(serial_port, (void *)pkt->packet, pkt->size );
 //temp = rt_spwrite_timed(RTCOM1, (void *)pkt->packet, pkt->size, DELAY_FOREVER);

 if (temp < 0)
 {
  #ifdef DEBUG_ARCOS
  perror("Send");
  #endif
  return(1);
 }

 return(0);
}

#ifdef  MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

ROBOT_DR.H

```
#ifndef _P2OSDRV_H
#define _P2OSDRV_H


#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>
#include <signal.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <stddef.h>
#include <string.h>
#include <strings.h>
#include <math.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <termios.h>
#include <sys/socket.h>
#include <netinet/tcp.h>
#include <netdb.h>
#include <limits.h>
#include <pthread.h>
#include <stdint.h>


//#include <rtai_lxrt.h>
//#include <rtai_sem.h>


// Llamada de RT Serial (debug IO)
//#define DEBUG_IO

//#define RT_HARD // Enable Hard Real Time Task
//#define DEBUG_ARCOS

// #define DEBUG_ARCOS // Enable to DEBUG
#define RTCOM1  0 // RT Serial

#define DEFAULT_BAUD B115200
#define MAX_NUM_SONAR 8

#ifndef Ts
#define Ts 0.01
#endif

#ifndef V_SAT
#define V_SAT  1.0
#endif

#ifndef O_SAT
#define O_SAT  3.0
#endif

#ifndef M_PI
   #define M_PI 3.14159265358979323846
#endif


// Configuración de aceleración máxima
#define MAX_ACCEL 2000

#define FILTER_CONSTANT (0.5)

#define ENABLE_FILTER

//Server Remote Center
#define PORT_UDP_COVE0      6500
#define DIR_UDP_COVE0       "172.29.28.50"

#define PORT_UDP_COVE1      6500
#define DIR_UDP_COVE1       "172.29.28.51"

#define PORT_UDP_COVE2      6500
#define DIR_UDP_COVE2   "172.29.28.52"

#define PORT_UDP_COVE3      6500
#define DIR_UDP_COVE3       "172.29.28.53"

#define PORT_UDP_COVE4  6500
#define DIR_UDP_COVE4   "172.29.28.54"
```

```
#define PORT_UDP_COVE5  6500
#define DIR_UDP_COVE5   "192.168.11.61"



typedef struct
{
 int data_number;
 float lineal;
 float angular;
}comm_data_t;



//SERVER STRUCT
typedef struct
{
 int sockfd;
 struct sockaddr_in dir_cli;
 int long_dir_cli;
 struct sockaddr_in dir_serv;
 comm_data_t send_data;
 comm_data_t receive_data;
 comm_data_t storage_data[4];
}server_t;




// Readings from a robot's sonars.
typedef struct sonar_data
{
  unsigned int ranges[MAX_NUM_SONAR];     // The range
readings [mm]
} sonar_data_t;

// The power interface data in this format.
typedef struct power_data
{
 float volts; // Battery voltage [V]
} power_data_t;

typedef struct vel2d
{
 double vlin; // m/s
 //double vlin_acc; // m/s
 double vrot; // rad/s
 //double vrot_acc; // rad/s
} vel2d_t;

typedef struct p2os_data
{
 vel2d_t vel; //  velocities [m/s,rad/s]
 char stall;  // Are the motors stalled?
 sonar_data_t sonar;
 power_data_t power;
} p2os_data_t;

typedef struct p2os_cmd
{
 double lin_speed; // m/s
 double rot_speed; // rad/seg
 int sonar_enable;
} p2os_cmd_t;

 // Variable to read data and write commands
extern p2os_data_t p2os_data_read;
extern p2os_cmd_t p2os_cmd_write;

// Mutex to protect variable access
//extern pthread_mutex_t p2os_mutex;
//extern SEM *sem_p2os;



/**
 Main Program
 Interface to P3-DX robot
 (Non-returning function)
*/
void * P2OS_Main(void *devicep);
/// This function tell to interface that must finish
void P2OS_Terminate(void);

#endif
```

