Universidad de Alcalá
Escuela Politécnica Superior

GRADO EN INGENIERÍA ELECTRÓNICA
DE COMUNICACIONES

Trabajo Fin de Grado

Control de Robots Aéreos en entorno Matlab-ROS utilizando el simulador V-REP

Autor: Óscar Pérez Gil

Tutor/es: M. Elena López Guillén

2018

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

Trabajo Fin de Grado

‘‘Control de Robots Aéreos en entorno Matlab-ROS utilizando el simulador V-REP’’

Óscar Pérez Gil

2018

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

GRADO EN INGENIERÍA ELECTRÓNICA DE COMUNICACIONES

Trabajo Fin de Grado

‘‘Control de Robots Aéreos en entorno Matlab-ROS utilizando el simulador V-REP’’

Autor: Óscar Pérez Gil

Tutor: María Elena López Guillén

TRIBUNAL:

Presidente: Pedro Revenga de Toro

Vocal 1º: Rafael Barea Navarro

Vocal 2º: María Elena López Guillén

FECHA:

ÍNDICE GENERAL

RESUMEN

RESUMEN EN CASTELLANO.

El fin de este proyecto, es la creación de un sistema de seguimiento entre un robot aéreo y uno terrestre. Lo primero que se va a realizar es una estructura de comunicación entre tres plataformas distintas, V-REP, ROS y Matlab/Simulink, con el fin de crear un entorno de simulación de drones, de manera que intercambien datos entre ellas y conseguir actuar sobre un dron.

Una vez realizado esto, se implementarán varias aplicaciones utilizando dicha estructura, creando varios lazos de control (en función de cada aplicación) para que el dron actúe de una forma determinada dependiendo de los datos disponibles para cada aplicación.

Palabras clave: dron, ROS, V-REP, Matlab/Simulink, seguimiento.

ABSTRACT.

The purpose of this project is the creation of a tracking system between an aerial and a terrestrial robot. The first thing that is going to be done is a communication structure between three different platforms, V-REP, ROS and Matlab / Simulink, in order to create a drone simulation environment, so that they exchange data between them and get to act on a drone.

Once this was done, several applications were implemented using this structure, creating several control loops for each application so that the drone acts in a certain way depending on the data available in each application.

Keywords: drone, V-REP, ROS, Matlab/Simulink, tracking.

RESUMEN EXTENDIDO.

El proyecto está enmarcado en la necesidad de la sociedad de utilizar las nuevas tecnologías para facilitar la vida a la especie humana. En este contexto, un grupo de la Universidad de Alcalá (*Robesafe*) mantiene activo un proyecto de creación desde cero, de un vehículo autónomo.

Dicho vehículo utiliza diversos sensores para recibir información del entorno y poder circular de forma correcta por la vía. Dicha información, puede ser reducida por las características propias del vehículo, por lo que podría ser útil implementar distintas técnicas para poder adquirir más datos complementarios para ayudar al sistema del vehículo a seguir la ruta correcta.

Una de estas técnicas es la que se trata en este proyecto, basada en la utilización de un dron o cuadricóptero de seguimiento del vehículo, el cual disponga de más datos de los que puede obtener en un principio el vehículo autónomo, y que pueda enviárselos. La utilización de un dron es interesante ya que en la actualidad tienen un gran auge en el mercado y se dispone de una gran variedad de ellos.

En este proyecto se implementa una primera etapa basada en simulación mediante el simulador robótico V-REP. En dicha plataforma se va a implementar el cuadricóptero para estudiar su comportamiento, poder realizar los ajustes necesarios para que siga al vehículo mediante Matlab/Simulink y todo ello teniendo como puente de comunicación el entorno de desarrollo ROS, puesto que en él se basa el proyecto del grupo *Robesafe*. Para alcanzar el objetivo final, el trabajo se va a dividir en diferentes etapas, siendo cada etapa necesaria para la implementación de la siguiente.

El primer escalón estará basado en el estudio de las plataformas a utilizar al nivel necesario para poder trabajar correctamente con ellas. De modo que el estudio de funcionamiento de ROS, V-REP y Matlab/Simulink es básico para la realización del proyecto.

Una vez establecidos los conocimientos básicos anteriores, el siguiente paso va a consistir en la creación o la búsqueda de un modelo de dron que pueda ser útil para el desarrollo de la aplicación utilizando la gran cantidad de ejemplos disponibles tanto en las librerías de V-REP, como en documentación disponible de otras universidades.

Para la realización de la comunicación entre ROS y V-REP se utiliza un puente entre ambos, el cual ya está configurado y es difícil de cambiar. Por tanto, a partir de dicho puente y de los datos que se pueden obtener a partir de él, se utiliza como medio para la realización del sistema de control del dron el programa matemático Matlab/Simulink.

Cuando ya se dispone de un modelo de dron, la primera aproximación sería la realización de un control de posición mediante coordenadas proporcionadas por algún dispositivo como podría ser un sistema de posicionamiento global convencional.

Para ello se realizará un modelo de Simulink utilizando la información que proporciona el propio V-REP de la posición y ángulos del dron, construyendo un sistema de control basado en la transformación de coordenadas del dron, en coordenadas absolutas del sistema de referencia elegido.

En la siguiente etapa se diseña una plataforma móvil en el simulador, cuya misión es simular la visión superior de un vehículo que tendría el dron al estar en el aire. Para poder realizar un control de seguimiento de dicha plataforma, se le proporciona un color característico para que a la hora de segmentar la imagen se pueda detectar con facilidad; a su vez, también se le añade una marca para poder detectar su orientación.

El lazo de control en esta parte se realiza a través de la imagen que se recibe de un sensor de visión incorporado en el avión no tripulado, de manera que, a partir de dicha información, el dron pueda avanzar, subir, bajar, etc, según lo haga la plataforma.

La última parte no está tan relacionada con el dron como las anteriores, ya que se basa en la automatización de la plataforma construida con anterioridad, de modo que se pueda disponer de un robot móvil, el cual haga un recorrido dentro de la superficie del simulador, y presente una estructura que pueda sostener al dron antes del despegue y después de su aterrizaje. Para ello se crea un robot simple, cuyo desplazamiento se programa dentro de V-REP y es independiente del dron, y que dispone de la misma plataforma que se utilizó en la etapa anterior para que el dron realice el seguimiento y pueda despegar y aterrizar en ella.

GLOSARIO DE ACRÓNIMOS Y ABREVIATURAS

GLOSARIO DE ACRÓNIMOS Y ABREVIATURAS.

V-REP: Virtual Robot Experimentation Platform.
ROS: Robot Operating System.
VANT: Vehículo Aéreo No Tripulado.
UAV: Unmanned Aerial Vehicle.
HD: High Definition.
3D: 3 dimensiones
MCU: Microcontrolador.
DSP: Digital Signal Processor.
FPGA: Field-Programable Gate Array.
ROI: Region of Interest.
GPU: Graphics Processing Unit.
API: Application Programming Interface.
PID: Proporcional, Integral, Derivada (referente a un sistema de control).
GPS: Global Position System.
FOH: First-Order-Hold.
Vx: Velocidad en el eje X.
Vy: Velocidad en el eje Y.
xr: Posición en el eje X de referencia.
yr: Posición en el eje Y de referencia.
xd: Posición deseada en el eje X.
yd: Posición deseada en el eje Y.
RGB: Red, Green, Blue (sistema de colores).
PC: Personal Computer.
RAM: Random Access Memory.

MEMORIA

1. INTRODUCCIÓN.

El propósito principal de este trabajo es integrar varias herramientas software para implementar un entorno de desarrollo de aplicaciones para drones. Estas herramientas son el simulador V-REP, el entorno de desarrollo ROS y la aplicación Matlab/Simulink. Una vez integradas estas herramientas, se desarrollarán varias aplicaciones de control de un dron: control absoluto de posición y control para el seguimiento visual de objetos utilizando una cámara vertical con el fin de poder integrarlo como apoyo para vehículos autónomos mediante su seguimiento.

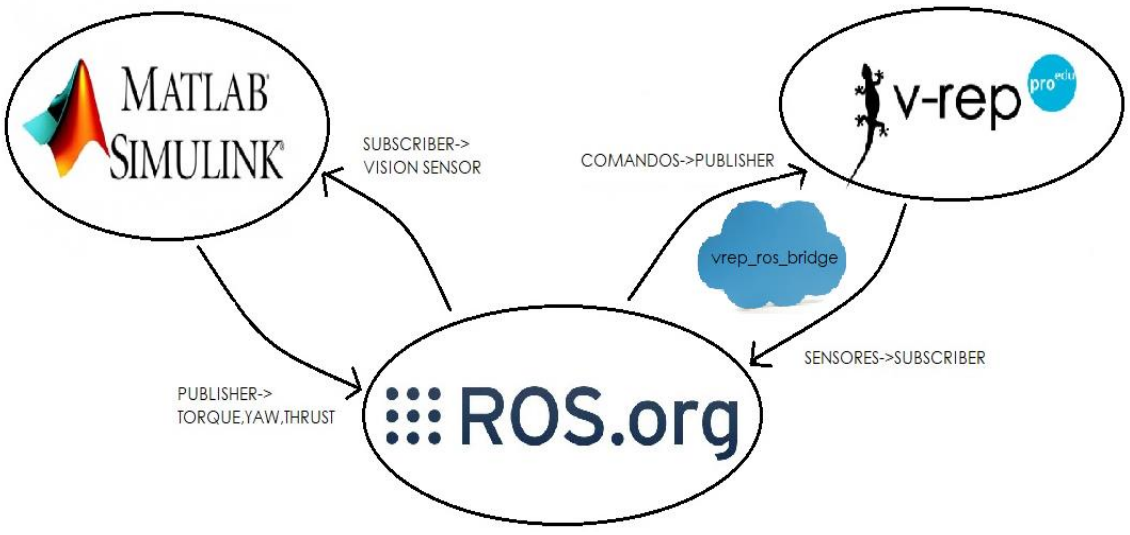


Figura 1.1. Relación ROS/V-REP/Matlab.

Los robots aéreos (drones) son actualmente un área de investigación en auge dentro de la robótica autónoma. Su control y el diseño de sistemas de navegación para los mismos presenta importantes retos respecto a los robots terrestres: el movimiento se produce en tres dimensiones, su dinámica es muy rápida, los procesadores y sensores a utilizar deben ser más ligeros para poder ser embarcados a bordo, y en consecuencia los algoritmos deben ser ágiles para poder ser ejecutados en tiempo real.

Por otro lado, la programación de aplicaciones robóticas ha sufrido en las últimas décadas una evolución hacia la estandarización, lo que ha hecho que proliferen diferentes entornos de desarrollo. Entre todos ellos destaca actualmente ROS (*Robot Operating System*), que ofrece funcionalidad a nivel de drivers para acceso a dispositivos robóticos, programación de aplicaciones, comunicación y sincronización entre procesos, herramientas de visualización, simuladores robóticos, etc. Además, la comunidad científica contribuye a esta herramienta mediante un completo repositorio de algoritmos para robótica.

Una de las principales ventajas de un entorno de programación robótica como ROS es que ofrece diferentes simuladores que permiten desarrollar y depurar las aplicaciones antes de trasladarlas a la plataforma robótica real. Esto se hace especialmente interesante al trabajar con robots aéreos, en los que la realización de pruebas reales puede ser tediosa y estar además dificultada por temas de legislación y permisos de vuelo. El simulador V-REP es muy intuitivo, aunque no se encuentra del todo integrado en ROS, por lo que hay que recurrir a paquetes de intercomunicación (*bridge*).

1.1. ESTADO DEL ARTE.

Un vehículo aéreo no tripulado (VANT), UAV en inglés (*Unnamed Aerial Vehicle*), o más comúnmente dron, es una aeronave que vuela sin tripulación, que puede ser controlada mediante radiofrecuencia o volar de forma autónoma. El diseño de estos dispositivos tiene una amplia variación en relación con su tamaño, configuraciones, características, etc.

La industria de los drones ha sido objeto de un gran crecimiento durante los últimos años. Además de un aumento de popularidad entre los consumidores, la utilización de estos dispositivos por compañías como Amazon o Google ha proporcionado una mayor visibilidad y les ha dotado de un gran futuro repleto de posibilidades, más allá del uso militar o científico.

Estos cuadricópteros han obtenido variados usos en todos los ámbitos, como puede ser el uso doméstico por puro entretenimiento, en eventos como medio de reportaje fotográfico, en el ámbito militar como medio de reconocimiento, en investigación, etc.

En este capítulo se abordará la situación actual en el apartado de la investigación ya que es lo que realmente mueve la realización de este proyecto, aunque todos los resultados obtenidos en un ámbito determinado son utilizados por los otros, completándose y apoyándose unos en otros.

Líneas de investigación.

En el ámbito de la investigación existen muchas corrientes diferentes de uso de drones, en las cuales, estos se utilizan con diferentes propósitos.

Una de las líneas es las que sigue [1] en su propuesta de proyecto, la cual persigue como fin la implementación de algoritmos que permitan la coordinación y el control para múltiples UAV's, pudiendo estos realizar de forma autónoma diversas tareas colectivas como pueden ser el vuelo de formación, vigilancia, búsqueda y rescate, persecuciones, además de la creación de un entorno apropiado para la prueba de todo ello.

Otra corriente de investigación se refleja en [2] cuyo objetivo principal es el mapeado de una zona universitaria mediante el uso de drones a modo de red o Network, de modo que entre ellos completen la información necesaria para conseguir el mapeado buscado, substituyendo otras técnicas antiguas. Siguiendo con esta línea, también existe otra investigación [3] que refuerza la importancia de los drones en el desarrollo de aplicaciones de este tipo.

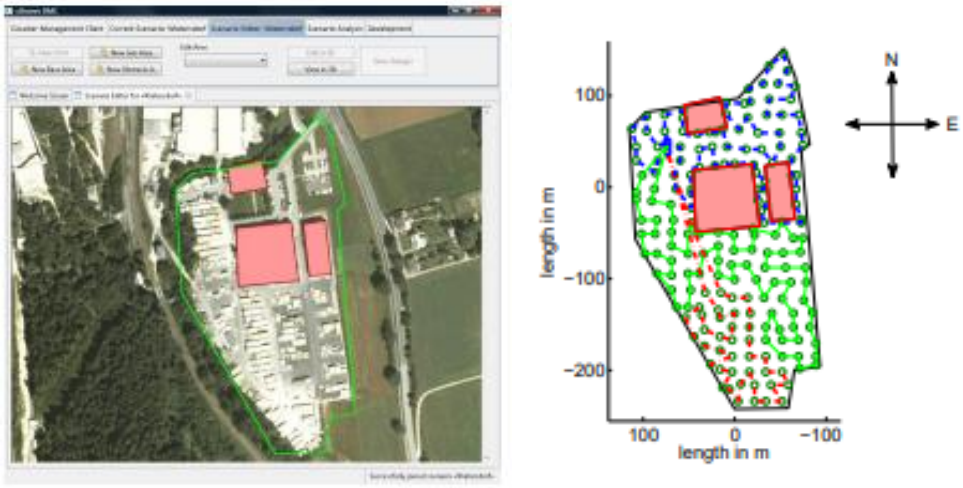


Figura 1.1.1. Mapeado de zona universitaria [3].

Otra utilidad de los drones se muestra en [4] donde se identifica, analiza, detalla y expone los posibles usos y beneficios que estos dispositivos podría aportar al campo de la prevención de riesgos laborales en la ejecución de trabajos que conllevan una innegable exposición a un riesgo.

Otra investigación más cercana a este proyecto es [5] dónde se ha creado una aplicación en la cual se dispone de un dron cuyo movimiento se basa en seguir ciertas referencias dentro del simulador, de manera que el comportamiento del dron está determinado por la posición y orientación de las referencias marcadas en el simulador. Dicha aplicación va a ser muy útil a lo largo del proyecto.

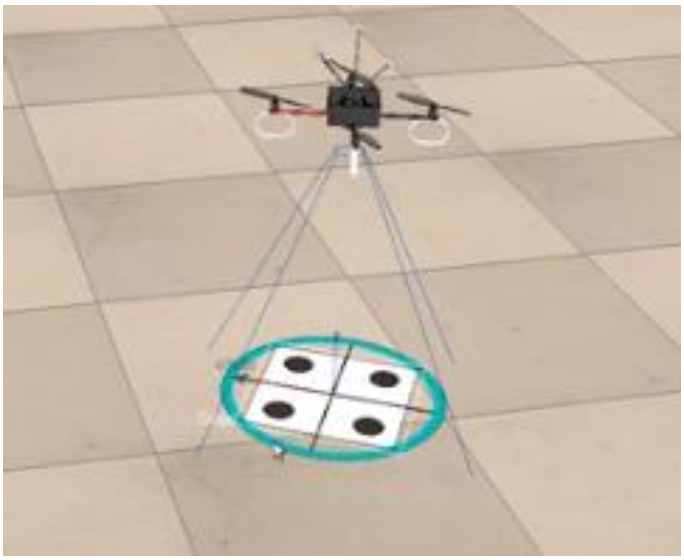


Figura 1.1.2. Escena de V-REP [5].

A su vez, en [6] también se utiliza un cuadricóptero siguiendo la temática de detección de referencias, solo que esta vez se ha dado el salto a la implementación física. Este proyecto se basa en la detección de un determinado objeto y rectificación de la posición del dron en función de dónde se encuentre ese objeto, solamente a partir de la información visual que pueda recibir el dron mediante una cámara.

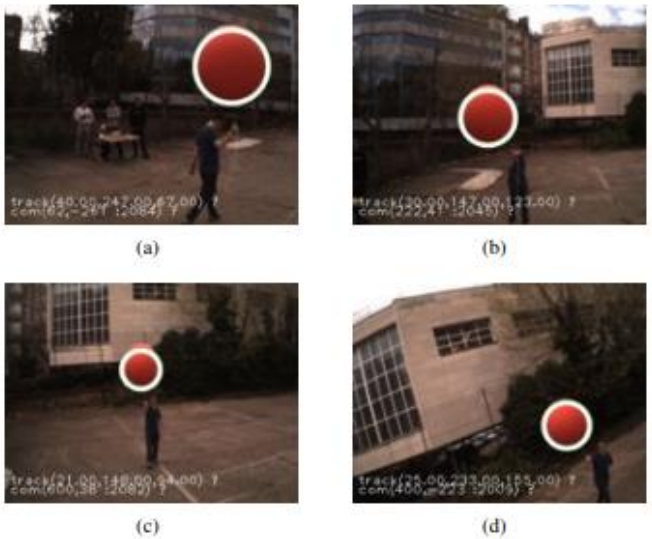


Figura 1.1.3. Imagen de detección [6].

Todo ello crea un contexto adecuado para la elaboración de proyectos de este tipo, ya que las posibilidades son muy grandes y el atractivo de estos dispositivos es aún mayor. En la actualidad, estos dispositivos son vistos como un posible campo de investigación muy amplio para la realización de determinadas tareas en muchos sectores, dándole muchas posibilidades a su expansión y desarrollo.

Herramientas hardware.

A pesar de que en este proyecto no se va a implementar el sistema de forma física o real, es importante contextualizar el estado de los drones comerciales disponibles para la implementación de estos estudios de investigación. Esta contextualización también es importante a la hora de elegir un dron para la simulación, ya que, si en un futuro se quiere llevar una línea continuista con el proyecto, y dar el salto a la implementación real de lo que se va a exponer a continuación, tiene sentido que el dispositivo elegido sea el utilizado en la simulación (aunque no es un factor de decisión principal).

Por tanto, en este apartado se van a mostrar algunos de los drones más utilizados en el campo de la investigación, debido en su mayor parte, a la posibilidad de programar su comportamiento, y no solo al mero hecho de poder utilizarlo a través de una aplicación o un control remoto.

Algunos de estos drones son:

- Phantom de DJI.

Este dron de la compañía DJI, a diferencia de otros, no incluye una cámara, por lo que el usuario tiene la libertad de incorporar la cámara que le resulte más útil para cada uso.

Figura 1.1.4. Phantom Drone.

Este dispositivo controlado mediante radiocontrol incorpora una plataforma, llamada Naza-M, que contiene un microcontrolador y sensores necesarios para un dron como

son un barómetro, un giróscopo, un acelerómetro, formando en conjunto la IMU del dron.

Algunas especificaciones importantes del *Phantom* son: la frecuencia de trabajo es de 2.4 GHz, con 7 canales y una distancia máxima de comunicación de 1000 m; la velocidad máxima de ascenso/descenso es de 6 m/s y la de vuelo de 10 m/s; puede llevar un peso máximo de 1200 g siendo su autonomía de 10 a 15 minutos dependiendo del peso que lleve de carga.

- AR. Drone 2.0 de Parrot.

También fabricado por Parrot en 2010 (ya no está disponible su venta en la propia compañía, pero si en otras plataformas como Amazon) tiene la característica de su estructura modular, de modo que es fácil el intercambio de piezas y su montaje.

Figura 1.1.5. AR. Drone 2.0.

Dispone de una aplicación para dispositivos móviles a través de la cual se puede ver lo que recoge su cámara HD, así como pilotarlo desde ella. Contiene un procesador *ARM Cortex A8* con sistema operativo Linux 2.6.32. A ello, habría que añadirle un giroscopio, un acelerómetro, un magnetómetro, un sensor de presión y un sensor de ultrasonidos.
Este quadricóptero dispone de unas características adecuadas para su uso en el ámbito de la investigación.

- AscTec Hummingbird de Ascencing Technologies.

Creado por la compañía *Ascencing Technologies,* la cual colabora con *Intel*, de manera que dicho dispositivo contiene un ordenador de abordo *Intel Atom Processor Z530*. Este es el dron sobre el que se va a basar el proyecto, obteniendo su modelo en V-REP para poder trabajar con él.

Figura 1.1.6. AscTec Hummingbrid.

La comunicación se produce a 2.4 GHz, con una autonomía de 20 minutos, y unas velocidades características de 5 m/s para el ascenso y descenso, y 15 m/s de velocidad de vuelo. Las características más detalladas se muestran a continuación:

Modelo	AscTec Hummingbird
Manufacturación	Ascending Technologies GmbH
Peso de despegue	480 g
Distancia entre ejes de motor	34 cm
Control de vuelo e IMU	AscTec Autopilot
Hélices	8'' hélices grises estándar flexibles
Motores	AscTec X-BL 52s
Controlador de motores	AscTec X-BLDC
Empuje por motor	0.05 3.5 N
Sistema de radiocontrol	Futaba Fasst 2.4 GHz
Sistema de telemetría	Xbee 2.4 GHz
Momento de inercia	$Ixx = Iyy = 5.6*10^{-3}$ kg*m; $Izz = 8.1*10^{-3}$ kg*m

Tabla 1.1.1. Características AscTec Hummingbrid.

Herramientas Software.

Otro de los apartados importantes a tener en cuenta es la elección de los sistemas software que se quieren o necesitan utilizar en el proyecto, ya que existen infinidad de plataformas, cada una más completa que la anterior.

Para el caso de los robots, ya sean aéreos, terrestres, o de cualquier tipo, el entorno de desarrollo más extendido para su control e implementación de cualquier sistema es ROS. Este entorno es utilizado por la comunidad de robótica prácticamente en su totalidad (sus características se explicarán más a fondo a continuación). Una de sus ventajas es el trabajo en comunidad de los usuarios, ya que todos aportan significativamente su conocimiento para su desarrollo e implementación.

ROS tiene incorporado un simulador robótico llamado Gazebo. Es un simulador multirobot en 3D completo con física dinámica y cinemática, y un motor de física conectable. La integración entre ROS y Gazebo es proporcionada por un conjunto de complementos de Gazebo que admiten muchos robots y sensores existentes. Debido a que los complementos presentan la misma interfaz de mensajes que el resto del ecosistema ROS, puede escribir nodos ROS que sean compatibles con la simulación, los datos registrados y el hardware. El desarrollo de la aplicación en simulación, y su posterior implementación en hardware implica pocos o ningún cambio en el código. Por otra parte, al realizar simulaciones amplias, se producen de forma muy lenta.

La elección de V-REP como simulador para la realización del proyecto se debe que el trabajo en él es bastante más intuitivo que en Gazebo, siendo muy ventajoso a la hora de trabajar. En contraposición a esto, su integración en ROS está menos desarrollada, por lo que conlleva cierto trabajo la correcta conexión entre ambos.

1.2. OBJETIVOS DEL PROYECTO.

Con esta contextualización de la situación de los drones en la actualidad, el proyecto que se va a realizar es muy atractivo. El principal objetivo del mismo es la creación de una estructura de simulación de vuelo de drones con el fin de utilizarla para la programación de una aplicación de seguimiento de robots de forma autónoma.

Para la realización de este proyecto se va a diferenciar una serie de objetivos parciales para lograr alcanzar el objetivo final descrito anteriormente. Esta serie de objetivos son los siguientes:

- **Estudio del entorno de desarrollo ROS, del simulador V-REP y del entorno Matlab/Simulink:** se deberán obtener unos conocimientos básicos sobre las plataformas a utilizar durante el proyecto puesto que el control del dron se basará en Simulink, la simulación en V-REP y la comunicación entre ambos entornos en ROS.

- **Creación de un entorno de simulación en V-REP:** con lo aprendido anteriormente, se diseñará una escena en el simulador, la cual contenga al dron que se utilizará a lo largo del trabajo. Dicha escena, además contendrá otros elementos como plataformas móviles y robots.

- **Integración de ROS en V-REP:** una de las partes más complicadas del proceso será lograr comunicar de una manera útil el simulador con ROS. Esto se realizará a través de un paquete facilitado por ROS llamado *vrep_ros_bridge*. Este paquete proporcionará lo necesario para la comunicación V-REP/ROS.

- **Conexión entre V-REP/ROS/Matlab:** para la realización del sistema de control del dron, se utiliza Matlab/Simulink, utilizando ROS como puente entre Matlab y V-REP. De esta manera, los *topics* disponibles en ROS que llevan la información de V-REP, o que contienen la información necesaria para que en V-REP se actúe sobre el dron, son obtenidos en Simulink mediante la *toolbox Robotics System Toolbox* [4] disponible en Simulink.

- **Creación de un primer modelo de control de posición mediante coordenadas absolutas:** una primera aproximación al sistema que se desea crear será lograr controlar el cuadricóptero haciendo uso de los *topics* de ROS, de manera que se desplace por la escena de manera controlada y conocida.

- **Creación de un segundo modelo de control basado en adquisición de datos a través de una imagen:** para obtener datos de posición de los objetos de la escena, se añadirá un sensor de visión o cámara al dron, a partir del cual se conseguirán datos del movimiento que realicen los objetos y así poder actuar en consecuencia. En este modelo se utilizará una plataforma móvil, de forma manual a través del simulador, con forma rectangular para simular la vista aérea de un coche, de forma que el dron siga los movimientos de la plataforma.

- **Automatización de la plataforma:** esta parte del proyecto se basará en conseguir que un robot móvil se desplace por la escena, para así poder comprobar el correcto funcionamiento del sistema anterior, sin necesidad de mover de forma manual el objeto de la escena. Además, se añadirá una plataforma a dicho robot, que hará las veces de soporte para despegue y aterrizaje del dron.

1.3. ESTRUCTURA DEL DOCUMENTO.

Este documento tiene diferentes partes diferenciadas, con el fin de entender lo realizado en el proyecto:

- La primera parte del documento consta de una primera parte introductoria que pone en contexto la situación de los drones en la actualidad.

- La siguiente parte trata de explicar las herramientas software utilizadas y sus características más importantes. Se explica de forma detallada el entorno de desarrollo ROS, el simulador V-REP y el software matemático Matlab/Simulink.

- A continuación, se presenta el grueso del trabajo, en el cual se explica de forma detallada el proceso que se ha seguido en la realización del proyecto. Contiene una parte introductoria a modo de explicación genérica y posteriormente se diferencian las tres aplicaciones realizadas: control absoluto de posición, control para el seguimiento visual de un objeto y seguimiento de un robot móvil. Este punto contiene la explicación de los controladores del dron para las tres aplicaciones mencionadas anteriormente.

- Finalmente, la última parte del documento se forma con los planos de diagramas de los sistemas de control, un pliego de condiciones necesarias para el funcionamiento de las aplicaciones, el presupuesto del proyecto, un manual de instalación y lanzamiento de las aplicaciones y la bibliografía utilizada en el proyecto.

2. HERRAMIENTAS UTILIZADAS.

2.1. ROS (ROBOT OPERATING SYSTEM).

2.1.1. ASPECTOS BÁSICOS DE ROS.

ROS [10] es un sistema flexible para la elaboración de software robótico. Se basa en un conjunto de herramientas, librerías y convenciones que simplifican la labor de crear software complejo y robusto para robótica y que pueda ser implementado en un gran número de plataformas.

Desde el punto de vista del robot, los problemas que pueden parecer triviales para los seres humanos a menudo varían enormemente entre instancias de tareas y entornos. Lidiar con estas variaciones es difícil para que un individuo, laboratorio o institución trabaje de forma autónoma.

ROS es un sistema ``*Open Source´´*, construido desde cero para incrementar el desarrollo de software de robótica colaborativa. Su idea inicial era que cada grupo de expertos que trabajase con *ROS* pudiera aportar sus avances software y que pudieran ser utilizados por cualquier otro en el mundo.

Por ello, fue diseñado de la forma más modular posible, de modo que los usuarios pudiesen elegir qué partes utilizar en función de las necesidades de su aplicación.

Su estructura hace que se trate de un tipo de sistema operativo, ya que proporciona servicios tales como comunicación a través de mensajes entre procesos, mantenimiento de paquetes, control de dispositivos de bajo nivel, abstracción de hardware e implementación de funcionalidad. Se habla de que trabaja como un sistema operativo, pero no lo es, ya que se instala sobre otro (generalmente Linux), por lo que también recibe el nombre de Meta-Sistema Operativo.

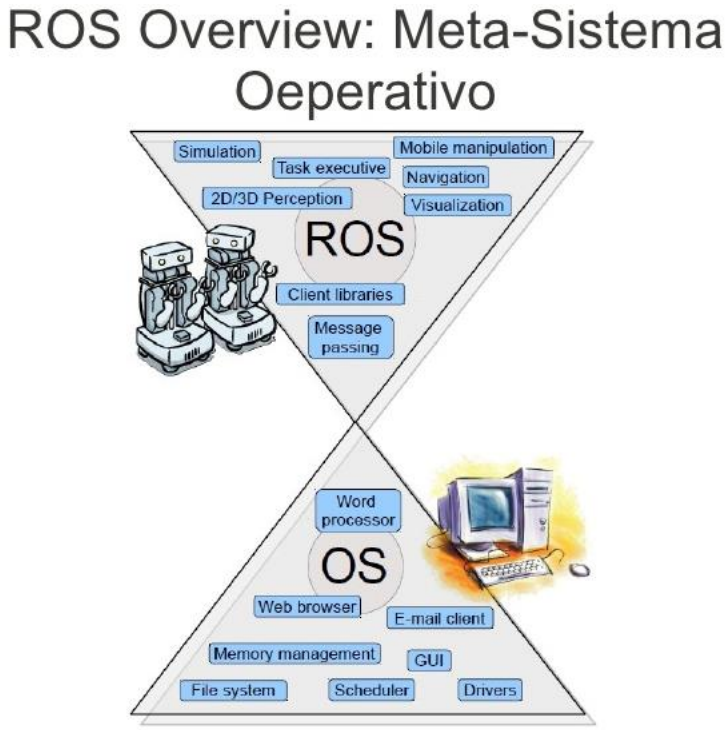


Figura 2.1.1. Vista general de ROS.

Por sí mismo ofrece muchos proyectos de robótica, pero también presenta la oportunidad establecer contactos y colaborar con los ingenieros de clase mundial que forman parte de la comunidad de ROS. Esta comunidad de usuarios se basa en una infraestructura común para proporcionar un punto de integración que ofrece acceso a controladores hardware, capacidades de robot genéricas, herramientas de desarrollo, bibliotecas externas útiles, etc.

2.1.2. HERRAMIENTAS BÁSICAS DE ROS.

- **Distribuciones:** Lo primero que se tiene que tener en cuenta al elegir ROS, es la gran variedad de distribuciones de las que dispone. Una distribución es una versión determinada de paquetes de ROS. El propósito de estas distribuciones es permitir a los desarrolladores trabajar sobre una base de código relativamente estable hasta que estén listos para llevar a cabo su proyecto.

Una vez que las distribuciones son lanzadas, se intentan solucionar los pequeños fallos que tienen sin alterar los paquetes principales, de modo que el trabajo que haya sido desarrollado hasta entonces por los usuarios, no se vea afectado o lo haga de la menor forma posible.

Las distribuciones disponibles son las siguientes:

Distro	Release date	Poster	*Tuturtle*, turtle in tutorial	EOL date
ROS Melodic Morenia	May 23rd, 2018			May, 2023 (Bionic EOL)
ROS Lunar Loggerhead	May 23rd, 2017			May, 2019
ROS Kinetic Kame (**Recommended**)	May 23rd, 2016			April, 2021 (Xenial EOL)
ROS Jade Turtle	May 23rd, 2015			May, 2017
ROS Indigo Igloo	July 22nd, 2014			April, 2019 (Trusty EOL)
ROS Hydro Medusa	September 4th, 2013			May, 2015
ROS Groovy Galapagos	December 31, 2012			July, 2014
ROS Fuerte Turtle	April 23, 2012			--
ROS Electric Emys	August 30, 2011			--

Figura 2.1.2. Distribuciones de ROS.

- **Paquetes (*packages*):** Los paquetes son la unidad principal para la organización de software en *ROS*. Un paquete puede contener procesos de tiempo de ejecución ROS (nodos), una librería dependiente de ROS, conjuntos de datos, archivos de configuración, o cualquier cosa que sea útil organizar. El paquete es el elemento de construcción y lanzamiento más atómico de ROS.

- **Pilas (*stacks*):** Es el conjunto de paquetes que tienen como objetivo la organización de uno o varios proyectos para llevar a cabo funciones de alto nivel.

- **Nodos (*nodes*):** Los nodos son los procesos que realizan los cálculos como tal. ROS está diseñado para ser modular; un sistema de control robótico generalmente comprende muchos nodos (por ejemplo: un nodo controla un láser, otro nodo el movimiento del robot, otro realiza la localización, etc). Un nodo *ROS* se define utilizando una librería cliente, como puede ser *roscpp (C++)* o *rospy (Python).*

- **Maestro (*master*):** Es el nodo principal. Este nodo proporciona el registro de nombres y la vista del resto del nivel computacional gráfico. Sin este nodo, el resto no sería capaz de comunicarse entre sí.

- **Servidor de parámetros (*parameter server*):** Permite el almacenamiento de datos en una localización central.

- **Mensajes (*messages*):** Los nodos se comunican unos con otros mediante el paso de mensajes. Un mensaje es una simple estructura de datos la cual puede contener tipos de datos primitivos o estructuras arbitrariamente anidadas.

- **Temas (*topics*):** Los mensajes se enrutan a través de un sistema de transporte con semántica publicación/suscripción. Un nodo envía información publicando un *topic*, y la recibe suscribiéndose a él.

- **Servicios (*services*):** Los *topics* no permiten una comunicación petición/respuesta. Este tipo de comunicación viene dada por los servicios, que vienen definidos por un par de estructuras de mensaje: una para la petición, y otra para la respuesta.

- **Bolsas (*bags*):** Un *bag* es un formato especial que permite al usuario guardar y reproducir de nuevo mensajes de datos. Son importantes para el almacenamiento de datos, como pudieran ser los datos de un sensor.

