UNIVERSIDAD POLITÉCNICA DE MADRID

DEPARTAMENTO DE AUTOMÁTICA, INGENIERÍA ELECTRÓNICA E INFORMÁTICA INDUSTRIAL

División de Ingeniería de Sistemas y Automática (DISAM)

Título: "Sistema de control para el seguimiento de trayectorias de un UGV no holonómico tipo Ackermann"

Autor: José Luis Sánchez López

Tutor: Pascual Campoy Cervera

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS INDUSTRIALES
C/ José Gutiérrez Abascal, 2
28006 Madrid

Teléfono: 91 336 30 61
Fax: 91 336 30 10

INDICE

INDICE DE ILUSTRACIONES

INDICE DE TABLAS

1. INTRODUCCIÓN

El proyecto en el que se ha trabajado, tiene como principal aplicación la automatización de la conducción consiguiendo desplazamientos más eficientes, seguros y por supuesto más cómodos para el usuario.

1.1.Motivación

Las principales motivaciones para llevar a cabo la automatización de la conducción son, como se ha mencionado antes, la seguridad, la eficiencia, y por supuesto la comodidad para el usuario.

Según la Organización Mundial de la Salud (OMS), los siniestros de tráfico provocan anualmente la pérdida de 1.200.000 vidas (dato en el año 2005), y que si no se pone remedio, constituirán en 2020 la tercera causa global de lesiones y de muerte en el mundo [ver 23]. Según la Dirección General de Tráfico (DGT) [24], en el año 2003, los accidentes de tráfico por distracción del conductor supusieron el 24.9% de los accidentes de tráfico, factor más frecuente. Los tres siguientes factores más frecuentes fueron velocidad inadecuada, maniobra antirreglamentaria e invasión de la izquierda de la calzada, representando el 20.3%, el 13.9% y el 11.6%. Sólo estos cuatro factores, suponen el 70.7% de los accidentes; constituyendo todos ellos errores humanos. La tarea de la automatización de la conducción podría evitar todos los accidentes debidos a los errores humanos, pudiéndose evitar muchas muertes.

Por otro lado, el tiempo de reacción de un ser humano es de media de 0.2 segundos, aunque depende de muchos factores tales como el cansancio, alcohol, drogas, edad, entrenamiento,… Sin embargo, el tiempo de reacción de la electrónica del coche, es de unos pocos milisegundos (como máximo 0.04 segundos). Este hecho de responder más de 5 veces más rápido, permite a los coches guiados autónomamente, mantener una distancia de seguridad con respecto al vehículo de delante mucho menor que los vehículos conducidos por un humano. Esto supondría un mayor empaquetamiento de los vehículos a lo largo de la carretera, consiguiéndose con ello un uso más eficiente de las carreteras, disminuyéndose el tráfico y por ende el tiempo de desplazamiento [ver 1]. Por otra parte, esta disminución del tiempo de desplazamiento aporta una ventaja de menor uso de combustible y las ventajas que lleva asociadas: menor contaminación medioambiental y ahorro económico de combustible.

Si comentamos ahora los aspectos sociales de por qué se deben automatizar la conducción, encontramos multitud de ventajas. Se pueden citar algunas de ellas, como son el posible uso de vehículos por personas discapacitadas, la posibilidad de realizar otras tareas como leer o utilizar un ordenador en el vehículo sin correrse ningún riesgo,…; incluso se obtienen mejoras de conciliación de la vida familiar al emplearse menos tiempo en los desplazamientos, ya que según [25], el 64% de los trabajadores utiliza el coche para ir a trabajar en España, estando el tráfico de las ciudades altamente congestionado.

1.2.Antecedentes de los automóviles autónomos

El desarrollo de los circuitos integrados (IC) y más tarde, de los microprocesadores, fueron factores muy importantes en el desarrollo del control electrónico en los automóviles. La importancia de los microprocesadores es tal que es el "cerebro" que controla muchos sistemas en los automóviles de hoy. Por ejemplo, en un sistema de control de crucero, el controlador ajusta la velocidad deseada y permite que el sistema pulsando un botón. Un microprocesador controla entonces la velocidad real del vehículo a partir de datos de los sensores de velocidad. La velocidad real se compara con la velocidad deseada y el controlador ajusta el acelerador cuando sea necesario.

Un vehículo totalmente autónomo es aquel en el que un equipo realiza todas las tareas que el conductor humano normalmente haría. En última instancia, esto significaría conseguir que una vez dentro de un coche, se introdujese el destino en una computadora y, a partir de ese instante, el coche sería capaz de llegar al destino sin intervención humana. El coche sería capaz de percibir su entorno y realizar cambios de dirección y de velocidad según sea necesario.

Esto requeriría de todas las tecnologías de automatización del automóvil que se mencionan a continuación: detección de carril para ayudar a superar vehículos más lentos o salir de una carretera, detección de obstáculos para localizar a otros vehículos, peatones, animales, etc., control de crucero adaptativo para mantener una velocidad segura, detector de obstáculos para evitar colisiones y evitar golpear los obstáculos en la calzada, y control lateral para mantener la posición del coche en la carretera. Además, se necesitarían sensores para alertar al coche de las condiciones del camino o el tiempo y así garantizar la seguridad las velocidades de viaje. Por ejemplo, el coche tendría que reducir la velocidad en condiciones de nieve o hielo.

Se llevan a cabo multitud de tareas mientras se conduce sin siquiera pensar en ello. Automatizar completamente el coche es una tarea difícil y algo lejana a día de hoy. Sin embargo, se han logrado avances en los sistemas individuales. El control de crucero es común en los coches actuales. El control adaptativo de crucero, en el que el coche frena en caso de detectar un vehículo lento en movimiento delante de ella, está empezando a estar disponible en los modelos de gama alta. Además, algunos vehículos están equipados con sensores para determinar si un obstáculo está cerca y suena una alarma acústica para el conductor cuando está demasiado cerca.

Respecto a la completa automatización de la conducción, existen algunos prototipos surgidos en los últimos años:

A nivel nacional, se tiene el representativo proyecto UTOPIA [ver 26] desarrollado por el Consejo Superior de Investigaciones Científicas (CSIC), en el que se pretenden transferir las técnicas desarrolladas para el control de robots autónomos al control de vehículos, modificando en la menor medida posible el entorno en que éstos han de evolucionar. Esto es, en entornos muy sencillos y poco realistas.

A nivel internacional, quizás los más representativos son los coches autónomos de Google [ver 27], que han logrado recorrer unas 140.000 miles, pero a muy bajas velocidades.

Cabe destacar también, la carrera de vehículos no tripulados DARPA Urban Challenge [28] que se celebra todos los años en EE.UU. y en la que participan muchas universidades americanas y europeas, como la universidad de Pensilvania o la universidad de Karlsruhe y algunos fabricantes de coches como por ejemplo Volkswagen y de sistemas como Honeywell.

La gran parte de los coches autónomos requieren de un gran despliegue de sensores capaces de medir y compensar la dinámica del automóvil en tiempo real, teniendo un coste tanto económico como computacional muy alto, y además no son capaces de alcanzar altas velocidades (con dificultad pasan de 50 Km/h) con la suficiente seguridad que la conducción en la carretera o ciudad requiere.

El objetivo de este proyecto es el control lateral. Con este tipo de control del vehículo, el conductor podría retirar sus manos del volante y dejar al coche girar el volante por sí sólo.

1.3.Definiciones y problema de control

De acuerdo a la norma ISO 8373, se define robot móvil como aquel robot que contiene todo lo necesario para su pilotaje y movimiento (potencia, control y sistema de navegación) [3]. En ocasiones, se designan con las siglas AGV (Autonomous Guided Vehicle o vehículo autónomo guidado).

Los robots móviles se pueden clasificar según el medio por el que se muevan, existiendo así robots móviles terrestres (denominados UGV Unmanned Ground Vehicles), robots móviles aéreos (conocidos como UAV Unmanned Aerial Vehicles) o robots móviles submarinos (identificados como AUV Autonomous Underwater Vehicles).

Dentro de los UGVs, se pueden diferenciar los robots móviles terrestres con ruedas (wheeled mobile robots) de aquellos que tienen otro sistema diferente de propulsión como puede ser cadenas o patas [4].

Una breve clasificación de los robots móviles terrestres con ruedas es según su estructura, según se propone en [20]:

- Robot tipo (3,0) o robot omnimóvil.
- Robot tipo (2,0) o robot tipo uniciclo
- Robot tipo (2,1)
- Robot tipo (1,1) o robot tipo Ackermann, que es la configuración de los automóviles convencionales de uso humano.
- Robot tipo (1,2)

Otra posible clasificación de los robots móviles en general (y de los robots móviles terrestres con ruedas en particular), es atendiendo a las restricciones a de movilidad [7], existiendo así:

- Robots holonómicos u holónomos, que pueden alcanzar cualquier punto del espacio de configuración desde cualquier punto, sin importar la trayectoria
- Robots no holonómicos o no holónomos, que pueden alcanzar cualquier punto del espacio de configuración desde cualquier punto, pero la trayectoria seguida es determinante. Éste es el caso de los automóviles convencionales de uso humano.

El caso que aplica a este proyecto es un UGV con ruedas no holonómico tipo Ackermann.

Los tres problemas de control genéricos objeto de múltiples investigaciones en los últimos tiempos de los UGV con ruedas son [20]:

- Seguimiento de trayectorias (Path following)
- Estabilización de trayectorias (Stabilization of trajectories)
- Estabilización de posturas fijas (Stabilization of fixed postures)

El problema de control que se va a tratar en este proyecto es el seguimiento de trayectorias o path following, cuyo objetivo es conseguir que dada una curva en el plano, una velocidad longitudinal del chasis del robot, y un punto P de interés ligado al chasis, dicho punto de interés siga la curva cuando el UGV se mueve a la velocidad especificada.

2. OBJETIVO DEL PROYECTO

2.1.Marco del proyecto

Este proyecto es sólo una parte de un proyecto más grande realizada por varias empresas y grupos de investigación colaboradores.

Debido a una cláusula de confidencialidad en el contrato del proyecto no se puede proporcionar más información del marco del proyecto o entidades colaboradoras.

2.2.Objetivos del proyecto

El objetivo del proyecto será diseñar un controlador capaz de conseguir que un UGV no holonómico tipo Ackermann (ver Apartado 1.) sea capaz de seguir adecuadamente una trayectoria dibujada en el suelo, si se parte de la información de la distancia a la curva en cada instante, obtenida por un sistema de visión ya existente que trabaja a una frecuencia máxima de 29 Hz. Debido a calentamientos anómalos la frecuencia de trabajo del sistema de visión puede ser menor. La velocidad deseada para el movimiento del UGV estará en torno a 10-15 Km/h.

El sistema de visión también es capaz de detectar y decodificar, en caso de existir, unos códigos que poseen información del circuito, en particular el tipo de tramo en el que se entra y el radio de curvatura del mismo. Dichos códigos se encontrarán a la derecha de la línea, y justo en la zona de cambio de tramo.

El UGV tiene un hardware y un software que le permite moverse adecuadamente, a través de unas consignas de velocidad en kilómetros por hora y de giro de volante en grados, transmitidas del modo que se verá en el Apartado 3. Para trabajar a lo largo del proyecto se emplearán consignas de velocidad en metros por segundo y de giro de ruedas en radianes, teniendo que hacer la transformación a valores del UGV, en el momento de la implementación.

La aceptación del controlador del UGV diseñado vendrá dada por dos criterios:

- Correcta realización de las pruebas que se muestran en el Anexo I.
- Criterio subjetivo del probador (cliente) que acepte las aceleraciones laterales y las oscilaciones como las habituales en una conducción, tratando de eliminarlas.

3. DESCRIPCIÓN DEL UNMANNED GROUND VEHICLE (UGV)

En éste apartado se describirá de forma breve el sistema con el que se va a trabajar en el proyecto.

Debido a la cláusula de confidencialidad mencionada en el Apartado 2., no se pueden llevar a cabo mayores descripciones que las estrictamente necesarias para la comprensión del proyecto.

3.1.Descripción del sistema mecánico

El UGV a controlar es un Citroën C3 modificado convenientemente para llevar a cabo las tareas de control.

Ilustración 1: Fotos C3

Posee las siguientes características de importancia en el proyecto, medidas sobre el prototipo:

- Distancia entre ejes: L=2.46 m.
- Giro máximo de las ruedas: $\phi_{max} = 26º$
- Tiempo de establecimiento del giro de ruedas: depende de la velocidad y del giro de referencia, pero se puede aproximar a un sistema de primer orden de tiempo de establecimiento $t_s = 0.05\ seg$.

3.2.Descripción del sistema visión

El sistema de adquisición de datos del exterior es un sistema de visión situado en la parte delantera del vehículo, cuyo centro se encuentra en el eje del UGV ($l_2 = 0$), y a una distancia de $l_1 = 3.41\ m$ del eje trasero del vehículo.

Ilustración 2: Foto del sistema de visión

El sistema de visión es capaz de obtener, mediante complejos algoritmos desarrollados en otro proyecto, la distancia del centro del sistema de visión, a la curva pintada en el suelo, realizado de dos maneras posibles:

- Distancia mínima del centro del sistema de visión a la curva (d_{min})
- Distancia a la curva medida sobre un eje perpendicular al eje de simetría del UGV (d_{per}). En éste caso también se obtiene el ángulo (β_{lin})

De modo que el uso de una u otra distancia es equivalente, dado por la relación: $d_{min} = d_{per} \cdot \cos(\beta_{lin})$.

Ilustración 3: Foto de la línea del circuito

En la imagen que se muestra a continuación se observa una representación de una imagen captada por la cámara en un instante de tiempo determinado. El punto P es el centro del sistema de visión y punto de interés que debe ir siguiendo la curva. En azul se muestra la curva a seguir.

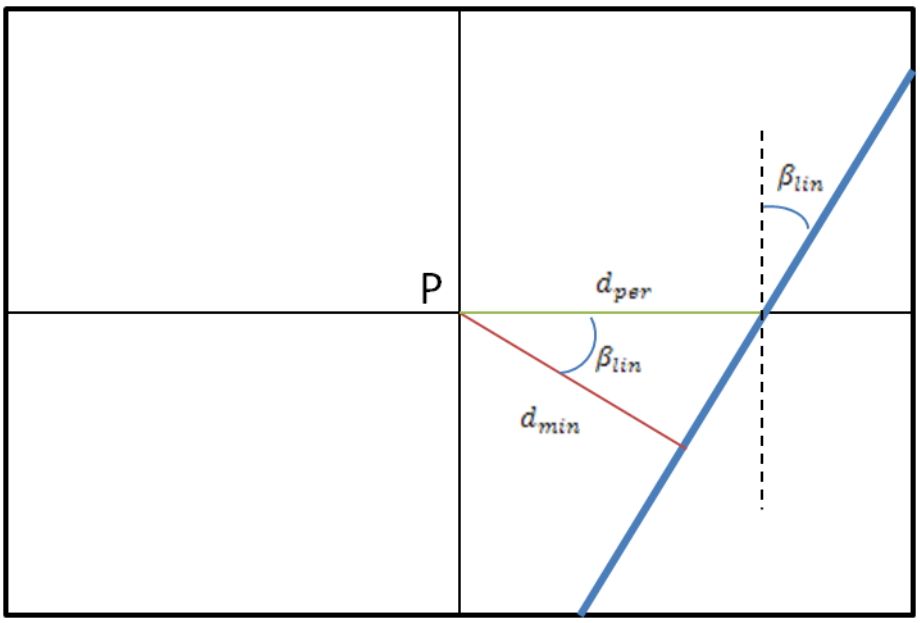


Ilustración 4: Representación de una imagen captada por el sistema de visión

Como se verá más adelante, se preferirá utilizar la distancia mínima a la curva.

La distancia máxima que el sistema de visión es capaz de leer es $d_{max} = 0.2\,m$, ya que una distancia mayor significa que la línea no es observada por el sistema de visión.

Por otro lado, el sistema de visión es capaz de leer una serie de marcas codificadas que hay al comienzo de cada tramo diferente del circuito, que indican el tipo de tramo en el que se va a entrar, para conseguir que el controlador ejecute las órdenes oportunas con mayor información del circuito.

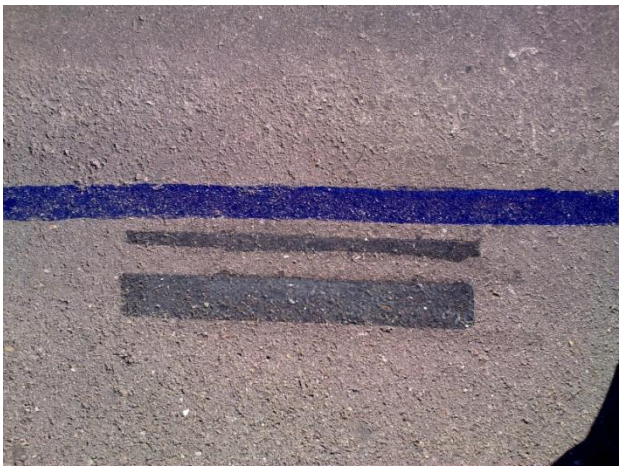


Ilustración 5: Foto de los códigos

El sistema de visión trabajará a una frecuencia máxima de $f = 29\,Hz$, aunque debido a calentamientos anormales puede bajar, funcionando en condiciones de emergencia a $f_{min} = 20\,Hz$.

3.3.Descripción del HW de control

Un esquema general del hardware de control del UGV es el siguiente:

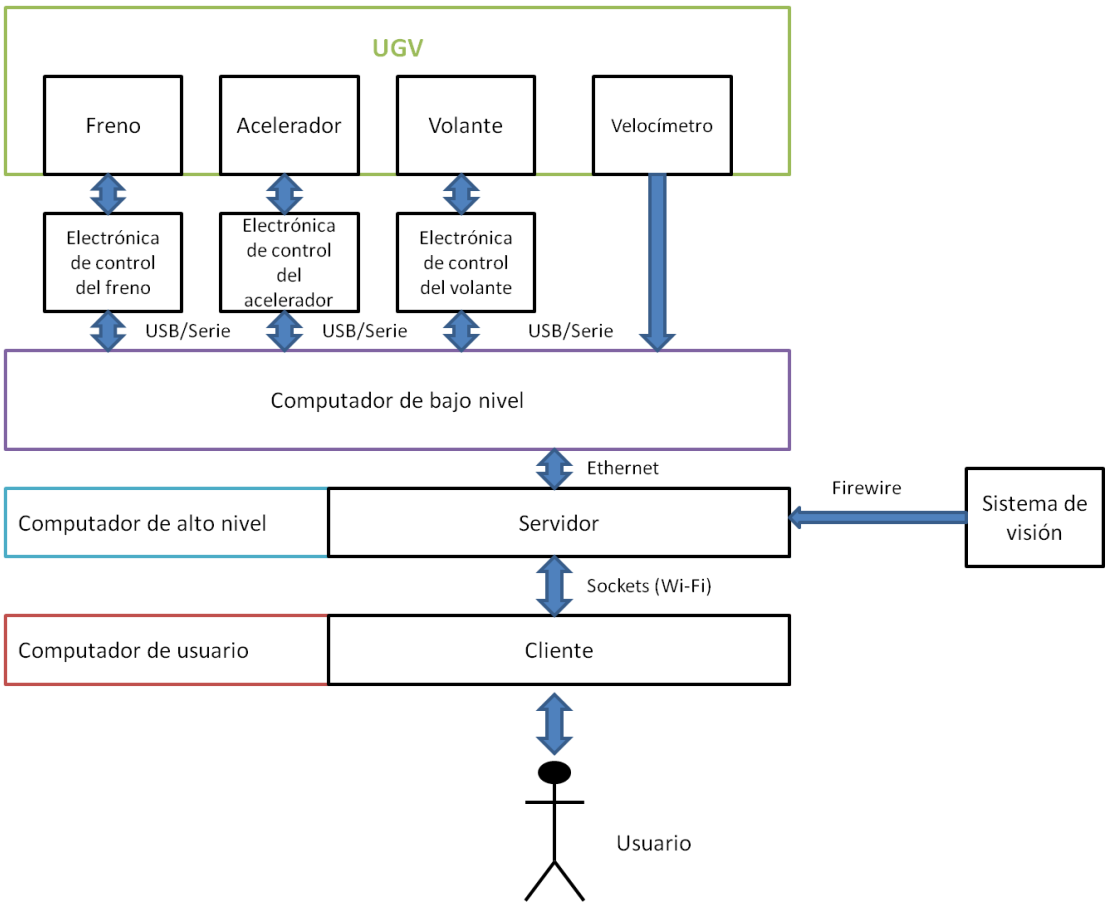


Ilustración 6: Descripción HW del UGV

El UGV tiene cambio automático de modo que con la exclusiva actuación de freno, acelerador y volante, se puede conseguir conducir el coche, sin necesidad de controlar embrague y cambio de marchas.

Ilustración 7: Foto del cambio automático

Tanto el computador de bajo nivel como toda la electrónica que hay por debajo suya hacia el UGV ha sido diseñado e implementado por una entidad externa al Grupo de Visión por Computador de la UPM, en el que se ha trabajado, por lo que se utilizará exclusivamente a nivel usuario.

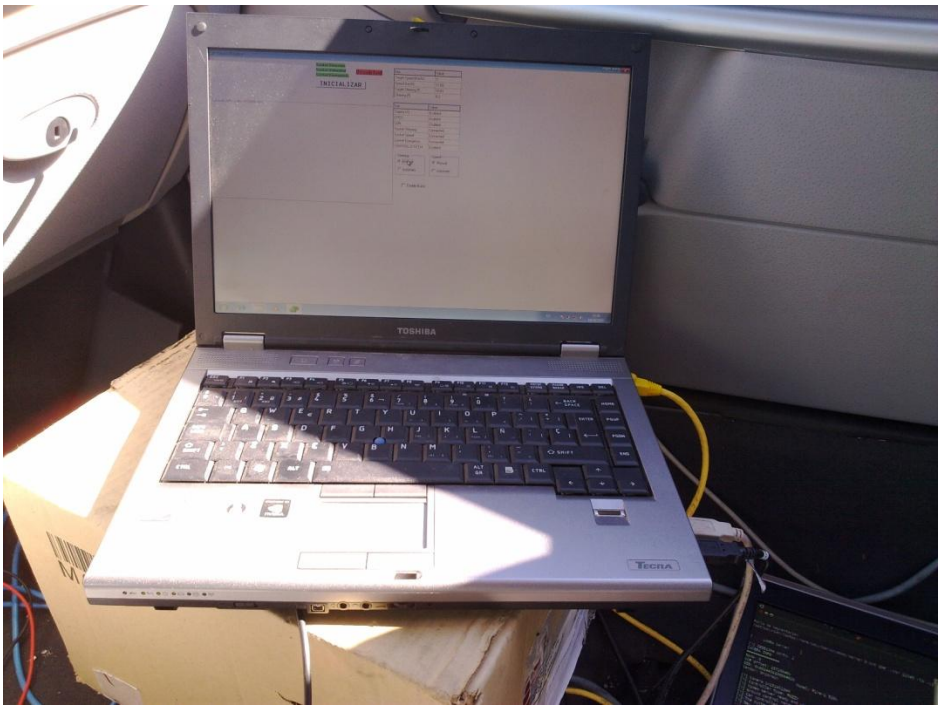


Ilustración 8: Foto del computador de bajo nivel

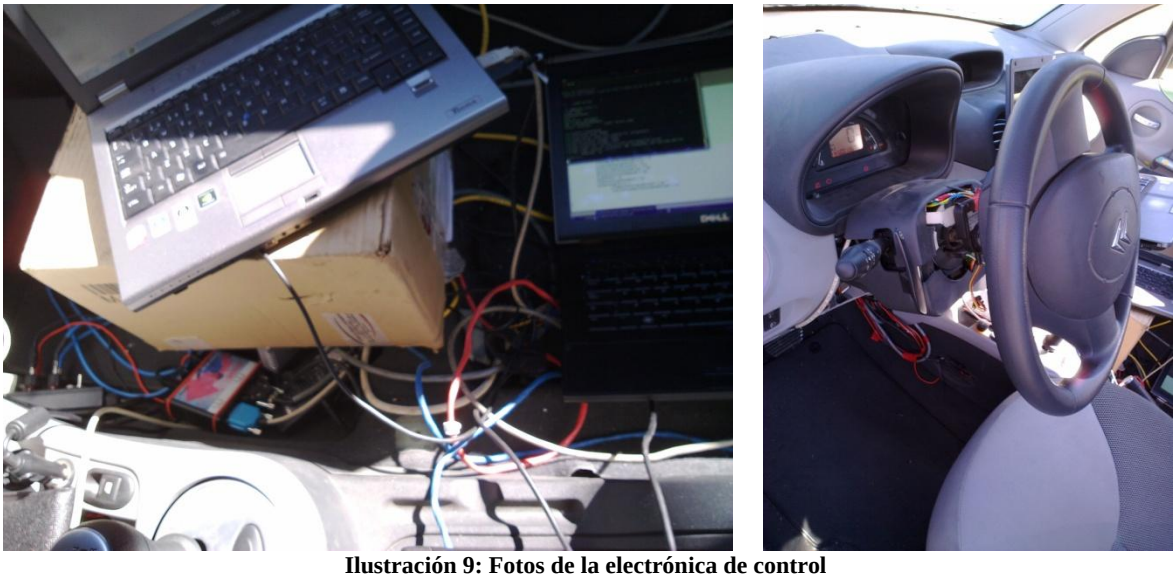


Ilustración 9: Fotos de la electrónica de control

En el computador de bajo nivel están implementados los siguientes lazos de control:

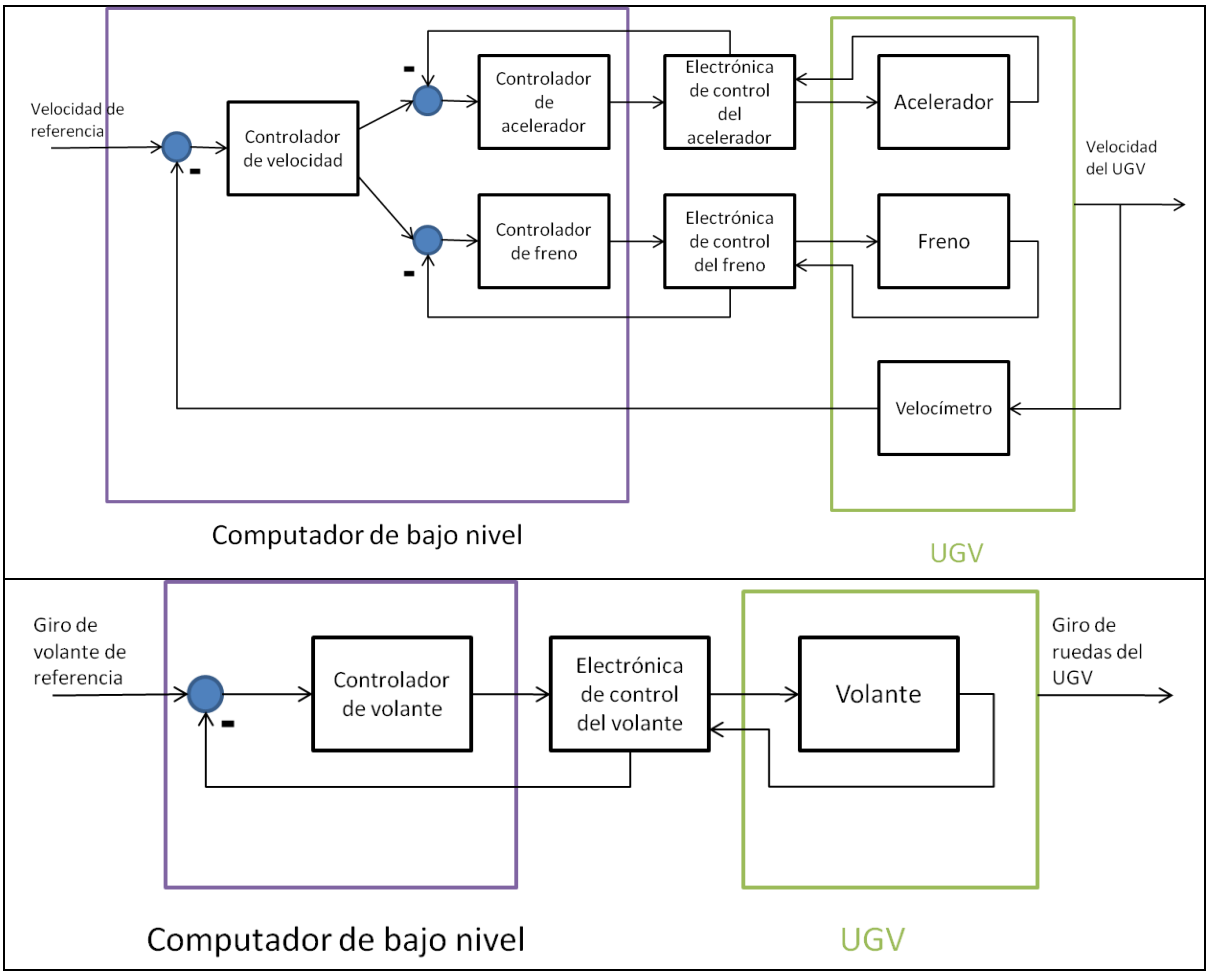


Ilustración 10: Lazos de control del computador de bajo nivel

Tanto el computador de alto nivel como el sistema de visión, como el computador de usuario han sido diseñados e implementados por el Grupo de Visión por computador de la UPM, en un proyecto distinto a éste, por lo que se utilizarán a nivel usuario, y no se aportará mayor descripción.

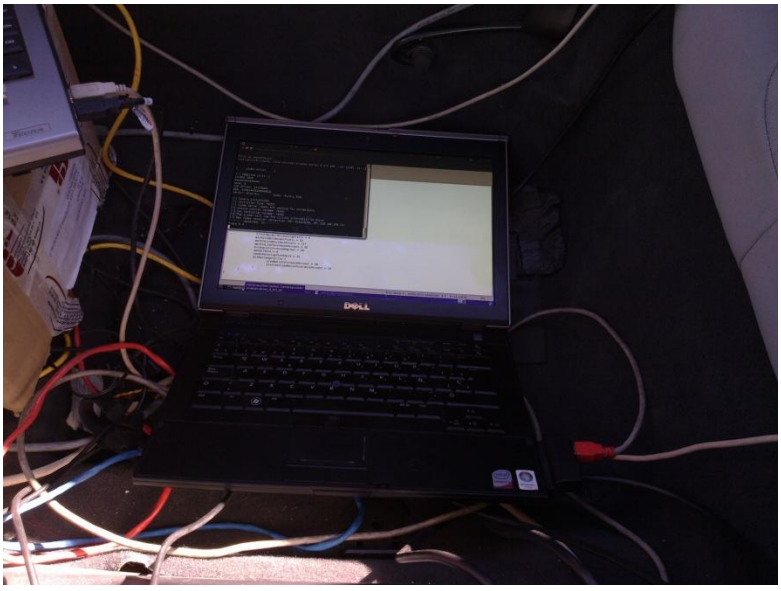


Ilustración 11: Foto del computador de alto nivel

Ilustración 12: Foto del computador de usuario

3.4.Descripción del SW de control

Como se ha dicho en el apartado 3.3., el software del computador de bajo nivel ha sido diseñado por una entidad externa al grupo de Visión por Computador, por lo que el software corrido en él es exclusivamente utilizado a nivel usuario, y no se pueden dar más detalles debido a la cláusula de confidencialidad. El sistema operativo del computador de bajo nivel es Microsoft Windows Vista.

El software del computador de alto nivel sí que ha sido diseñado por el Grupo de Visión por Computador, no obstante forma parte un sub-proyecto ajeno a éste, sometido a la misma cláusula de confidencialidad, siendo utilizado en este proyecto exclusivamente a nivel usuario, incluyendo en una zona del programa servidor, el código del controlador diseñado objeto de este proyecto.

La zona de código está perfectamente delimitada, e incluye el envío y recepción de información, así como el tratamiento de excepciones. Habrá una zona en la que se incluye el código que trabaje a la frecuencia del sistema de visión, mediante un thread adecuadamente controlado. Otra de las zonas trabajará a la frecuencia máxima que permita el computador, mediante otro thread perfectamente controlado.

Se trabaja en lenguaje C++ corrido sobre el sistema operativo Linux Ubuntu RT.

Las variables con las que se trabajará son las siguientes:

Nombre de variable	Tipo de dato	Descripción
turn	double	Será la variable en la que se incluya la consigna de giro de volante en grados, calculada por el controlador. [1º rueda=18º volante] Tomará valores positivos para giros de volante en el sentido de las agujas del reloj, y negativos para el contrario. Su valor máximo vendrá limitado por el giro máximo de las ruedas ($\phi_{max} = 26º$), luego el valor máximo, en valor absoluto será 468º.
vel_actual	double	Será la variable que lleve información de la velocidad actual del UGV en Km/h.
error	double	Será la variable que contenga la información de la distancia del centro del sistema de visión (punto de interés) a la línea, en píxeles. [1 pixel=0.00132 m] Tomará valores positivos si el UGV se encuentra por debajo de la línea según el sentido de marcha.
tramo	int	Variable que contiene la información del tipo de tramo y su curvatura, leida de los códigos pintados en el circuito. Toma los siguientes valores: - Tramo=0: tiene doble significado. O bien no se ha leído ningún código, o bien indica que se está en el primer tramo del circuito de prueba, que es una recta. - Tramo=1: Indica que se está en el segundo tramo del circuito de prueba, que es una circunferencia de radio 20 m. - Tramo=2: Indica que se está en el tercer tramo del circuito de prueba, que es una recta. - Tramo=3: indica que se está en el cuarto tramo del circuito de prueba, que es una circunferencia de radio 11.2 m.
T	float	Frecuencia de trabajo del sistema de visión en hertzios.

Tabla 1: Variables del SW de control

El software del computador de usuario también ha sido diseñado por el Grupo de Visión, en otro proyecto ajeno a éste. Utiliza el sistema operativo Windows 7, y exclusivamente sirve de interfaz de control para el usuario no programador.

3.5.Descripción del circuito de pruebas

Para la verificación de la bondad del controlador diseñado en este proyecto, existe un circuito de pruebas en el recinto del Instituto de investigación del Automóvil (INSIA) de la UPM.

El circuito está compuesto por cuatro tramos:

El primer tramo será una recta de 42 metros de longitud. El segundo tramo será una circunferencia de 20 m de radio. El tercer tramo otra recta de 42 metros de longitud. El cuarto y último tramo es una circunferencia de 11.2 metros de longitud.

Cada tramo es perfectamente tangente con su tramo posterior, de modo que se forma un circuito cerrado cuya curva es continua y derivable (será un requisito que se analizará más adelante).

El circuito será siempre recorrido en sentido antihorario. El sentido de recorrer el circuito será esencial, ya que las variables tramo vistas en el apartado anterior, dan la información del tipo de tramo y de su sentido de recorrerlo. En el apartado 4., se verá la diferencia de recorrer el circuito en un sentido o en otro.

El circuito es plano, es decir, no existe desnivel alguno, siendo perfectamente normal a la línea de aceleración de la gravedad.

En el Anexo II se pueden ver los cálculos llevados a cabos para definir perfectamente el circuito en el plano, y su posterior simulación.

4. MODELADO DEL UGV

4.1.Modelo cinemático frente a modelo dinámico

El modelo dinámico de un robot móvil [3], establece la relación matemática entre la localización del robot móvil en el espacio ($\vec{q}$), y sus derivadas ($\dot{\vec{q}}, \ddot{\vec{q}}$); las fuerzas y pares aplicados ($\vec{\tau}$), y los parámetros dimensionales del robot (longitudes, masas e inercias).

La localización del robot móvil en el espacio se representa en general por el vector $\vec{q} = [q_0, q_1, .., q_n]^t$, donde $q_i = [x_i, y_i, z_i, \alpha_i, \beta_i, \gamma_i]^t$ indica la posición y orientación del sistema de referencia ligado a cada grado de libertad interno del robot, siendo n el número de grados de libertad internos del robot. El vector $\vec{q}$ tendrá entonces dimensión $(6 \cdot (\mathrm{n} + 1))$ x 1.

En el caso de que el robot móvil no tuviese ningún grado de libertad interno, es decir, el robot fuese una estructura no articulada, se tendría $\vec{q} = [x, y, z, \alpha, \beta, \gamma]^t$; y el sistema de referencia podría estar ligado a cualquier punto del robot. Por sencillez se suele elegir el centro de gravedad del mismo, aunque no es estrictamente necesario.

Las fuerzas y pares aplicados $\vec{\tau} = [\tau_0, \tau_1, .., \tau_n]^t$ miden la resultante efectiva sobre el origen de cada sistema de referencia ligado a cada grado de libertad interno del robot ($\tau_i = [F_{x_i}, F_{y_i}, F_{z_i}, M_{\alpha_i}, M_{\beta_i}, M_{\gamma_i}]$).

Los parámetros dimensionales del robot necesarios en el modelo dinámico se representan mediante las matrices de inercia, de fuerzas de Coriolis y de fuerzas gravitatorias, siendo:

$I(\vec{q})$: Matriz de inercia (dimensión $(6 \cdot (\mathrm{n} + 1))$ x $(6 \cdot (\mathrm{n} + 1))$)

$C(\vec{q}, \dot{\vec{q}})$: Matriz de fuerzas de Coriolis (dimensión $(6 \cdot (\mathrm{n} + 1))$ x 1)

$G(\vec{q})$: Matriz de fuerzas de gravedad (dimensión $(6 \cdot (\mathrm{n} + 1))$ x 1)

El modelo dinámico del sistema se representará entonces por:

$$\vec{\tau} = I(\vec{q}) \cdot \ddot{\vec{q}} + C(\vec{q}, \dot{\vec{q}}) + G(\vec{q}) \qquad (4.1.1)$$

Puesto que resulta más sencillo medir los pares y fuerzas aplicadas por cada actuador, se puede hacer una transformación de los pares y fuerzas a $\vec{\tau}_a$ que indica la fuerza y/o par efectivo aplicado por cada actuador.

$$\vec{\tau}_a = I(\vec{q}) \cdot \ddot{\vec{q}} + C(\vec{q}, \dot{\vec{q}}) + G(\vec{q}) \qquad (4.1.2)$$

Las matrices de inercia, fuerzas de Coriolis y fuerzas gravitatorias serán ahora diferentes. Del mismo modo, dichas matrices cambiarían si se modificase la elección del sistema de referencia ligado a cada grado de libertad interno del robot.

En el caso que nos aplica, de un robot móvil terrestre (UGV) no holonómico tipo Ackermann que se va a mover en el plano, se tiene:

$$n = 0$$
$$q = [x, y, \vartheta]^t$$
$$\tau_a = [\tau_m, \tau_v]$$

Donde τ_m representa el par motor efectivo que se ejerce en las ruedas tractoras, y τ_v el par efectivo ejercido por el servo de dirección. El número de grados de libertad del sistema de referencia del robot ha pasado de 6 a 3, puesto que se ha supuesto movimiento plano.

El modelo cinemático de un robot móvil [3] trata de expresar la relación matemática entre la localización del robot móvil en el espacio ($\vec{q}$), y su primera derivada ($\dot{\vec{q}}$); las consignas de velocidad o posición de los actuadores del robot ($\vec{u}$), y los parámetros cinemáticos del robot (longitudes).

La localización del robot móvil en el espacio $\vec{q}$ representa lo mismo que se representaba en el modelo dinámico, por lo que no requiere mayor aclaración.

El vector de consignas de velocidad o posición de los actuadores del robot ($\vec{u}$), será de dimensión $k\ x\ 1$, siendo k el número de actuadores del robot (suponiendo que cada actuador sólo permite una única consigna).

Los parámetros cinemáticos del robot son función de las longitudes del robot, y se representan mediante una matriz cinemática $K(\vec{q}, \vec{u})$. Dicha matriz será diferente según se elija el sistema de referencia ligado a cada grado de libertad interno del robot.

El modelo cinemático será entonces:

$$\dot{\vec{q}} = K(\vec{q}, \vec{u}) \tag{4.1.3}$$

Tanto el modelo cinemático como el dinámico se pueden representar como modelos de estado, en la forma habitual de la teoría de control $\dot{x} = f(x, u)$, donde x representa el vector de variables de estado del sistema, y u el vector de entradas al sistema.

Si se transforma el modelo dinámico a un modelo de estado, resulta:

$$\begin{bmatrix} \dot{\vec{q}} \\ \ddot{\vec{q}} \end{bmatrix} = D\left(\begin{bmatrix} \vec{q} \\ \dot{\vec{q}} \end{bmatrix}, \vec{\tau}_a \right) \tag{4.1.4}$$

Siendo:

$\begin{bmatrix} \vec{q} \\ \dot{\vec{q}} \end{bmatrix}$: vector de variables de estado

$\vec{\tau}_a$: vector de entradas

La transformación del modelo cinemático a un modelo de estado lleva a:

$$\dot{\vec{q}} = K(\vec{q}, \vec{u}) \qquad (4.1.5)$$

Donde:

$\vec{q}$: vector de variables de estado

$\vec{u}$: vector de entradas

Tanto el modelo cinemático como el modelo dinámico pueden ser utilizados para el diseño del controlador objeto del proyecto, partiendo de su transformación a modelo de estado, no obstante, el uso del modelo cinemático tiene una serie de ventajas frente al modelo dinámico:

- El modelo cinemático es más simple que el modelo dinámico, esto es, el modelo dinámico requiere el conocimiento de las matrices I, C y G cuya determinación suele ser complicada; mientras que el modelo cinemático requiere el conocimiento de una sola matriz K, que toma valores más sencillos, ya que exclusivamente dependen de valores de distancias.
- Las entradas de los actuadores suelen ser consignas de velocidad o posición, no de fuerza o par, por lo que la ejecución de la acción de control va a ser más sencilla en el modelo cinemático.

Debido a esto, se utilizará el modelo cinemático en lugar del modelo dinámico para el diseño del controlador.

4.2.Modelo cinemático plano. Simplificación.

El modelo general de robot móvil no holonómico tipo Ackermann de cuatro ruedas y estructura fija en el plano, se representa en el dibujo que se muestra a continuación:

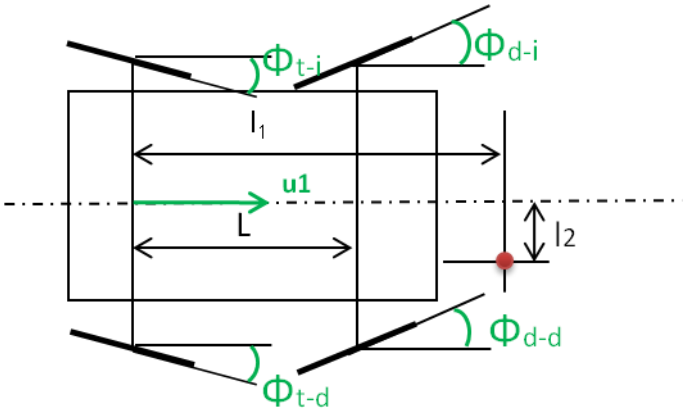


Ilustración 13: Modelo general de robot móvil no holonómico tipo Ackermann

En el caso que nos aplica las ruedas traseras del UGV no son articuladas, con lo cual se hace $\phi_{t-d} = \phi_{t-i} = 0$.

Si se imagina una rueda en el eje del UGV, cuyo ángulo de articulación vale $\phi = \frac{\phi_{d-d}+\phi_{d-i}}{2}$, el modelo del UGV se puede simplificar al llamado modelo de la bicicleta del siguiente modo:

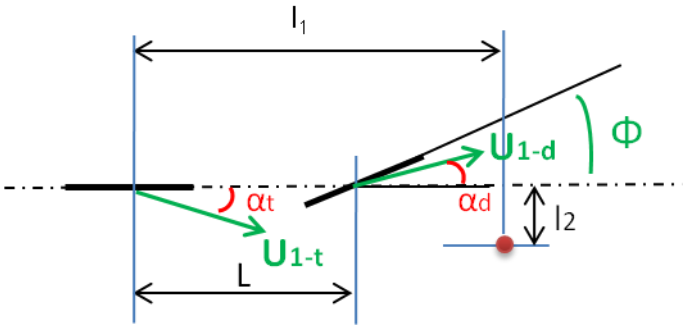


Ilustración 14: Modelo de la bicicleta

El hacer esa simplificación no supone pérdida de información puesto que en el UGV real, la consigna de dirección es única y representa precisamente el valor de la rueda imaginaria del eje.

Si se hace la hipótesis de que los vectores de velocidad de cada rueda están alineados con la dirección de las ruedas, es decir, que los ángulos de deriva de las ruedas delantera y trasera son nulos ($\alpha_d = \alpha_t = 0$), el modelo se simplifica aún más. Ésta hipótesis es cierta si el UGV se mueve a velocidades suficientemente bajas, como es el caso que aplica.

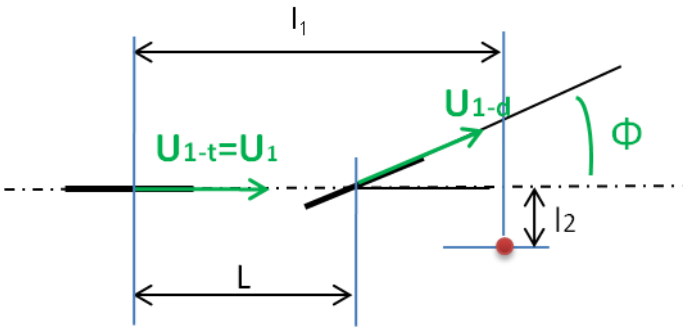


Ilustración 15: Modelo de la bicicleta simplificado

Si se trata de encontrar las ecuaciones del modelo cinemático en el plano del modelo de UGV que hemos obtenido, para el punto que se encuentra en la intersección del eje de la rueda trasera con el eje del UGV, del siguiente modo:

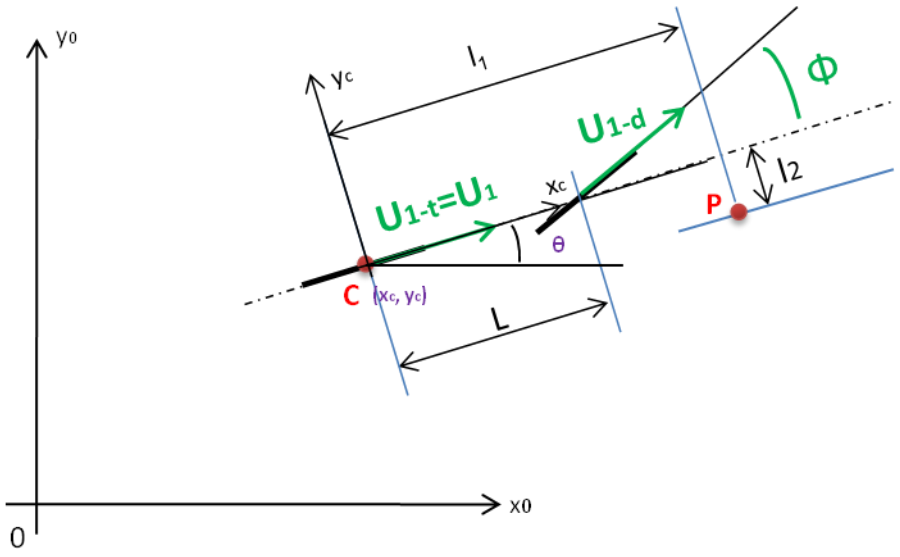


Ilustración 16: Modelo de la bicicleta simplificado con sistema de referencia en el plano

Por un lado, se tiene el desplazamiento del punto en el plano, de modo que:

$$\dot{x}_c = u_1 \cdot \cos\vartheta \quad (4.2.1)$$
$$\dot{y}_c = u_1 \cdot \sin\vartheta \quad (4.2.2)$$

Para calcular la rotación utilizamos las propiedades del centro instantáneo de rotación (CIR):

$$u_1 = R_c \cdot \dot{\vartheta} \quad (4.2.3)$$

Donde R_c es el radio de la circunferencia descrita con un giro de la rueda directriz ϕ.

Por otro lado, geométricamente, se tiene:

$$\tan\phi = \frac{L}{R_c} \rightarrow R_c = \frac{L}{\tan\phi} \quad (4.2.4)$$

Con lo que sustituyendo y despejando, se obtiene:

$$\dot{\vartheta} = u_1 \cdot \frac{\tan\phi}{L} \quad (4.2.5)$$

Luego las ecuaciones cinemáticas del modelo de bicicleta sobre el plano cuyo punto de interés es el centro de la rueda trasera son:

$$\dot{x}_c = u_1 \cdot \cos\vartheta \quad (4.2.6)$$
$$\dot{y}_c = u_1 \cdot \sin\vartheta \quad (4.2.7)$$
$$\dot{\vartheta} = u_1 \cdot \frac{\tan\phi}{L} \quad (4.2.8)$$

Como el punto de interés de el UGV objeto del proyecto es el sistema de visión y se encuentra desplazado del centro del eje de la rueda trasera, es necesario calcular el modelo cinemático plano del modelo de bicicleta sobre el plano con el punto de interés desplazado.

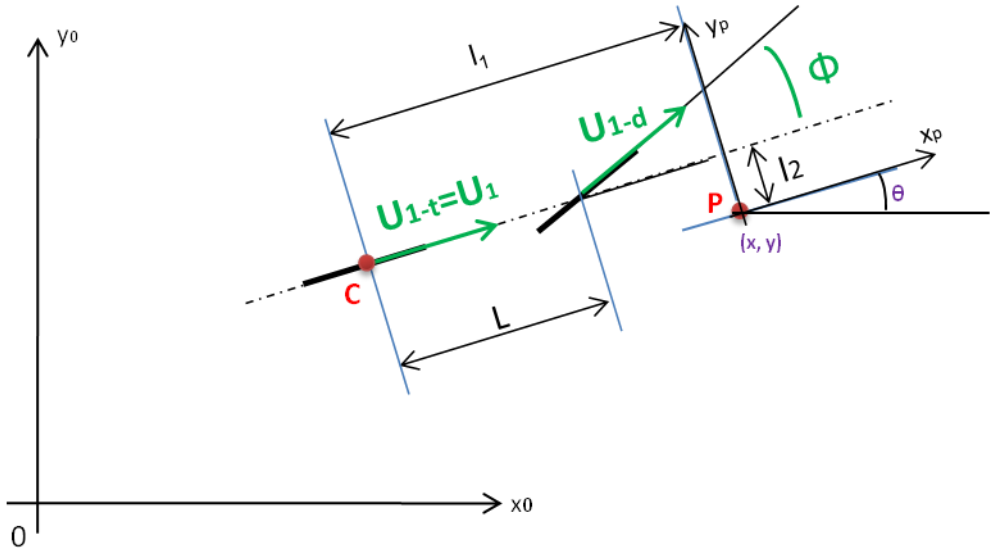


Ilustración 17: Modelo de la bicicleta simplificado con sistema de referencia en el plano con punto de interés

Bastará con hacer un desplazamiento de las ecuaciones, ya que la rotación se conserva, obteniéndose [20]:

$$\dot{x} = u_1 \cdot \left[\cos\vartheta - \frac{\tan\phi}{L} \cdot (l_2 \cdot \cos\vartheta + l_1 \cdot \sin\vartheta)\right] \quad (4.2.9)$$

$$\dot{y} = u_1 \cdot \left[\sin\vartheta + \frac{\tan\phi}{L} \cdot (l_1 \cdot \cos\vartheta - l_2 \cdot \sin\vartheta)\right] \quad (4.2.10)$$

$$\dot{\vartheta} = \frac{u_1}{L} \cdot \tan\phi \quad (4.2.11)$$

NOTA:

El sentido de los ángulos (tanto ϕ, como ϑ) se ha elegido en dirección del eje z_0, de modo que el sistema de referencia fijo $\{0; x_0, y_0, z_0\}$ esté definido a derechas (z_0 saliente del papel hacia el lector).

4.3.Modelo cinemático en base de Frenet

Como el objetivo del proyecto es conseguir que el UGV siga una trayectoria pintada en el plano, es necesario hacer una modificación el modelo cinemático plano calculado en el apartado anterior, de modo que se obtenga un modelo cinemático que relacione los parámetros de actuación del UGV, que serán las entradas del sistema (u_1, ϕ), con la trayectoria plana que se desea seguir.

La mejor manera de conseguir esto es transformar el modelo cinemático a una base de Frenet ligada a la curva [6]. Se definirá la base de Frenet de modo que el origen de la base P_s pertenezca

en todo momento a la curva, el eje x_s sea tangente en todo momento a la curva, el eje y_s pase por el punto P de interés del UGV, y el eje z_s forme una base a derechas.

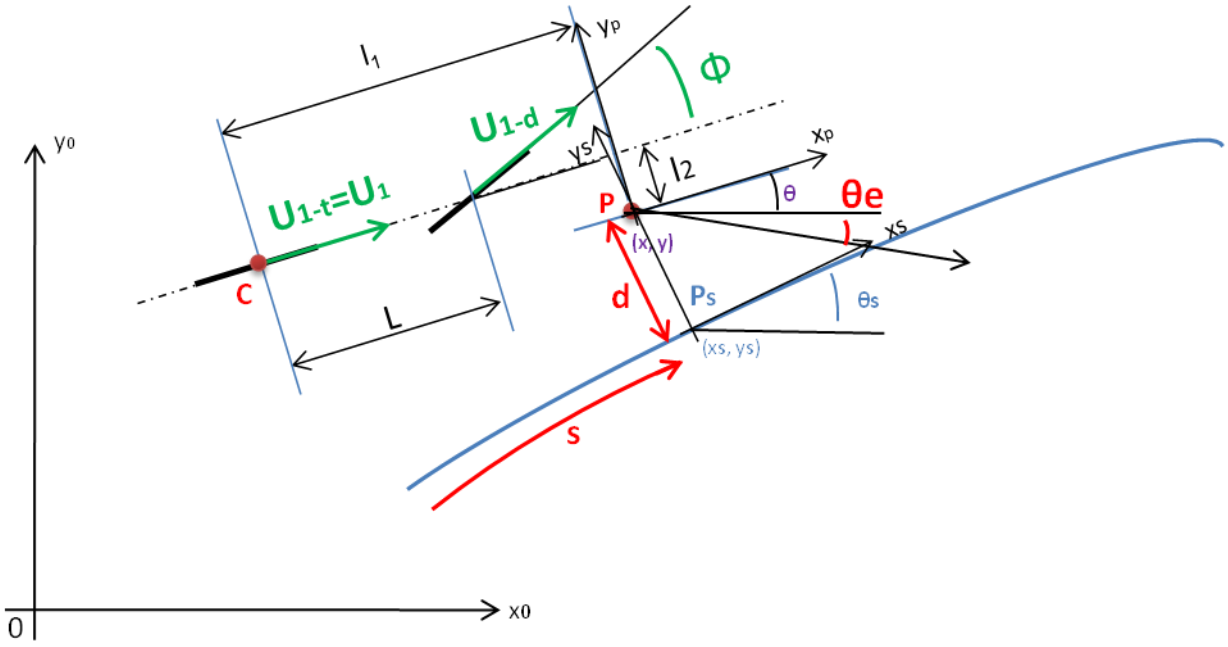


Ilustración 18: Modelo de la bicicleta simplificado en base de Frenet

Se define c como la curvatura de la trayectoria pintada en el plano, y es función del punto de la curva sobre la que se encuentre la base de Frenet $c(s) = \frac{1}{R(s)}$, siendo R(s) el radio de la curva en el punto s.

Bajo ciertas hipótesis que se analizarán a continuación, se puede definir matemáticamente la curvatura de la siguiente manera:

$$c(s) = \frac{\partial \vartheta_s}{\partial s} = \frac{d\vartheta_s}{dt} \cdot \frac{dt}{ds} = \frac{\dot{\vartheta}_s}{\dot{s}} \tag{4.3.1}$$

Por otro lado se define el ángulo de rotación de la base de Frenet como:

$$\vartheta_e = \vartheta - \vartheta_s \tag{4.3.2}$$

Derivando la ecuación, se tiene:

$$\dot{\vartheta}_e = \dot{\vartheta} - \dot{\vartheta}_s \tag{4.3.3}$$

Y sustituyendo los valores de $\dot{\vartheta}$ de la ecuación 4.2.11 y el de $\dot{\vartheta}_s$ de la ecuación 4.3.1, se obtiene:

$$\dot{\vartheta}_e = \frac{u_1}{L} \cdot \tan\phi - \dot{s} \cdot c(s) \tag{4.3.4}$$

Haciendo una serie de transformaciones sobre los vectores que ligan la posición de los puntos P y P_s, se obtienen las siguientes ecuaciones:

$$\dot{s} = \frac{u_1}{1 - d \cdot c(s)} \cdot \left[\cos\vartheta_e - \frac{\tan\phi}{L} \cdot (l_2 \cdot \cos\vartheta_e + l_1 \cdot \sin\vartheta_e)\right] \quad (4.3.5)$$

$$\dot{d} = u_1 \cdot \left[\sin\vartheta_e + \frac{\tan\phi}{L} \cdot (l_1 \cdot \cos\vartheta_e - l_2 \cdot \sin\vartheta_e)\right] \quad (4.3.6)$$

Recopilando las ecuaciones 4.3.5, 4.3.6 y 4.3.4, se obtiene el modelo cinemático plano del UGV simplificado a bicicleta en una base de Frenet [5] :

$$\dot{s} = \frac{u_1}{1 - d \cdot c(s)} \cdot \left[\cos\vartheta_e - \frac{\tan\phi}{L} \cdot (l_2 \cdot \cos\vartheta_e + l_1 \cdot \sin\vartheta_e)\right] \quad (4.3.7)$$

$$\dot{d} = u_1 \cdot \left[\sin\vartheta_e + \frac{\tan\phi}{L} \cdot (l_1 \cdot \cos\vartheta_e - l_2 \cdot \sin\vartheta_e)\right] \quad (4.3.8)$$

$$\dot{\vartheta}_e = \frac{u_1}{L} \cdot \tan\phi - \dot{s} \cdot c(s) \quad (4.3.9)$$

4.3.1. Requisitos del modelo en base de Frenet

Hay dos limitaciones derivadas del uso del modelo en base de Frenet.

Por un lado están los requisitos matemáticos de la curva, que se representan en las ecuaciones a través de la curvatura $c(s)$, siendo necesario que c exista para todo valor de s.

Lo cual se traduce en:

- Que haya curva a lo largo de toda la trayectoria del UGV, es decir que la curva sea continua.
- Y que además de que haya curva, se pueda definir un valor de la curvatura, lo cual se traduce en que la curva ha de ser de clase C^1. Esto significa que para cada punto de la curva, el ángulo ϑ_e ha de ser continuo (lo cual es obligatorio por ser ϑ_e variable de estado), y derivable (que es la condición adicional impuesta por la necesidad de existencia de curvatura a lo largo de toda la trayectoria).