III.6. ANEXO VI: Guía de Instalación de ANYDESK

Guía de Instalación AnyDesk con MacOS

1. Acceder a la página oficial del Software AnyDesk https://anydesk.com/es desde el navegador donde nos encontraremos lo siguiente:

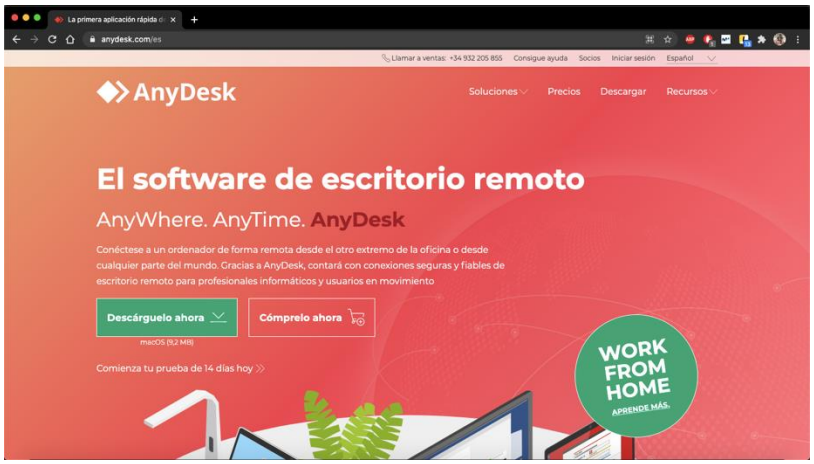


Figura 35: Página principal Anydesk

2. Clicamos en *Descárguelo Ahora* donde ya nos marca el sistema operativo que estamos utilizando y nos aparecerá donde queremos guardar el archivo de descarga.