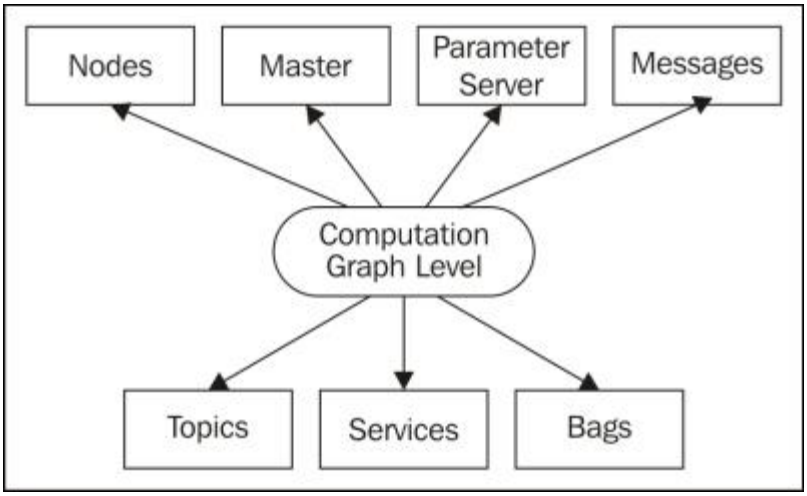


Figura 2.1.3. Herramientas disponibles en ROS.

2.1.3. COMANDOS ÚTILES EN ROS.

ROS se ejecuta desde la consola de comandos del sistema operativo (Linux en este caso), por lo que es importante conocer los comandos básicos y más utilizados en este entorno de desarrollo:

- ***roscore*:** Es la herramienta que ejecuta el núcleo de ROS. Debe haber un *roscore* corriendo para que los nodos de ROS puedan comunicarse. Se lanza utilizando el comando *roscore*.

- ***roscreate*-pkg:** Este comando crea un nuevo paquete ROS con archivos de paquetes comunes. Aborda el problema de la creación de paquetes utilizando paquetes preexistentes. Se lanza con el comando *roscreate-pkg [nombre del paquete] [dependencias].*

- ***rospack*:** Es un comando a través del cual se recibe información de un determinado paquete. Su llamada se realiza con el comando *rospack [comando] [paquete].*

- ***rosstack*:** Este comando devuelve información de una pila deseada con *rosstack [comando] [stack].*

- ***roscd*:** Permite acceder a un paquete mediante *roscd [paquete].*

- ***rosls*:** Permite al usuario ver el contenido de un paquete: *rosls [paquete].*

- ***rosnode*:** Al introducir el comando *rosnode [comando] [nodo]* se obtienen los nodos en ejecución, la información de un nodo y también se pueden eliminar nodos de la ejecución.

- ***rosdep*:** Permite descargar dependencias de paquetes con *rosdep install [paquete].*

- ***rostopic*:** El comando *rostopic* permite listar información de depuración de los *topics*, incluidos *publishers*, *subscribers*, mensajes de ROS, y más. Las posibilidades de este *topic* se muestran a continuación:

```
rostopic bw     display bandwidth used by topic
rostopic delay display delay for topic which has header
rostopic echo   print messages to screen
rostopic find   find topics by type
rostopic hz     display publishing rate of topic
rostopic info   print information about active topic
rostopic list   print information about active topics
rostopic pub    publish data to topic
rostopic type   print topic type
```

Figura 2.1.4. Opciones del comando rostopic.

Este comando se envía con la siguiente estructura: *rostopic [comando] [topic] [argumentos].*

- ***rosservice*:** Tiene la misma utilidad y estructura que *rostopic* pero para los servicios: *rosservice [comando] [servicio] [argumentos].*

- ***rosmsg*:** Este comando obtiene la información de un tipo de mensaje: *rosmsg [comando] [mensaje].*

- ***rosrun*:** Permite la ejecución de un archivo ejecutable dentro de un paquete específico mediante la estructura de comando *rosrun [paquete] [ejecutable].*

- ***roslaunch*:** Es una herramienta sencilla de lanzamiento de múltiples nodos ROS de manera local o remota via *SSH*. Los archivos con extensión *.launch* pueden configurarse con varios nodos pudiendo ejecutarse a la vez facilitando al usuario el uso global de ROS.

2.1.4. HERRAMIENTAS DE ROS.

- RVIZ.

RVIZ es un visualizador 3D para mostrar datos de sensores e información de estado de ROS. Se puede visualizar representaciones en vivo de los valores de los sensores que aparecen en los *topics* de ROS, incluidos datos de cámara, mediciones de distancias, datos de sensores, etc.

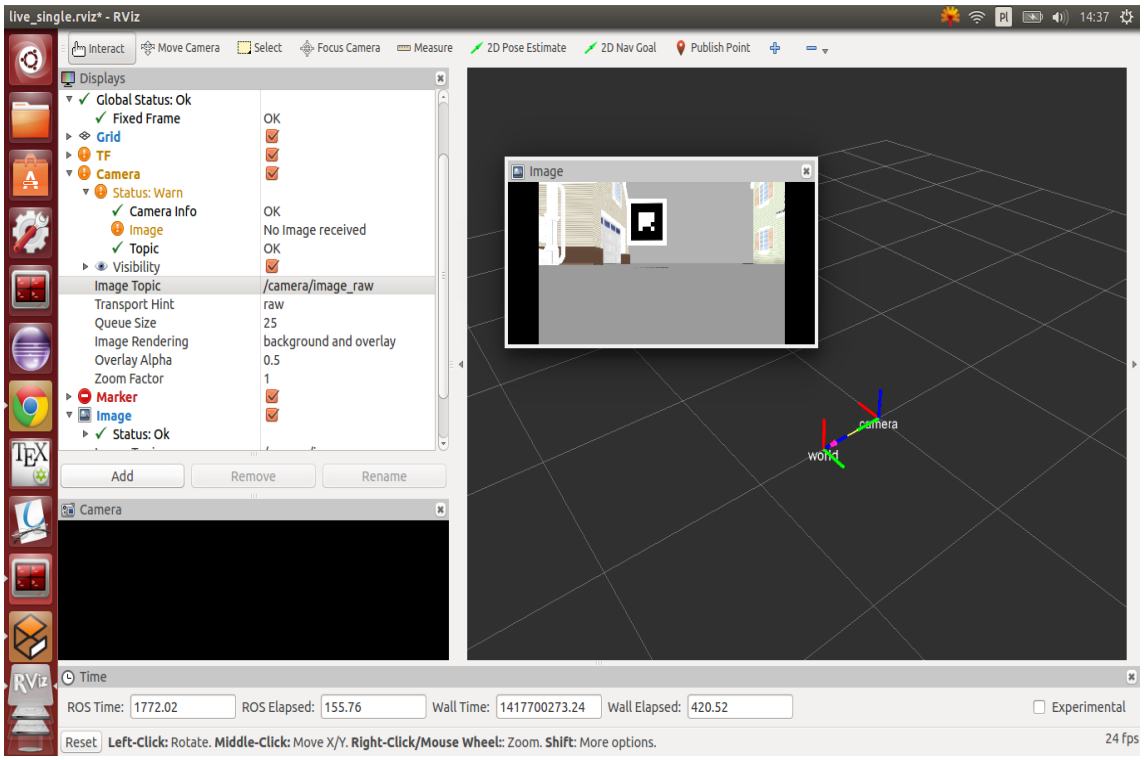


Figura 2.1.5. Imagen de RVIZ.

- RQT.

RQT es un sistema software de ROS que implementa las diversas herramientas GUI en forma de complementos. RQT permite un manejo más sencillo de todas las ventanas en pantalla en un momento dado.

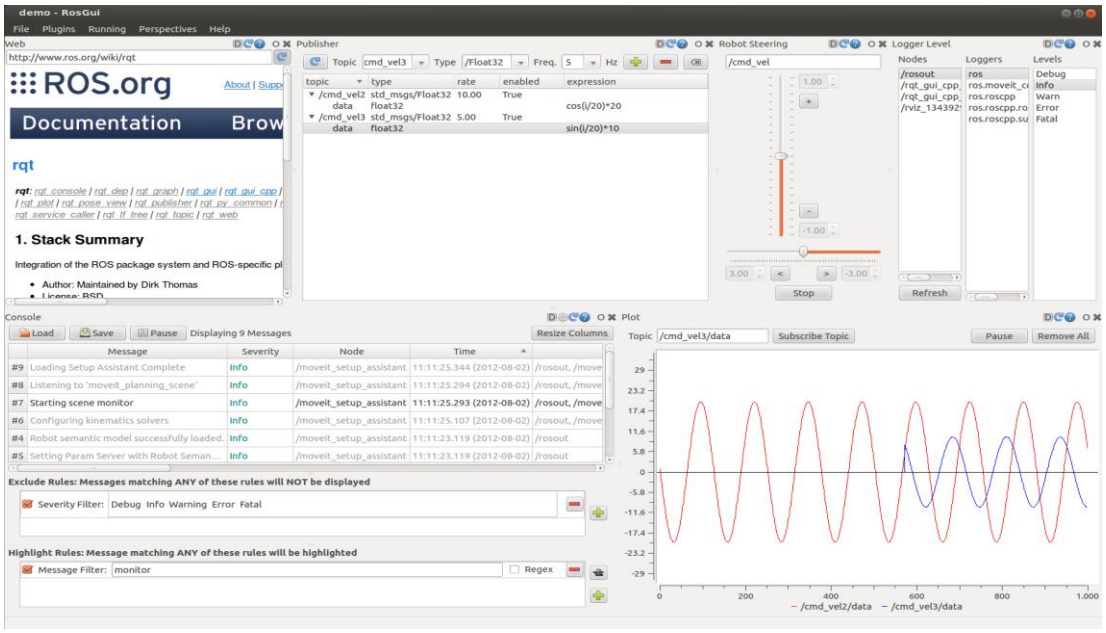


Figura 2.1.6. Imagen de RQT.

2.1.5. PAQUETE *vrep_ros_bridge.*

Este paquete de ROS [11] es uno de los principales apoyos en los que se sustenta este proyecto, debido a que implementa el puente entre V-REP y ROS, y además incluye un ejemplo de control de un dron a partir del cual se va a desarrollar el trabajo.

Este paquete contiene un *plugin*, el cual es cargado por V-REP, y que proporciona la posibilidad de comunicación entre el simulador y ROS. Este plugin crea *subscribers* y *publishers* y mediante estos *topics*, se realiza la comunicación del lazo de control entre todas las plataformas.

El uso y modificación de dicho *plugin* no dispone de ningún tipo de información, por lo que es complicado poder recompilar los archivos que forman dicho plugin para adaptarlo a las necesidades del proyecto. Aun así, resulta de mucha utilidad, ya que contiene la declaración de un *topic* llamado *effort*, el cual manda 4 parámetros a V-REP a través de ROS para modificar la posición del dron en el simulador. Por lo tanto, se adaptará el desarrollo del trabajo a dicho *topic*, estudiando cómo afectan esos 4 parámetros al cuadricóptero.

2.2. MATLAB/SIMULINK.

Matlab [12] es una herramienta de software matemático que ofrece un entorno de desarrollo integrado con un lenguaje de programación propio, disponible para las plataformas Unix, Windows, Mac OS X y GNU/Linux.

Matlab integra análisis numérico, cálculo matricial, procesamiento de señales, gráficos, etc, en un entorno fácil de usar, donde los problemas y las soluciones son expresados como se escriben matemáticamente, sin la programación tradicional. Es un sistema interactivo cuyo elemento básico de datos es una matriz que no requiere dimensionamiento. Esto permite resolver muchos problemas numéricos en una fracción del tiempo que le llevaría hacerlo en otros lenguajes como pueden ser *C*, *C++*, *BASIC* o *FORTRAN*.

Este entorno ha evolucionado a lo largo de los años a partir de la colaboración de muchos usuarios:

- En entornos universitarios se ha convertido en la herramienta de enseñanza estándar para cursos de introducción en álgebra lineal, así como en cursos avanzados en otras áreas y especialidades.
- En industria, Matlab se utiliza para la investigación y para resolver problemas prácticos de ingeniería y matemáticas, con un gran énfasis en aplicaciones de control y procesamiento de señales.

Matlab también proporciona una serie de soluciones específicas llamadas *TOOLBOXES*. Estas son muy importantes para la mayoría de los usuarios de Matlab y son conjuntos de funciones que extienden el entorno para resolver clases particulares de problemas como procesamiento de señales, diseño de sistemas de control, simulaciones de sistemas dinámicos, identificación de sistemas, redes neuronales y muchos más.

En resumen, las prestaciones más importantes de Matlab son:

- Escritura del programa en lenguaje matemático.
- Implementaciones de matrices como elemento básico del leguaje, lo que permite una gran reducción del código, al no necesitar implementar el cálculo matricial.
- Implementación de aritmética compleja.
- Un gran contenido de funciones específicas, agrupadas en las *TOOLBOXES.*
- Posibilidad de adaptar y ampliar el lenguaje mediante ficheros de *script* y funciones en lenguaje Matlab con extensión .m.

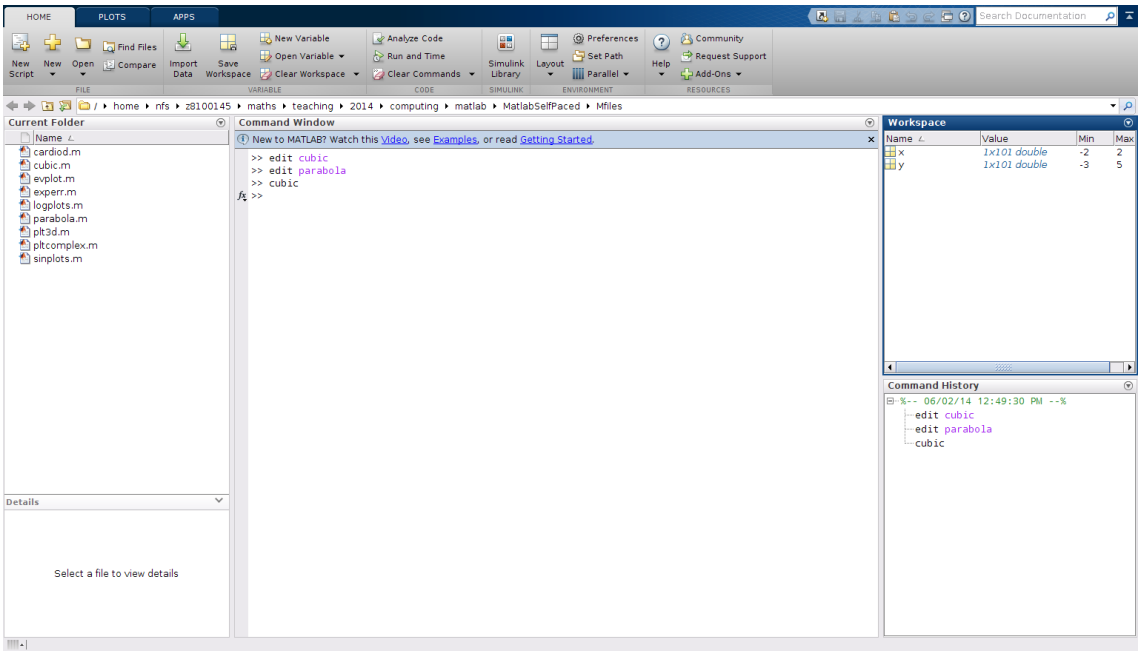


Figura 2.2.1. Imagen de la ventana principal de Matlab.

2.2.1. APLICACIONES DE MATLAB.

El uso que se puede dar a este entorno es muy amplio, variado y potente. Como ejemplo de ello, estas son algunas de sus aplicaciones más extendidas y relacionadas con este proyecto:

Visión artificial.

El desarrollo del algoritmo es fundamental para el procesamiento de imágenes y la visión artificial porque cada situación es única y las buenas soluciones requieren múltiples iteraciones de diseño. *Mathworks* (compañía creadora de Matlab) proporciona un entorno integral para obtener información sobre la imagen y datos de video, desarrollar algoritmos y explorar intercambios de implementación.

Procesamiento de señales.

El procesamiento de señal es algo esencial para un rango amplio de aplicaciones, desde la ciencia de los datos a sistemas embebidos de tiempo real. Matlab hace que sea más sencillo utilizar técnicas de procesamiento de señal para explorar y analizar daros de series de tiempo, y proporciona un flujo de trabajo unificado para el desarrollo de sistemas integrados y aplicaciones de transmisión.

Robótica.

Los desarrolladores robóticos utilizan Matlab para diseñar y sintonizar algoritmos, modelar sistemas reales, y generar código automatizado, todo desde un único entorno de desarrollo.

Sistemas de control.

Las herramientas para el diseño de sistemas de control respaldan cada etapa del proceso de desarrollo, desde el modelado de la planta hasta la implementación mediante generación automática de código. Su adopción generalizada entre los ingenieros de control de todo el mundo proviene de la flexibilidad de las herramientas para adaptarse a diferentes tipos de problemas de control.

Simulink.

Simulink es una *toolbox* especial de Matlab que sirve para simular el comportamiento de los sistemas dinámicos. Puede simular sistemas lineales y no lineales, modelos en tiempo continuo y tiempo discreto, así como sistemas híbridos de todos los anteriores. Es un entorno gráfico en el cual el modelo a simular se construye clicando y arrastrando los bloques que lo constituyen. Los modelos creados por Simulink tienen extensión *.mdl* o *.slx*.

El software de simulación ayuda a predecir el comportamiento de un sistema, se puede utilizar para evaluar un nuevo diseño, diagnosticar problemas de un diseño existente y probar un sistema en condiciones que son difíciles de reproducir, como por ejemplo un satélite en el espacio exterior. Para ejecutar una simulación se necesita un modelo matemático del sistema, que se puede expresar como un diagrama de bloques, un esquema, un diagrama de estados, incluso código. Este software de simulación calcula el comportamiento del modelo a medida que la condiciones evolucionan con el tiempo o a medida que se producen eventos. El software también incluye herramientas de visualización, tales como *displays* y animaciones 3D, para contribuir a supervisar la simulación a medida que se ejecuta.

El uso de Simulink se extiende debido a que éste permite:

- Explorar un amplio espacio de diseño mediante el modelado del sistema y de la planta física. El equipo al completo puede utilizar un solo entorno multi-dominio para simular el comportamiento de todas las partes del sistema.
- Reducir los prototipos caros mediante la realización de pruebas del sistema en condiciones que, en otras circunstancias, serían demasiado peligrosas o lentas.
- Validar el diseño con pruebas hardware *in-the-loop* y crear prototipos de forma rápida.
- Generar miles de líneas de código *C* y *HDL* automáticamente de calidad que se comporte igual que el modelo creado en Simulink para desplegarlo directamente en *MCU, DSP* o *FPGA.*

Simulink dispone de distintos tipos de simulación, como son las basadas en el tiempo, en eventos o en sistemas físicos reales. Para cada uno de estos tipos se tienen diferentes herramientas para su implementación.

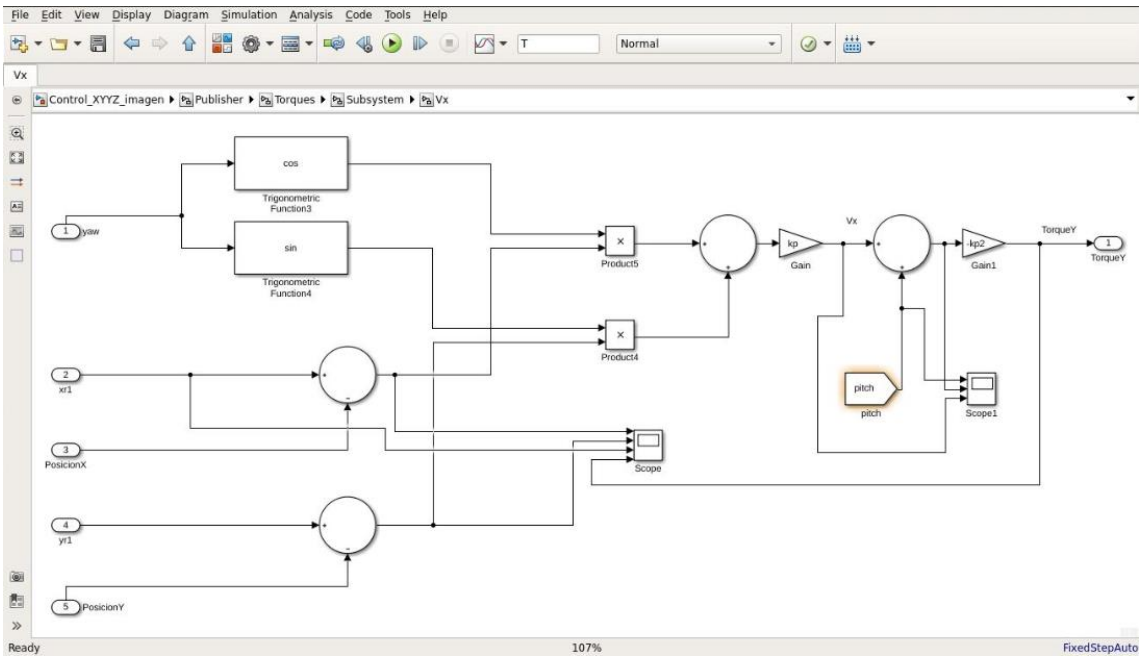


Figura 2.2.2. Imagen ejemplo de un modelo de bloques en Simulink.

Otra de las grandes ventajas de Simulink es que trabaja juntamente con Matlab, es decir, se combina la programación textual y gráfica para diseñar el sistema en un entorno de simulación. Se pueden utilizar los algoritmos existentes en Matlab, incluyendo su código en un bloque de Simulink; además, se pueden crear conjuntos de datos de entrada que pongan en marcha la simulación, ejecutar varias simulaciones en paralelo y visualizar resultados en Matlab, etc.

2.2.2. TOOLBOXES.

Las toolboxes de Matlab/Simulink son un conjunto de funciones y algoritmos agrupados según su funcionamiento. Las dos toolboxes que se van a utilizar en este proyecto y que están relacionadas con él son:

Robotics System Toolbox.

Robotics System Toolbox [13] está contenida en Matlab y proporciona algoritmos y conectividad hardware para el desarrollo de aplicaciones robóticas autónomas para vehículos aéreos y terrestres. Estos algoritmos incluyen patrones de planificación y de seguimiento para distintos robots de unidad diferencial, *scan matching*, prevención de obstáculos y estimación de estados.

Esta *toolbox* proporciona una interfaz entre Matlab, Simulink y ROS, que permite verificar y probar las aplicaciones; sostiene generación de código C++, permitiendo la generación de nodos de ROS desde un modelo de Simulink y desplegándolos de forma automática en una red de ROS.

Se pueden obtener datos de sensores o de cualquier robot que soporte ROS o desde un simulador en Simulink o Matlab para su visualización, exploración, identificación de sistemas, calibración, etc. Del mismo modo, se pueden mandar actuadores desde Matlab/Simulink a los diferentes sistemas que lo forman.

Image Processing Toolbox.

Image Processing Toolbox [14] proporciona un conjunto completo de algoritmos estándar de referencia y aplicaciones de flujo de trabajo para el procesamiento, el análisis y la visualización de imágenes, así como para el desarrollo de algoritmos. Puede llevar a cabo segmentación de imágenes, mejora de imágenes, reducción de ruido, transformaciones geométricas, registro de imágenes y procesamiento de imágenes 3D.

Las aplicaciones de *Image Processing Toolbox* le permiten automatizar los flujos de trabajo habituales de procesamiento de imágenes. Puede segmentar datos de imagen, comparar técnicas de registro de imágenes y procesar por lotes conjuntos de datos extensos de forma interactiva. Las aplicaciones y las funciones de visualización le permiten explorar imágenes, volúmenes 3D y vídeos, ajustar el contraste, crear histogramas y manipular regiones de interés (ROIs).

Puede acelerar los algoritmos mediante su ejecución en procesadores multinúcleo y GPUs. Muchas de las funciones de esta *toolbox* soportan la generación de código C/C++ para el prototipado y el desarrollo de sistemas de visión embebidos.

2.3. V-REP (Virtual Robot Experimental Platform).

V-REP [15] es un producto desarrollado por *Coppelia Robotics* cuyo propósito de desarrollo es la simulación robótica. Una interfaz de usurario personalizada y una estructura modular integrada al entorno de desarrollo son las principales características del simulador. La modularidad es de alto nivel tanto para la simulación de objetos, como para los métodos de control. El fácil uso del entorno de desarrollo que incorpora el simulador proporciona y facilita la creación de robots y los casos de simulación. Esta capacidad permite un prototipado, diseño de algoritmos e implementación rápidos. Durante la simulación, esta área actúa como un mundo real 3D y proporciona una realimentación en tiempo real acorde al comportamiento de los modelos. El objeto que compone la escena, el mecanismo de control y los módulos de computación son las tres funcionalidades del simulador.

Existen varios métodos de control para manejar el comportamiento de cada objeto en la simulación. Estos controladores pueden ser implementados tanto dentro de la simulación, como fuera del entorno.

El principal mecanismo de control interno se realiza utilizando *child scripts*, los cuales pueden ser asociados a cualquier elemento de la escena. Estos scripts secundarios manejan una cierta parte de la simulación, y son parte fundamental de su objeto asociado; esto permite que puedan ser duplicados, puestos en serie, etc.

Estos scripts pueden ser sin hilos (ejecutan una función y retorna el control del programa) o con hilos (son lanzados por el programa principal y trabajan en paralelo).

Los scripts abren y manejan líneas de comunicación con *API's* remotas, lanzan ejecutables, cargan y descargan *plugins*, incluso pueden registrar *publishers* y *subscribers* de ROS.

2.3.1. CARACTERÍSTICAS PRINCIPALES.

Plataforma cruzada y contenido portable.

V-REP es multiplataforma y permite la creación de contenido portable, escalable y fácil de mantener o modificar: un solo archivo portable puede contener un modelo o escena totalmente funcional, incluido el código de control.

API remota.

Más de 100 funciones de V-REP empotradas: control de la simulación o del simulador por sí mismo de forma remota. Es fácil de utilizar, soporta operaciones síncronas o asíncronas, es optimizado por una gran transferencia de datos y minimiza el retraso de comunicación. Además, se puede escribir código en seis lenguajes distintos.

Detección de colisión y cálculo de distancias.

V-REP proporciona un chequeo de interferencias rápido y cálculo de mínima distancia entre cualquier malla o nube de puntos.

Dinámica/Física

Tiene una dinámica de cálculo para simular la física del mundo real y la interacción de los objetos rápido y personalizable. Puede implementar cuatro modelos distintos.

Simulación de sensores de visión y sensores de proximidad.

Posee una simulación de sensores de visión con procesamiento de imagen totalmente personalizable.

La simulación de los sensores de proximidad es potente y totalmente personalizable. Realiza un cálculo exacto de mínima distancia con un volumen de detección modificable.

Seis aproximaciones de programación.

Tanto el simulador como las simulaciones son totalmente personalizables, con seis posibles formas de programación, las cuales son mutuamente compatibles y pueden trabajar de manera conjunta.

2.3.2. TIPOS DE OBJETOS EN V-REP.

- ***Shapes*:** las formas son objetos de malla rígida con caras triangulares. Estos objetos pueden ser utilizados en detección de colisiones y cálculo de distancias respecto a otros objetos; también pueden ser detectados por sensores de proximidad y de visión.

- ***Joints*:** los *joints* son las herramientas utilizadas para construir mecanismos y objetos móviles, los cuales tienen al menos un DOF (*Degree of Freedom*). Hay cuatro tipos de *joints*: *revolute*, *prismatic* and *spherical* and *screw joints*. Los modos de operación de las articulaciones pueden ser: modo pasivo, modo cinemático inverso, modo dependiente, modo movimiento y finalmente modo fuerza o par. Un modelo dinámico del actuador puede ser modelado habilitando el modo fuerza o par, el cual dispone de un método de control de posición (PID).

- ***Proximity sensors*:** desde ultrasonidos hasta infrarrojos cercanos, todo tipo de sensores de proximidad pueden ser simulados y modelados. Obtienen un cálculo exacto de distancias entre su punto de sensado y cualquier entidad que interfiera en su volumen de detección.

- ***Force sensors*:** los sensores de fuerza son objetos que miden la fuerza o el par transmitido entre dos o más objetos. Estos sensores se modelan como un sensor real, por lo que pueden romperse si se supera el umbral permitido del sensor.

- ***Vision sensors*:** estos sensores pueden regenerar los objetos de una simulación (color, tamaño de objetos, mapas de profundidad…) y extraer información compleja de la imagen.

- ***Graphs*:** las gráficas son objetos para grabar, visualizar y exportar datos desde la simulación. Estos objetos pueden ser generados por el tipo de datos aplicado a cada objeto para ser grabados.

- ***Cameras*:** las cámaras so objetos que pueden monitorizar la simulación desde diferentes puntos de vista. Puedes incluir una multivisión con varias ventanas para poder observar bien la escena.

- ***Lights*:** con las luces se puede influir en los sensores y las cámaras.

- ***Paths*:** los patrones son objetos que definen una rotación, traslación o una combinación.

- ***Dummies*:** un maniquí es un tipo de objeto que puede ser definido como un marco de referencia o un punto de orientación unido al objeto. Son especialmente útiles para planificar un patrón de trayectoria y seguimiento.

- ***Mills***: utilizando fresadores se puede modelar casi cualquier tipo de volúmenes de corte, siempre que sean convexos. Las fresas siempre tienen un volumen de corte convexo; sin embargo, se pueden combinar para generar un volumen de corte no convexo o volúmenes más complejos.

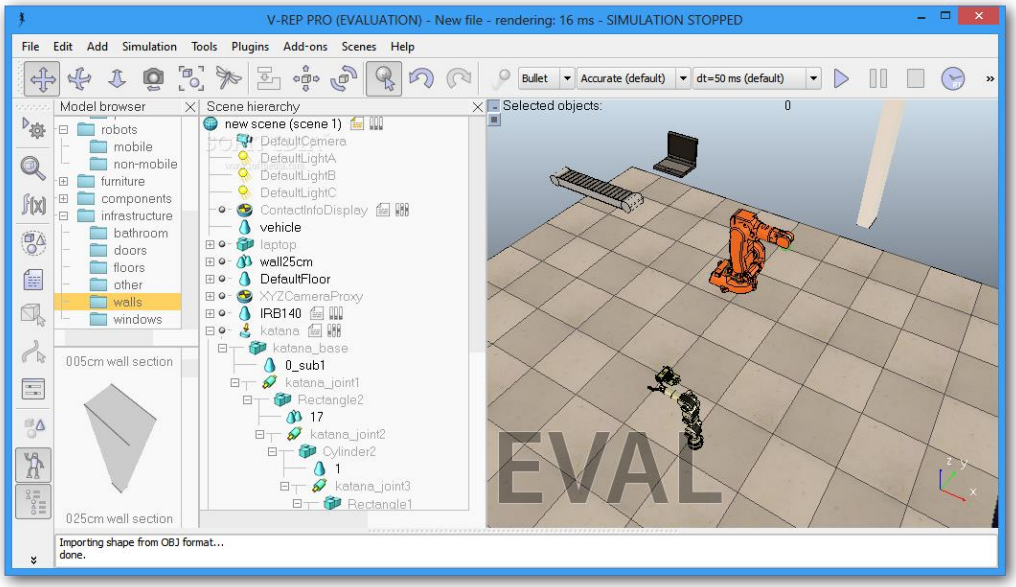


Figura 2.3.1. Imagen de una escena ejemplo de V-REP.

La combinación de los objetos descritos anteriormente permite la creación de sensores complejos (acelerómetros, giroscopios, GPS, *kinetic*, etc), y modelos complejos, como puede ser un robot móvil. V-REP dispone de una amplia librería de modelos de sensores y robots, que pueden ser añadidos de forma sencilla a la escena (pudiendo ser modificados totalmente).

3. DESARROLLO.

1. INTRODUCCIÓN.

Para la realización del proyecto, se utiliza el paquete *vrep_ros_bridge* el cual contiene una escena ejemplo en V-REP con el dron que se va a utilizar en la simulación, junto con un brazo robótico. Para poder trabajar con esta escena se limpia de todo aquello que es innecesario, dejando únicamente en ella el dron. La comparativa de ambas escenas se muestra a continuación:

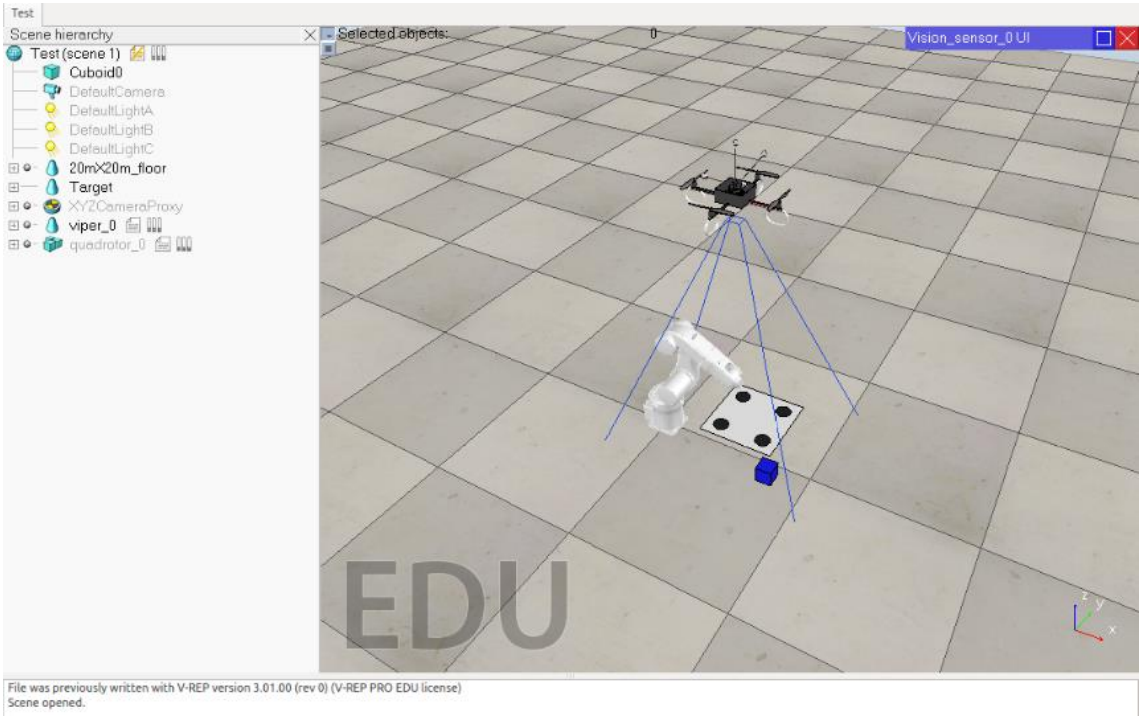


Figura 4.1.1. Escena de V-REP (Test.ttt).