Luego las exigencias del modelo cinemático en base de Frenet a la curva son que sea continua y derivable.

Por otro lado están los requisitos físicos al sistema percepción (sistema de visión), que obliga, para que sea válido el sistema en base de Frenet, a calcular el valor de d, como la distancia mínima a la curva. Usualmente, la distancia a la curva se suele medir sobre una recta fija perpendicular al eje del UGV. El hecho de medir la distancia a la curva de la manera usual supone una aproximación que en el caso de funcionar el sistema adecuadamente no debería ser inconveniente.

4.3.2. Notas sobre la curvatura $c(s)$

Como hemos visto, se define la curvatura como $c(s) = \frac{\partial \vartheta_s}{\partial s}$. Al definirse el ángulo ϑ_s en sentido del eje z_s creciente, significará que una curva en la que la variación de dicho ángulo es positiva (luego la curvatura es c>0) será tomada en el sentido contrario a las agujas del reloj. En caso contrario, si la c<0, la curva será descrita en sentido de las agujas del reloj.

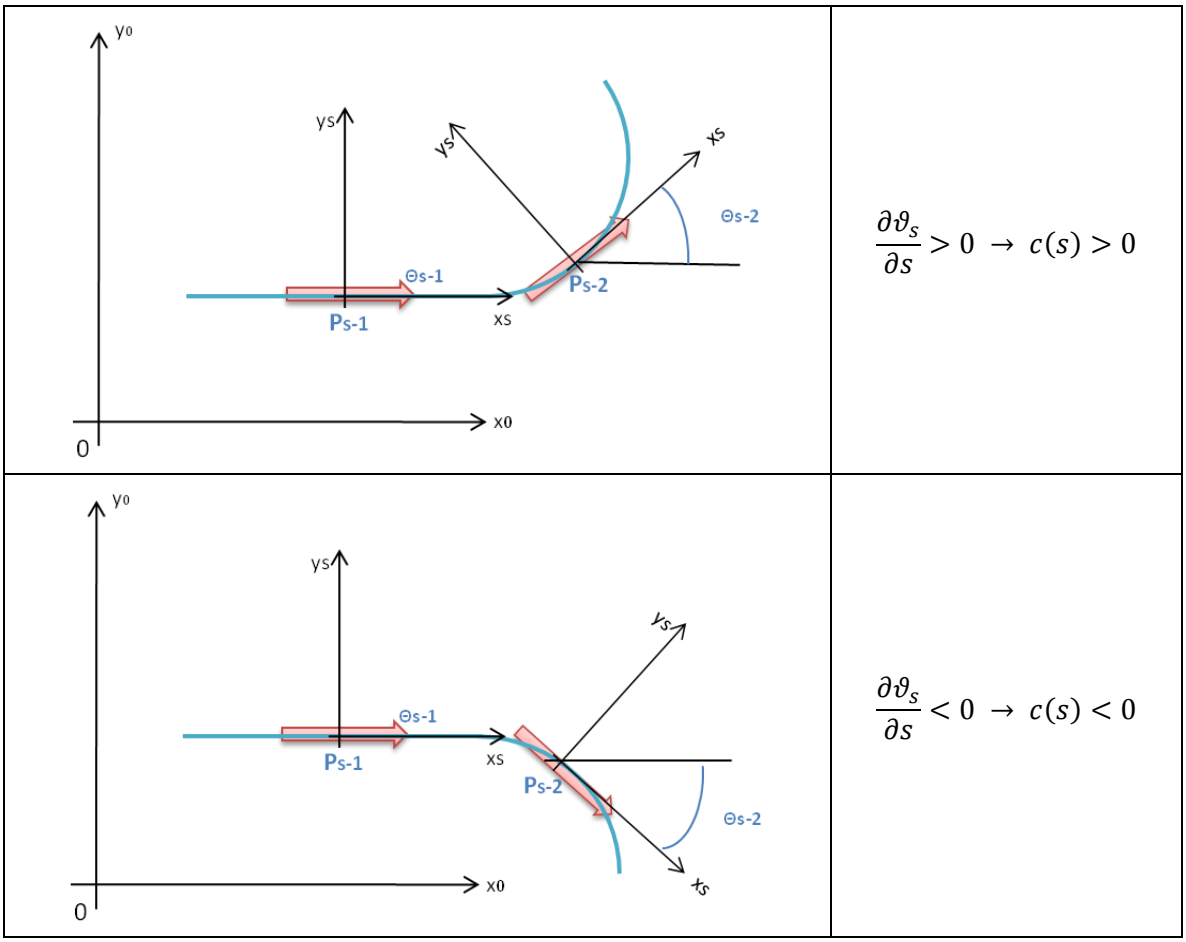


Ilustración 19: Nota sobre la curvatura

Si se conocen las ecuaciones paramétricas de una curva en el plano:

$$x_s = f_x(s) \qquad (4.3.2.1)$$
$$y_s = f_y(s) \qquad (4.3.2.2)$$

Se define la pendiente de la recta tangente a la curva en el punto P_s como:

$$m(s) = \tan \vartheta_s = \frac{dy_s}{dx_s} = \frac{\frac{dy_s}{ds}}{\frac{dx_s}{ds}} = \frac{f_y{}'(s)}{f_x{}'(s)} \rightarrow \vartheta_s = \operatorname{atan}\big(m(s)\big) \qquad (4.3.2.3)$$

Puesto que la curvatura se definía como:

$$c(s) = \frac{\partial \vartheta_s}{\partial s} = \frac{\partial}{\partial s}\big(\operatorname{atan}\big(m(s)\big)\big) = \frac{f_y{}'' \cdot f_x{}' - f_y{}' \cdot f_x{}''}{(f_x')^2 + \big(f_y'\big)^2} \qquad (4.3.2.4)$$

Por lo que se requiere que las ecuaciones paramétricas sean de clase C^2.

Como aplicación se va a calcular la c(s) de una recta y de una circunferencia.

i. Recta

Se tiene una recta definida por $y = m \cdot x + y_0$.

Unas ecuaciones paramétricas de la recta son:

$$x = t \tag{4.3.2.5}$$
$$y = a \cdot t + y_0 \tag{4.3.2.6}$$

Siendo t un parámetro que toma los valores del dominio de x.

El arco de trayectoria recorrido sobre la recta desde el punto (x_1, y_1) hasta el (x_2, y_2) se calcula mediante:

$$s = \sqrt{(y_2 - y_1)^2 + (x_2 - x_1)^2} = \sqrt{a^2 + 1} \cdot (x_2 - x_1) = \sqrt{a^2 + 1} \cdot t \rightarrow t = \frac{s}{\sqrt{a^2 + 1}} \tag{4.3.2.7}$$

Con lo que las ecuaciones paramétricas se transforman en:

$$x = \frac{s}{\sqrt{a^2 + 1}} \tag{4.3.2.8}$$
$$y = a \cdot \frac{s}{\sqrt{a^2 + 1}} + y_0 \tag{4.3.2.9}$$

Calculando las primeras y segundas derivadas con respecto de s, se tiene:

$$x' = \frac{1}{\sqrt{a^2 + 1}} \rightarrow x'' = 0 \tag{4.3.2.10}$$
$$y' = \frac{a}{\sqrt{a^2 + 1}} \rightarrow y'' = 0 \tag{4.3.2.11}$$

Con lo que se obtiene $c(s) = 0$.

ii. Circunferencia

Se tiene una circunferencia definida por $(x - x_c)^2 + (y - y_c)^2 = R^2$, cuyas ecuaciones paramétricas son:

$$x = R \cdot \cos t + x_c \tag{4.3.2.12}$$
$$y = R \cdot \sin t + y_c \tag{4.3.2.13}$$

El arco de la trayectoria recorrida sobre la curva es:

$$s = \pm t \cdot R \rightarrow t = \pm \frac{s}{R} \tag{4.3.2.14}$$

Con lo que las ecuaciones paramétricas se transforman en:

$$x = R \cdot \cos\left(\pm \frac{s}{R}\right) + x_c \tag{4.3.2.15}$$

$$y = R \cdot \sin\left(\pm \frac{s}{R}\right) + y_c \tag{4.3.2.16}$$

Y derivando, se tiene:

$$x' = \pm \sin\left(\pm \frac{s}{R}\right) \rightarrow x'' = \mp \frac{1}{R} \cdot \cos\left(\pm \frac{s}{R}\right) \tag{4.3.2.17}$$

$$y' = \mp \cos\left(\pm \frac{s}{R}\right) \rightarrow y'' = \mp \frac{1}{R} \cdot \sin\left(\pm \frac{s}{R}\right) \tag{4.3.2.18}$$

Luego, se tiene la siguiente curvatura:

$$c(s) = \frac{\mp \frac{1}{R} \cdot \left(\sin^2\left(\pm \frac{s}{R}\right) + \cos^2\left(\pm \frac{s}{R}\right)\right)}{\mp \left(\sin^2\left(\pm \frac{s}{R}\right) + \cos^2\left(\pm \frac{s}{R}\right)\right)} = \pm \frac{1}{R} \tag{4.3.2.19}$$

Lo cual significa que si el ángulo crece (t>0), se tiene $c(s) = \frac{1}{R}$, y si decrece (t<0), se tiene $c(s) = -\frac{1}{R}$, con lo que queda demostrado lo que se dijo al principio del apartado 4.3.2.

4.4.Simplificación del modelo cinemático en base de Frenet

Puesto que el UGV con el que se va a trabajar, tiene el punto de interés (sistema de visión) centrado en su eje, se toma $l_2 = 0$, luego las ecuaciones que representan al sistema se simplifican a:

$$\dot{s} = \frac{u_1}{1 - d \cdot c(s)} \cdot \left[\cos \vartheta_e - \frac{\tan \phi}{L} \cdot l_1 \cdot \sin \vartheta_e\right] \tag{4.4.1}$$

$$\dot{d} = u_1 \cdot \left[\sin \vartheta_e + \frac{\tan \phi}{L} \cdot l_1 \cdot \cos \vartheta_e\right] \tag{4.4.2}$$

$$\dot{\vartheta}_e = \frac{u_1}{L} \cdot \tan \phi - \dot{s} \cdot c(s) \tag{4.4.3}$$

4.5.Ecuación general de las ecuaciones representativas del modelo

A lo largo de éste apartado se han obtenido las ecuaciones que modelan el UGV objeto del proyecto en su movimiento sobre la trayectoria existente en el plano.

Si se echa un vistazo a dichas ecuaciones, se observa que son muy no lineales, puesto que dependen de tangentes, senos y cosenos, y de productos de ellos. Esto va a complicar notablemente el análisis y el diseño del controlador, ya que el sistema no va a trabajar en el entorno de un punto de trabajo fijo, sino que se va a encontrar condiciones de trabajo muy diferentes.

5. SIMULACIÓN DEL UGV

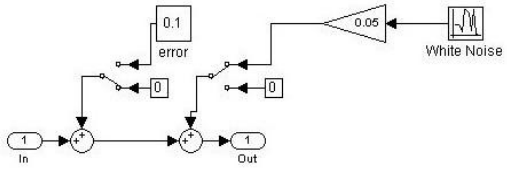
Para llevar a cabo las pruebas necesarias a la hora de diseñar el controlador objeto de este proyecto, es necesario el desarrollo de un programa de simulación corrido bajo un entorno sencillo de usar.

5.1. Utilización de Simulink como entorno de simulación

Se eligió Simulink como entorno de simulación, debido a su extendido uso en el campo del control a nivel universitario y a la sencillez con la que se realizan esquemas de control.

Se han utilizado tres estrategias diferentes para el desarrollo de los bloques de simulación:

- Unión directa de bloques sencillos de Simulink



- Uso de Embedded Matlab Function

- Uso de Level-1 M-file S-function.

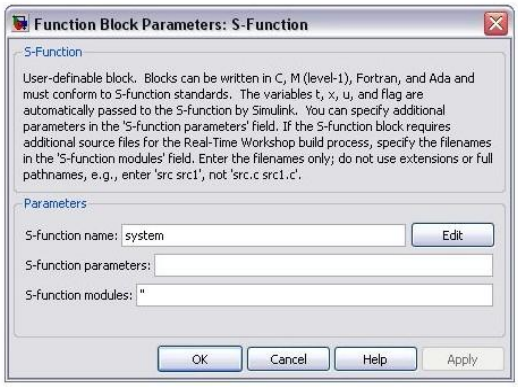


Ilustración 20: Bloques de Simulink usados para la simulación del UGV

En el caso de tener necesidad de desarrollar bloques lo suficientemente sencillos como para poder hacerlo mediante unión de bloques sencillos de Simulink, se prefirió hacerlo de ese modo.

En el caso de tener la necesidad de desarrollar funciones complejas, pero sin ninguna variable que almacene un estado anterior, se han utilizado Embedded Matlab Functions.

Si se tenía que desarrollar un bloque suficientemente complejo, como para resultar imposible hacerlo mediante bloques sencillos de Simulink, y con la necesidad de almacenar variables con estados anteriores, de modo que sea imposible desarrollarlo con Embedded Matlab Function, se han utilizado Level-1 M-file S-functions.

El uso de las Level-1 M-file S-functions está especialmente indicado para la creación de sistemas que se representan mediante un modelo de estado complejo [13]. El flujograma de funcionamiento de dicho bloque es el siguiente:

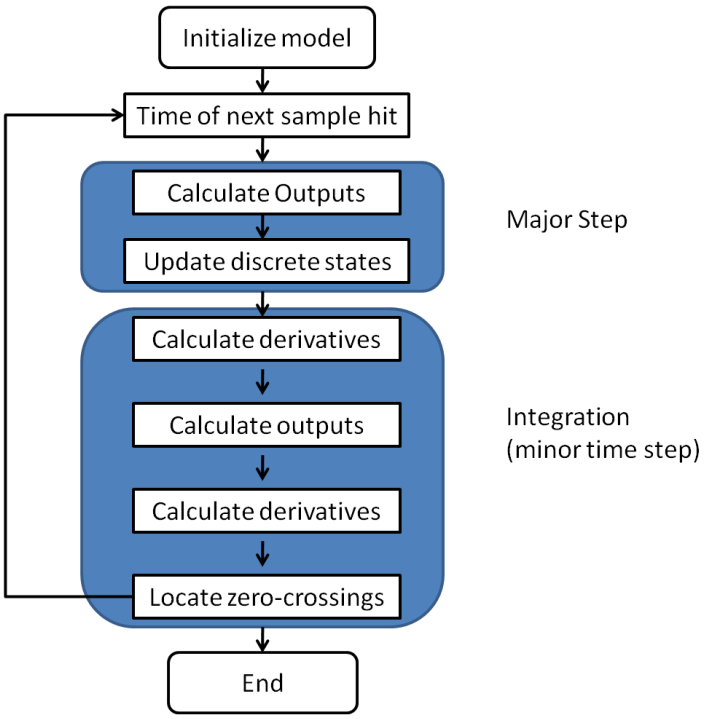


Ilustración 21: Flujograma general de una Level-1 M-file S-function

La fase de Major Step, es la que ejecuta el tiempo discreto, mientras la fase de Integration es la que ejecuta el tiempo continuo.

Efectivamente se puede demostrar la bondad en el uso de s-functions para trabajar con sistemas con un modelo de estado, tomando como ejemplo el siguiente sistema cuyo modelo de estado es:

$$x_{k+1} = A \cdot x_k + B \cdot u_k \quad (5.1.1)$$
$$y_k = C \cdot x_k + D \cdot u_k \quad (5.1.2)$$

La salida y_k se calculará en la fase Calculate outputs del Major step, utilizando un valor del estado almacenado x_k. El estado del paso siguiente x_{k+1} se calculará a continuación en la fase Update discrete states, almacenándose el valor para usarlo en el Calculate outputs del siguiente paso.

Si se quiere realizar un programa habitual con un propósito general, el flujograma necesario sería:

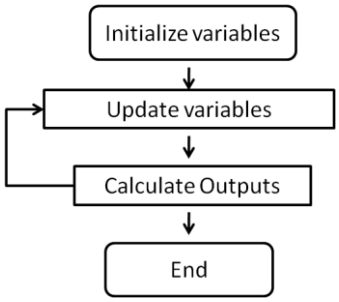


Ilustración 22: Flujograma necesario para programa de propósito general

Sin embargo, el uso de una s-function como un programa habitual con un propósito general, se complica demasiado, y hace que se cometan ciertos errores.

En ese caso, se utilizan los estados discretos de la s-function, como almacén de variables que pueden o no cambiar en cada paso de simulación. Se usará el valor de esas variables para calcular las salidas del programa en cada paso. El flujograma de la s-function será:

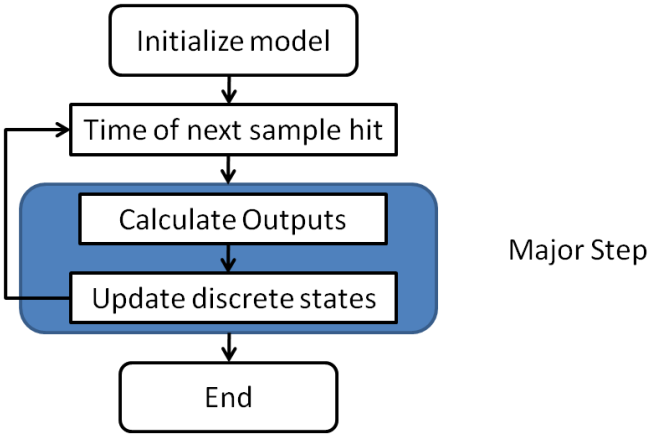


Ilustración 23: Flujograma simplificado de una Level-1 M-file S-function

Y como se observa, se calculan las salidas en el paso k, con el valor de las variables k-1; y a continuación se actualiza el valor de las variables.

Esto creará un retardo que asumiremos y tomaremos como perturbación.

Una forma sencilla pero ineficiente de resolver este problema es calcular la actualización de las variables en ambas fases, usando en el Calculate Outputs el estado cálculo para mostrar las salidas, y actualizando en Update Discrete States el valor de dichas variables. Se utilizará sólo en caso de necesitar exactitud y siempre y cuando los cálculos no sean muy costosos computacionalmente hablando.

5.2.Simulador del sistema básico

Para simular el movimiento del UGV, en particular la distancia a la línea del circuito según se mueve el UGV por el plano mediante unas consignas de velocidad en las ruedas motrices (en metros por segundo) y giro de ruedas directrices (en radianes), con el criterio de signos empleado en el Apartado 4., se crea un módulo en Simulink que representa el sistema real mediante un modelo matemático.

Se pretende crear en Simulink el siguiente bloque:

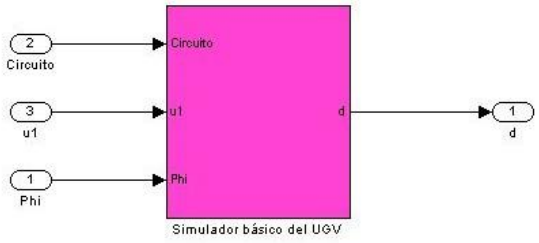


Ilustración 24: Simulador del UGV en Simulink

Esto se puede llevar a cabo de dos maneras que se detallan a continuación.

5.2.1. Simulador en base de Frenet

La primera opción consiste en simular por un lado las ecuaciones cinemáticas en base de Frenet, y por otro lado, un simulador de circuitos válido para dichas ecuaciones, de la siguiente manera:

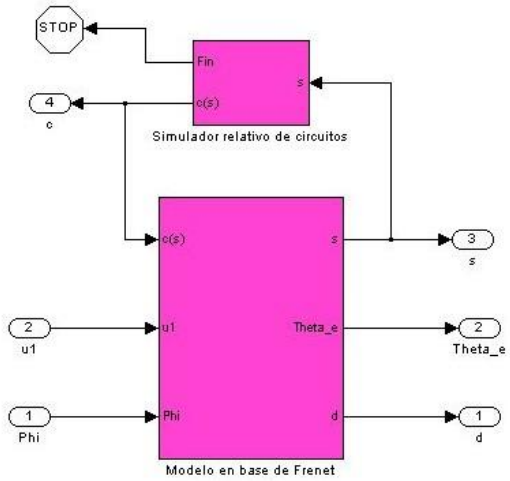


Ilustración 25: Simulador del UGV en base de Frenet

Que está compuesto por dos sub-bloques:

- Modelo en base de Frenet:

En éste primer bloque, se simulan, mediante la unión de bloques elementales de Simulink, las ecuaciones del modelo cinemático en base de Frenet (ecuaciones de 4.3.7 a 4.3.9).

Dichas ecuaciones tienen tres entradas, dos de ellas con carácter de entrada externa, que son u_1 y ϕ, y la otra entrada, que depende del circuito, que es c.

Requiere de unos valores iniciales de $s_0, d_0, \vartheta_{e-0}$ de posicionado y orientación respecto de la curva.

- Simulador relativo de circuitos:

Este segundo bloque trata de calcular el valor de c, partiendo de la información del circuito pintado en el suelo, que se introduce como parámetro en el bloque, y del avance s que se produce sobre el

circuito. Tiene una salida que indica el final del circuito simulado, valiendo 0 en caso de que se siga en el circuito y 1 en caso de que se haya acabado el circuito.

Está programado mediante una S-function, cuyo código y su forma de definir el circuito se puede ver en el Anexo III.

5.2.2. Simulador en el espacio físico

La segunda opción consiste en simular por una parte el movimiento del UGV en el plano mediante su modelo cinemático, y luego, calcular los parámetros del UGV sobre un circuito dibujado en el plano y definido previamente.

Se hace con los siguientes sub-bloques:

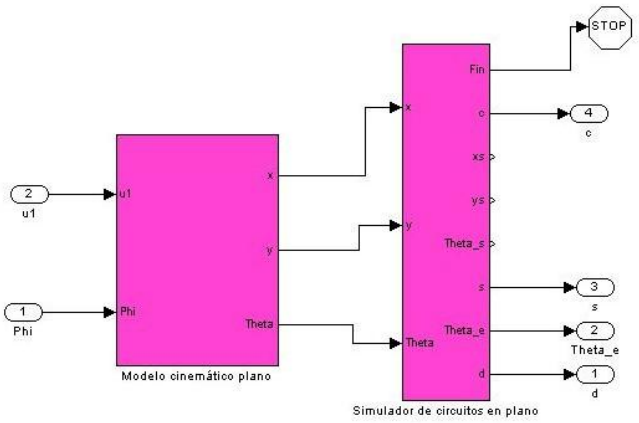


Ilustración 26: Simulador del UGV en el espacio físico

- Modelo cinemático plano:

Este bloque, representa, mediante uniones de bloques elementales de Simulink las ecuaciones del modelo cinemático plano del UGV.

Dichas ecuaciones tienen dos entradas u_1 y ϕ, y tres salidas x, y, ϑ, que es la posición y orientación del UGV en el plano.

Requiere de unos valores iniciales x_0, y_0, ϑ_0 de posicionado y orientación del UGV en el plano.

- Simulador de circuitos en el plano:

Se calcula mediante una S-function, la distancia d principalmente, y secundariamente otros muchos parámetros, que relacionan el UGV con el circuito plano definido matricialmente de la manera que se muestra en el Anexo III.

El código del simulador de circuitos en el plano se puede ver también en el Anexo III.

5.2.3. Comparación de ambos simuladores

Puesto que se tienen dos simuladores desarrollados, conviene hacer una breve comparación de ambos:

Simulador en base de Frenet	Simulador en el espacio físico
Facilidad de uso y definición del circuito	Dificultad de uso y definición del circuito
Robustez debido a la sencillez de código	Poca robustez debido al código largo y complejo
Requiere modelo en base de Frenet	Requiere modelo cinemático
No permite una sencilla representación en el plano	Se puede representar fácilmente en el plano la posición del UGV y del circuito.
No permite incluir relaciones dinámicas de forma sencilla, al estar representado el sistema con el modelo en base de Frenet	Permite incluir en un futuro relaciones dinámicas de forma relativamente sencilla.

Tabla 2: Comparación de simuladores del UGV

Se utilizarán ambos modelos según los objetivos de la simulación.

5.3.Elementos adicionales del sistema real

Con el objeto de dar mayor realismo a la simulación, se añaden al simulador del sistema básico una serie de elementos que se analizan a continuación.

5.3.1. Saturación del giro de ruedas

El UGV real no admite un giro de ruedas que se mueva en el intervalo $-90º < \phi < 90º$, sino que tiene la siguiente limitación física:

$$-26º = \phi_{\max -d} \leq \phi \leq \phi_{\max -i} = 26º$$

Para ello se coloca el siguiente bloque elemental de Simulink, en cascada con el sistema básico, a la entrada de la señal de giro de ruedas directrices:

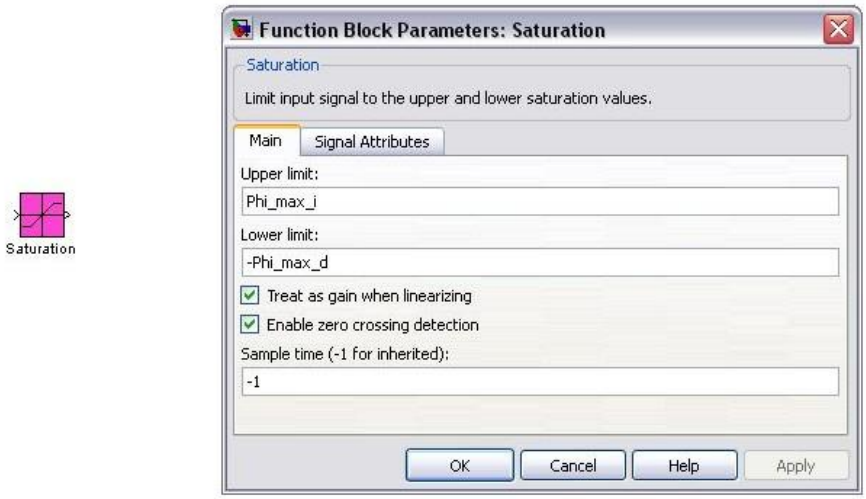


Ilustración 27: Bloque de simulación de la saturación del giro de ruedas

Con este bloque se consigue limitar la acción de giro de la rueda a los valores físicos máximos.

5.3.2. Dinámica del giro de ruedas

El giro de las ruedas del UGV no es instantáneo, sino que tiene una cierta dinámica.

La dinámica del giro de las ruedas será no lineal y dependerá de la velocidad a la que circule el UGV; sin embargo, como primera aproximación, se puede considerar como un sistema de primer orden con un tiempo de establecimiento t_s (hasta alcanzar el 95% del valor en estado estacionario) de 0.05 segundos, y ganancia unidad.

En ese caso, la dinámica de las ruedas se puede representar por:

$$V(s) = \frac{k_v}{1 + T_v \cdot s} \tag{5.3.2.1}$$

El T_v representa el tiempo característico del sistema, es decir, el tiempo que tarda en alcanzar el 63.21% del valor en estado estacionario.

Se calcula T_v a partir de t_s del siguiente modo:

$$T_v = \frac{t_s}{\ln \frac{1}{1 - 0.95}} = \frac{0.05}{\ln 20} = 1.669 \cdot 10^{-2}\ seg \tag{5.3.2.2}$$

Luego el sistema de primer orden que aproxima la dinámica de las ruedas es:

$$V(s) = \frac{1}{1 + 1.669 \cdot 10^{-2} \cdot s} = \frac{59.916}{s + 59.916} \tag{5.3.2.3}$$

Dicho sistema se colocará en cascada con el sistema básico, a la entrada de la señal de giro de ruedas directrices.

Para su simulación en Simulink se utiliza un bloque de función de transferencia elemental:

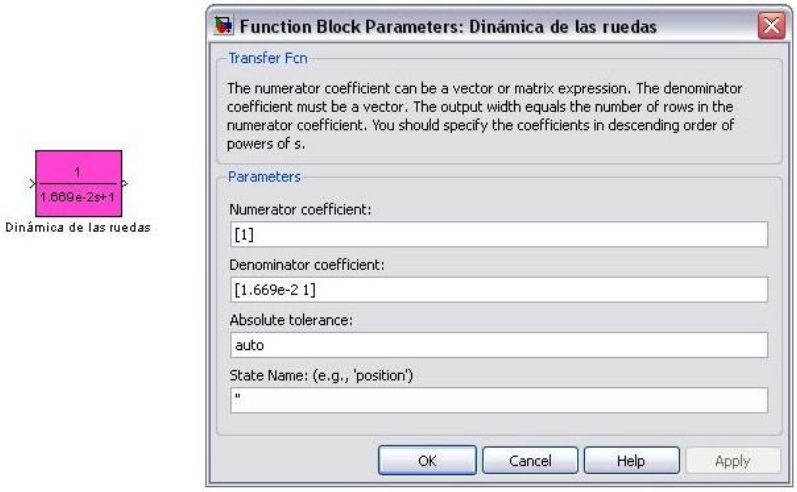


Ilustración 28: Bloque para la simulación de la dinámica del giro de las ruedas

NOTA: En un trabajo futuro se podría modelar de mejor manera la dinámica del giro de ruedas para hacerlo aún más realista.

Del mismo modo, se podría incluir una limitación en la derivada del giro de las ruedas, de acuerdo con la realidad del UGV.

5.3.3. Errores de modelado

Además de la dinámica de las ruedas, y de la saturación en el giro de éstas, hay que considerar que el sistema real no se adapta perfectamente a un modelo matemático como el utilizado en el simulador básico.

Por ello es conveniente incluir en cascada, tanto a la entrada de la señal de control de las ruedas motrices, como a la salida del simulador (distancia d a la curva); unos bloques que incluyan un error de calibración, y un error de precisión.

El primero será un valor constante, mientras que el segundo será un ruido blanco, representado por una distribución normal $N(\mu = 0, \sigma)$, afectado por una ganancia que adapta los valores de la normal a los calculados como error de precisión. Dicho ruido blanco tendrá una frecuencia determinada que variará según las necesidades de la simulación.

Para la simulación se crean dos bloques:

Ilustración 29: Bloques para la simulación de errores

Dentro de cada bloque hay:

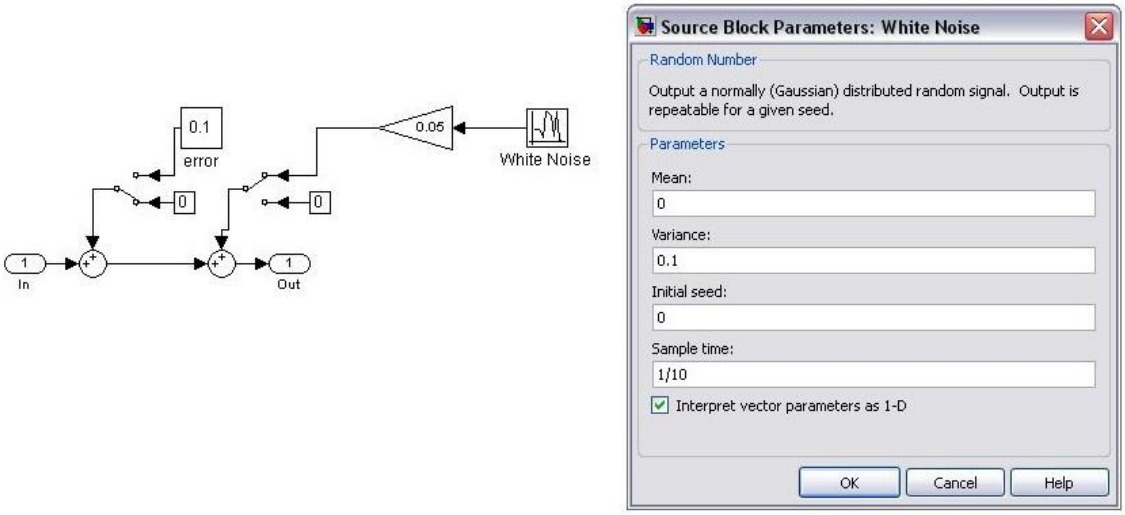


Ilustración 30: Vista interna del bloque de simulación de errores

Con lo que se puede activar o desactivar cada tipo de error, y configurar a placer.

En el caso mostrado en la figura se ha elegido:

Error de calibración	Desactivado. En caso de estar activado tendría un valor de 0.1.
Error de precisión	Activado - Media=0 - Varianza=0.1 - Frecuencia del error: 10 Hz - Ganancia=0.05

Uniendo todos los elementos de simulación del UGV, se tiene una correcta representación de éste, resultando el simulador del UGV completo en Simulink:

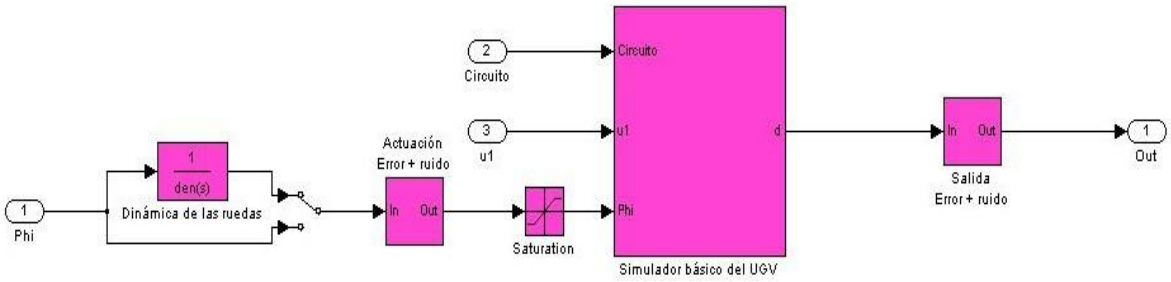


Ilustración 31: Simulador completo del UGV

NOTA:

Se ha incluido en la simulación la posibilidad de desactivar la dinámica de las ruedas puenteándolas en su caso.

6. MODELADO LINEAL Y ANÁLISIS DEL UGV

6.1.Linealización del modelo

En este apartado, se procederá a la linealización de las ecuaciones cinemáticas del modelo en base de Frenet, con la simplificación que se va a utilizar en el sistema a controlar, tal como se vio en el apartado 4.4.

6.1.1. Linealización de las ecuaciones

Se parten de las ecuaciones del modelo cinemático en base de Frenet 4.3.7 a 4.3.9:

$$\dot{s} = \frac{u_1}{1 - d \cdot c(s)} \cdot \left[\cos\vartheta_e - \frac{\tan\phi}{L} \cdot l_1 \cdot \sin\vartheta_e\right] \tag{6.1.1.1}$$

$$\dot{d} = u_1 \cdot \left[\sin\vartheta_e + \frac{\tan\phi}{L} \cdot l_1 \cdot \cos\vartheta_e\right] \tag{6.1.1.2}$$

$$\dot{\vartheta}_e = \frac{u_1}{L} \cdot \tan\phi - \dot{s} \cdot c(s) \tag{6.1.1.3}$$

Sustituyendo la ecuación 6.1.1.1 en la ecuación 6.1.1.3, las tres ecuaciones anteriores, se transforman en las dos siguientes:

$$\dot{d} = u_1 \cdot \left[\sin\vartheta_e + \frac{\tan\phi}{L} \cdot l_1 \cdot \cos\vartheta_e\right] \tag{6.1.1.4}$$

$$\dot{\vartheta}_e = \frac{u_1}{L} \cdot \tan\phi - u_1 \cdot \frac{c(s)}{1 - d \cdot c(s)} \cdot \left[\cos\vartheta_e - \frac{\tan\phi}{L} \cdot l_1 \cdot \sin\vartheta_e\right] \tag{6.1.1.5}$$

Se linealiza en torno a los siguientes puntos de linealización: $u_{1-lin}, \vartheta_{e-lin}, \phi_{lin}, c_{lin}, d_{lin}$, que más tarde, en el Apartado 6.1.2., serán calculados. Se consideran los parámetros l_1 y L constantes y perfectamente definidos.

Las linealización de las ecuaciones es:

$$\Delta\dot{d} = b_1 \cdot \Delta u_1 + b_2 \cdot \Delta\vartheta_e + b_3 \cdot \Delta\phi \tag{6.1.1.6}$$

$$\Delta\dot{\vartheta}_e = b_4 \cdot \Delta u_1 + b_5 \cdot \Delta\phi + b_6 \cdot \Delta\vartheta_e + b_7 \cdot \Delta d + b_8 \cdot \Delta c \tag{6.1.1.7}$$

Siendo:

$$b_1 = \sin\vartheta_{e-lin} + \frac{\tan\phi_{lin}}{L} \cdot l_1 \cdot \cos\vartheta_{e-lin} \tag{6.1.1.8}$$

$$b_2 = u_{1-lin} \cdot \cos\vartheta_{e-lin} - u_{1-lin} \cdot \frac{\tan\phi_{lin}}{L} \cdot l_1 \cdot \sin\vartheta_{e-lin} \tag{6.1.1.9}$$

$$b_3 = u_{1-lin} \cdot \frac{l_1}{L} \cdot \frac{1}{\cos^2\phi_{lin}} \cdot \cos\vartheta_{e-lin} \tag{6.1.1.10}$$

$$b_4 = \frac{\tan\phi_{lin}}{L} - \frac{c_{lin}}{1 - d_{lin} \cdot c_{lin}} \cdot \left(\cos\vartheta_{e-lin} - \frac{l_1}{L} \cdot \tan\phi_{lin} \cdot \sin\vartheta_{e-lin}\right) \tag{6.1.1.11}$$

$$b_5 = \frac{u_{1-lin}}{L} \cdot \frac{1}{\cos^2 \phi_{lin}} \cdot \left(1 + \frac{c_{lin}}{1 - d_{lin} \cdot c_{lin}} \cdot l_1 \cdot \sin \vartheta_{e-lin}\right) \quad (6.1.1.12)$$

$$b_6 = u_{1-lin} \cdot \frac{c_{lin}}{1 - d_{lin} \cdot c_{lin}} \cdot \left(\sin \vartheta_{e-lin} + \frac{\tan \phi_{lin}}{L} \cdot l_1 \cdot \cos \vartheta_{e-lin}\right) \quad (6.1.1.13)$$

$$b_7 = -u_{1-lin} \cdot \frac{c_{lin}^2}{(1 - d_{lin} \cdot c_{lin})^2} \cdot \left(\cos \vartheta_{e-lin} - \frac{\tan \phi_{lin}}{L} \cdot l_1 \cdot \sin \vartheta_{e-lin}\right) \quad (6.1.1.14)$$

$$b_8 = -u_{1-lin} \cdot \frac{1}{(1 - d_{lin} \cdot c_{lin})^2} \cdot \left(\cos \vartheta_{e-lin} - \frac{\tan \phi_{lin}}{L} \cdot l_1 \cdot \sin \vartheta_{e-lin}\right) \quad (6.1.1.15)$$

6.1.2. Puntos de linealización

Se procederá en este apartado a la obtención de los puntos de linealización.

Por un lado, se debe partir de las ecuaciones del modelo cinemático en base de Frenet simplificadas, aplicadas a los puntos de linealización, de la siguiente manera:

$$\dot{d}_{lin} = u_{1-lin} \cdot \left[\sin \vartheta_{e-lin} + \frac{\tan \phi_{lin}}{L} \cdot l_1 \cdot \cos \vartheta_{e-lin}\right] \quad (6.1.2.1)$$

$$\dot{\vartheta}_{e-lin} = \frac{u_{1-lin}}{L} \cdot \tan \phi_{lin} - u_{1-lin} \cdot \frac{c_{lin}}{1 - d_{lin} \cdot c_{lin}} \cdot \left[\cos \vartheta_{e-lin} - \frac{\tan \phi_{lin}}{L} \cdot l_1 \cdot \sin \vartheta_{e-lin}\right] \quad (6.1.2.2)$$

Por otra parte, se definen por hipótesis los siguientes puntos de linealización:

- $d_{lin} = 0$, ya que, en el entorno de trabajo del sistema siempre estaremos trabajando en las proximidades de la línea ($d_{ref} = 0$).
- $u_{1-lin} = u_{1-lin}(t)$, que será un valor conocido en cada instante de tiempo t, y que en general será variable en el tiempo.
- $c_{lin} = c_{lin}(t)$, que puede ser un valor conocido o no y que normalmente variará a lo largo del tiempo.

Para el cálculo de los otros dos puntos de linealización $\vartheta_{e-lin}, \phi_{lin}$, se usarán las ecuaciones del modelo cinemático en base de Frenet simplificadas, aplicadas a los puntos de linealización, y combinadas con ciertas hipótesis, según se quiera obtener la evolución temporal exacta de dichos puntos, o simplemente el valor final.

6.1.2.1. Puntos de linealización sin dinámica. Valor final de los puntos de linealización.

Como primer cálculo aproximado, se determinarán los puntos de linealización $\vartheta_{e-lin}, \phi_{lin}$ sin ningún tipo de dinámica, es decir, se obtendrá el valor final de dichos puntos de linealización, conocidos los otros puntos de linealización $u_{1-lin}, c_{lin}, d_{lin}$

Se utilizan las siguientes hipótesis:

$$\dot{d}_{lin} = 0 \quad (6.1.2.1.1)$$

$$\dot{\vartheta}_{e-lin} = 0 \quad (6.1.2.1.2)$$

Con lo que las ecuaciones 6.1.2.1 y 6.1.2.2 resultan:

$$0 = u_{1-lin} \cdot \left[\sin \vartheta_{e-lin} + \frac{\tan \phi_{lin}}{L} \cdot l_1 \cdot \cos \vartheta_{e-lin}\right] \quad (6.1.2.1.3)$$

$$0 = \frac{u_{1-lin}}{L} \cdot \tan \phi_{lin} - u_{1-lin} \cdot \frac{c_{lin}}{1 - d_{lin} \cdot c_{lin}} \cdot \left[\cos \vartheta_{e-lin} - \frac{\tan \phi_{lin}}{L} \cdot l_1 \cdot \sin \vartheta_{e-lin}\right] \quad (6.1.2.1.4)$$

Teniendo dos ecuaciones para dos incógnitas.

En el caso en el que $u_{1-lin} \neq 0$ y $|\vartheta_{e-lin}| < 90º$, lo cual es totalmente razonable y está dentro de las condiciones de trabajo, se obtiene de la primera ecuación:

$$\tan \phi_{lin} = -\frac{L}{l_1} \cdot \tan \vartheta_{e-lin} \rightarrow \phi_{lin} = \operatorname{atan}\left(-\frac{L}{l_1} \cdot tan\, \vartheta_{e-lin}\right) \quad (6.1.2.1.5)$$

Con lo que se obtiene el valor de ϕ_{lin} conocido el valor de ϑ_{e-lin}

Sustituyendo la ecuación 6.1.2.1.5 en la ecuación 6.1.2.1.4, se obtiene:

$$0 = \frac{u_{1-lin}}{\cancel{L}} \cdot \left(-\frac{\cancel{L}}{l_1} \cdot \tan \vartheta_{e-lin}\right) - u_{1-lin} \cdot \frac{c_{lin}}{1 - d_{lin} \cdot c_{lin}} \cdot \left[\cos \vartheta_{e-lin} - \frac{\cancel{l_1}}{\cancel{L}} \cdot \left(-\frac{\cancel{L}}{\cancel{l_1}} \cdot \tan \vartheta_{e-lin}\right) \cdot \sin \vartheta_{e-lin}\right] \quad (6.1.2.1.6)$$

Y en el caso de que $u_{1-lin} \neq 0$ y $|\vartheta_{e-lin}| < 90º$, se obtiene:

$$\sin \vartheta_{e-lin} = -\frac{c_{lin}}{1 - d_{lin} \cdot c_{lin}} \cdot l_1 \rightarrow \vartheta_{e-lin} = -\operatorname{asin}\left(\frac{c_{lin}}{1 - d_{lin} \cdot c_{lin}} \cdot l_1\right) \quad (6.1.2.1.7)$$

Y haciendo $d_{lin} = 0$, se simplifica a:

$$\boxed{\vartheta_{e-lin} = -\operatorname{asin}(c_{lin} \cdot l_1)} \quad (6.1.2.1.8)$$

Y sustituyendo en la ecuación 6.1.2.1.5 se tiene:

$$\boxed{\phi_{lin} = \operatorname{atan}\left(\frac{L}{l_1} \cdot tan(asin(c_{lin} \cdot l_1))\right)} \quad (6.1.2.1.9)$$

6.1.2.2. Puntos de linealización con dinámica

En el apartado 6.1.2.1, se han obtenido los puntos de linealización $\vartheta_{e-lin}, \phi_{lin}$ sin ningún tipo de dinámica. Esto significa que ante una variación de uno de los puntos de linealización conocidos $u_{1-lin}, c_{lin}, d_{lin}$, se tendrá una variación de los puntos de linealización calculados sin ningún tipo

de evolución continua y suave, es decir, bruscamente, lo cual supondrá un inconveniente en la parte de control, como se analizará en el apartado 7.2.2.

Para el cálculo de los puntos de linealización $\vartheta_{e-lin}, \phi_{lin}$ con dinámica, se partirá de la siguiente hipótesis:

$$\dot{d}_{lin} = 0 \tag{6.1.2.2.1}$$

Con lo que, de forma similar al apartado anterior, y bajo las mismas condiciones de funcionamiento, se obtiene:

$$\tan \phi_{lin} = -\frac{L}{l_1} \cdot \tan \vartheta_{e-lin} \rightarrow \boxed{\phi_{lin} = \text{atan}\left(-\frac{L}{l_1} \cdot tan\, \vartheta_{e-lin}\right)} \tag{6.1.2.2.2}$$

Obteniéndose ϕ_{lin}, una vez sea conocido ϑ_{e-lin}.

Por otro lado, la ecuación 6.1.2.2 ya no se anulará, sino que será una ecuación diferencial que tendrá que ser resuelta.

Sustituyendo la ecuación 6.1.2.2.2 en la ecuación 6.1.2.2, se obtiene la siguiente ecuación diferencial no lineal de primer orden:

$$\dot{\vartheta}_{e-lin} = -\frac{u_{1-lin}}{l_1} \cdot \tan \vartheta_{e-lin} - u_{1-lin} \cdot \frac{c_{lin}}{1 - d_{lin} \cdot c_{lin}} \cdot \frac{1}{\cos \vartheta_{e-lin}} \tag{6.1.2.2.3}$$

Que se simplifica, al sustituir el punto de linealización $d_{lin} = 0$, de la siguiente manera:

$$\dot{\vartheta}_{e-lin} = -\frac{u_{1-lin}}{l_1} \cdot \tan \vartheta_{e-lin} - u_{1-lin} \cdot c_{lin} \cdot \frac{1}{\cos \vartheta_{e-lin}} \tag{6.1.2.2.4}$$

Y que para su resolución se debe utilizar la siguiente condición inicial, calculada a partir de la ecuación 6.1.2.1.8:

$$\vartheta_{e-lin-ini} = -\,\text{asin}(c_{lin-ini} \cdot l_1) \tag{6.1.2.2.5}$$

La anterior ecuación diferencial se puede resolver de dos maneras (ver Anexo IV):

- <u>Resolución analítica</u>

Se resuelve analíticamente con Matlab, considerando $u_{1-lin} = cte$, obteniéndose:

$$\boxed{\vartheta_{e-lin}(t) = \text{asin}\left(e^{-\frac{u_{1-lin}}{l_1}t} \cdot (c_{lin} - c_{lin-ini}) \cdot l_1 - c_{lin} \cdot l_1\right)} \tag{6.1.2.2.6}$$

- Resolución aproximada

Se resuelve de forma aproximada, suponiendo que se comporta como un sistema de primer orden, y considerando $u_{1-lin} = cte$, obteniéndose:

$$\vartheta_{e-lin}(t) = -\text{asin}(c_{\text{lin}} \cdot l_1) \cdot \left(1 - e^{-\frac{u_{1-lin}}{l_1} t}\right) - \text{asin}(c_{lin-ini} \cdot l_1) \cdot e^{-\frac{u_{1-lin}}{l_1} t} \qquad (6.1.2.2.7)$$

En la resolución por ambos métodos se ha considerado $u_{1-lin} = cte$ en cada instante de tiempo por sencillez.

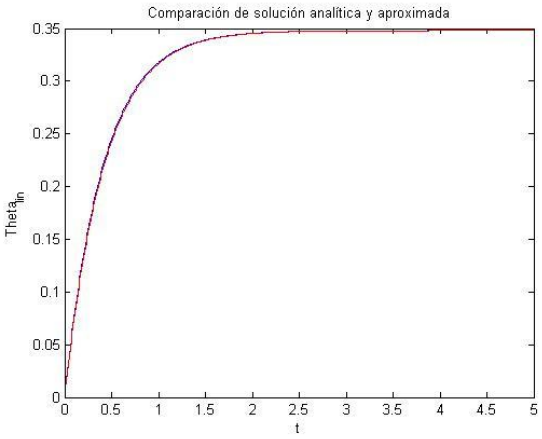
Se puede comparar la evolución temporal de ambas soluciones al producirse un cambio en c_{lin} obteniéndose como conclusión, que tanto la resolución analítica como la aproximada representan adecuadamente la dinámica de ϑ_{e-lin}.

Se muestran varios ejemplos, donde aparece en azul la solución analítica, y en rojo la solución aproximada:

- *Ejemplo 1*

l_1	3.41 m
u_{1-lin}	30 km/h = 8.333 m/s
$c_{lin-ini}$	0
c_{lin}	-0.1

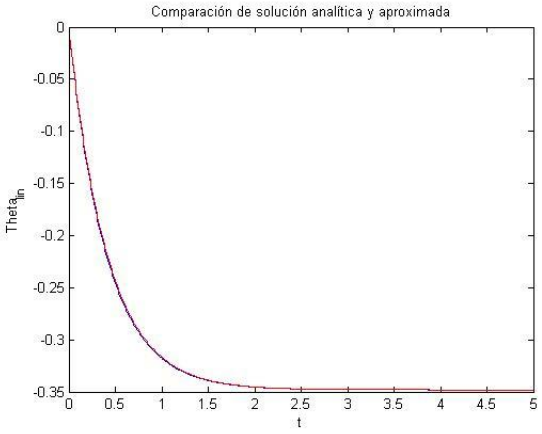
Error medio cu drático	0.0213



- *Ejemplo 2*

l_1	3.41 m
u_{1-lin}	30 km/h = 8.333 m/s
$c_{lin-ini}$	0
c_{lin}	0.1

Error medio cuadrático	0.0213



- *Ejemplo 3*

l_1	.41 m
u_{1-lin}	10 km/h = 2.778 m/s
$c_{lin-ini}$	-0.1
c_{lin}	0.1

Error medio cuadrático	0.0407

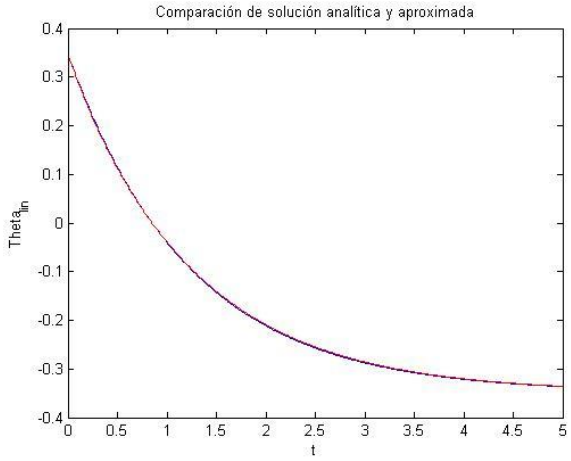


Tabla 3: Comparación de resolución de ecuaciones del punto de linealización

De las ecuaciones 6.1.2.2.6 y 6.1.2.2.7, se deduce que tanto ϑ_{e-lin} como ϕ_{lin} únicamente evolucionan con una cierta dinámica, si se produce un cambio en c_{lin}. En caso de que se produzca un cambio aislado en u_{1-lin}, no se modificarán los puntos de linealización calculados, sólo la constante de tiempo ante un cambio en c_{lin}.

6.1.3. Coeficientes del modelo linealizado

Se estudiarán en este apartado los valores de b_i de las ecuaciones representativas del modelo linealizado, teniendo en cuenta el valor de los puntos de linealización.

En primer lugar, se hace $d_{lin} = 0$, con lo que se obtiene:

$$b_1 = \sin\vartheta_{e-lin} + \frac{\tan\phi_{lin}}{L} \cdot l_1 \cdot \cos\vartheta_{e-lin} \quad (6.1.3.1)$$

$$b_2 = u_{1-lin} \cdot \cos\vartheta_{e-lin} - u_{1-lin} \cdot \frac{\tan\phi_{lin}}{L} \cdot l_1 \cdot \sin\vartheta_{e-lin} \quad (6.1.3.2)$$

$$b_3 = u_{1-lin} \cdot \frac{l_1}{L} \cdot \frac{1}{\cos^2\phi_{lin}} \cdot \cos\vartheta_{e-lin} \quad (6.1.3.3)$$

$$b_4 = \frac{\tan\phi_{lin}}{L} - c_{lin} \cdot \left(\cos\vartheta_{e-lin} - \frac{l_1}{L} \cdot \tan\phi_{lin} \cdot \sin\vartheta_{e-lin}\right) \quad (6.1.3.4)$$

$$b_5 = \frac{u_{1-lin}}{L} \cdot \frac{1}{\cos^2\phi_{lin}} \cdot (1 + c_{lin} \cdot l_1 \cdot \sin\vartheta_{e-lin}) \quad (6.1.3.5)$$

$$b_6 = u_{1-lin} \cdot c_{lin} \cdot \left(\sin\vartheta_{e-lin} + \frac{\tan\phi_{lin}}{L} \cdot l_1 \cdot \cos\vartheta_{e-lin}\right) \quad (6.1.3.6)$$

$$b_7 = -u_{1-lin} \cdot c_{lin}^2 \cdot \left(\cos\vartheta_{e-lin} - \frac{\tan\phi_{lin}}{L} \cdot l_1 \cdot \sin\vartheta_{e-lin}\right) \quad (6.1.3.7)$$

$$b_8 = -u_{1-lin} \cdot \left(\cos\vartheta_{e-lin} - \frac{\tan\phi_{lin}}{L} \cdot l_1 \cdot \sin\vartheta_{e-lin}\right) \quad (6.1.3.8)$$

Incluyendo la condición que se obtiene de hacer $\dot{d}_{lin} = 0$, es decir:

$$u_{1-lin} \cdot \left[\sin\vartheta_{e-lin} + \frac{\tan\phi_{lin}}{L} \cdot l_1 \cdot \cos\vartheta_{e-lin}\right] = 0 \quad (6.1.3.9)$$

Se simplifican aún más los b_i, resultando:

$$b_1 = \sin\vartheta_{e-lin} + \frac{\tan\phi_{lin}}{L}\cdot l_1 \cdot \cos\vartheta_{e-lin} = 0 \tag{6.1.3.10}$$

$$b_2 = u_{1-lin}\cdot\left(\cos\vartheta_{e-lin} - \frac{\tan\phi_{lin}}{L}\cdot l_1\cdot\sin\vartheta_{e-lin}\right) \tag{6.1.3.11}$$

$$b_3 = u_{1-lin}\cdot\frac{l_1}{L}\cdot\frac{1}{\cos^2\phi_{lin}}\cdot\cos\vartheta_{e-lin} \tag{6.1.3.12}$$

$$\begin{aligned} b_4 &= \frac{\tan\phi_{lin}}{L} - c_{lin}\cdot\left(\cos\vartheta_{e-lin} - \frac{l_1}{L}\cdot\tan\phi_{lin}\cdot\sin\vartheta_{e-lin}\right) \\ &= \frac{\tan\phi_{lin}}{L} - \frac{c_{lin}}{u_{1-lin}}\cdot b_2 \end{aligned} \tag{6.1.3.13}$$

$$b_5 = \frac{u_{1-lin}}{L}\cdot\frac{1}{\cos^2\phi_{lin}}\cdot(1 + c_{lin}\cdot l_1\cdot\sin\vartheta_{e-lin}) \tag{6.1.3.14}$$

$$b_6 = u_{1-lin}\cdot c_{lin}\cdot\left(\sin\vartheta_{e-lin} + \frac{\tan\phi_{lin}}{L}\cdot l_1\cdot\cos\vartheta_{e-lin}\right) = 0 \tag{6.1.3.15}$$

$$b_7 = -u_{1-lin}\cdot c_{lin}^2\cdot\left(\cos\vartheta_{e-lin} - \frac{\tan\phi_{lin}}{L}\cdot l_1\cdot\sin\vartheta_{e-lin}\right) = c_{lin}^2\cdot b_8 \tag{6.1.3.16}$$

$$b_8 = -u_{1-lin}\cdot\left(\cos\vartheta_{e-lin} - \frac{\tan\phi_{lin}}{L}\cdot l_1\cdot\sin\vartheta_{e-lin}\right) = -b_2 \tag{6.1.3.17}$$

Sustituyendo ahora $\phi_{lin} = \operatorname{atan}\left(-\frac{L}{l_1}\cdot tan\,\vartheta_{e-lin}\right)$ en los coeficientes b_i, se obtiene el valor de los b_i, dependiendo exclusivamente de $u_{1-lin}, c_{lin}\ y\ \vartheta_{e-lin}$:

$$b_1 = 0 \tag{6.1.3.18}$$

$$b_2 = \frac{u_{1-lin}}{\cos\vartheta_{e-lin}} \tag{6.1.3.19}$$

$$b_3 = u_{1-lin}\cdot\frac{l_1}{L}\cdot\frac{1}{\cos^2\left(\operatorname{atan}\left(-\frac{L}{l_1}\cdot\tan\vartheta_{e-lin}\right)\right)}\cdot\cos\vartheta_{e-lin} \tag{6.1.3.20}$$

$$b_4 = -\frac{1}{\cos\vartheta_{e-lin}}\cdot\left(\frac{\sin\vartheta_{e-lin}}{l_1} + c_{lin}\right) \tag{6.1.3.21}$$

$$b_5 = \frac{u_{1-lin}}{L}\cdot\frac{1}{\cos^2\left(\operatorname{atan}\left(-\frac{L}{l_1}\cdot\tan\vartheta_{e-lin}\right)\right)}\cdot(1 + c_{lin}\cdot l_1\cdot\sin\vartheta_{e-lin}) \tag{6.1.3.22}$$

$$b_6 = 0 \tag{6.1.3.23}$$

$$b_7 = -c_{lin}^2\cdot\frac{u_{1-lin}}{\cos\vartheta_{e-lin}} \tag{6.1.3.24}$$

$$b_8 = -\frac{u_{1-lin}}{\cos\vartheta_{e-lin}} \tag{6.1.3.25}$$

Luego, el modelo linealizado, se transforma en:

$$\Delta\dot{d} = b_2\cdot\Delta\vartheta_e + b_3\cdot\Delta\phi \tag{6.1.3.26}$$

$$\Delta\dot{\vartheta}_e = b_4\cdot\Delta u_1 + b_5\cdot\Delta\phi + b_7\cdot\Delta d + b_8\cdot\Delta c \tag{6.1.3.27}$$

6.2.Función de transferencia continua del modelo linealizado

Se tratará en este apartado de calcular la función de transferencia $G(s) = \frac{\Delta d}{\Delta \phi}$ que representa el sistema con el que se está trabajando. Será objetivo de este apartado analizar la estabilidad y la posición de los polos y ceros del sistema.