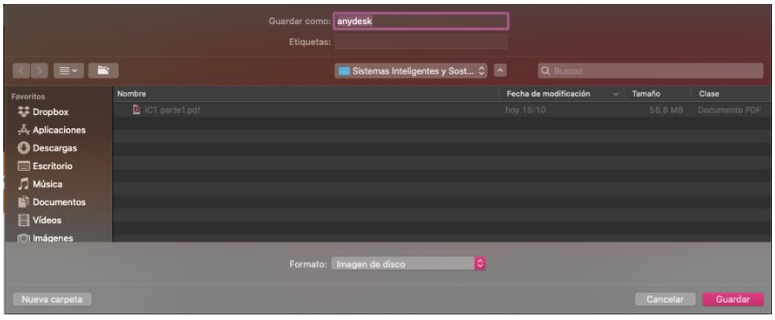


Figura 36: Descarga del fichero para la instalación

3. Guardamos en el archivo en el directorio deseado donde se descargará él .dmg y lo abrimos.

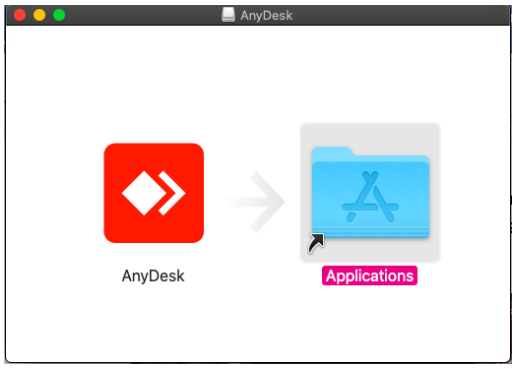


Figura 37: Apertura del archivo .dmg

4. Arrastramos el icono de AnyDesk a la carpeta de Aplicaciones donde ya tendríamos el programa listo para usar.

5. Abrimos el programa donde accederemos al menú principal del Software

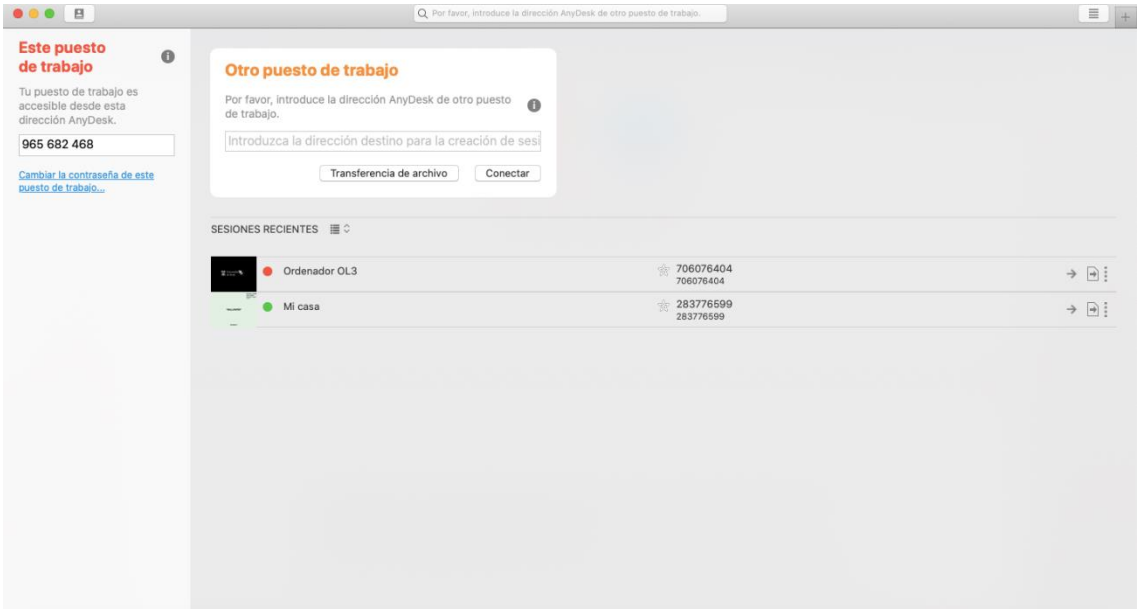


Figura 38: Pantalla Principal AnyDesk

Guía de Instalación AnyDesk con Windows

1. Acceder a la página oficial del Software AnyDesk https://anydesk.com/es desde el navegador donde nos encontraremos lo siguiente:

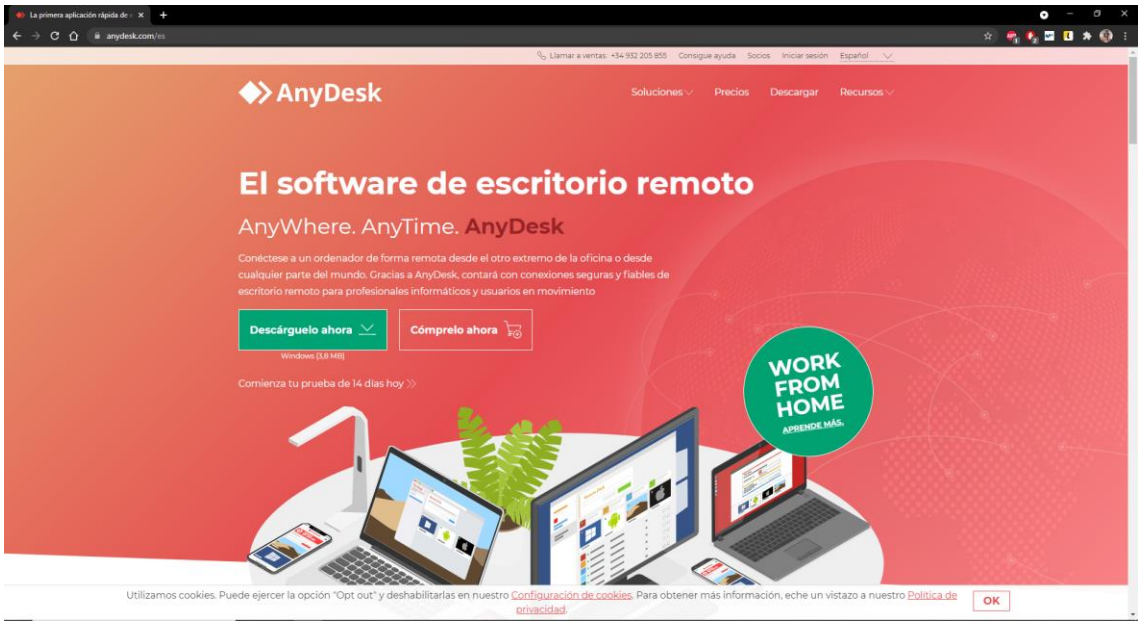


Figura 39: Página principal Anydesk

2. Clicamos en *Descárguelo Ahora* donde ya nos marca el sistema operativo que estamos utilizando y nos aparecerá donde queremos guardar el archivo de descarga.

3. Guardamos en el archivo en el directorio deseado donde se descargará él .exe y lo ejecutamos.

4. Ejecutamos el archivo ejecutable para completar la instalación de AnyDesk.

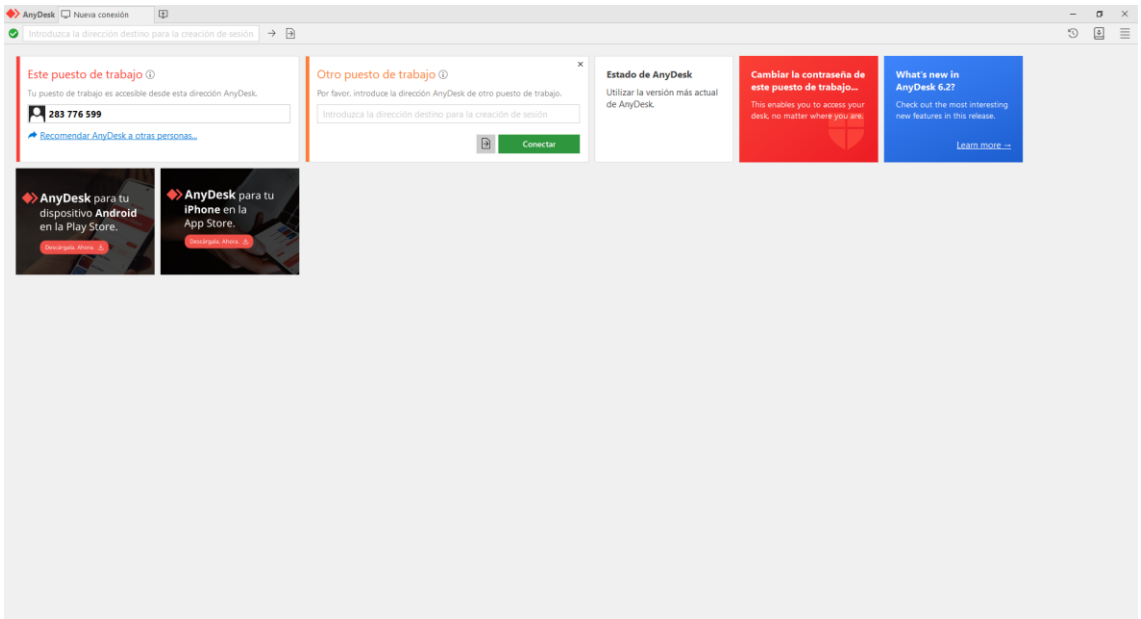
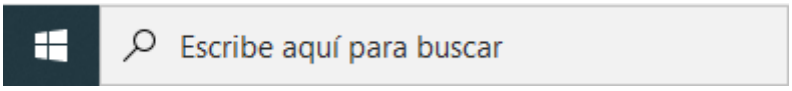


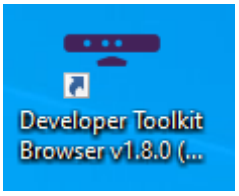
Figura 40: Página principal AnyDesk en Windows