La figura 4.1.1. muestra la escena original, llamada Test.ttt, disponible en la documentación del paquete *vrep_ros_bridge*. Como se observa en el *Scene hierarchy* de V-REP, esta escena contiene un cubo (*Cuboid0*), un brazo robótico (*viper_0*), una plataforma con círculos incorporados (*Target*) y el dron (*quadrotor_0*). Como se dijo

anteriormente, se va a eliminar de la escena todo lo que no es necesario para el proyecto, obteniendo como resultado:

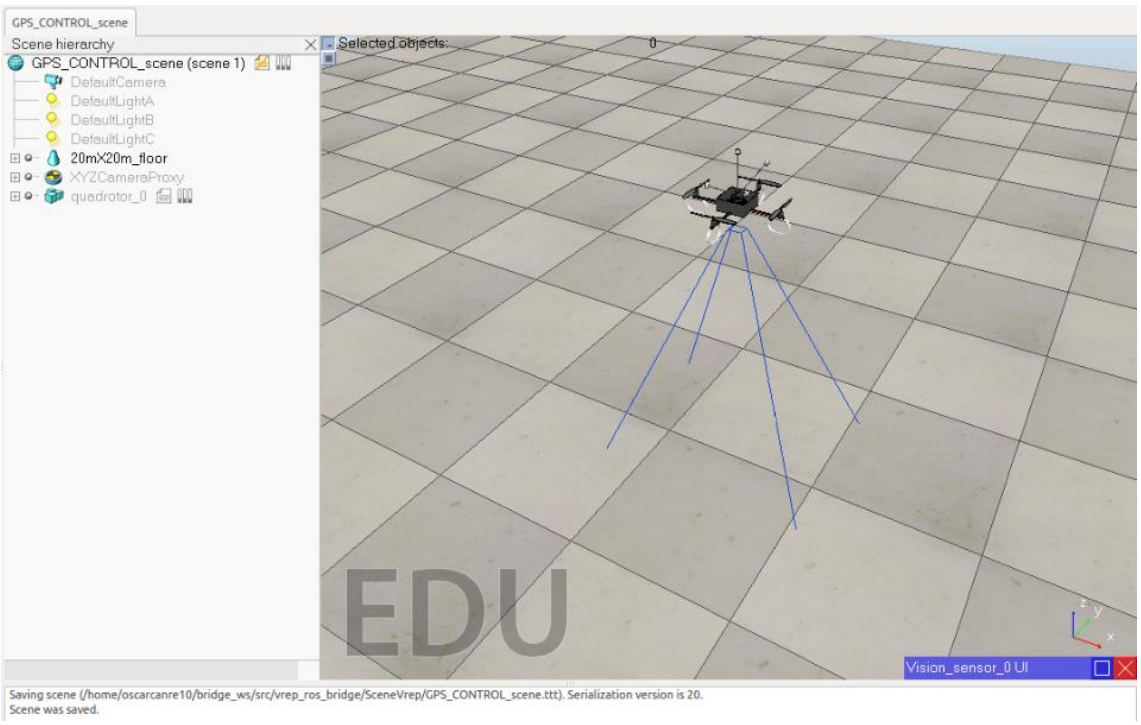


Figura 4.1.2. Escena V-REP (GPS_CONTROL_scene.ttt).

En esta última figura, ya se presenta únicamente el dron a utilizar para el proyecto, cuyo modelo en V-REP es:

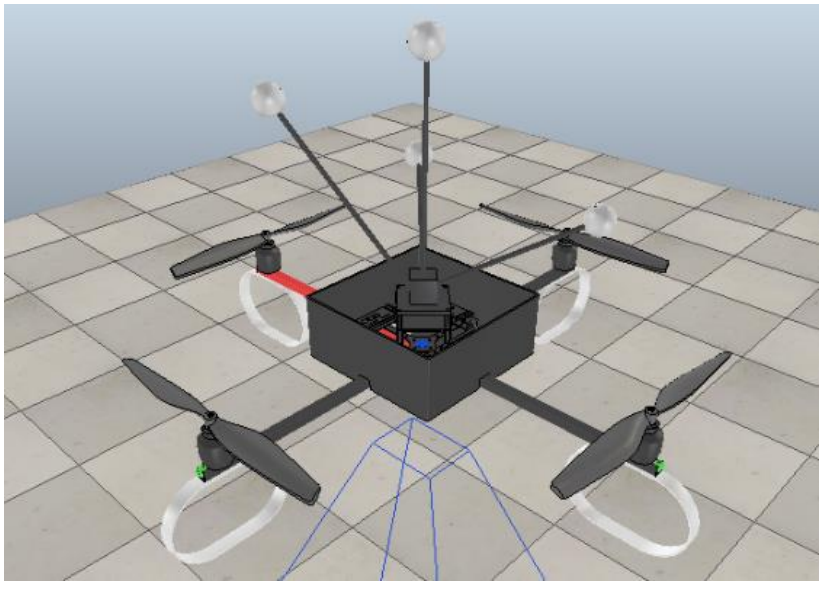


Figura 4.1.3. AscTec Hummingbrid modelado en simulador V-REP.

La escena con la que se trabaja en este apartado y que contiene lo mencionado previamente tiene como nombre *GPS_CONTROL_scene.ttt*.

El siguiente paso, y una de las partes más importantes, es entender cómo actuar sobre el dispositivo desde ROS mediante comandos. Para ello se lanza el simulador junto con un *roscore* y se introduce el comando *rostopic list* para saber qué *topics* se han declarado y cuáles son *subscribers* o *publishers*.

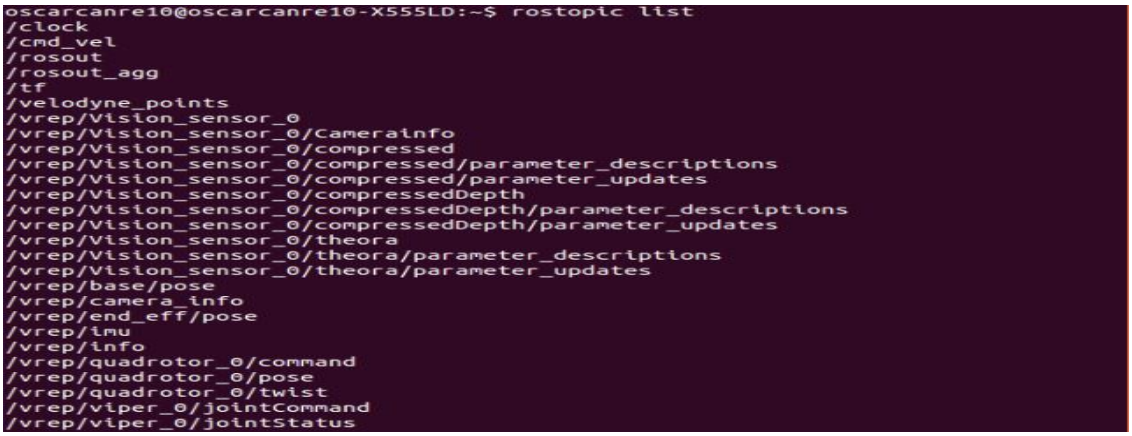


Figura 4.1.4. Listado de topics.

En la figura 4.1.4. se puede ver la lista de *topics*, pero de todos los que aparecen, solo va a ser útil en este apartado uno de ellos, *command,* el cual es un topic de tipo *sensor_msgs/JointState*.

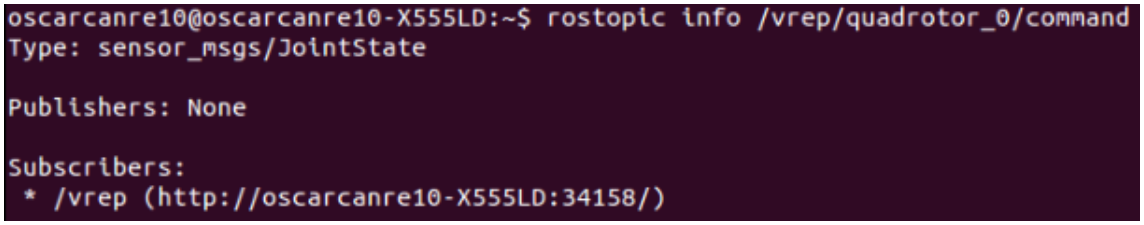


Figura 4.1.5. Información del topic /vrep/quadrotor_0/command.

Este mensaje consiste en múltiples arrays, uno para cada parte del Joint state. Cuando el sistema no requiere de alguno de estos campos, se puede dejar en blanco, no actuando sobre él. Todos los arrays deben tener el mismo tamaño o estar vacíos. Estos campos son *name, position, velocity* y *effort;* siendo el primero tipo string y los restantes de tipo float64. Observado el código C++ disponible en el paquete *vrep_ros_bridge*, puede verse como solamente es necesario el mensaje *effort*.

Lo próximo a tener en cuenta es saber cómo actúa el mensaje *effort* sobre el drone, para lo cual se mandan varios comandos y se observa en la escena cómo reacciona el cuadricóptero. Una vez realizadas estas pruebas, se llega a la conclusión de que el primer y el segundo parámetro son el par o *torque* en el eje x e y respectivamente; el tercer parámetro es el yaw del dron y el último el empuje o *thrust*. De modo que el tipo de mensaje a publicar quedaría de la siguiente forma:

```
rostopic pub /vrep/quadrotor_0/command sensor_msgs/JointState '{effort: [0.0, 0.0, 0.0, 0.0]}'—once
```

2. CONTROL ABSOLUTO DE POSICIÓN.

Para poder llegar a realizar un sistema en el que el dron siga de forma autónoma a un robot, primero hay que saber cómo puede moverse dicho dron por la escena de manera real (siguiendo las leyes físicas) a partir de los cuatro parámetros descritos anteriormente. Es por esto por lo que sería conveniente dar un primer paso creando un sistema en el que la aeronave se mueva dentro de su entorno a un punto dado y conocido.

Suponiendo que la posición absoluta del dron es conocida, se va a utilizar un *topic* más, declarado como *subscriber*, este elemento proporciona a ROS la posición y orientación del dron disponibles en V-REP, a partir de los cuales se va a realizar el lazo de control en Simulink.

```
oscarcanre10@oscarcanre10-X555LD:~$ rostopic info /vrep/quadrotor_0/pose
Type: geometry_msgs/PoseStamped

Publishers:
 * /vrep (http://oscarcanre10-X555LD:34158/)

Subscribers: None
```

Figura 4.2.1. Información del topic /vrep/quadrotor_0/pose.

Lo más importante de este apartado es encontrar la relación que tienen los parámetros obtenidos a partir de estos dos *topics* con los cuatro parámetros que se deben publicar en el *topic effort*.

Como resultado se van a crear 2 bloques diferenciados en un modelo de Simulink, los cuales se desarrollarán a continuación, siendo el modelo final utilizado:

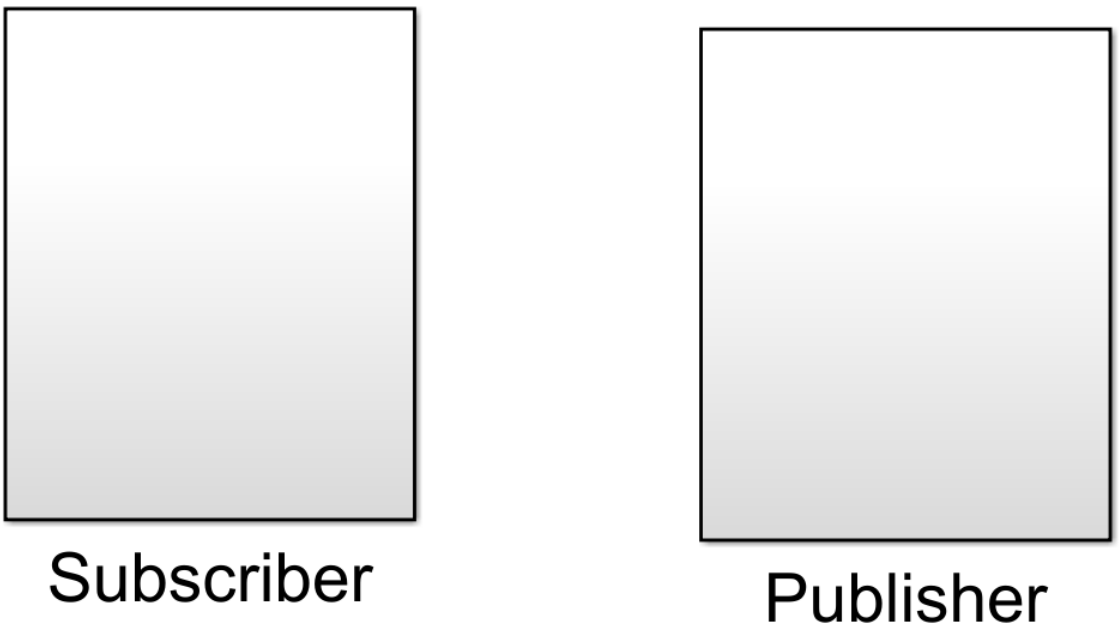


Figura 4.2.2. Visión general modelo Simulink (CONTROL_GPS.slx).

Donde el ‘‘Subscriber’’ obtiene los datos de la posición y orientación del dron y el bloque ‘‘Publisher’’ publica los datos para actuar sobre el dron y donde se realizan los controladores.

2.1. SUBSCRIBER.

Lo primero que se realiza en Simulink es el sistema de adquisición de los datos de V-REP de manera que se puedan tratar a continuación y poder formar el mensaje de actuación sobre el dron.

La información necesaria para realizar el lazo de control mediante la posición del dron se encuentra en el *topic* mencionado anteriormente (/vrep/quadrotor_0/pose). La adquisición de estos datos en Simulink es muy simple, ya que solo se necesita conocer el tipo de mensaje que se recibe y su nombre para configurar el bloque *Subscribe* que recibe los datos que se van a utilizar. La configuración de este bloque sería:

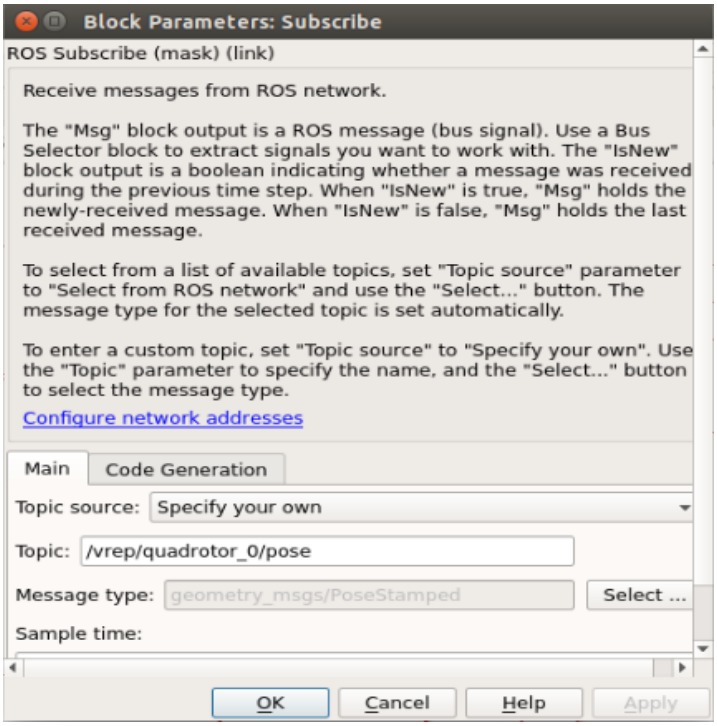


Figura 4.2.3. Configuración bloque Subscribe.

Para el correcto funcionamiento, se configura dicho bloque con el tipo de mensaje y el nombre del *topic*, en este caso, dicho *topic* tiene de nombre */vrep/quadrotor_0/pose* a través del cual se obtienen los datos de la posición en ejes X, Y y Z, y la orientación cuaternio. Para ello se tiene que utilizar también el bloque llamado *Bus Selector* el cual separa cada componente del mensaje de *pose* obteniendo las coordenadas de los datos mencionados con anterioridad, de manera independiente, de forma que el conjunto de bloques para obtener dicha información sería:

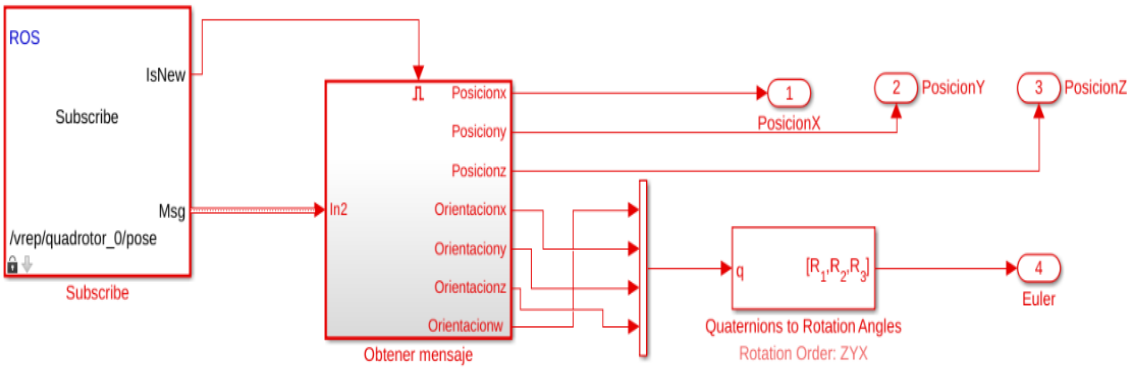
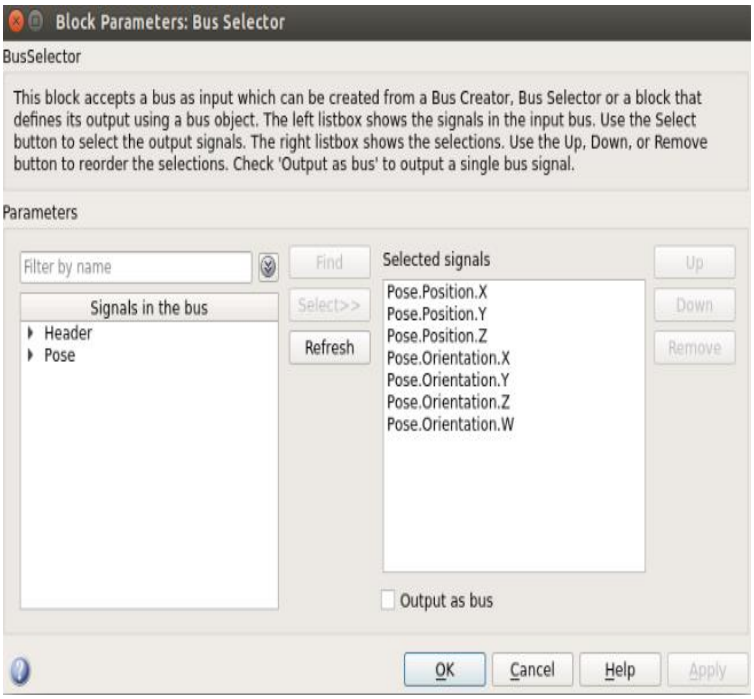


Figura 4.2.4. Subsistema Topics.

Para la realización del lazo de control posterior, será necesario transformar los ángulos en cuaternio obtenidos de V-REP (X, Y, Z y W) a ángulos de Euler, de manera que se utilizará el bloque *Quaternions to Rotation Angles*. La entrada de dicho bloque, como se ve en la figura x. tiene que estar formada por un vector con el orden de cada componente de forma adecuada, de la manera indicada en dicha imagen. El resultado final es un subsistema llamado **Topics**, cuyas salidas son PosicionX, PosicionY, PosicionZ y el vector Euler.

El subsistema llamado ‘‘Obtener mensaje’’ contiene un bloque Bus Selector, el cual se encarga de separar la información contenida en el mensaje de la siguiente forma:



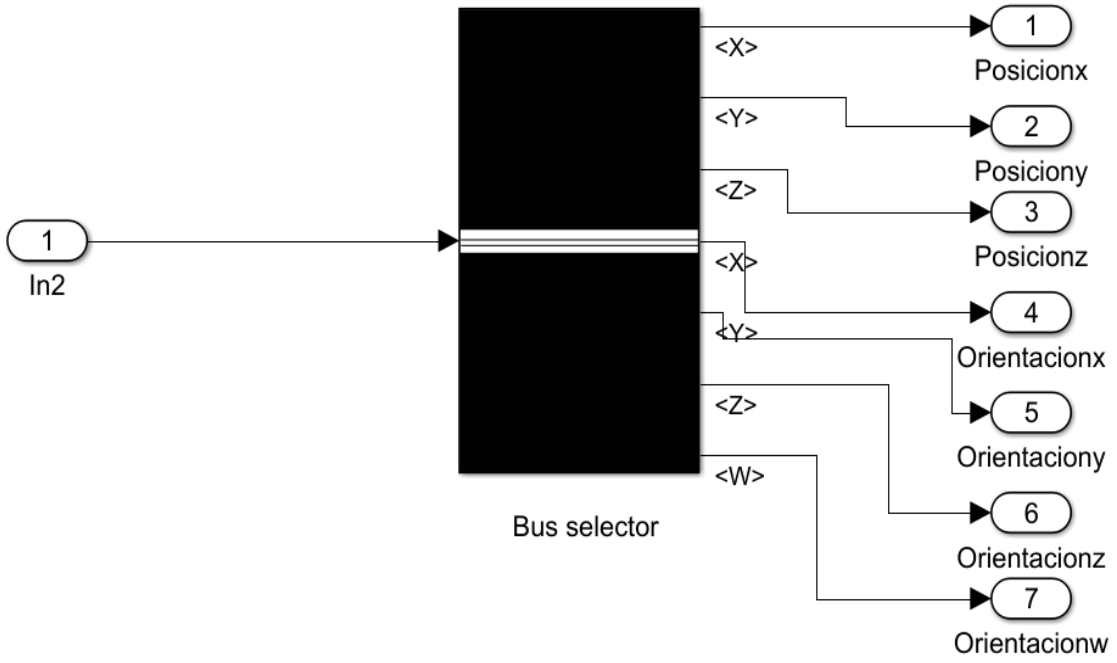


Figura 4.2.5. Subsistema Obtener mensaje y configuración del Bus Selector.

Para que el modelo de Simulink sea más claro, se crea otro subsistema llamado **Subscriber**, el cual contiene el bloque Topics a partir del cual se crean etiquetas de las salidas útiles, para posteriormente utilizarlas en otros bloques. Además, se indican en *Displays* los valores de dichas componentes para poder comprobar su correcto funcionamiento de forma rápida. El modelo del bloque Subscriber se muestra a continuación:

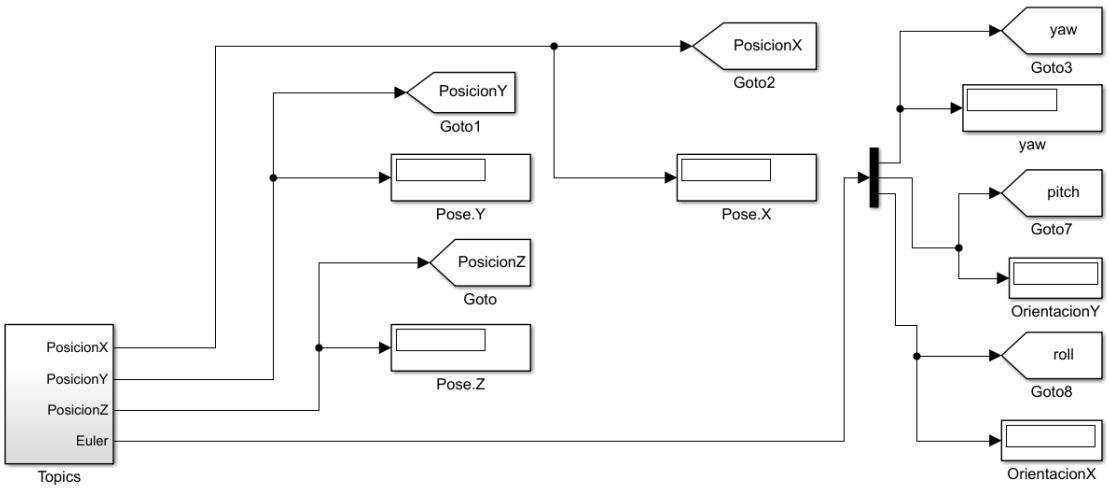


Figura 4.2.6. Subsistema Subscriber.

2.2. PUBLISHER.

La siguiente parte a tratar, es la formación del mensaje a enviar desde Simulink mediante ROS al simulador. Para poder formar el mensaje que se tiene que enviar, hay que obtener sus componentes, por lo que se realiza un controlador para cada

componente del mensaje. La formación del mensaje y los controladores de sus componentes se encuentran en el subsistema de Simulink llamado **Publisher**.

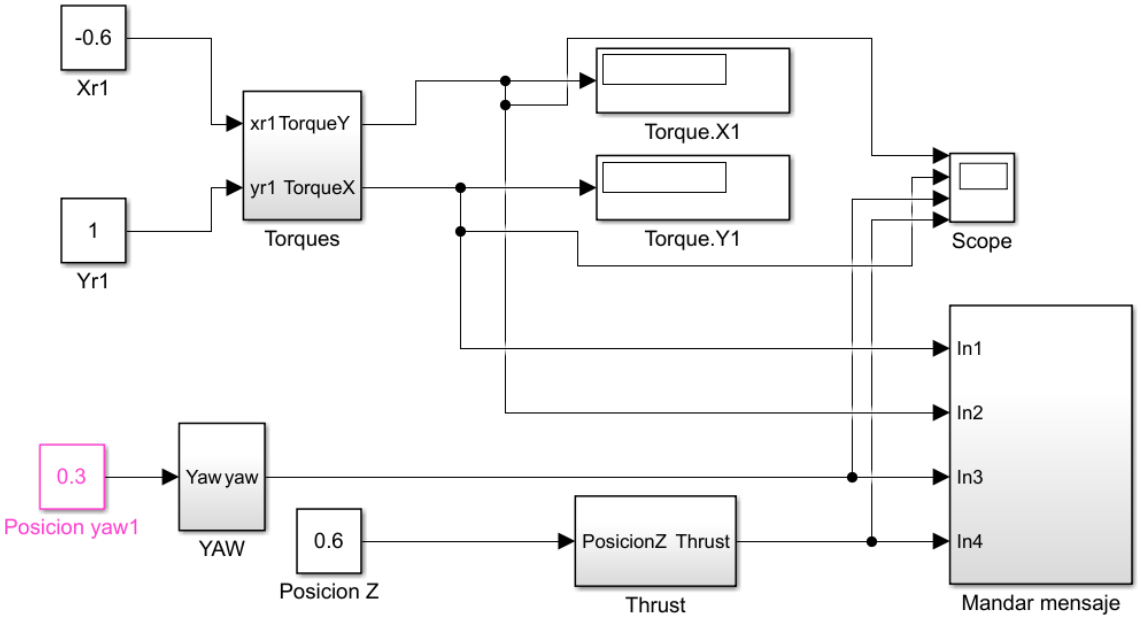


Figura 4.2.7. Subsistema Publisher.

Dicho subsistema contiene otros bloques que se explican a continuación:

Mandar mensaje

En la figura 4.2.7, se observa el bloque Publisher, el cual contiene los subsistemas necesarios para formar las componentes del *topic effort* (‘‘Thrust’’, ‘‘YAW’’ y ‘‘Torques’’), así como el bloque que crea en sí el mensaje a enviar, denominado ‘‘Mandar mensaje’’, el cual tiene el siguiente modelo:

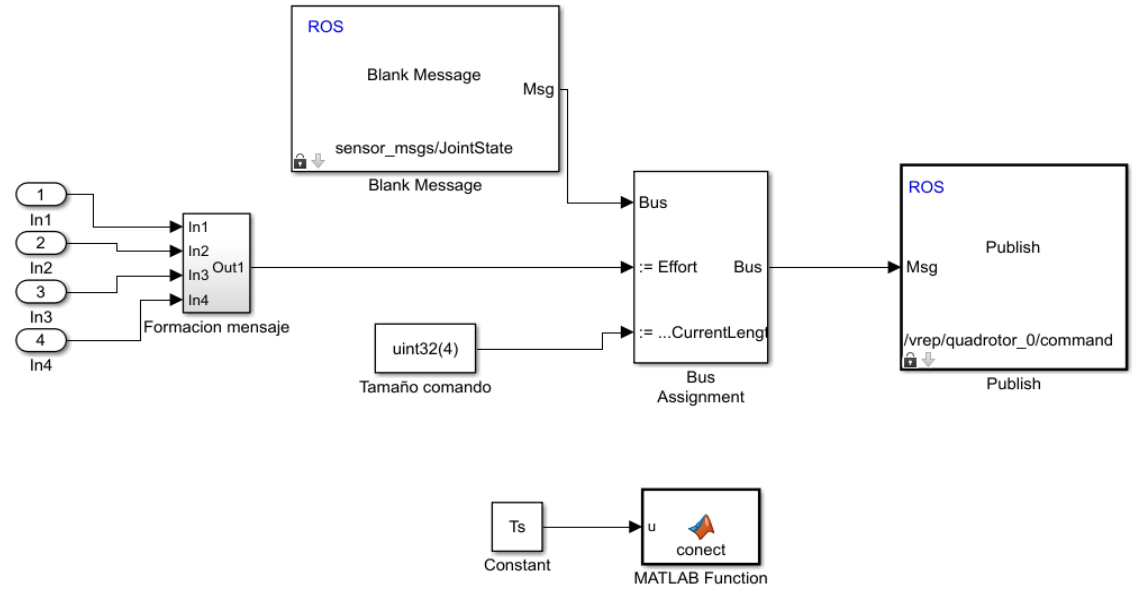


Figura 4.2.8. Subsistema Mandar mensaje.

Este subsistema está constituido por otra serie de bloques:

- **Formación mensaje:** Este bloque se encarga de formar el vector de 128 componentes que necesitan ROS y V-REP para funcionar de manera correcta. Esta es la razón por la que el vector está compuesta por las 4 componentes obtenidas y 124 componentes con valor 0.

- **Bus Assignment:** Es el bloque encargado de formar el mensaje a partir del vector obtenido anteriormente, el tamaño de los comandos (‘‘Tamaño comando’’) y el tipo de mensaje en ROS (‘‘Blank Message’’).

- **Matlab function:** este bloque es utilizado como método de sincronismo entre V-REP y Simulink. Contiene una función pause (Ts) que permite que el simulador y Simulink operen a la misma velocidad.

- **Publish:** Se encarga de publicar los datos obtenidos para que el dron se mueva a la posición indicada, para ello, se configura de la siguiente manera:

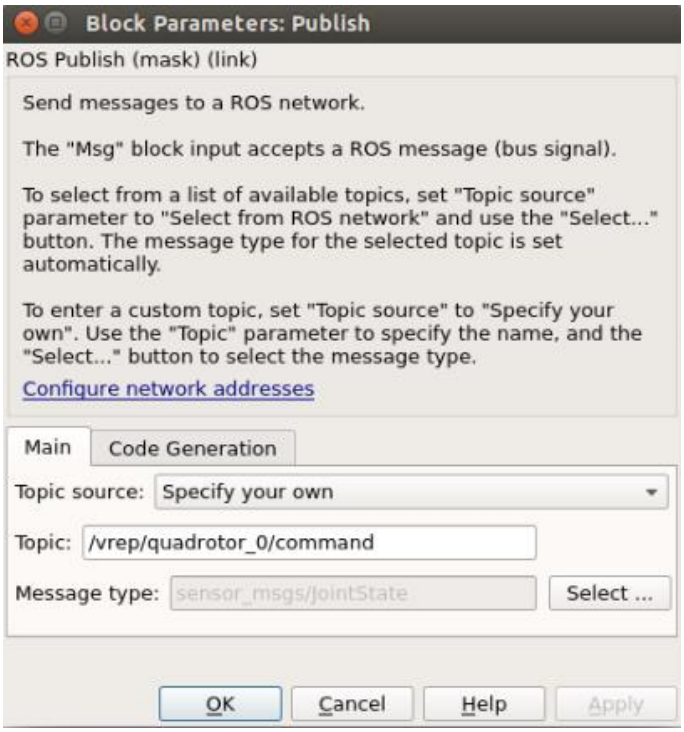


Figura 4.2.9. Configuración del bloque Publish.

Las 4 componentes que entran a este bloque deben ser de tipo continuo, ya que, si no es así, el mensaje no puede ser formado, dando como resultado una simulación errónea.

Control del thrust.

La primera componente del mensaje que se compondrá es el *Thrust* o empuje del dron, de manera que, imponiendo una cierta altura, el dron debe estabilizarse a esa distancia del suelo de la escena.

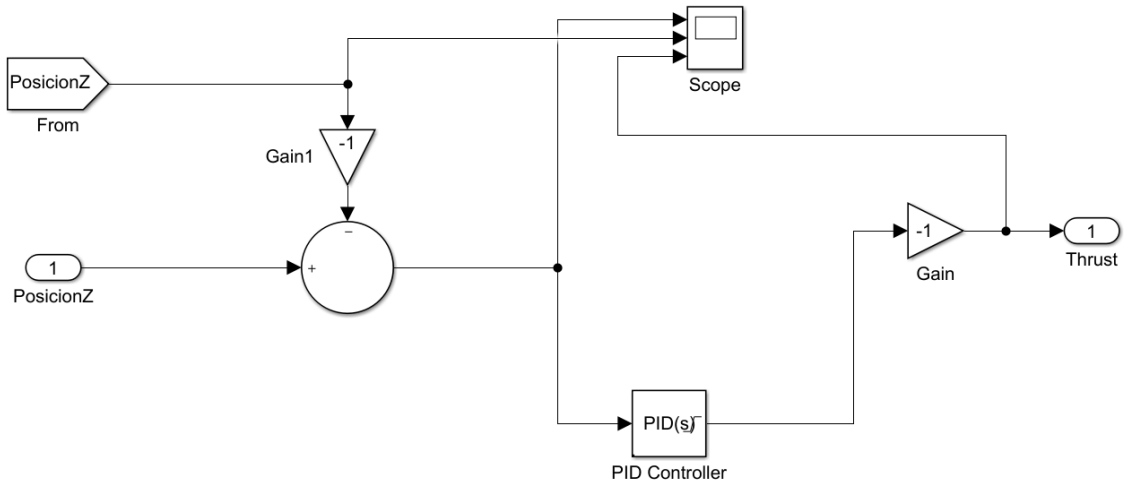
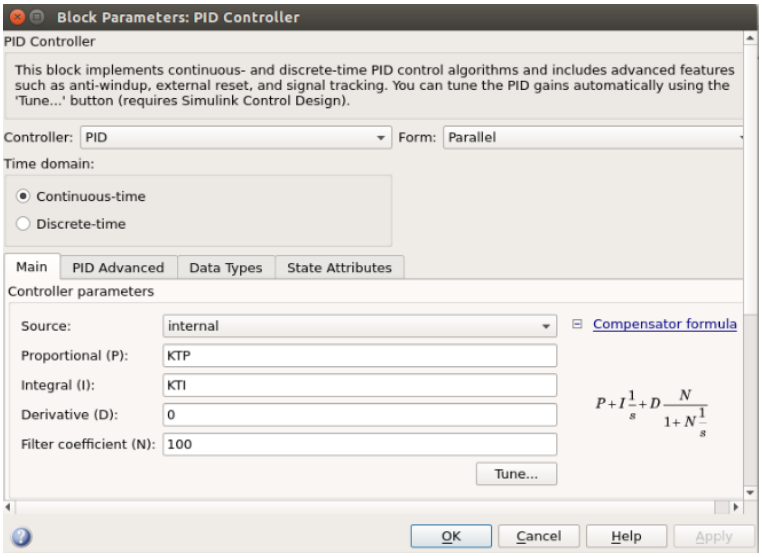


Figura 4.2.10. Subsistema Thrust.

Este bloque tiene como entrada la posición en el eje Z, con valor positivo, y se obtiene como salida el valor del empuje que es necesario dar al dron para que llegue a ese objetivo, bien que suba, baje o se mantenga a una cierta altura. Como se observa en la figura 4.2.10, en el lazo de control hay dos bloques de ganancia que invierten el signo. Esto se debe a que en el mundo de V-REP, el dron tiene la coordenada Z negativa respecto a la escena, por lo que desde el Publisher se obtiene una componente Z negativa, y para el correcto funcionamiento del dron se deben invertir.