6.2.1. Definición de la función de transferencia continua del modelo linealizado

Partiendo de las ecuaciones del modelo linealizado, simplificadas (ecuaciones 6.1.3.26 y 6.1.3.27), se puede calcular la función de transferencia continua que representa al sistema del siguiente modo:

$$\Delta \dot{d} = s \cdot \Delta d = b_2 \cdot \Delta \vartheta_e + b_3 \cdot \Delta \phi \quad (6.2.1.1)$$

$$\Delta \dot{\vartheta_e} = s \cdot \Delta \vartheta_e = b_4 \cdot \Delta u_1 + b_5 \cdot \Delta \phi + b_7 \cdot \Delta d + b_8 \cdot \Delta c \quad (6.2.1.2)$$

Sustituyendo la segunda ecuación en la primera, se obtiene:

$$s \cdot \Delta d = b_2 \cdot b_4 \cdot \frac{1}{s} \cdot \Delta u_1 + b_2 \cdot b_7 \cdot \frac{1}{s} \cdot \Delta d + b_2 \cdot b_8 \cdot \frac{1}{s} \cdot \Delta c + (b_2 \cdot b_5 \cdot \frac{1}{s} + b_3) \cdot \Delta \phi \quad (6.2.1.3)$$

Y reordenando, se tiene:

$$\begin{aligned} (s - b_2 \cdot b_7 \cdot \frac{1}{s}) \cdot \Delta d &= b_2 \cdot b_4 \cdot \frac{1}{s} \cdot \Delta u_1 + b_2 \cdot b_8 \cdot \frac{1}{s} \cdot \Delta c + (b_2 \cdot b_5 \cdot \frac{1}{s} + b_3) \cdot \Delta \phi \\ (s^2 - b_2 \cdot b_7) \cdot \Delta d &= b_2 \cdot b_4 \cdot \Delta u_1 + b_2 \cdot b_8 \cdot \Delta c + (b_2 \cdot b_5 + b_3 \cdot s) \cdot \Delta \phi \end{aligned} \quad (6.2.1.4)$$

Obteniéndose:

$$\begin{aligned} \Delta d &= b_3 \cdot \frac{s + \frac{b_2 \cdot b_5}{b_3}}{s^2 - b_2 \cdot b_7} \cdot \Delta \phi + \frac{b_2 \cdot b_4}{s^2 - b_2 \cdot b_7} \cdot \Delta u_1 + \frac{b_2 \cdot b_8}{s^2 - b_2 \cdot b_7} \cdot \Delta c \\ &= A_1 \cdot \frac{s + A_2}{s^2 + A_3} \cdot \Delta \phi + \frac{A_4}{s^2 + A_3} \cdot \Delta \mathrm{u}_1 + \frac{A_5}{s^2 + A_3} \cdot \Delta \mathrm{c} \\ &= G(s) \cdot \Delta \phi + P_{u_1}(s) \cdot \Delta u_1 + P_c(s) \cdot \Delta c \end{aligned} \quad (6.2.1.5)$$

Donde, la función de transferencia continua que representa el sistema es:

$$\boxed{\mathrm{G(s)} = \frac{\Delta d}{\Delta \phi} = A_1 \cdot \frac{s + A_2}{s^2 + A_3}} \quad (6.2.1.6)$$

Y las funciones de transferencia continuas que representan perturbaciones son:

$$P_{u_1}(s) = \frac{\Delta d}{\Delta u_1} = \frac{A_4}{s^2 + A_3} \quad (6.2.1.7)$$

$$P_c(s) = \frac{\Delta d}{\Delta c} = \frac{A_5}{s^2 + A_3} \tag{6.2.1.8}$$

Siendo:

$$A_1 = b_3 = u_{1-lin} \cdot \frac{l_1}{L} \cdot \frac{1}{\cos^2\left(\operatorname{atan}\left(-\frac{L}{l_1} \cdot \tan \vartheta_{e-lin}\right)\right)} \cdot \cos \vartheta_{e-lin} \tag{6.2.1.9}$$

$$A_2 = \frac{b_2 \cdot b_5}{b_3} = \frac{u_{1-lin}}{l_1 \cdot \cos^2 \vartheta_{e-lin}} \cdot (1 + c_{lin} \cdot l_1 \cdot \sin \vartheta_{e-lin}) \tag{6.2.1.10}$$

$$A_3 = -b_2 \cdot b_7 = c_{lin}^2 \cdot \frac{u_{1-lin}^2}{\cos^2 \vartheta_{e-lin}} \tag{6.2.1.11}$$

$$A_4 = b_2 \cdot b_4 = -u_{1-lin} \cdot \frac{1}{\cos^2 \vartheta_{e-lin}} \cdot \left(\frac{\sin \vartheta_{e-lin}}{l_1} + c_{lin}\right) \tag{6.2.1.12}$$

$$A_5 = b_2 \cdot b_8 = -\frac{u_{1-lin}^2}{\cos^2 \vartheta_{e-lin}} \tag{6.2.1.13}$$

La representación gráfica del sistema linealizado y simplificado, mediante funciones de transferencia es la que se muestra a continuación:

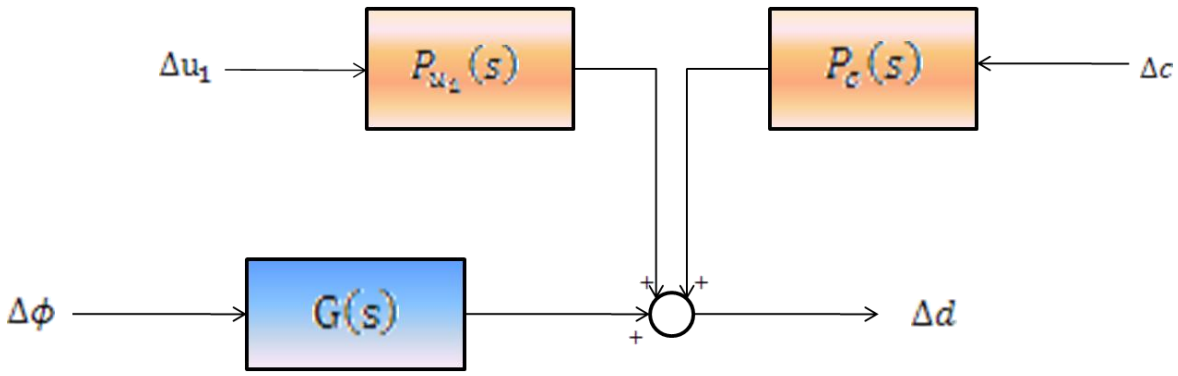


Ilustración 32: Representación del UGV linealizado mediante funciones de transferencia

6.2.2. Representación del sistema real mediante funciones de transferencia

En el apartado anterior, se ha conseguido linealizar las ecuaciones del modelo y obtener las funciones de transferencia que lo representan, con el objetivo de trabajar más fácilmente con el sistema.

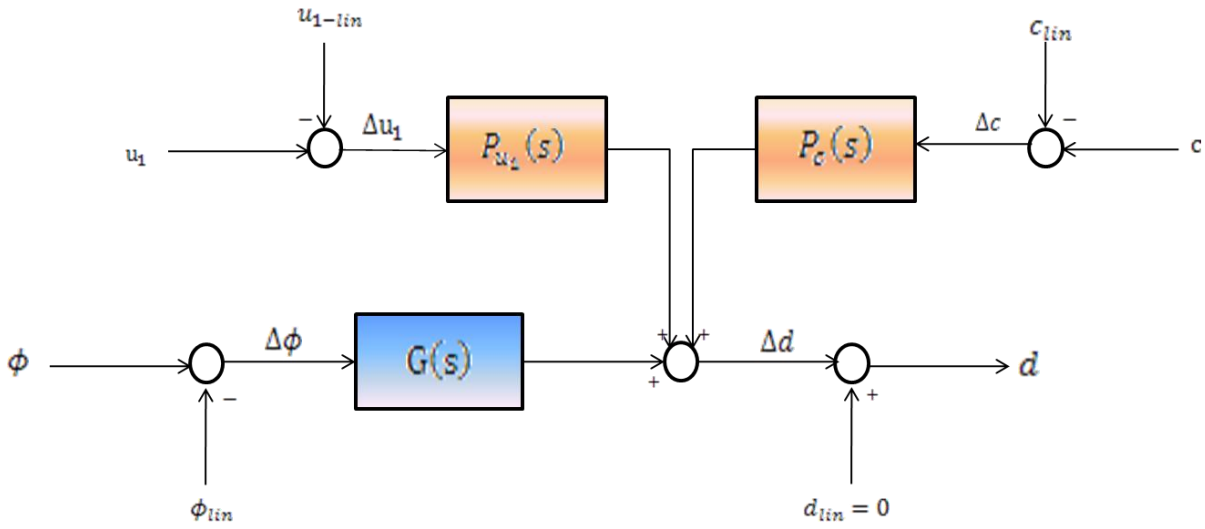


Ilustración 33: Representación del UGV linealizado mediante funciones de transferencia sumando puntos de linealización

Existen tres opciones a la hora de trabajar con la función de transferencia del sistema.

La primera opción es elegir unos puntos de linealización constantes:

$$u_{1-lin} = cte \quad (6.2.2.1)$$
$$c_{lin} = cte \quad (6.2.2.2)$$
$$\vartheta_{e-lin} = cte \rightarrow \phi_{lin} = cte \quad (6.2.2.3)$$

Luego, las funciones de transferencia $G(s), P_{u_1}(s)$ y $P_c(s)$ que representan al sistema, tendrán coeficientes constantes; y al alejarse de los puntos de linealización, las funciones de transferencia, representarán peor el comportamiento del sistema real.

En este caso, se tendrá en general:

$$\Delta u_1 = u_1 - u_{1-lin} \neq 0 \quad (6.2.2.4)$$
$$\Delta c = c - c_{lin} \neq 0 \quad (6.2.2.5)$$

Luego las funciones de transferencia que representan perturbaciones afectarán a la distancia d, y por lo tanto, la función de transferencia G no modelizará adecuadamente el sistema. El sistema vendrá dado por:

$$\Delta d = G(s) \cdot \Delta\phi + P_{u_1}(s) \cdot \Delta u_1 + P_c(s) \cdot \Delta c \quad (6.2.2.6)$$

La segunda opción es elegir los puntos de linealización variables del siguiente modo:

$$u_{1-lin} = u_1 \quad (6.2.2.7)$$
$$c_{lin} = c \quad (6.2.2.8)$$
$$\vartheta_{e-lin} \approx \vartheta_{e-lin}(t) \rightarrow \phi_{lin} \approx \phi_{lin}(t) \quad (6.2.2.9)$$

Ahora, los coeficientes de las funciones de transferencia serán variables, y representarán mejor al sistema, aunque las condiciones de trabajo sean variables.

Además, se tiene:

$$\Delta u_1 = u_1 - u_{1-lin} = 0 \quad (6.2.2.10)$$
$$\Delta c = c - c_{lin} = 0 \quad (6.2.2.11)$$

Con lo que se consigue que el sistema se simplifique a:

$$\Delta d = G(s) \cdot \Delta\phi \quad (6.2.2.12)$$

Nota:

Si se resuelve de forma exacta la ecuación diferencial de ϑ_{e-lin}, se tendrá $\vartheta_{e-lin} = \vartheta_{e-lin}(t)$, y por lo tanto $\phi_{lin} = \phi_{lin}(t)$, con lo que se tendrá, teóricamente $\Delta\phi = \phi - \phi_{lin} = 0$.

En este caso se conseguiría, teóricamente que $d = d_{lin} = 0$.

No obstante, debido a las imprecisiones del modelo, esto no será así, y se tendrán ciertas perturbaciones $d_{pert} \neq 0$, que harán que se tenga que ejercer una acción de control en ϕ, lo que provocará que $\Delta\phi \neq 0$.

La tercera opción es un caso intermedio de ambas. En el caso en que se desconozca el valor de c, habrá que linealizar el sistema para un

$$c_{lin} = cte \quad (6.2.2.13)$$

Pero se puede elegir:

$$u_{1-lin} = u_1 \quad (6.2.2.14)$$

Ahora, debido a que se desconoce el valor de c, se tendrá que linealizar en torno a:

$$\vartheta_{e-lin} = cte \rightarrow \phi_{lin} = cte \quad (6.2.2.15)$$

En este caso, los coeficientes de las funciones de transferencia que representan al sistema serán variables, pero también dependerán del punto de linealización, y al alejarse de éste, el comportamiento será peor.

Se tendrá ahora:

$$\Delta u_1 = u_1 - u_{1-lin} = 0 \quad (6.2.2.16)$$
$$\Delta c = c - c_{lin} \neq 0 \quad (6.2.2.17)$$

Y el sistema vendrá representado por:

$$\Delta \mathrm{d} = G(s) \cdot \Delta\phi + P_c(s) \cdot \Delta c \qquad (6.2.2.18)$$

Con lo que la función de transferencia G no representará adecuadamente el sistema.

Se utilizará, siempre que sea posible la segunda opción de trabajo (y en su caso la tercera opción), puesto que aunque los puntos de linealización varían, y por lo tanto varía la función de transferencia G(s), lo cual incrementa la dificultad; es lo más adecuado, ya que el sistema no va a trabajar en el entorno de un punto de linealización constante.

6.2.3. Análisis de la función de transferencia continua del modelo linealizado

Se analizará ahora la función de transferencia G(s) del sistema en cadena abierta, calculando los polos y ceros, y estudiando la estabilidad.

6.2.3.1. Polos y ceros

Se calculará la posición de los polos y los ceros del sistema G(s).

- Polos

El sistema tiene dos polos, que vienen dados por:

$$p_{1,2} = \pm\sqrt{-A_3} = \pm\sqrt{b_2 \cdot b_7} = \pm\sqrt{-c_{lin}^2 \cdot \frac{u_{1-lin}^2}{\cos^2 \vartheta_{e-lin}}} = \pm\left| c_{lin} \cdot \frac{u_{1-lin}}{\cos \vartheta_{e-lin}} \right| \cdot i \qquad (6.2.3.1.1)$$

Luego se tienen dos polos imaginarios puros, complejos conjugados cuya posición en el eje imaginario depende del punto de linealización.

Si se elige el valor de ϑ_{e-lin}, sin tener en cuenta la dinámica, es decir, se toma:

$$\vartheta_{e-lin} = \operatorname{asin}(-c_{lin} \cdot l_1) = \operatorname{acos}\sqrt{1 - c_{lin}^2 \cdot l_1^2} \qquad (6.2.3.1.2)$$

Resulta:

$$p_{1,2} = \pm c_{lin} \cdot \frac{u_{1-lin}}{\sqrt{1 - c_{lin}^2 \cdot l_1^2}} \cdot i \qquad (6.2.3.1.3)$$

Que será adonde tenderán los polos ante un cambio en los puntos de linealización.

- Ceros

El sistema tiene un cero que viene dado por:

$$z_1 = -A_2 = -\frac{u_{1-lin}}{l_1 \cdot \cos^2 \vartheta_{e-lin}} \cdot (1 + c_{lin} \cdot l_1 \cdot \sin \vartheta_{e-lin}) \qquad (6.2.3.1.4)$$

Si se cumple que $u_{1-lin} > 0$, $|\vartheta_{e-lin}| < 90º$, entonces el cero estará situado en el semieje real negativo.

Si se toma el valor de ϑ_{e-lin}, sin tener en cuenta la dinámica, ante un cambio en los puntos de linealización, el cero tenderá a situarse en:

$$z_1 = -\frac{u_{1-lin}}{l_1} \cdot \sqrt{1 - c_{lin}^2 \cdot l_1^2} \qquad (6.2.3.1.5)$$

6.2.3.2. Estabilidad del sistema

El sistema es inestable en cadena abierta, puesto que tiene dos polos imaginarios puros.

Al ser el sistema inestable, no se puede hablar de valor final ante respuesta escalón.

6.2.3.3. Ganancia en cadena abierta

La ganancia del sistema en cadena abierta viene dada por:

$$K_{ab} = A_1 = u_{1-lin} \cdot \frac{l_1}{L} \cdot \frac{1}{\cos^2\left(\operatorname{atan}\left(-\frac{L}{l_1} \cdot \tan \vartheta_{e-lin}\right)\right)} \cdot \cos \vartheta_{e-lin} \qquad (6.2.3.3.1)$$

Si se elige el valor de ϑ_{e-lin}, sin tener en cuenta la dinámica, la ganancia del sistema en cadena abierta tenderá a ser:

$$K_{ab} = u_{1-lin} \cdot \frac{l_1}{L} \cdot \frac{1}{\cos^2\left(\operatorname{atan}\dfrac{L \cdot c_{lin}}{\sqrt{1 - c_{lin}^2 \cdot l_1^2}}\right)} \cdot \sqrt{1 - c_{lin}^2 \cdot l_1^2} \qquad (6.2.3.3.2)$$

6.2.4. Variación de la función de transferencia continua con los puntos de linealización

Como se ha comentado en el Apartado 6.2.2, se va a elegir una función de transferencia representativa del sistema que varíe con los puntos de linealización escogidos, considerando el caso de que se tome con dinámica o bien sin dinámica (en el caso de ϑ_{e-lin}).

A causa de esto, la posición de los ceros y los polos, así como la ganancia del sistema, variarán con los puntos de linealización.

En este apartado se describirá cómo varia la función de transferencia al variar el punto de linealización.

- Variación de polos y ceros

Por simplicidad, se analizará exclusivamente la variación sin dinámica de los puntos de linealización. Se asume que entre el valor inicial y el valor final habrá una cierta dinámica que por ahora despreciamos.

Como se vio en el apartado 6.2.3.1., la posición final de los polos venia dada por:

$$p_{1,2} = \pm c_{lin} \cdot \frac{u_{1-lin}}{\sqrt{1 - c_{lin}^2 \cdot l_1^2}} \cdot i \qquad (6.2.4.1)$$

Analizando dicha fórmula, se llega a:

- Los polos son imaginarios puros conjugados, es decir, están sobre el eje imaginario.
- Si la velocidad u_{1-lin} es nula, hay un polo doble en cero. Según crece la velocidad, los polos escalan de forma lineal por el eje imaginario, uno hacia valores imaginarios puros negativos y el otro hacia valores positivos. No importa el signo de la velocidad, ya que lo único que ocurre es que se intercambian el polo imaginario positivo con el negativo.
- Si c_{lin} es nula (caso de recta), hay un polo doble en cero. Según crece el valor de c_{lin}, los polos escalan por el eje imaginario. No importa el signo de c_{lin}, ya que lo único que sucede es que se intercambian el polo imaginario positivo con el negativo. Hay un valor máximo de c_{lin} dado por: $|c_{lin}| < \frac{1}{l_1}$, lo cual no es un problema, ya que la hipótesis de trabajo es más restrictiva.

La posición del cero venia dada por:

$$z_1 = -\frac{u_{1-lin}}{l_1} \cdot \sqrt{1 - c_{lin}^2 \cdot l_1^2} \qquad (6.2.4.2)$$

Luego:

- Hay un cero sobre el eje real.
- Si u_{1-lin} es nula, el cero se anula. Si u_{1-lin} crece hacia valores positivos ($u_{1-lin} > 0$), el cero se hará real negativo, pero si u_{1-lin} crece hacia valores negativos ($u_{1-lin} < 0$), el cero será real positivo.
- Si c_{lin} es nula (caso recta), la posición del cero vendrá dada por: $z_1 = -\frac{u_{1-lin}}{l_1}$. Según crezca el valor absoluto de c_{lin}, el cero se irá acercando al eje imaginario. Hay un valor máximo de c_{lin} dado por: $|c_{lin}| < \frac{1}{l_1}$.

Representando gráficamente, se tiene:

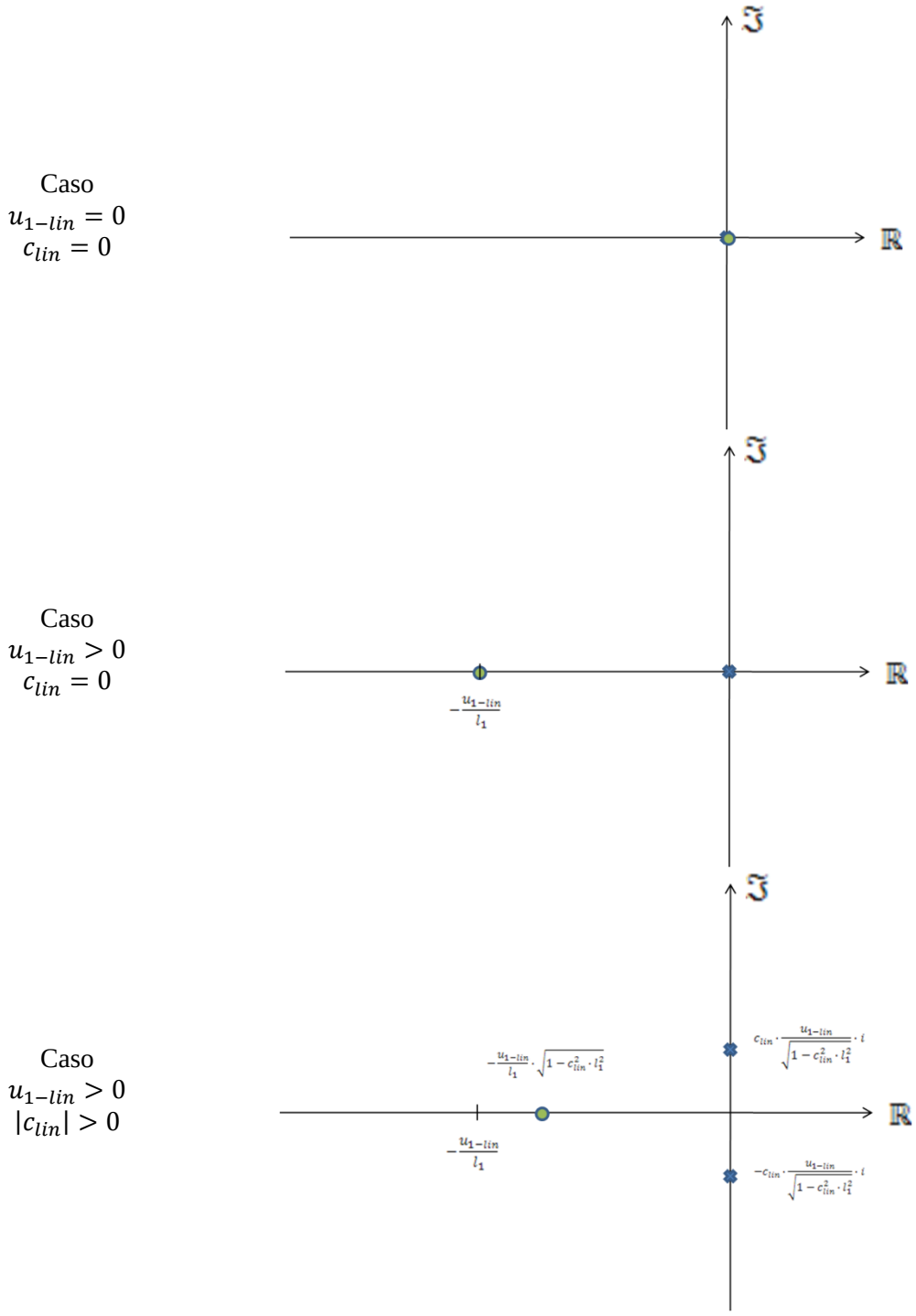


Ilustración 34: Diversas posiciones de polos y ceros según puntos de linealización

- Estabilidad del sistema

Como se ha visto, los polos siempre van a estar en el eje imaginario, luego el sistema siempre va a ser inestable en cadena abierta.

6.2.5. Comprobación de la hipótesis de linealización

En este apartado se va a comprobar la hipótesis de linealización, analizando si el sistema linealizado va a representar adecuadamente el sistema real. Dicho análisis se hará mediante simulaciones cuyo resultado se puede ver en el Anexo V.

Se extraen de todas las simulaciones llevadas a cabo, las siguientes conclusiones:

- El sistema linealizado compuesto por las funciones de transferencia $G(s)$, $P_c(s)$ y $P_{u_1}(s)$, representan adecuadamente al sistema real siempre que se esté en el entorno del punto de $d_{lin} = 0$
- La función de transferencia $P_{u_1}(s)$, aporta error cuando $c_{lin} \neq 0$, y cuando la diferencia $u_1 - u_{1-lin}$, crece.
- La función de transferencia $P_c(s)$, aporta un error más importante, cuanto mayor sea la diferencia, en valor absoluto: $c - c_{lin}$
- La función de transferencia $G(s)$, representa adecuadamente al sistema real, aunque $c \neq c_{lin}$ y $u_1 \neq u_{1-lin}$, siempre que d se esté en el entorno del punto de linealización $d_{lin} = 0$

Luego se puede aceptar la linealización en el caso que nos aplica, que será ($d \approx d_{lin} = 0$), y está será mejor si las diferencias $u_1 - u_{1-lin}$, y $|c - c_{lin}|$ son menores.

6.3.Función de transferencia discreta del modelo linealizado

Como el sistema con el que se va a trabajar, va a funcionar en tiempo discreto, se hace imprescindible modelar la función de transferencia discreta del sistema.

6.3.1. Discretización del sistema continuo

Se parte de la función de transferencia continua G(s) con puntos de linealización variables, tal como se vio en el Apartado 6.2.2. Se discretiza con un bloqueador de orden cero, y un muestreador de periodo T.

Ilustración 35: Discretización de la función de transferencia que representa al UGV

Donde la función de transferencia discreta del sistema $\frac{\Delta d_k}{\Delta \phi_k} = B_0 G(z)$ se obtiene, al muestrear mediante rectángulos con un bloqueador de orden cero, haciendo la siguiente aproximación:

$$s = \frac{z-1}{T} \qquad (6.3.1.1)$$

Y se obtiene:

$$B_0G(z) = \frac{\Delta \mathrm{d_k}}{\Delta \phi_k} = (A_1 \cdot T) \cdot \frac{z + (A_2 \cdot T - 1)}{z^2 - 2 \cdot z + (A_3 \cdot T^2 + 1)} \quad (6.3.1.2)$$

Donde las constantes A_i son función del punto de linealización.

Por sencillez, se define:

$$B_0G(z) = \frac{\Delta \mathrm{d_k}}{\Delta \phi_k} = B_1 \cdot \frac{z + B_2}{z^2 + B_3 \cdot z + B_4} \quad (6.3.1.3)$$

Siendo:

$$B_1 = A_1 \cdot T \quad (6.3.1.4)$$
$$B_2 = A_2 \cdot T - 1 \quad (6.3.1.5)$$
$$B_3 = -2 \quad (6.3.1.6)$$
$$B_4 = A_3 \cdot T^2 + 1 \quad (6.3.1.7)$$

Si se utiliza el método de los residuos exacto, se tiene:

Para el caso $A_3 \neq 0$

$$A_p = A_2 \cdot \cos\left(\sqrt{A_3} \cdot T\right) + \sqrt{A_3} \cdot \sin\left(\sqrt{A_3} \cdot T\right) \quad (6.3.1.8)$$
$$A_{pp} = A_2 + A_p - 2 \cdot A_2 \cdot \cos(\sqrt{A_3} \cdot T) \quad (6.3.1.9)$$
$$B_1 = \frac{A_1}{A_3} \cdot A_{pp} \quad (6.3.1.10)$$
$$B_2 = \frac{A_2 - A_p}{A_{pp}} \quad (6.3.1.11)$$
$$B_3 = -2 \cdot \cos(\sqrt{A_3} \cdot T) \quad (6.3.1.12)$$
$$B_4 = A_3 \cdot T^2 + 1 \quad (6.3.1.13)$$

Para el caso $A_3 = 0$

$$B_1 = \frac{A_1 \cdot T \cdot (2 + T \cdot A_2)}{2} \quad (6.3.1.14)$$
$$B_2 = \frac{A_2 \cdot T - 2}{A_2 \cdot T + 2} \quad (6.3.1.15)$$
$$B_3 = -2 \quad (6.3.1.16)$$
$$B_4 = A_3 \cdot T^2 + 1 \quad (6.3.1.17)$$

6.3.2. Análisis del sistema discreto

Se analizará ahora la función de transferencia del sistema discreto $B_0G(z)$, del mismo modo que se analizó la función de transferencia del sistema continuo G(s).

6.3.2.1. Polos y ceros

La posición de los polos viene dada por:

$$p_{1d,2d} = 1 \pm i \cdot T \cdot \sqrt{A_3} = 1 \pm T \cdot \left| c_{lin} \cdot \frac{u_{1-lin}}{\cos \vartheta_{e-lin}} \right| \cdot i \qquad (6.3.2.1.1)$$

Y tomando exclusivamente el valor final de los puntos de linealización (sin dinámica):

$$p_{1d,2d} = 1 \pm T \cdot c_{lin} \cdot \frac{u_{1-lin}}{\sqrt{1 - c_{lin}^2 \cdot l_1^2}} \cdot i \qquad (6.3.2.1.2)$$

Luego los polos tienen parte real igual a 1, y parte imaginaria similar al caso continuo, pero escalada con T.

La posición del cero viene dada por:

$$z_{1d} = 1 - A_2 \cdot T = 1 - \frac{u_{1-lin}}{l_1 \cdot \cos^2 \vartheta_{e-lin}} \cdot (1 + c_{lin} \cdot l_1 \cdot \sin \vartheta_{e-lin}) \cdot T \qquad (6.3.2.1.3)$$

Y tomando el valor final de los puntos de linealización, despreciando la dinámica:

$$z_{1d} = 1 - T \cdot \frac{u_{1-lin}}{l_1} \cdot \sqrt{1 - c_{lin}^2 \cdot l_1^2} \qquad (6.3.2.1.4)$$

Por lo tanto, el cero se encuentra escalado con T y desplazado una unidad hacia el eje real positivo, respecto al sistema continuo.

6.3.2.2. Estabilidad

Puesto que el sistema tiene sus polos, bien situados sobre la circunferencia unidad, bien fuera de ésta, el sistema es inestable en cadena abierta.

6.3.2.3. Ganancia en cadena abierta

La ganancia del sistema en cadena abierta viene dada por:

$$K_{ab-d} = A_1 \cdot T = u_{1-lin} \cdot \frac{l_1}{L} \cdot \frac{1}{\cos^2 \left(\operatorname{atan} \left(-\frac{L}{l_1} \cdot \tan \vartheta_{e-lin} \right) \right)} \cdot \cos \vartheta_{e-lin} \cdot T \qquad (6.3.2.3.1)$$

Tomando el valor final de los puntos de linealización, despreciando la dinámica, se tiene:

$$K_{ab-d} = u_{1-lin} \cdot \frac{l_1}{L} \cdot \frac{1}{\cos^2\left(\operatorname{atan}\frac{L \cdot c_{lin}}{\sqrt{1 - c_{lin}^2 \cdot l_1^2}}\right)} \cdot T \cdot \sqrt{1 - c_{lin}^2 \cdot l_1^2} \qquad (6.3.2.3.2)$$

7. DISEÑO DE LA ESTRUCTURA DE CONTROL DEL UGV

Una vez modelado y analizado el sistema en cadena abierta, se ha de proceder al diseño de la estructura de control que consiga satisfacer el objetivo del proyecto.

7.1.Estructura de control

La estructura de control más sencilla será:

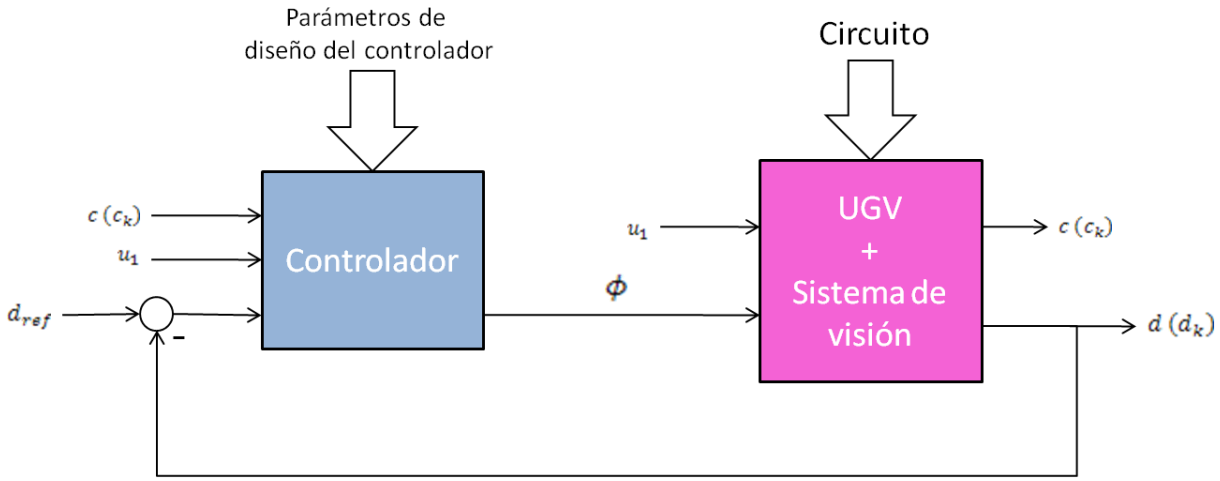


Ilustración 36: Estructura de control general

7.2.Controlador continuo

En primer lugar se diseñará un controlador continuo, como si el UGV y el sistema de visión trabajasen en tiempo continuo, siendo el bucle de control ($d_{ref} = 0$):

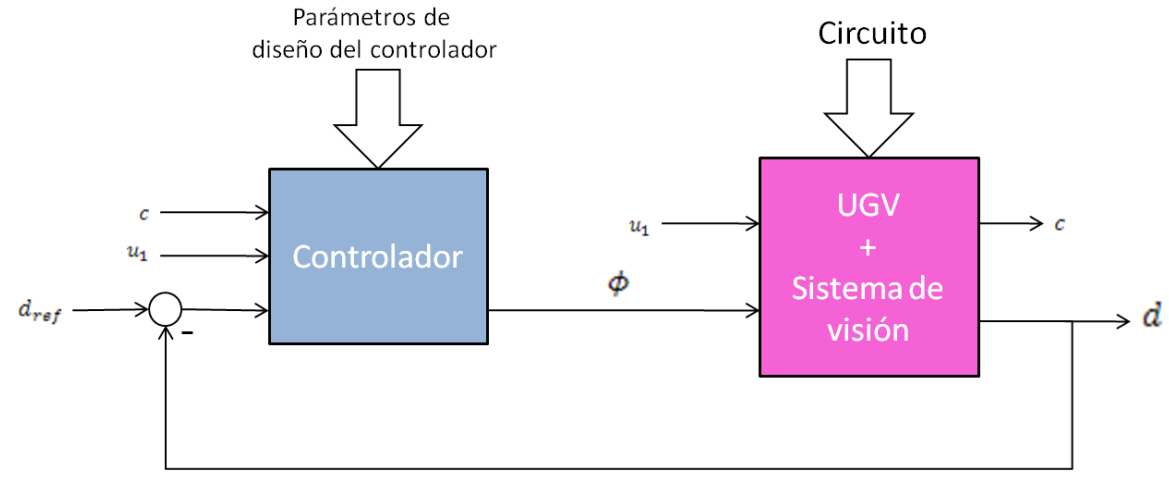


Ilustración 37: Estructura de control continuo

Se diseñará un controlador incremental, que necesitará de unos valores de deslinealización.

$$\phi = \Delta\phi + \phi_{lin} \quad (7.2.1)$$
$$d = \Delta d + d_{lin} \quad (7.2.2)$$

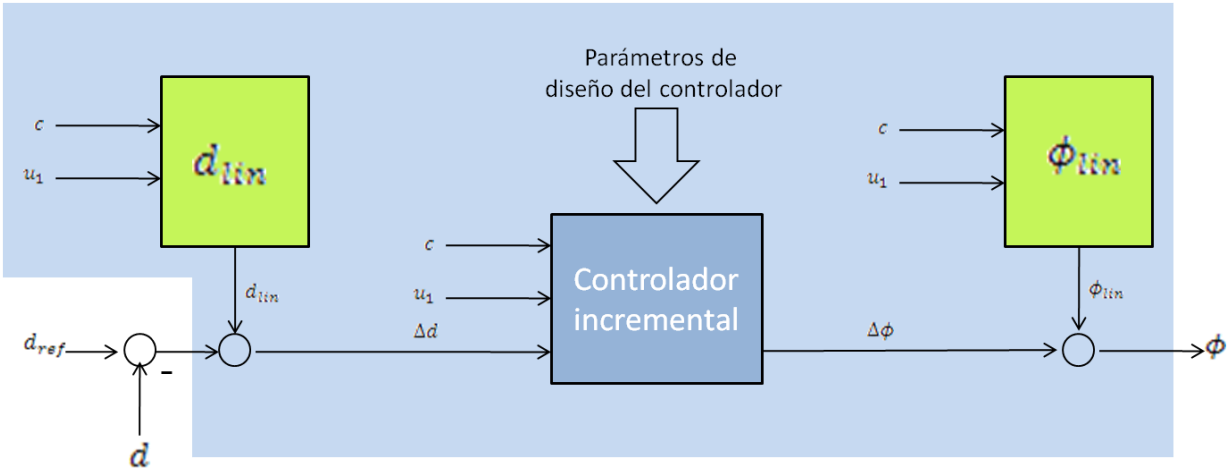


Ilustración 38: Detalle de controlador continuo

7.2.1. Regulador continuo incremental

El regulador incremental R(s) se diseñará, de modo que sea capaz de controlar el sistema representado mediante la función de transferencia:

$$G(s) = \frac{\Delta d}{\Delta \phi} = A_1 \cdot \frac{s + A_2}{s^2 + A_3} \quad (7.2.1.1)$$

Obtenida y estudiada en el Apartado 6.

Se vio en el Apartado 6., que la posición final de los polos y del cero, era:

$$p_{1,2} = \pm c_{lin} \cdot \frac{u_{1-lin}}{\sqrt{1 - c_{lin}^2 \cdot l_1^2}} \cdot i \quad (7.2.1.2)$$

$$z_1 = -\frac{u_{1-lin}}{l_1} \cdot \sqrt{1 - c_{lin}^2 \cdot l_1^2} \quad (7.2.1.3)$$

Y el lugar de las raíces se representa mediante:

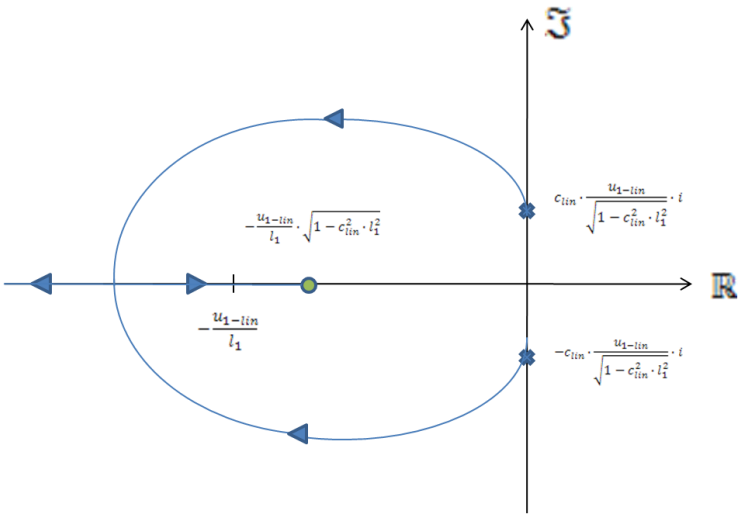


Ilustración 39: Lugar de las raices de la función de transferencia que representa al UGV

7.2.1.1. Selección del tipo de regulador

Se analizará en este apartado los posibles reguladores que se pueden utilizar (P, PI, PID).

i. Acción proporcional

Con una simple acción proporcional (regulador P), se consigue:

- Estabilizar el sistema, al situar los polos en el semiplano real negativo.
- Ajustar cualquier especificación dinámica, situando los polos en la posición deseada de manera que se pueda obtener cualquier valor de sobreoscilación, tiempo de establecimiento o tiempo de pico (sólo uno de ellos simultáneamente).

Se van a estudiar a continuación los errores en régimen permanente y los errores de seguimiento con un regulador P, para saber si se cumplen las condiciones estáticas, cuya función de transferencia sea:

$$R(s) = \frac{\Delta\phi}{\Delta\varepsilon} = K_R \qquad (7.2.1.1.1)$$

En éste caso, el sistema realimentado, definido con funciones de transferencia, se representa mediante:

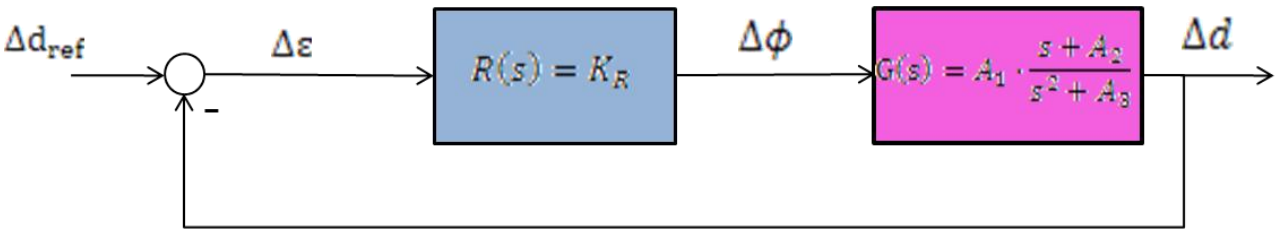


Ilustración 40: Bucle de control con acción proporcional

Y la función de transferencia equivalente al sistema realimentado es:

$$F(s) = \frac{R(s)\cdot G(s)}{1 + R(s)\cdot G(s)} = \frac{K_R \cdot A_1 \cdot \frac{s + A_2}{s^2 + A_3}}{1 + K_R \cdot A_1 \cdot \frac{s + A_2}{s^2 + A_3}} = K_R \cdot A_1 \cdot \frac{s + A_2}{s^2 + K_R \cdot A_1 \cdot s + (K_R \cdot A_1 \cdot A_2 + A_3)} \qquad (7.2.1.1.2)$$

- Error de posición en régimen permanente:

El error de posición se define para sistemas estables como la diferencia entre la salida del sistema (Δd) y la referencia (Δd_{ref}) al excitar el sistema realimentado con una entrada escalón unitario (Δd_{ref} será escalón unitario).

Se tiene entonces:

$$\Delta e_p = \lim_{t\to\infty}(\Delta d_{ref} - \Delta d) = 1 - \lim_{t\to\infty}\left(F(s)\cdot\frac{1}{s}\right) = 1 - \lim_{s\to 0}(s\cdot\left(F(s)\cdot\frac{1}{s}\right)) = 1 - F(0) = 1 - K_R \cdot A_1 \cdot \frac{A_2}{K_R \cdot A_1 \cdot A_2 + A_3} = \frac{A_3}{K_R \cdot A_1 \cdot A_2 + A_3} = \frac{1}{1 + \frac{K_R \cdot A_1 \cdot A_2}{A_3}} = \frac{1}{1 + K_p} \qquad (7.2.1.1.3)$$

Y se tiene un error de posición no nulo que depende de los valores de los coeficientes A_i que a su vez son función de los puntos de linealización del sistema.

- Error de seguimiento ante perturbación Δc

Si se tiene una perturbación Δc, el sistema realimentado se representa mediante:

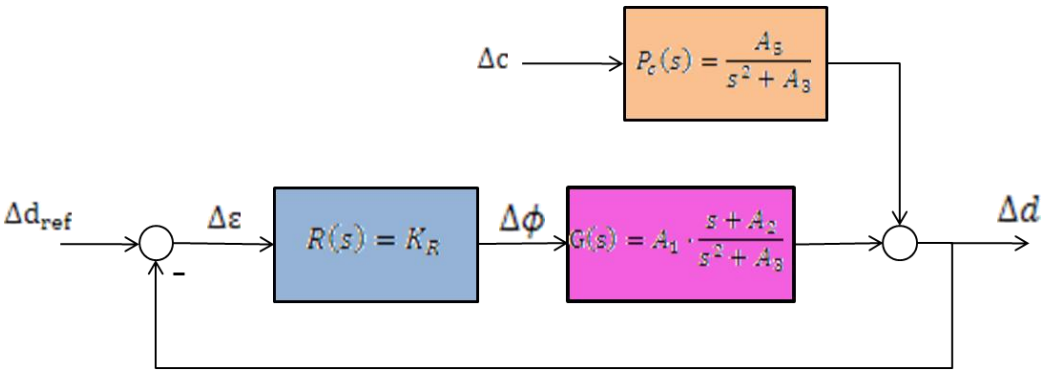


Ilustración 41: Bucle de control con acción proporcional y perturbacion en curvatura

En este caso se tiene:

$$\Delta d = R(s)\cdot G(s)\cdot \Delta\varepsilon + P_c(s)\cdot \Delta c = R(s)\cdot G(s)\cdot(\Delta d_{ref} - \Delta d) + P_c(s)\cdot \Delta c \qquad (7.2.1.1.4)$$

Y reordenando, se tiene:

$$\Delta d = \frac{P_c(s)}{1 + R(s) \cdot G(s)} \Delta c + \frac{R(s) \cdot G(s)}{1 + R(s) \cdot G(s)} \cdot \Delta d_{ref} \qquad (7.2.1.1.5)$$

Y el error de seguimiento ante perturbación Δc, se define para $\Delta d_{ref} = 0$, ante perturbación escalón unitario, como el valor de salida del sistema.

$$\Delta e_{P_c} = \lim_{t \to \infty} (\Delta d) = \lim_{t \to \infty} \left(\frac{P_c(s)}{1 + R(s) \cdot G(s)} \cdot \frac{1}{s} \right) = \lim_{s \to 0} (s \cdot \left(\frac{P_c(s)}{1 + R(s) \cdot G(s)} \cdot \frac{1}{s} \right))$$
$$= \frac{P_c(0)}{1 + R(0) \cdot G(0)} = \frac{\frac{A_5}{A_3}}{1 + K_R \cdot A_1 \cdot \frac{A_2}{A_3}} = \frac{A_5}{A_3 + K_R \cdot A_1 \cdot A_2} \qquad (7.2.1.1.6)$$

Y se tiene un error de seguimiento ante perturbación Δc no nulo que depende de los valores de los coeficientes A_i que a su vez son función de los puntos de linealización del sistema.

- Error de seguimiento ante perturbación Δu_1

Si se tiene una perturbación Δu_1, el sistema realimentado se representa mediante:

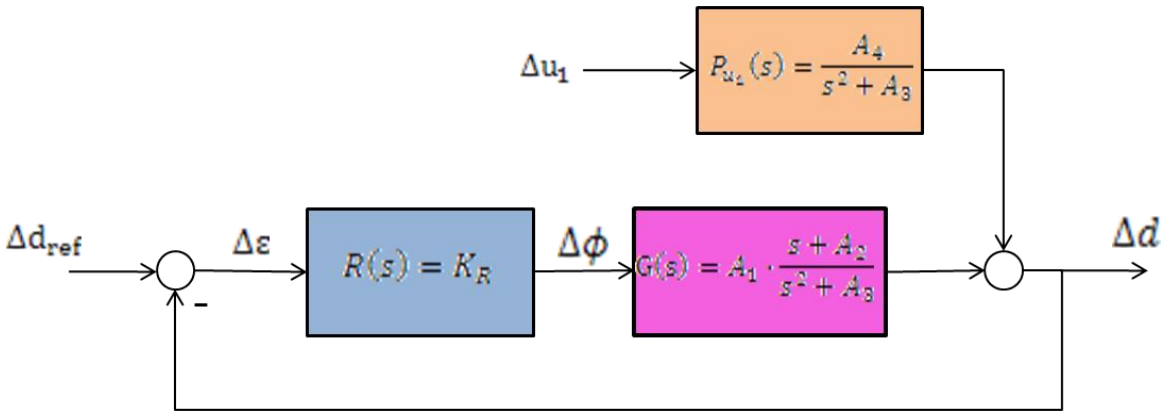


Ilustración 42: Bucle de control con acción proporcional y perturbacion en velocidad

Y se tiene:

$$\Delta d = \frac{P_{u_1}(s)}{1 + R(s) \cdot G(s)} \Delta u_1 + \frac{R(s) \cdot G(s)}{1 + R(s) \cdot G(s)} \cdot \Delta d_{ref} \qquad (7.2.1.1.7)$$

Y el error de seguimiento ante perturbación Δu_1, se define para $\Delta d_{ref} = 0$, ante perturbación escalón unitario, como el valor de salida del sistema.

$$\Delta e_{P_{u_1}} = \lim_{t \to \infty} (\Delta d) = \lim_{t \to \infty} \left(\frac{P_{u_1}(s)}{1 + R(s) \cdot G(s)} \cdot \frac{1}{s} \right) = \lim_{s \to 0} (s \cdot \left(\frac{P_{u_1}(s)}{1 + R(s) \cdot G(s)} \cdot \frac{1}{s} \right))$$
$$= \frac{P_{u_1}(0)}{1 + R(0) \cdot G(0)} = \frac{\frac{A_4}{A_3}}{1 + K_R \cdot A_1 \cdot \frac{A_2}{A_3}} = \frac{A_4}{A_3 + K_R \cdot A_1 \cdot A_2} \qquad (7.2.1.1.8)$$

Y se tiene un error de seguimiento ante perturbación Δu_1 no nulo que depende de los valores de los coeficientes A_i que a su vez son función de los puntos de linealización del sistema.

- Error de seguimiento ante perturbación Δd_{pert}

Si se tiene una perturbación Δd_{pert}, el sistema realimentado se representa mediante:

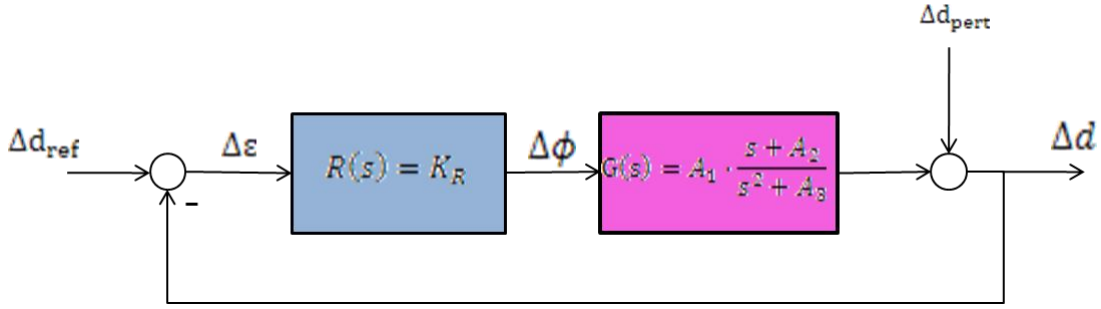


Ilustración 43: Bucle de control con acción proporcional y perturbacion en error

Y se tiene:

$$\Delta d = \frac{1}{1 + R(s) \cdot G(s)} \Delta d_{pert} + \frac{R(s) \cdot G(s)}{1 + R(s) \cdot G(s)} \cdot \Delta d_{ref} \qquad (7.2.1.1.9)$$

Y el error de seguimiento ante perturbación Δd_{pert}, se define para $\Delta d_{ref} = 0$, ante perturbación escalón unitario, como el valor de salida del sistema.

$$\Delta e_{d_{pert}} = \lim_{t \to \infty} (\Delta d) = \lim_{t \to \infty} \left(\frac{1}{1 + R(s) \cdot G(s)} \cdot \frac{1}{s} \right) = \lim_{s \to 0} (s \cdot \left(\frac{1}{1 + R(s) \cdot G(s)} \cdot \frac{1}{s} \right))$$
$$= \frac{1}{1 + R(0) \cdot G(0)} = \frac{1}{1 + K_R \cdot A_1 \cdot \frac{A_2}{A_3}} \qquad (7.2.1.1.10)$$

Y se tiene un error de seguimiento ante perturbación Δd_{pert} no nulo que depende de los valores de los coeficientes A_i que a su vez son función de los puntos de linealización del sistema.

Debido a que se tiene un error de posición, y errores de seguimiento no nulos, se descarta la posibilidad de usar un regulador con acción proporcional exclusivamente.

ii. Acción integral

Si además de la acción proporcional, se incluye una acción integral en el regulador, para intentar eliminar los errores de posición y seguimiento:

$$R(s) = \frac{\Delta \phi}{\Delta \varepsilon} = K_R \cdot \frac{s + a}{s} \qquad (7.2.1.1.11)$$

El sistema realimentado se transforma ahora en:

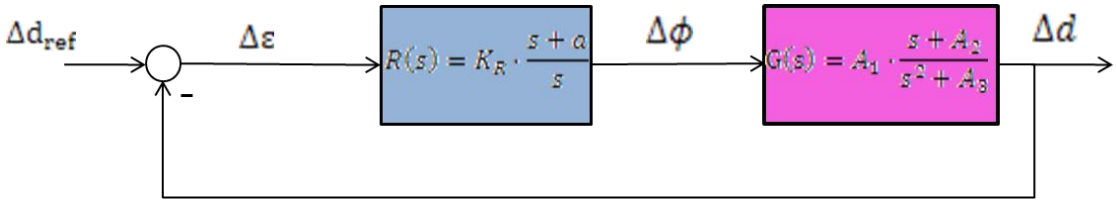


Ilustración 44: Bucle de control con acción integral

Y la función de transferencia del sistema en cadena cerrada se convierte en:

$$F(s) = \frac{R(s)\cdot G(s)}{1 + R(s)\cdot G(s)} = \frac{K_R \cdot \frac{s+a}{s} \cdot A_1 \cdot \frac{s+A_2}{s^2+A_3}}{1 + K_R \cdot \frac{s+a}{s} \cdot A_1 \cdot \frac{s+A_2}{s^2+A_3}}$$
$$= K_R \cdot A_1 \cdot \frac{(s+a)\cdot(s+A_2)}{s\cdot(s^2+A_3) + K_R \cdot A_1 \cdot (s+a)\cdot(s+A_2)} \qquad (7.2.1.1.12)$$
$$= K_R \cdot A_1 \cdot \frac{s^2 + (a+A_2)\cdot s + a\cdot A_2}{s^3 + K_R\cdot A_1 \cdot s^2 + (K_R \cdot A_1 \cdot (a+A_2) + A_3)\cdot s + K_R \cdot A_1 \cdot a \cdot A_2}$$
$$= D_1 \cdot \frac{s^2 + D_2 \cdot s + D_3}{s^3 + D_4 \cdot s^2 + D_5 \cdot s + D_6}$$

Siendo:

$$D_1 = K_R \cdot A_1 \qquad (7.2.1.1.13)$$
$$D_2 = a + A_2 \qquad (7.2.1.1.14)$$
$$D_3 = a \cdot A_2 \qquad (7.2.1.1.15)$$
$$D_4 = K_R \cdot A_1 = D_1 \qquad (7.2.1.1.16)$$
$$D_5 = K_R \cdot A_1 \cdot (a + A_2) + A_3 \qquad (7.2.1.1.17)$$
$$D_6 = K_R \cdot A_1 \cdot a \cdot A_2 \qquad (7.2.1.1.18)$$

El lugar de las raíces se transforma en:

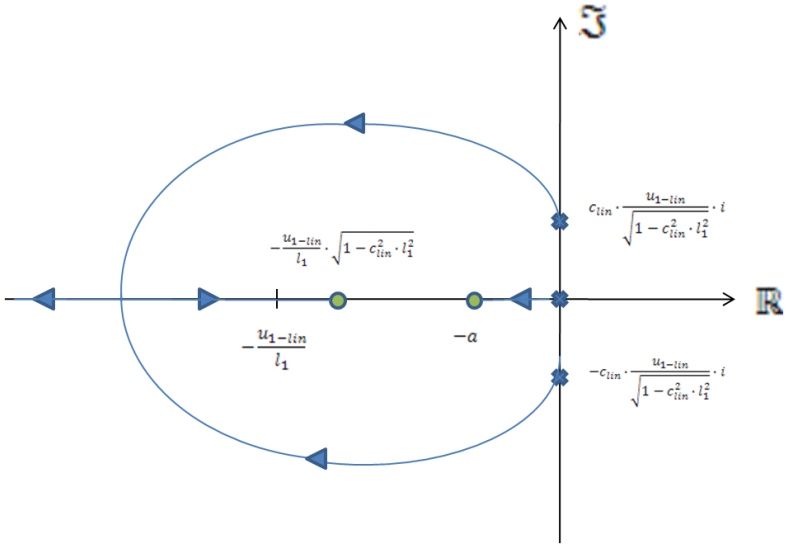


Ilustración 45: Lugar de las raices con acción integral

Se conservan las características que aportaba el regulador P de estabilización del sistema y ajuste de la dinámica del sistema. Ahora los polos dominantes serán aquellos que estén más cerca del eje imaginario.

Con el regulador PI se consiguen anular los errores de posición y de seguimiento, tal y como se demuestra a continuación:

- Error de posición en régimen permanente

Se calculaba como:

$$\begin{aligned}\Delta e_p = \lim_{t\to\infty}(\Delta d_{ref} - \Delta d) = 1 - \lim_{t\to\infty}\left(F(s)\cdot\frac{1}{s}\right) = 1 - \lim_{s\to 0}(s\cdot\left(F(s)\cdot\frac{1}{s}\right)) \\ = 1 - F(0) = 1 - K_R\cdot A_1\frac{a\cdot A_2}{K_R\cdot A_1\cdot a\cdot A_2} = 0\end{aligned} \quad (7.2.1.1.19)$$

Luego no existe error de posición en régimen permanente.

- Error de seguimiento ante perturbación Δc

Se calculaba como:

$$\begin{aligned}\Delta e_{P_c} = \lim_{t\to\infty}(\Delta d) = \lim_{t\to\infty}\left(\frac{P_c(s)}{1+R(s)\cdot G(s)}\cdot\frac{1}{s}\right) = \lim_{s\to 0}(s\cdot\left(\frac{P_c(s)}{1+R(s)\cdot G(s)}\cdot\frac{1}{s}\right)) \\ = \frac{P_c(0)}{1+R(0)\cdot G(0)} = 0\end{aligned} \quad (7.2.1.1.20)$$

No existe error de seguimiento ante perturbación Δc.