III.7. ANEXO VII: Pasos para la conexión de Audio y Video con la cámara Kinect

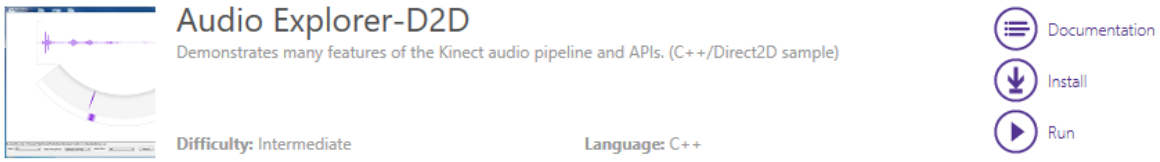
AUDIO

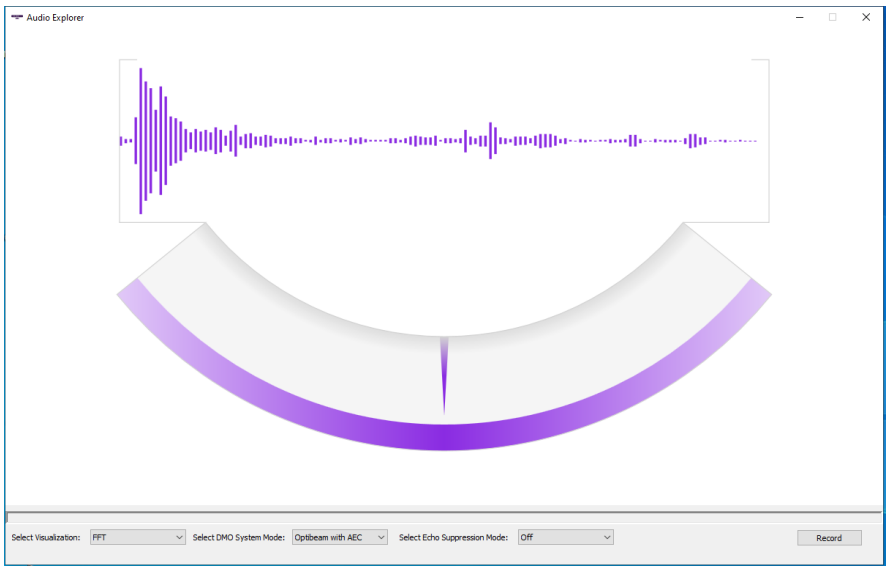
1. Buscador inicio Windows (Abajo a la izquierda), buscar la aplicación DEVELOPER TOOLKIT BROWSER V1.8.0 (Kinect for Windows).





2. En el apartado de ALL de la barra superior, buscar la aplicación AUDIO EXPLORER - D2D, donde se puede comprobar por señal de audio el correcto funcionamiento del sistema. Para acceder clicamos en RUN y se abrirá un sistema donde puedes observar la señal de audio de la dirección donde procede. Se puede visualizar la señal tanto en osciloscopio como para Transformada Rápida de Fourier FFT.