El control del thrust aplicado al dispositivo se realiza con un controlador PID de la siguiente manera:



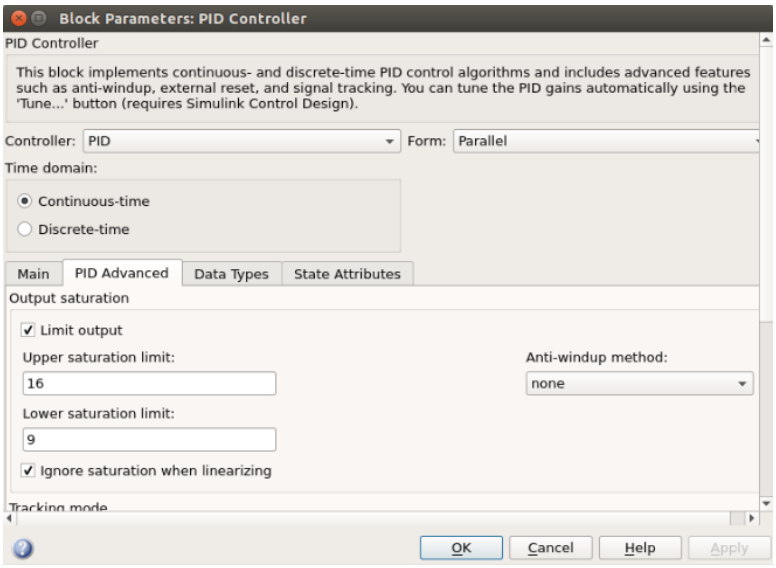


Figura 4.2.11. Configuración del PID.

El valor de la constante proporcional (KTP) del PID toma el valor de 12.7 debido a que es el valor necesario para que el dron no ascienda ni descienda en la escena, mientras que KTI o constante integral es igual a 7, obteniéndose de manera experimental.

De esta forma se consigue la corrección del error en el eje Z deseada, como por ejemplo se muestra en la siguiente figura:

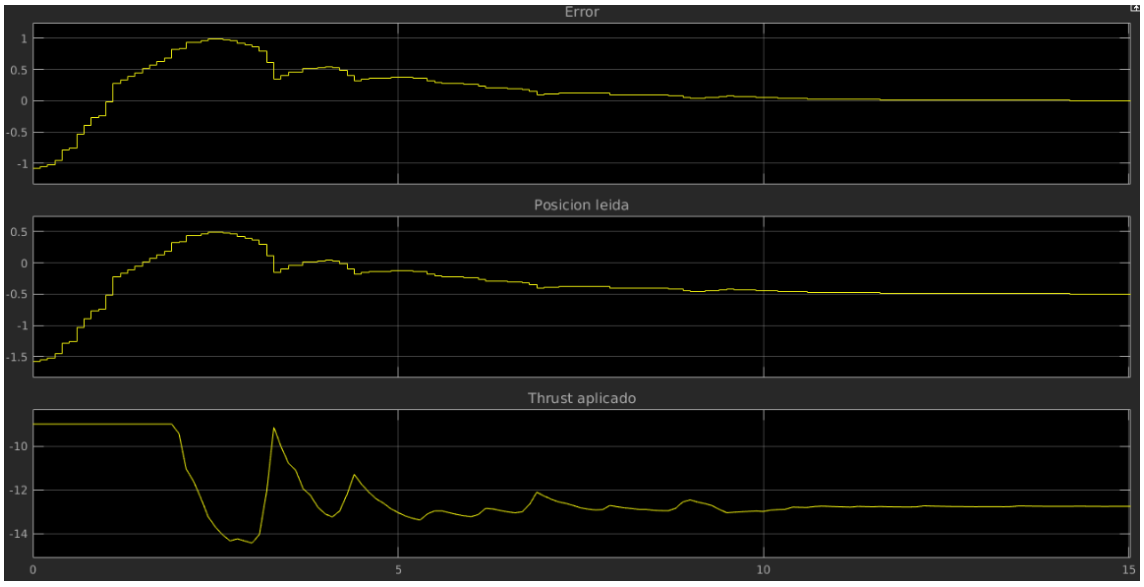


Figura 4.2.12. Scope del lazo de control del thrust.

En la figura anterior se tiene una posición inicial del dron en el eje Z de 1.577 metros, y se quiere llegar a una altura 0.5 metros en el mismo eje, por lo que se aplica el thrust necesario para ello, hasta conseguir error nulo, como se muestra en la gráfica. De esta forma se comprueba el correcto funcionamiento del sistema de control para la primera componente del mensaje estudiada.

Control del yaw.

El ángulo yaw es una componente de los ángulos de Euler, también denominado ángulo de rotación. Este ángulo es importante pues es el que marca la orientación del dron, mediante el ángulo formado respecto al eje z.

Un inconveniente a tener en cuenta es que el topic a través del cual se obtiene la información de la orientación del dron en la escena, proporciona un cuaternio, compuesto por un escalar (W) y un vector (X, Y, y Z). De manera que se transforma dicho quaternion a los ángulos de Euler mediante el bloque de Simulink *Quaternion to Euler angles* según se ha descrito anteriormente.

Una vez ya se dispone del yaw del dron, se puede realizar su bloque de control, obteniéndose lo siguiente:

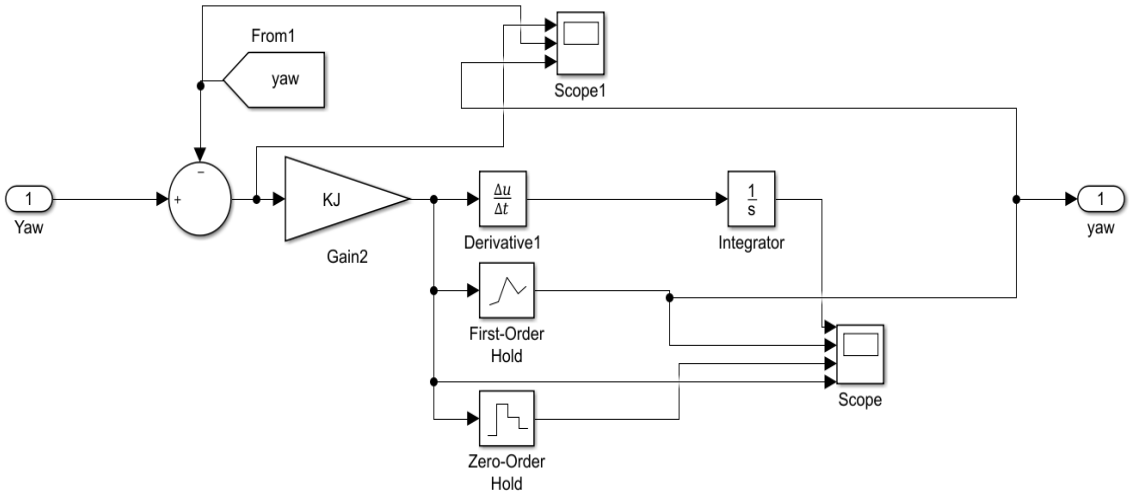


Figura 4.2.13. Subsistema YAW.

Como entrada al bloque YAW se utiliza el valor del ángulo final deseado, siendo realimentado con el valor instantáneo del mismo ángulo, por lo que el control es directo mediante una constante de proporcionalidad de valor KJ=0.4. La salida de este bloque es el torque entorno al eje Z a aplicar en cada momento sobre el dron para que se obtenga el valor final deseado.

Como se observa en la figura 4.2.14, la salida del lazo de control tras pasar por la constante proporcional es de tipo discreto, debido a esto se proponen los tres posibles casos que aparecen en dicha figura:

- Derivador + integrador.
- First-Order-Hold.
- Zero-Order-Hold.

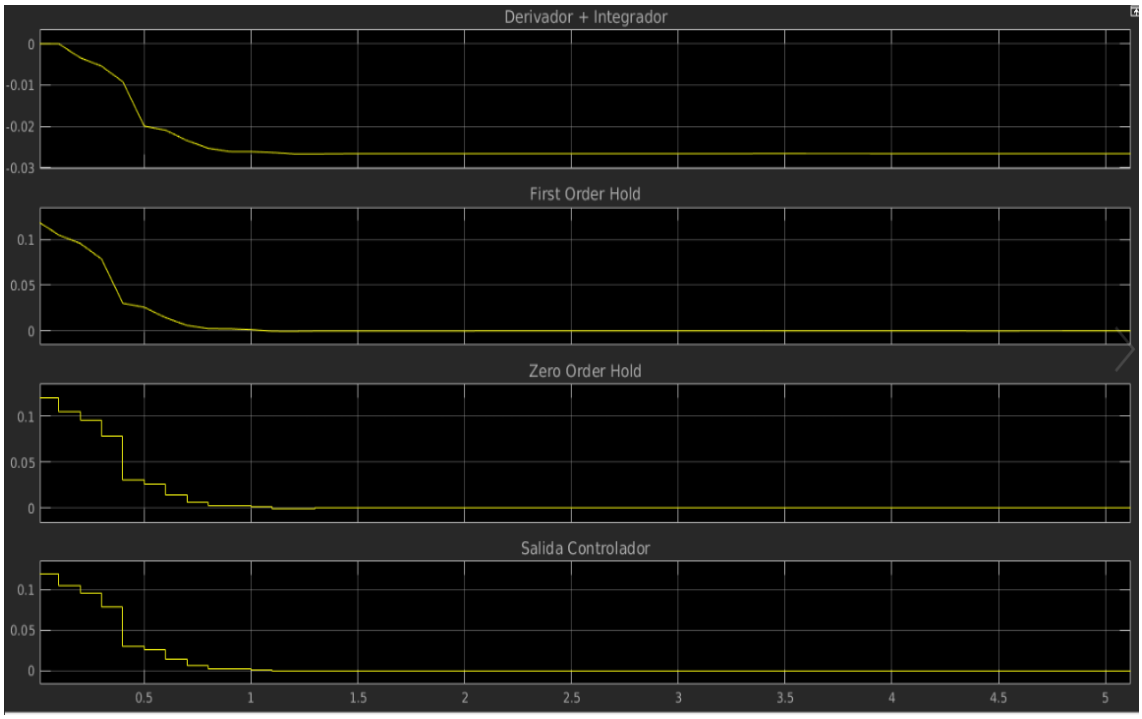


Figura 4.2.14. Scope comparativo.

Se decide utilizar la opción del FOH ya que es la solución más realista, por tanto, cada vez que en las posteriores simulaciones el dato que se debe mandar a V-REP sea de tipo discreto, se añadirá este bloque para realizar su conversión.

Una prueba del correcto funcionamiento se muestra a continuación, donde se tiene un valor inicial de 0 rad/s, llegando a un valor final de 0.3 rad/s. El lazo de control actúa de la siguiente forma en esta componente del mensaje para conseguir error nulo en el lazo de control:

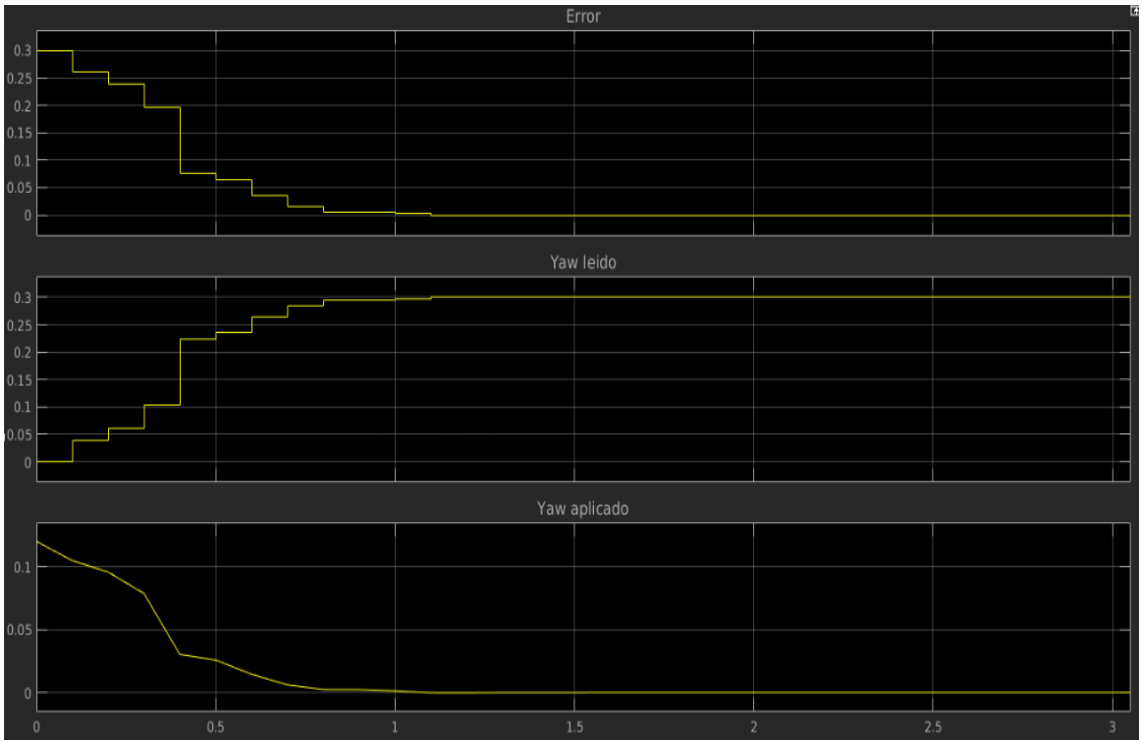
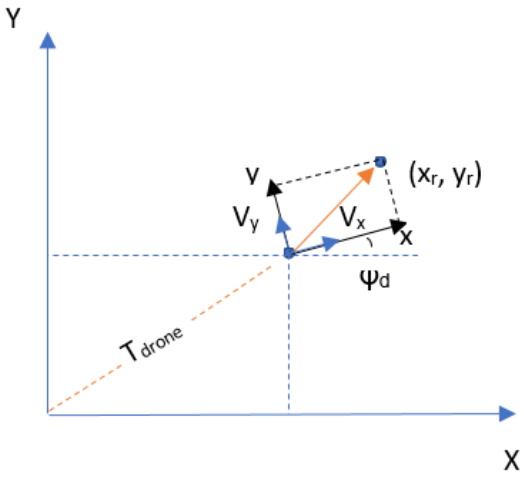


Figura 4.2.15. Scope lazo de control del yaw.

Control de los torques en X e Y.

El torque o par que se aplica en cada eje afecta a las hélices y es lo que provoca el movimiento en los ejes cartesianos X e Y del dron. Para poder realizar el lazo de control de los torques del dron, se deben obtener las coordenadas locales del dispositivo conociendo las globales debido a lo que nos proporciona V-REP. Por lo tanto, lo primero que se debe conseguir es la velocidad en los ejes X e Y (Vx y Vy) para posteriormente convertirlo a par, y así formar las dos últimas componentes del mensaje que falta por completar:



La matriz de transformación homogénea que caracteriza la traslación y rotación del dron respecto del mundo es:

$$\text{Tdrone} = \begin{bmatrix} C_{\Psi_d} & -S_{\Psi_d} & xd \\ S_{\Psi_d} & C_{\Psi_d} & yd \\ 0 & 0 & 1 \end{bmatrix}$$

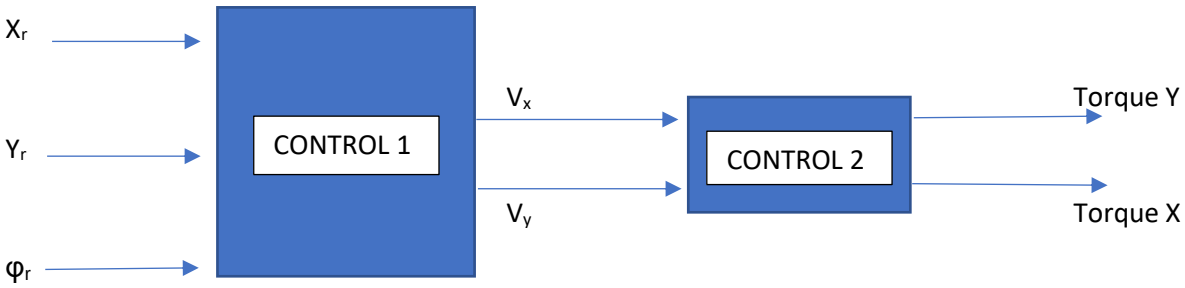
Por lo tanto, las coordenadas globales del punto a alcanzar (xr, yr) a partir de las velocidades del dron (Vx, Vy) son:

$$\begin{bmatrix} xr \\ yr \\ 1 \end{bmatrix} = \begin{bmatrix} C_{\Psi_d} & -S_{\Psi_d} & xd \\ S_{\Psi_d} & C_{\Psi_d} & yd \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} Vx \\ Vy \\ 1 \end{bmatrix}$$

Así, se obtienen las ecuaciones de las velocidades a aplicar al dron:

$$Vx = Kp * [cos\psi_d * (x_r - x_d) + sen\psi_d * (y_r - y_d)]$$
$$Vy = Kp * [cos\psi_d * (y_r - y_d) - sen\psi_d * (x_r - x_d)]$$

Estas ecuaciones han sido obtenidas con los ejes cartesianos X e Y canónicos, es decir, la parte positiva del eje X es la derecha, y la parte positiva del eje Y es la superior. El problema es que en el sistema de V-REP, la escena cambia los ejes, por lo que el eje Y positivo pasa a ser la parte inferior. Con estos cambios, se obtiene las mismas ecuaciones que se mostraban anteriormente.



El modelo de bloques anterior se traslada directamente al Simulink, de modo que las entradas al bloque en el que se realizan los controles, llamado X, son las posiciones deseadas en X e Y, y las obtenidas mediante el sistema Subscriber explicado anteriormente (ángulo de orientación φ y posición en X e Y); siendo las salidas el par en X e Y que actúan sobre el dron, formando el mensaje que se va a mandar.

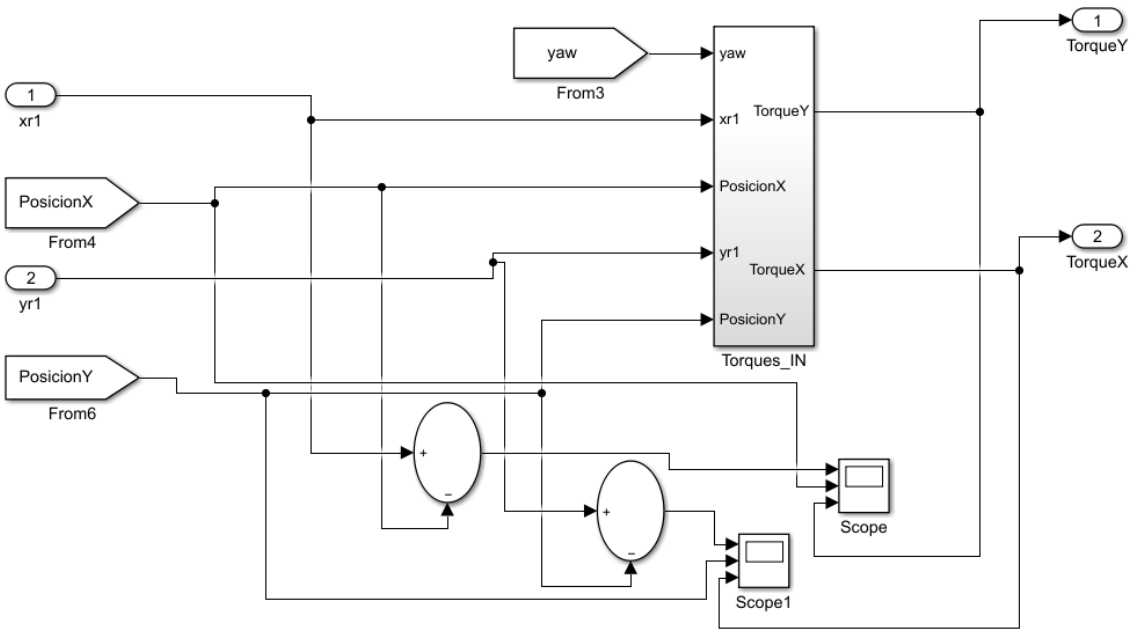


Figura 4.2.16. Subsistema Torques.

En el interior del bloque X solamente se divide en dos el lazo de control mediante dos bloques: ‘‘Vy’’ y ‘‘Vx’’, en los que se realiza el controlador de ambas componentes. La salida de ambos bloques es de tipo discreto, por lo que se utiliza el bloque First-Order-Hold para obtener una señal de tipo continuo y poder añadirla al mensaje:

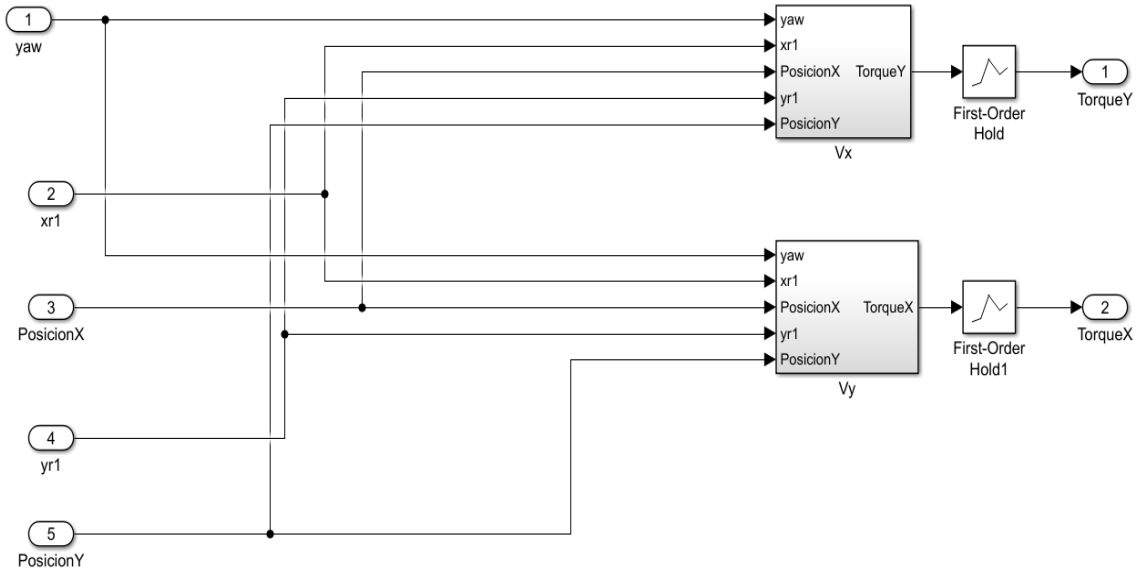


Figura 4.2.17. Subsistema Torques_IN.

Lo importante de este modelo de Simulink es el contenido de los bloques ‘‘Vy’’ y ‘‘Vx’’, los cuales se dividen en dos lazos de realimentación. Todo ello se describe a continuación:

➢ **Vx:**

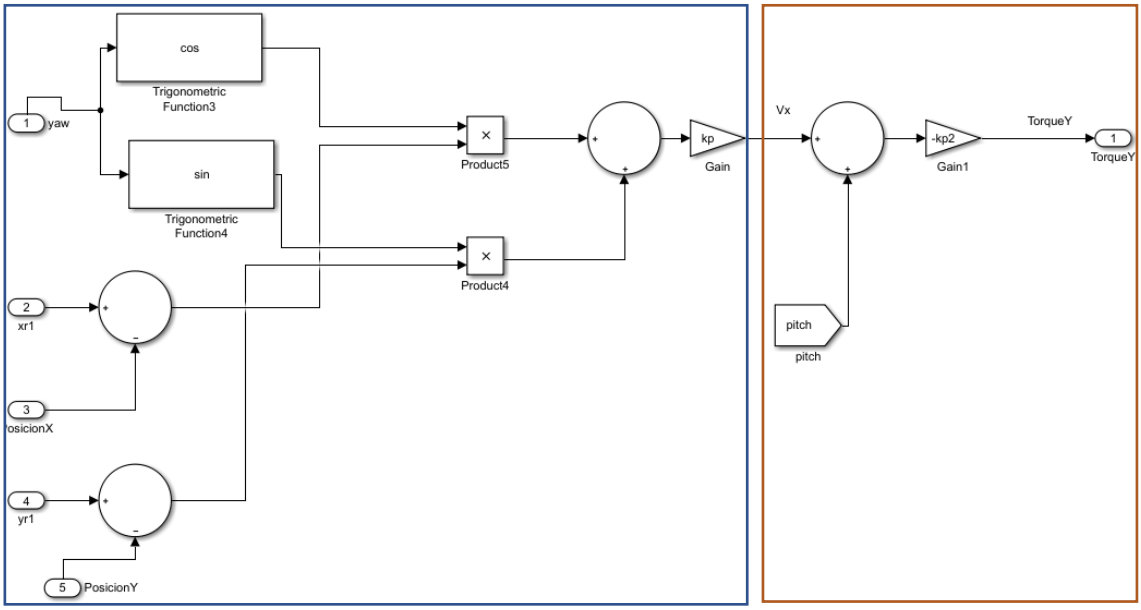


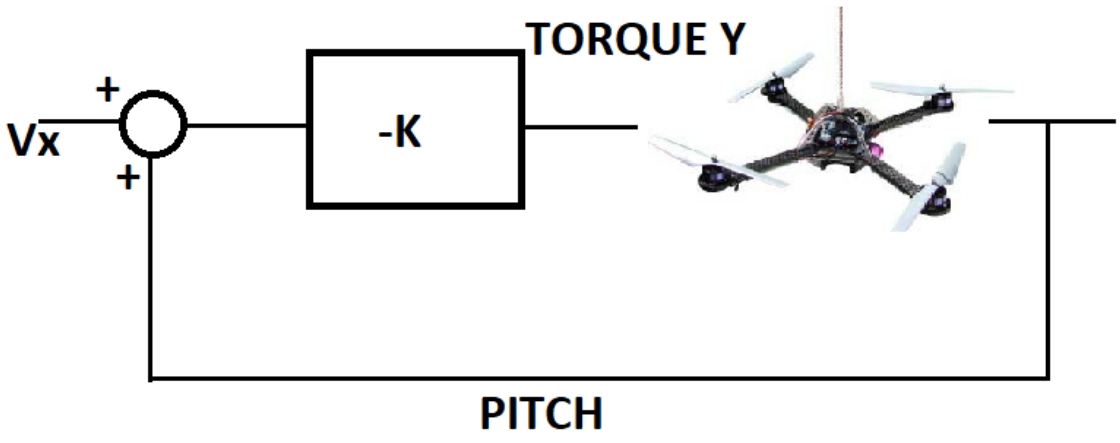
Figura 4.2.18. Subsistema Vx.

Los dos lazos de realimentación se dividen como se puede observar en la figura 4.2.18, de manera que en el recuadro azul se obtiene la V_x en función de las entradas descritas antes, y en el recuadro marrón se consigue el torque a aplicar en el eje Y para que el dron se mueva de forma correcta a partir de la V_x calculada justo antes.

La primera parte del modelo cumple con una de las ecuaciones descritas al comienzo del apartado de manera que se realiza en bloques de Simulink siendo kp=0.05 y kp2=0.1:

$$Vx = Kp * [cos\psi_d * (x_r - x_d) + sen\psi_d * (y_r - y_d)]$$

Una vez se obtiene la V_x, se debe obtener el torque correspondiente, que en este caso es el par en el eje Y. Para ello, hay que ayudarse de otro ángulo de Euler de los calculados previamente, el *Pitch*.



El ángulo pitch viene dado por el giro entorno al eje Y, por lo tanto, para dar velocidad en el eje X y que el dron se desplace a lo largo de ese eje, hay que aplicar un par negativo entorno al eje Y (siguiendo la regla de la mano derecha). Como las coordenadas en el sistema están orientadas de forma opuesta (X e Y) a como las recibe Matlab de V-REP, el lazo de control se realimenta de forma positiva.

- **Vy:**

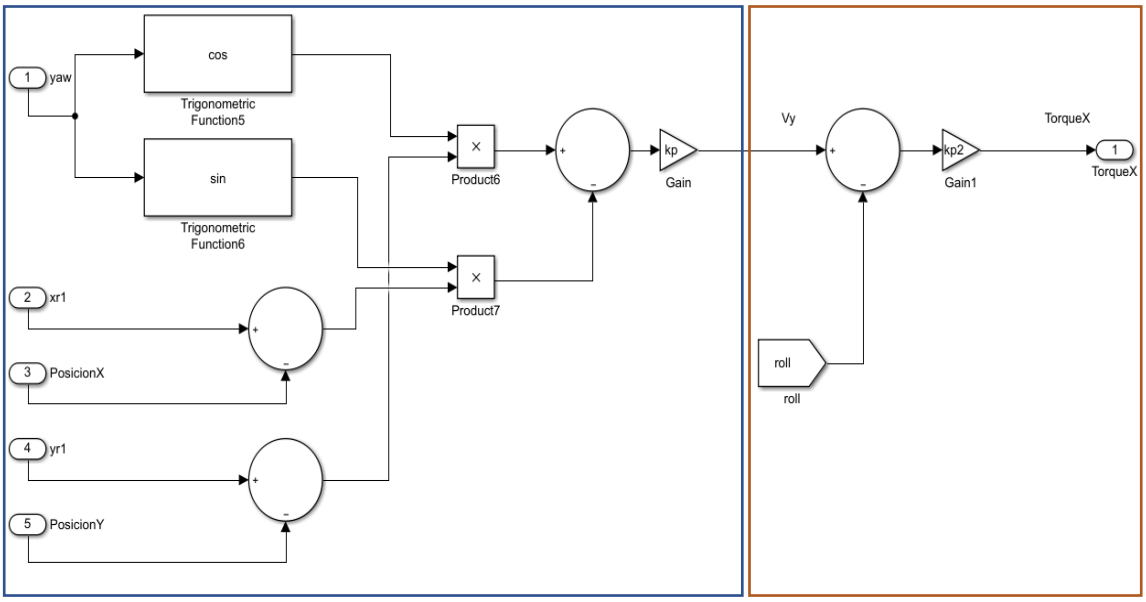
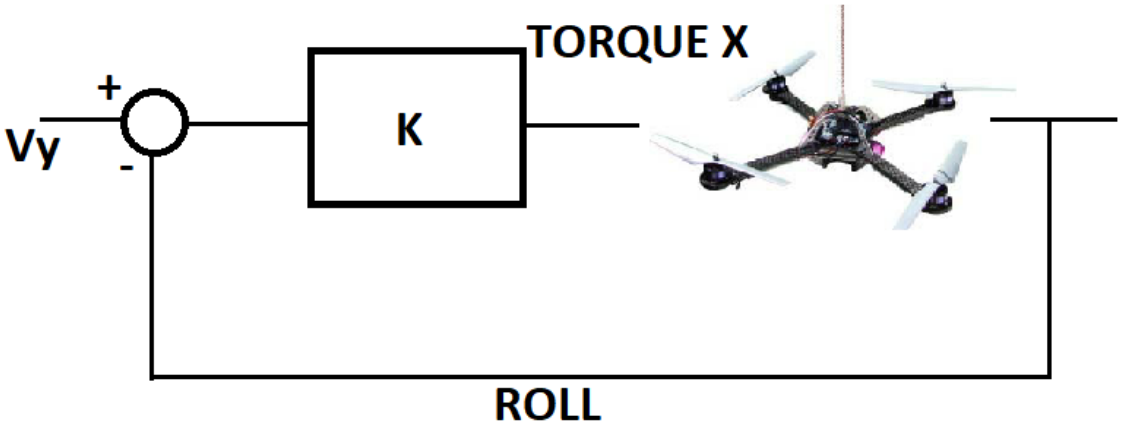


Figura 4.2.19. Subsistema Vy.

En este caso, la ecuación que cumple el recuadro azul es la que obtiene V_y como resultado y siendo kp=0.05 y kp2=0.1:

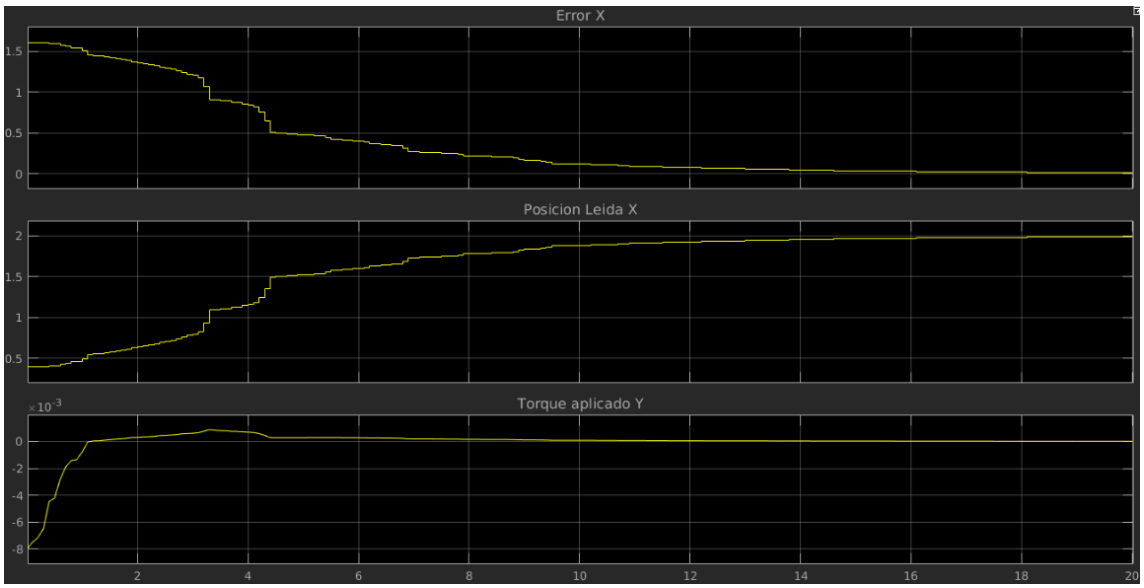
$$Vy = Kp * [cos\psi_d * (y_r - y_d) - sen\psi_d * (x_r - x_d)]$$

Para el segundo controlador, también ha de utilizarse otro de los ángulos de Euler calculados, el restante, *Roll*. De manera que:



Del mismo modo que con el pitch para que el dron se desplace a lo largo del eje Y, se debe aplicar un torque positivo en torno al eje X, de modo que el dron se incline hasta la posición adecuada y se mueva por el eje Y. En este caso, la coordenada X coincide en ambos sistemas, por lo que el lazo de control se realimenta negativamente.

Para comprobar el correcto funcionamiento del dron con el controlador implementado, se realiza una prueba de modo que el cuadricóptero se desplace desde las coordenadas (0.396, 0.163) hasta el punto (2, -0.3) dentro de la escena de V-REP. Con un desplazamiento de este tipo, debería corregirse la posición en ambos ejes, aplicando el torque necesario para que el drone realice esa trayectoria hasta su destino. Como resultado de la prueba se obtiene lo siguiente:



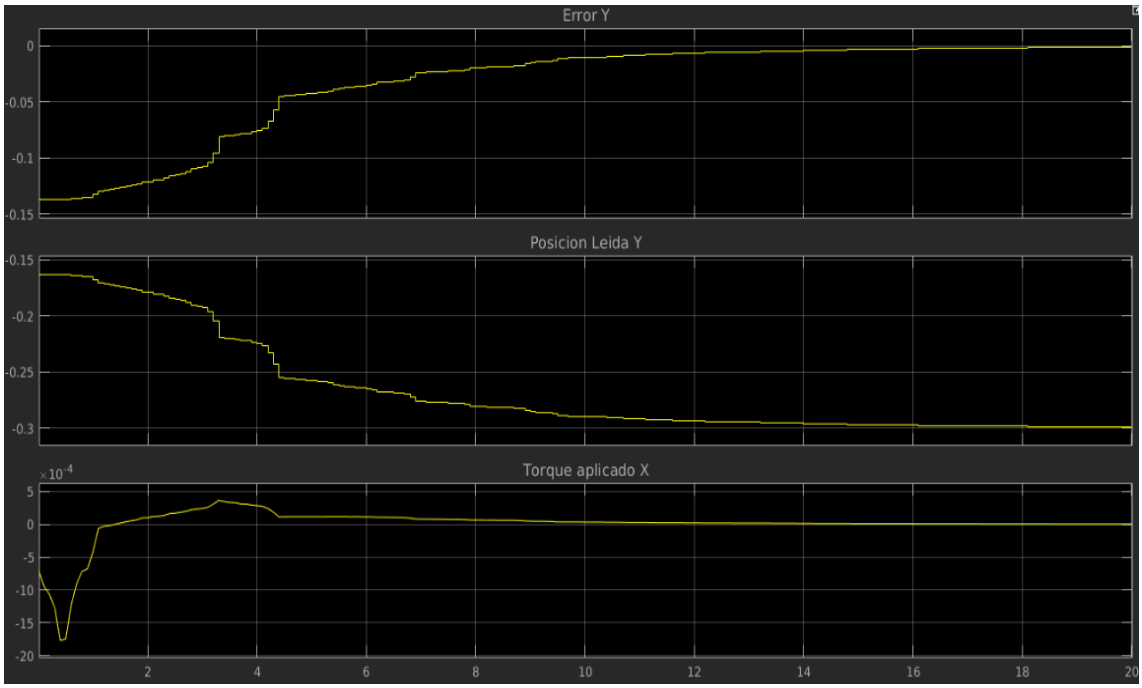


Figura 4.2.20. Scope lazos de control de los torques en X e Y.