- Error de seguimiento ante perturbación Δu_1

Venía dado por:

$$\begin{aligned}\Delta e_{P_{u_1}} = \lim_{t\to\infty}(\Delta d) = \lim_{t\to\infty}\left(\frac{P_{u_1}(s)}{1+R(s)\cdot G(s)}\cdot\frac{1}{s}\right) = \lim_{s\to 0}(s\cdot\left(\frac{P_{u_1}(s)}{1+R(s)\cdot G(s)}\cdot\frac{1}{s}\right)) \\ = \frac{P_{u_1}(0)}{1+R(0)\cdot G(0)} = 0\end{aligned} \quad (7.2.1.1.21)$$

Luego no hay error de seguimiento ante perturbación Δu_1.

- Error de seguimiento ante perturbación Δd_{pert}

Se calculaba como:

$$\begin{aligned}\Delta e_{d_{pert}} = \lim_{t\to\infty}(\Delta d) = \lim_{t\to\infty}\left(\frac{1}{1+R(s)\cdot G(s)}\cdot\frac{1}{s}\right) = \lim_{s\to 0}(s\cdot\left(\frac{1}{1+R(s)\cdot G(s)}\cdot\frac{1}{s}\right)) \\ = \frac{1}{1+R(0)\cdot G(0)} = 0\end{aligned} \quad (7.2.1.1.22)$$

Luego tampoco hay error de seguimiento ante perturbación Δd_{pert}.

Se ha visto que es imprescindible la acción integral, y que con un regulador PI es suficiente para estabilizar el sistema, poner la dinámica que se quiera, y anular los errores de posición y seguimiento.

iii. Acción derivativa

Se analizará la posibilidad de introducir un regulador PID para controlar el sistema:

$$R(s) = \frac{\Delta\phi}{\Delta\varepsilon} = K_R \cdot \frac{(s+a)\cdot(s+b)}{s} \tag{7.2.1.1.23}$$

Con lo que la función de transferencia del sistema en cadena cerrada es:

$$F(s) = \frac{R(s)\cdot G(s)}{1 + R(s)\cdot G(s)} = \frac{K_R \cdot A_1}{1 + K_R \cdot A_1} \cdot \frac{s^3 + (a+b+A_2)\cdot s^2 + (a\cdot b + (a+b)\cdot A_2)\cdot s + a\cdot b\cdot A_2}{s^3 + \frac{K_R\cdot A_1}{1+K_R\cdot A_1}\cdot(a+b+A_2)\cdot s^2 + \frac{K_R\cdot A_1}{1+K_R\cdot A_1}\cdot\left(\frac{A_3}{K_R\cdot A_1} + a\cdot b + (a+b)\cdot A_2\right)\cdot s + \frac{K_R\cdot A_1}{1+K_R\cdot A_1}\cdot a\cdot b\cdot A_2} \tag{7.2.1.1.24}$$

El lugar de las raíces es:

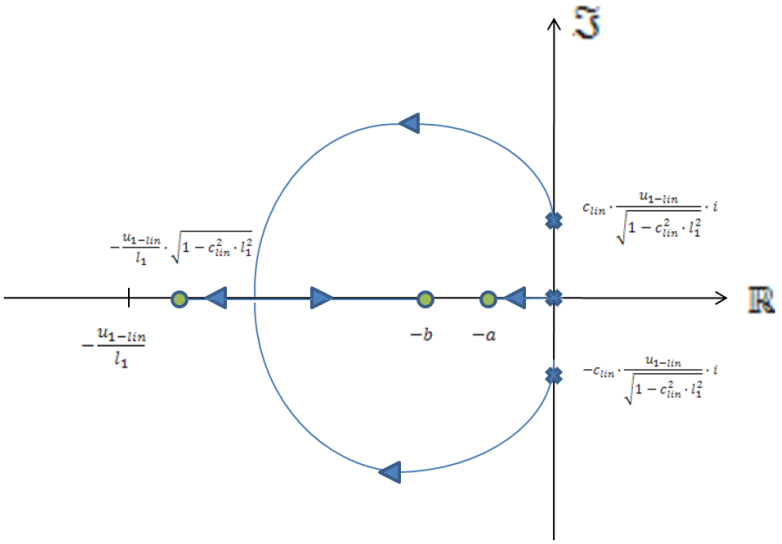


Ilustración 46: Lugar de la raices con acción integral y derivativa

Con lo que el sistema tiene las mismas características respecto de la estabilidad que con un regulador P o PI. La dinámica sin embargo mejora, pudiendo ahora sí, situar los polos del sistema en cualquier punto del plano complejo. Se tienen también las mismas características estáticas que con un regulador PI.

Como se pretende que el sistema sea críticamente amortiguado, se necesitará que todos los polos sean reales, luego el posicionado de éstos en cualquier zona del plano no complejo no es una ventaja. Por otro lado, el polo dominante será el que esté más cerca del eje imaginario, y si se consigue que todos los polos sean reales, éste será el polo introducido con la acción integral, con lo que la ventaja de mover los otros dos polos del sistema gracias a la acción derivativa no tiene especial sentido. Por lo tanto, el uso de un regulador PID no aporta ninguna ventaja frente al uso de un regulador PI. Se utilizará pues un regulador PI.

7.2.1.2. Diseño del regulador PI

Como se ha visto en el apartado anterior, el mejor regulador que se puede elegir es un regulador PI.

Para llevar a cabo el diseño del regulador PI, se pretenden alcanzar los siguientes objetivos:

- Sobreoscilación nula
- Mínimo tiempo de establecimiento sin llegar a saturar la acción de control

Habrá que calcular los parámetros del regulador PI (parámetros a y K_R). Puesto que la posición de los polos y el cero del sistema varían con los puntos de linealización, los parámetros de diseño del regulador también han de variar, luego los parámetros a y K_R, deben ser recalculados en cada instante de tiempo. Se consigue así un control adaptativo.

i. Cálculo de a

Existen varios posibles métodos para calcular la posición del cero del regulador PI:

- Método 1: Partir del regulador P

Se calcula en primer lugar el regulador P que hace que se satisfagan los criterios dinámicos.

Para obtener sobreoscilación nula y mínimo tiempo de establecimiento con un regulador P, el sistema ha de estar críticamente amortiguado, es decir, que sus dos polos sean reales puros e iguales.

La posición de los polos vendrá dada por:

$$s^2 + K_R \cdot A_1 \cdot s + (K_R \cdot A_1 \cdot A_2 + A_3) = 0 \qquad (7.2.1.2.1)$$

$$p_{1,2} = \frac{-K_R \cdot A_1 \pm \sqrt{(K_R \cdot A_1)^2 - 4 \cdot (K_R \cdot A_1 \cdot A_2 + A_3)}}{2} \qquad (7.2.1.2.2)$$

Y los polos serán reales e iguales si:

$$(K_R \cdot A_1)^2 - 4 \cdot (K_R \cdot A_1 \cdot A_2 + A_3) = 0 \qquad (7.2.1.2.3)$$

Obteniéndose:

$$K_R = \frac{2 \cdot (A_2 + \sqrt{A_2^2 + A_3})}{A_1} \quad (7.2.1.2.4)$$

Luego los polos estarán en:

$$p_{1,2} = -\frac{K_R \cdot A_1}{2} = -A_2 - \sqrt{A_2^2 + A_3} \quad (7.2.1.2.5)$$

Una vez calculada la posición de los polos con el regulador P, se comienza la fase de diseño con el regulador PI.

Se sitúa el cero del regulador PI a una distancia de $^1/_3$ de donde estarían los polos dominantes del sistema con el regulador P, tal y como sugiere [14] luego:

$$a = \frac{1}{3} \cdot \left(A_2 + \sqrt{A_2^2 + A_3} \right) \quad (7.2.1.2.6)$$

- Método 2: Diseño experimental

Se calcula experimentalmente el valor de a en función de la velocidad, intentando que el sistema no llegue a saturar.

El método experimental que se va a probar es:

- Velocidad < 1Km/h: a=0.
- 1 Km/h ≤ Velocidad ≤ 5 Km/h: a=0.05.
- 5 Km/h ≤ Velocidad ≤ 40 Km/h: a se calcula en función de la velocidad mediante una recta, cuyo valor mínimo será $vel_{min} = 5\text{Km/h}$ para $a_{min} = 0.2$; y cuyo valor máximo es $vel_{max} = 40\text{Km/h}$, y $a_{max} = 1.5$. Los parámetros de la recta serán: $m = \frac{a_{max} - a_{min}}{vel_{max} - vel_{min}}$, y $b = a_{max} - m \cdot vel_{max}$. La recta será: $a = m \cdot velocidad + b$.
- Velocidad > 40 Km/h: a=1.5.

ii. Cálculo de K_R

Una vez obtenido el valor de a, es necesario calcular la K_R del regulador PI. Para su cálculo se impone la condición de que el sistema no tenga sobreoscilación; es decir, que tenga todos los polos reales. En particular, y para minimizar la acción de control, se obliga a que el sistema sea críticamente amortiguado.

El cálculo teórico que habría que hacer es:

En primer lugar se obtienen los polos del sistema en función de K_R, igualando a cero el polinomio característico de la función de transferencia del sistema realimentado con un regulador PI:

$$s^3 + K_R \cdot A_1 \cdot s^2 + (K_R \cdot A_1 \cdot (a + A_2) + A_3) \cdot s + K_R \cdot A_1 \cdot a \cdot A_2 = 0 \quad (7.2.1.2.7)$$

Se obtendrían entonces tres soluciones (en función de K_R), una real y dos complejas con parte imaginaria no nula para algunos valores de K_R. La ecuación para calcular K_R, se obtendría de igualar la parte imaginaria de dichos polos a cero.

Esto no se puede resolver analíticamente, y su resolución computacional es muy costosa como para hacerlo en tiempo real a la frecuencia del sistema de visión.

Utilizando ciertas propiedades del lugar de las raíces, y el criterio del módulo, el cálculo de K_R se simplifica notablemente.

La función de transferencia del sistema realimentado era:

$$F(s) = \frac{R(s) \cdot G(s)}{1 + R(s) \cdot G(s)} = K_R \cdot A_1 \cdot \frac{(s + a) \cdot (s + A_2)}{s \cdot (s^2 + A_3) + K_R \cdot A_1 \cdot (s + a) \cdot (s + A_2)} \qquad (7.2.1.2.8)$$

Siendo el polinomio característico:

$$p(s) = s \cdot (s^2 + A_3) + K_R \cdot A_1 \cdot (s + a) \cdot (s + A_2) \qquad (7.2.1.2.9)$$

Los puntos del plano complejo que pertenecen al lugar de las raíces cumplen $p(s) = 0$, con lo que se tiene:

$$-K = -K_R \cdot A_1 = \frac{s \cdot (s^2 + A_3)}{(s + a) \cdot (s + A_2)} \qquad (7.2.1.2.10)$$

De [14], se extrae que los puntos en los que varias ramas confluyen en el lugar de las raíces, vienen dados por: $\frac{dK}{ds} = 0$.

Derivando se tiene:

$$\frac{dK}{ds} = -\frac{(3 \cdot s^2 + A_3) \cdot (s^2 + (a + A_2) \cdot s + a \cdot A_2) - (s^3 + A_3 \cdot s) \cdot (2 \cdot s + (a + A_2))}{\left((s + a) \cdot (s + A_2)\right)^2} \qquad (7.2.1.2.11)$$

Igualando a cero y agrupando, se tiene:

$$s^4 + (2 \cdot a + 2 \cdot A_2) \cdot s^3 + (-A_3 + 3 \cdot a \cdot A_2) \cdot s^2 + a \cdot A_2 \cdot A_3 = 0 \qquad (7.2.1.2.12)$$

Ecuación que da la posición en el plano complejo de los puntos de confluencia o divergencia de ramas en el lugar de las raíces.

El lugar de las raíces directo ($K_R > 0$), tendrá un punto de confluencia a la izquierda del cero que esté más alejado del eje imaginario. El lugar de las raíces inverso ($K_R < 0$), tendrá un punto de dispersión entre los dos ceros, y dos puntos de confluencia.

Como el criterio de diseño que se desea seguir es que el sistema sea críticamente amortiguado, es decir, que todos sus polos sean reales, y con la menor K_R, dos polos del sistema estarán

exactamente en el punto de confluencia del lugar de las raíces directo, luego el cálculo de dicho punto utilizando la ecuación anterior, permite obtener la posición de dichos polos.

El cálculo de las raíces de dicha ecuación es sencillo, ya que todos los coeficientes son conocidos en cada momento. De hecho, no es necesario el cálculo de todas las soluciones, sino que basta con obtener la que está a la izquierda del último cero, como se acaba de comentar.

Por otro lado, el criterio del módulo decía:

$$|K| = |K_R| \cdot |A_1| = \frac{d_{P_1} \cdot d_{P_2} \cdot d_{P_3}}{d_{C_1} \cdot d_{C_2}} \qquad (7.2.1.2.13)$$

Y si lo aplicamos sobre el punto de confluencia calculado (cuya distancia al eje imaginario será d_1), sabiendo que:

- Distancia al cero que mete el regulador (a): $d_{C_1} = (d_1 - a)$
- Distancia al cero del sistema: $d_{C_2} = (d_1 - A_2)$
- Distancia al polo que mete el regulador: $d_{P_1} = d_1$
- Distancia a los polos del sistema: $d_{P_1} = d_{P_2} = \sqrt{d_1^2 + A_3}$

Se tiene:

$$|K_R| = \frac{1}{|A_1|} \cdot \frac{d_{P_1} \cdot d_{P_2} \cdot d_{P_3}}{d_{C_1} \cdot d_{C_2}} = \frac{1}{|A_1|} \cdot \frac{d_1 \cdot (d_1^2 + A_3)}{(d_1 - a) \cdot (d_1 - A_2)} \qquad (7.2.1.2.14)$$

Y como se ha calculado un punto del lugar de las raíces directo, se tendrá:

$$K_R = |K_R| \qquad (7.2.1.2.15)$$

De éste modo se ha calculado de forma sencilla el valor de K_R.

iii. Estabilidad

El sistema será siempre estable en las condiciones de funcionamiento (siempre que se esté en el entorno del punto de linealización), ya que por diseño se ha hecho así. Se ha obligado a que todos los polos estén siempre en el semiplano negativo, en concreto en el semieje real negativo. Debido a esto, no será necesaria la comprobación de la estabilidad del sistema.

iv. Características dinámicas

El tiempo de establecimiento t_s se define como:

$$t_s \cong \frac{\pi}{\sigma} \qquad (7.2.1.2.16)$$

Siendo σ el valor absoluto de la parte real del polo dominante. Será mejor aproximación cuanto más dominante sea el polo dominante.

El cálculo de σ es trivial una vez se conoce la K_R y d_1, dividiendo la ecuación característica del sistema realimentado entre $(s - d_1)^2$.

NOTA: Es imprescindible desactivar la acción integral (a=0) cuando la velocidad sea nula puesto que sino saturaría la acción de control. Tanto el método 1, como el método 2 de cálculo de parámetros del regulador lo cumplen.

7.2.2. Puntos de linealización del regulador incremental

En el Apartado 7.2.1., se ha diseñado un regulador PI adaptativo incremental. Ahora es necesario calcular los puntos de linealización entre los que va a funcionar el controlador.

$$\Delta d = d - d_{lin} \quad (7.2.2.1)$$
$$\phi = \Delta\phi + \phi_{lin} \quad (7.2.2.2)$$

El punto d_{lin}, como se vio en el apartado 6, se definía como:

$$d_{lin} = 0 \quad (7.2.2.3)$$

El punto ϕ_{lin}, tal como se vio en el apartado 6, se calculaba a partir de ϑ_{e-lin}, y éste a partir de u_{1-lin} y c_{lin}, y de una hipótesis de cálculo que podía tener en cuenta la dinámica o no, y en caso de tenerla en cuenta, se podía calcular de forma aproximada o no.

Si se parte del conocimiento de c_{lin}, y se calcula ϕ_{lin} de la forma más exacta posible, el regulador incremental exclusivamente tendrá que ejercer acción de control para compensar las perturbaciones de medida d_{pert} y los errores de modelado.

Si se desconoce c_{lin}, entonces no se obtendrá la ϕ_{lin} exacta del sistema, luego el regulador incremental deberá ejercer acción de control, no sólo para compensar las perturbaciones y errores de modelado, sino para compensar el error debido a trabajar lejos del punto de linealización.

7.2.3. Pruebas del controlador en el simulador

Para comprobar que el diseño del regulador es el adecuado se han llevado a cabo varias simulaciones sobre el circuito del INSIA (ver Anexo II). Los bloques que representan al simulador del controlador se pueden ver en el Anexo VI. Los resultados de las simulaciones se recogen en el Anexo VII.

De todas las simulaciones se extrae que:

- Ambos métodos utilizados para el cálculo de a son bueno, pero es mejor el método que utiliza el regulador P.

- El conocimiento de los códigos aporta importantes ventajas a la hora de minimizar el error.
- Aunque se desconozcan los códigos y exista un error en d no modelado, el controlador continuo diseñado es adecuado.

7.3.Controlador discreto

En el apartado 7.2., se ha diseñado el controlador continuo que haría funcionar al sistema adecuadamente; sin embargo, el sistema real objeto del proyecto, trabaja en tiempo discreto a la frecuencia del sistema de visión (en condiciones normales $f = 29Hz$).

El computador de control trabajará a una frecuencia mayor que se puede aproximar por $f_2 = 500Hz$.

Se utilizará el mismo método que en el Apartado 7.2., diseñando un controlador formado por un punto de linealización y un regulador incremental.

La fórmula del regulador discreto será:

$$\phi_{k_2} = \Delta\phi_k + \phi_{lin-k_2} \tag{7.3.1}$$

7.3.1. Regulador incremental

Para diseñar el regulador incremental se plantean dos posibilidades:

- Diseño del regulador discreto a partir del modelo del sistema discreto
- Discretización del regulador continuo

Pese a cometerse un cierto error, se elegirá la segunda opción, puesto que es más sencillo y se parte de un trabajo ya desarrollado en el regulador continuo.

i. Discretización del regulador continuo

El regulador incremental PI se discretiza utilizando una aproximación por rectángulos, con lo que:

$$R(z) = \frac{\Delta\phi_k}{\Delta\varepsilon_k} = R\left(s = \frac{z-1}{T\cdot z}\right) = K_R \cdot \frac{\frac{z-1}{T\cdot z} - a}{\frac{z-1}{T\cdot z}} = K_{Rd}\cdot\frac{z-a_d}{z-1} \tag{7.3.1.1}$$

Donde $\Delta\varepsilon_k = \Delta \mathrm{d}_{\mathrm{ref-k}} - \Delta \mathrm{d}_{\mathrm{k}}$

Siendo:

$$K_{Rd} = K_R \cdot (a\cdot T + 1) \tag{7.3.1.2}$$

$$a_d = \frac{1}{a\cdot T + 1} \tag{7.3.1.3}$$

$$T = \frac{1}{f} \tag{7.3.1.4}$$

Llevar a cabo esta discretización significa que se van a calcular los parámetros del regulador incremental continuo como si fuese éste el que trabaja, y posteriormente se va a llevar a cabo una discretización de dicho regulador continuo.

La fórmula desarrollada del regulador incremental discreto será:

$$\Delta\phi_k = \Delta\phi_{k-1} + K_{rd} \cdot (\Delta\varepsilon_k - a_d \cdot \Delta\varepsilon_{k-1}) \quad (7.3.1.5)$$

ii. Estabilidad del controlador discreto

Puesto que se trabaja en el campo discreto, habrá que analizar los posibles problemas de estabilidad.

La función de transferencia discreta del sistema a controlar era (como se vio en el Apartado 6.3.):

$$B_0G(z) = \frac{\Delta \mathrm{d_k}}{\Delta\phi_k} = B_1 \cdot \frac{z + B_2}{z^2 + B_3 \cdot z + B_4} \quad (7.3.1.6)$$

Y la del regulador PI discretizado:

$$R(z) = \frac{\Delta\phi_k}{\Delta\varepsilon_k} = K_{Rd} \cdot \frac{z - a_d}{z - 1} \quad (7.3.1.7)$$

Luego la función de transferencia discreta del sistema realimentado es:

$$F = \frac{R(z) \cdot B_0G(z)}{1 + R(z) \cdot B_0G(z)} = B_1 \cdot K_{rd} \cdot \frac{(z + B_2) \cdot (z - a_d)}{z^3 + C_1 \cdot z^2 + C_2 \cdot z + C_3} \quad (7.3.1.8)$$

Siendo:

$$C_1 = B_3 - 1 + K_{rd} \cdot B_1 \quad (7.3.1.9)$$
$$C_2 = B_4 - B_3 + K_{rd} \cdot B_1 \cdot (B_2 - a_d) \quad (7.3.1.10)$$
$$C_3 = -B_4 - K_{rd} \cdot B_1 \cdot B_2 \cdot a_d \quad (7.3.1.11)$$

Y según el criterio de Jury para la estabilidad [9], el sistema será estable si se cumplen las siguientes condiciones:

- Condición 1: $p(1) = 1 + C_1 + C_2 + C_3 - 3 > 0$
- Condición 2: $\mathrm{p}(-1) = -1 + \mathrm{C}_1 - \mathrm{C}_2 + \mathrm{C}_3 < 0$
- Condición 3: $|C_3| < 1$ y $\left|C_3^2 - 1\right| > |C_1 \cdot C_3 - C_2|$

Habrá que comprobar la estabilidad en cada instante de tiempo puesto que los coeficientes C_i varían constantemente.

En caso de que el sistema se inestabilizase, habría que proceder a su detención inmediata en condiciones de emergencia.

7.3.2. Puntos de linealización del controlador discreto

Respecto a los puntos de linealización, se obtendrá su evolución temporal, pero muestreada a la frecuencia del computador f_2.

El trabajar a la frecuencia del computador permite obtener la evolución de ϕ_{lin} de una forma más exacta que si se calculase a la frecuencia de los datos de visión.

7.3.3. Pruebas del controlador en el simulador

Se han llevado a cabo varias pruebas de simulación para comprobar la bondad del controlador diseñado sobre el circuito del INSIA (ver Anexo II). Los bloques que representan al simulador del controlador se pueden ver en el Anexo VI.

De las simulaciones del controlador discreto se concluye que:

- El controlador no inestabiliza al sistema.
- El uso de una frecuencia de trabajo mayor para el punto de linealización que la del regulador incremental apenas aporta ventajas, con lo que se puede hacer trabajar a la misma frecuencia.
- Aunque la frecuencia de trabajo del sistema de visión se reduzca, el controlador sigue funcionando adecuadamente.

8. IMPLEMENTACIÓN SOBRE EL UGV REAL

8.1.Migración del controlador

Como se ha visto en el Apartado 7., la fórmula del controlador discreto diseñado será:

$$\phi_{k_2} = \Delta\phi_k + \phi_{lin-k_2} \tag{8.1.1}$$

Donde ϕ_{lin-k_2} será el punto de linealización calculado con dinámica (bien de la forma exacta, bien aproximada) podrá ir a la frecuencia del computador, o a la del sistema de visión, como se ha visto en el Apartado 7.3.3.

Siendo el regulador incremental:

$$\Delta\phi_k = \Delta\phi_{k-1} + K_{rd} \cdot (\Delta\varepsilon_k - a_d \cdot \Delta\varepsilon_{k-1}) \tag{8.1.2}$$

El primer término de la derecha de la ecuación del controlador discreto ($\Delta\phi_k$), deberá ir a la frecuencia de muestreo del sistema de visión; sin embargo, el punto de linealización deberá sacar valores lo más rápidamente posible.

Como se ha detallado en el apartado 3., la acción de control se mide en grados de volante y no en radianes de rueda, y además se miden en sentidos contrarios. El error o distancia a la curva, en el UGV real se mide en pixeles, mientras que en el controlador diseñado se mide en metros, y de signo contrario.

Se hace la siguiente transformación del controlador incremental:

$$\Delta d(m) = -\Delta x(pix) \cdot 1.32 \cdot 10^{-3}\ \frac{m}{pix} \tag{8.1.3}$$

$$\Delta\alpha(º\ volante) = -\Delta\phi(rad\ rueda) \cdot \frac{360\ º\ rueda}{2 \cdot \pi\ rad\ rueda} \cdot 18\ \frac{º\ volante}{º\ rueda} \tag{8.1.4}$$

Luego sustituyendo en la fórmula del controlador discreto diseñado, se obtiene:

$$-\frac{2\pi}{360 \cdot 18} \cdot \Delta\alpha_k = -\frac{2\pi}{360 \cdot 18} \cdot \Delta\alpha_{k-1} + K_{rd} \cdot \left(-1.32 \cdot 10^{-3} \cdot \Delta x_k - a_d \cdot (-1.32 \cdot 10^{-3} \cdot \Delta x_{k-1})\right) \tag{8.1.5}$$

Y reordenando, se tiene:

$$\Delta\alpha_k = \Delta\alpha_{k-1} + K_{rd} \cdot \frac{360}{2 \cdot \pi} \cdot \frac{18}{1} \cdot \frac{0.00132}{1} \cdot (\Delta error_k - a_d \cdot \Delta error_{k-1}) \tag{8.1.6}$$

Se lleva a cabo la siguiente transformación del punto de linealización:

$$\alpha_{lin}(º\ volante) = -\Delta\phi_{lin}(rad\ rueda) \cdot \frac{360\ º\ rueda}{2 \cdot \pi\ rad\ rueda} \cdot 18\ \frac{º\ volante}{º\ rueda} \tag{8.1.7}$$

Con lo que se obtiene:

$$\alpha_{lin-k_2} = -\frac{360}{2 \cdot \pi} \cdot \frac{18}{1} \cdot \phi_{lin-k_2} \tag{8.1.8}$$

Tras la transformación, resulta, en las variables con las que se controla el UGV real:

$$\alpha_{k_2} = \Delta\alpha_k + \alpha_{lin-k_2} \tag{8.1.9}$$

El punto incremental $\Delta\alpha_k$ se calculará a la frecuencia del sistema de visión (en condiciones normales $f = 39\ Hz$), y su valor se bloqueará mediante un bloqueador de orden cero.

8.2.Programación

Para la programación del controlador se utilizan unos fragmentos de código en lenguaje C++, que se incluirán adecuadamente dentro del programa de control del UGV que está corriendo en el lado del servidor del computador de alto nivel, como se vio en el Apartado 3.

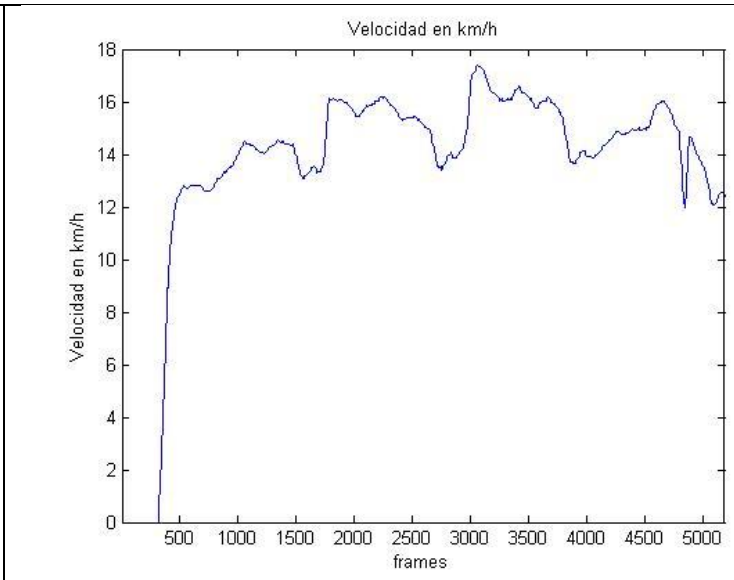
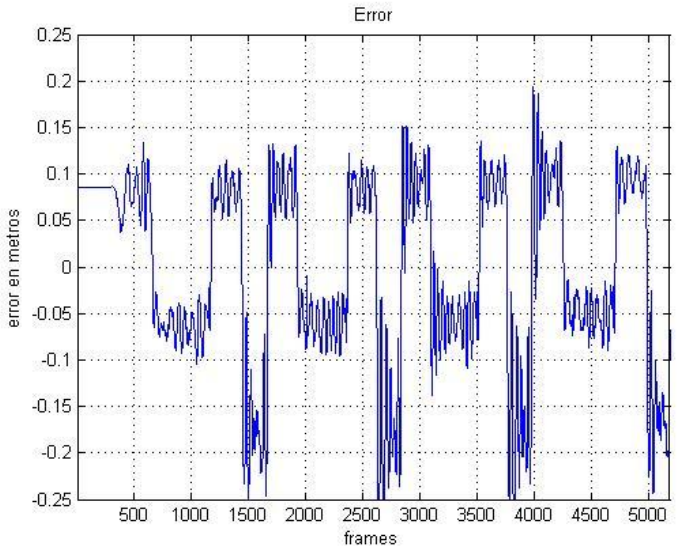
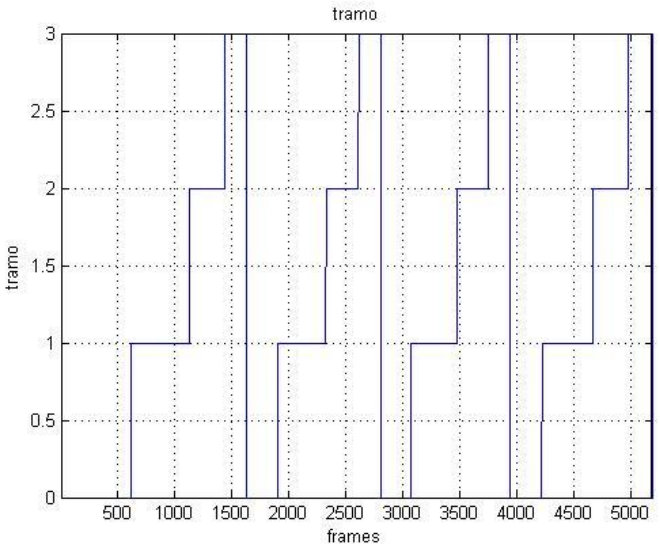
En el Anexo VIII se pueden ver dichos fragmentos de código.

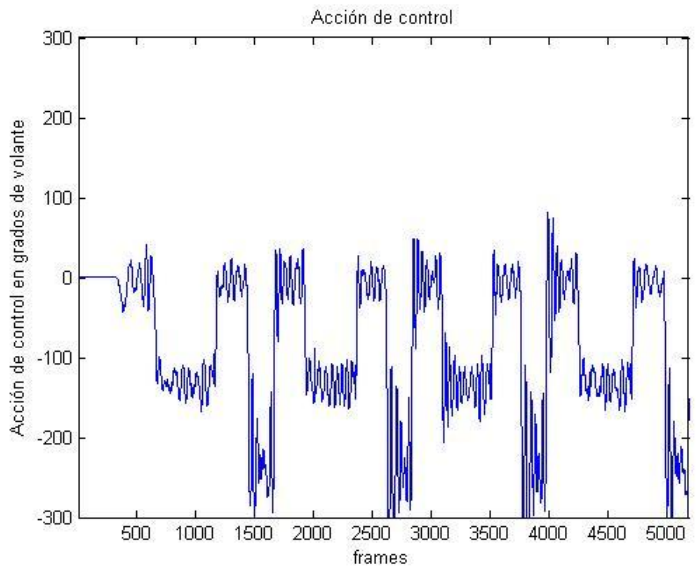
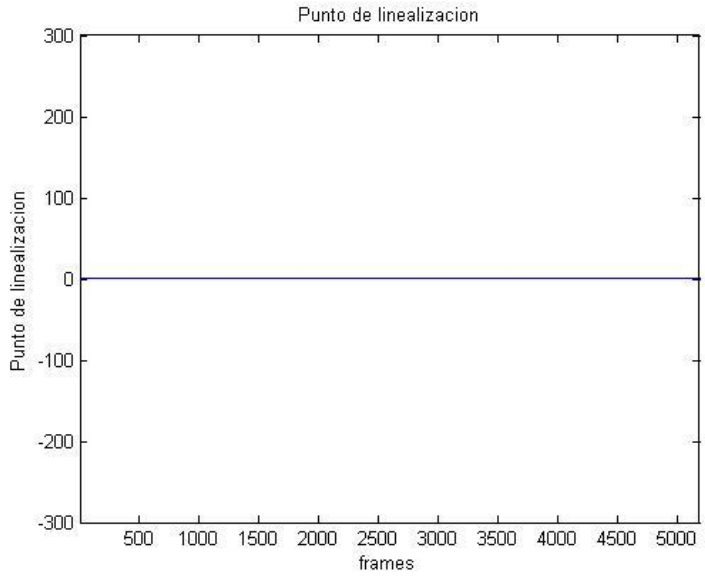
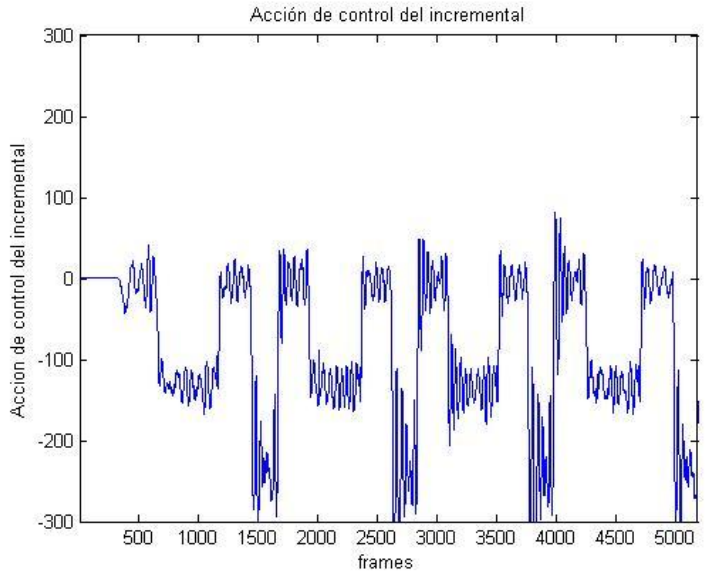
8.3.Pruebas

Se han llevado a cabo dos tipos de pruebas. Por un lado aquellas con plena información del circuito, y por otro, aquellas sin más información del circuito que la distancia a la curva (se desconoce el tramo).

8.3.1. Pruebas sin lectura de información del circuito

Las gráficas relativas a las pruebas sin lectura de información del circuito son:

Velocidad	
Error (en metros) - Error medio cuadrático: 0.1023 - Media del error: $8.0706 \cdot 10^{-4}$	
Tramo	

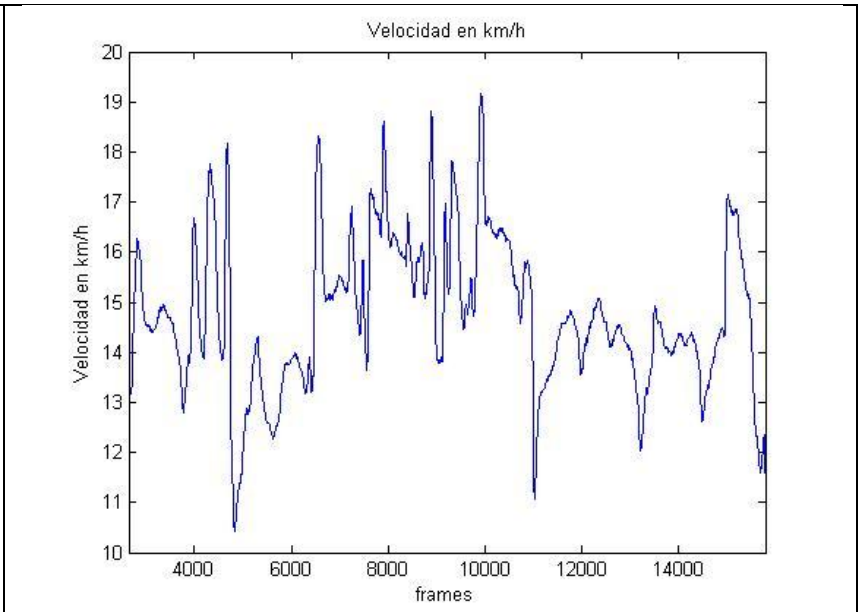
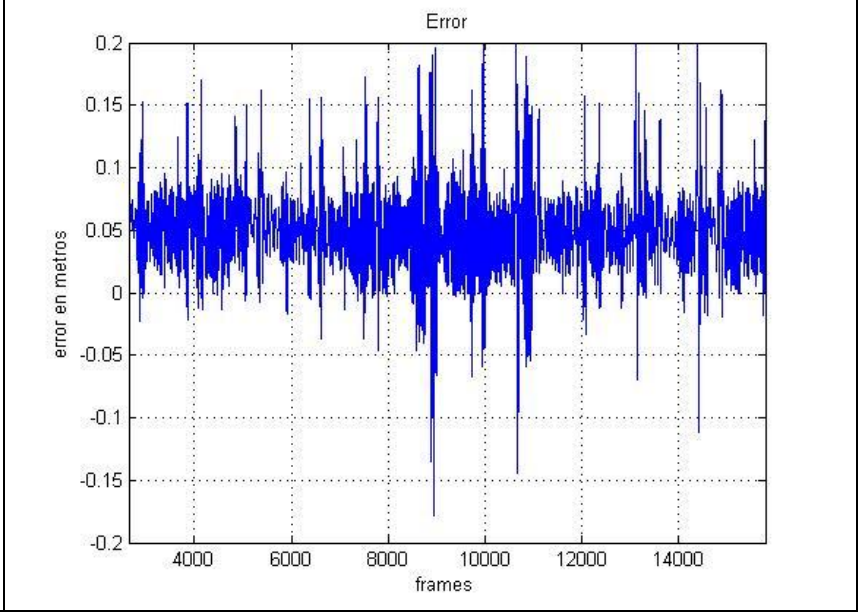
Alfa (acción de control total en grados de volante)	
Alfa de linealización	
Incremento de alfa (acción de control del regulador incremental)	

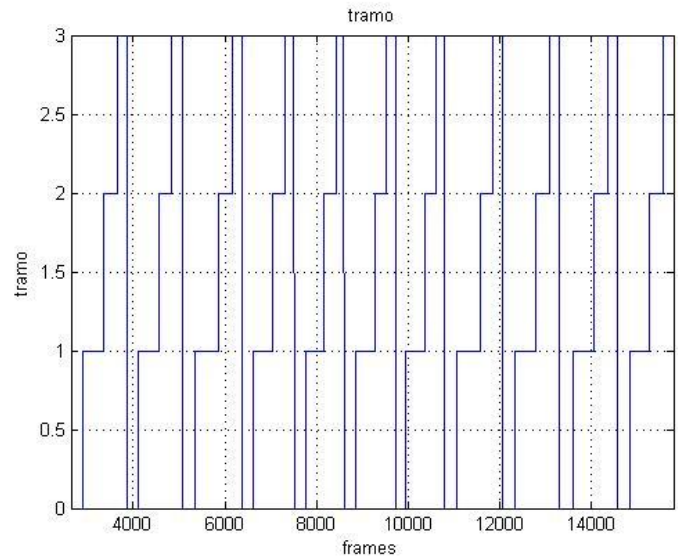
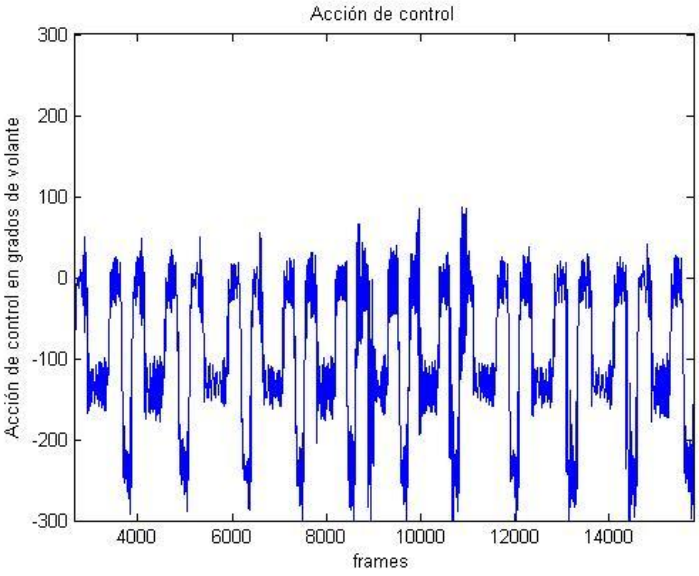
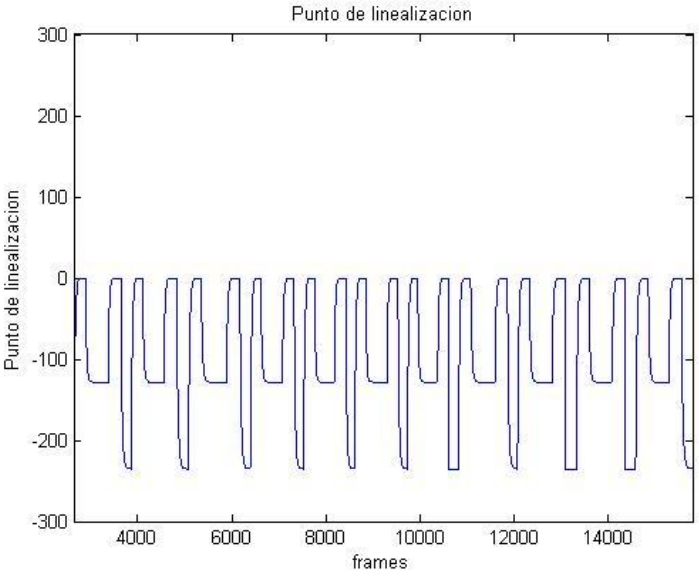
Se comprueba que el controlador es capaz de seguir el circuito aún sin poseer información del tipo de tramo en que se encuentra, a distintas velocidades.

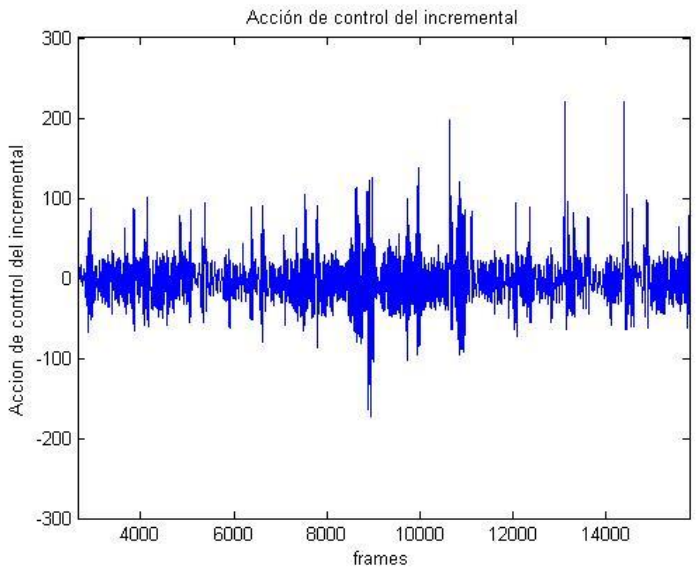
Se han pasado correctamente las pruebas de verificación C2 desarrolladas en el Anexo I.

8.3.2. Pruebas con lectura de información del circuito

Las gráficas de dichas pruebas son:

Velocidad	
Error (en metros) - Error medio cuadrático: 0.0629 - Media del error: 0.0504	

Tramo	
Alfa (acción de control total en grados de volante)	
Alfa de linealización	

Incremento de alfa (acción de control del regulador incremental)	

Se ha comprobado que el controlador es capaz de seguir el circuito de una manera más suave con conocimiento del tipo de tramo a diferentes velocidades.

Se han pasado correctamente las pruebas de verificación C1 desarrolladas en el Anexo I.

9. PLANIFICACIÓN DEL PROYECTO

9.1.Estructura de descomposición del proyecto (EDP)

El proyecto general se descompone en los paquetes de trabajo que se muestran en la estructura de descomposición del proyecto (EDP) en el Anexo IX.

Si se analizan los paquetes de trabajo (WP) en los que se compone directamente el proyecto

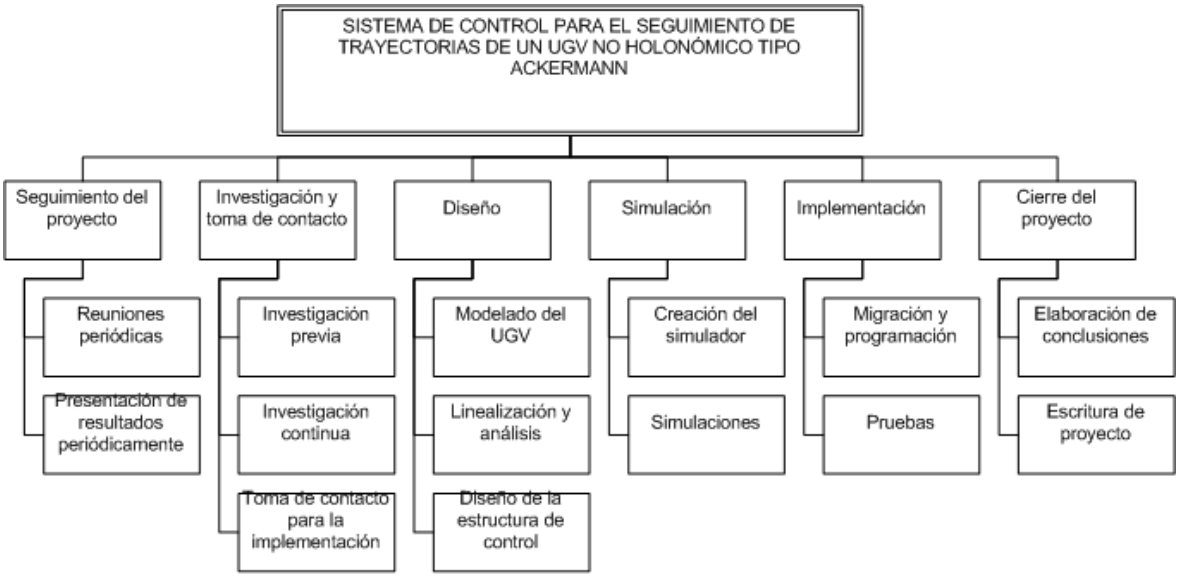


Ilustración 47: EDP con primeras WP

Se tiene:

- Seguimiento del proyecto: Es una actividad que se ha desarrollado a lo largo de todo el proyecto, que constaba de reuniones periódicas con el tutor y con el equipo de trabajo, en las que se presentaba el avance, se hacía una puesta en común, compartiendo información, y se ofrecían posibles caminos y soluciones a los problemas no resueltos.

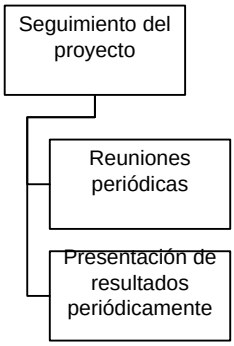


Ilustración 48: WP1

- Investigación y toma de contacto: Es una actividad de aprendizaje que ha constado de una etapa de investigación inicial, en la que se propició la toma de contacto con el proyecto,

una etapa de investigación continua a lo largo de todo el proyecto, y una toma de contacto con el UGV real, previa a la fase de implementación física.

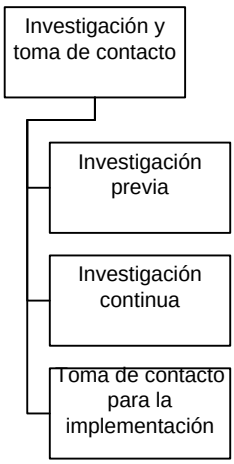


Ilustración 49: WP2

- Diseño: Es una actividad desarrollada tras la etapa de investigación inicial que es la primera de las partes centrales del proyecto. Representa la parte teórica del mismo, en la que hay un modelado del sistema, un análisis del sistema y un diseño del controlador. Cada sub-actividad engloba otras sub-actividades tal como se puede observar en la EDP del Anexo X.

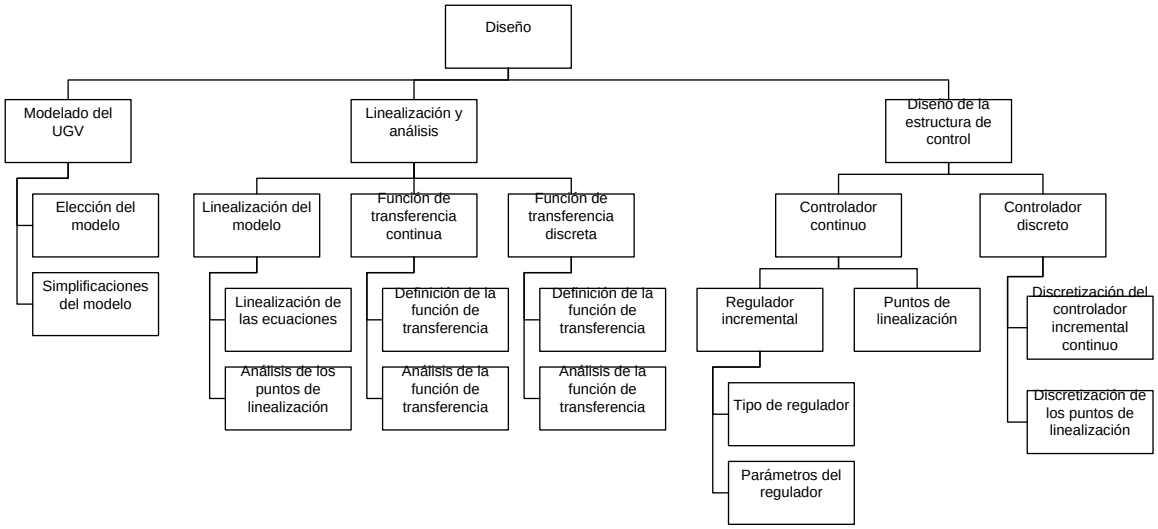


Ilustración 50: WP3

- Simulación: La actividad de simulación comienza justo cuando se obtienen los primeros modelos teóricos del sistema y se han de comprobar los resultados. Consta de dos actividades fundamentales que son la creación del simulador, y la etapa continuada en la que se llevan a cabo las simulaciones. La primera actividad está acotada en el tiempo, y está situada justo antes de necesitar realizar cualquier simulación. La segunda actividad puede ser más extensa, y se sitúa temporalmente justo después de cada desarrollo teórico que ha de ser probado.

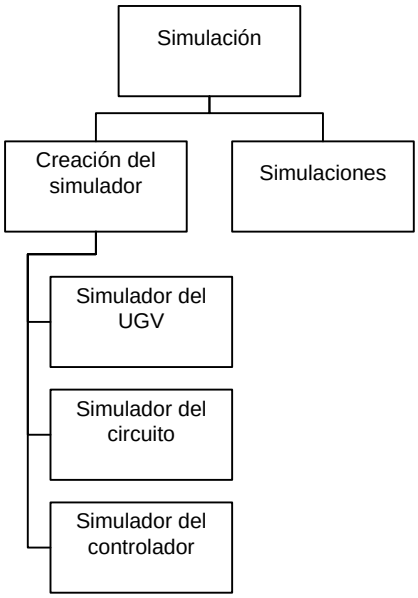


Ilustración 51: WP4

- Implementación: Es la última de las actividades centrales del proyecto. Una vez llevado a cabo el desarrollo teórico y su simulación hay que implementarlo en el UGV y comprobar los resultados. Se subdivide en dos actividades. La primera se lleva a cabo antes de cada implementación, y consiste en la programación del UGV en código C++. La segunda actividad es la prueba, con su toma y análisis de resultados. En caso de no ser satisfactorios, se ha de analizar la causa de éstos y volver a diseñarlo teóricamente.

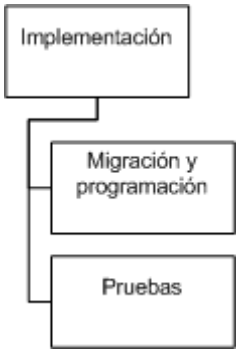


Ilustración 52: WP5

- Cierre del proyecto: La última actividad del proyecto es el cierre de éste, una vez se tenga implementado un controlador satisfactorio. En el cierre se elaboran las conclusiones y se escribe el proyecto con las conclusiones y resultados. Más tarde se escribirá un artículo con el objeto de difundir el conocimiento adquirido.

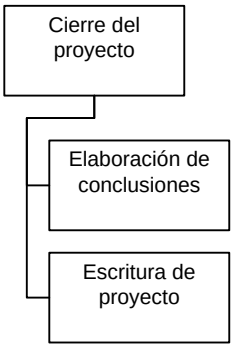


Ilustración 53: WP6

9.2.Planificación en el tiempo. Diagrama de GANTT

El proyecto se planificó para realizarse en el tiempo de 12 meses, teniendo en cuenta los periodos de vacaciones no lectivos. El proyecto se empezó el día 1 de octubre de 2009, y se debía haber terminado el 17 de septiembre de 2010.

Debido a complicaciones diversas, el proyecto ha sufrido un retraso de más de un mes, desplazándose el hito que marcaba el fin del proyecto hasta el 18 de octubre de 2010.

En el Anexo X se pueden ver el diagrama de GANTT planificado, y el diagrama de GANTT que representa la realidad del proyecto.

9.3.Planificación económica. Presupuesto

Debido a la cláusula de confidencialidad existente en el contrato del proyecto, el presupuesto no puede ser desvelado, no obstante, se propone el siguiente presupuesto a nivel orientativo:

Presupuesto		
Capítulo	**Título**	**Total**
I	Sistema mecánico	20.000,00 €
II	Sistema de accionamiento	4.000,00 €
III	Sistema de visión	2.500,00 €
IV	Sistema de control	2.700,00 €
Presupuesto total		**29.200,00 €**

Tabla 4: Resumen del presupuesto

Siendo los diferentes capítulos:

Capítulo I: Sistema mecánico				
Cantidad	**Unidad**	**Denominación**	**Precio unitario**	**Total**
Citroën C3				
1,00	ud	Citroën C3 pluriel con cambio automático	20.000,00 €	20.000,00 €
		Total capítulo I: Sistema mecánico		**20.000,00 €**

Tabla 5: Capitulo I del presupuesto

Capítulo II: Sistema de accionamiento				
Cantidad	**Unidad**	**Denominación**	**Precio unitario**	**Total**
Sistema de accionamiento				
1,00	ud	Sistema de accionamiento de pedales y giro de volante, controlado con un PC de bajo nivel	4.000,00 €	4.000,00 €
		Total capítulo II: Sistema de accionamiento		**4.000,00 €**

Tabla 6: Capitulo II del presupuesto

Capítulo III: Sistema de visión				
Cantidad	**Unidad**	**Denominación**	**Precio unitario**	**Total**
Sistema de visión				
1,00	ud	Sistema de visión que incluye cámara de captación de datos y software.	2.500,00 €	2.500,00 €
		Total capítulo III: Sistema de visión		**2.500,00 €**

Tabla 7: Capítulo III del presupuesto

Capítulo IV: Sistema de control				
Cantidad	**Unidad**	**Denominación**	**Precio unitario**	**Total**
Hardware de control				
1,00	ud	PC de control del UGV	1.500,00 €	1.500,00 €
1,00	ud	Electrónica necesaria para el control del UGV	500,00 €	500,00 €
Software de control				
1,00	ud	SW del PC de control	200,00 €	200,00 €
1,00	ud	SW del controlador del UGV	500,00 €	500,00 €
		Total capítulo IV: Sistema de control		**2.700,00 €**

Tabla 8: Capítulo IV del presupuesto

10. CONCLUSIONES Y TRABAJO FUTURO

A lo largo del proyecto se ha modelado cinemáticamente un UGV no holonómico tipo Ackermann, se ha diseñado un simulador de éste bajo el entorno de Matlab-Simulink; se ha diseñado y simulado una estructura de control que incluye un punto de linealización variable y calculado online y un controlador PI adaptativo; y se ha implementado dicha estructura de control sobre una plataforma de un vehículo real (un Citroën C3 Pluriel), consiguiendo un funcionamiento óptimo, siguiendo el circuito adecuadamente, sin perderlo y sin obtener brusquedad en la acción de control a velocidades variables hasta 50 km/h en rectas, 25 km/h en curvas cerradas con información del circuito y 20 km/h en curvas cerradas sin información del circuito.

Con todo ello se han conseguido cumplir con éxito los objetivos del proyecto, y se ha demostrado que es posible desarrollar un vehículo autónomo utilizando exclusivamente información visual del circuito pintado en el suelo.

Como trabajo futuro se deja la puerta abierta al diseño de un controlador de estructura PID, añadiendo la acción derivativa, aunque no sea teóricamente necesaria, para mejorar la suavidad del controlador.

Además se abre la posibilidad de hacer un modelado de la dinámica del volante o incluso de la dinámica del vehículo, para obtener así un modelo más exacto, recalculando entonces el controlador diseñado.

Otro posible trabajo futuro sería implementar un controlador con lógica borrosa (fuzzy) de más alto nivel para complementar el controlador desarrollado y así mejorar la respuesta del sistema minimizando los efectos de las no-linealidades del modelo, y de los errores de modelado.

11.BIBLIOGRAFÍA Y REFERENCIAS

(1) **Aparicio, F y otros. 2008.** *Ingeniería del transporte.* s.l. : CIE DOSAT 2000, 2008. 978-84-96437-82-1.

(2) **Aracil, R. y Jimenez, A. 1993.** *Sistemas discretos de control.* s.l. : Sección de publicaciones de la ETSII-UPM, 1993. 84-7484-014-7.

(3) **Barrientos, A., y otros. 2007.** *Fundamentos de robótica.* s.l. : McGrawHill, 2007. 978-84-481-5636-7.

(4) **Carelli, R. 2009.** *Conferencia: Control de robots móviles.* ETSII-UPM : s.n., 2009.

(5) **Collado, F. García. 2009.** *Proyecto fin de carrera.* Madrid : s.n., 2009.

(6) **Cruz, J.M. Diaz de la y Sánchez, A.M. 2002.** *Mecánica I.* s.l. : Sección de publicaciones de la ETSII-UPM, 2002. 84-7484-142-9.

(7) **Cruz, J.M. Díaz de la, Pérez, A.M. Sánchez y Herranz, F. Ramiro. 2001.** *Mecánica Analítica.* s.l. : Sección de publicaciones de la ETSII-UPM, 2001. 84-7484-145-3.

(8) **Dominguez, S., y otros. 2006.** *Control en el especio de estado.* s.l. : Pearson Prentice Hall, 2006. 879-84-8322-297-3.

(9) **García, O. Reinoso, y otros. 2004.** *Control de sistemas discretos.* s.l. : McGrawHill, 2004. 84-481-4204-7.