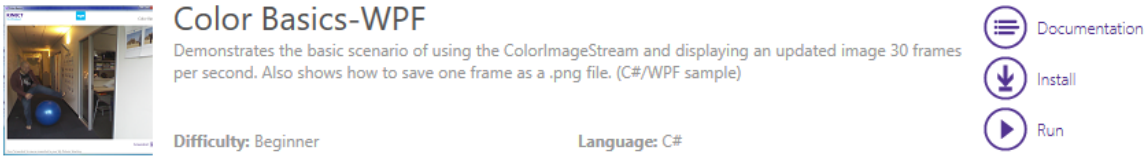


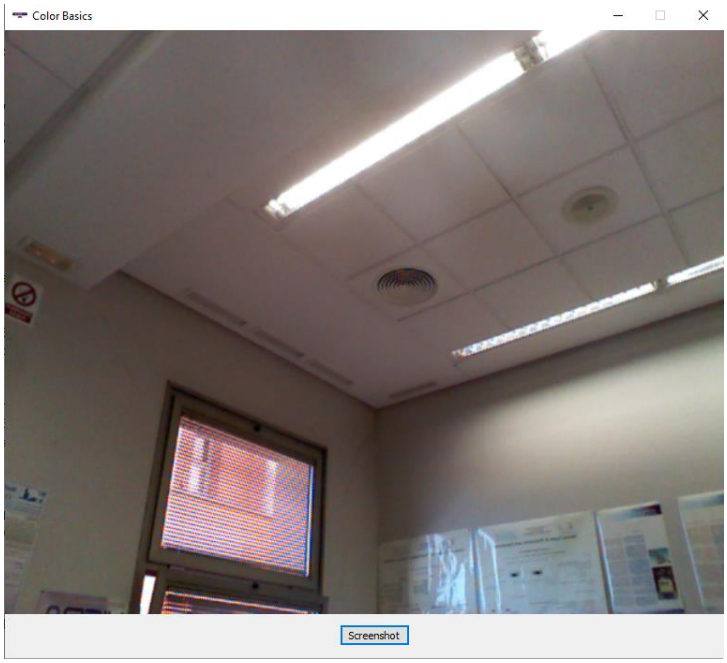


VIDEO

1. Si tenemos alguna otra aplicación abierta la cerramos dejando abierto el DEVELOPER TOOLKIT BROWSER V1.8.0 (Kinect for Windows).

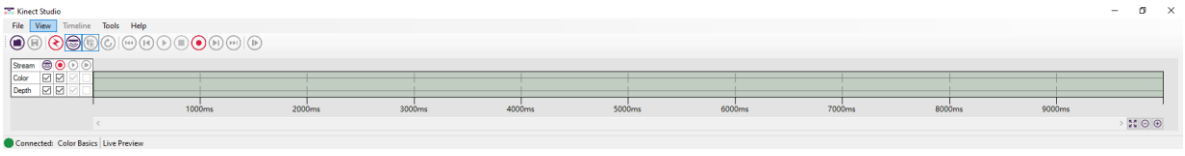
2. Buscamos dentro del apartado ALL de la barra superior COLOR BASICS - D2D. Al darle a RUN, aparecerá una pantalla donde se puede ver y comprobar el robot.





3. Para grabar la comprobación visual del funcionamiento remoto, sin cerrar COLOR BASICS - D2D, dentro de DEVELOPER TOOLKIT BROWSER V1.8.0 (Kinect for Windows), buscamos dentro del apartado de ALL en la barra superior, KINECT STUDIO, o la otra opción es buscarlo en el inicio de Windows (Abajo a la izquierda), con el mismo nombre.

4. Al abrir KINECT STUDIO, se abrirá una pantalla con una línea de tiempo y otra donde nos dirá que aplicación esta abierta, en este caso será COLOR BASICS - D2D que es la última que está abierta.



5. Clicamos en CONECT.

6. Si necesitamos ver la cámara a otro tamaño, dentro de la pantalla que queda (la de la línea de tiempo), en la pestaña VIEW, podemos usar las 3 opciones:

    - La parte de COLOR, es la cámara normal, similar a la abierta en COLOR BASICS - D2D.
    - Las otras dos son opciones para otro tipo de aplicaciones KINECT más avanzadas.

7. Para grabar, en la línea temporal (KINECT ESTUDIO), pulsamos en RECORD, es decir, los dos círculos concéntricos que estarán en rojo.

8. Para parar la grabación pulsamos STOP, dentro de los botones de la linea temporal.

9. En la barra se generará el video en cuestión, donde sobre ella podemos verlo repetido, frame a frame o incluso, repetir otra toma.

10. Para guardar el video grabado, clicamos en el botón del disquete, donde se nos abrirá una pestaña nueva para decidir donde lo guardamos. El formato con el que se guardara será .xed (Xbox Edition Develope).

11. El archivo grabado esta en Este equipo -> Documentos -> Kinect Studio -> Repository.

Universidad de Alcalá
Escuela Politécnica Superior

ESCUELA POLITECNICA
SUPERIOR

Universidad
de Alcalá