Con los resultados mostrados en la figura 4.2.20, queda demostrado el funcionamiento óptimo del controlador, de manera que el sistema aplica par en ambos ejes mientras que el error no ha llegado a ser nulo, provocando un desplazamiento del dron en el espacio. Una vez que el error de la posición en los ejes desaparece, el torque se hace nulo para evitar que el dron se desplace de forma anómala.

Por tanto, ya se puede formar el mensaje completo para enviar los datos de actuación sobre el dron.

3. CONTROL PARA EL SEGUIMIENTO VISUAL DE UN OBJETO.

Una vez comprendido el funcionamiento del dron y hecha una primera aproximación a un sistema de control de posición del mismo entrelazando los tres softwares que se utilizan, se puede dar el paso hacia delante necesario para lograr llegar al objetivo del proyecto: el seguimiento de un robot.

La filosofía de trabajo en este caso a estudiar es similar a la seguida en el apartado anterior. Ahora, además de un dron en la escena de V-REP, se va a disponer de una especie de plataforma móvil manual que va a hacer las veces de robot, a partir de la cual se obtienen los datos de movimiento necesarios para que el dispositivo siga el movimiento de dicha plataforma. Por tanto, la escena que se va a utilizar en el simulador resulta:

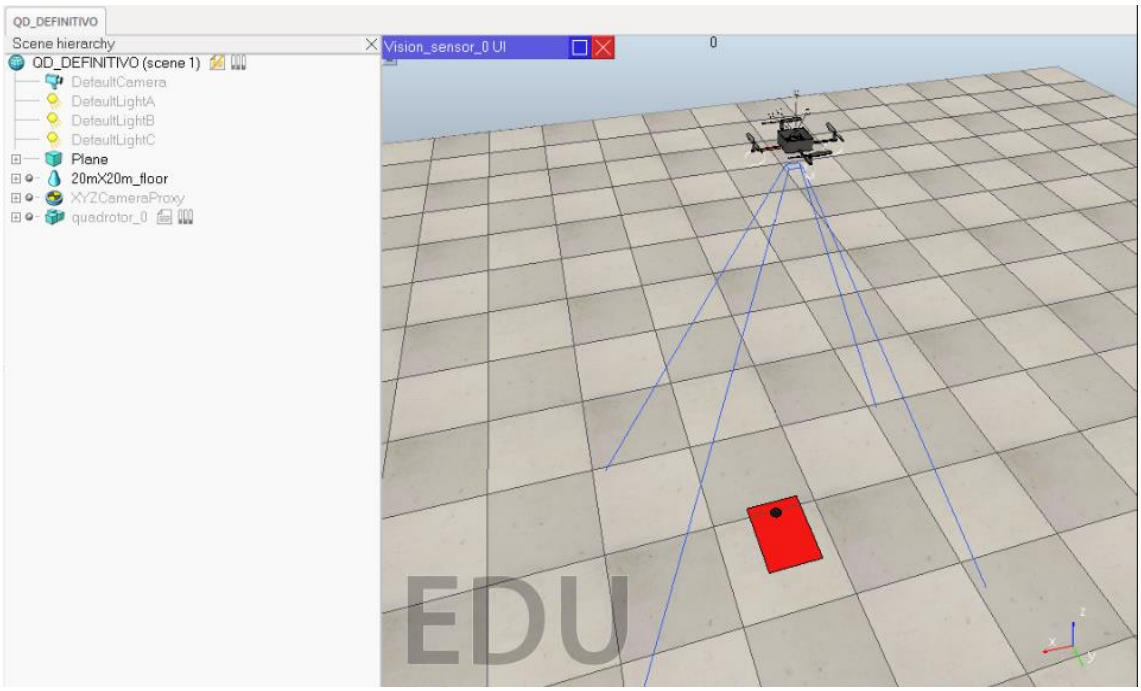


Figura 4.3.1. Escena V-REP (QD_platform.ttt).

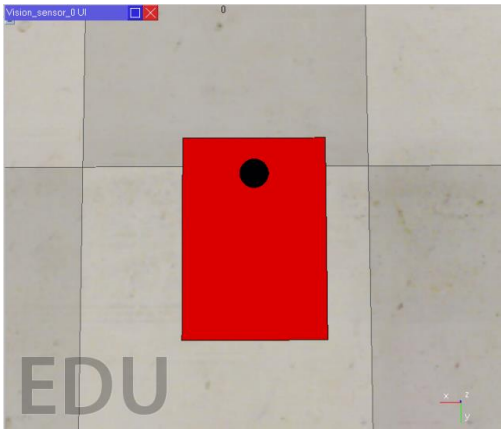


Figura 4.3.2. Modelado de la plataforma en V-REP.

Como se puede ver en las figuras 4.3.2/3, la plataforma tiene un color rojo, y en su interior se encuentra un círculo de color negro. El color rojo se va a utilizar para poder segmentar la imagen, y saber en qué posición se encuentra la plataforma dentro de la imagen; mientras que el círculo de color negro va a ser utilizado para conseguir el ángulo de rotación adecuado de tal manera que, si la plataforma se desplaza, el dron se desplace, y si la plataforma rota, el dron lo haga de la misma manera consiguiendo así que la imagen vista por el dron, que se recibe y muestra en Simulink, se intente que esté centrada y bien orientada, haciendo las correcciones necesarias:

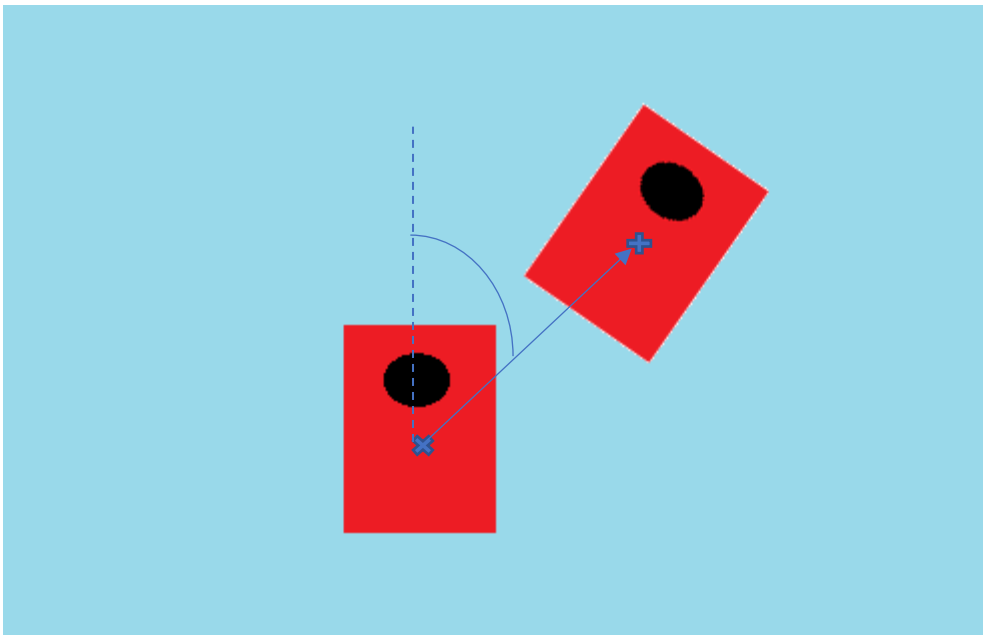


Figura 4.3.3. Imagen orientativa de las correcciones de posición y orientación.

En la figura 4.3.3 se muestra como la cámara del dron recoge un cambio de posición y orientación de la plataforma, de modo que el objetivo será reducir la distancia entre los centros de la plataforma y eliminar el ángulo que forman los centros de los círculos negros.

A partir de la información de la posición y orientación calculada a partir del tratamiento de imagen en Simulink, se crea un controlador similar al modelado en el apartado anterior, a partir del cual se forma el mensaje a enviar a través de ROS para que el dron actúe según lo necesario en el simulador.

Esta vez, la información relativa a la posición del dron que suministra V-REP no se va a utilizar para el modelo del controlador, ya que se supone que el dron no cuenta con ella para realizar sus movimientos, teniendo sentido así la obtención de esos datos mediante el tratamiento de la imagen.

Como en el apartado anterior, el modelo general se divide en 3 grandes bloques que facilitan la explicación y el entendimiento del sistema. Estos bloques son los mismos subsistemas que se presentaron en el apartado anterior, pero con diferencias en su interior:

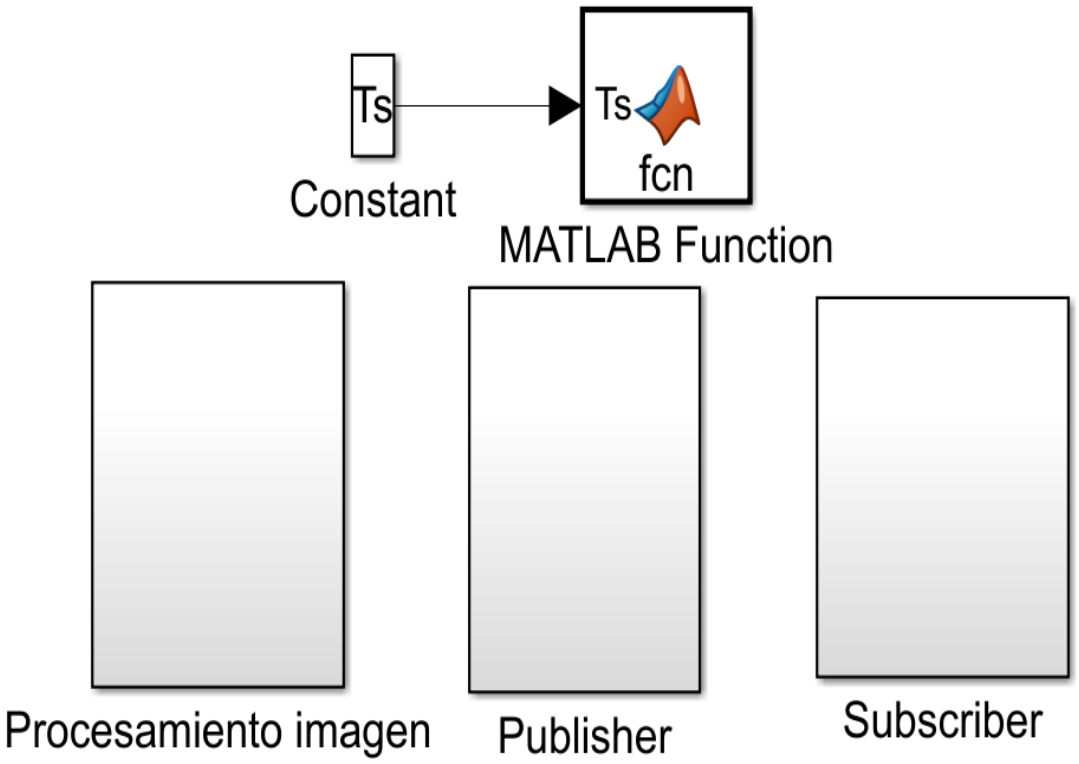


Figura 4.3.4. Imagen general modelo Simulink (Control_imagen.slx).

En este apartado la función pause de sincronismo se ha ubicado en el diagrama más externo pero el funcionamiento es exactamente el mismo que en el apartado anterior, diferenciando el valor por el cual se divide la constante Ts, siendo en este caso Ts/10 ya que para este sistema es más difícil que Matlab simule en un tiempo semejante al tiempo real del simulador.

3.1. SUBSCRIBER.

Este subsistema realiza exactamente las mismas funciones que el bloque con el mismo nombre descrito en el apartado 2.1. con la salvedad de que, en este caso, los datos de posición no van a ser utilizados para el control del dron, así como el ángulo yaw o de rotación. Esta información se va a obtener mediante el tratamiento de la imagen que recoge el sensor de visión del dron.

Por otra parte, como el dron recoge datos en 2D del sensor de visión, no se pueden obtener los otros dos ángulos de Euler a partir de esa imagen, es por esto por lo que el Pitch y el Roll sí que se utilizan para el controlador, asumiendo que son datos que nos proporcionaría la IMU del dron, elemento que es muy común en estos dispositivos.

De esta forma, la única diferencia entre ambos modelos de Simulink estaría en qué se hace con los datos recibidos del *topic /vrep/quadrotor_0/pose*:

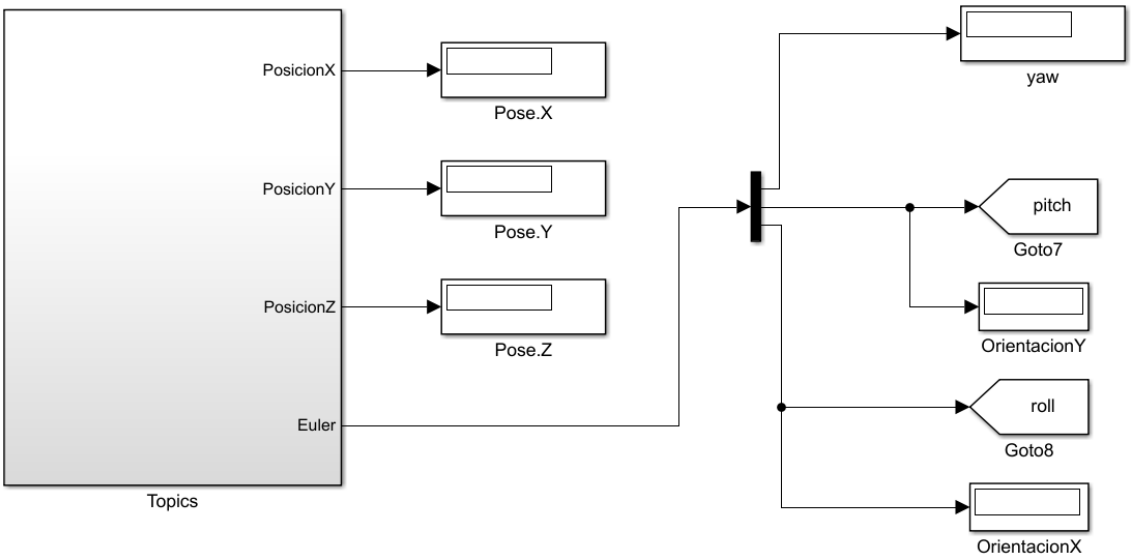


Figura 4.3.5. Subsistema Subscriber.

Corroborando lo anterior, en la figura 4.3.5, únicamente se etiquetan los ángulos pitch y roll, que van a ser utilizados posteriormente, mientras que el resto de componentes del mensaje recibido solo se muestran por pantalla para posibles comprobaciones durante la simulación.

3.2. PROCESAMIENTO IMAGEN.

Este apartado trata de la adquisición en Simulink de los datos del sensor de visión que contiene el dron. Este proceso es idéntico al explicado en el bloque ‘‘Subscriber’’, ya que las herramientas a utilizar y el fin son los mismos.

Lo primero que hay que hacer es saber en qué *topic* se publican los datos de la imagen que se desea obtener, y para ello, se vuelve a la lista mostrada por el *rostopic list* del apartado 2.1:

```
oscarcanre10@oscarcanre10-X555LD:~$ rostopic list
/clock
/cmd_vel
/rosout
/rosout_agg
/tf
/velodyne_points
/vrep/Vision_sensor_0
/vrep/Vision_sensor_0/CameraInfo
/vrep/Vision_sensor_0/compressed
/vrep/Vision_sensor_0/compressed/parameter_descriptions
/vrep/Vision_sensor_0/compressed/parameter_updates
/vrep/Vision_sensor_0/compressedDepth
/vrep/Vision_sensor_0/compressedDepth/parameter_descriptions
/vrep/Vision_sensor_0/compressedDepth/parameter_updates
/vrep/Vision_sensor_0/theora
/vrep/Vision_sensor_0/theora/parameter_descriptions
/vrep/Vision_sensor_0/theora/parameter_updates
/vrep/base/pose
/vrep/camera_info
/vrep/end_eff/pose
/vrep/imu
/vrep/info
/vrep/quadrotor_0/command
/vrep/quadrotor_0/pose
/vrep/quadrotor_0/twist
/vrep/viper_0/jointCommand
/vrep/viper_0/jointStatus
```

Figura 4.3.6. Listado de topics.

Como se ve en la figura, hay varios *topics* relacionados con el sensor de visión, pero el que contiene toda la información que se necesita es el primero: */vrep/Vision_sensor_0.*

```
oscarcanre10@oscarcanre10-X555LD:~$ rostopic info /vrep/Vision_sensor_0
Type: sensor_msgs/Image

Publishers:
 * /vrep (http://oscarcanre10-X555LD:43967/)

Subscribers: None
```

Figura 4.3.7. Información del topic /vrep/Vision_sensor_0.

Una vez que conocemos el tipo del mensaje y el *topic* que publica la información, se crea el siguiente modelo de bloques en Simulink:

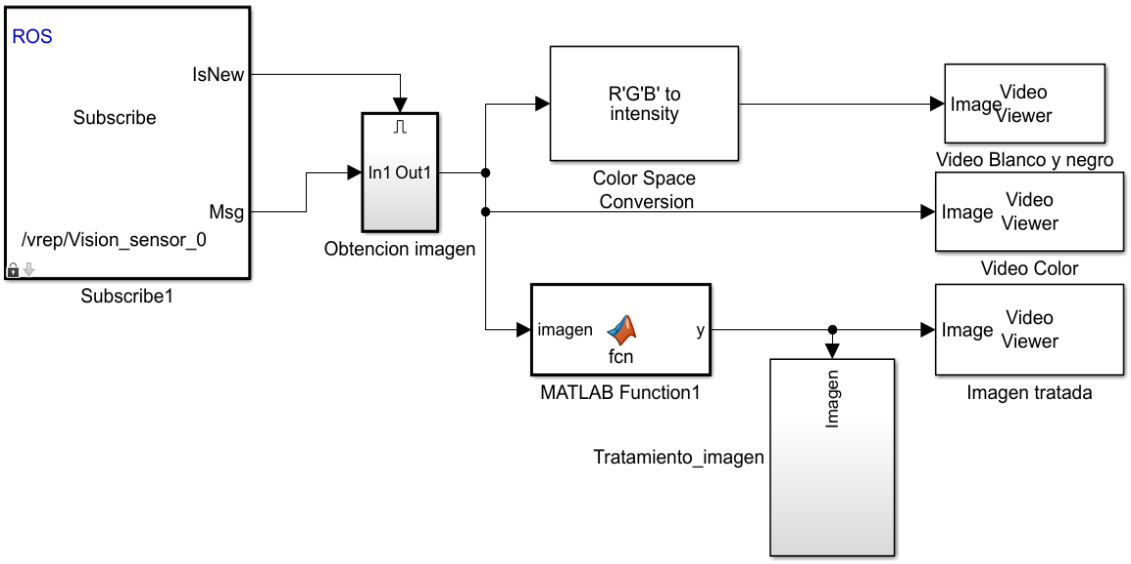


Figura 4.3.8. Subsistema Procesamiento imagen.

El modelo debe incluir un bloque *Subscribe* y un *Bus selector* (el cual se encuentra en el subsistema ‘‘Obtencion imagen’’. De manera que el bloque *Subscribe1* se configura así:

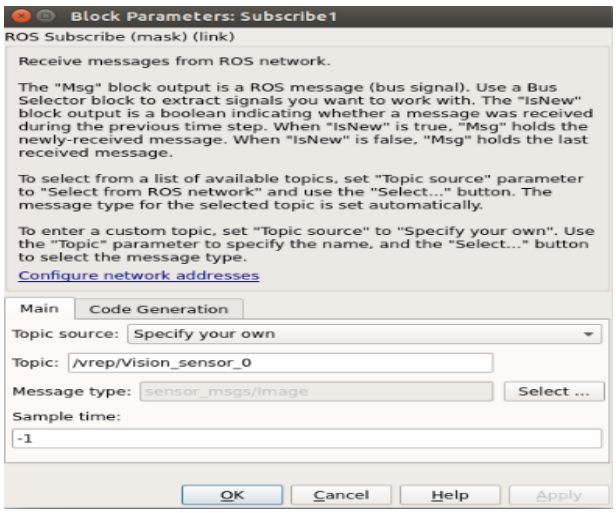
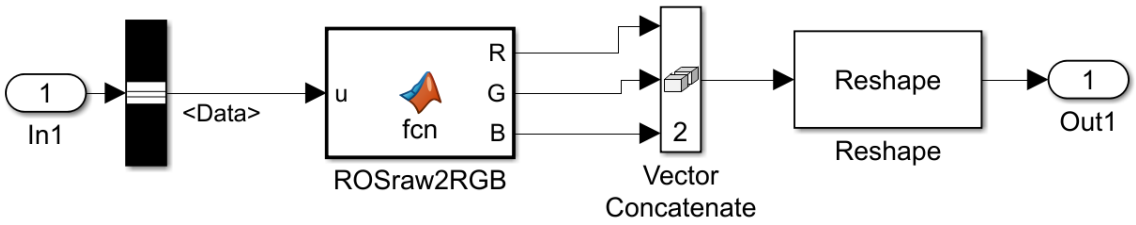


Figura 4.3.9. Configuración del bloque Subscribe1.

Obtención imagen.

Este bloque contiene lo necesario para separar la imagen del resto del mensaje genérico que se publica en el *topic* y adaptar dicha imagen reajustando su tamaño y su formato para que pueda ser tratada en Simulink de forma correcta y eficiente.



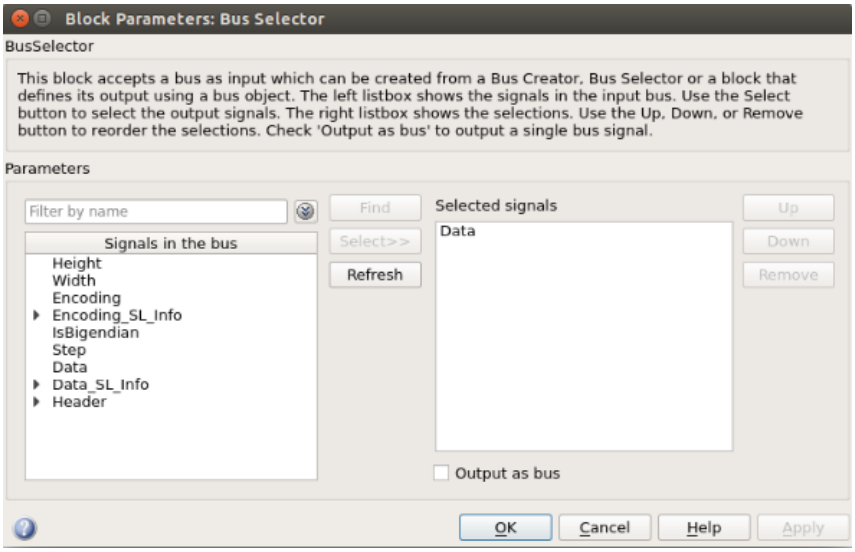


Figura 4.3.10. Subsistema Obtención Imagen y configuración del Bus Selector.

En la imagen de la figura 4.3.10, se muestran los campos que son recibidos a partir del *topic* del sensor de visión. De todos ellos, únicamente se utiliza el campo *Data* el cual tiene, por ello se separa del resto del mensaje.

Cuando el campo *Data* se separa del resto del mensaje, se hace pasar por el bloque *ROSraw2RGB,* el cual se encarga de separar la imagen en sus componentes RGB. Esto es necesario, ya que los datos que salen del *Bus selector* están en forma matricial de una única fila, de manera que las componentes se presentan intercaladas unas con otras siguiendo el patrón RGB. Después, estos vectores separados, se vuelven a concatenar formando una imagen legible, y posteriormente se reajusta su tamaño a 640x480.

3.2.1. Base teórica.

Antes de realizar el modelo de bloques en Simulink, hay que coseguir un método a partir del cual se pueda obtener lo necesario para poder formar el mensaje *effort* del *topic* en el que se tiene que publicar para que el dron realice los movimientos adecuados. De

manera que se deben calcular los torques en X e Y, el thrust y el yaw del dron, para lo que se ha utilizado lo siguiente:

- Desplazamiento en los ejes X e Y (Torques).

Para poder ordenar al dron que se mueva, primero hay que saber hacia dónde tiene que hacerlo, y hay que saberlo a partir de la imagen recibida del sensor de visión. Por lo tanto, se parte de la siguiente situación:

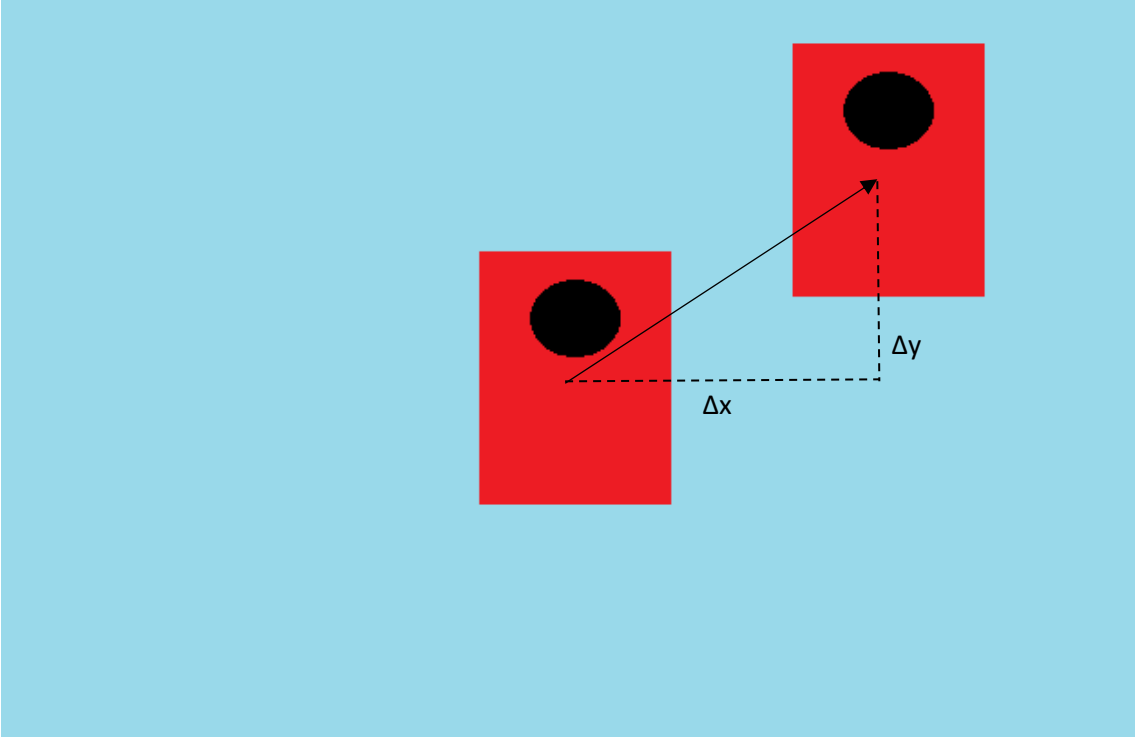


Figura 4.3.11. Desplazamiento en los ejes X e Y.

En la figura 4.3.11 se muestra el desplazamiento de la plataforma en los ejes X e Y desde la vista del dron. Para conocer el incremento de píxeles que se ha producido, se debe calcular el centro del rectángulo y fijar un sistema de referencia, que va a ser el centro de la imagen. De esta forma, con las coordenadas del centro del rectángulo desplazado y las coordenadas del centro de la imagen se calculará el desplazamiento para realizar el controlador de las dos primeras componentes del mensaje *effort*.

➢ Rotación sobre el eje Z (Yaw).

Como ayuda para poder calcular este ángulo de giro se ha incluido en la plataforma el círculo negro, a forma de referencia:

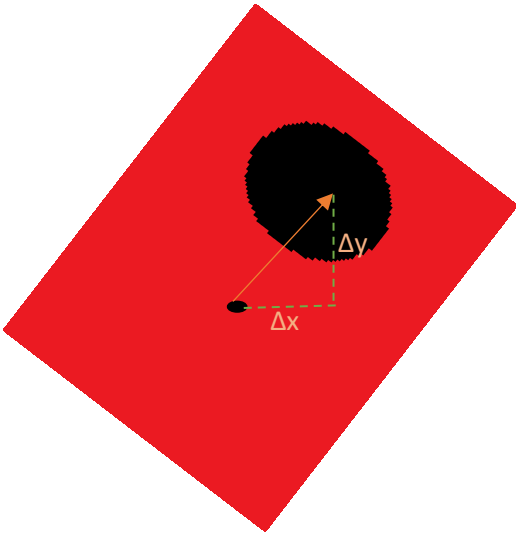


Figura 4.3.12. Rotación sobre el eje Z.

En la figura 4.3.12 se muestra un giro sobre el eje Z de la plataforma, por lo que a partir de la posición del centro del círculo y del centro del rectángulo, se obtendrán las componentes en X e Y del vector distancia entre ambos puntos, necesarios para calcular el ángulo de rotación a corregir por el dron de la siguiente manera:

$$Yaw = atg(\frac{\Delta x}{\Delta y})$$

➢ Desplazamiento en el eje Z (Thrust).

Para el desplazamiento en el eje Z también es necesario apoyarse en la imagen suministrada por el dron, de manera que, cuando el dron asciende, el área del rectángulo que forma la plataforma desciende; y si el dron cae, esta área asciende. Es esta área lo que se utiliza para controlar el ascenso y descenso del cuadricóptero.

3.2.2. Modelado del sistema.

En este apartado se explica cómo se ha puesto en práctica lo contado en el punto anterior en Simulink, de manera que se obtenga lo necesario para poder realizar el lazo de control sobre las 4 componentes del dron.

Lo primera parte necesaria para poder utilizar la imagen con los bloques de Simulink, es hacer pasar la imagen de salida del subsistema ‘‘Obtencion imagen’’ por el bloque *Matlab function*. Este bloque, realiza un tratamiento de la imagen a partir de un *script* asociado, el cual tiene el siguiente código fuente:

```
function y = fcn(imagen)
    S=zeros(480,640);
    media_rojo=0.9961;%Obtenido mediante entrenamiento
    media_verde=0;%Obtenido mediante entrenamiento
    media_azul=0;%Obtenido mediante entrenamiento
    media=[media_rojo,media_verde,media_azul];
    umbral=0.2;
    imagen2=im2double(imagen);
    for i=1:1:(480) %Para todas las columnas de la imagen
    for j=1:1:(640) %Para todas las filas de la imagen
    red=imagen2(i,j,1);
    green=imagen2(i,j,2);
    blue=imagen2(i,j,3);
    media2=[red, green, blue];
%Se calcula la distancia entre el vector objeto y el del pixel
    d=norm(media-media2);
    if d<umbral
    S(i,j)=1;
    end
    end
    end
y=S;
end
```

Así, la imagen de entrada es segmentada por color, de modo que la parte importante de la imagen resultante es el rectángulo de la plataforma como se muestra a continuación:

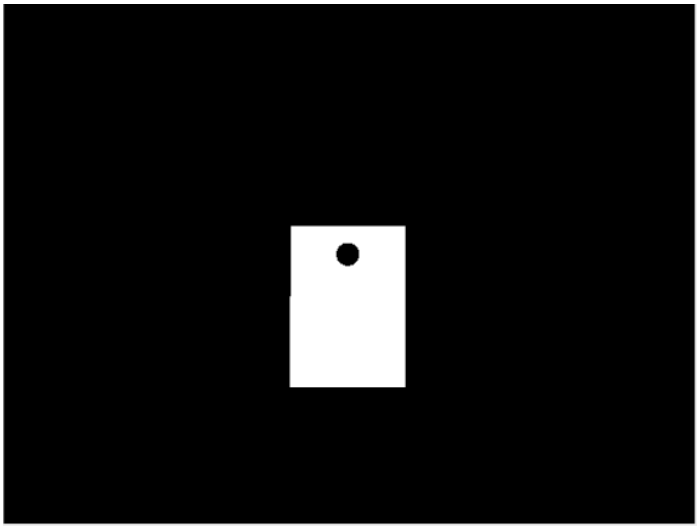


Figura 4.3.13. Imagen segmentada de la plataforma.

De esta forma, en el bloque que le sucede, puede tratarse la imagen obteniendo únicamente la información necesaria de la plataforma, tanto del rectángulo, como del circulo que contiene.

Como se ha explicado en el punto anterior, se debe obtener la posición en pixeles X e Y del centro del rectángulo, el ángulo de giro del centro del círculo respecto del centro del rectángulo, y el área que ocupa dicho rectángulo en la imagen. Todo esto se realiza en el subsistema ‘‘Tratamiento_imagen’’.

Tratamiento imagen.

El modelo de bloques en Simulink de este subsistema es el siguiente:

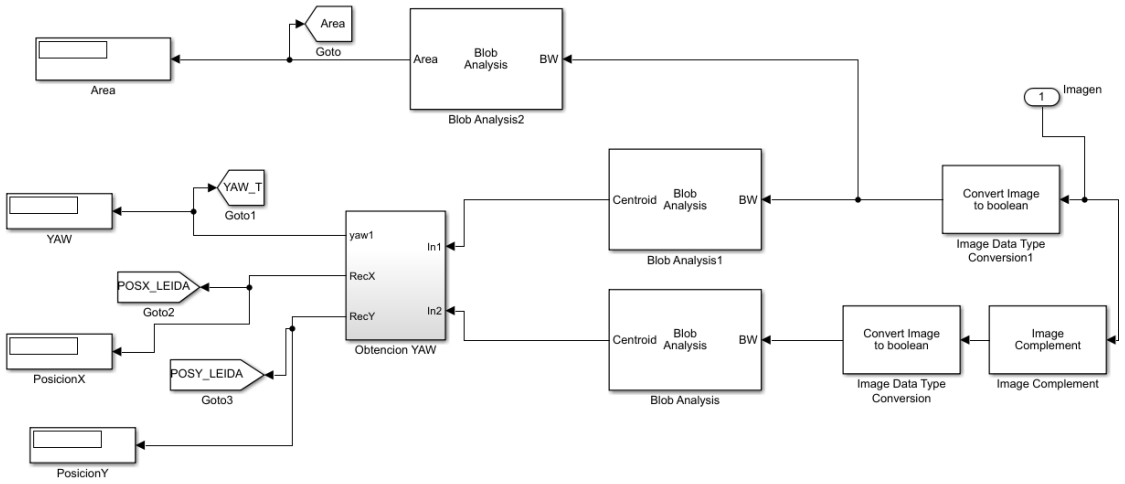


Figura 4.3.14. Subsistema Tratamiento imagen.