(10) **Jalón, J. García de. 2004.** *Aprenda Matlab 6.5 como si estuviera en primero.* 2004.

(11) **Joyanes, L. 2006.** *Programación en C++.* s.l. : McGrawHill, 2006. 84-481-4645-X.

(12) **Marcos, I. Ortiz y Alejo, F.J. Sánchez. 2005.** *El proyecto fin de carrera. Normas de realización, presentación y defensa.* s.l. : Sección de publicaciones de la ETSII-UPM, 2005. 84-7484-182-8.

(13) **Mathwork. 1998.** *Simulink. Dynamic System Simultion for Matlab. Writing S-Functiones. Version 3.* 1998.

(14) **Matía, F., y otros. 2003.** *Teoría de sistemas.* s.l. : Sección de publicaciones de la ETSII-UPM, 2003. 84-7484-158-5.

(15) **Mellodge, P. 2002.** *Thesis: Feedback Control for a Path Following Robotic Car.* 2002.

(16) **Mellodge, P. y Kachroo, P. 2008.** *Model Abstraction in Dynamical Systems: Aplication to Moble Robot Control.* s.l. : Springer, 2008. 978-3-540-70792-9.

(17) **Morin, P. y Samson, C. 2004.** *Trajectory tracking for non-holonomic vehicles: overview and case study.* s.l. : Route del Lucioles, 2004.

(18) **Pinto, E. y Matía, F. 2010.** *Fundamentos de control con Matlab.* s.l. : Pearson, 2010. 978-84-8322-651-3.

(19) **S.Gottfried, Byron. 1994.** *Programación en C.* s.l. : McGrawHill, 1994. 84-7615-572-7.

(20) **Siciliano y Kathib. 2008.** *Springer Handbook of Robotics.* s.l. : Springer, 2008. 978-3-540-23957-4.

(21) **Stumpf, Juan. 1996.** *Matlab edición estudiante. Versión 4. Guía de usuario.* s.l. : Prentice Hall, 1996. 0-13-459793-1.

(22) **Toibero, J.M., Roberti, F. y Carelli, R. 2009.** *Stable contour-following control of wheeled mobile robots.* s.l. : Robotica 27, 2009. Vol. 27.

(23) *http://cajamadrid.cronicasocial.com/anteriores/pg040405/nacional/salud/salud1.htm*

(24) *http://www.dgt.es*

(25) *http://paroyempleo.blogspot.com/2010/08/el-tiempo-de-desplazamiento-al-trabajo.html*

(26) *Programa AUTOPIA: http://www.iai.csic.es/autopia/*

(27) *Coches de Google: http://googleblog.blogspot.com/2010/10/what-were-driving-at.html*

(28) *DARPA urban challenge: http://www.darpa.mil/grandchallenge/index.asp*

ANEXO I: PRUEBAS DE VERIFICACIÓN DEL CONTROLADOR

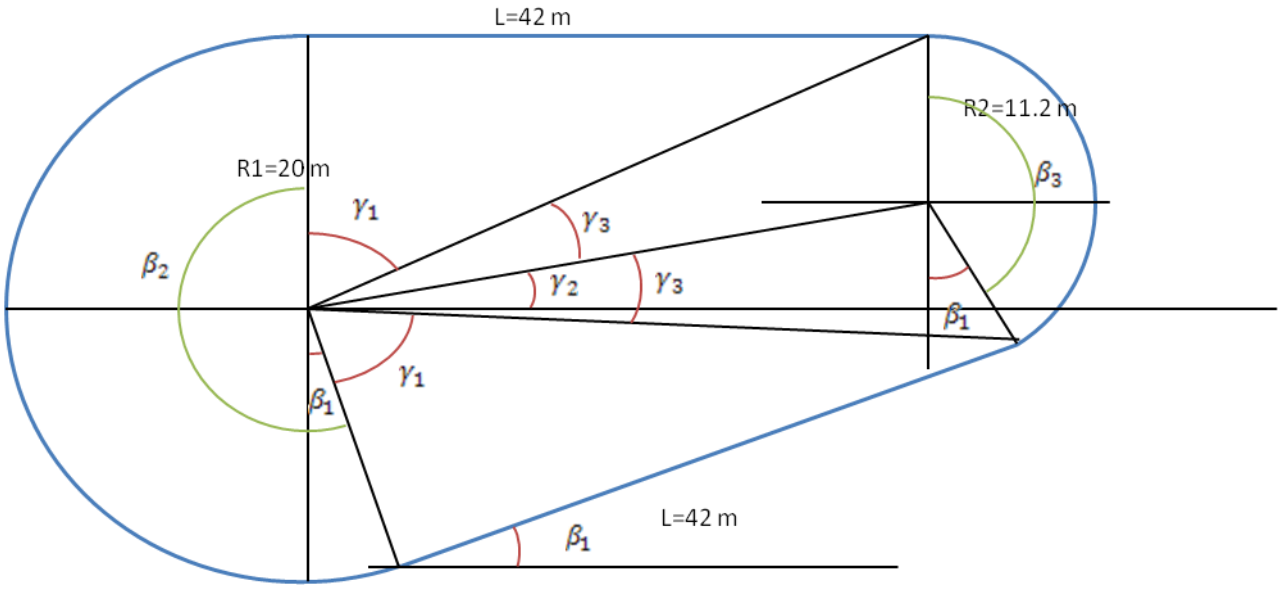
Se detallan en éste apartado las pruebas que se han de llevar a cabo para la aceptación del controlador.

C1		Seguimiento del circuito durante 5 vueltas seguidas con interpretacción de línea y detección e interpretación de códigos marcados en pavimento, con las siguientes velocidades máximas:
C1.1		1ª Recta (Recta inicial): 20 km/h
C1.2		Entrada 1ª curva: 15 km/h
C1.3		Aumento velocidad 1ª curva hasta los 20 km/h
C1.4		Salida 1ª curva: 15 km/h
C1.5		2ª Recta: 15 km/h
C1.6		Entrada 2ª curva: 13 km/h
C1.7		Salida 2ª curva: 13 km/h
C2		Seguimiento del circuito durante 5 vueltas seguidas con interpretacción de línea y detección e interpretación de códigos marcados en pavimento, con las siguientes velocidades máximas:
C2.1		1ª Recta (Recta inicial): 20 km/h
C2.2		Entrada 1ª curva: 15 km/h
C2.3		Detección marca frenado y arranque en curva
C2.4		Salida 1ª curva: 15 km/h
C2.5		2ª Recta: 15 km/h
C2.6		Entrada 2ª curva: 13 km/h
C2.7		Salida 2ª curva: 13 km/h

ANEXO II: DEFINICIÓN DEL CIRCUITO DE PRUEBAS DEL INSIA

El circuito de pruebas del INSIA viene descrito en el apartado 3.5. Lo que se hará ahora será calcularlo exactamente para su exacta definición en el plano y poder incluirlo en los simuladores.

Primeramente se calculan los ángulos de interés del circuito según se muestra en la figura:



Siendo:

$$\gamma_1 = \text{atan}\frac{L}{R_1} = 64.5367º$$

$$\gamma_2 = \text{atan}\frac{R_1 - R_2}{L} = 11.8336º$$

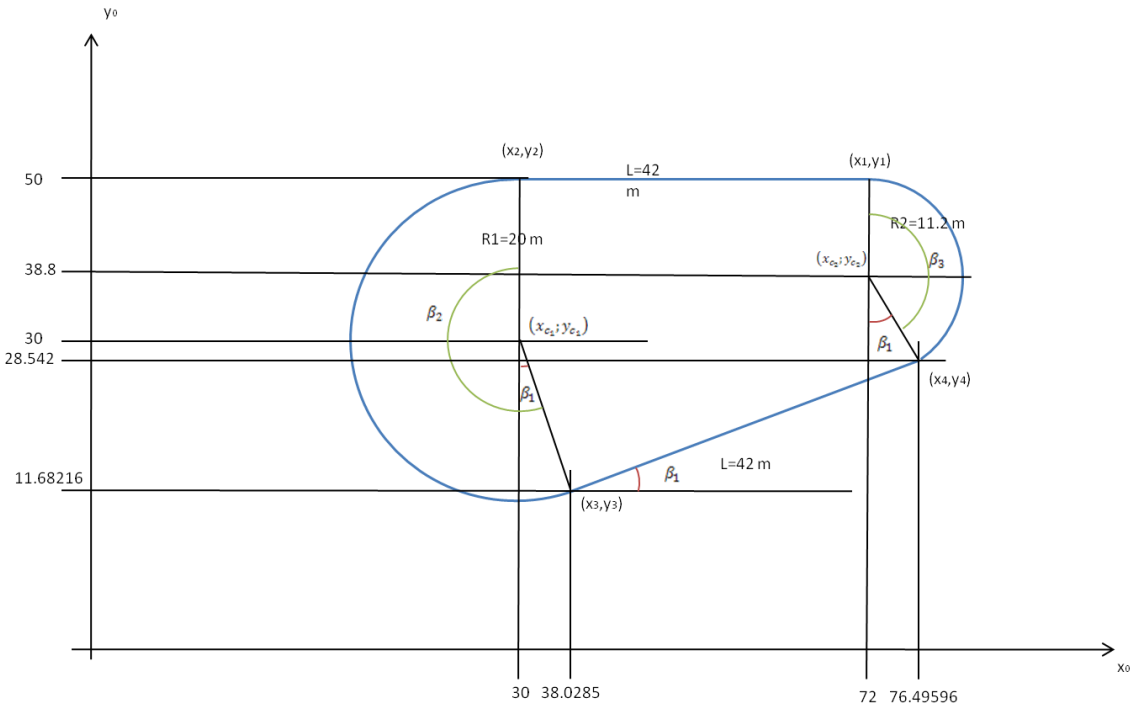
$$\gamma_3 = 90º - \gamma_1 - \gamma_2 = 13.6297º$$

$$\beta_1 = 180º - 2 \cdot \gamma_1 - 2 \cdot \gamma_3 = 2 \cdot \gamma_2 = 23.6673º$$

$$\beta_2 = 180º + \beta_1 = 203.6673º$$

$$\beta_3 = 180 - \beta_1 = 156.3327º$$

Para proseguir con el cálculo, se define un sistema de referencia fijo. Se puede elegir cualquiera, pero se tomará uno que haga que tanto el circuito como el UGV estén siempre en el primer cuadrante.



Donde los centros de las circunferencias vienen definidos como:

$$(x_{c_1}; y_{c_1}) = (30; 30)$$

$$(x_{c_2}; y_{c_2}) = (72; 38.8)$$

Y se han calculado los puntos de cambio de tramo como:

$$(x_1; y_1) = (72; 50)$$

$$(x_2; y_2) = (30; 50)$$

$$(x_3; y_3) = \left(x_{c_1} + R_1 \cdot \sin\beta_1 \,;\, y_{c_1} - R_1 \cdot \cos\beta_1\right) = (38.0285; 11.6822)$$

$$(x_4; y_4) = \left(x_{c_2} + R_2 \cdot \sin\beta_1 \,;\, y_{c_2} - R_2 \cdot \cos\beta_1\right) = (76.4960; 28.5420)$$

Una vez definido perfectamente el circuito, se puede proceder a su definición en los simuladores:

- <u>Definición en simulador relativo de circuitos:</u>

El código de definición del circuito en Matlab será:

```
%Definición de circunferencias
R1=20;
R2=11.2;
%Definicion de rectas
L_rect=42;
%Calculo de ángulos
B1=2*atan((R1-R2)/L_rect);
```

```
B2=pi+B1;
B3=pi-B1;

%Matriz  de circuito
MC=[0 L_rect 0; %Recta=0; Longitud=42; parámetro adicional=0;
    2 B2*R1 R1; %Curva izq=2; Angulo=203.6673°; parametro=Radio=20;
    0 L_rect 0; %Recta=0; Longitud=42; parámetro adicional=0;
    2 B3*R2 R2]; %Curva izq=2; Angulo=156.3327°; parametro=Radio=11.2;

%Indicar si circuito cerrado
cerrado=0; %Si es cerrado (se repite)
```

Habrá que incluir además el punto inicial en el que se encuentra el UGV, que puede ser:

```
%Punto inicial del UGV
s_inicial=0;
Theta_e_inicial=0;
d_inicial=-0.1;
```

- Definición en simulador de circuitos en el plano

El código de definición del circuito en Matlab será:

```
%Definición de circunferencias
xc1=30;yc1=30;R1=20;
xc2=72;yc2=38.8;R2=11.2;
%Definición de rectas
L_rect=42;
%Calculo de ángulo B1 y puntos para evitar errores de redondeo
B1=2*atan((R1-R2)/L_rect);
x1=72;y1=50;
x2=30;y2=50;
x3=xc1+R1*sin(B1);y3=yc1-R1*cos(B1);
x4=xc2+R2*sin(B1);y4=yc2-R2*cos(B1);

%Matriz  de componentes del circuito
Circuito.MCC=[0 x1 y1 pi 1;
              1 xc1 yc1 R1 1;
              0 x3 y3 B1 1;
              1 xc2 yc2 R2 1];

%Matriz de hilado del circuito
Circuito.MHC=[1 x1 y1;
              2 x2 y2;
              3 x3 y3;
              4 x4 y4;
              -1 x1 y1];
```

La definición del punto inicial podría ser:

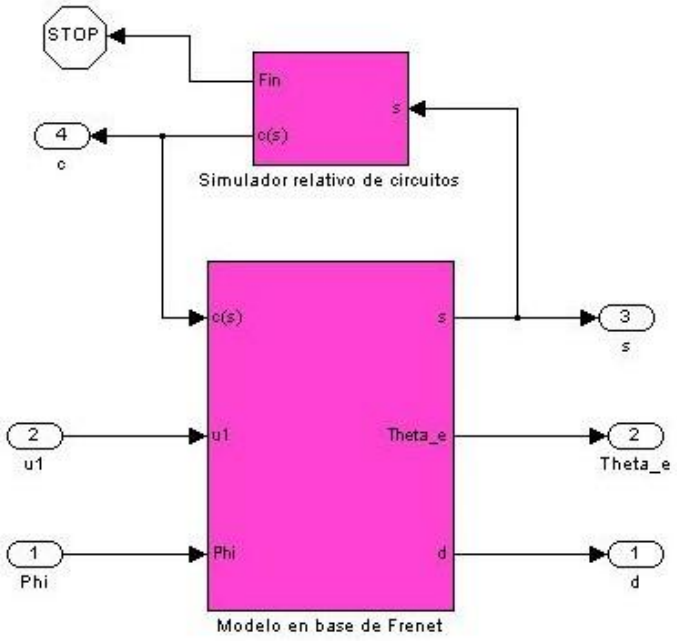
```
%Punto inicial del UGV
y_inicial=50.1;
Theta_inicial=pi;
x_inicial=72;
```

ANEXO III: SIMULADORES DEL CIRCUITO

En este anexo se expondrá los bloques de los simuladores de circuitos, tanto el relativo como el plano. Se mostrarán tanto los bloques como los códigos de cada bloque como la forma de introducir el circuito en el simulador.

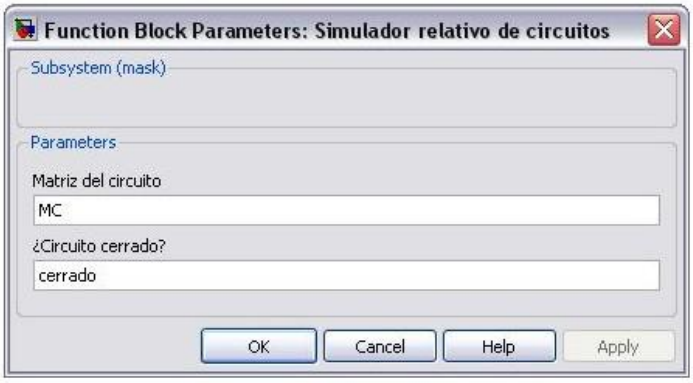
i. SIMULADOR RELATIVO DE CIRCUITOS

Como se vio en el apartado 5., el simulador en base de Frenet del UGV estaba compuesto por el modelo cinemático en base de Frenet, y el simulador relativo de circuitos.



Se analizará ahora el bloque simulador relativo de circuitos:

Dicho bloque requiere de dos parámetros

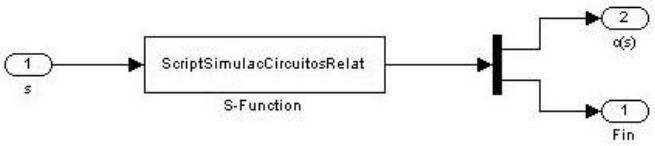


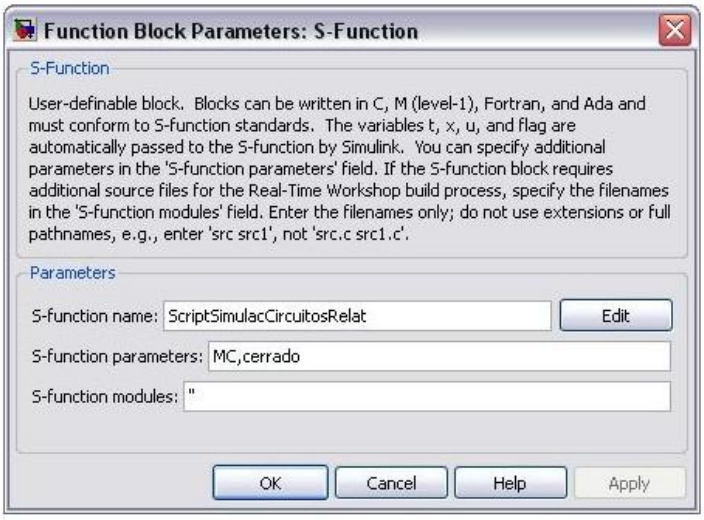
Dichos parámetros se definen del siguiente modo:

Matriz del circuito	MC	Es una variable matricial de dimensión nx3, donde n es el número de tramos diferentes que tiene el circuito.	En el caso de que el tramo i fuese una recta de longitud L, la fila i de la matriz sería: $[0;\ L;\ 0]$ Siendo el primer número el indicador de recta, el segundo el valor de la longitud, y el tercero un valor que es siempre cero. Si el tramo i fuese una circunferencia recorrida hacia la derecha, de radio R, y ángulo a, la fila i de la matriz sería: $[1;\ a \cdot R;\ R]$ Siendo el primer número el indicador de curva a la derecha, el segundo número el arco de circunferencia que ocupa el tramo, y el tercer número el radio de la circunferencia. Si el tramo i fuese una circunferencia recorrida hacia la izquierda, de radio R, y ángulo a, la fila i de la matriz sería: $[2;\ a \cdot R;\ R]$ Siendo el primer número el indicador de curva a la izquierda, el segundo número el arco de circunferencia que ocupa el tramo, y el tercer número el radio de la circunferencia.
¿Circuito cerrado?	cerrado	Es una variable binaria que indica si se debe repetir el circuito o no durante el tiempo de simulación.	Si cerrado=1, al recorrerse todos los tramos, mientras quede tiempo de simulación, se volverá de nuevo al primer tramo. Esto es útil para circuitos cerrados o circuitos que se repiten. Si cerrado=0, al recorrerse todos los

Se puede ver un ejemplo en el Anexo I.

El bloque simulador relativo de circuitos, internamente es:





El código de la S-Function es ScriptSimulacCircuitosRelat, e incluye el siguiente código:

```
function  [sys,x0,str,ts] =
ScriptSimulacCircuitosRelat(t,x,u,flag,MC,cerrado)


    switch flag
          %%%%%%%%%%%%%%%%%%
          % Initialization %
          %%%%%%%%%%%%%%%%%%
          case 0,
            [sys,x0,str,ts]=mdlInitializeSizes(MC);

          %%%%%%%%%%%%%%%%
          % Derivatives %
          %%%%%%%%%%%%%%%%
          case 1,
            sys=mdlDerivatives(t,x,u);

          %%%%%%%%%%
          % Update %
          %%%%%%%%%%
          case 2,
            sys=mdlUpdate(t,x,u,MC,cerrado);

          %%%%%%%%%%
          % Outputs %
          %%%%%%%%%%
          case 3,
            sys=mdlOutputs(t,x,u,MC,cerrado);

          %%%%%%%%%%%%%%%%%%%%%%%
          % GetTimeOfNextVarHit %
          %%%%%%%%%%%%%%%%%%%%%%%
          case 4,
```

```
            sys=mdlGetTimeOfNextVarHit(t,x,u);

          %%%%%%%%%%%%%
          % Terminate %
          %%%%%%%%%%%%%
          case 9,
            sys=mdlTerminate(t,x,u);

          %%%%%%%%%%%%%%%%%%%
          % Unexpected flags %
          %%%%%%%%%%%%%%%%%%%
          otherwise
            DAStudio.error('Simulink:blocks:unhandledFlag',
num2str(flag));

    end;

end
```

```
%
%=============================================================================
=====
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the S-
function.
%=============================================================================
=====
%
function [sys,x0,str,ts]=mdlInitializeSizes(MC)

    %
    % call simsizes for a sizes structure, fill it in and convert it to a
    % sizes array.
    %
    % Note that in this example, the values are hard coded.  This is not
a
    % recommended practice as the characteristics of the block are
typically
    % defined by the S-function parameters.
    %



    %Datos de entrada
    %MC = matriz del circuito
    %cerrado = indica si el circuito está cerrado y se repite (1) o acaba
en linea recta(0)

    %Estados
    %x=[tramo,s,sinic,sfin]
    %Entradas
    %u=[s]
    %Salidas
    %y=[c, fin]

    sizes = simsizes;

    sizes.NumContStates  = 0;      % number of continuous states
    sizes.NumDiscStates  = 3;      % number of discrete states
```

```
[tramo,sinic,sfin]
    sizes.NumOutputs     = 2;      % number of outputs  [c, fin]
    sizes.NumInputs      = 1;      % number of inputs   [s]
    sizes.DirFeedthrough = 0;      % direct feedthrough flag
    sizes.NumSampleTimes = 1;     % number of sample times, at least one
sample time is needed

    sys = simsizes(sizes);

    %
    % initialize the initial conditions
    %

    tramo=1;

    x0  = [tramo; %tramo
        0; %sinictramo
        MC(tramo,2)]; %sfintramo

    %
    % str is always an empty matrix
    %
    str = [];

    %
    % initialize the array of sample times
    %
    ts  = [-1 0]; % sample time: [period, offset]

    % end mdlInitializeSizes

end
```

```
%
%=============================================================================
=====
% mdlDerivatives
% Return the derivatives for the continuous states.
%=============================================================================
=====
%
function sys=mdlDerivatives(t,x,u)

    sys=[];

    % end mdlDerivatives

end
```

```
%
%=============================================================================
=====
```

```
% mdlUpdate
% Handle discrete state updates, sample time hits, and major time step
% requirements.
%=============================================================================
=====
%
function sys=mdlUpdate(t,x,u,MC,cerrado)


    %Actualizamos el tramo

    %Elegimos entradas
    s=u;

    %Elegimos el estado
    tramo=x(1);
    sinictramo=x(2);
    sfintramo=x(3);

    %Cálculo del tramo
    if tramo>0, %Todavia no hemos acabado el circuito
        %Comprobamos si seguimos en el tramo
        if s>=sfintramo, %Hemos acabado el tramo
            %Comprobamos si sigue habiendo circuito
            dim=size(MC);
            ntramos=dim(1); %Numero de tramos
            if (tramo==ntramos) %Hemos dado una vuelta
                if (cerrado==1), %Comprobamos si está cerrado
                    tramo=1; %Volvemos a empezar los tramos
                else %No esta cerrado
                    tramo=-1;
                end;
            else
                tramo=tramo+1;
            end;

            %Actualizamos valores en su caso
            if tramo>0, %No hemos acabado
                sinictramo=sfintramo;
                sfintramo=sinictramo+MC(tramo,2);
            end;
        end;
    end;


    %Definimos el estado, ya actualizado
    sys(1)=tramo;
    sys(2)=sinictramo;
    sys(3)=sfintramo;
    % end mdlUpdate

end
```

```
%
%=============================================================================
=====
% mdlOutputs
% Return the block outputs.
```

```
%=========================================================================
=====
%
function sys=mdlOutputs(t,x,u,MC,cerrado)

    %Recalculamos el tramo, para evitar errores

    %Elegimos entradas
    s=u;

    %Elegimos el estado
    tramo=x(1);
    sfintramo=x(3);

    %Cálculo del tramo
    if tramo>0, %Todavia no hemos acabado el circuito
        %Comprobamos si seguimos en el tramo
        if s>=sfintramo, %Hemos acabado el tramo
            %Comprobamos si sigue habiendo circuito
            dim=size(MC);
            ntramos=dim(1); %Numero de tramos
            if (tramo==ntramos) %Hemos dado una vuelta
                if (cerrado==1), %Comprobamos si está cerrado
                    tramo=1; %Volvemos a empezar los tramos
                else %No esta cerrado
                    tramo=-1;
                end;
            else
                tramo=tramo+1;
            end;

        end;
    end;



    %Calculamos las salidas
    %Definicion de la curvatura
    if tramo > 0,
        if MC(tramo,1) == 0, c=0; end;%Recta
        if MC(tramo,1) == 1, c=-1/MC(tramo,3); end;   %Curva derecha
        if MC(tramo,1) == 2, c=1/MC(tramo,3); end;  %Curva izquierda
    else c=0;
    end;


    %Miramos si es el fin
    fin=0;
    if(tramo==-1)
        fin=1;
    end;

    %Definimos la salida
    sys(1)=c;
    sys(2)=fin;


    % end mdlOutputs
```

```
end
```

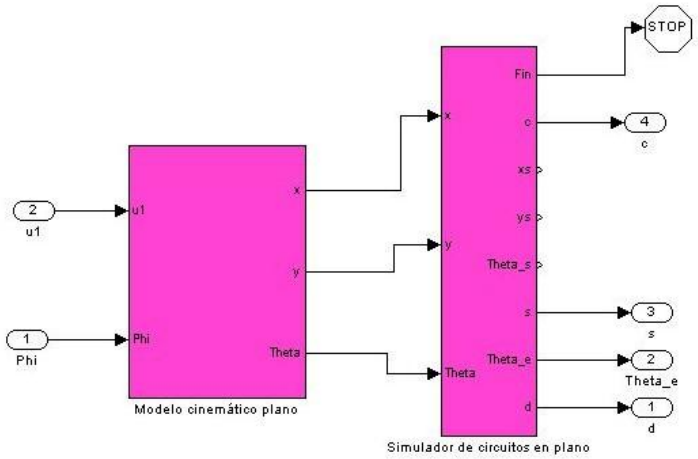
```
%
%=============================================================================
=====
% mdlGetTimeOfNextVarHit
% Return the time of the next hit for this block.  Note that the result
is
% absolute time.  Note that this function is only used when you specify a
% variable discrete-time sample time [-2 0] in the sample time array in
% mdlInitializeSizes.
%=============================================================================
=====
%
function sys=mdlGetTimeOfNextVarHit(t,x,u)

    %sampleTime = 1;    %  Example, set the next hit to be one second
later.
    %sys = t + sampleTime;

    sys=[];

    % end mdlGetTimeOfNextVarHit

end
```

```
%
%=============================================================================
=====
% mdlTerminate
% Perform any end of simulation tasks.
%=============================================================================
=====
%
function sys=mdlTerminate(t,x,u)

    sys = [];

    % end mdlTerminate

end
```

Como se observa, se ha utilizado el truco indicado en el Apartado 5., de duplicar el código tanto en Calculate Outputs como en Update Discrete States para conseguir mayor precisión y evitar los errores que se mencionaron en dicho apartado.

ii. SIMULADOR DE CIRCUITOS EN EL PLANO

Como se vio en el apartado 5., el simulador en el espacio físico del UGV estaba compuesto por el modelo cinemático plano, y el simulador de circuitos en el plano.



Se analizará ahora el bloque simulador de circuitos en el plano:

Siendo la definición del circuito del siguiente modo:

La variable Circuito es una estructura formada por dos matrices:

- *Matriz de componentes del circuito* (Circuito.MCC), que define los tramos que componen el circuito. Tendrá dimensión nx5 siendo n el número de tramos que componen el circuito.

  En el caso de querer definir una <u>recta</u> en el tramo i, la fila i estará compuesta por:

  Primer número: 0, indicador de recta

  Segundo número: xc, punto x caracteristico de la recta (m)

  Tercer número: yc, punto y caracteristico de la recta (m)

  Cuarto número: Angulo, Angulo de pendiente (radianes). Da idea del sentido de recorrer la curva (siempre positivo).

  Quinto número: cualquiera, no tiene importancia.

NOTA: La recta se caracteriza en el plano con un punto (xc, yc) y un ángulo de pendiente (Angulo).

En el caso de querer definir una *circunferencia* en el tramo i, la fila i estará compuesta por:

Primer número: 1, indicador de circunferencia

Segundo número: xcc, centro de circunferencia

Tercer número: ycc, centro de circunferencia

Cuarto número: R, radio de circunferencia

Quinto número: Indicación de sentido de recorrer la curva. Si vale 0, sentido horario; si vale 1, sentido antihorario

NOTA: La circunferencia se caracteriza en el plano por su centro (xcc, ycc), su radio R, y el sentido en el que se recorre.

- *Matriz de hilado del circuito* (Circuito.MHC), define los puntos de cambio de tramo, una vez determinados todos los tramos adecuadamente en la MCC. Tendrá dimensión (n+1)x3.

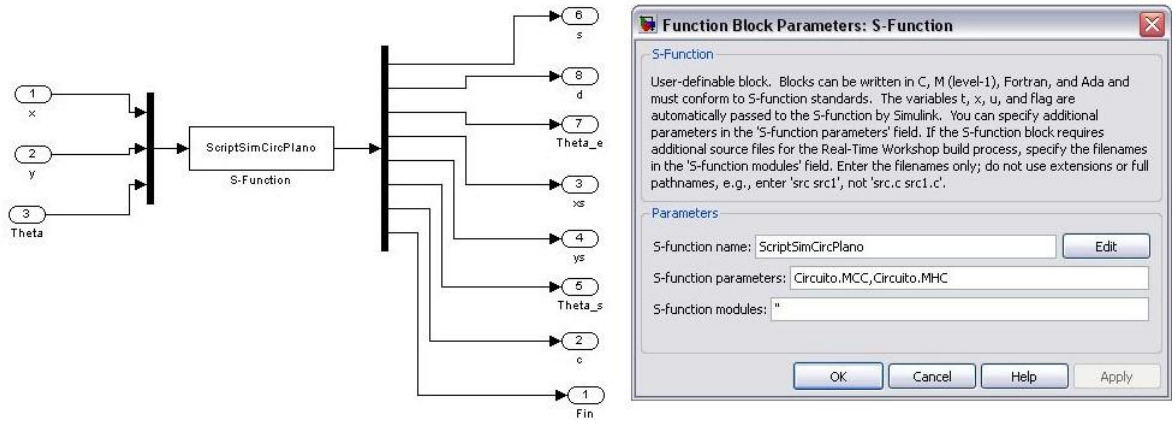
  El primer número de cada fila, indica el tramo que va a comenzar. El segundo y tercer número indican las coordenadas (x, y) del punto en el que comienza dicho tramo.

  Si el primer número vale -1, indica que no hay más tramos, y el punto definido a continuación será en el que acabe el tramo n.

El simulador no comprueba que el circuito esté bien definido, y dará un error en caso de no estarlo, asique es imprescindible su correcta definición.

Se puede ver un ejemplo de definición de la variable Circuito en el Anexo I.

El bloque del simulador de circuitos en el plano, internamente es:



El código de la S-Function está incluida en el archivo ScriptSimCircPlano, cuyo código es:

```
function  [sys,x0,str,ts] =
ScriptSimCircPlano(t,x,u,flag,CircuitoMCC,CircuitoMHC)
switch flag,
```

```
  %%%%%%%%%%%%%%%%%%%
  % Initialization %
  %%%%%%%%%%%%%%%%%%%
  case 0,
    [sys,x0,str,ts]=mdlInitializeSizes(CircuitoMCC,CircuitoMHC);

  %%%%%%%%%%%%%%%
  % Derivatives %
  %%%%%%%%%%%%%%%
  case 1,
    sys=mdlDerivatives(t,x,u);

  %%%%%%%%%%
  % Update %
  %%%%%%%%%%
  case 2,
    sys=mdlUpdate(t,x,u,CircuitoMCC,CircuitoMHC);

  %%%%%%%%%%
  % Outputs %
  %%%%%%%%%%
  case 3,
    sys=mdlOutputs(t,x,u);

  %%%%%%%%%%%%%%%%%%%%%%%
  % GetTimeOfNextVarHit %
  %%%%%%%%%%%%%%%%%%%%%%%
  case 4,
    sys=mdlGetTimeOfNextVarHit(t,x,u);

  %%%%%%%%%%%%
  % Terminate %
  %%%%%%%%%%%%
  case 9,
    sys=mdlTerminate(t,x,u);

  %%%%%%%%%%%%%%%%%%%
  % Unexpected flags %
  %%%%%%%%%%%%%%%%%%%
  otherwise
    DAStudio.error('Simulink:blocks:unhandledFlag', num2str(flag));

end

end
```

```
%
%=============================================================================
=====
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the S-
function.
%=============================================================================
=====
%
function [sys,x0,str,ts]=mdlInitializeSizes(CircuitoMCC,CircuitoMHC)

%
```

```
% call simsizes for a sizes structure, fill it in and convert it to a
% sizes array.
%
% Note that in this example, the values are hard coded.  This is not a
% recommended practice as the characteristics of the block are typically
% defined by the S-function parameters.
%

%Estados
%x=[s, d, Theta_e, xs, ys, Theta_s, c, fin, tramo, sinictramo, sfintramo,
xc0, yc0]
%Entradas
%u=[xp, yp, Theta]
%Salidas
%y=[s, d, Theta_e, xs, ys, Theta_s, c, fin]

sizes = simsizes;

sizes.NumContStates  = 0;      % number of continuous states
sizes.NumDiscStates  = 11;      % number of discrete states [s, d,
Theta_e, xs, ys, Theta_s, c] 1-7 [fin, tramo, sinictramo, sfintramo] 8-11
sizes.NumOutputs     = 8;      % number of outputs  [s, d, Theta_e, xs,
ys, Theta_s, c, fin]
sizes.NumInputs      = 3;      % number of inputs   [x, y, Theta]
sizes.DirFeedthrough = 0;      % direct feedthrough flag
sizes.NumSampleTimes = 1;     % number of sample times, at least one
sample time is needed

sys = simsizes(sizes);

%
% initialize the initial conditions
%

%Calculamos la distancia sfintramo
sfintramo=ObtenAvanceLocal(CircuitoMHC(1,2),CircuitoMHC(1,3),CircuitoMHC(
2,2),CircuitoMHC(2,3),1,CircuitoMCC);



x0  = [0; %s
    0; %d
    0; %Theta_e
    CircuitoMHC(1,2); %xs
    CircuitoMHC(1,3); %ys
    0; %Theta_s
    0; %c
    0; %fin
    1; %tramo
    0; %sinictramo
    sfintramo]; %sfintramo

%
% str is always an empty matrix
%
str = [];

%
% initialize the array of sample times
```

```
%
ts  = [-1 0]; % sample time: [period, offset]

% end mdlInitializeSizes

%Pintamos
%figure;
%hold on;


End
```

```
%
%=============================================================================
=====
% mdlDerivatives
% Return the derivatives for the continuous states.
%=============================================================================
=====
%
function sys=mdlDerivatives(t,x,u)

    sys=[];

% end mdlDerivatives

end
```

```
%
%=============================================================================
=====
% mdlUpdate
% Handle discrete state updates, sample time hits, and major time step
% requirements.
%=============================================================================
=====
%
function sys=mdlUpdate(t,x,u,CircuitoMCC,CircuitoMHC)

    %Lectura de Estados anteriores
    s_ant=x(1);
    d_ant=x(2);
    Theta_e_ant=x(3);
    xs_ant=x(4);
    ys_ant=x(5);
    Theta_s_ant=x(6);
    c_ant=x(7);
    fin=x(8);
    tramo=x(9);
    sinictramo=x(10);
    sfintramo=x(11);
    %Lectura de entradas
    xp=u(1);
    yp=u(2);
    Theta=u(3);
```

```
    %Funcion
     %Comprobamos que el circuito no ha acabado en el paso anterior
    if fin ~= 1 %Todavia hay circuito
        %Hacemos la hipótesis de que el circuito no ha acabado

[xs,ys,Theta_s,s,d,Theta_e,c,tramo,sinictramo,sfintramo,fin]=BucleCircuit
o(xp,yp,Theta,Theta_s_ant,s_ant,tramo,sinictramo,sfintramo,CircuitoMCC,Ci
rcuitoMHC);

        %Comprobamos la hipótesis
        if fin == 1 %No se cumple la hipótesis
            %Manten valores antiguos
            xs=xs_ant;
            ys=ys_ant;
            Theta_s=Theta_s_ant;
            s=s_ant;
            d=d_ant;
            Theta_e=Theta_e_ant;
            c=c_ant;
        end;
    else
        %El circuito ha acabado. Mantén valores
        xs=xs_ant;
        ys=ys_ant;
        Theta_s=Theta_s_ant;
        s=s_ant;
        d=d_ant;
        Theta_e=Theta_e_ant;
        c=c_ant;
    end;

    %Comprobamos c
    l1=3.41;
    if(abs(c)>1/l1)
        c=c_ant;
    end;

    %Actualizamos estados

    %Necesarios para poder sacarlos
    sys(1)=s; %Almacena info
    sys(2)=d;
    sys(3)=Theta_e;
    sys(4)=xs;
    sys(5)=ys;
    sys(6)=Theta_s; %Almacena info
    sys(7)=c;
    sys(8)=fin;
    %Almacenan info
    sys(9)=tramo;
    sys(10)=sinictramo;
    sys(11)=sfintramo;

    sys;



    % end mdlUpdate

end
```

```
%
%==========================================================================
=====
% mdlOutputs
% Return the block outputs.
%==========================================================================
=====
%
function sys=mdlOutputs(t,x,u)

%Definimos la salida
sys(1)=x(1); %s;
sys(2)=x(2); %d;
sys(3)=x(3); %Theta_e;
sys(4)=x(4); %xs;
sys(5)=x(5); %ys;
sys(6)=x(6); %Theta_s;
sys(7)=x(7); %c;
sys(8)=x(8); %fin;

% end mdlOutputs

end
```

```
 %
%==========================================================================
=====
% mdlGetTimeOfNextVarHit
% Return the time of the next hit for this block.  Note that the result
is
% absolute time.  Note that this function is only used when you specify a
% variable discrete-time sample time [-2 0] in the sample time array in
% mdlInitializeSizes.
%==========================================================================
=====
%
function sys=mdlGetTimeOfNextVarHit(t,x,u)

sys=[];

% end mdlGetTimeOfNextVarHit

end
```

```
%
%==========================================================================
=====
% mdlTerminate
% Perform any end of simulation tasks.
%==========================================================================
=====
%
function sys=mdlTerminate(t,x,u)
```

```
sys = [];

% end mdlTerminate

end
```

Como se observa, en éste simulador no se ha utilizado la misma artimaña de duplicar el código para evitar el error mencionado en el Apartado 5., debido a que sería muy costoso computacionalmente. Se asume el error como una perturbación no modelada.

Las funciones que faltan por definirse son:

```
function
[xs,ys,Theta_s,s,d,Theta_e,c,tramo,sinictramo,sfintramo,fin]=BucleCircuit
o(xp,yp,Theta,Theta_s_ant,s_ant,tramo,sinictramo,sfintramo,CircuitoMCC,Ci
rcuitoMHC)



    %Valores iniciales
    fin=0; %No ha acabado el circuito

    %Suponemos que el tramo todavia no se ha acabado

    %CALCULAMOS EL TRAMO

[xs,ys,Theta_s,s,d,Theta_e,c]=ObtenTramo(xp,yp,Theta,Theta_s_ant,s_ant,tr
amo,sinictramo,CircuitoMCC,CircuitoMHC);

    %Comprobamos la hipotesis
    if (s<=sfintramo)
        %Todo correcto. No hacemos nada

    else
        %El tramo se ha acabado
        %Hay que cambiar de tramo y recalcular

        %Comprobamos que hay mas tramos
        dim=size(CircuitoMCC);
        if (tramo<dim(1)) %Todavia qeda otro tramo del circuito
            %Nuevo tramo
            tramo=tramo+1;

            %Actualizamos valor de sinictramo
            sinictramo=sfintramo;

            %Actualizamos valor de sfintramo

s_tramo_total=ObtenAvanceLocal(CircuitoMHC(tramo,2),CircuitoMHC(tramo,3),
CircuitoMHC(tramo+1,2),CircuitoMHC(tramo+1,3),tramo,CircuitoMCC);
            sfintramo=sinictramo+s_tramo_total;

            %Recalculamos EL TRAMO

[xs,ys,Theta_s,s,d,Theta_e,c]=ObtenTramo(xp,yp,Theta,Theta_s_ant,s_ant,tr
amo,sinictramo,CircuitoMCC,CircuitoMHC);
```

```
        else %No queda circuito
            fin=1;

        end;

    end;

end
```

```
function
[xs,ys,Theta_s,s,d,Theta_e,c]=ObtenTramo(xp,yp,Theta,Theta_s_ant,s_ant,tr
amo,sinictramo,CircuitoMCC,CircuitoMHC)

    %Calculamos el punto
    [xs,ys,d]=ObtenPuntoCurva(xp,yp,tramo,CircuitoMCC);
    %Calculamos los ángulos
    Theta_s=ObtenAnguloCurva(Theta,xs,ys,tramo,CircuitoMCC);
    %Calculamos Theta_e
    Theta_e=Theta-Theta_s;
    %Calculamos avance local sobre la curva

s_local=ObtenAvanceLocal(CircuitoMHC(tramo,2),CircuitoMHC(tramo,3),xs,ys,
tramo,CircuitoMCC);
    %%%%OJO, si el coche va hacia atrás funciona mal!!
    s=sinictramo+s_local;
    %Calculamos c
    c=Calcula_c(s,s_ant,Theta_s,Theta_s_ant);

end
```

```
function [xs,ys,d]=ObtenPuntoCurva(xp,yp,tramo,CircuitoMCC)

    %COMPROBAMOS EL TIPO DE TRAMO EN EL QUE ESTAMOS
    tipotramo=CircuitoMCC(tramo,1);

    %ELEGIMOS EL TIPO DE TRAMO
    switch tipotramo
        %RECTA
        case 0
            %Leemos los parametros de la recta
            xc0=CircuitoMCC(tramo,2);
            yc0=CircuitoMCC(tramo,3);
            AngPend=CircuitoMCC(tramo,4);

            %Calculamos el punto sobre la recta
            [xs,ys,d]=CalculaPuntoRecta(xp,yp,xc0,yc0,AngPend);


        %CIRCUNFERENCIA
        case 1, %Estamos en una circunferencia
            %Leemos los parámetros
            xcc=CircuitoMCC(tramo,2);
            ycc=CircuitoMCC(tramo,3);
```

```
            R=CircuitoMCC(tramo,4);
            sentido_curva=CircuitoMCC(tramo,5);

            %Calculamos el punto sobre la circunferencia

[xs,ys,d]=CalculaPuntoCircunf(xp,yp,xcc,ycc,R,sentido_curva);

    end

end
```

```
function [xs,ys,d]=CalculaPuntoRecta(xp,yp,xc0,yc0,AngPend)


    %CALCULAMOS EL PUNTO SOBRE LA RECTA [xs, ys]
    switch AngPend
        case {0,pi} %AngPend=0,180°
            %Calcula xs, ys
            ys=yc0;
            xs=xp;
        case {pi/2,3*pi/2} %AngPend=90,270°
            %Calcula xs, ys
            ys=yp;
            xs=xc0;
        otherwise %AngPend cualquiera
            %Calcula xs, ys
            xs=(yp-yc0+tan(pi/2-
AngPend)*xp+tan(AngPend)*xc0)/(tan(AngPend)+tan(pi/2-AngPend));
            ys=yp+tan(pi/2-AngPend)*(xp-xs);
     end;

     %CALCULAMOS LA DISTANCIA d
     %Módulo
     modulod=sqrt((xp-xs)^2+(yp-ys)^2);
     %Sentido
     sentidod=sign(cos(AngPend)*(yp-ys)-sin(AngPend)*(xp-xs));
     %d
     d=sentidod*modulod;


end
```

```
function [xs,ys,d]=CalculaPuntoCircunf(xp,yp,xcc,ycc,R,sentido_curva)


    %CALCULAMOS EL PUNTO SOBRE LA CURVA [xs,ys] y el módulo de la
distancia
    %Posibles puntos
    if ((xp-xcc)~=0 ) && ((yp-ycc)~=0)
        %Resolvemos
        %[xs,ys]=solve('(xp-xcc)*ys-ycc*(xp-xcc)-(yp-ycc)*(xs-xcc)','(xs-
xcc)^2+(ys-ycc)^2-R^2','xs','ys');
        %Evaluamos
        %xs=eval(xs);
```

```
        %ys=eval(ys);

        %Resolviendo la ecuación, se obtiene:
        xs=[(-(-2*ycc^2*yp+ycc*yp^2+xcc^2*ycc+xp^2*ycc-
2*xcc*ycc*xp+ycc^3+(6*yp^2*ycc^2*R^2-4*yp^3*ycc*R^2-2*xcc*xp*R^2*yp^2-
4*yp*ycc^3*R^2+4*xcc*xp*R^2*yp*ycc+xp^2*R^2*yp^2-2*xcc*xp*R^2*ycc^2-
2*xp^2*R^2*yp*ycc-
2*xcc^2*R^2*yp*ycc+yp^4*R^2+xp^2*R^2*ycc^2+xcc^2*R^2*yp^2+xcc^2*R^2*ycc^2
+ycc^4*R^2)^(1/2))/(-2*yp*ycc-2*xcc*xp+xp^2+xcc^2+yp^2+ycc^2)*xp+(-
2*ycc^2*yp+ycc*yp^2+xcc^2*ycc+xp^2*ycc-
2*xcc*ycc*xp+ycc^3+(6*yp^2*ycc^2*R^2-4*yp^3*ycc*R^2-2*xcc*xp*R^2*yp^2-
4*yp*ycc^3*R^2+4*xcc*xp*R^2*yp*ycc+xp^2*R^2*yp^2-2*xcc*xp*R^2*ycc^2-
2*xp^2*R^2*yp*ycc-
2*xcc^2*R^2*yp*ycc+yp^4*R^2+xp^2*R^2*ycc^2+xcc^2*R^2*yp^2+xcc^2*R^2*ycc^2
+ycc^4*R^2)^(1/2))/(-2*yp*ycc-2*xcc*xp+xp^2+xcc^2+yp^2+ycc^2)*xcc+ycc*xp-
yp*xcc)/(-yp+ycc);
            (-(-2*ycc^2*yp+ycc*yp^2+xcc^2*ycc+xp^2*ycc-
2*xcc*ycc*xp+ycc^3-(6*yp^2*ycc^2*R^2-4*yp^3*ycc*R^2-2*xcc*xp*R^2*yp^2-
4*yp*ycc^3*R^2+4*xcc*xp*R^2*yp*ycc+xp^2*R^2*yp^2-2*xcc*xp*R^2*ycc^2-
2*xp^2*R^2*yp*ycc-
2*xcc^2*R^2*yp*ycc+yp^4*R^2+xp^2*R^2*ycc^2+xcc^2*R^2*yp^2+xcc^2*R^2*ycc^2
+ycc^4*R^2)^(1/2))/(-2*yp*ycc-2*xcc*xp+xp^2+xcc^2+yp^2+ycc^2)*xp+(-
2*ycc^2*yp+ycc*yp^2+xcc^2*ycc+xp^2*ycc-2*xcc*ycc*xp+ycc^3-
(6*yp^2*ycc^2*R^2-4*yp^3*ycc*R^2-2*xcc*xp*R^2*yp^2-
4*yp*ycc^3*R^2+4*xcc*xp*R^2*yp*ycc+xp^2*R^2*yp^2-2*xcc*xp*R^2*ycc^2-
2*xp^2*R^2*yp*ycc-
2*xcc^2*R^2*yp*ycc+yp^4*R^2+xp^2*R^2*ycc^2+xcc^2*R^2*yp^2+xcc^2*R^2*ycc^2
+ycc^4*R^2)^(1/2))/(-2*yp*ycc-2*xcc*xp+xp^2+xcc^2+yp^2+ycc^2)*xcc+ycc*xp-
yp*xcc)/(-yp+ycc)];

        ys=[(-2*ycc^2*yp+ycc*yp^2+xcc^2*ycc+xp^2*ycc-
2*xcc*ycc*xp+ycc^3+(6*yp^2*ycc^2*R^2-4*yp^3*ycc*R^2-2*xcc*xp*R^2*yp^2-
4*yp*ycc^3*R^2+4*xcc*xp*R^2*yp*ycc+xp^2*R^2*yp^2-2*xcc*xp*R^2*ycc^2-
2*xp^2*R^2*yp*ycc-
2*xcc^2*R^2*yp*ycc+yp^4*R^2+xp^2*R^2*ycc^2+xcc^2*R^2*yp^2+xcc^2*R^2*ycc^2
+ycc^4*R^2)^(1/2))/(-2*yp*ycc-2*xcc*xp+xp^2+xcc^2+yp^2+ycc^2);
            (-2*ycc^2*yp+ycc*yp^2+xcc^2*ycc+xp^2*ycc-2*xcc*ycc*xp+ycc^3-
(6*yp^2*ycc^2*R^2-4*yp^3*ycc*R^2-2*xcc*xp*R^2*yp^2-
4*yp*ycc^3*R^2+4*xcc*xp*R^2*yp*ycc+xp^2*R^2*yp^2-2*xcc*xp*R^2*ycc^2-
2*xp^2*R^2*yp*ycc-
2*xcc^2*R^2*yp*ycc+yp^4*R^2+xp^2*R^2*ycc^2+xcc^2*R^2*yp^2+xcc^2*R^2*ycc^2
+ycc^4*R^2)^(1/2))/(-2*yp*ycc-2*xcc*xp+xp^2+xcc^2+yp^2+ycc^2)];

        %Puntos:
        P1=[xs(1),ys(1)];
        dP1=sqrt((yp-P1(2))^2+(xp-P1(1))^2);
        P2=[xs(2),ys(2)];
        dP2=sqrt((yp-P2(2))^2+(xp-P2(1))^2);
    elseif ( (xp-xcc)==0 ) && ((yp-ycc)~=0 )
        xs=xcc;
        ys=[ycc+R,ycc-R];
        %Puntos
        P1=[xs, ys(1)];
        dP1=abs(yp-P1(2));
        P2=[xs, ys(2)];
        dP2=abs(yp-P2(2));
    elseif ( (xp-xcc)~=0 ) && ((yp-ycc)==0 )
        xs=[xcc+R,ycc-R];
        ys=ycc;
        %Puntos
        P1=[xs(1), ys];
```

```
        dP1=abs(xp-P1(1));
        P2=[xs(2), ys];
        dP2=abs(xp-P2(1));
    elseif ( (xp-xcc)==0 ) && ((yp-ycc)==0 )
        %Estamos en el centro y cualquier punto valdría
        %no se calcula, produciria un error
        display('Error 2 en CalculaPuntoCircunf');
    end;
    %ELECCIÓN DEL PUNTO DE DISTANCIA MINIMA!!
    if dP1<dP2
        modulod=dP1;
        P=P1;
    elseif dP1>dP2
        modulod=dP2;
        P=P2;
    elseif dP1==dP2
        %Estamos en el centro
        display('Error 2 en CalculaPuntoCircunf');
    end;

    %Actualizamos valores del punto
    xs=P(1);
    ys=P(2);


    %Comprobamos que está bien. El punto cae en la circunferencia
    %Definimos una tolerancia
    Tolerancia=1e-4;
    if ( (xs-xcc)^2+(ys-ycc)^2>=R^2-Tolerancia && (xs-xcc)^2+(ys-
ycc)^2<=R^2+Tolerancia )
        %Correcto
    else
        %Problema
        display('Error 1 en CalculaPuntoCircunf');
    end;


    %CALCULO sentidod
    %Distancia del coche al centro
    dc=sqrt((yp-ycc)^2+(xp-xcc)^2);
    %Elegimo el sentido de d segun se recorra la curva
    switch sentido_curva,
        case 0, %Horario
            if (dc>R)
                sentidod=1;
            elseif (dc<R)
                sentidod=-1;
            elseif (dc==R) %Estamos sobre la curva
                sentidod=0;
            end;
        case 1, %Antihorario
            if (dc>R)
                sentidod=-1;
            elseif (dc<R)
                sentidod=1;
            elseif (dc==R) %Estamos sobre la curva
                sentidod=0;
            end;
    end;
```

```
    %CALCULO DE d
    d=sentidod*modulod;


end
```

```
function Theta_s=ObtenAnguloCurva(Theta,xs,ys,tramo,CircuitoMCC)

    %COMPROBAMOS EL TIPO DE TRAMO EN EL QUE ESTAMOS
    tipotramo=CircuitoMCC(tramo,1);

    %ELEGIMOS EL TIPO DE TRAMO
    switch tipotramo
        %RECTA
        case 0
            %Leemos los parametros de la recta
            xc0=CircuitoMCC(tramo,2);
            yc0=CircuitoMCC(tramo,3);
            AngPend=CircuitoMCC(tramo,4);
            sentido_curva=CircuitoMCC(tramo,5);

            %Calculamos el angulo sobre la recta

Theta_s=CalculaAnguloRecta(Theta,xs,ys,xc0,yc0,AngPend,sentido_curva);


        %CIRCUNFERENCIA
        case 1, %Estamos en una circunferencia
            %Leemos los parámetros
            xcc=CircuitoMCC(tramo,2);
            ycc=CircuitoMCC(tramo,3);
            R=CircuitoMCC(tramo,4);
            sentido_curva=CircuitoMCC(tramo,5);

            %Calculamos el angulo sobre la circunferencia

Theta_s=CalculaAnguloCircunf(Theta,xs,ys,xcc,ycc,R,sentido_curva);

    end


    %Por si acaso
    Theta_s=real(Theta_s);

end
```

```
function
Theta_s=CalculaAnguloRecta(Theta,xs,ys,xc0,yc0,AngPend,sent_recta)

     %Ponemos Theta_s en el mismo numero de vueltas que Theta
     Num=CalculaNumeroVueltas(Theta);
     Theta_s=AngPend+2*pi*Num;

end
```

```
function
Theta_s=CalculaAnguloCircunf(Theta,xs,ys,xcc,ycc,R,sentido_curva)


    %CALCULO DE Theta_s
    Ang_sinsent=CalculaAnguloCosSen((xs-xcc)/R,(ys-ycc)/R);
    %Se ha obtenido un ánuglo 0<=Theta_s_sinsent<2*pi


    %Adaptamos el angulo al sentido
    switch sentido_curva
        case 0 %Horario
            Theta_s=-Ang_sinsent-pi/2;
        case 1 %Anthorario
            Theta_s=Ang_sinsent+pi/2;
    end;


    Num_coche=CalculaNumeroVueltas(Theta);
    Num_circ=CalculaNumeroVueltas(Theta_s);
    if(abs(Num_coche)>abs(Num_circ))
        Theta_s=Theta_s+2*pi*(Num_coche-Num_circ);
    end;

end
```

```
function Num=CalculaNumeroVueltas(Theta)

Num=sign(Theta)*floor(abs(Theta/(2*pi)));

end
```

```
function [a]=CalculaAnguloCosSen(cose,sen)


    %Definimos una tolerancia
    Tolerancia=1e-4;
    %Comprobamos si se cumple el teorema fundamental de la trigonometrís
    if ( (cose^2+sen^2)<=1+Tolerancia && (cose^2+sen^2)>=1-Tolerancia)
        %Todo bien
    else
        %Error 1
        display('Error 1 en CalculaAnguloCosSen');
    end;

    a1=acos(abs(cose));


    if cose>=0 && sen>=0
       a=a1;
    elseif cose>0 && sen<0
       a=2*pi-a1;
```

```
    elseif cose<0 && sen>0
       a=pi-a1;
    elseif cose<=0 && sen<=0
       a=pi+a1;
    end;


    %Comprobamos que 0 <= a < 2Pi, y en caso de no estar bien lo
reducimos
    a=ReduceAngPos(a);

    %Comprobamos que todo ha ido bien con el teorema fundamental de la
    %trigonometria
    %Definimos una Tolerancia
    Tolerancia=1e-4;
    if ( (cos(a)^2+sin(a)^2)<=1+Tolerancia && (cos(a)^2+sin(a)^2)>=1-
Tolerancia)
        %todo bien
    else
        %Error 2
        display('Error 2 en CalculaAnguloCosSen');
    end;


end
```

```
%Calcula el avance local (s_local) desde un punto inicial (x1,y1) a uno
final (x2,y2) en funcion del
%tramo en el que estemos
function s_local=ObtenAvanceLocal(x1,y1,x2,y2,tramo,CircuitoMCC)

    %COMPROBAMOS EL TIPO DE TRAMO EN EL QUE ESTAMOS
    tipotramo=CircuitoMCC(tramo,1);

    switch tipotramo

        %RECTA
        case 0
            %Lectura de parametros
            AnguloPendiente=CircuitoMCC(tramo,4);

            %Calcula s del tramo

s_local=CalculoAvanceLocalRecta(x1,y1,x2,y2,AnguloPendiente);


        %CIRCUNFERENCIA
        case 1
            %Lectura de parametros
            xcc=CircuitoMCC(tramo,2);
            ycc=CircuitoMCC(tramo,3);
            R=CircuitoMCC(tramo,4);
            sentido_curva=CircuitoMCC(tramo,5);


            %Calculamos el avance local sobre la circunferencia
```

```
s_local=CalculoAvanceLocalCircunf(x1,y1,x2,y2,xcc,ycc,R,sentido_curva);


    end;

end
```

```
%Calcula el avance local (s_local) desde un punto inicial (x1,y1) a uno
%final (x2,y2) sobre la recta definida por su ángulo de pendiente
%Da idea de hacia donde se recorre
function s_local=CalculoAvanceLocalRecta(x1,y1,x2,y2,AnguloPendiente)


    %Calcula s del tramo en valor absoluto
    s_abs=sqrt((x1-x2)^2+(y1-y2)^2);

    %Calculamos el sentido del avance
    sent=sign((x2-x1)*(cos(AnguloPendiente))+(y2-
y1)*(sin(AnguloPendiente)));

    %Calculamos el avance con sentido
    s_local=sent*s_abs;

end
```

```
function
s_local=CalculoAvanceLocalCircunf(x1,y1,x2,y2,xcc,ycc,R,sentido_curva)


    %Calculamos el ángulo entre los dos vectores, recorrido en
    %sentido antihorario >0
    a_antih=CalculaAnguloVect(x1-xcc,y1-ycc,x2-xcc,y2-ycc);


    %Calculo del angulo teniendo en cuenta el sentido
    if a_antih==0 %Angulo nulo
        a=0;
    else
        switch sentido_curva
            case 0 %Horario
                a=2*pi-a_antih;
            case 1 %Antihorario
                a=a_antih;

        end;
    end;

    %Calculo de s
    s_local=a*R;

end
```

```
%Funcion que calcula el ángulo positivo <2*pi que forman dos vectores
%Angulo del primero al segundo en sentido antihorario >0!!!!
function [a]=CalculaAnguloVect(x1,y1,x2,y2)


    %Comprobamos que no hay errores al introducir los vectores
    %Los vecotres han de tener módulo
    if ( sqrt(x1^2+y1^2)~=0 ) && ( sqrt(x2^2+y2^2)~=0 )
        %Todo correcto
    else
        %Error 1
        display('Error 1 en CalculaAnguloVect');
    end;

    %Calculamos el seno y coseno
    %Producto escalar
    cose=((x1)*(x2)+(y1)*(y2))/(sqrt(x1^2+y1^2)*sqrt(x2^2+y2^2));
    %Producto vectorial
    sen=((x1)*(y2)-(x2)*(y1))/(sqrt(x1^2+y1^2)*sqrt(x2^2+y2^2));

    %Calculamos el ángulo
    a=CalculaAnguloCosSen(cose,sen);

    %Comprobamos
    %Definimos una tolerancia
    Tolerancia=1e-4;
    if ( ( sin(a)<=sen+Tolerancia && sin(a)>=sen-Tolerancia ) && (
cos(a)<=cose+Tolerancia && cos(a)>=cose-Tolerancia ) )
        %Todo correcto
    else
        %Error 2
        display('Error 2 en CalculaAnguloVect');
    end;

end
```

```
%Calcula c en funcion de valores actuales y anteriores de s, y Theta_s
function c = Calcula_c(s,s_ant,Theta_s,Theta_s_ant)

    if s_ant==0
        %Punto inicial
        c=0;
    else
        %Punto del circuito
        %Calcula Ds
        Ds=s-s_ant;
        %Calcula DTheta_e
        Dtheta_s=Theta_s-Theta_s_ant;
        %Calcula c
        if (Ds~=0) %Para no dividir por cero
            c=Dtheta_s/Ds;
        else c=sign(DTheta)*20;
        end;
    end;

end
```

Se comprueba de un vistazo rápido el gran número de líneas de código que tiene este simulador, con lo cual se confirma lo dicho en el Apartado 5., sobre la complejidad de dicho simulador.

Se han detectado algunos errores que se asumen como perturbación a la simulación.

ANEXO IV: PUNTO DE LINEALIZACIÓN

En este anexo se incluirán los cálculos llevados a cabo para obtener la ecuación temporal con dinámica del punto de linealización ϑ_{e-lin} del Apartado 6.

La ecuación diferencial a resolver era:

$$\dot{\vartheta}_{e-lin} = -\frac{u_{1-lin}}{l_1} \cdot \tan \vartheta_{e-lin} - u_{1-lin} \cdot c_{lin} \cdot \frac{1}{\cos \vartheta_{e-lin}}$$

Y la condición inicial venía dada por:

$$\vartheta_{e-lin-ini} = -\text{asin}(c_{lin-ini} \cdot l_1)$$

i. Resolución analítica de la ecuación diferencial utilizando Matlab

Se tratará aquí de resolver la ecuación diferencial de forma analítica utilizando Matlab.

Con los siguientes comandos de Matlab, se obtiene la solución de la ecuación diferencial, dependiente de una constante:

```
syms l1 u1 c Theta_e real;
Theta_e=dsolve('DTheta_e=-u1/l1*tan(Theta_e)-c*u1/cos(Theta_e)','t');
```

Obteniéndose:

$$\vartheta_{e-lin}(t) = \text{asin}(-\exp\left(-\frac{u_{1-lin}}{l_1} \cdot t - \frac{u_{1-lin}}{l_1} \cdot Cte\right) + c_{lin} \cdot l_1)$$

Como en el instante t=0 se ha de cumplir la condición inicial, se tiene:

$$\vartheta_{e-lin-ini} = -\text{asin}(c_{lin-ini} \cdot l_1) = \vartheta_{e-lin}(0) = \text{asin}(-\exp\left(-\frac{u_{1-lin}}{l_1} \cdot Cte\right) + c_{lin} \cdot l_1)$$

Con lo que resulta:

$$Cte = -\frac{u_{1-lin}}{l_1} \cdot \ln((c_{lin} - c_{lin-ini}) \cdot l_1)$$

Obteniéndose:

$$\vartheta_{e-lin}(t) = \text{asin}\left(e^{-\frac{u_{1-lin}}{l_1} \cdot t} \cdot (c_{lin} - c_{lin-ini}) \cdot l_1 - c_{lin} \cdot l_1\right)$$

ii. Resolución aproximada de la ecuación diferencial

La resolución aproximada consiste en aproximar la solución de la ecuación diferencial a la solución de un sistema lineal de primer orden, siendo la solución:

$$\vartheta_{e-lin}(t) = \vartheta_{e-lin}(t \to \infty) \cdot \left(1 - e^{-\frac{t}{\tau}}\right) - \vartheta_{e-lin-ini} \cdot e^{-\frac{t}{\tau}}$$

Donde se tiene:

- Constante de tiempo: $\tau = \frac{u_1}{l_1}$
- Punto inicial: $\vartheta_{e-lin-ini} = -\text{asin}(c_{lin-ini} \cdot l_1)$
- Punto final: $\vartheta_{e-lin}(t \to \infty) = -\text{asin}(c_{lin} \cdot l_1)$

Con lo que la ecuación resultante es:

$$\vartheta_{e-lin}(t) = -\text{asin}(\text{c}_{\text{lin}} \cdot \text{l}_1) \cdot \left(1 - e^{-\frac{u_{1-lin}}{l_1} \cdot t}\right) - \text{asin}(c_{lin-ini} \cdot l_1) \cdot e^{-\frac{u_{1-lin}}{l_1} \cdot t}$$

En el Apartado 6., se ha comparado la evolución temporal de ambas resoluciones.

ANEXO V: SIMULACIONES DE LA COMPROBACIÓN DE LA LINEALIZACIÓN

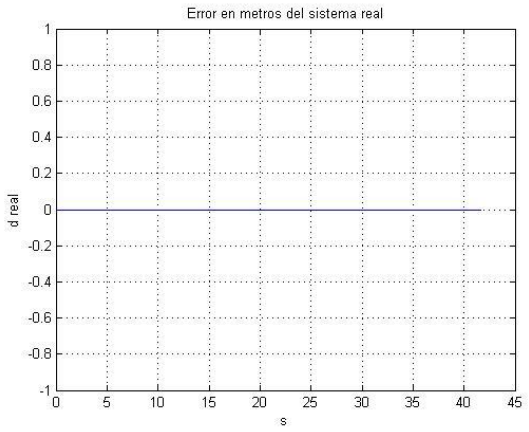
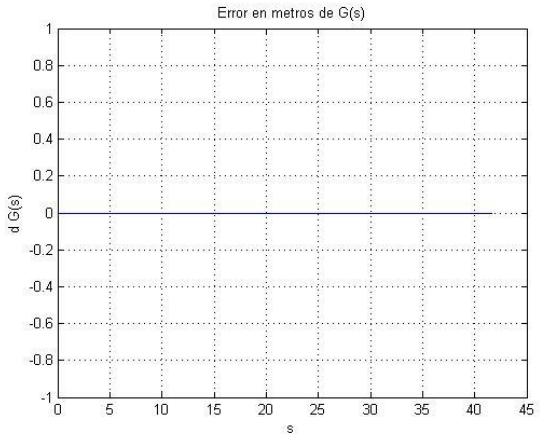
En este anexo se recogen las gráficas de las simulaciones llevadas a cabo para la comprobación de la hipótesis de linealización del Apartado 6.2.5.

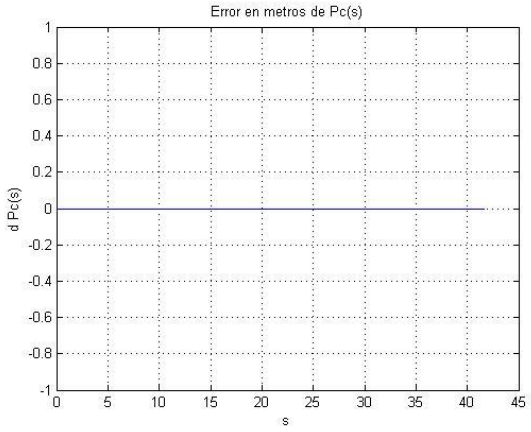
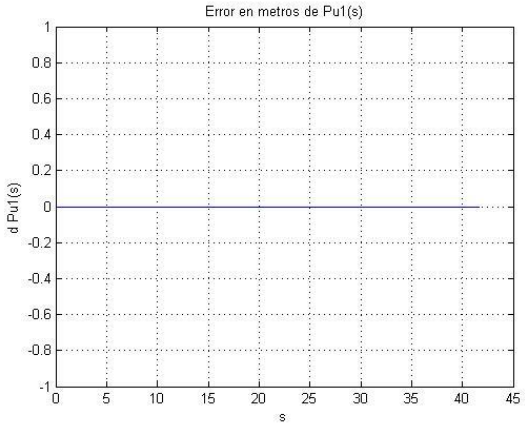
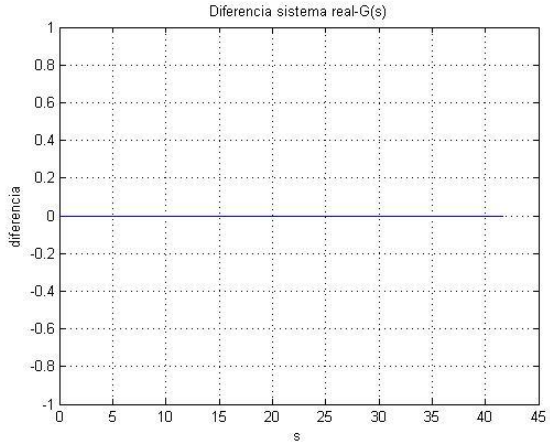
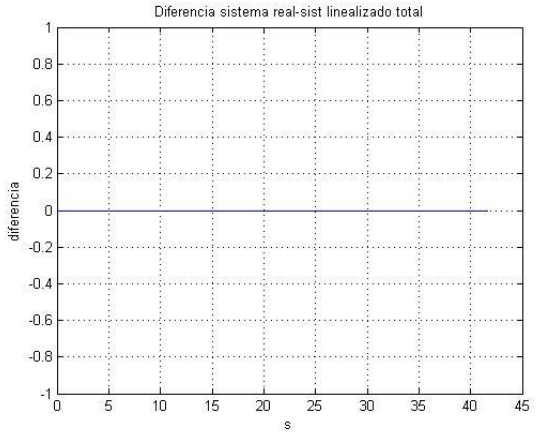
- Simulación 1:

Se eligen los siguientes parámetros para la simulación

$d_{inicial} = 0$	$d_{lin} = 0$
$c = 0$	$c_{lin} = 0$
$u_1 = 4.166$	$u_{1-lin} = 4.166$
$\vartheta_{e-inicial} = 0$	ϑ_{e-lin} por el método exacto
$\Delta\phi = 0$	$\phi_{lin} = \text{atan}\left(-\frac{L}{l_1} \cdot tan\,\vartheta_{e-lin}\right)$

Obteniéndose:

Salida del sistema real	
Salida de $G(s)$	

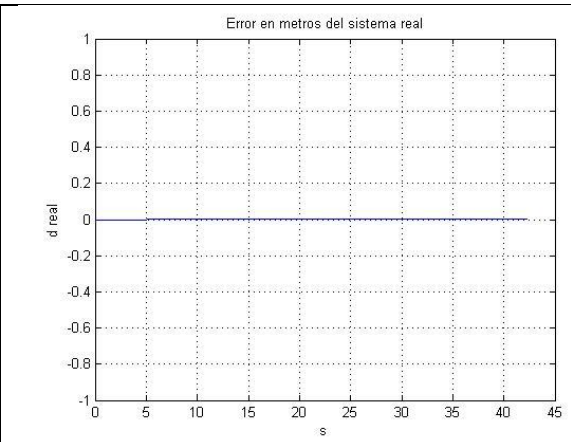
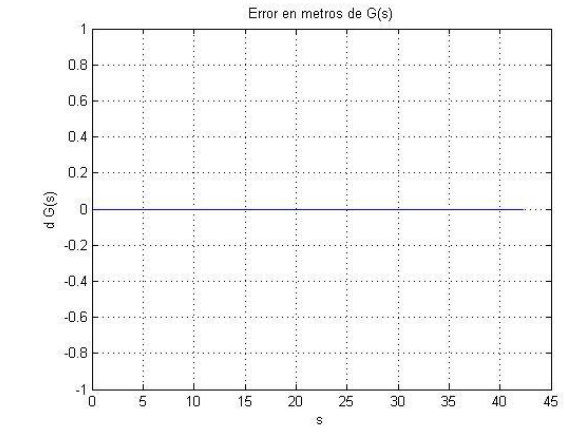
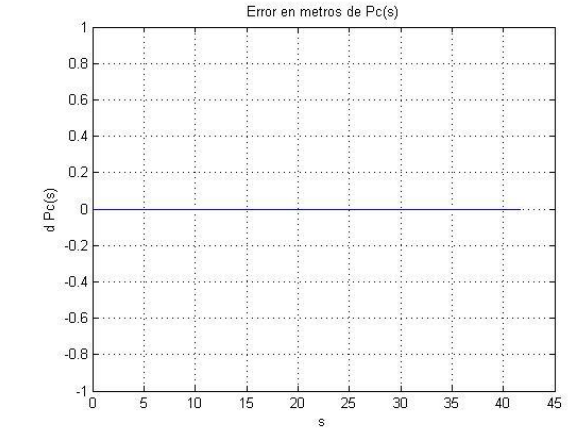
Salida de $P_c(s)$	
Salida de $P_{u_1}(s)$	
Diferencia sistema real, $G(s)$	
Diferencia sistema real, suma de funciones de transferencia	

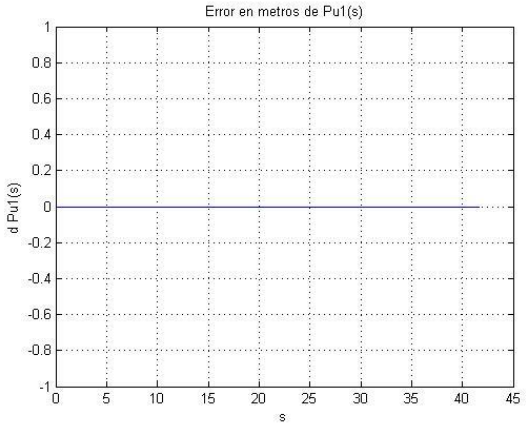
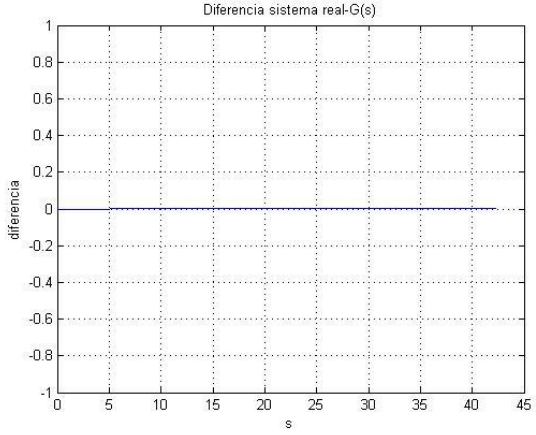
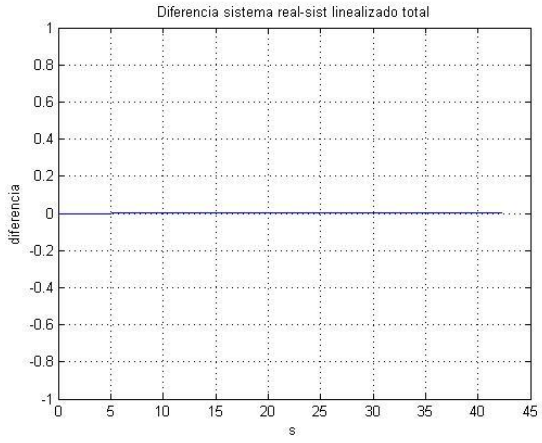
- Simulación 2:

Se eligen los siguientes parámetros para la simulación

$d_{inicial} = 0$	$d_{lin} = 0$
$c = 0.05$	$c_{lin} = 0.05$
$u_1 = 4.166$	$u_{1-lin} = 4.166$
$\vartheta_{e-inicial} = 0$	ϑ_{e-lin} por el método exacto
$\Delta\phi = 0$	$\phi_{lin} = \text{atan}\left(-\frac{L}{l_1} \cdot tan\,\vartheta_{e-lin}\right)$

Obteniéndose:

Salida del sistema real	
Salida de $G(s)$	
Salida de $P_c(s)$	

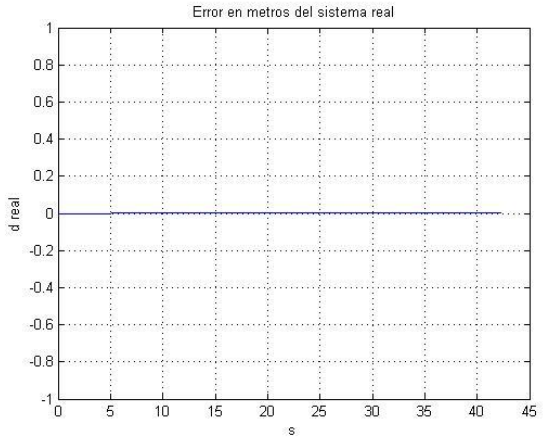
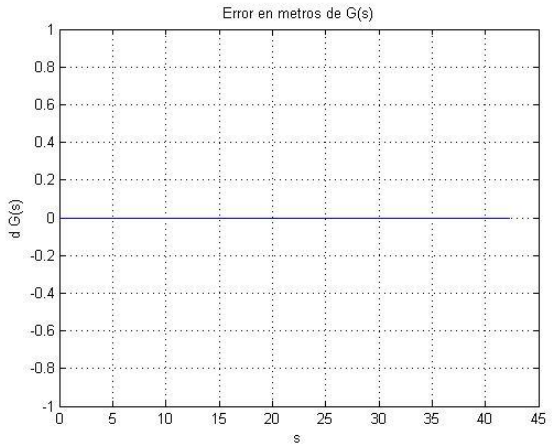
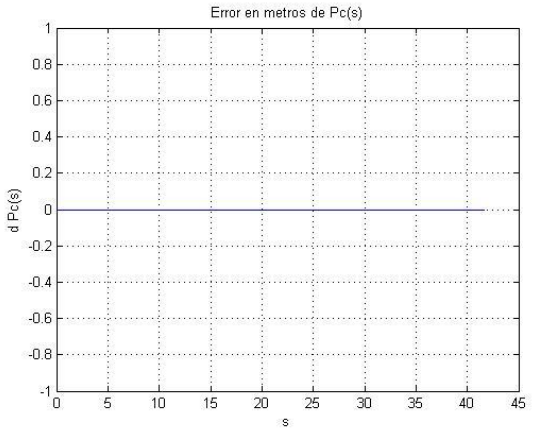
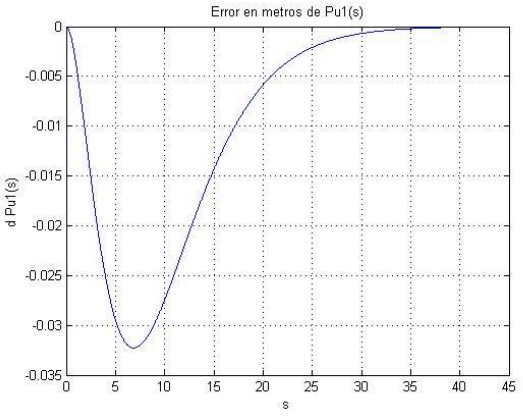
Salida de $P_{u_1}(s)$	
Diferencia sistema real, $G(s)$	
Diferencia sistema real, suma de funciones de transferencia	

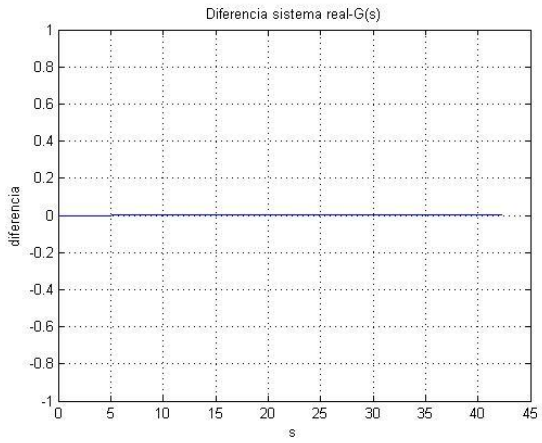
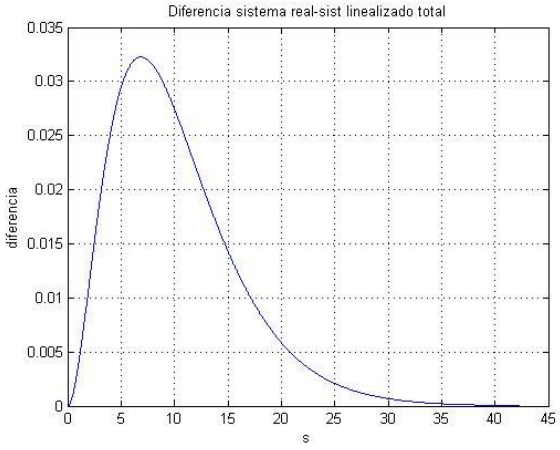
- <u>Simulación 3:</u>

Se eligen los siguientes parámetros para la simulación

$d_{inicial} = 0$	$d_{lin} = 0$
$c = 0.05$	$c_{lin} = 0.05$
$u_1 = 4.166$	$u_{1-lin} = 3$
$\vartheta_{e-inicial} = 0$	ϑ_{e-lin} por el método exacto
$\Delta\phi = 0$	$\phi_{lin} = \text{atan}\left(-\frac{L}{l_1} \cdot tan\,\vartheta_{e-lin}\right)$

Obteniéndose:

Salida del sistema real	
Salida de $G(s)$	
Salida de $P_c(s)$	
Salida de $P_{u_1}(s)$	

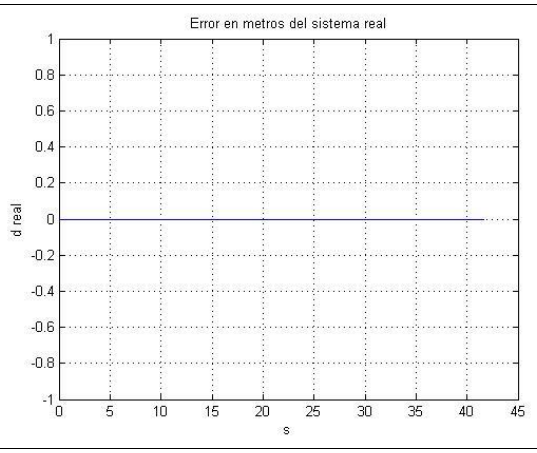
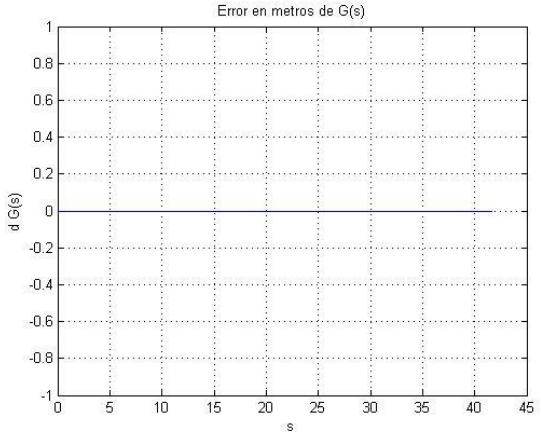
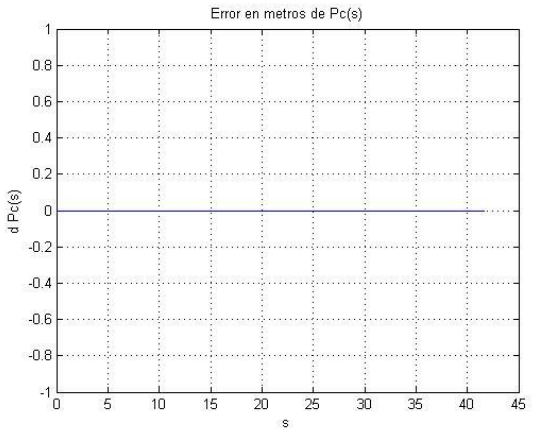
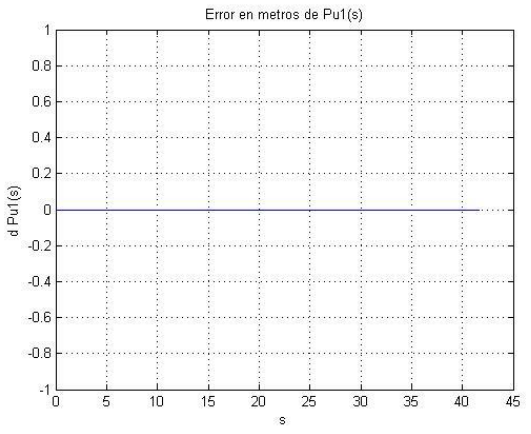
Diferencia sistema real, $G(s)$	
Diferencia sistema real, suma de funciones de transferencia	

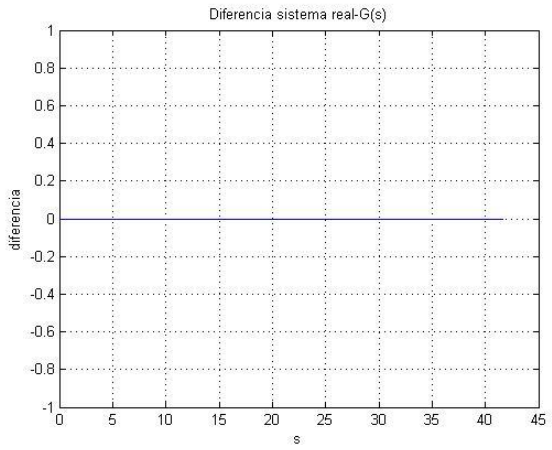
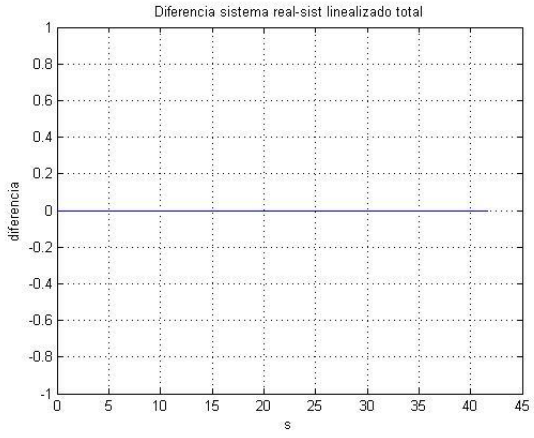
- <u>Simulación 4:</u>

Se eligen los siguientes parámetros para la simulación

$d_{inicial} = 0$	$d_{lin} = 0$
$c = 0$	$c_{lin} = 0$
$u_1 = 4.166$	$u_{1-lin} = 3$
$\vartheta_{e-inicial} = 0$	ϑ_{e-lin} por el método exacto
$\Delta\phi = 0$	$\phi_{lin} = \text{atan}\left(-\frac{L}{l_1} \cdot tan\, \vartheta_{e-lin}\right)$

Obteniéndose:

Salida del sistema real	
Salida de $G(s)$	
Salida de $P_c(s)$	
Salida de $P_{u_1}(s)$	

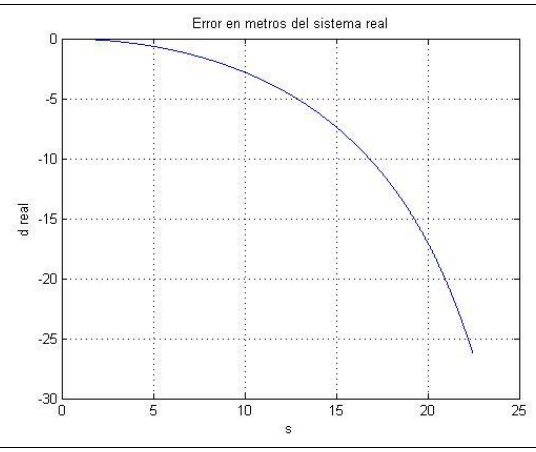
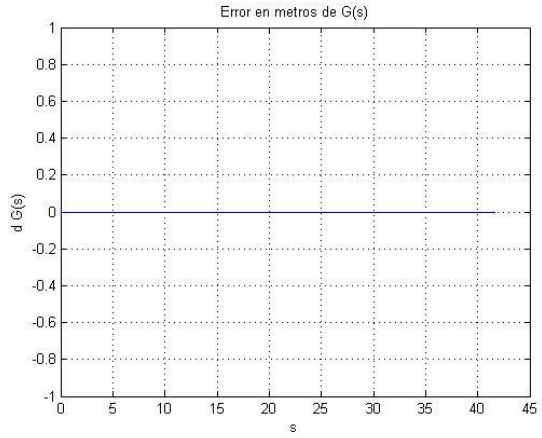
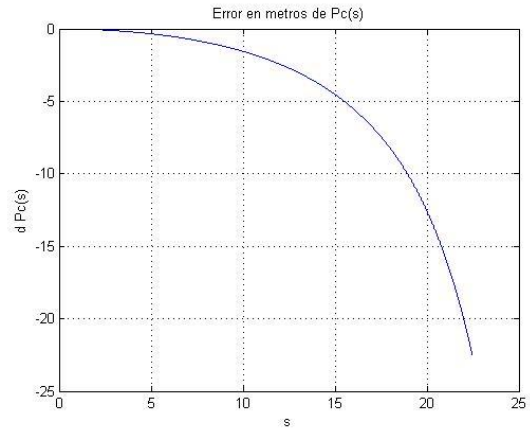
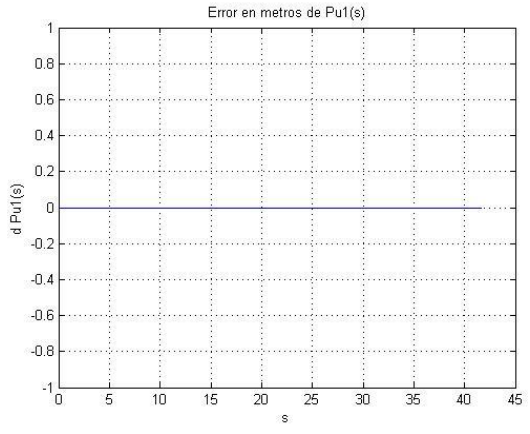
Diferencia sistema real, $G(s)$	
Diferencia sistema real, suma de funciones de transferencia	

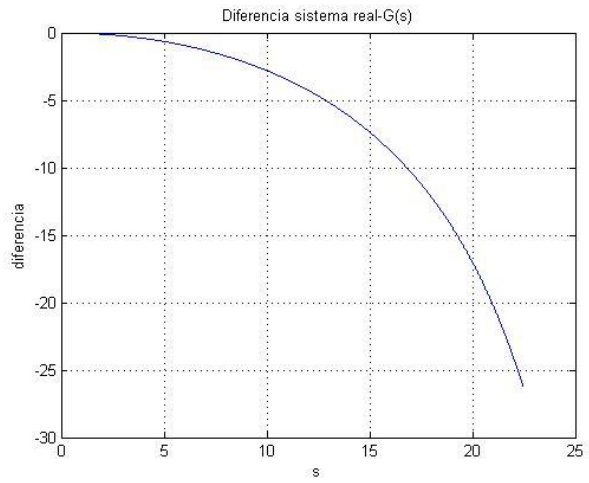
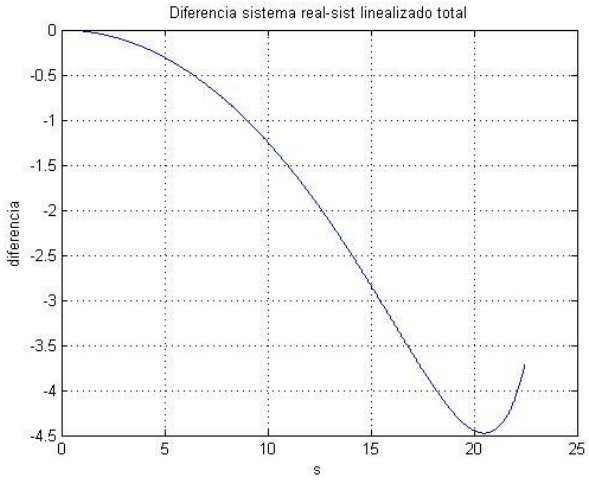
- <u>Simulación 5:</u>

Se eligen los siguientes parámetros para la simulación

$d_{inicial} = 0$	$d_{lin} = 0$
$c = 0.05$	$c_{lin} = 0$
$u_1 = 4.166$	$u_{1-lin} = 3$
$\vartheta_{e-inicial} = 0$	ϑ_{e-lin} por el método exacto
$\Delta\phi = 0$	$\phi_{lin} = \text{atan}\left(-\frac{L}{l_1} \cdot tan\,\vartheta_{e-lin}\right)$

Obteniéndose:

Salida del sistema real	
Salida de $G(s)$	
Salida de $P_c(s)$	
Salida de $P_{u_1}(s)$	

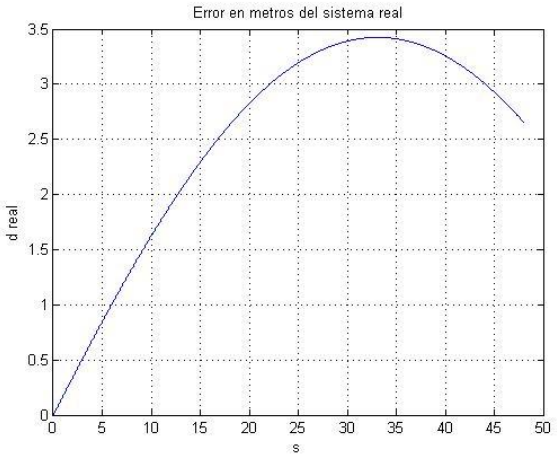
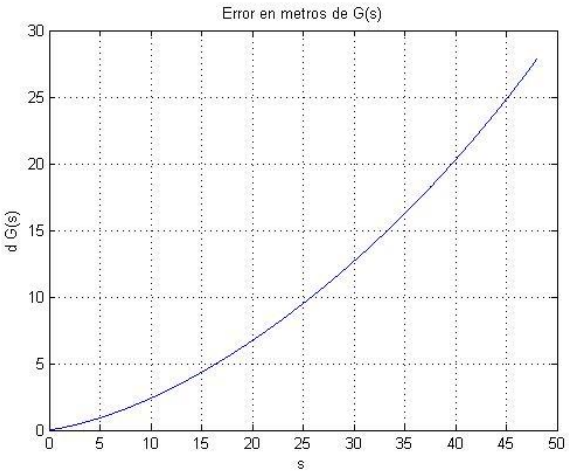
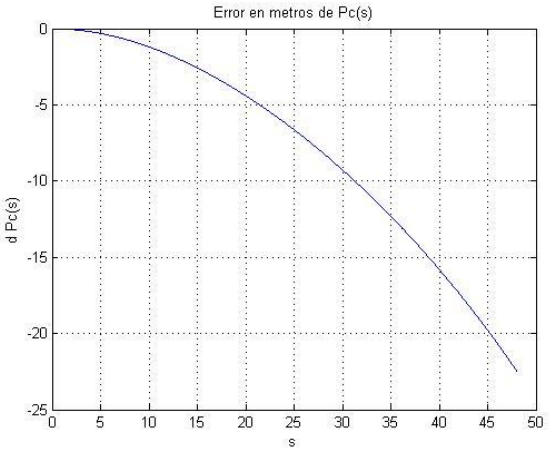
Diferencia sistema real, $G(s)$	
Diferencia sistema real, suma de funciones de transferencia	

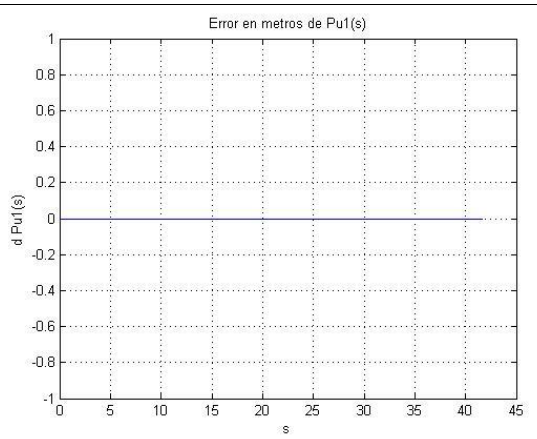
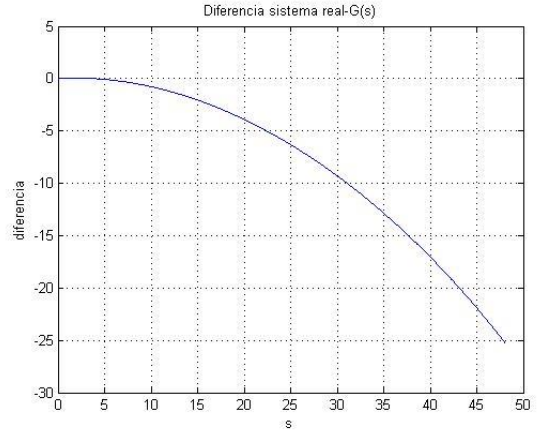
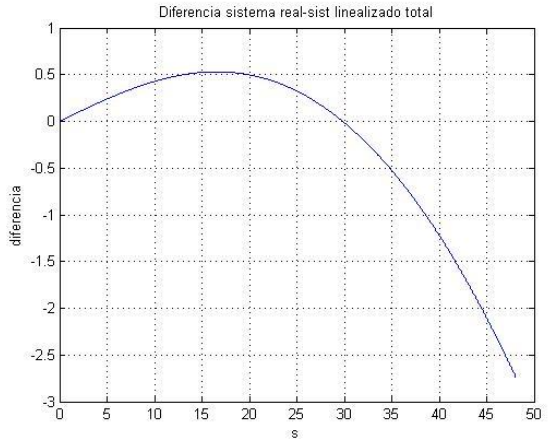
- <u>Simulación 6:</u>

Se eligen los siguientes parámetros para la simulación

$d_{inicial} = 0$	$d_{lin} = 0$
$c = 0.05$	$c_{lin} = 0$
$u_1 = 4.166$	$u_{1-lin} = 3$
$\vartheta_{e-inicial} = 0$	ϑ_{e-lin} por el método exacto
$\Delta\phi = \text{atan}\left(\frac{L}{l_1} \cdot tan(asin(c_{lin} \cdot l_1))\right) = 0.1242$	$\phi_{lin} = \text{atan}\left(-\frac{L}{l_1} \cdot tan\,\vartheta_{e-lin}\right)$

Obteniéndose:

Salida del sistema real	
Salida de $G(s)$	
Salida de $P_c(s)$	

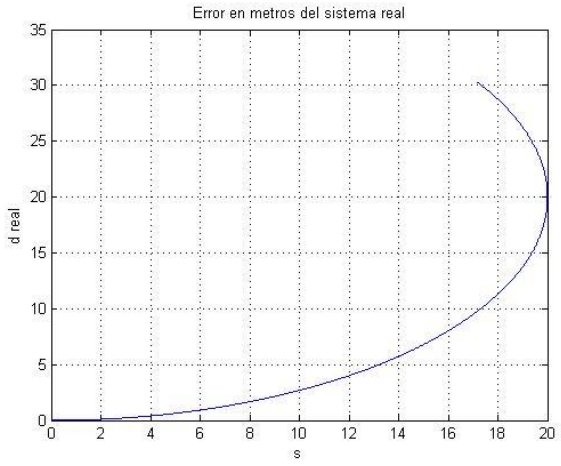
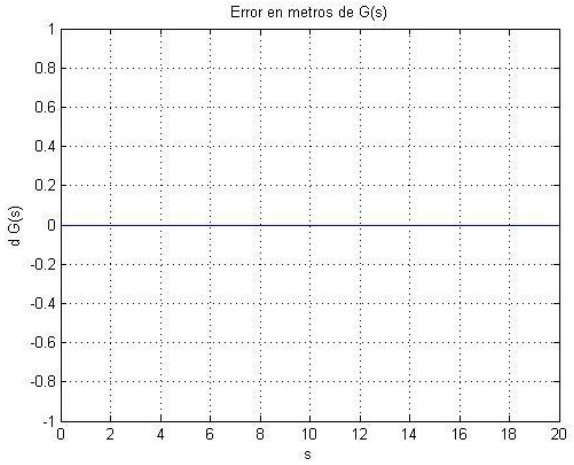
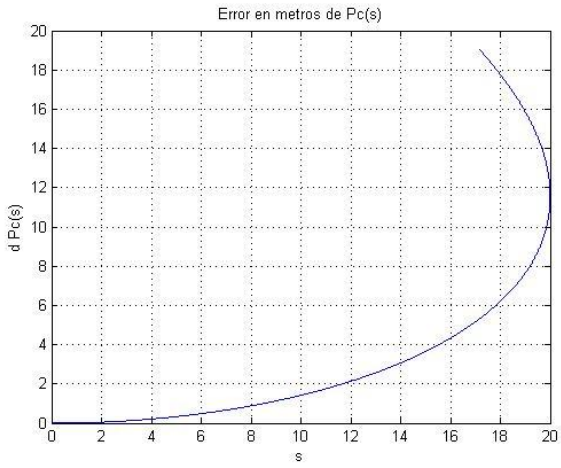
Salida de $P_{u_1}(s)$	
Diferencia sistema real, $G(s)$	
Diferencia sistema real, suma de funciones de transferencia	

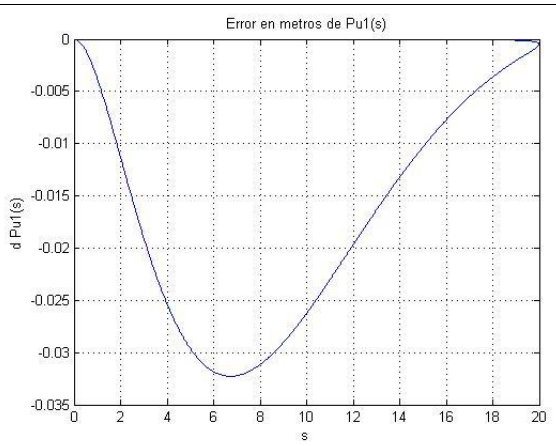
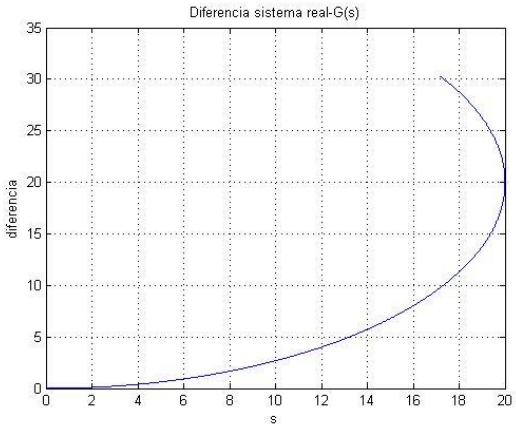
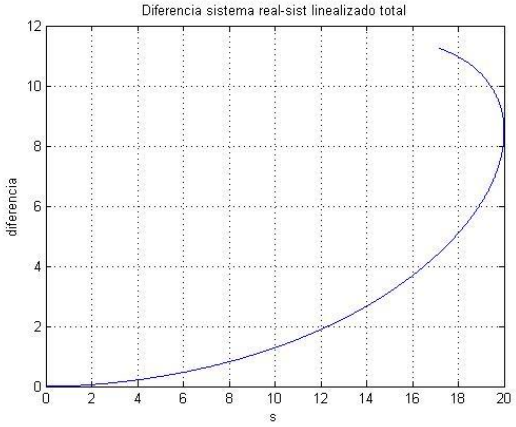
- <u>Simulación 7:</u>

Se eligen los siguientes parámetros para la simulación

$d_{inicial} = 0$	$d_{lin} = 0$
$c = 0$	$c_{lin} = 0.05$
$u_1 = 4.166$	$u_{1-lin} = 3$
$\vartheta_{e-inicial} = 0$	ϑ_{e-lin} por el método exacto
$\Delta\phi = 0$	$\phi_{lin} = \text{atan}\left(-\frac{L}{l_1} \cdot tan\, \vartheta_{e-lin}\right)$

Obteniéndose:

Salida del sistema real	
Salida de $G(s)$	
Salida de $P_c(s)$	

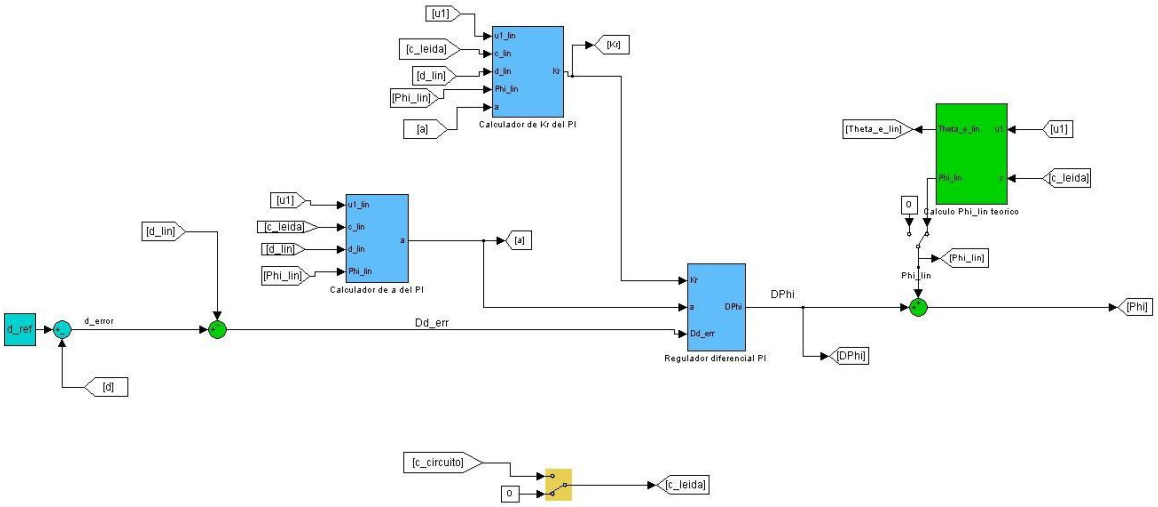
Salida de $P_{u_1}(s)$	
Diferencia sistema real, $G(s)$	
Diferencia sistema real, suma de funciones de transferencia	

ANEXO VI: SIMULADORES DE LOS CONTROLADORES

En este anexo se pondrán los bloques de Simulink y su código .m del los simuladores de los controladores utilizados en los apartados 7.2.3., y 7.3.3.

i. Controlador continuo

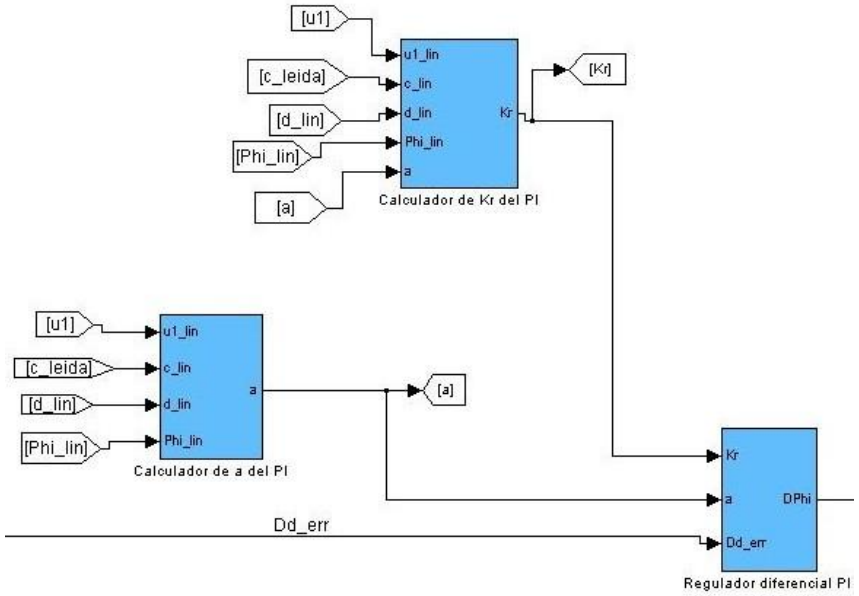
Se utiliza el siguiente esquema de bloques en Simulink:



Teniendo:

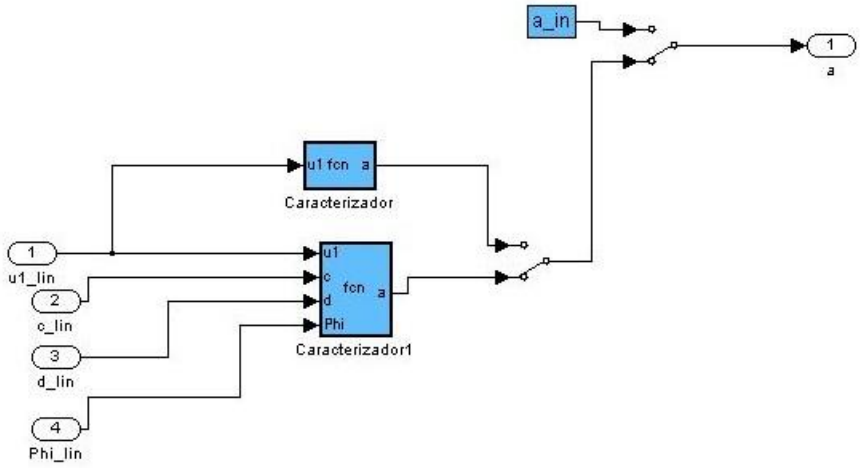
- En amarillo un selector, que permite habilitar o deshabilitar la detección de códigos.
- En verde los puntos de linealización, siendo d_{lin} un valor constante e igual a cero; y ϕ_{lin} se calcula por el método exacto mediante la resolución interna de la ecuación diferencial por unión directa de bloques elementales de Simulink. El punto ϕ_{lin}, incluye un selector para deshabilitarlo en caso necesario.
- En turquesa aparece el calculador del error, que se ocupa de restar el $d_{ref} = 0$ con el d.
- En color azul se representa el regulador incremental, y los bloques de ajuste de sus parámetros. Se analizan a continuación.

El regulador incremental está formado por tres bloques:



Por un lado están los dos bloques que se encargan de calcular los parámetros a y K_R, y por otro lado está el regulador incremental PI.

El calculador de a, internamente se muestra a continuación:



Donde los bloques Caracterizador y Caracterizador1, permiten calcular el valor de a de forma experimental, y utilizando el regulador P (como se vio en el Apartado 7.2.1.2.), mediante Embedded Matlab Function, cuyo código es respectivamente:

```
function a  = fcn(u1)
% This block supports the Embedded MATLAB subset.
% See the help menu for details.

%Valores de diseño
if (u1<1/3.6)
    a=0;
elseif (u1>=(1/3.6) && u1<(5/3.6))
    a=0.05;
elseif (u1>=(5/3.6) && u1<(40/3.6))
    %Parametros minimos
    u1min=5/3.6;
```

```
    amin=0.2;
    %Parametros maximos
    u1max=40/3.6;
    amax=1.5;
    %Recta
    m=(amax-amin)/(u1max-u1min);
    b=amax-m*u1max;
    %Ecuacion
    a=m*u1+b;
else
    a=1.5;
end;

end
```

```
function a  = fcn(u1,c,d,Phi)
% This block supports the Embedded MATLAB subset.
% See the help menu for details.

%Punto de situacion
PUNT=1/3;

%Parametros del coche
L=2.46;
l1=3.41;
l2=0;

%Calculo de puntos de linealizacion
Phi=atan((L*c)/sqrt(1-(l1*c)^2)); %Se deberia poner el valor de Off-set
real!!!!!
Theta_e=atan(-l1*tan(Phi)/L); %Se calcula a partir de Phi

%Coeficientes
b2=u1*cos(Theta_e)-u1*tan(Phi)*l1*sin(Theta_e)/L;
b3=u1*l1*cos(Theta_e)/(L*(cos(Phi))^2);
b5=u1/(L*(cos(Phi))^2)+u1*l1*sin(Theta_e)*c/((1-d*c)*L*(cos(Phi))^2);
b7=-c^2*u1*cos(Theta_e)/(1-d*c)^2+u1*tan(Phi)/L*l1*sin(Theta_e)*c^2/(1-
d*c)^2;
%Coeficientes sist 1
e2=-b2*b7;
e5=b3;
e6=b2*b5;
%Coefs sist realimentado
A1=e5;
A2=e6/e5;
A3=e2;


%Regulador PI
a=PUNT*(A2+(A2^2+A3)^(1/2));

end
```

El calculador de K_R es otra Embedded Matlab Function, cuyo código es:

```
function KR = fcn(u1,c,d,Phi,a)
% This block supports the Embedded MATLAB subset.
```

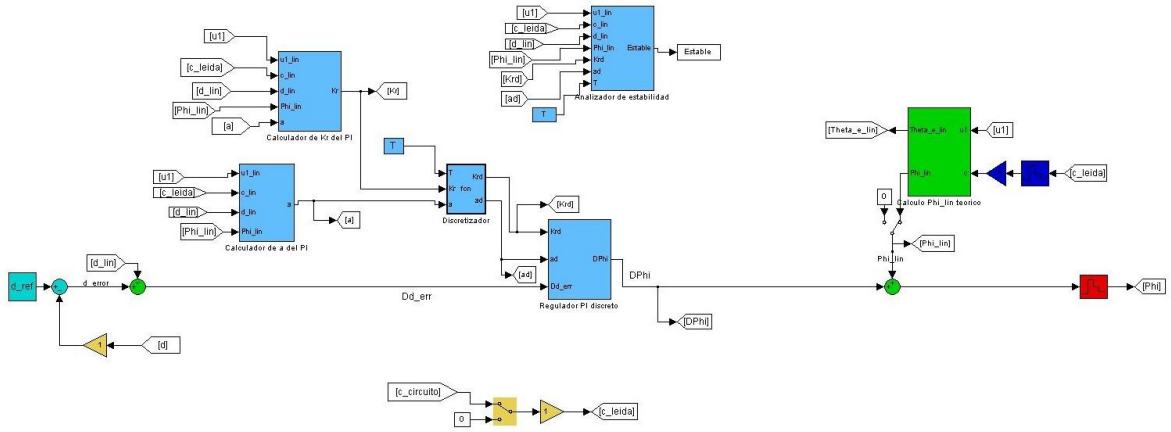
```
% See the help menu for details.


%Parametros del coche
L=2.46;
l1=3.41;
l2=0;

%Calculo de puntos de linealizacion
Phi=atan((L*c)/sqrt(1-(l1*c)^2)); %Se deberia poner el valor de Off-set
real!!!!!
Theta_e=atan(-l1*tan(Phi)/L); %Se calcula a partir de Phi

%Coeficientes
b2=u1*cos(Theta_e)-u1*tan(Phi)*l1*sin(Theta_e)/L;
b3=u1*l1*cos(Theta_e)/(L*(cos(Phi))^2);
b5=u1/(L*(cos(Phi))^2)+u1*l1*sin(Theta_e)*c/((1-d*c)*L*(cos(Phi))^2);
b7=-c^2*u1*cos(Theta_e)/(1-d*c)^2+u1*tan(Phi)/L*l1*sin(Theta_e)*c^2/(1-
d*c)^2;

%Coefs sist realimentado
A1=b3;
A2=b2*b5/b3;
A3=-b2*b7;



%%%%%%%%%%%%Calculo del punto de polos reales dobles

%Tanteo
%No llega. Se toma la posicion del cero
%Posición del cero
z1=-A2;
s_nllega=z1;
nllega=s_nllega^4+(2*a+2*A2)*s_nllega^3+(-A3+3*a*A2)*s_nllega^2+A3*a*A2;
while(nllega>0)
    s_nllega=s_nllega/1.5;
    nllega=s_nllega^4+(2*a+2*A2)*s_nllega^3+(-
A3+3*a*A2)*s_nllega^2+A3*a*A2;
end;


%Se pasa. se tantea
s_spasa=1.5*z1;
spasa=s_spasa^4+(2*a+2*A2)*s_spasa^3+(-A3+3*a*A2)*s_spasa^2+A3*a*A2;
while(spasa<0)
    s_spasa=1.5*s_spasa;
    spasa=s_spasa^4+(2*a+2*A2)*s_spasa^3+(-A3+3*a*A2)*s_spasa^2+A3*a*A2;
end;


%Se calcula la posicion exacta por iteracion
fin=0;
tol=1e-4;
pos_polos=0;
%i=0;

while(fin==0)
```

```
    s_nueva=(s_spasa+s_nllega)/2;
    que_pasa=s_nueva^4+(2*a+2*A2)*s_nueva^3+(-
A3+3*a*A2)*s_nueva^2+A3*a*A2;
    %Se comprueba que ha sucedido
    if (que_pasa==0 || abs(que_pasa)<tol)
        fin=1;
        pos_polos=s_nueva;
    elseif(que_pasa>0)
        s_spasa=s_nueva;
    elseif (que_pasa<0)
        s_nllega=s_nueva;
    end;
    %i=i+1;

end;

%Numero de iteraciones
%i

%%%%%%%%%%Calculo KR utilizando el criterio del modulo
d1=-pos_polos;
d2=sqrt(d1^2+A3);


KR=d1*d2^2/((d1-a)*(d1-A2)*abs(A1));


End
```

El regulador PI incremental está formado por la unión de bloques elementales, aunque también se podría haber hecho una transformación de la función de transferencia del PI a su modelo de estado y haberlo programado con una S-Function.

ii. Controlador discreto

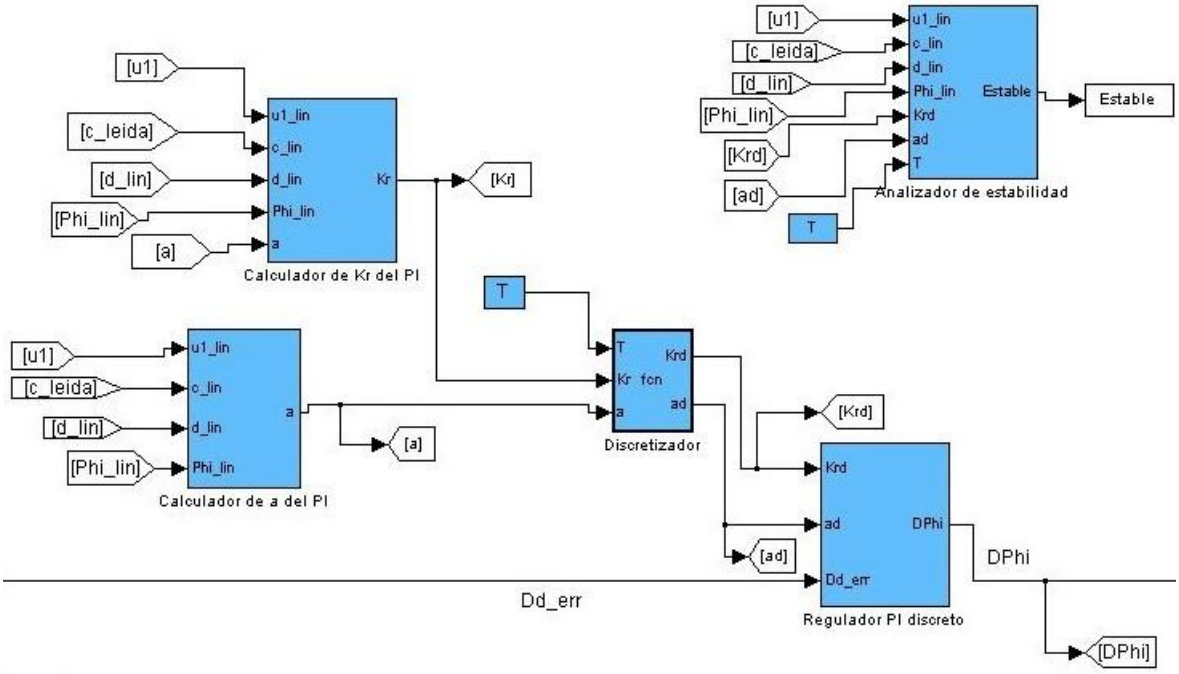
El controlador discreto incluye el siguiente esquema en Simulink:



Que contiene los siguientes bloques:

- En amarillo un selector, que permite habilitar o deshabilitar la detección de códigos, así como muestreadores a la frecuencia del sistema de visión.
- En verde los puntos de linealización, siendo d_{lin} un valor constante e igual a cero; y ϕ_{lin} se calcula por el método exacto mediante la resolución interna de la ecuación diferencial por unión directa de bloques elementales de Simulink. El punto ϕ_{lin}, incluye un selector para deshabilitarlo en caso necesario.
- En turquesa aparece el calculador del error, que se ocupa de restar el $d_{ref} = 0$ con el d.
- En color azul claro se representa el regulador discreto PI incremental, los bloques de ajuste de sus parámetros, el bloque discretizador y el bloque que analiza la estabilidad. Se analizan a continuación.
- En rojo se representa un bloqueador de orden cero, para su entrada al sistema (que es continuo)
- En azul oscuro se representa un bloqueador y un muestreador a la frecuencia del computador.