Como se ve en la figura 4.3.14, la imagen, tanto siguiendo un camino como otro, se convierte a booleana, ya que es necesario para poder utilizar los bloques sucesivos.

Mientras que la imagen segmentada del rectángulo ya se ha obtenido mediante el bloque anterior, para el círculo hay que obtener su complementaria, añadiendo además información de otro objeto adicional, que sería el fondo de la imagen.

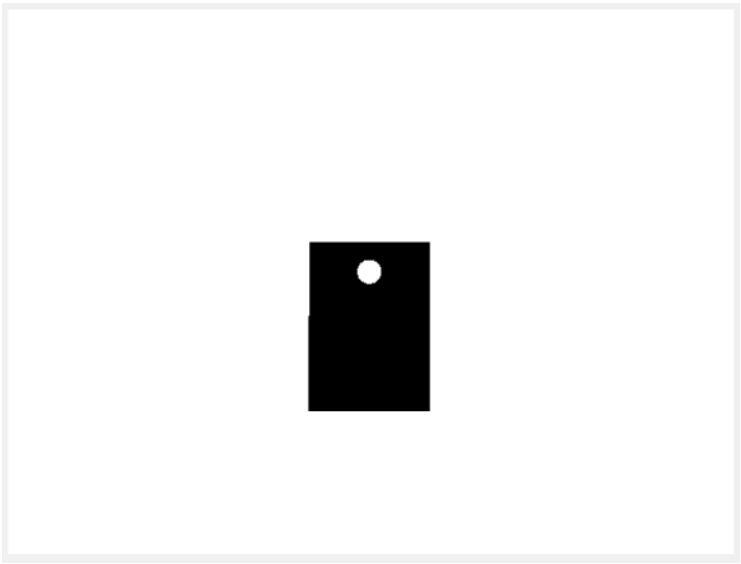


Figura 4.3.15. Imagen complementaria de la segmentación.

El bloque de Simulink *Blob Analysis* va a ayudar mucho en la labor de conseguir todo lo que se ha indicado anteriormente como necesario:

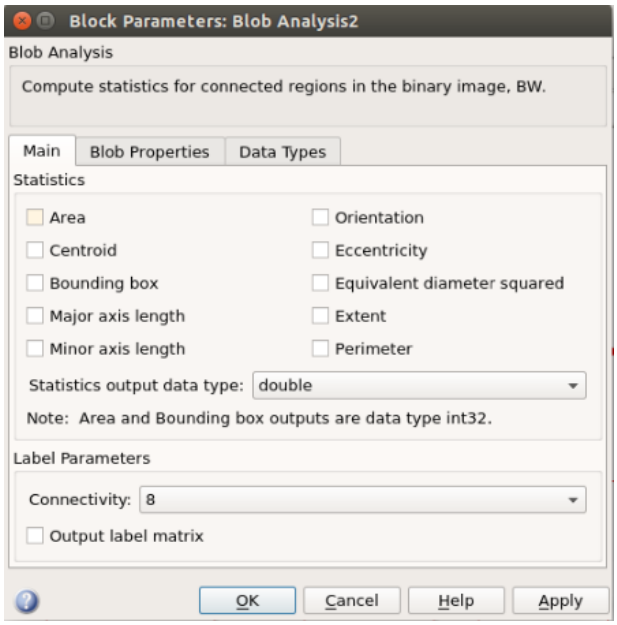


Figura 4.3.16. Configuración del bloque Blob Analysis.

Por tanto, de manera directa ya se puede obtener el área y el centroide del rectángulo siguiendo el primer camino, y el centroide de los objetos resultantes después de complementar la imagen siguiendo el segundo. Estos centroides se hacen pasar por otro subsistema, ‘‘Obtencion YAW’’.

➢ Obtención YAW.

Este subsistema tiene como entradas el centroide del rectángulo y los centroides de la imagen complementaria.

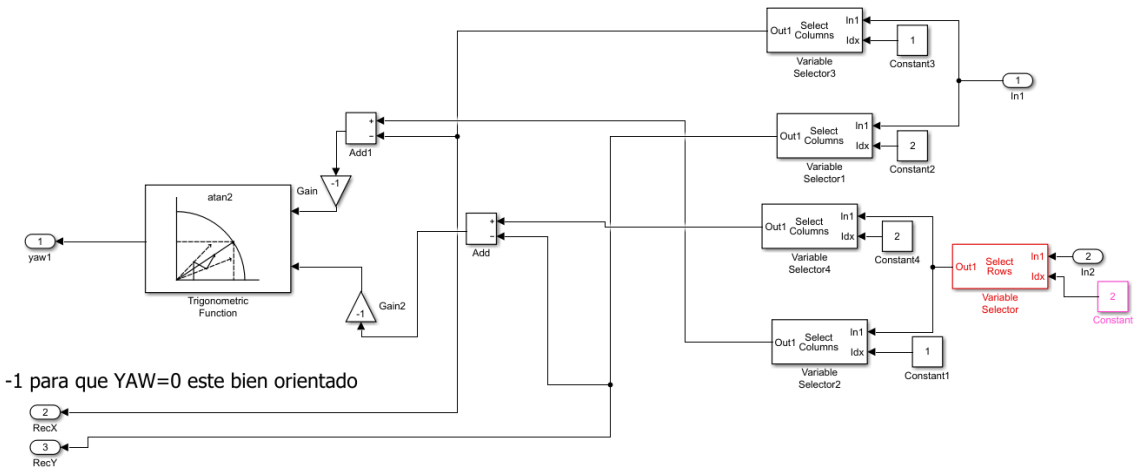


Figura 4.3.17. Subsistema Obtencion YAW.

A partir del centroide del rectángulo, ya se obtienen de forma directa las coordenadas que se necesitan, únicamente se separan las columnas del vector centroide mediante el bloque *Select Columns.* Estas dos componentes son 2 de las 3 salidas del subsistema.

Para obtener el centroide del círculo, hay que diferenciar los dos objetos dentro de la imagen complementaria obtenida, de modo que el vector centroide que tiene como entrada este subsistema, contiene información tanto del círculo, como del fondo de la imagen. Es por esto por lo que se utiliza *Select Rows* para separar ambos centroides y seleccionar el que realmente interesa. Una vez realizado esto, también se utiliza *Select Columns* para contar con las coordenadas X e Y del centroide de forma separada.

Una vez que se obtienen tanto las coordenadas del centro del rectángulo como del centro del círculo, se procede a restar las componentes X e Y respectivamente, de forma independiente, siguiendo la fórmula para el cálculo del ángulo:

$$Yaw = atg\left(\frac{\Delta x}{\Delta y}\right); siendo\ \Delta x = Xc - Xr\ y\ \Delta y = Yc - Yr$$

Para que el ángulo que resulte esté bien orientado, es decir, si la plataforma está perfectamente orientada el ángulo calculado sea 0, hay que hacer ambos resultados negativos.

Finalmente se hacen pasar ambas componentes calculadas por el bloque *Trigonometric Function* configurada como *atan2*, consiguiendo como salida el ángulo yaw del dron.

3.3. PUBLISHER.

Este subsistema tiene la función de formar el mensaje a enviar a V-REP mediante ROS para que el dron realice los movimientos adecuados en función del movimiento de la plataforma. Tiene la misma función que en la primera aplicación, pero los medios a través de los cuales se consigue hacerlo son distintos.

Para ello, se crean los 4 controladores necesarios para obtener el thrust, el yaw y los torques en X e Y. Estos lazos de control se implementan en los subsistemas **Torques**, **YAW** y **Thrust.**

Puesto que ya se ha obtenido el área del rectángulo formado por la plataforma, su centroide y el ángulo de giro, el siguiente paso es utilizar estos datos para realizar los lazos de control necesarios para poder formar el mensaje *effort* y poder mandarlo en el *topic command*. Para ello, el subsistema ‘‘Publisher’’ sigue el modelo siguiente:

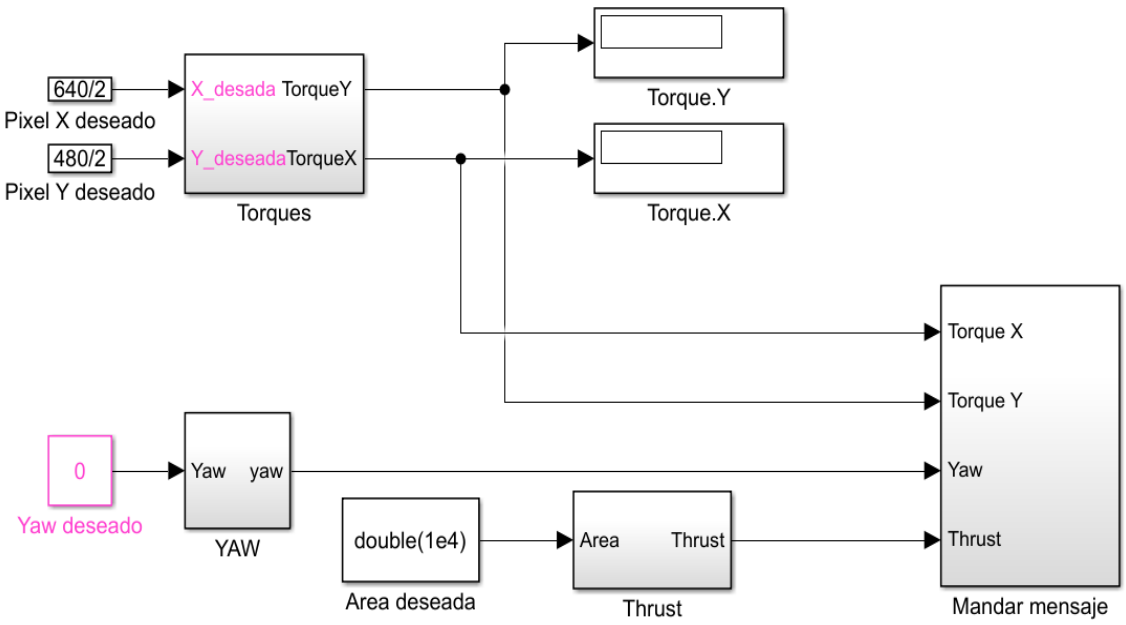


Figura 4.3.18. Subsistema Pusblisher.

Si se compara con el modelo de la aplicación anterior para este subsistema, es prácticamente idéntico. El subsistema ‘‘Mandar mensaje’’ es el único que realmente es el mismo que el explicado con anterioridad, mientras el contenido de los subsistemas que calculan sus entradas cambia de forma significativa. Las entradas a dichos subsistemas son los objetivos a los que tiene que aproximarse el controlador.

Control del thrust.

El objetivo de este subsistema es conseguir que el área de la plataforma que vea el dron sea de un determinado valor, el cual se indica como entrada al bloque. Que el dron vea una determinada área implica que esté a una determinada altura, por lo que, si el área es grande, el dron estará cerca de la plataforma; mientras que, por el contrario, si el área es pequeña, el dron estará a una altura considerable del dron. Esto en simulación no tiene limitaciones ni inferiores ni superiores, es decir, el dron puede descender todo lo que pueda y puede ascender de la misma manera. Es por esto por lo que las limitaciones le son impuestas, de tal manera que, si el dron desciende mucho, golpeará contra el suelo o la plataforma, lo que conlleva una pérdida de control del dispositivo; mientras que, si asciende demasiado, el sensor de visión deja de recoger datos y provoca que se pierda la referencia de control.

El subsistema ‘‘Thrust’’ tiene como entrada el área calculada en cada instante y como salida el empuje que se debe aplicar al dron. El modelo del controlador es el siguiente:

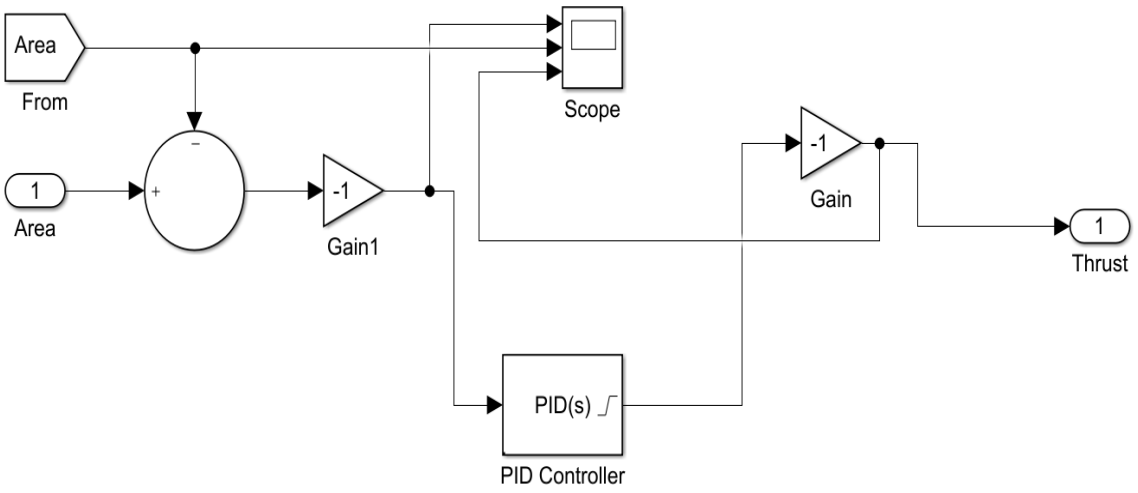


Figura 4.3.19. Subsistema Thrust.

Cuando el área leída es menor que el área objetivo del dron, se produce un error positivo, pero en esta situación el dron debería descender, por tanto, al controlador PID debe entrar cambiado de signo, volviendo a cambiar en su salida. El bloque PID se configura, siendo KTP=12.7:

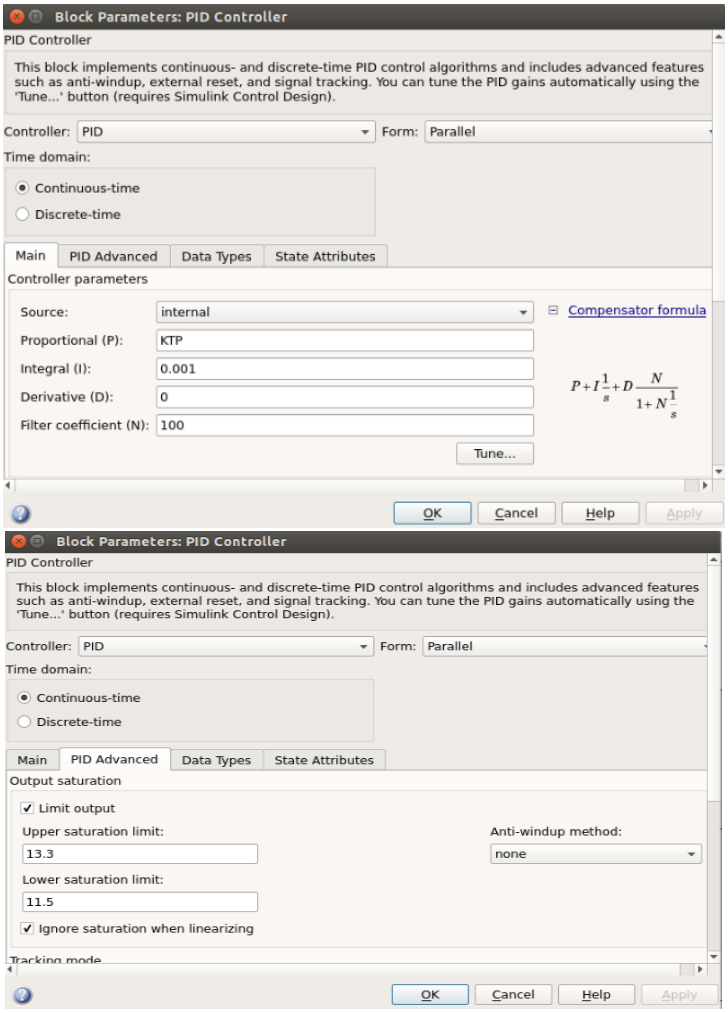


Figura 4.3.20. Configuración del PID.

Como ejemplo de funcionamiento del control de altura, se muestra una prueba de lo realizado por el controlador, cuando se varía el área objetivo que debe ver el sensor de visión del dron.

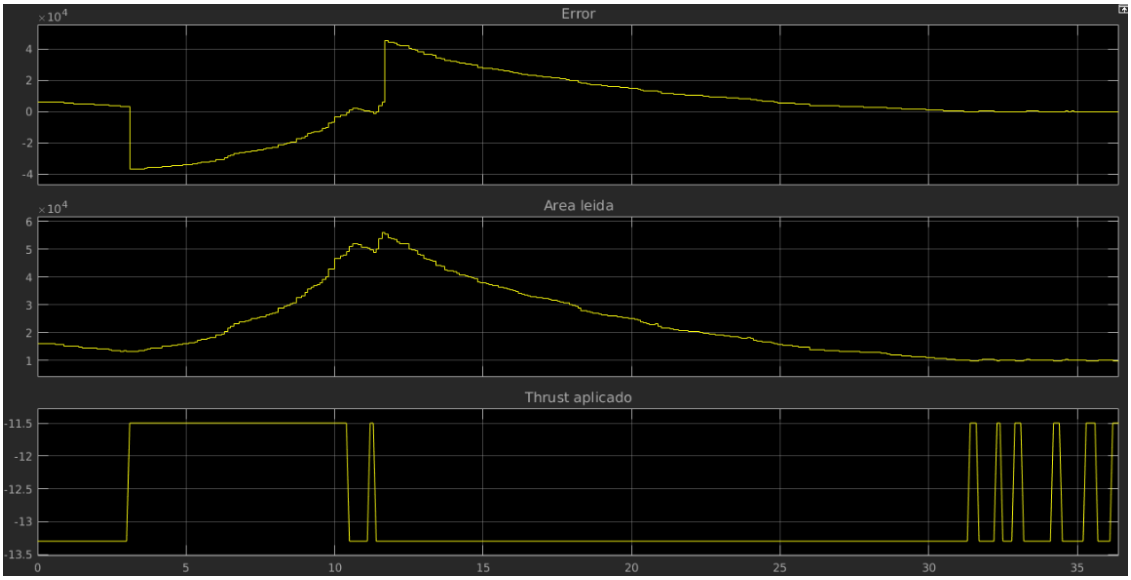


Figura 4.3.21. Scope del lazo de control del thrust.

En la figura 4.3.21 se muestra un caso en el que el área que el dron debe ver pasa de ser estable, con un valor de 10^4 unidades cuadradas a $5*10^4$, de modo que el dron debe disminuir su altura. Cuando se estabiliza en altura, se vuelve de nuevo a la misma área inicial. El funcionamiento de este control provoca más inestabilidad que en la aplicación anterior, debido a que el cálculo del área es menos exacto, y es prácticamente imposible que coincide la unidad de área medida con el objetivo. Por tanto, la componente thrust del mensaje *effort* ya se puede enviar a V-REP.

Control del yaw.

Este subsistema tiene como entrada una constante de valor 0, ya que es el ángulo (en rad/s) que provoca que el dron se encuentre bien alineado con la plataforma. Mientras que el contenido del subsistema es el mismo que en el apartado del control absoluto de posición.

Como comprobación de funcionamiento, se muestra una prueba de manera que, partiendo de la situación inicial con el dron correctamente orientado respecto a la plataforma, este se gira un cierto ángulo α, y el controlador actúa sobre el dron haciendo que recupere la posición deseada:

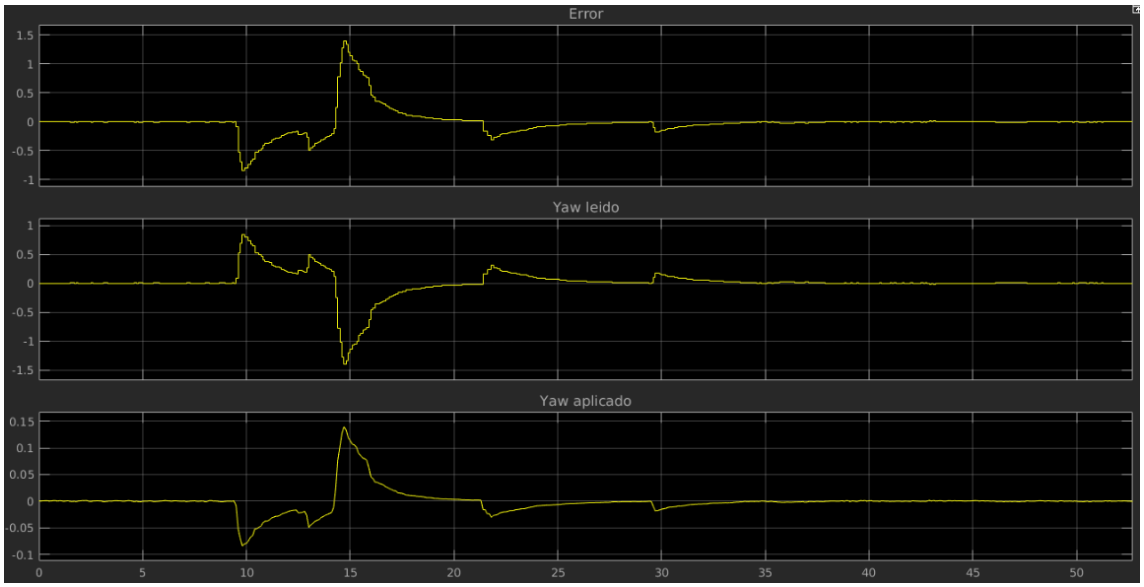


Figura 4.3.22. Scope del lazo de control del yaw.

Como se ve en la gráfica de la figura 4.3.22, el ángulo inicial es de 0 rad/s hasta que, en un determinado momento, la plataforma es girada hasta X rad/s. Es entonces cuando el controlador realiza su función, provocando la rotación del dron sobre su eje Z, hasta recuperarse de ese cambio y llegar de nuevo a los 0 rad/s. Por tanto, la tercera componente del mensaje *effort* ya ha sido obtenida de forma correcta.

Control de los torques X e Y.

En este caso, el subsistema tiene dos entradas, que son las posiciones destino del dron dentro de la imagen que se recibe del sensor de visión. Esto es, una vez que la plataforma se desplace, tendrá un centroide distinto al centro de la imagen, por lo tanto, se tiene que hacer lo necesario para que esa plataforma, vuelva a ser vista en el centro de la imagen. Para ello se modela el sistema de la siguiente manera:

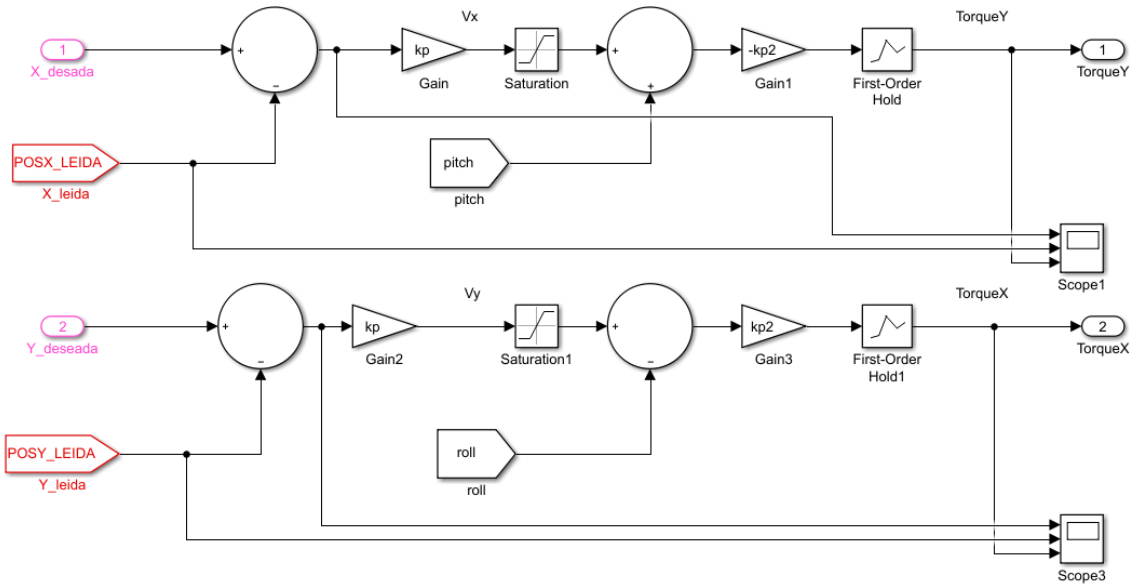


Figura 4.3.23. Subsistema Torques.

En este caso, el control de cada torque se divide en dos controladores de la misma forma que se hacía en la aplicación del GPS.

Para la obtención del torque en el eje X, primero se tiene en cuenta la coordenada Y obtenida anteriormente y la coordenada Y deseada. El error obtenido a partir de ellas se pasa por controlador de tipo P, con una constante kp=0.001, y un saturador, el cual evita que el error siga creciendo de forma incontrolada. Posteriormente se compara con el pitch, que sigue siendo obtenido gracias a la información que suministra V-REP del dron. Ese error vuelve a pasarse por otro controlador P con constante -kp2=-0.5, obteniendo finalmente el torque en X que se aplica al mandar el mensaje.

El primer lazo de control para el cálculo del torque en el eje Y se realiza de la misma manera que el anterior, pero utilizando las coordenadas en el eje X. Es en el segundo lazo de control donde se encuentra la diferencia, ya que, en esta ocasión, la salida del saturador se compara con el roll del dron y la constante del kp2 del segundo controlador tiene un valor positivo. Para comprobar el funcionamiento de este último subsistema, se lanza la simulación y se provoca movimiento en la plataforma a través de V-REP, de manera que el controlador debe actuar y seguir a la plataforma desplazándose en los ejex X e Y.

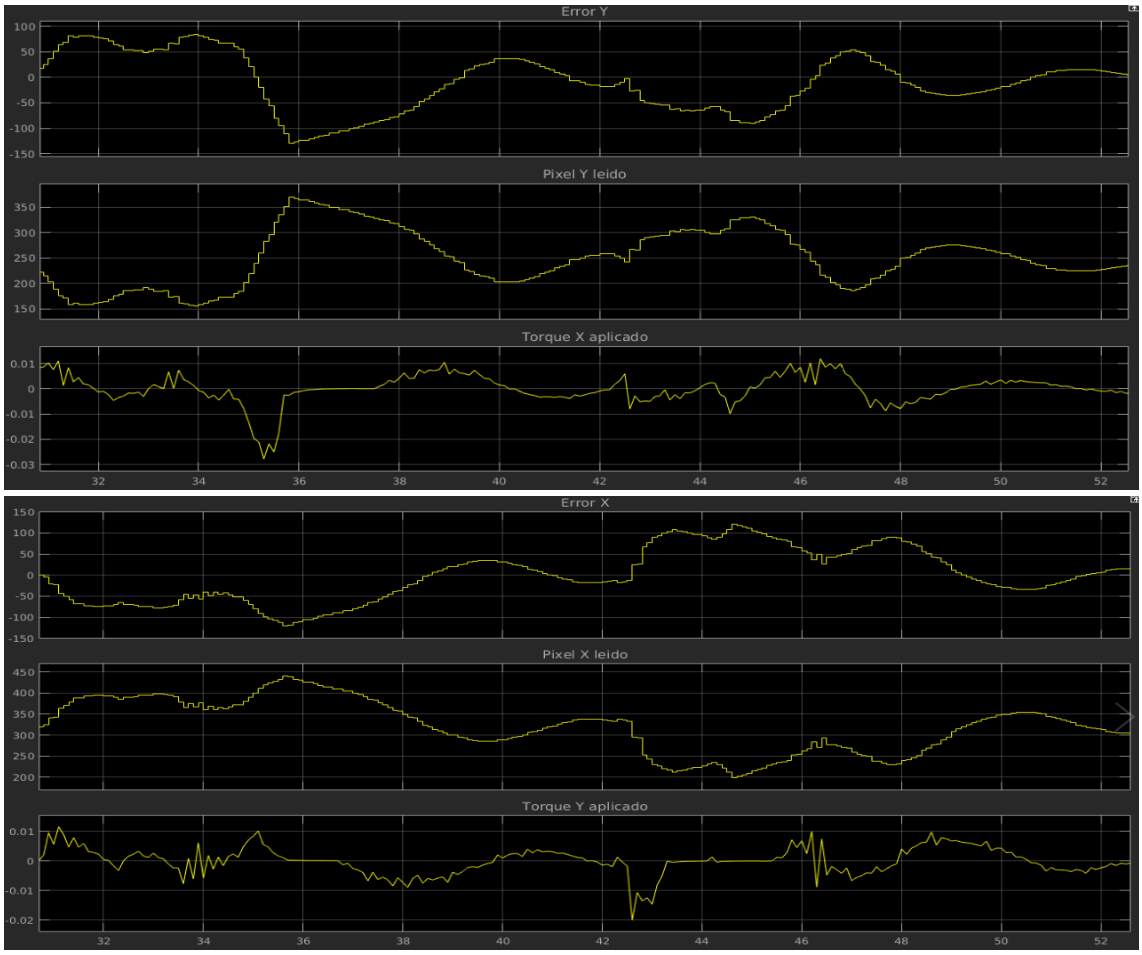


Figura 4.3.24. Scope del lazo de control de los torques.

La actuación del controlador no debe ser muy precisa respecto a la posición sobre la plataforma, lo verdaderamente importante es que no se pierda. En la figura anterior se muestra la corrección que se realiza sobre las posiciones en X e Y del dron para alinearse con la plataforma.

Como queda demostrado en las gráficas, la posición leída de ambas coordenadas es corregida a la perfección, haciendo que el error se vaya reduciendo. En este caso, el movimiento que se ha producido en la plataforma ha sido en ambos ejes a la vez. Para que quede más claro su correcto funcionamiento, se muestra a continuación un ejemplo en el que el desplazamiento de la plataforma se produce de forma independiente:

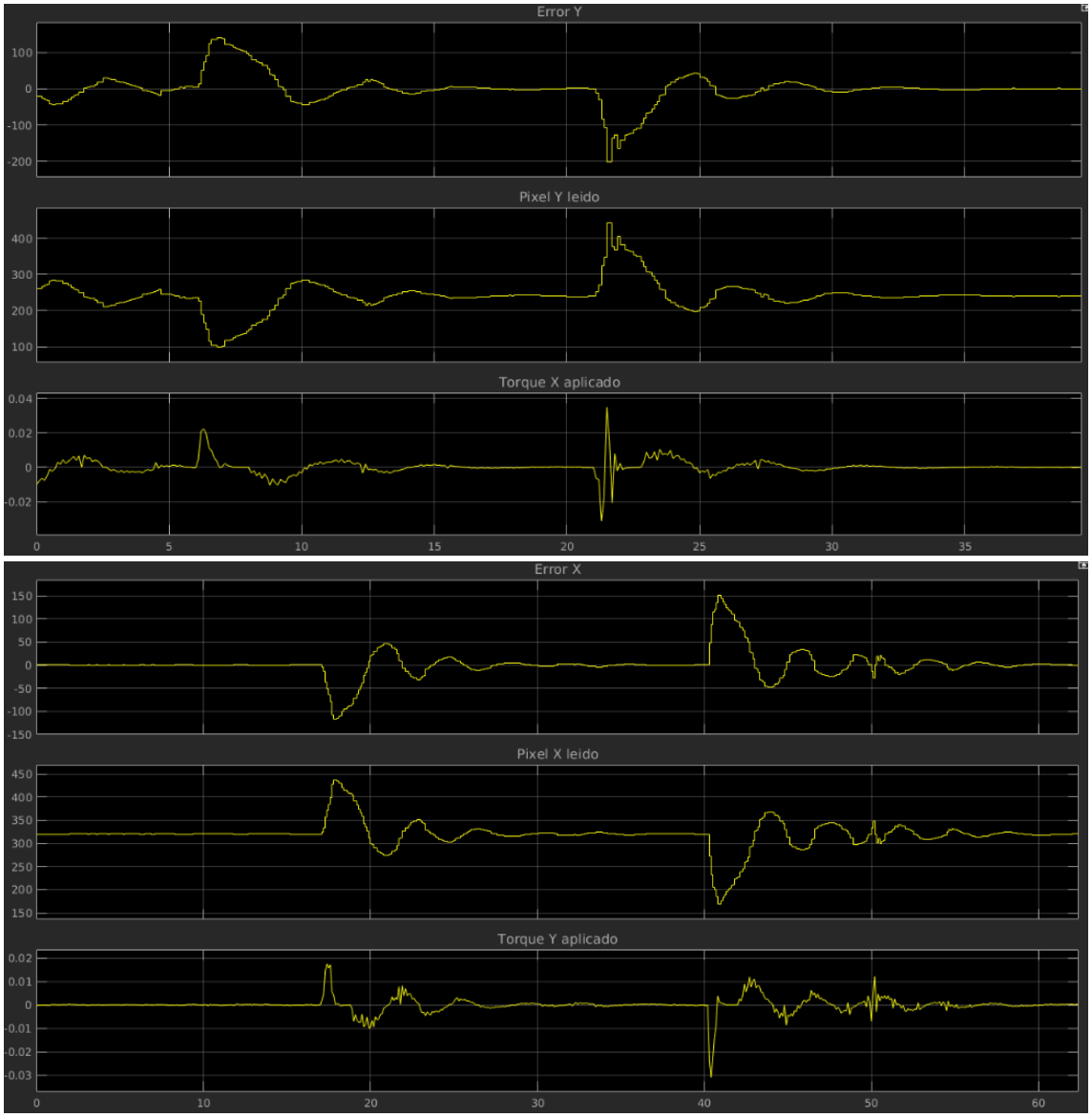


Figura 4.3.25. Scope del lazo de control de los torques independientes.

En esta última figura, se observa que el error cero tarda en conseguirse, pero no es algo relevante debido a que el dron sigue a la perfección a la plataforma, y ese error, en la imagen que se obtiene del sensor de visión del cuadricóptero, es prácticamente despreciable. En la parte de arriba de la figura se ha desplazado la plataforma a lo largo del eje Y, produciéndose correctamente la recuperación de la posición; mientras que, en la parte de abajo, se desplaza sobre el eje X, siendo correcta la corrección.

Por tanto, una vez acabado este apartado, ya se tiene todo lo necesario para poder enviar el mensaje completo a través del entrono ROS al simulador V-REP.

4. SEGUIMIENTO DE UN ROBOT MÓVIL.

Lo próximo que se va a realizar es la sustitución de la plataforma móvil por un robot, el cual contenga esa misma plataforma. Esto es útil por diferentes motivos:

- El movimiento del robot se puede programar de manera que se automatiza el proceso del movimiento de la plataforma.
- Se asemeja más al fin último, que era el de incluir este proyecto en uno más grande de conducción de vehículos autónoma.
- La verificación del correcto funcionamiento del sistema es visual, de modo que, si el dron sigue al robot, se confirma el funcionamiento.
- Se puede realizar la simulación del despegue y aterrizaje desde la plataforma que incluye el robot móvil.

Por tanto, para esta labor, solamente se debe modificar la escena del V-REP, mientras que el modelo de Matlab, para una primera simulación del funcionamiento de lo expuesto, es el mismo que en el punto anterior.

Para la realización de la escena, va a ser útil, una vez más, ejemplos ya implementados en el simulador. En V-REP hay disponible multitud de ejemplos de robots móviles y se ha probado muchos de ellos para comprobar si cumplen los requisitos de funcionamiento del sistema. Muchos de ellos dejan de funcionar o lo hacen de manera errónea al introducirlos a la escena del dron, debido a que se interrumpen o coinciden en configuraciones, de manera que se interfieren y tanto el dron como el robot no pueden compartir escena.

A pesar de ello, uno de los ejemplos disponibles en los tutoriales de V-REP (*BubbleRob.ttt*) contiene una esfera maciza sobre dos ejes sujetos a dos discos, que hacen las veces de ruedas y motores.