El regulador PI incremental está formado por:



- Calculador de parámetros del PI continuo (Calculador de K_R y Calculador da a), que son similares a los explicados anteriormente.
- Discretizador, que es una Embedded Matlab Function, que se encarga de discretizar los parámetros del regulador continuo, y cuyo código es:

```
function [Krd,ad] = fcn(T,Kr,a)
% This block supports the Embedded MATLAB subset.
% See the help menu for details.

%Krd
Krd=Kr*(1+a*T);

%ad
ad=1/(1+a*T);
```

```
end
```

- Analizador de estabilidad, que es otra Embedded Matlab Function, encargada de comprobar la estabilidad del sistema, cuyo código es:

```
function estable  = fcn(u1,c,d,Phi,Krd,ad,T)
% This block supports the Embedded MATLAB subset.
% See the help menu for details.



%Parametros del coche
L=2.46;
l1=3.41;
l2=0;

%Calculo de puntos de linealizacion
Phi=atan((L*c)/sqrt(1-(l1*c)^2)); %Se deberia poner el valor de Off-set
real!!!!!
Theta_e=atan(-l1*tan(Phi)/L); %Se calcula a partir de Phi

%Coeficientes
b2=u1*cos(Theta_e)-u1*tan(Phi)*l1*sin(Theta_e)/L;
b3=u1*l1*cos(Theta_e)/(L*(cos(Phi))^2);
b5=u1/(L*(cos(Phi))^2)+u1*l1*sin(Theta_e)*c/((1-d*c)*L*(cos(Phi))^2);
b7=-c^2*u1*cos(Theta_e)/(1-d*c)^2+u1*tan(Phi)/L*l1*sin(Theta_e)*c^2/(1-
d*c)^2;

%Coefs sist realimentado
A1=b3;
A2=b2*b5/b3;
A3=-b2*b7;


%Sistema muestreado
B1=A1*T;
B2=A2*T-1;
B3=-2;
B4=A3*T^2+1;


%Sistema realimentado
C1=B3-1+Krd*B1;
C2=B4-B3+Krd*B1*(B2-ad);
C3=-B4-Krd*B1*B2*ad;


%Criterio de Jury
estable=0;

if ( (1+C1+C2+C3>0) && (-1+C1-C2+C3<0) && (abs(C3)<1 && abs(C3^2-
1)>abs(C1*C3-C2)) )
    estable=1;
end;
```

```
end
```

- Regulador PI discreto que contiene el regulador PI discreto mediante la unión de bloques elementales de Simulink.

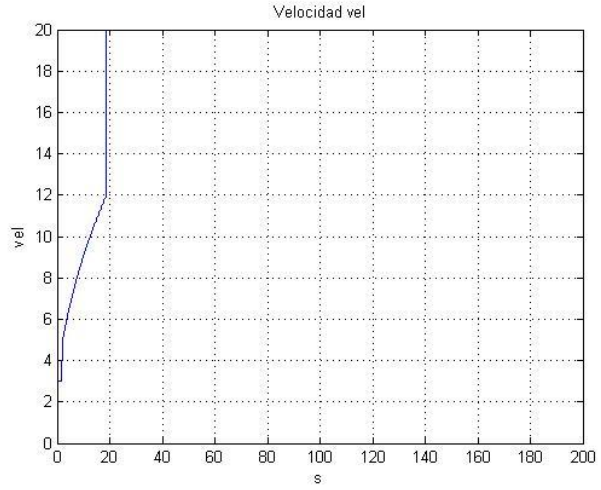
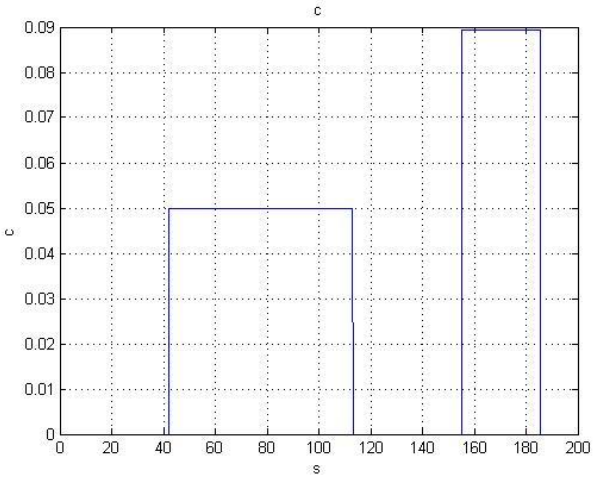
ANEXO VII: PRUEBAS DEL CONTROLADOR EN EL SIMULADOR

i. Simulación en tiempo continuo

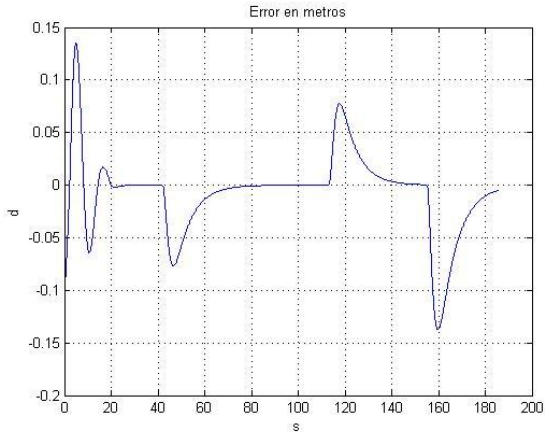
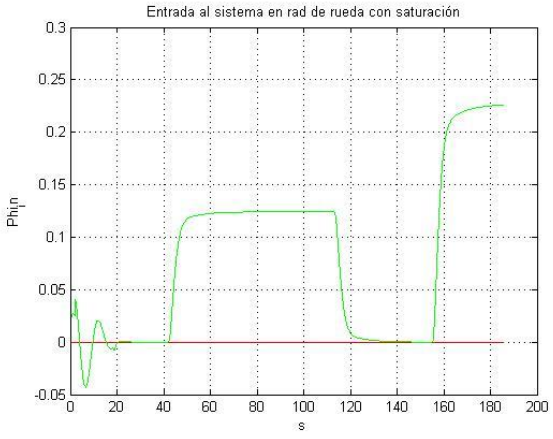
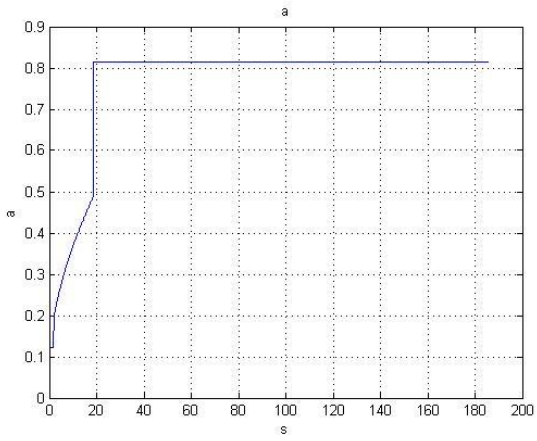
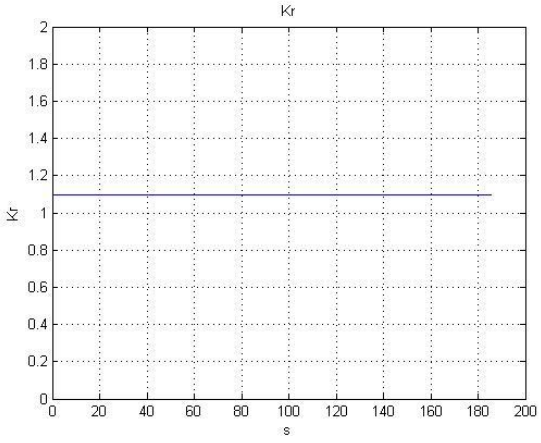
Se han llevado a cabo 7 simulaciones con diversos parámetros:

- Simulación 1

Se tienen los siguientes parámetros:

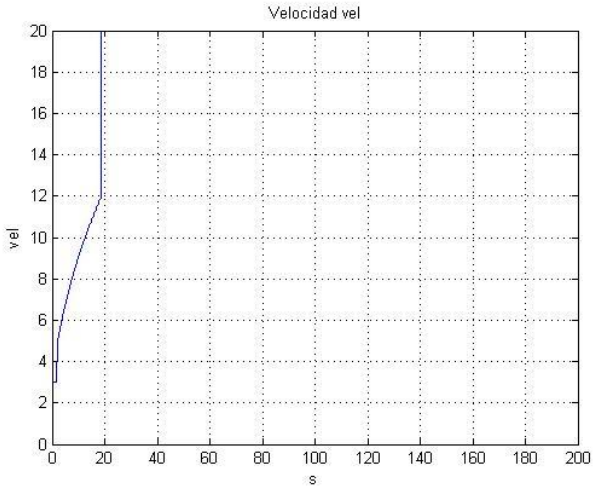
Método de cálculo de a	Utilizando regulador P
Errores en d	No
Errores de actuación	No
Punto inicial	$d_{inicial} = 0.1$ $\vartheta_{e-inicial} = 0$ $s_{inicial} = 0$
Consigna de velocidad (en Km/h)	
Lectura de códigos	No
Cálculo de ϕ_{lin}	Método exacto
Circuito	INSIA 

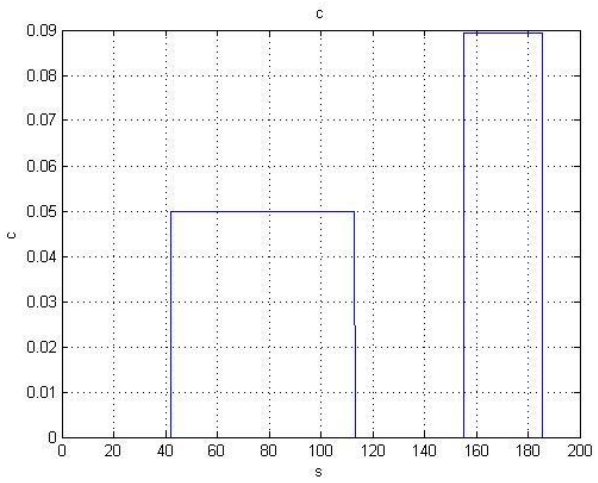
Se obtienen las siguientes gráficas de salida:

   	
Error medio cuadrático del error:	0.0470

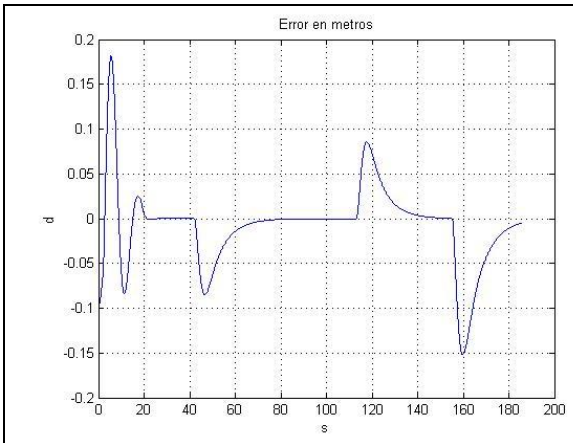
- Simulación 2

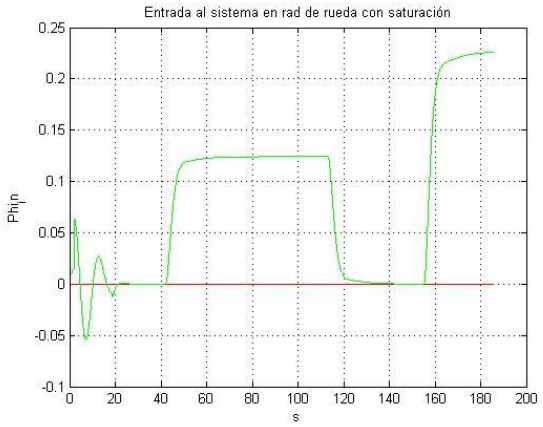
Se tienen los siguientes parámetros:

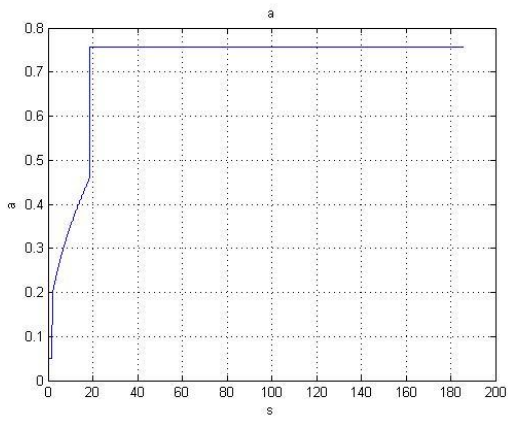
Método de cálculo de a	Utilizando método experimental
Errores en d	No
Errores de actuación	No
Punto inicial	$d_{inicial} = 0.1$ $\vartheta_{e-inicial} = 0$ $s_{inicial} = 0$
Consigna de velocidad (en Km/h)	

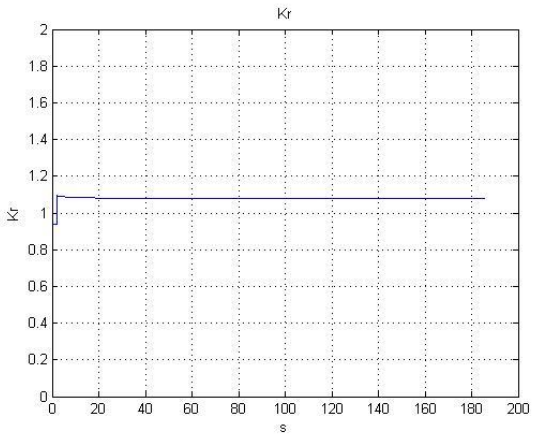
Lectura de códigos	No
Cálculo de ϕ_{lin}	Método exacto
Circuito	INSIA 

Se obtienen las siguientes gráficas de salida:





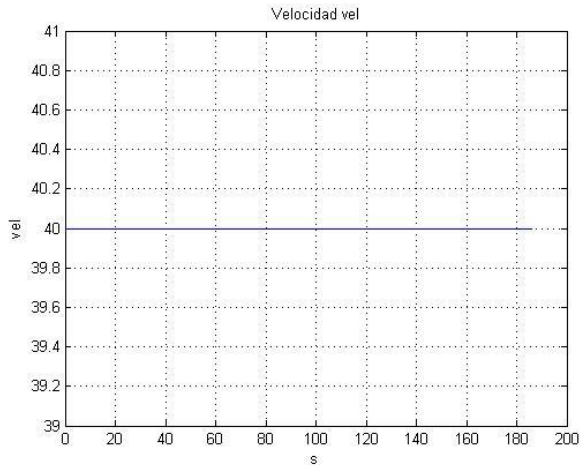
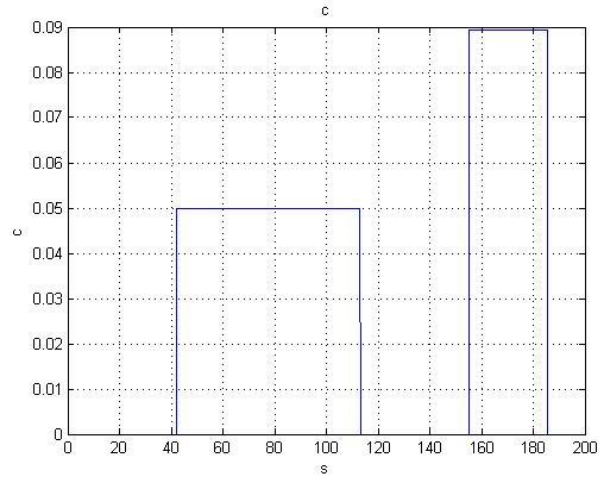




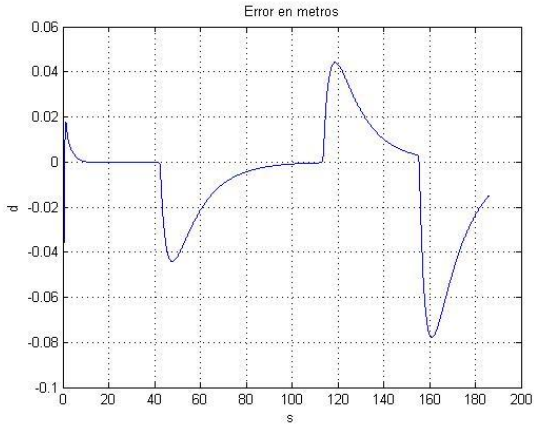
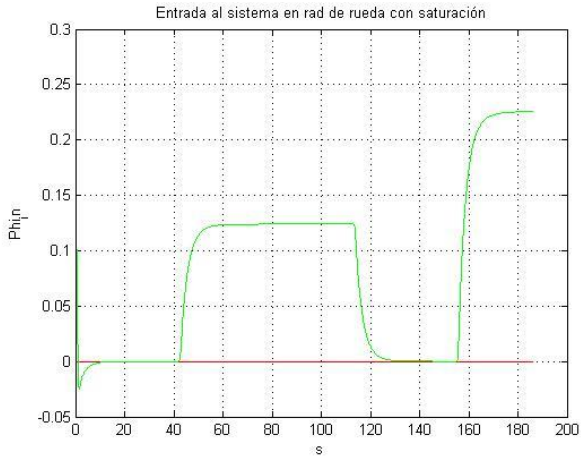
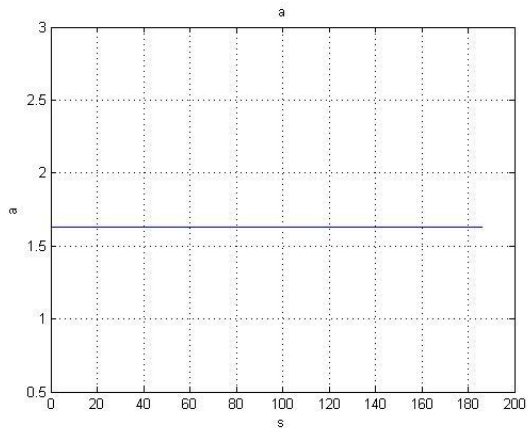
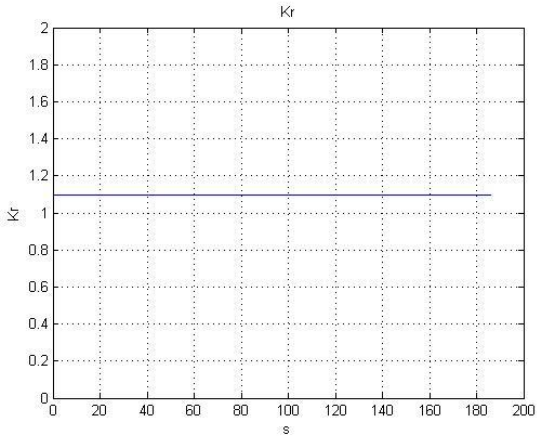
Error medio cuadrático del error:	0.0568

- <u>Simulación 3</u>

Se tienen los siguientes parámetros:

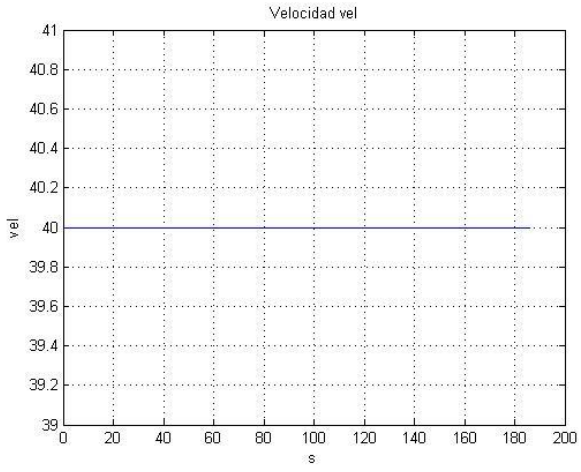
Método de cálculo de a	Utilizando regulador P
Errores en d	No
Errores de actuación	No
Punto inicial	$d_{inicial} = 0.1$ $\vartheta_{e-inicial} = 0$ $s_{inicial} = 0$
Consigna de velocidad (en Km/h)	
Lectura de códigos	No
Cálculo de ϕ_{lin}	Método exacto
Circuito	INSIA 

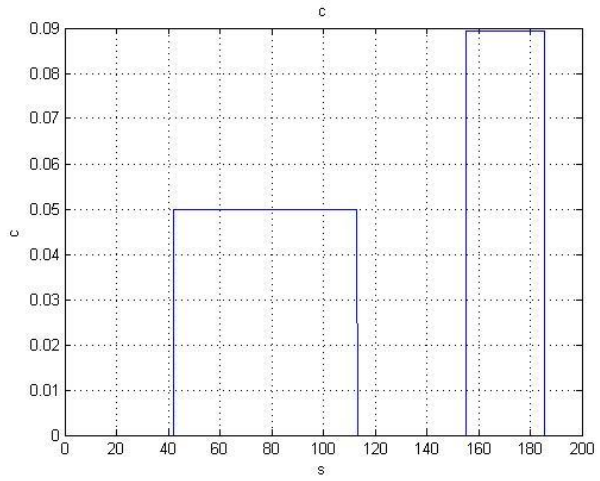
Se obtienen las siguientes gráficas de salida:

   	
Error medio cuadrático del error:	0.0259

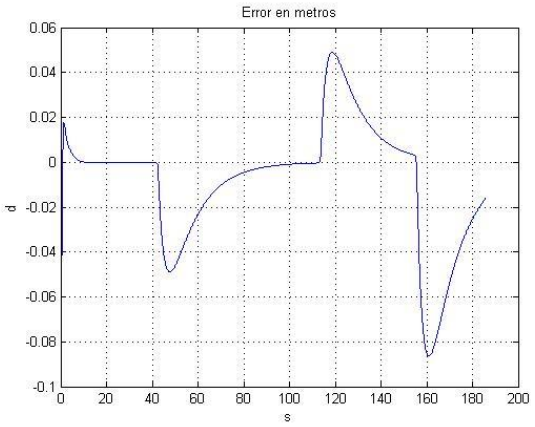
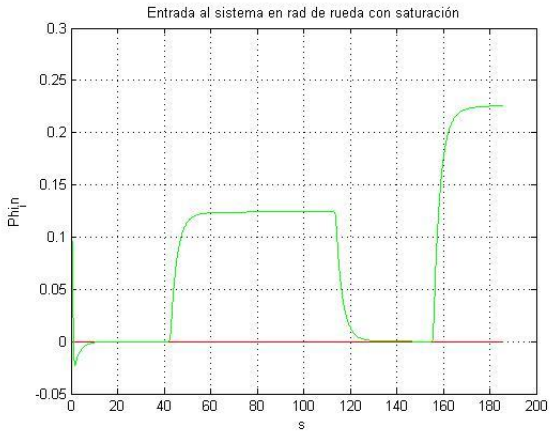
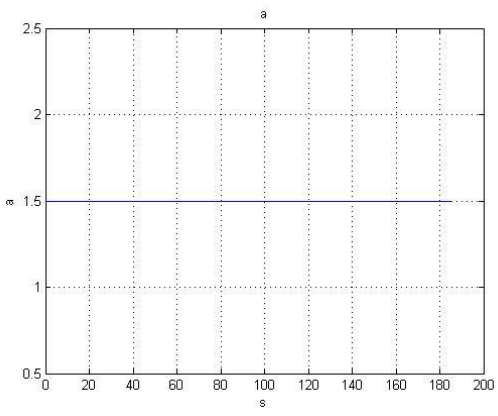
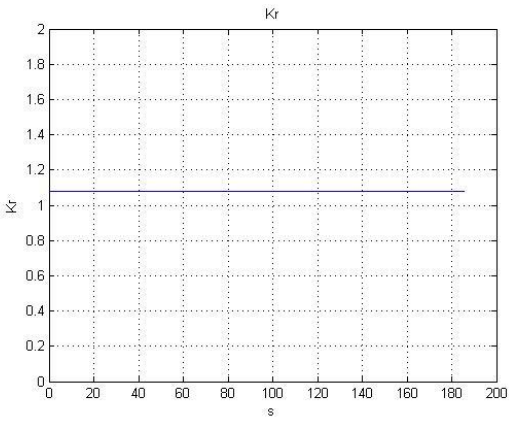
- <u>Simulación 4</u>

Se tienen los siguientes parámetros:

Método de cálculo de a	Utilizando método experimental
Errores en d	No
Errores de actuación	No
Punto inicial	$d_{inicial} = 0.1$ $\vartheta_{e-inicial} = 0$ $s_{inicial} = 0$
Consigna de velocidad (en Km/h)	

Lectura de códigos	No
Cálculo de ϕ_{lin}	Método exacto
Circuito	INSIA 

Se obtienen las siguientes gráficas de salida:

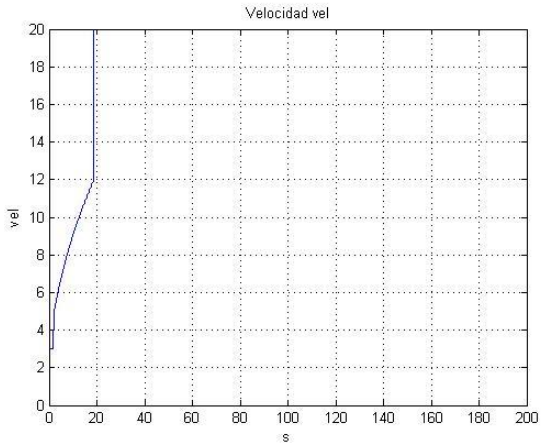
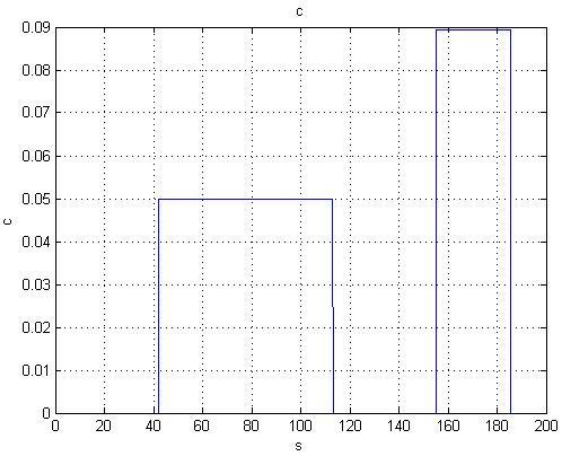
	
	
Error medio cuadrático del error:	0.0284

De las cuatro simulaciones anteriores, se extrae que más o menos, los dos métodos de cálculo de a, aportan los mismos resultados, aunque parece que el método que usa el regulador P da mejores

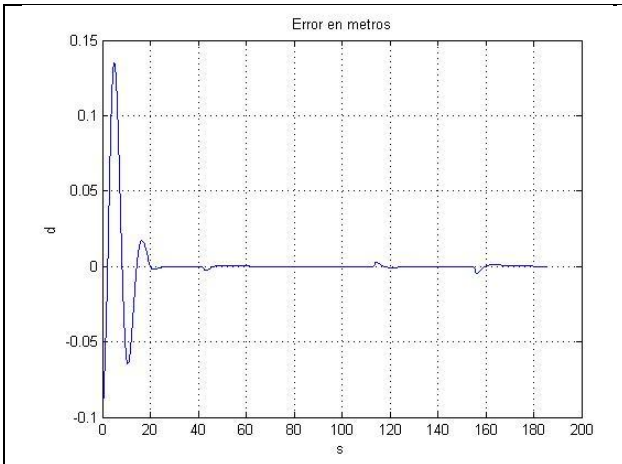
resultados (menor error medio cuadrático del error), por lo que de ahora en adelante será el que se use.

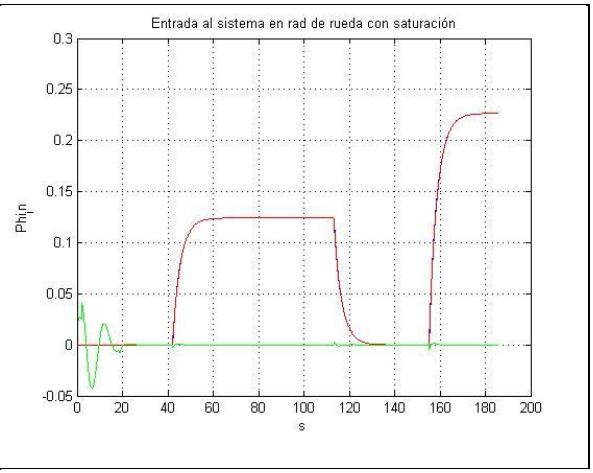
- Simulación 5

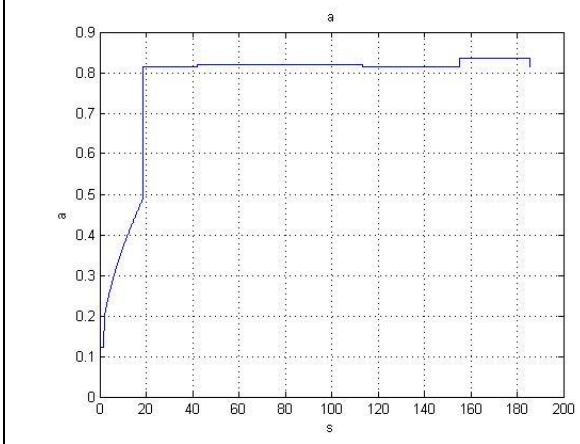
Se tienen los siguientes parámetros:

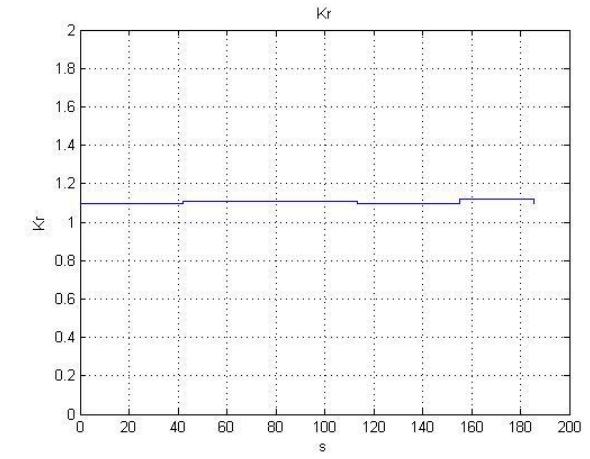
Método de cálculo de a	Utilizando regulador P
Errores en d	No
Errores de actuación	No
Punto inicial	$d_{inicial} = 0.1$ $\vartheta_{e-inicial} = 0$ $s_{inicial} = 0$
Consigna de velocidad (en Km/h)	
Lectura de códigos	Si
Cálculo de ϕ_{lin}	Método exacto
Circuito	INSIA 

Se obtienen las siguientes gráficas de salida:









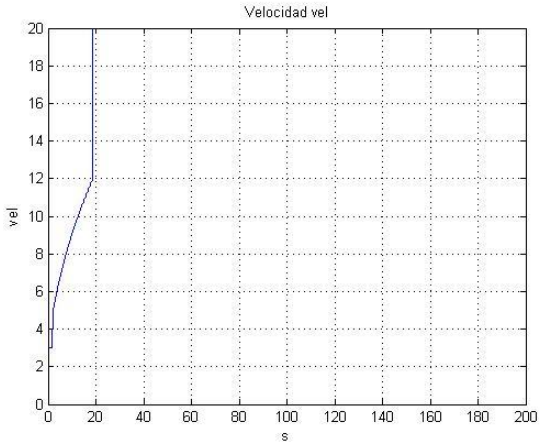
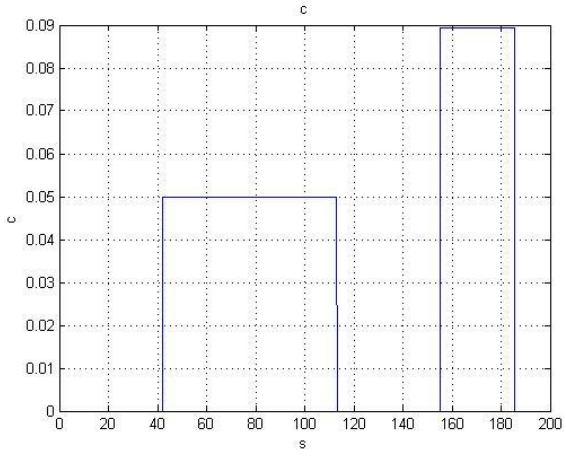
Error medio cuadrático del error:	0.0351

Se extrae de ésta simulación, que el conocimiento de los códigos que indican el circuito es mejor que su desconocimiento (menor error medio cuadrático del error).

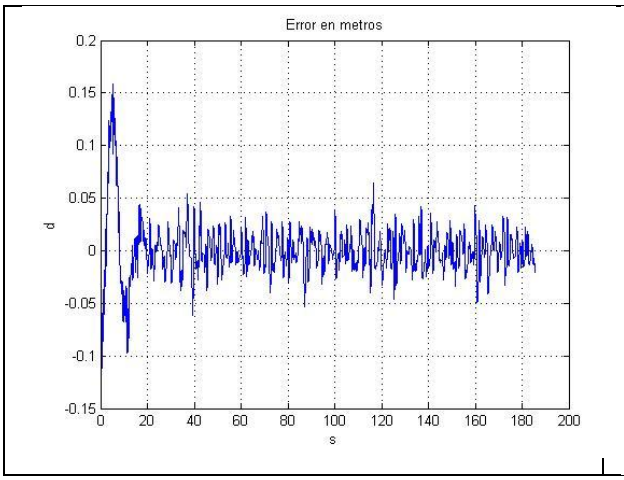
- <u>Simulación 6</u>

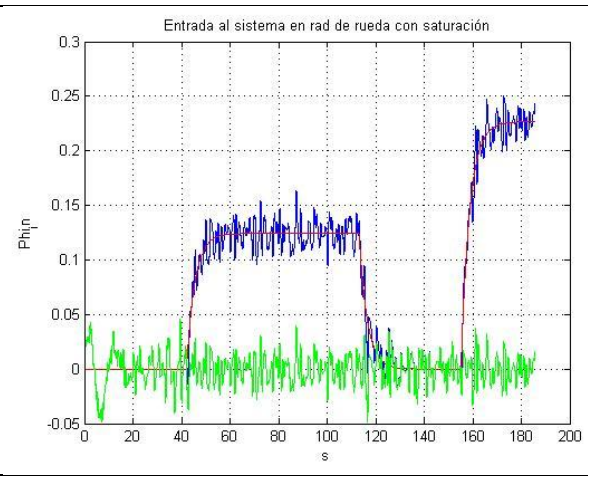
Se tienen los siguientes parámetros:

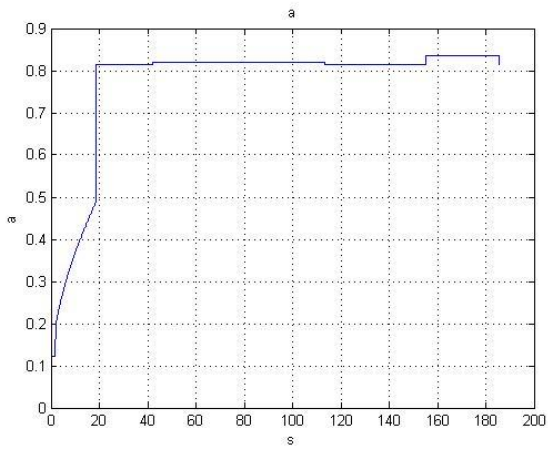
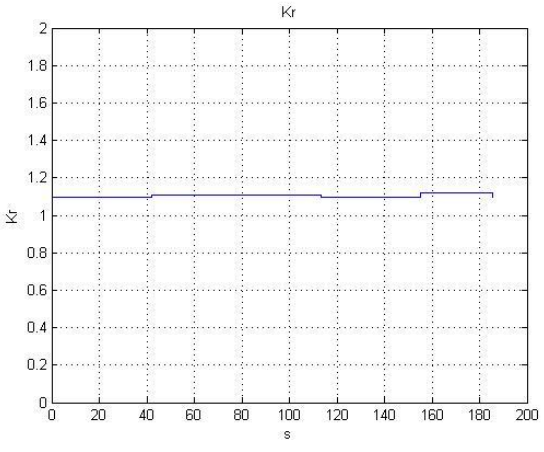
Método de cálculo de a	Utilizando regulador P
Errores en d	Si, error de precisión a 10 Hz
Errores de actuación	No
Punto inicial	$d_{inicial} = 0.1$ $\vartheta_{e-inicial} = 0$ $s_{inicial} = 0$

Consigna de velocidad (en Km/h)	
Lectura de códigos	Si
Cálculo de ϕ_{lin}	Método exacto
Circuito	INSIA 

Se obtienen las siguientes gráficas de salida:



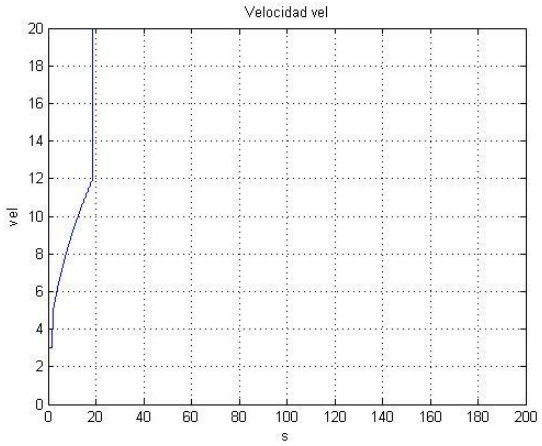


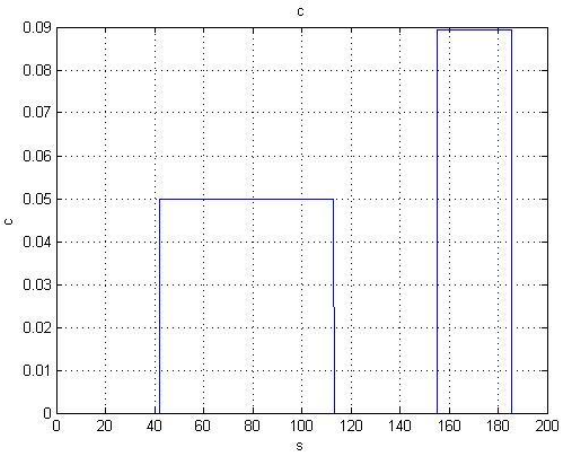
 	
Error medio cuadrático del error:	0.0383

Se concluye que si se conocen los códigos, aunque haya error no modelado en d, el controlador es capaz de seguir la línea sin dispararse su error medio cuadrático.

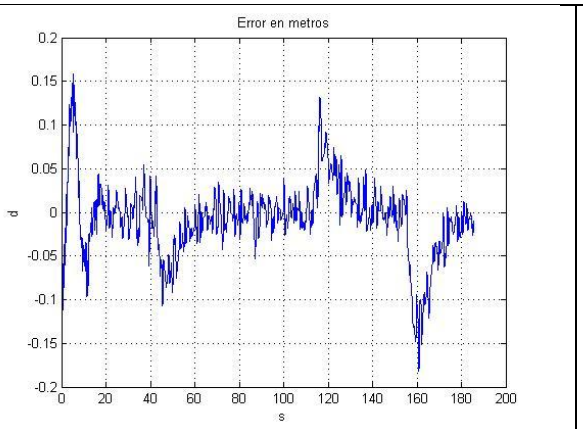
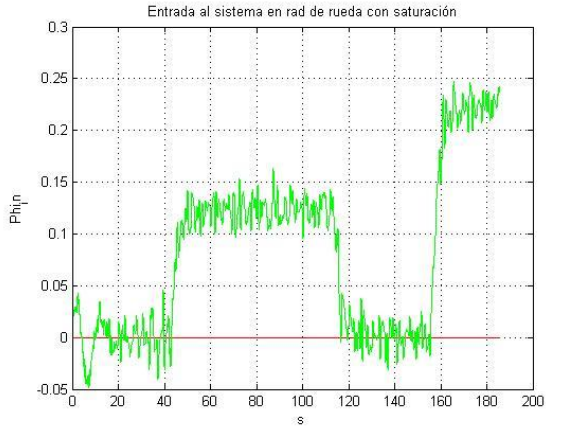
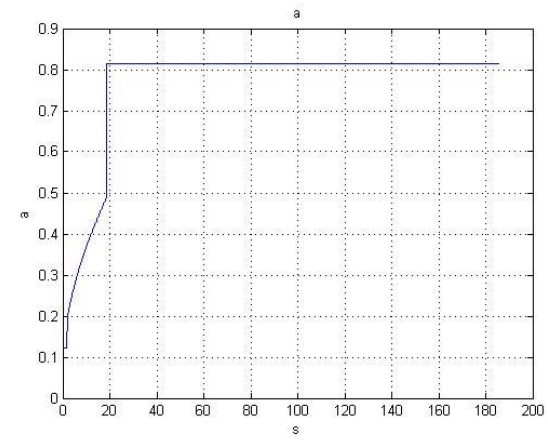
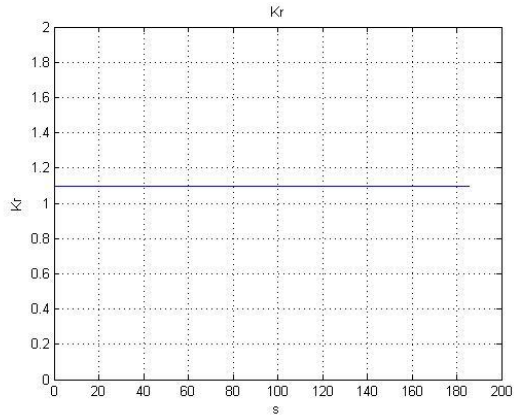
- Simulación 7

Se tienen los siguientes parámetros:

Método de cálculo de a	Utilizando regulador P
Errores en d	Si, error de precisión a 10 Hz
Errores de actuación	No
Punto inicial	$d_{inicial} = 0.1$ $\vartheta_{e-inicial} = 0$ $s_{inicial} = 0$
Consigna de velocidad (en Km/h)	
Lectura de códigos	No
Cálculo de ϕ_{lin}	Método exacto
Circuito	INSIA

Se obtienen las siguientes gráficas de salida:

	
	
Error medio cuadrático del error:	0.0495

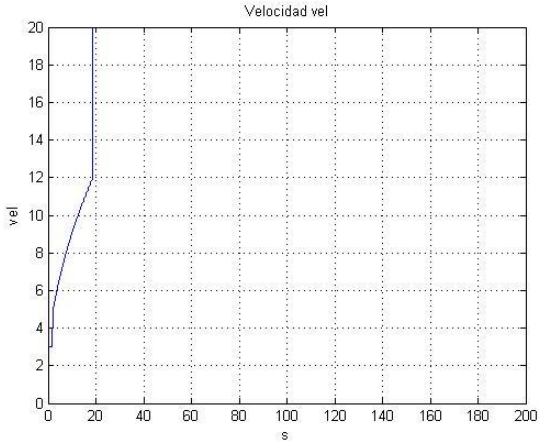
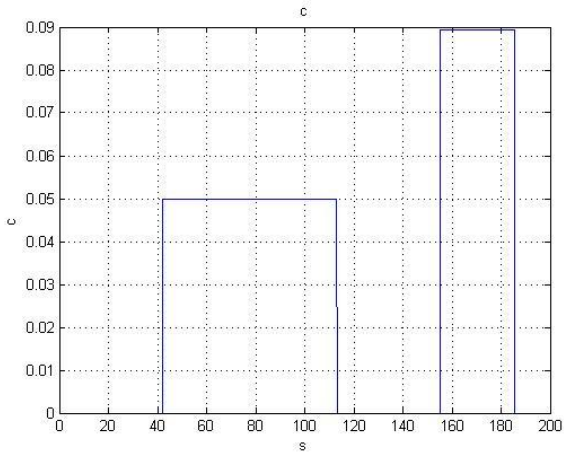
Se concluye que aunque no se conozcan los códigos, y aunque haya error no modelado en d, el controlador es capaz de seguir la línea sin dispararse su error medio cuadrático.

ii. Simulación en tiempo discreto

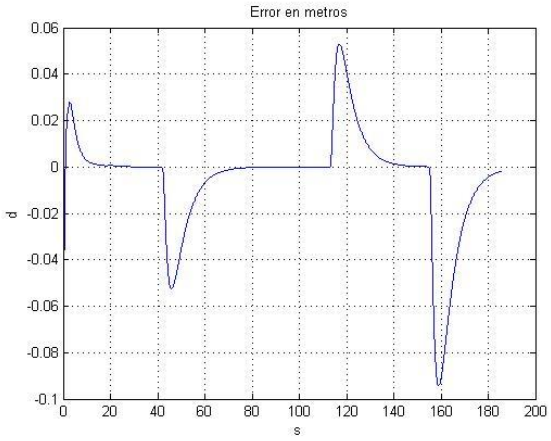
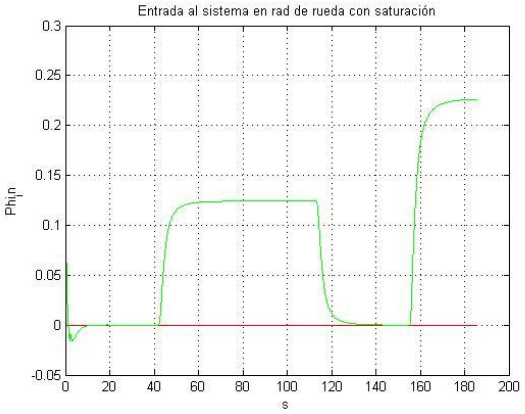
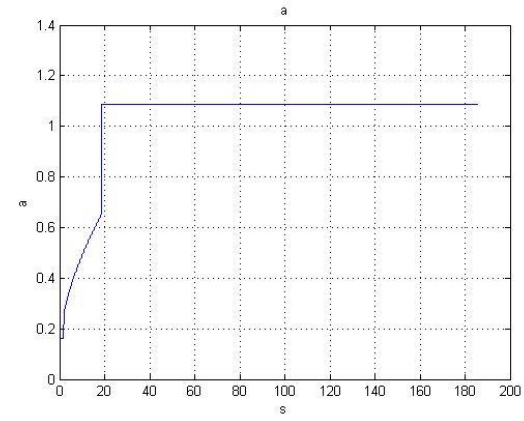
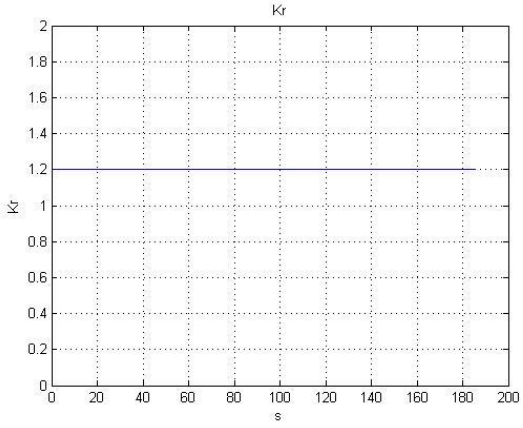
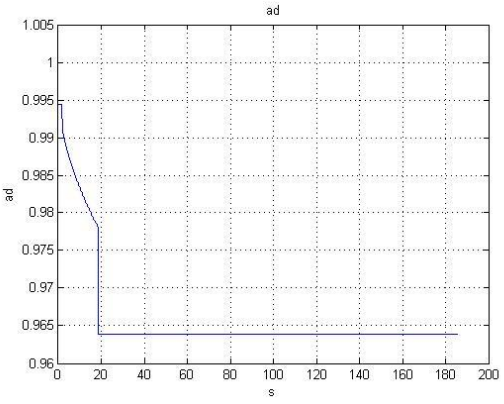
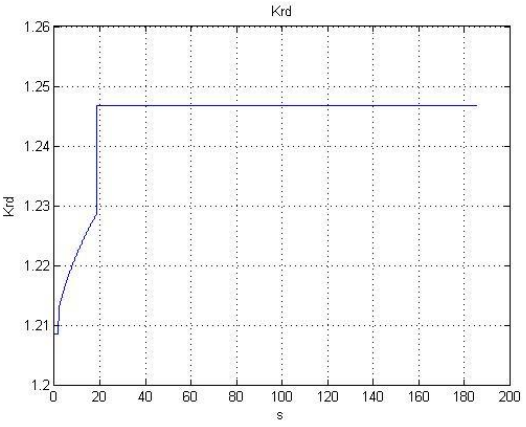
Se han llevado a cabo 6 simulaciones:

- Simulación 1

Se tienen los siguientes parámetros:

Método de cálculo de a	Utilizando regulador P
Método de cálculo del controlador discreto	Discretización del controlador continuo
Errores en d	No
Errores de actuación	No
Punto inicial	$d_{inicial} = 0.1$ $\vartheta_{e-inicial} = 0$ $s_{inicial} = 0$
Consigna de velocidad (en Km/h)	
Lectura de códigos	No
Cálculo de ϕ_{lin}	Método exacto
Circuito	INSIA 
Frecuencias de muestreo	$f = 29\ Hz$ $f_2 = 500\ Hz$

Se obtienen las siguientes gráficas de salida:

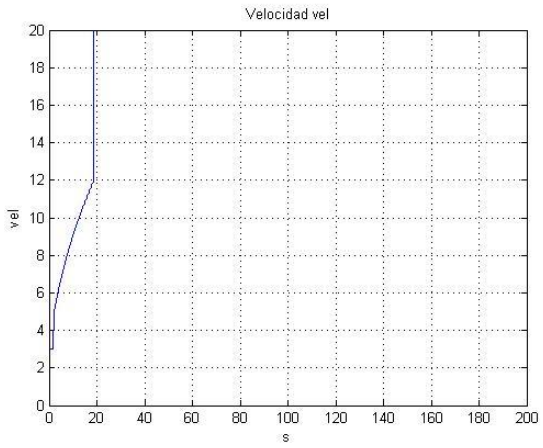
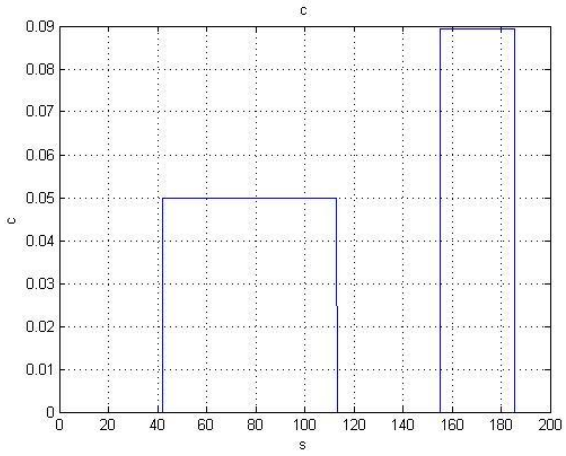
<table>
<tr><td> 
</td><td> 
</td></tr>
<tr><td> 
</td><td> 
</td></tr>
<tr><td> 
</td><td> 
</td></tr>
<tr><td>Error medio cuadrático del error:</td><td>0.0252</td></tr>
</table>

Se observa que el controlador es siempre estable, y responde adecuadamente.

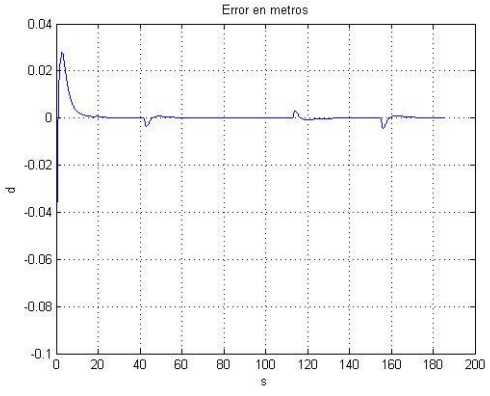
- <u>Simulación 2</u>

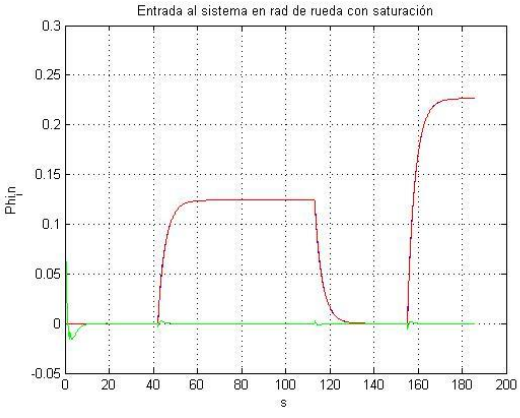
Se tienen los siguientes parámetros:

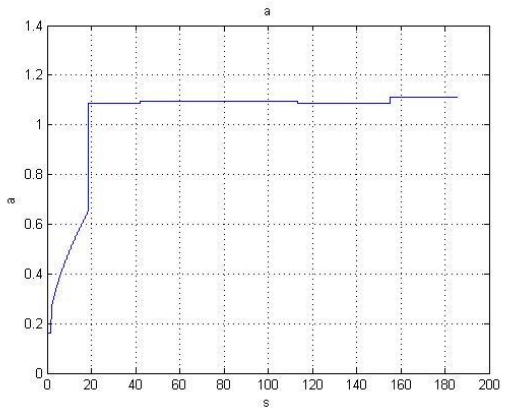
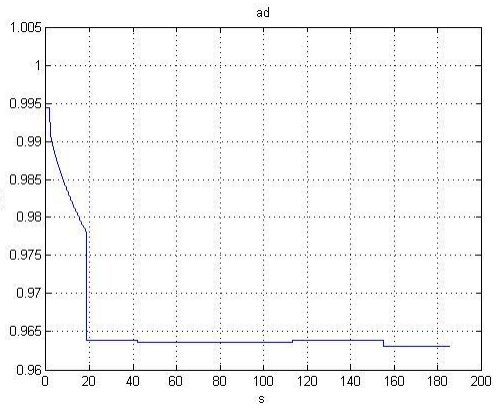
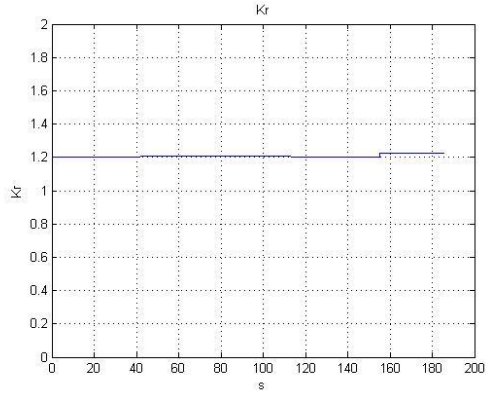
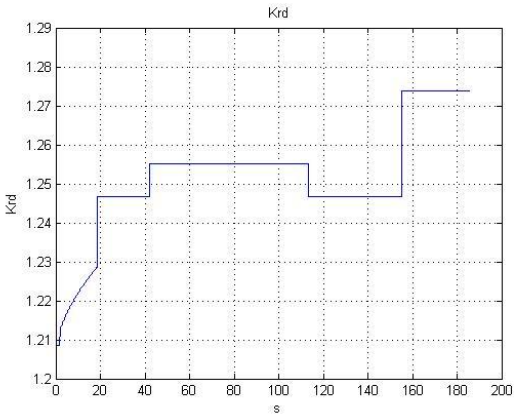
Método de cálculo de a	Utilizando regulador P
Método de cálculo del controlador discreto	Discretización del controlador continuo
Errores en d	No
Errores de actuación	No

Punto inicial	$d_{inicial} = 0.1$ $\vartheta_{e-inicial} = 0$ $s_{inicial} = 0$
Consigna de velocidad (en Km/h)	
Lectura de códigos	Si
Cálculo de ϕ_{lin}	Método exacto
Circuito	INSIA 
Frecuencias de muestreo	$f = 29\ Hz$ $f_2 = 500\ Hz$

Se obtienen las siguientes gráficas de salida:





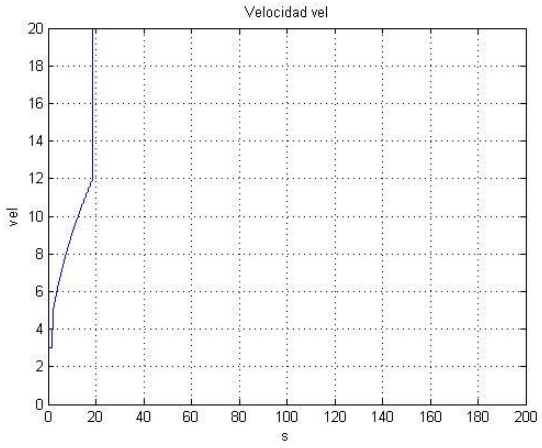
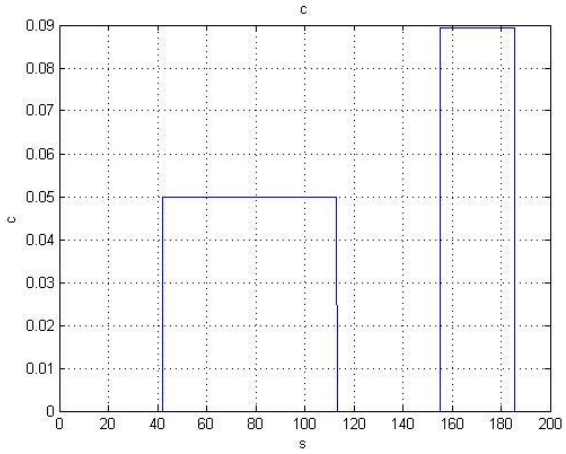
 	 
Error medio cuadrático del error:	0.0145

Se comprueba que si se conocen los códigos, el error es menor.

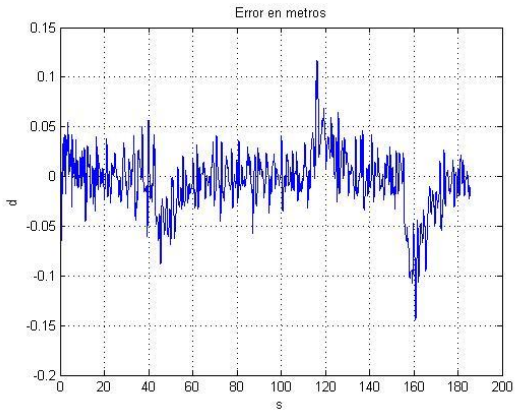
- <u>Simulación 3</u>

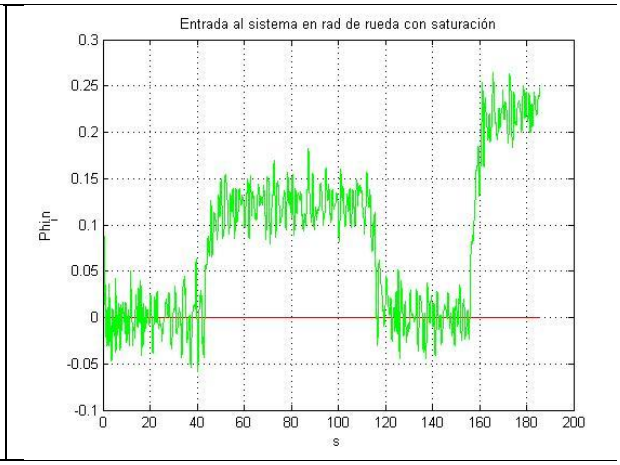
Se tienen los siguientes parámetros:

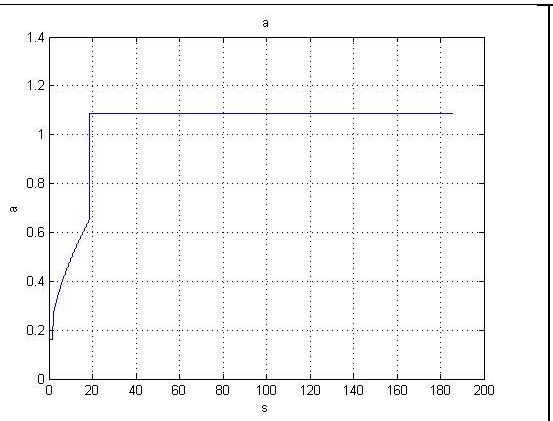
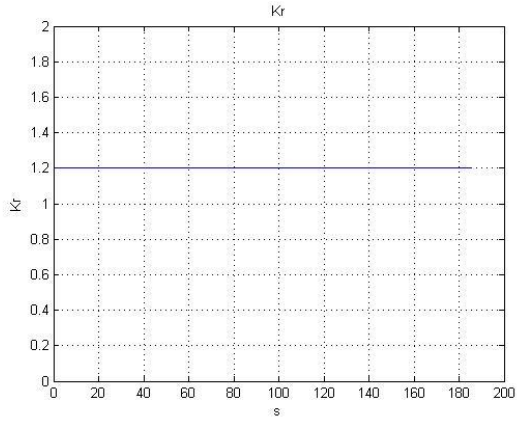
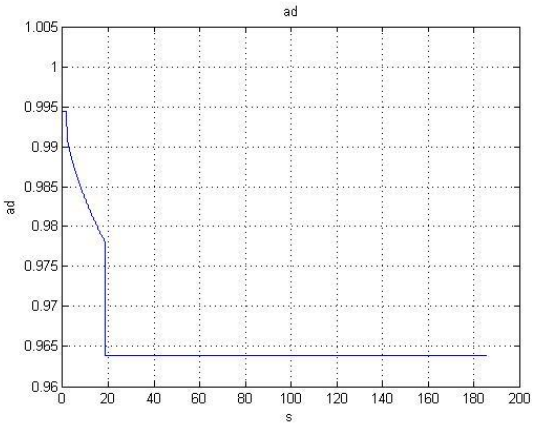
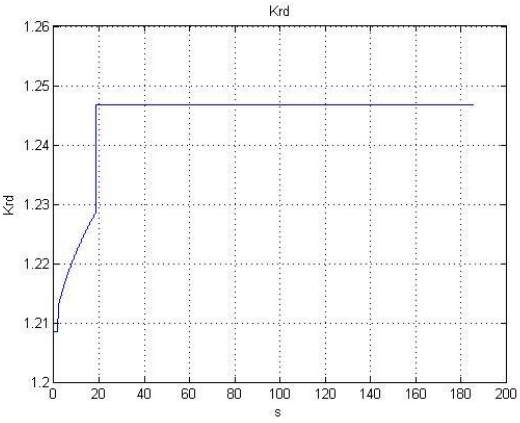
Método de cálculo de a	Utilizando regulador P
Método de cálculo del controlador discreto	Discretización del controlador continuo
Errores en d	Si, de precisión a 10 Hz
Errores de actuación	No
Punto inicial	$d_{inicial} = 0.1$ $\vartheta_{e-inicial} = 0$ $s_{inicial} = 0$

Consigna de velocidad (en Km/h)	
Lectura de códigos	No
Cálculo de ϕ_{lin}	Método exacto
Circuito	INSIA 
Frecuencias de muestreo	$f = 29\ Hz$ $f_2 = 500\ Hz$

Se obtienen las siguientes gráficas de salida:





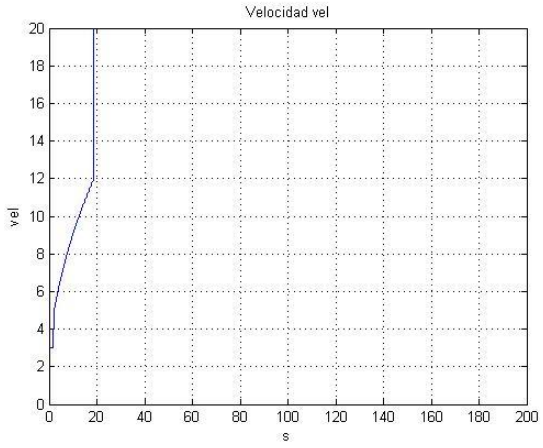
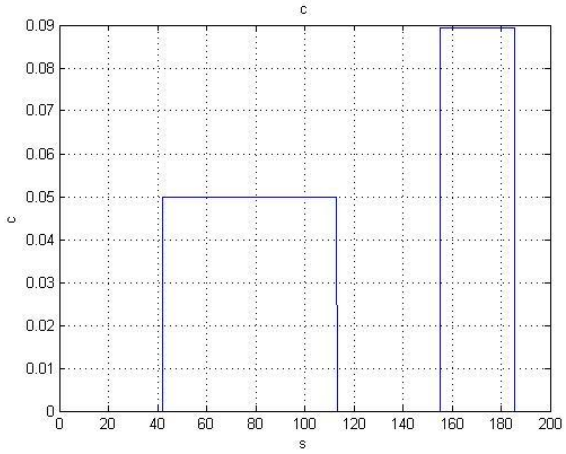
	
	
Error medio cuadrático del error:	0.0307

Aún con errores no modelados de precisión en d, y sin conocimiento de los códigos, el controlador responde adecuadamente.

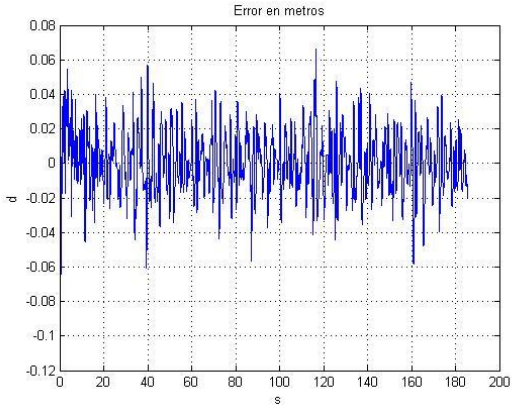
- <u>Simulación 4</u>

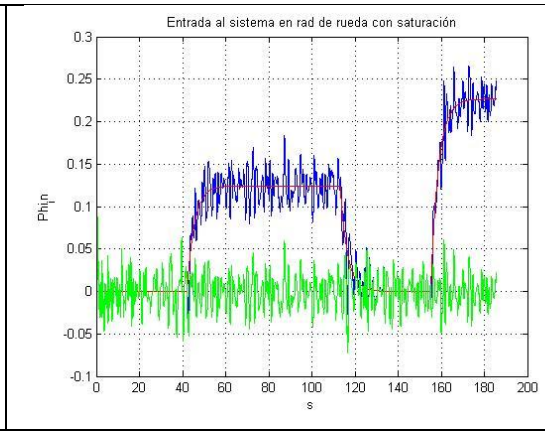
Se tienen los siguientes parámetros:

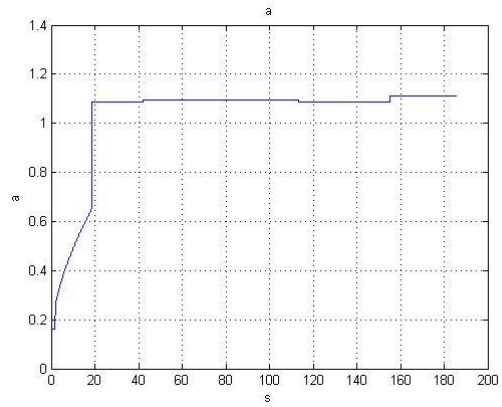
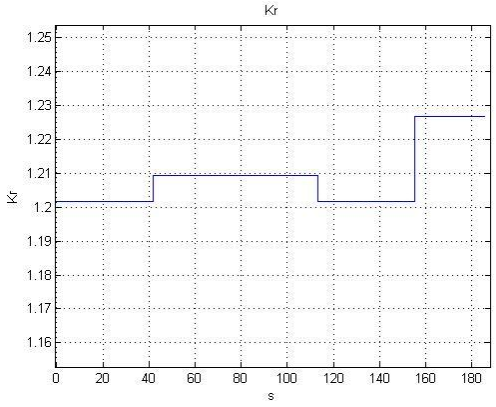
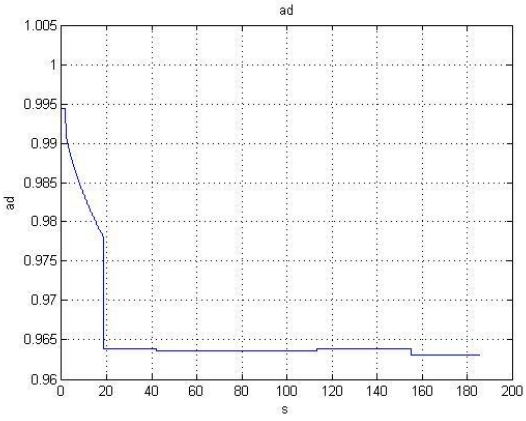
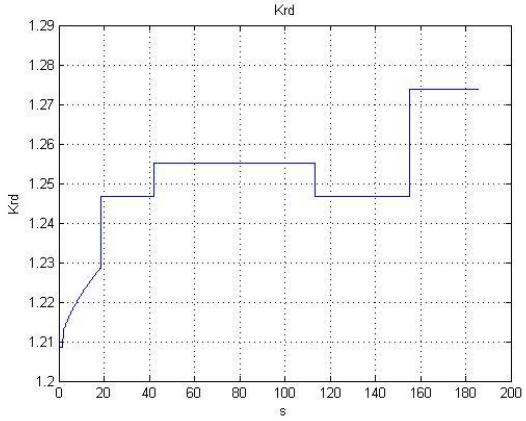
Método de cálculo de a	Utilizando regulador P
Método de cálculo del controlador discreto	Discretización del controlador continuo
Errores en d	Si, de precisión a 10 Hz
Errores de actuación	No
Punto inicial	$d_{inicial} = 0.1$ $\vartheta_{e-inicial} = 0$ $s_{inicial} = 0$

Consigna de velocidad (en Km/h)	
Lectura de códigos	Si
Cálculo de ϕ_{lin}	Método exacto
Circuito	INSIA 
Frecuencias de muestreo	$f = 29\ Hz$ $f_2 = 500\ Hz$

Se obtienen las siguientes gráficas de salida:





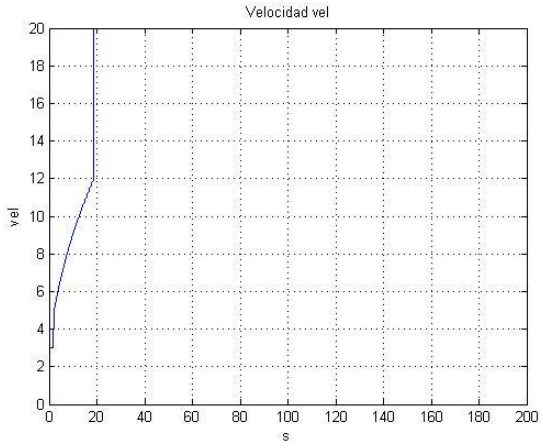
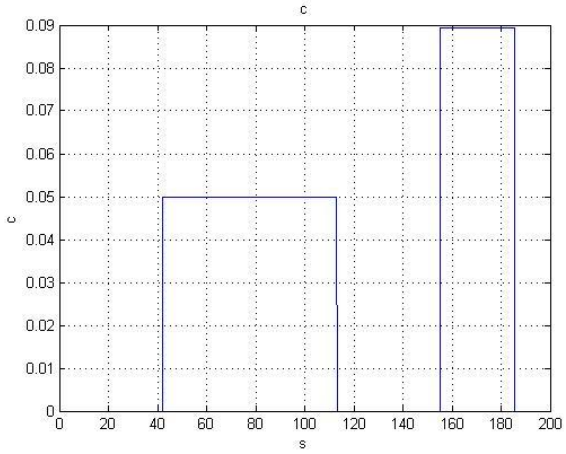
	
	
Error medio cuadrático del error:	0.0225

Aún con errores no modelados de precisión en d, el controlador responde adecuadamente, minimizándose el error en caso de conocer los códigos.

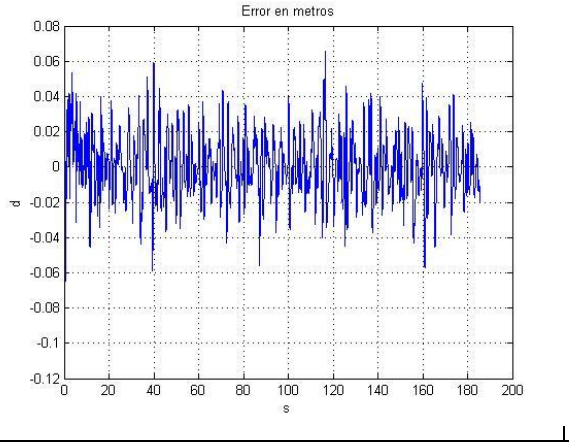
- Simulación 5

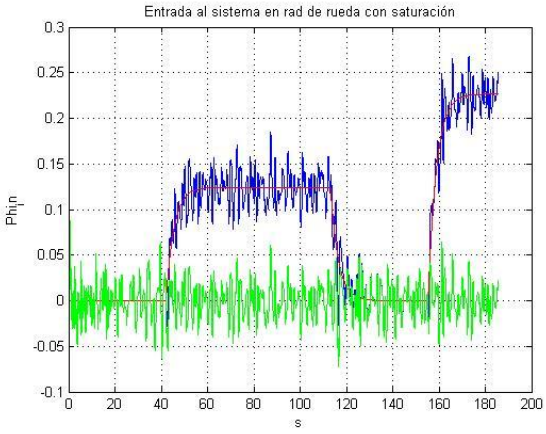
Se tienen los siguientes parámetros:

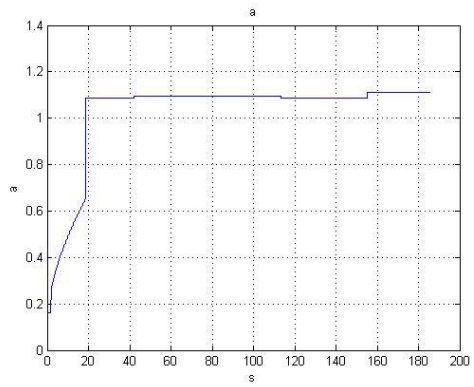
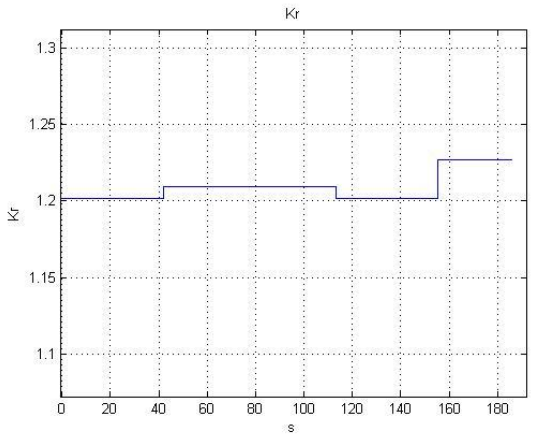
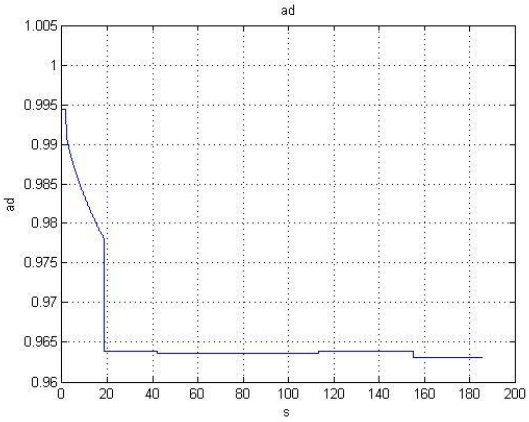
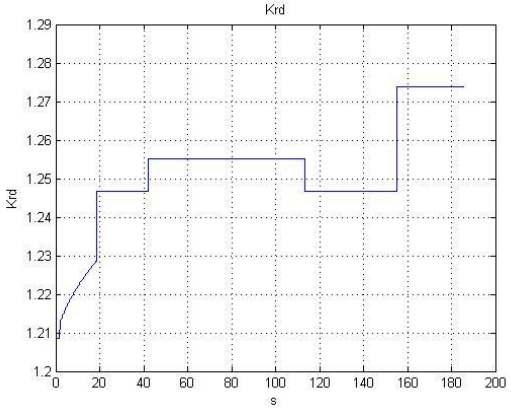
Método de cálculo de a	Utilizando regulador P
Método de cálculo del controlador discreto	Discretización del controlador continuo
Errores en d	Si, de precisión a 10 Hz
Errores de actuación	No
Punto inicial	$d_{inicial} = 0.1$ $\vartheta_{e-inicial} = 0$ $s_{inicial} = 0$

Consigna de velocidad (en Km/h)	
Lectura de códigos	Si
Cálculo de ϕ_{lin}	Método exacto
Circuito	INSIA 
Frecuencias de muestreo	$f = 29\ Hz$ $f_2 = 29\ Hz$

Se obtienen las siguientes gráficas de salida:





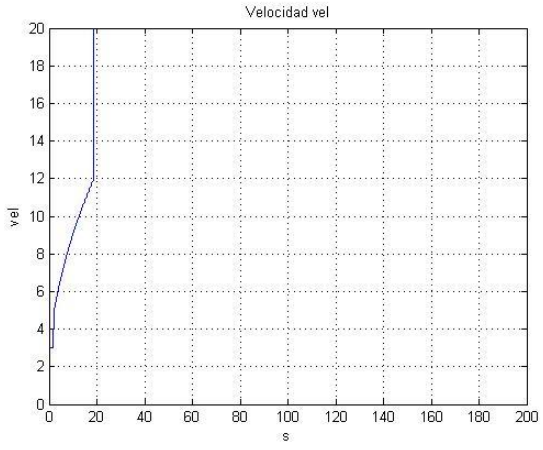
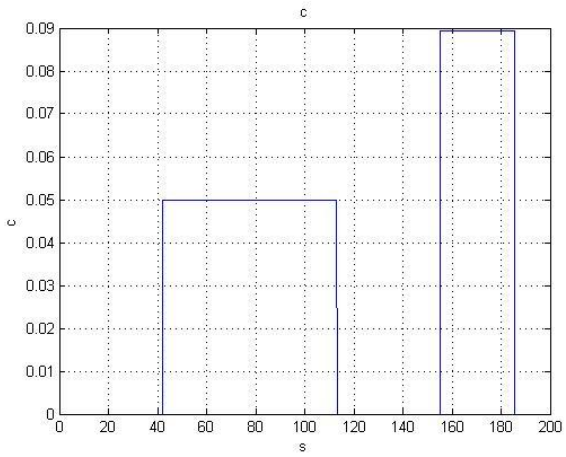
	
	
Error medio cuadrático del error:	0.0226

Se observa que el hecho de calcular el punto de linealización a una frecuencia mayor que la del sistema de visión aporta una ventaja ínfima, por lo que se podrá poner a trabajar a la misma frecuencia.

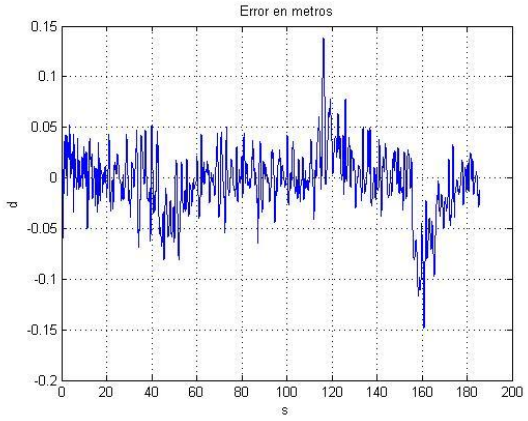
- Simulación 6

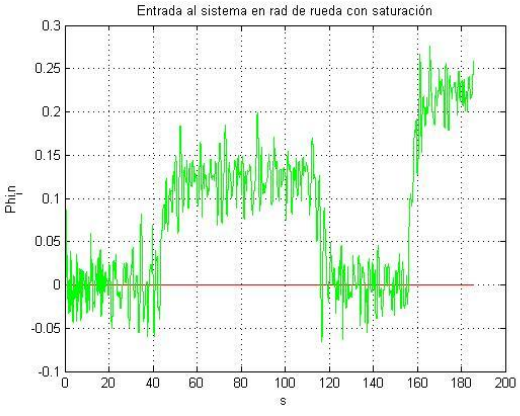
Se tienen los siguientes parámetros:

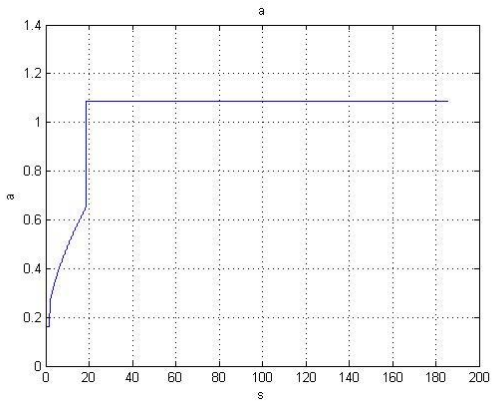
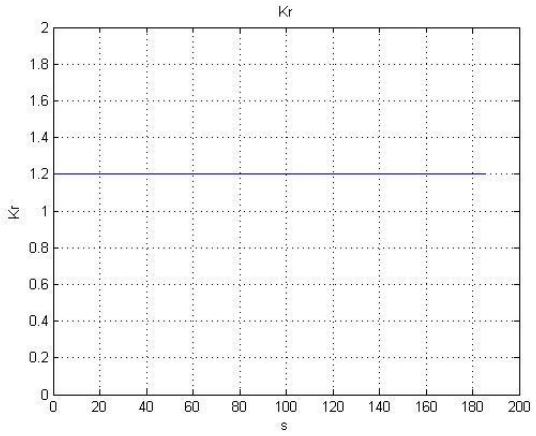
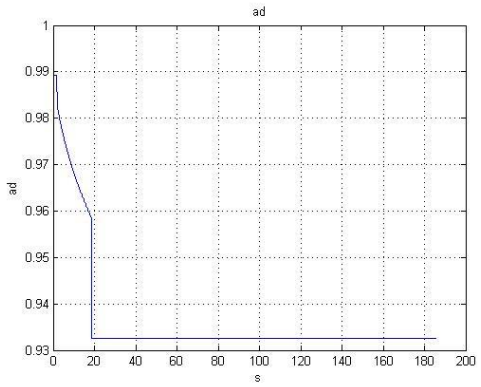
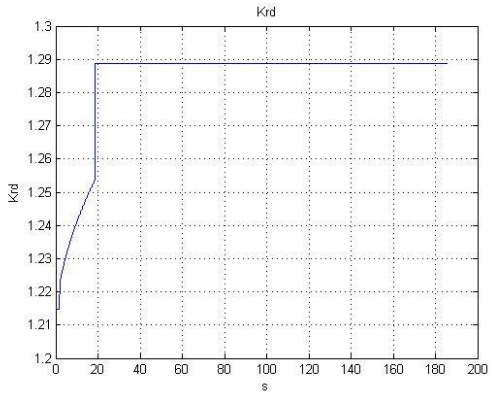
Método de cálculo de a	Utilizando regulador P
Método de cálculo del controlador discreto	Discretización del controlador continuo
Errores en d	Si, de precisión a 10 Hz
Errores de actuación	No
Punto inicial	$d_{inicial} = 0.1$ $\vartheta_{e-inicial} = 0$ $s_{inicial} = 0$

Consigna de velocidad (en Km/h)	
Lectura de códigos	No
Cálculo de ϕ_{lin}	Método exacto
Circuito	INSIA 
Frecuencias de muestreo	$f = 15\ Hz$ $f_2 = 500\ Hz$

Se obtienen las siguientes gráficas de salida:





	
	
Error medio cuadrático del error:	0.0323

Se concluye con esta simulación que aún funcionando a frecuencia reducida, el controlador responde adecuadamente.

ANEXO VIII: CÓDIGO C++ DEL CONTROLADOR PARA EL UGV REAL

Se incluye en este anexo el código en C++ utilizado en el UGV real para programar el controlador.

Se parte de las variables que se citaron en el Apartado 3.4.

```
      //Variables que ya vienen
      double turn; //en ° volante
      double error; //en pix
      double vel_actual; //En km/h
      int tramo;
      float T; //Hz Frecuencia
```

Se incluyen en el main del código las siguientes variables:

```
      //Variables necesarias para regulador incremental
      double Dturn;
      double Dturn_ant=0;
      double error_ant=0;


      //Variables necesarias para punto de linealizacion
      double Turn_lin=0;
      int cambio_tramo=0;
      int tramo_ant=0; //almacena información del tramo anterior
      double t_lin=0; //Almacena el tiempo desde que cambió de tramo
```

En la zona de código donde se detecta un nuevo tramo, se incluye el siguiente fragmento de código, justo después de la aceptación de detección de código de suelo válido, y antes de proceder al cambio de la variable tramo.

```
cambio_tramo=InicioCambioTramo(tramo,&tramo_ant,&t_lin);
```

En la zona destinada al control, que va después de la detección de nuevo tramo, y de la lectura y decodificación del error, se incluye el siguiente fragmento de código.

```
      //Comienza el control

      //////////////Regulador incremental///////////////////////
      Dturn=ReguladorIncremental(error,vel_actual,tramo,T,&error_ant,&Dtu
rn_ant);


      ////////////// Punto de linealizacion ///////////////////////
      //Actualizamos el tiempo, si toca
      if(cambio_tramo) cambio_tramo=0; //Si ha habido un cambio de tramo
en este periodo, no actualizamos el tiempo
      else TiempoLinealizacion(&t_lin,T); //Si no ha habido cambio de
tramo actualizamos
      //Calculo
      Turn_lin=PuntoLinealizacion(vel_actual,tramo,tramo_ant,t_lin);
```

```
      ////////////// Acción de control final ///////////////////////
      turn=AccionControlFinal(Dturn,Turn_lin);

      //Termina el control
```

Previamente, en la zona de includes, ha sido necesario poner:

```
#include "ControladorJL.h"
```

Este include contiene:

ControladorJL.h

```
#ifndef LIBCONTROLADORJL_H_INCLUDED
#define LIBCONTROLADORJL_H_INCLUDED


//Includes
#include <math.h>
#include <stdio.h>


//Constantes
#define L 2.46
#define l1 3.41
#define PUNT 1./3.
#define TURN_MAX 300.


//Prototipos
double ReguladorIncremental(double, double, int, float, double*,
double*);
double PuntoLinealizacion(double, int, int, double);
double AccionControlFinal(double, double);
double Calculac(int);

int InicioCambioTramo(int, int*, double*);
int TiempoLinealizacion(double*,double);


#endif
```

Las funciones necesarias para el funcionamiento correcto del controlador están incluidas en la librería libControladorJL.a, cuyo código está incluido en el archivo libControladorJL.cpp, y son:

```
double ReguladorIncremental(double error, double vel_actual,int tramo,
float T, double* error_ant, double* Dturn_ant)
{

      ////////////// Declaración de variables //////////////////
      //Curvatura
      double c;
      //Periodo de muestreo
      double per=1./T;
```

```
//Variables del controlador incremental
//Punto de accion
double Dturn;
//Parametros controlador continuo
double Kr, a=0;
//Parametros controlador discreto
double Krd, ad;


////////// Cálculo de c ///////////
c=Calculac(tramo);


////////////// Regulador incremental ////////////////////

//Calculo de parámetros del regulador continuo
double u1=vel_actual/3.6;
double Theta_e=-asin(c*l1);
double A1, A2, A3;

//Valores de Ai
A1=u1*l1/L*cos(Theta_e)/pow(cos(atan(-L/l1*tan(Theta_e))),2);
A2=u1/(l1*pow(cos(Theta_e),2))*(1+c*l1*sin(Theta_e));
A3=pow(c*u1/(cos(Theta_e)),2);


//Calculo de a

int Metodo_a=1;

//Método 0 -> Valor fijo
if (Metodo_a==0)
      {
      a=2;
      }

//Método 1 -> Usando el controlador P
if (Metodo_a==1)
      {
      //Valor de a
      a=PUNT*(A2+sqrt(pow(A2,2)+A3));
      }

//Método 2 -> Experimental
else if (Metodo_a==2)
      {
      //Bucle de aproximación
      if (vel_actual<1)
            a=0;
      else if (vel_actual>=1 && vel_actual<5)
            a=0.05;
      else if (vel_actual>=5)
            {
            //Parametros minimos
            double vel_actual_min=5;
            double amin=0.2;
            //Parametros maximos
            double vel_actual_max=40;
            double amax=1.5;
            //recta
```

```
                  double m=(amax-amin)/(vel_actual_max-vel_actual_min);
                  double b=amax-m*vel_actual_max;
                  //Ecuacion
                  a=m*vel_actual+b;
                  }
            else
                  a=0;
            }


      //Cálculo de Kr

      //Se calculará el Kr en cada bucle, opcion de mejora, ver si ha
cambiado.
      if(A2!=0) //El cero del sistema es no nulo
      {
            //Calculo del punto de polos reales dobles

            //Tanteo
            //No llega. Se toma la posicion del cero
            //Posición del cero
            double s_nllega, nllega;

            s_nllega=-A2;
            nllega=pow(s_nllega,4)+(2*a+2*A2)*pow(s_nllega,3)+(-
A3+3*a*A2)*pow(s_nllega,2)+A3*a*A2;
            while(nllega>0)
            {
                  s_nllega=s_nllega/1.5;
                  nllega=pow(s_nllega,4)+(2*a+2*A2)*pow(s_nllega,3)+(-
A3+3*a*A2)*pow(s_nllega,2)+A3*a*A2;
            }


            //Se pasa. se tantea
            double s_spasa, spasa;

            s_spasa=-1.5*A2;
            spasa=pow(s_spasa,4)+(2*a+2*A2)*pow(s_spasa,3)+(-
A3+3*a*A2)*pow(s_spasa,2)+A3*a*A2;
            while(spasa<0)
            {
                  s_spasa=1.5*s_spasa;
                  spasa=pow(s_spasa,4)+(2*a+2*A2)*pow(s_spasa,3)+(-
A3+3*a*A2)*pow(s_spasa,2)+A3*a*A2;
            }

            //Se calcula la posicion exacta por iteracion
            int fin=0;
            double tol=1e-4;
            double pos_polos=0;
            double s_nueva, que_pasa;

            while(fin==0)
            {
                  s_nueva=(s_spasa+s_nllega)/2;
                  que_pasa=pow(s_nueva,4)+(2*a+2*A2)*pow(s_nueva,3)+(-
A3+3*a*A2)*pow(s_nueva,2)+A3*a*A2;
                  //Se comprueba que ha sucedido
                  if (que_pasa==0 || abs(que_pasa)<tol)
                  {
```

```
                        fin=1;
                        pos_polos=s_nueva;
                  }
                  else if(que_pasa>0)
                        s_spasa=s_nueva;
                  else if (que_pasa<0)
                        s_nllega=s_nueva;
            }

            //////////////Calculo KR utilizando el criterio del modulo
            double d1=-pos_polos;
            Kr=d1*(pow(d1,2)+A3)/((d1-a)*(d1-A2)*abs(A1));
      }

      else Kr=0;


      //Cálculo de parámetros del regulador discretizado
      ad=1/(a*per+1);
      Krd=Kr*(a*per+1);


      //Estabilidad del controlador discreto
      double C1, C2, C3;
      C1=-3+Krd*A1*per;
      C2=(A3*pow(per,2)+1)+2+Krd*A1*per*((A2*per-1)-ad);
      C3=-(A3*pow(per,2)+1)-Krd*A1*per*(A2*per-1)*ad;

      //se utiliza el criterio de Jury para ver la estabilidad
      int estable;
      //inicializacion en cada bucle
      estable=0;
      //Criterio de Jury
      if ( (1+C1+C2+C3>0) && (-1+C1-C2+C3<0) && (abs(C3)<1 &&
abs(pow(C3,2)-1)>abs(C1*C3-C2)) )
            estable=1;

      if (!estable) printf("Sistema inestable\n");

      //Fórmula del regulador discreto
      Dturn=*Dturn_ant+Krd*(180*18*0.00132/3.1415)*(error-
ad*(*error_ant));


      //Actualizacion de valores para siguiente iteración
      *Dturn_ant=Dturn;
      *error_ant=error;


      //Resetea el controlador si la velocidad es nula
      if(vel_actual==0.0)
      {
        *Dturn_ant=0.0;
      }


      //Devuelve el valor
      return Dturn;
}
```

```
double PuntoLinealizacion(double vel_actual, int tramo, int tramo_ant,
double t_lin)
{
	//Variable a devolver
	double Turn_lin;
	//Parametros UGV
	double u1_lin=vel_actual/3.6;
	//Punto Theta_e
	double Theta_e_lin=0;

	//Curvatura actual
	double c;
	c=Calculac(tramo);

	//Curvatura antigua
	double c_ini;
	c_ini=Calculac(tramo_ant);

	//Calculo con ecuacion exacta
	Theta_e_lin=asin(exp(-u1_lin/l1*t_lin)*(c-c_ini)*l1-c*l1);

	//Salida Turn_lin
	Turn_lin=-180*18/3.1415*atan(-L/l1*tan(Theta_e_lin));

	return Turn_lin;

}
```

```
double AccionControlFinal(double Dturn, double Turn_lin)
{
	//Variables
	double turn;

	//Valor final
	turn=Dturn+Turn_lin;

	//Saturación SW para evitar problemas
	if(turn<0 && turn<(-1*TURN_MAX))
		turn=-1*TURN_MAX;
	else if(turn>0 && turn>TURN_MAX)
		turn=1*TURN_MAX;
	else
		turn=turn;

	//Devolvemos el valor final
	return turn;

}
```

```
double Calculac(int tramo)
{

	//Variable
	double c;
	//Bucle
	switch (tramo)
	{
		case 0:
		case 2:
```

```
                        c=0;
                        break;
                case 1:
                        c=1./20.;
                        break;
                case 3:
                        c=1./11.2;
                        break;
                default:
                        c=0;
        }
        return c;
}
```

```
//Funcion que actualiza el tiempo con la frecuencia
int TiempoLinealizacion(double* t_lin,double T)
{
        //Se suma el tiempo
        *t_lin=(*t_lin)+1./T;
        //Acaba
        return 1;

}
```

```
//Función que hay que correrse en cuanto se detecta un código válido,
antes de cambiar el tramo
int InicioCambioTramo(int tramo, int* tramo_ant, double* t_ini)
{
        //Se resetea el tiempo
        *t_ini=0;
        //Se actualiza el valor de tramo_ant
        *tramo_ant=tramo;
        //Acaba
        return 1;

}
```

El archivo libControladorJL.cpp, ha de contener el include al ControladorJL.h y el código de las funciones expuesto anteriormente:

El archivo makefile para compilar la libreria será:

```
#Variables de direcciones
LIB_S=../lib_sources/
SOURC=../sources/
INC=../includes/
OBJ=../objects/
LIB=../lib/



#Crear libreria
$(LIB)libControladorJL.a:       $(OBJ)libControladorJL.o
        ar -r $(LIB)libControladorJL.a $(OBJ)libControladorJL.o
```

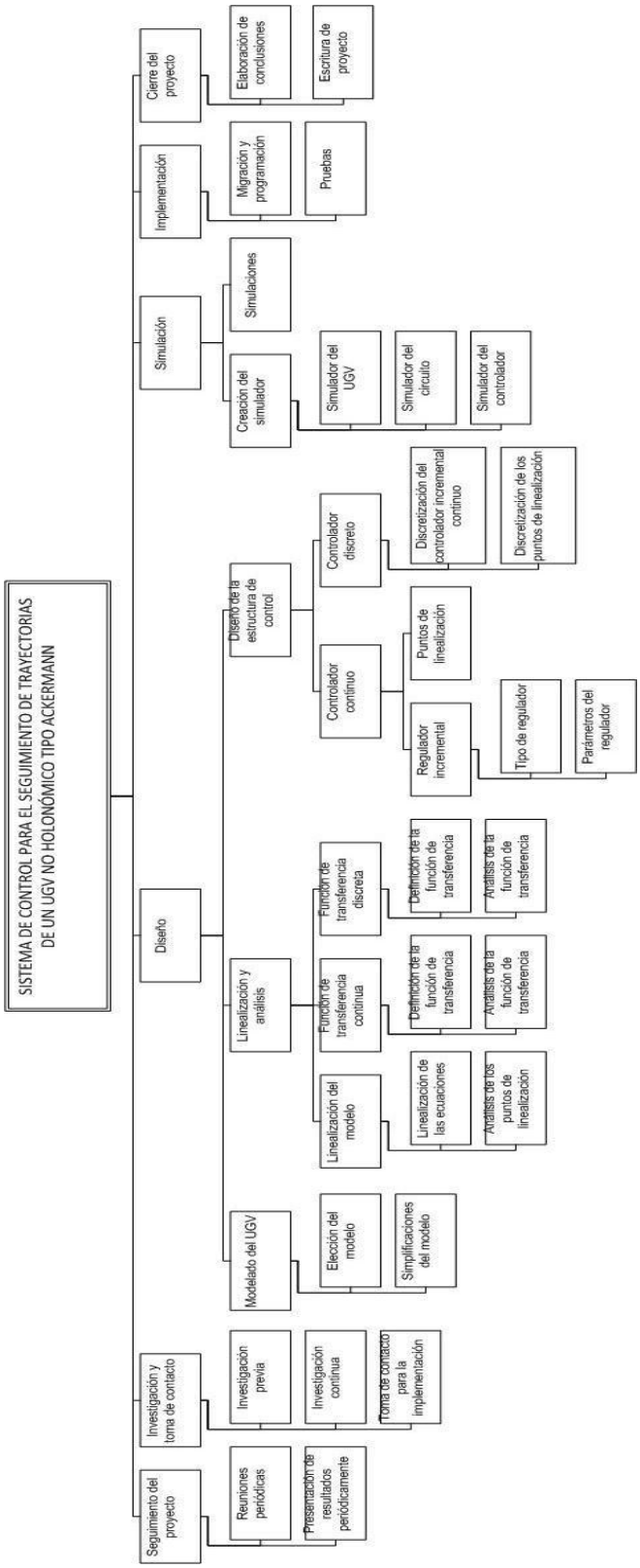
```
#Compiler
$(OBJ)libControladorJL.o: $(LIB_S)libControladorJL.cpp
	gcc -c $(LIB_S)libControladorJL.cpp -o $(OBJ)libControladorJL.o -I$(INC)
```

Que requiere de una serie de carpetas en las que está el código de la librería, y donde saldrá el archivo libControladorJL.a.

A la hora de compilar el programa que va a correr sobre el UGV será necesario linkar convenientemente la librería libControladorJL.a

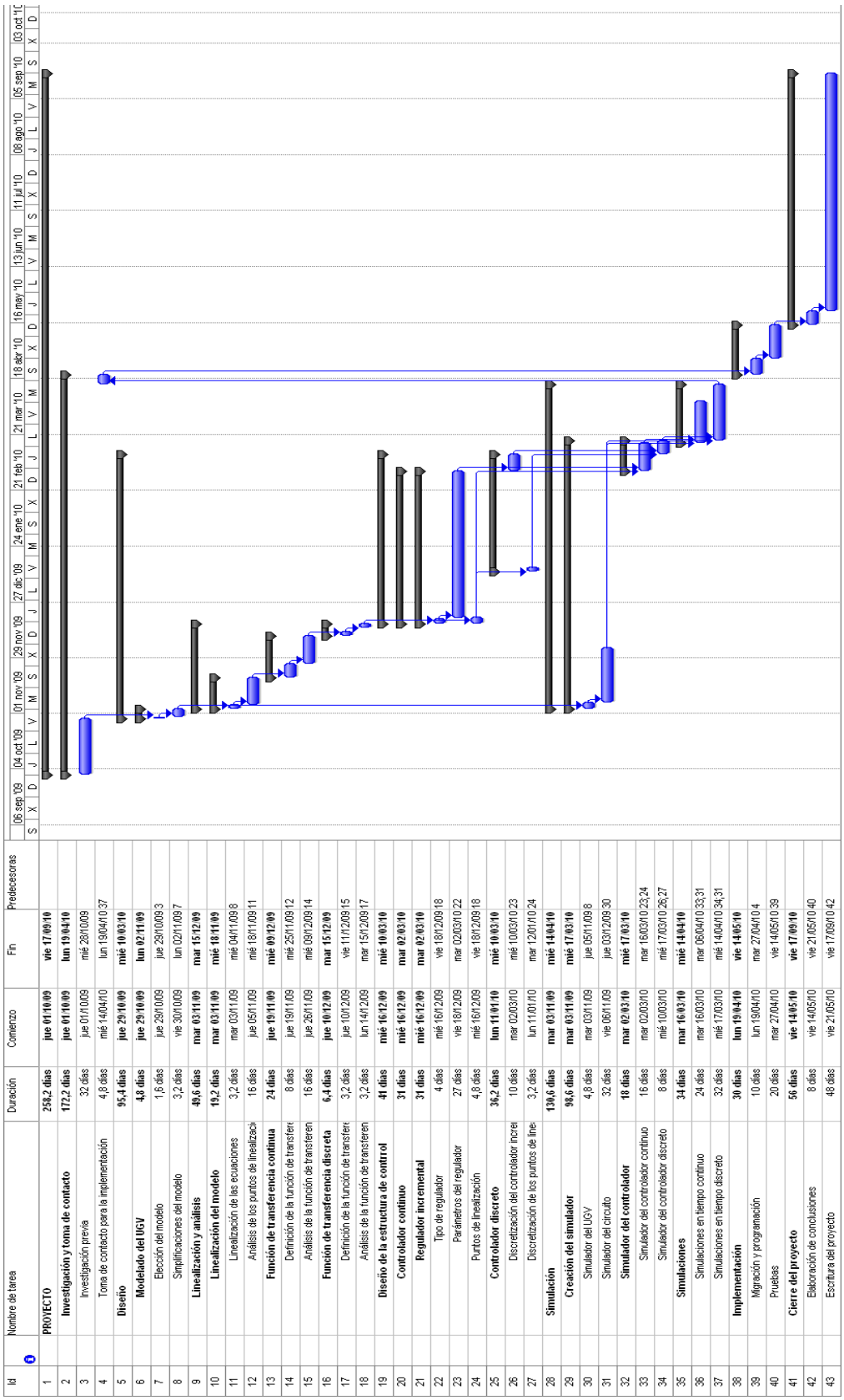
ANEXO IX: ESTRUCTURA DE DESCOMPOSICIÓN DEL PROYECTO (EDP)

Se incluye en este anexo la EDP completa, analizada en el Apartado 9.1.



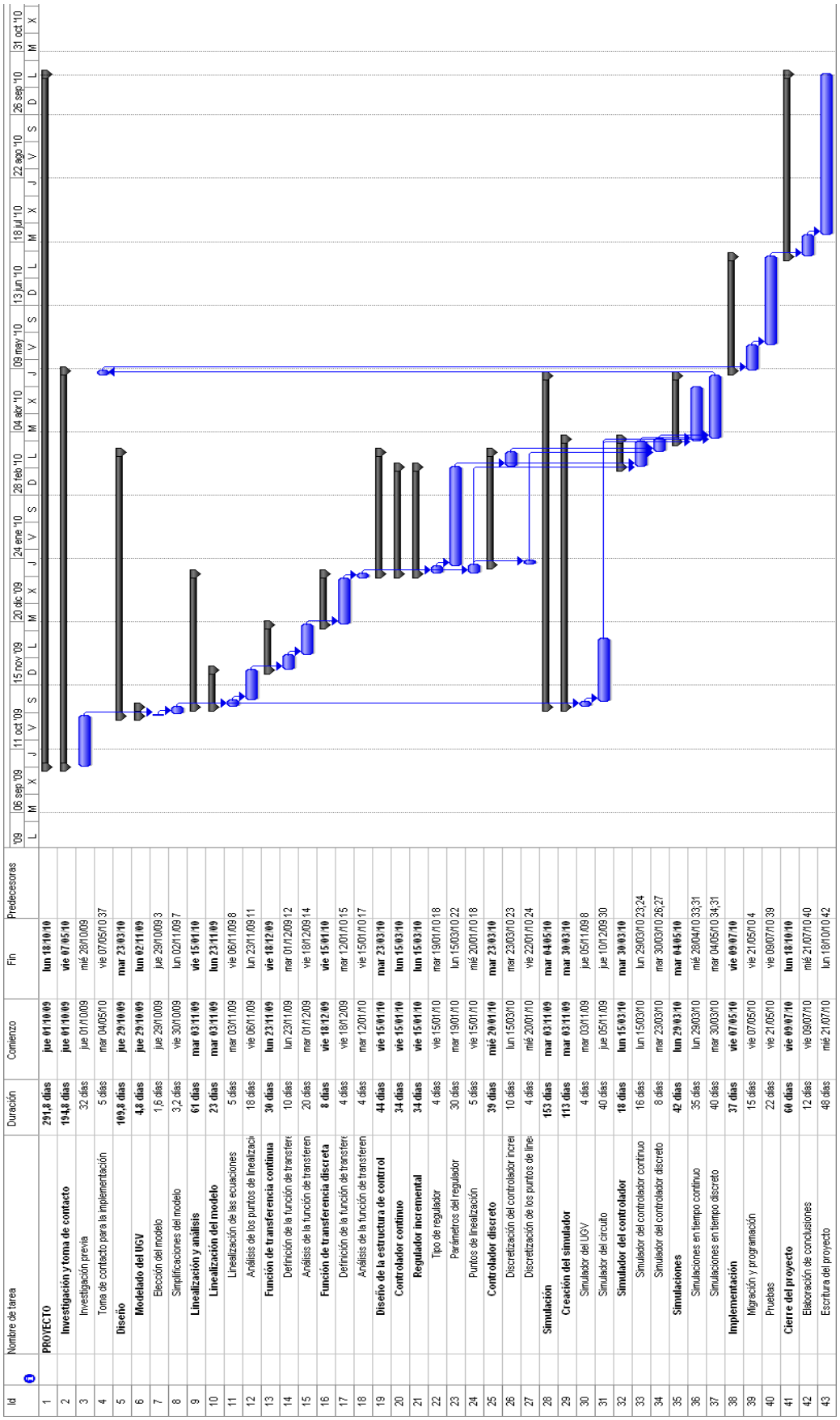
ANEXO X: DIAGRAMA DE GANTT

El diagrama de GANTT planificado es:



Id	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras
1	**PROYECTO**	**258,2 días**	**jue 01/10/09**	**vie 17/09/10**	
2	**Investigación y toma de contacto**	**172,2 días**	**jue 01/10/09**	**lun 19/04/10**	
3	Investigación previa	32 días	jue 01/10/09	mié 28/10/09	
4	Toma de contacto para la implementación	4,8 días	mié 14/04/10	lun 19/04/10	37
5	**Diseño**	**95,4 días**	**jue 29/10/09**	**mié 10/03/10**	
6	**Modelado del UGV**	**4,8 días**	**jue 29/10/09**	**lun 02/11/09**	
7	Elección del modelo	1,6 días	jue 29/10/09	jue 29/10/09	3
8	Simplificaciones del modelo	3,2 días	vie 30/10/09	lun 02/11/09	7
9	**Linealización y análisis**	**49,6 días**	**mar 03/11/09**	**mar 15/12/09**	
10	**Linealización del modelo**	**19,2 días**	**mar 03/11/09**	**mié 18/11/09**	
11	Linealización de las ecuaciones	3,2 días	mar 03/11/09	mié 04/11/09	8
12	Análisis de los puntos de linealizaci	16 días	jue 05/11/09	mié 18/11/09	11
13	**Función de transferencia continua**	**24 días**	**jue 19/11/09**	**mié 09/12/09**	
14	Definición de la función de transferi	8 días	jue 19/11/09	mié 25/11/09	12
15	Análisis de la función de transferen	16 días	jue 26/11/09	mié 09/12/09	14
16	**Función de transferencia discreta**	**6,4 días**	**jue 10/12/09**	**mar 15/12/09**	
17	Definición de la función de transferi	3,2 días	jue 10/12/09	vie 11/12/09	15
18	Análisis de la función de transferen	3,2 días	lun 14/12/09	mar 15/12/09	17
19	**Diseño de la estructura de control**	**41 días**	**mié 16/12/09**	**mié 10/03/10**	
20	**Controlador continuo**	**31 días**	**mié 16/12/09**	**mar 02/03/10**	
21	**Regulador incremental**	**31 días**	**mié 16/12/09**	**mar 02/03/10**	
22	Tipo de regulador	4 días	mié 16/12/09	vie 18/12/09	18
23	Parámetros del regulador	27 días	vie 18/12/09	mar 02/03/10	22
24	Puntos de linealización	4,8 días	mié 16/12/09	vie 18/12/09	18
25	**Controlador discreto**	**36,2 días**	**lun 11/01/10**	**mié 10/03/10**	
26	Discretización del controlador incre	10 días	mar 02/03/10	mié 10/03/10	23
27	Discretización de los puntos de line	3,2 días	lun 11/01/10	mar 12/01/10	24
28	**Simulación**	**130,6 días**	**mar 03/11/09**	**mié 14/04/10**	
29	**Creación del simulador**	**98,6 días**	**mar 03/11/09**	**mié 17/03/10**	
30	Simulador del UGV	4,8 días	mar 03/11/09	jue 05/11/09	8
31	Simulador del circuito	32 días	vie 06/11/09	jue 03/12/09	30
32	**Simulador del controlador**	**18 días**	**mar 02/03/10**	**mié 17/03/10**	
33	Simulador del controlador continuo	16 días	mar 02/03/10	mar 16/03/10	23;24
34	Simulador del controlador discreto	8 días	mié 10/03/10	mié 17/03/10	26;27
35	**Simulaciones**	**34 días**	**mar 16/03/10**	**mié 14/04/10**	
36	Simulaciones en tiempo continuo	24 días	mar 16/03/10	mar 06/04/10	33;31
37	Simulaciones en tiempo discreto	32 días	mié 17/03/10	mié 14/04/10	34;31
38	**Implementación**	**30 días**	**lun 19/04/10**	**vie 14/05/10**	
39	Migración y programación	10 días	lun 19/04/10	mar 27/04/10	4
40	Pruebas	20 días	mar 27/04/10	vie 14/05/10	39
41	**Cierre del proyecto**	**56 días**	**vie 14/05/10**	**vie 17/09/10**	
42	Elaboración de conclusiones	8 días	vie 14/05/10	vie 21/05/10	40
43	Escritura del proyecto	48 días	vie 21/05/10	vie 17/09/10	42

El diagrama de GANTT real es:



Id	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras
1	**PROYECTO**	**291,8 días**	**jue 01/10/09**	**lun 18/10/10**	
2	**Investigación y toma de contacto**	**194,8 días**	**jue 01/10/09**	**vie 07/05/10**	
3	Investigación previa	32 días	jue 01/10/09	mié 28/10/09	
4	Toma de contacto para la implementación	5 días	mar 04/05/10	vie 07/05/10	37
5	**Diseño**	**109,8 días**	**jue 29/10/09**	**mar 23/03/10**	
6	**Modelado del UGV**	**4,8 días**	**jue 29/10/09**	**lun 02/11/09**	
7	Elección del modelo	1,6 días	jue 29/10/09	jue 29/10/09	3
8	Simplificaciones del modelo	3,2 días	vie 30/10/09	lun 02/11/09	7
9	**Linealización y análisis**	**61 días**	**mar 03/11/09**	**vie 15/01/10**	
10	**Linealización del modelo**	**23 días**	**mar 03/11/09**	**lun 23/11/09**	
11	Linealización de las ecuaciones	5 días	mar 03/11/09	vie 06/11/09	8
12	Análisis de los puntos de linealizaci	18 días	vie 06/11/09	lun 23/11/09	11
13	**Función de transferencia continua**	**30 días**	**lun 23/11/09**	**vie 18/12/09**	
14	Definición de la función de transferc	10 días	lun 23/11/09	mar 01/12/09	12
15	Análisis de la función de transferen	20 días	mar 01/12/09	vie 18/12/09	14
16	**Función de transferencia discreta**	**8 días**	**vie 18/12/09**	**vie 15/01/10**	
17	Definición de la función de transferc	4 días	vie 18/12/09	mar 12/01/10	15
18	Análisis de la función de transferen	4 días	mar 12/01/10	vie 15/01/10	17
19	**Diseño de la estructura de control**	**44 días**	**vie 15/01/10**	**mar 23/03/10**	
20	**Controlador continuo**	**34 días**	**vie 15/01/10**	**lun 15/03/10**	
21	**Regulador incremental**	**34 días**	**vie 15/01/10**	**lun 15/03/10**	
22	Tipo de regulador	4 días	vie 15/01/10	mar 19/01/10	18
23	Parámetros del regulador	30 días	mar 19/01/10	lun 15/03/10	22
24	Puntos de linealización	5 días	vie 15/01/10	mié 20/01/10	18
25	**Controlador discreto**	**39 días**	**mié 20/01/10**	**mar 23/03/10**	
26	Discretización del controlador increi	10 días	lun 15/03/10	mar 23/03/10	23
27	Discretización de los puntos de line	4 días	mié 20/01/10	vie 22/01/10	24
28	**Simulación**	**153 días**	**mar 03/11/09**	**mar 04/05/10**	
29	**Creación del simulador**	**113 días**	**mar 03/11/09**	**mar 30/03/10**	
30	Simulador del UGV	4 días	mar 03/11/09	jue 05/11/09	8
31	Simulador del circuito	40 días	jue 05/11/09	jue 10/12/09	30
32	**Simulador del controlador**	**18 días**	**lun 15/03/10**	**mar 30/03/10**	
33	Simulador del controlador continuo	16 días	lun 15/03/10	lun 29/03/10	23;24
34	Simulador del controlador discreto	8 días	mar 23/03/10	mar 30/03/10	26;27
35	**Simulaciones**	**42 días**	**lun 29/03/10**	**mar 04/05/10**	
36	Simulaciones en tiempo continuo	35 días	lun 29/03/10	mié 28/04/10	33;31
37	Simulaciones en tiempo discreto	40 días	mar 30/03/10	mar 04/05/10	34;31
38	**Implementación**	**37 días**	**vie 07/05/10**	**vie 09/07/10**	
39	Migración y programación	15 días	vie 07/05/10	vie 21/05/10	4
40	Pruebas	22 días	vie 21/05/10	vie 09/07/10	39
41	**Cierre del proyecto**	**60 días**	**vie 09/07/10**	**lun 18/10/10**	
42	Elaboración de conclusiones	12 días	vie 09/07/10	mié 21/07/10	40
43	Escritura del proyecto	48 días	mié 21/07/10	lun 18/10/10	42