El robot de este ejemplo contiene un sensor, de manera que el avanza a una cierta velocidad hasta que el sensor detecta un obstáculo en su camino. Entonces, retrocede

girando sobre sí mismo durante un cierto tiempo. Transcurrido ese tiempo vuelve a avanzar hasta que se encuentre otro obstáculo y así sucesivamente.

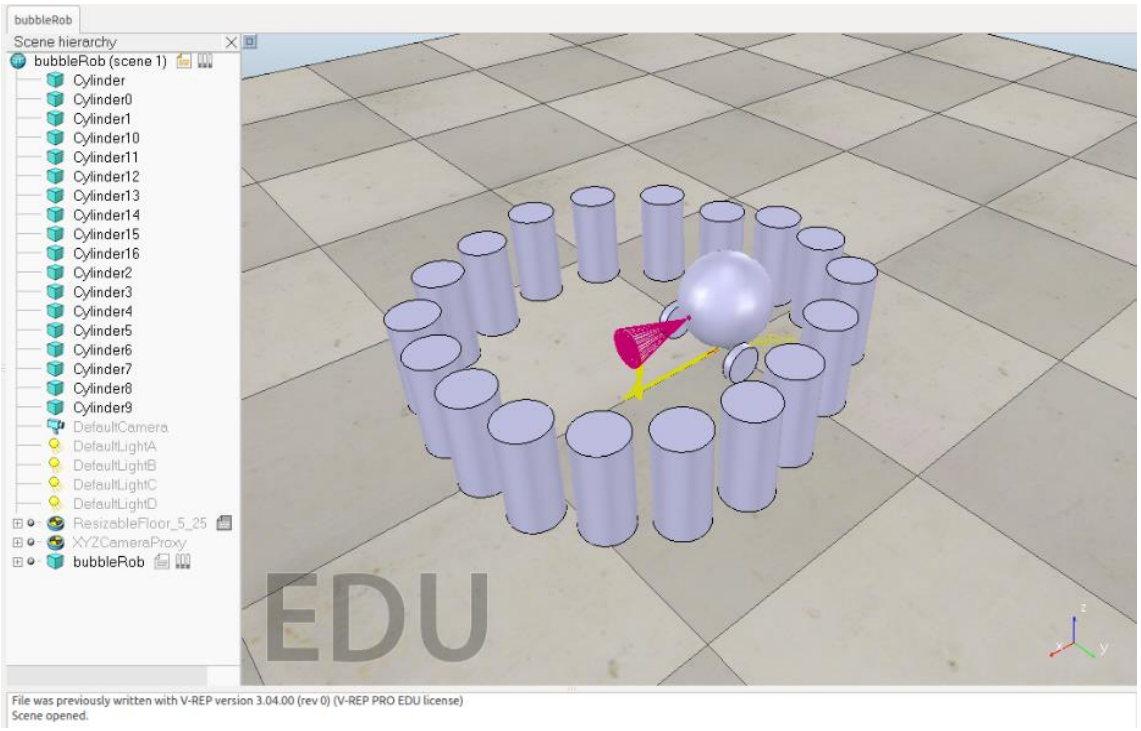


Figura 4.1. Escena V-REP (BubbleRob.ttt).

Lo primero que se hace es eliminar todo lo que no sea el robot de la escena, y se incluye una plataforma, que va a ser sujeta al robot. Lo más importante que se debe modificar de este ejemplo es el *child script* del robot para que se mueva de una manera que sea útil para comprobar el funcionamiento del controlador del dron. La escena resultante en V-REP es la siguiente:

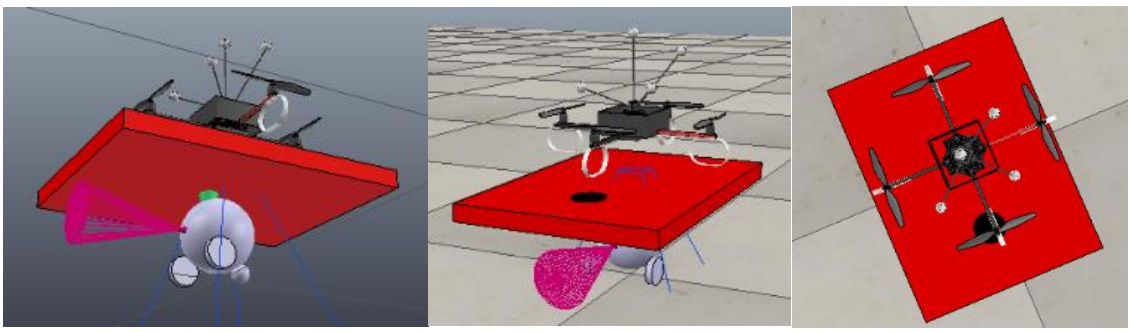


Figura 4.2. Imágenes del conjunto Robot/Plataforma/Dron.

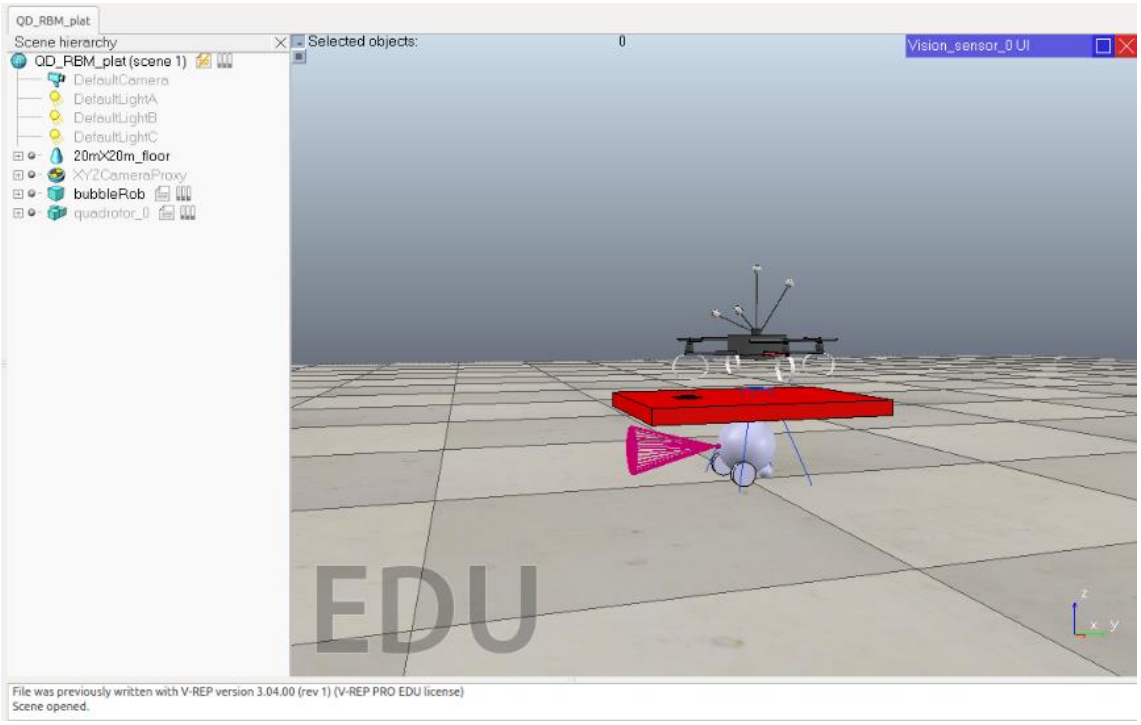


Figura 4.3. Escena V-REP (QD_RBM_plat.ttt).

En la figura 4.2, se muestran diferentes vistas de los objetos que componen la escena. Como se observa en las imágenes, el robot soporta una plataforma en la que el dron se apoya. De manera que el robot se mueve por la escena y permite al dron tomar la imagen de la plataforma para su tratamiento.

Lo que se va a simular en ese caso, también es el despegue del dron desde la plataforma, es por ello por lo que el dron debe ascender hasta alcanzar una cierta altura, y una vez eso haya ocurrido seguir al robot. Por lo tanto, se va a programar el dron de modo que se encuentre prácticamente detenido mientras que el dron empieza a coger altura, y entonces avance, gire, se detenga, etc. Todo esto se realiza dentro del *child script* del robot, añadiendo las siguientes líneas de código a la parte de actuación del *child script*:

```
if ((tiempo1==0)) then
 print('Estado 0')
 print(tiempo1)
 simSetJointTargetVelocity(leftMotor,speed/30)
 simSetJointTargetVelocity(rightMotor,speed/30)
 if((simGetSimulationTime()>N))then
  tiempo1=1
  print('Estoy en el estado 0 y cambio al estado 1')
  N=N+5
 end
end

if (tiempo1==1)then
 print('Estado 1')
 print(tiempo1)
 simSetJointTargetVelocity(leftMotor,speed/3+0.8)
 simSetJointTargetVelocity(rightMotor,speed/3)
 if((simGetSimulationTime()>N))then
  tiempo1=2
  print('Estoy en el estado 1 y cambio al estado 2')
  N=N+10
 end
end
if (tiempo1==2)then
 print('Estado 2')
 print(tiempo1)
 simSetJointTargetVelocity(leftMotor,speed/3+0)
 simSetJointTargetVelocity(rightMotor,speed/3)
 if((simGetSimulationTime()>N))then
  tiempo1=0
  print('Estoy en el estado 2 y cambio al estado 0')
  N=N+5
 end
end
```

Este código hace que el robot siga un patrón de movimiento de manera que en un principio se encuentra prácticamente detenido (no está parado totalmente debido a inferencias con el simulador), acto seguido el robot avanza y en un cierto momento gira y se vuelve a detener; así hasta que se termina la simulación.

Se producen dos cambios en el modelo de Simulink (Nombre.slx) respecto al utilizado en el punto anterior:

- La constante integral del bloque PID en el lazo de control del thrust pasa a tener un valor de 10.
- El límite superior del saturador en el PID del controlador del empuje toma un valor de 14 y el inferior de 11.5.

Para la demostración del correcto funcionamiento de este apartado se han tomado unas capturas del simulador y de la imagen captada en Simulink de manera que se observe el seguimiento del dron sobre el vehículo.

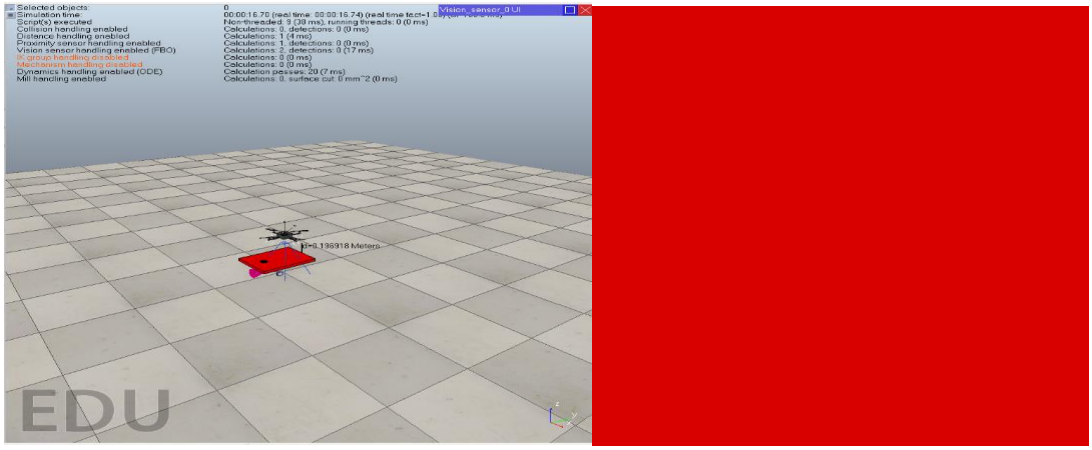


Figura 4.4. Momento 1 simulación.

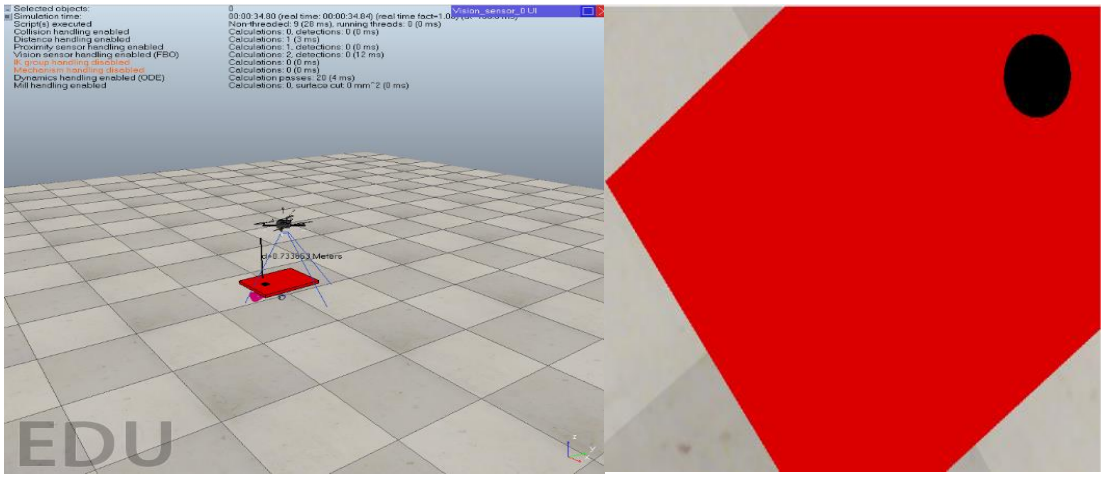


Figura 4.5. Momento 2 simulación.

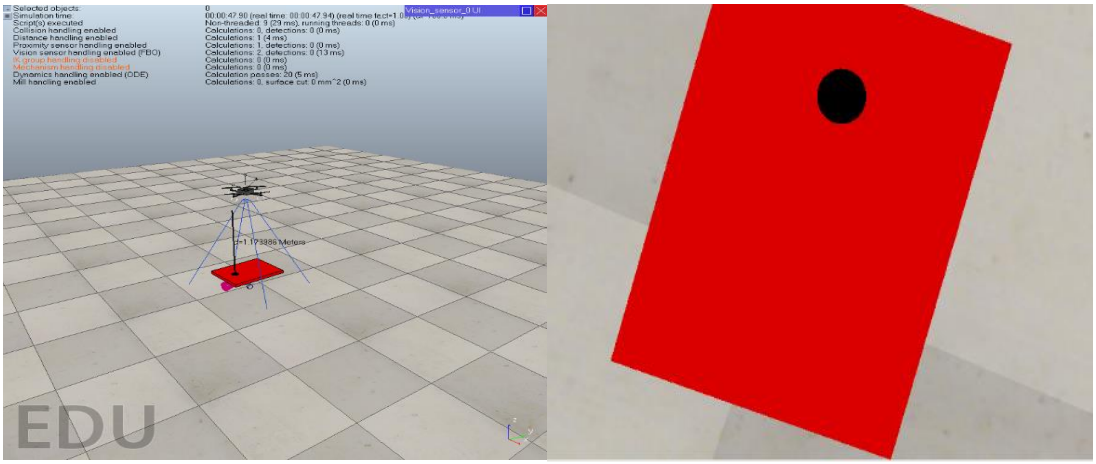


Figura 4.6. Momento 3 simulación.

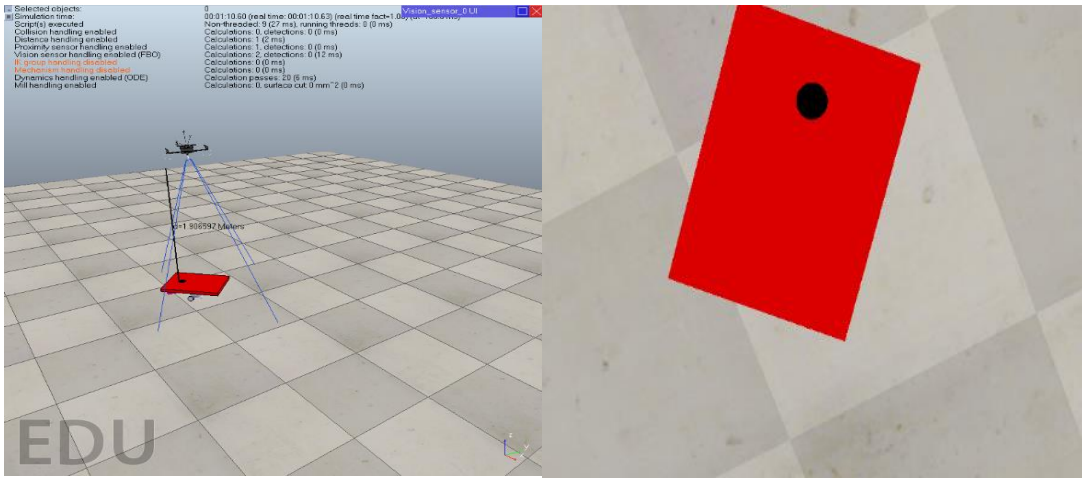


Figura 4.7. Momento 4 simulación.

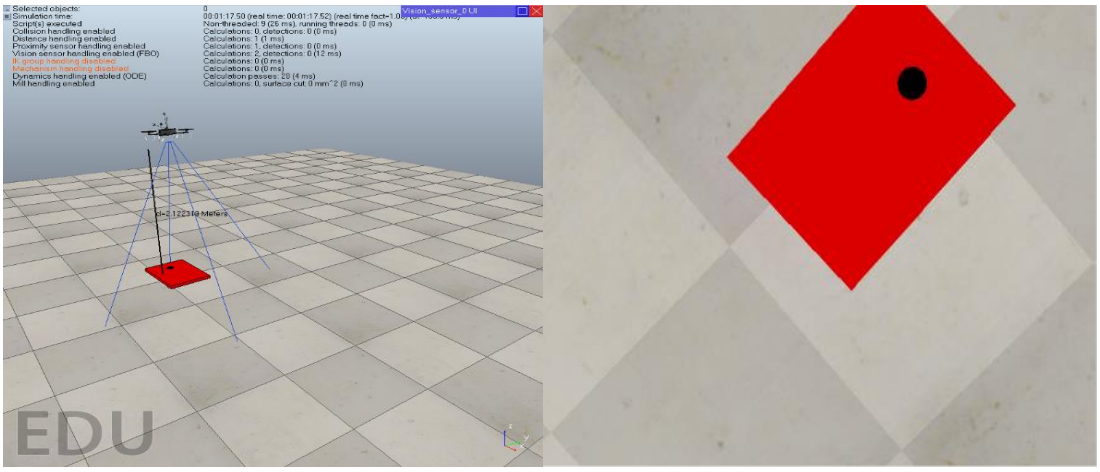


Figura 4.8. Momento 5 simulación.

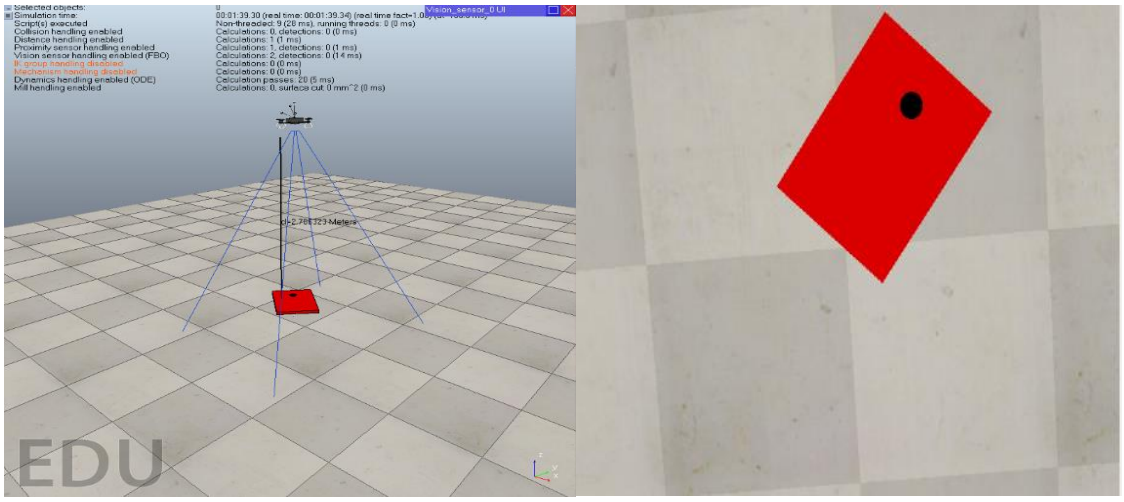


Figura 4.9. Momento 6 simulación.

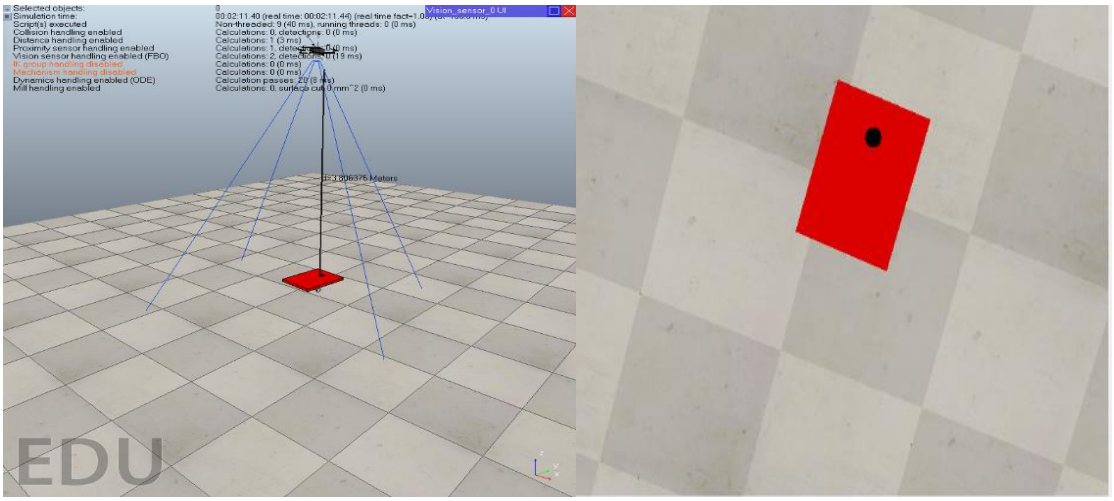


Figura 4.10. Momento 7 simulación.

En las figuras 4.4/5/6 se muestra la simulación en el momento del despegue del dron desde la plataforma. El dron asciende paulatinamente desde la plataforma hasta alcanzar una cierta altura, aunque el dron empieza a avanzar antes de que el dron alcance su altura máxima.

En las sucesivas capturas presentes en las figuras 4.7/8/9/10 el robot ya está en movimiento. Como se ve en las imágenes el dron siempre sigue al robot, corrigiendo su posición y su orientación en función de la posición y orientación de la plataforma; aunque la plataforma no se encuentre siempre centrada y bien orientada en la imagen, el funcionamiento es correcto.

4. CONCLUSIONES Y TRABAJOS FUTUROS.

4.1. CONCLUSIONES.

Después de la realización de este proyecto, se ha llegado a varias conclusiones:

- El uso de V-REP como simulador de robótica es muy intuitivo siempre que las escenas no sean demasiado complejas o se tenga el modelo del robot ya realizado en dicha plataforma.

- ROS tiene un uso bastante extendido en el mundo de la robótica y en este proyecto solamente se han utilizado las cosas más características del entorno, como pueden ser los *topics*.

- La interconexión entre V-REP y ROS no es trivial, de modo que resulta muy difícil poder creo un puente propio entre ambos entornos. Además, la información que es facilitada para realizarlo no está completa o es difusa.

- El uso de los drones es muy extendido, sobre todo en universidades en el ámbito de la investigación, actividades comerciales y uso militar, aunque todas estas actividades están estrechamente relacionadas.

- A la hora de realizar los controladores, resulta complicado ajustar las constantes de los mismos de forma experimental (o de ensayo y error). Una opción más eficiente sería el estudio del sistema, obteniendo la planta del dron y realizando un control más correcto.

- Los tiempos de simulación entre Simulink y V-REP no están sincronizados, de manera que el simulador puede perderse mientras Simulink sigue mandándole datos, y es algo a tener muy en cuenta ya que al realizar cualquier cambio en el

modelo de bloques de Simulink, su tiempo de simulación varía, provocando cambios en el comportamiento del dron en el simulador.

Para una mejor demostración de los resultados, se adjuntará un vídeo aclarativo en el que aparezca tanto la ventana del simulador como Simulink en las tres situaciones anteriormente explicadas, de modo que aparezcan el simulador V-REP y Simulink, observándose tanto los datos de actuación sobre el dron, como su comportamiento.

4.2. TRABAJOS FUTUROS.

Este proyecto puede tener una línea continuista tanto siguiendo con el modelo de simulación y software, como saltando al campo del hardware.

Software.

- Realizar el modelo de aterrizaje completando el modelo actual, de manera que el dron ascienda al principio de la simulación, y descienda posándose sobre la plataforma cuando haya recogido todos los datos necesarios, o una vez que no se desee seguir volando.

- Añadir el dron con su controlador al proyecto de conducción autónoma de vehículos, de modo que pueda recoger los datos necesarios por el vehículo para mandárselos y completar su base de datos.

Hardware.

- Realizar un primer paso con un dron real, de modo que reconozca una plataforma del mismo modo que se ha realizado en este proyecto.

- Una vez realizado lo anterior, podría sustituirse la plataforma por un vehículo, de modo que se cierre totalmente el proyecto. Haciendo que un dron siguiera de forma autónoma a un vehículo que a su vez también es autónomo.

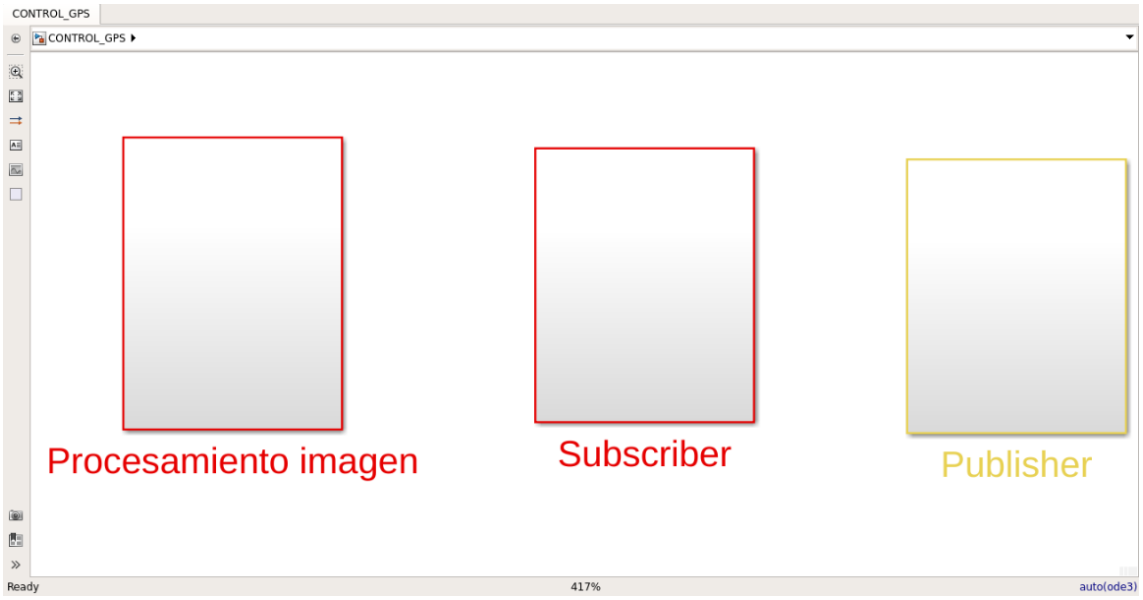
PLANOS Y DIAGRAMAS

PLANOS Y DIAGRAMAS

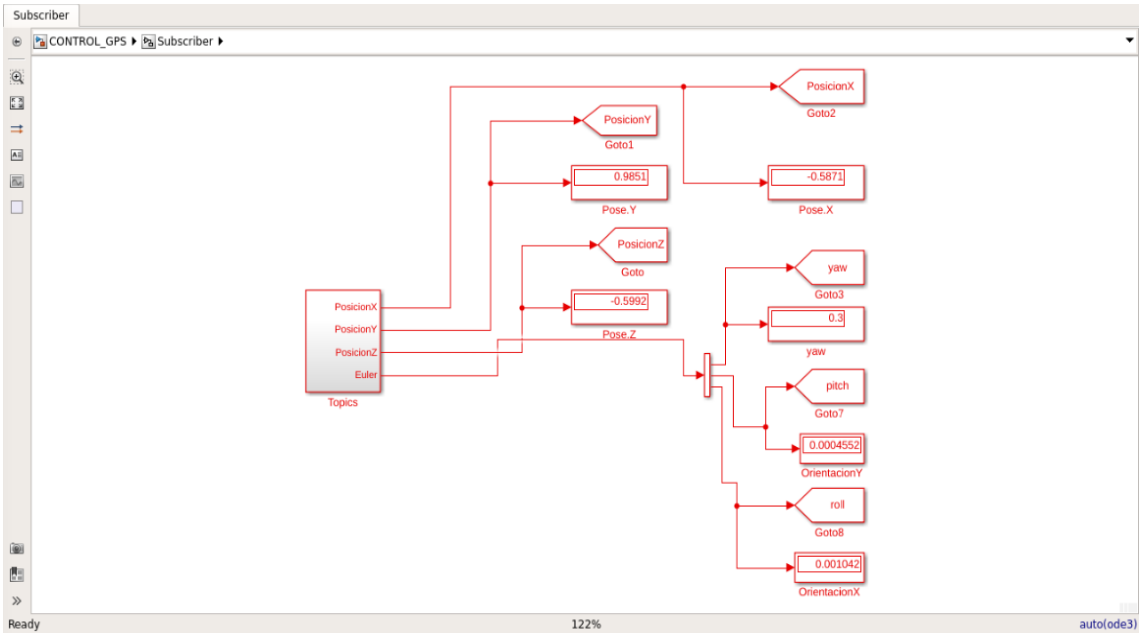
En este apartado se va a exponer de forma ordenada, de más externo a más interno, los modelos de bloques de Simulink que se han creado para realizar el proyecto.

1. Control absoluto de posición.

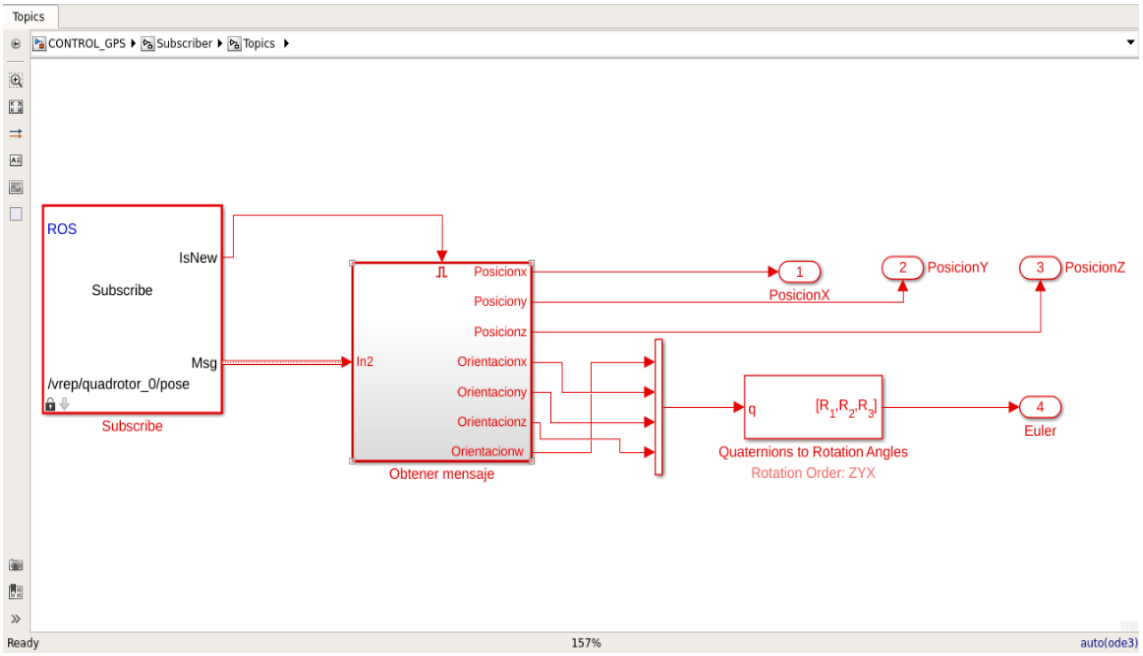
La vista general del modelo *Control_GPS.slx* contiene tres bloques formados a partir de su funcionamiento, mostrándose a continuación:



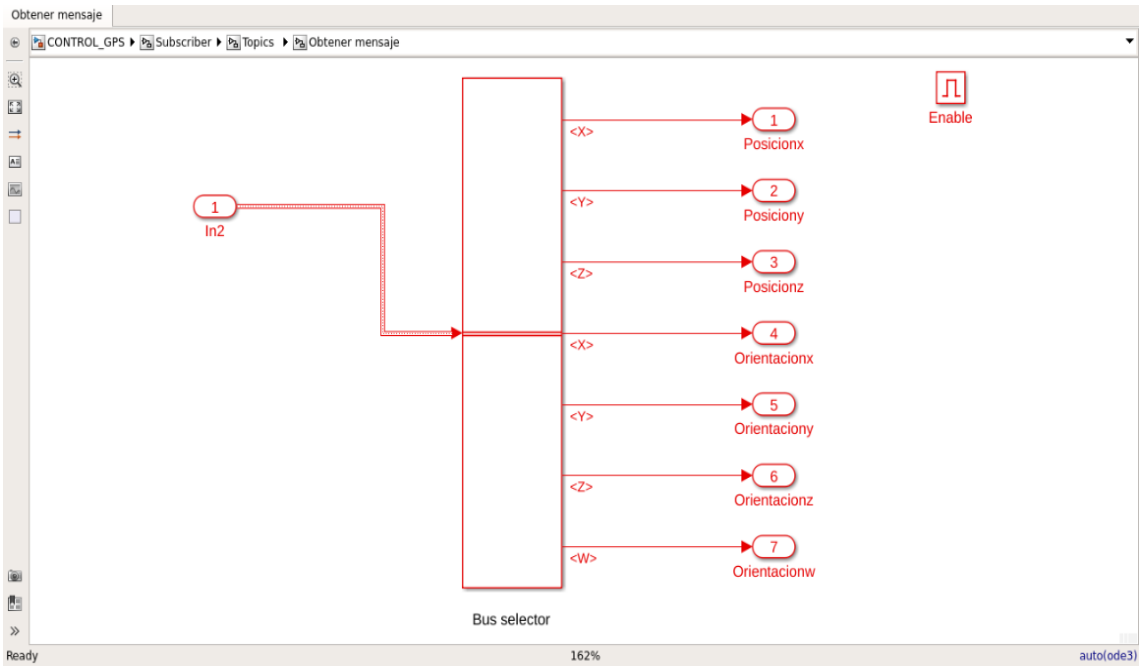
El subsistema **Subscriber** a su vez contiene:



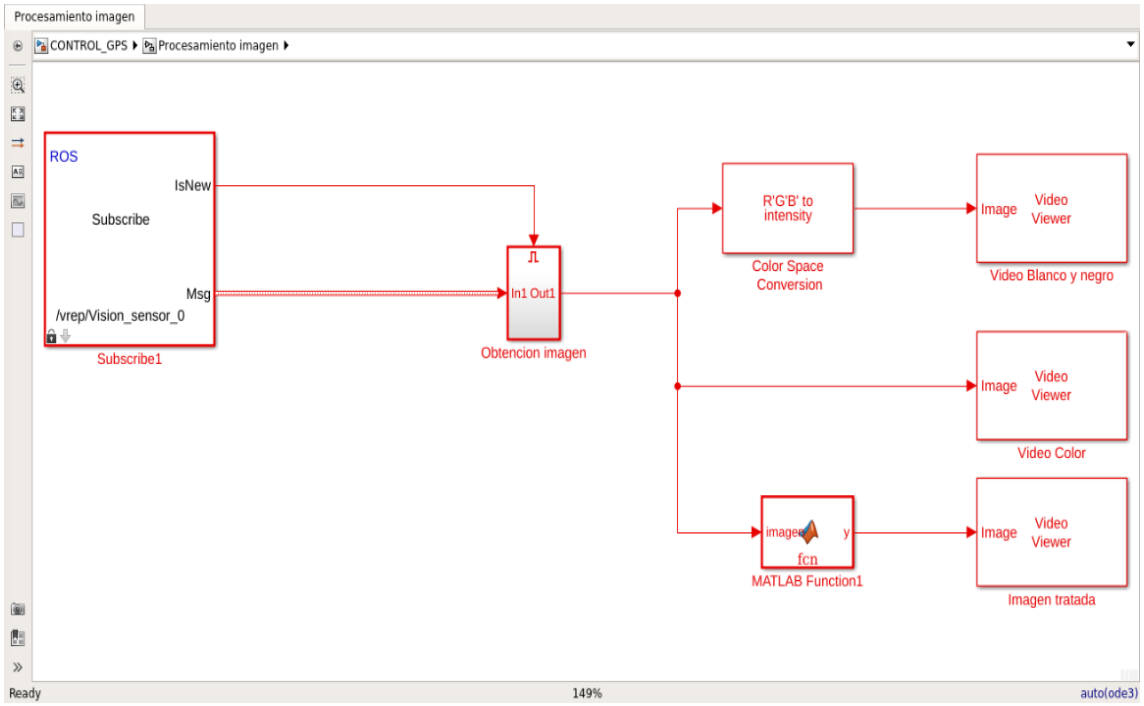
El subsistema **Topics** contiene los siguientes bloques:



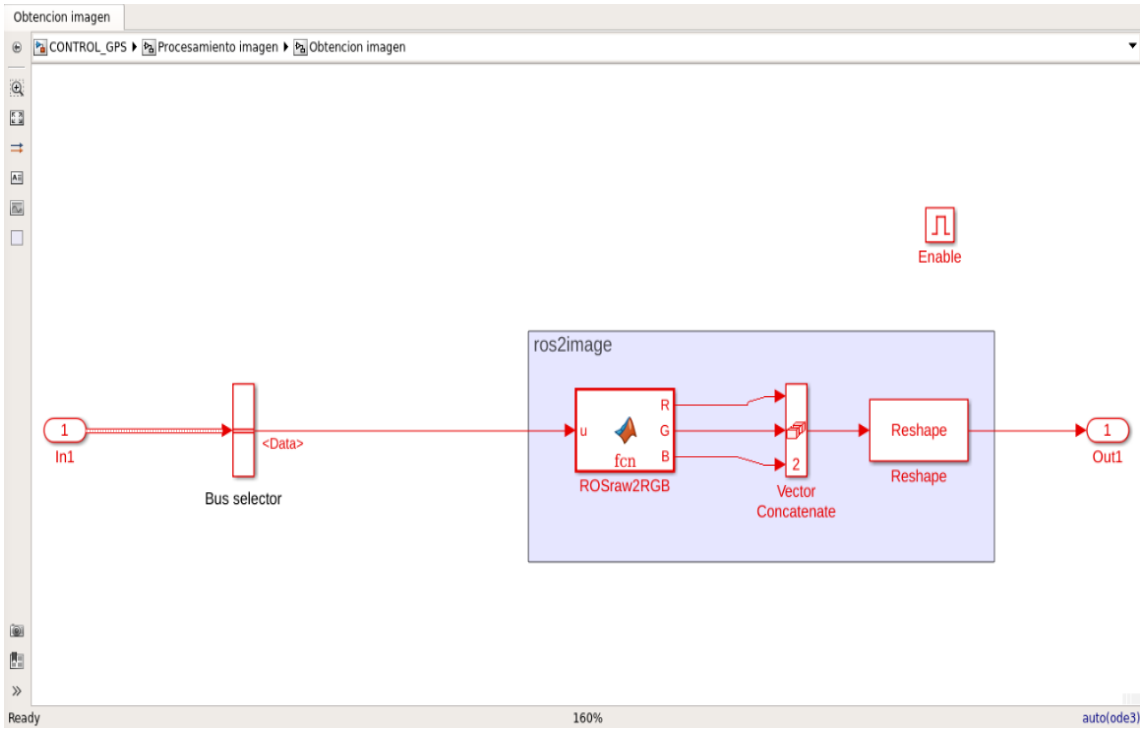
El subsistema **Obtener mensaje** a su vez contiene:



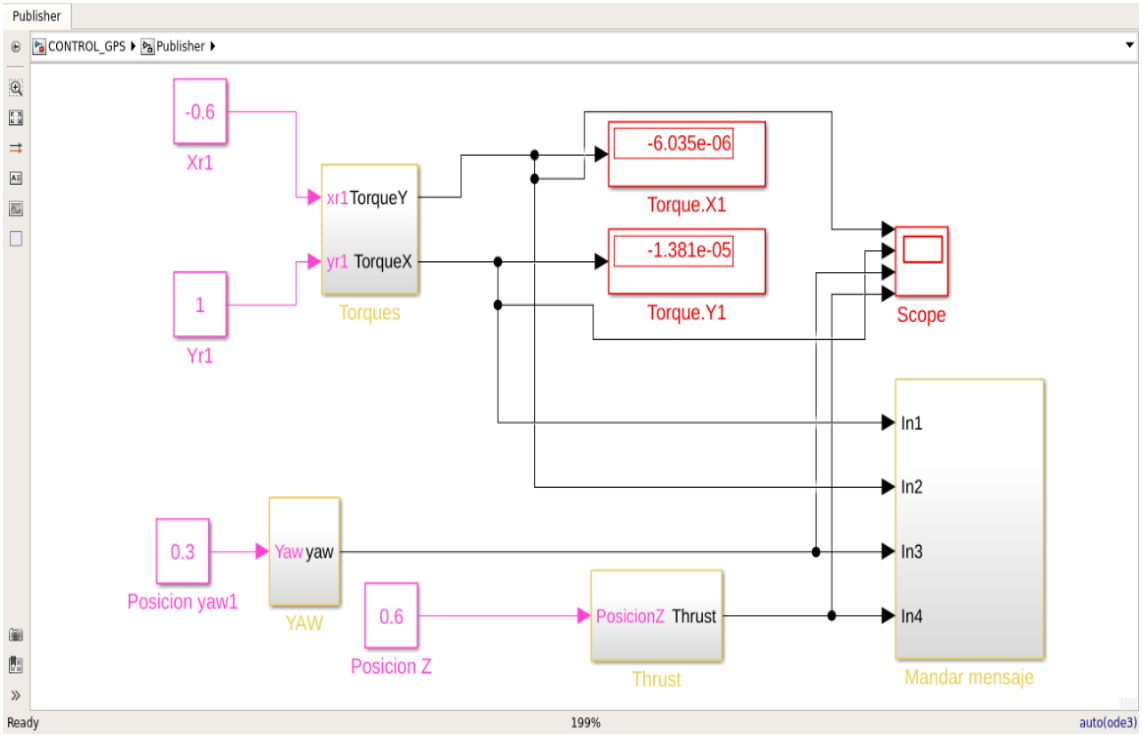
El subsistema **Procesamiento imagen** contiene estos bloques de Simulink:



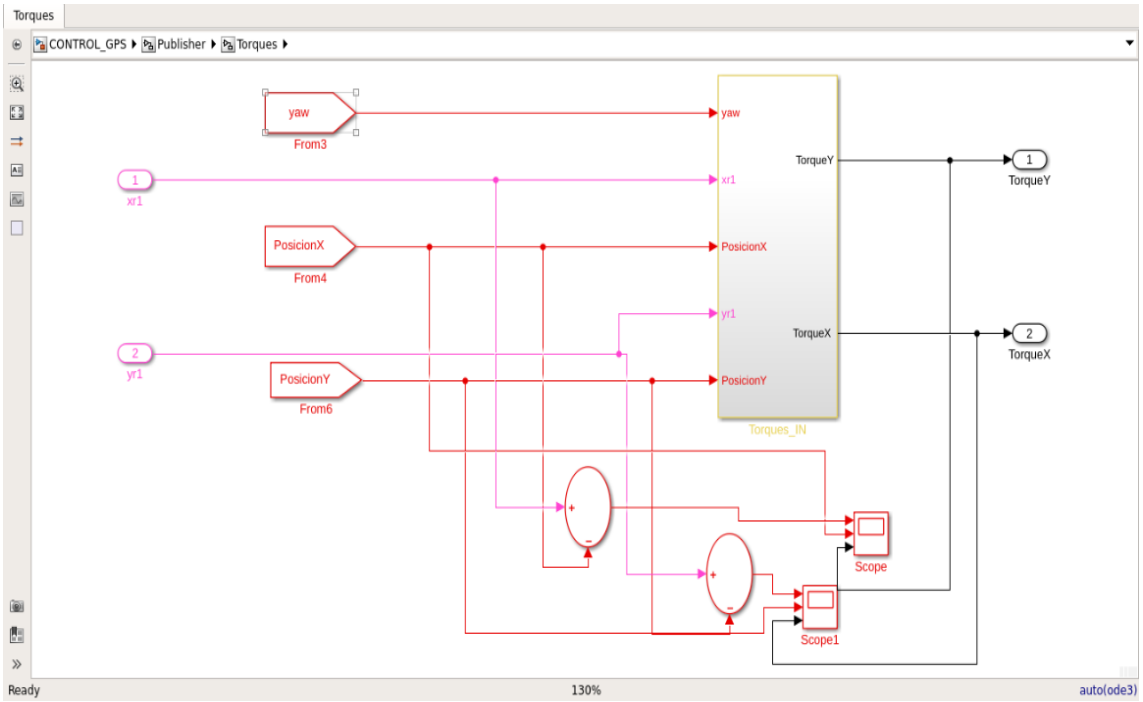
Del mismo modo, el subsistema **Obtención imagen** está formado por:



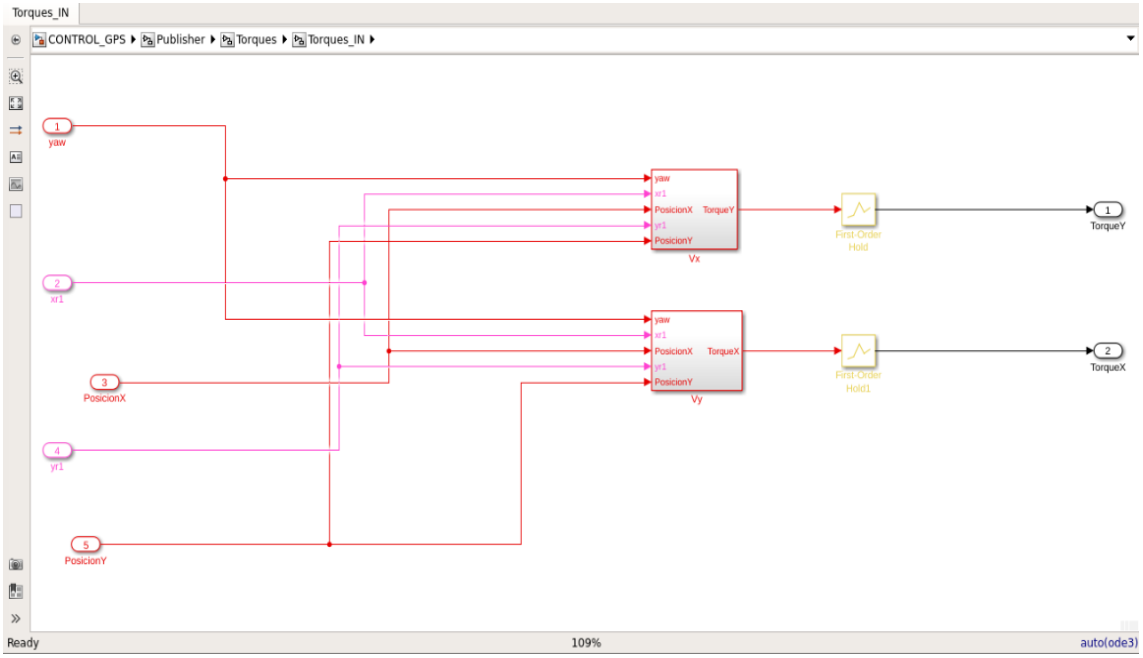
El último bloque del esquema general (**Publisher**), está formado por el siguiente diagrama:



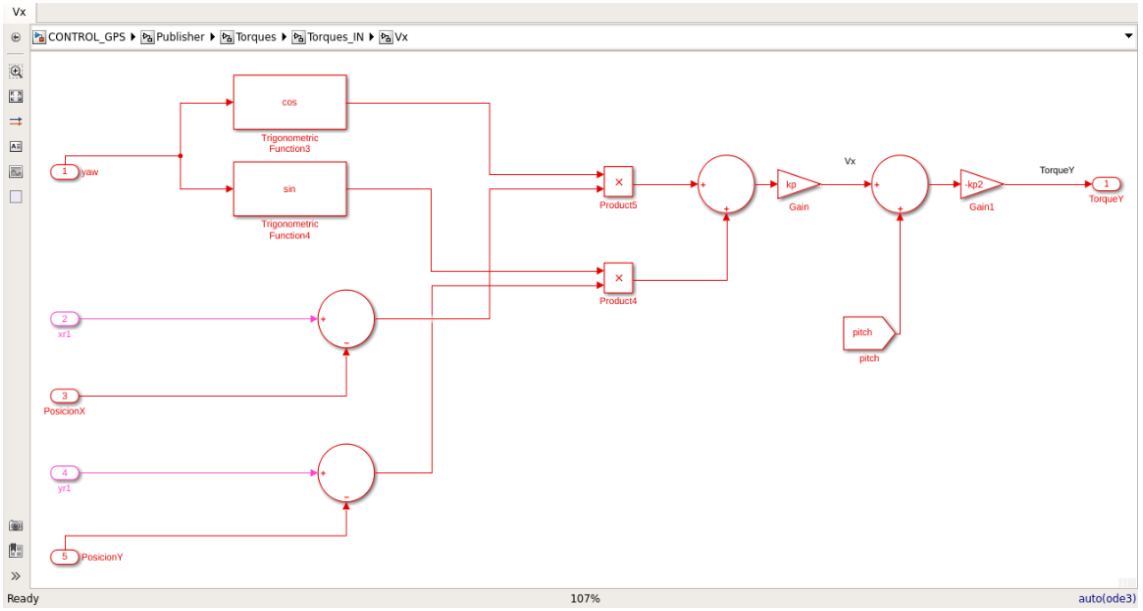
Uno de los subsistemas es el llamado **Torques**, el cual contiene los siguientes bloques:

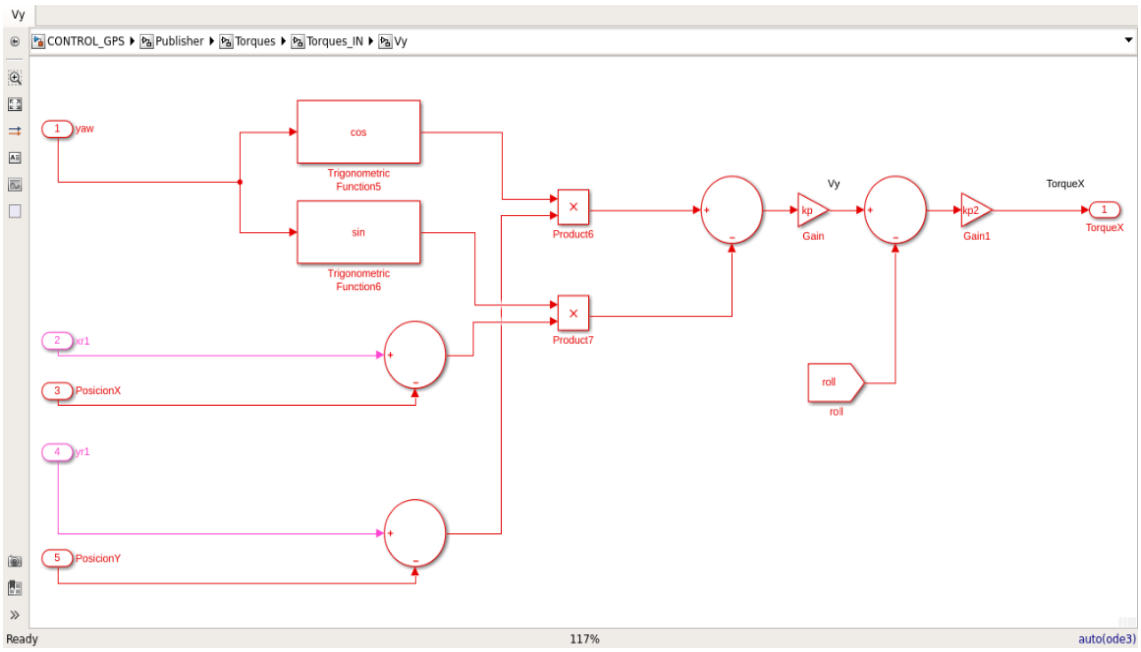


A su vez, el subsistema **Torques**, contiene otro llamado **Torques_IN**, que está formado por:

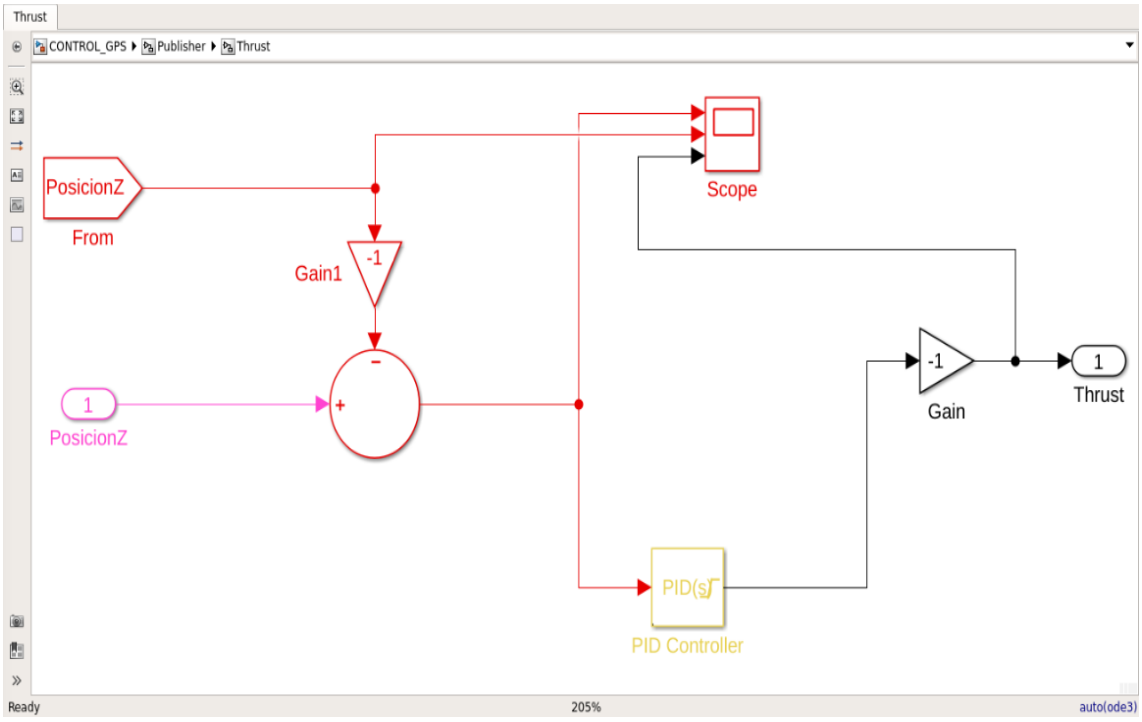


A su vez el subsistema **Torques_IN** contiene otros dos subsistemas llamados **Vx** y **Vy**, mostrados a continuación:

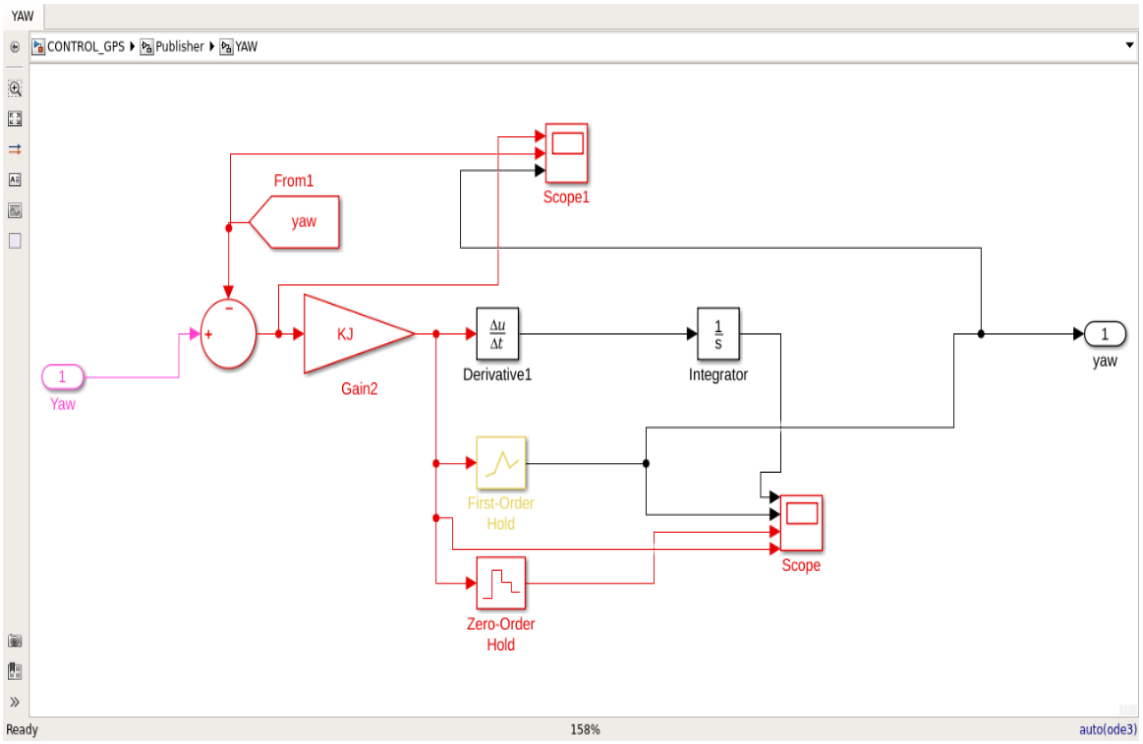




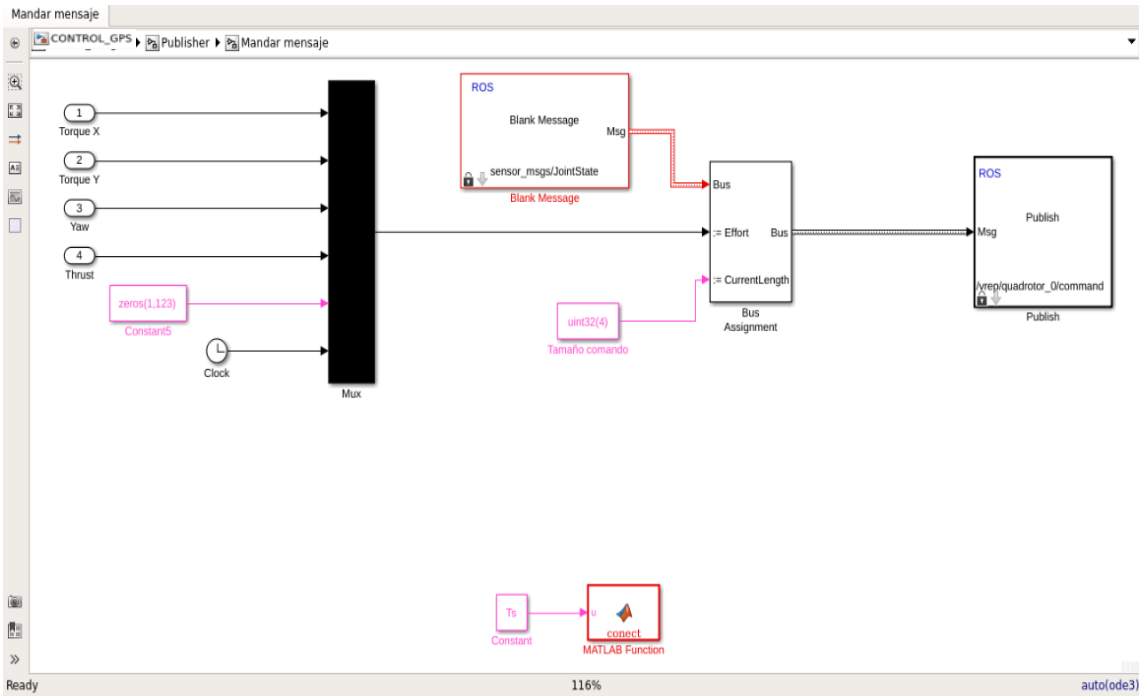
Otro de los subsistemas que completan el bloque **Publisher** es el **Thrust**:



Por último, se completa el bloque **Publisher** con el subsistema **YAW**:

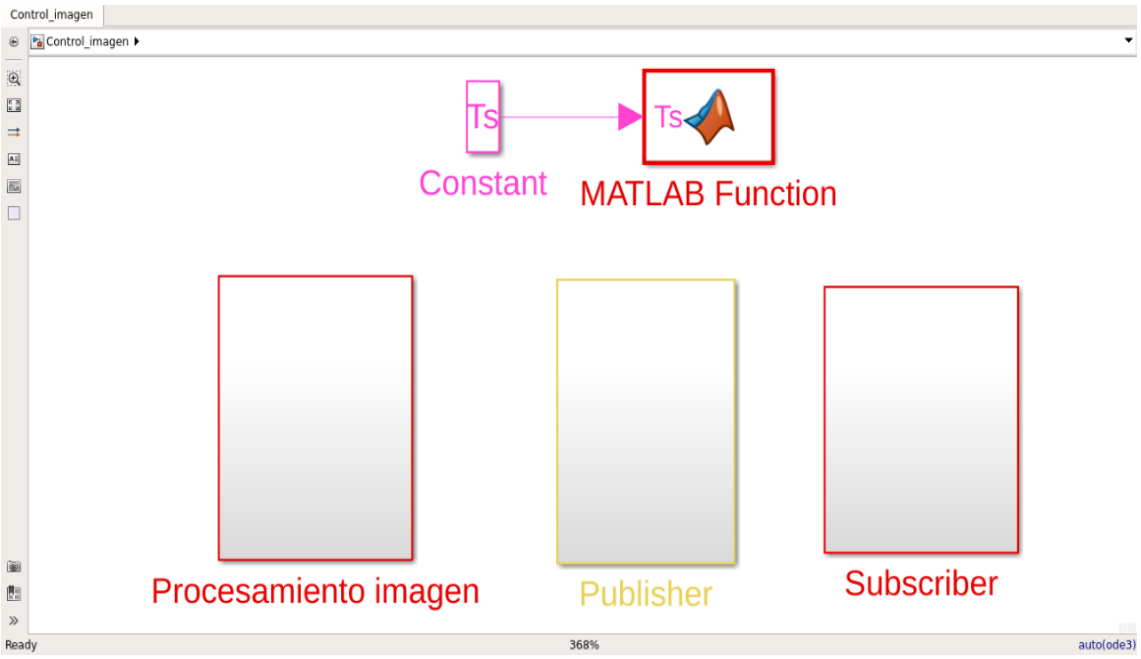


El último bloque desglosable que forma el subsistema **Publisher** es el bloque llamado **Mandar mensaje**:

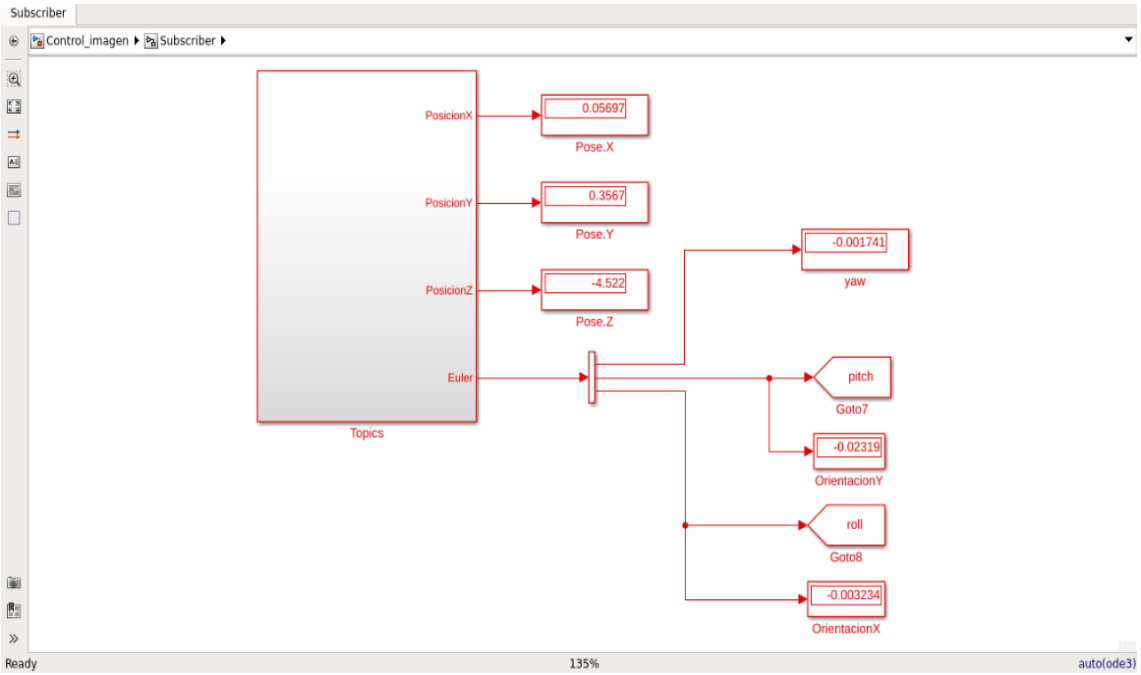


2. Control para el seguimiento visual de un objeto.

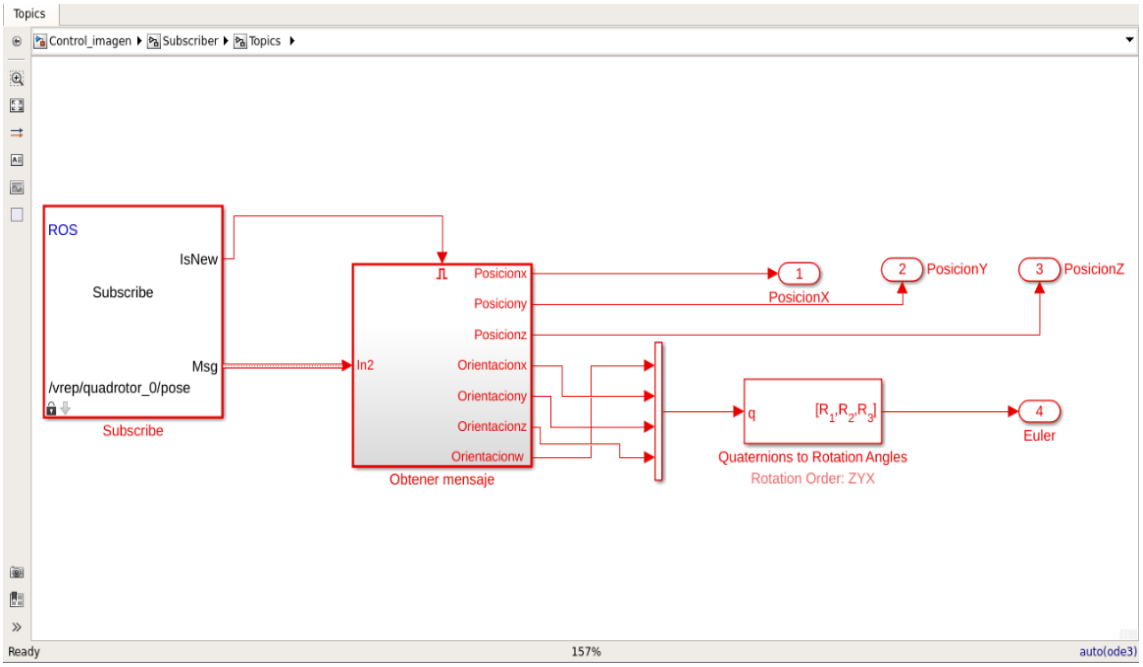
La vista general del modelo *Control_imagen.slx* de Simulink está formada por los tres bloques siguientes, los cuales se desglosarán a continuación:



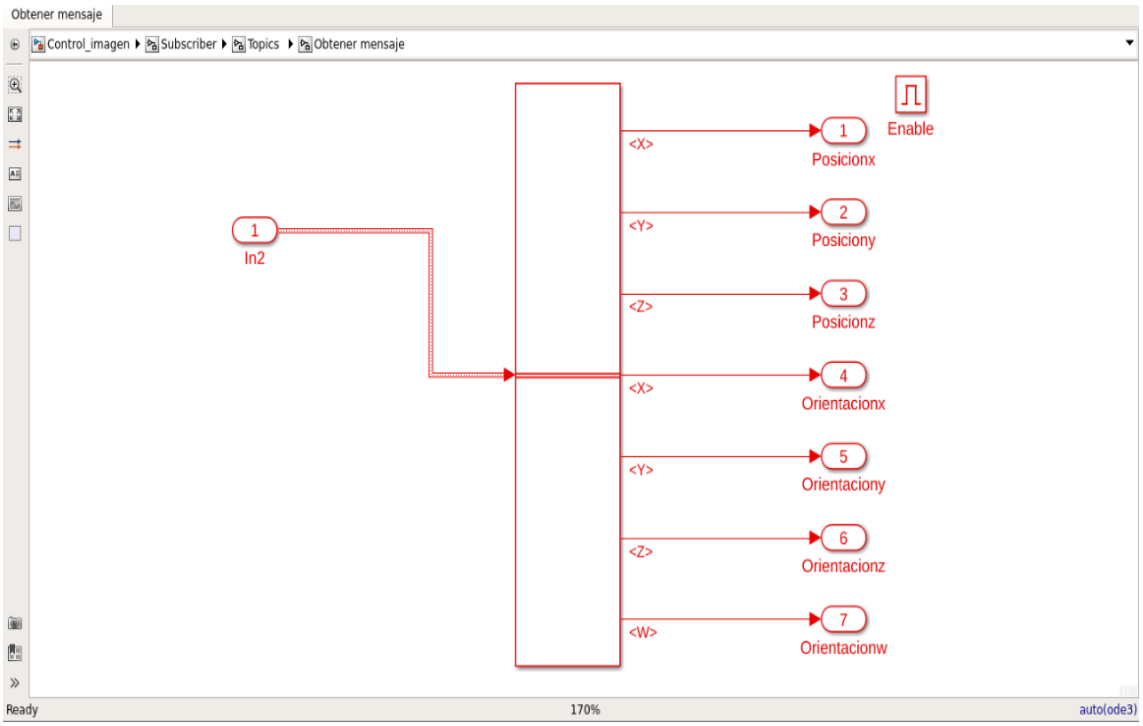
El bloque **Subscriber** contiene los siguientes bloques, el cual a su vez tiene otro subsistema:



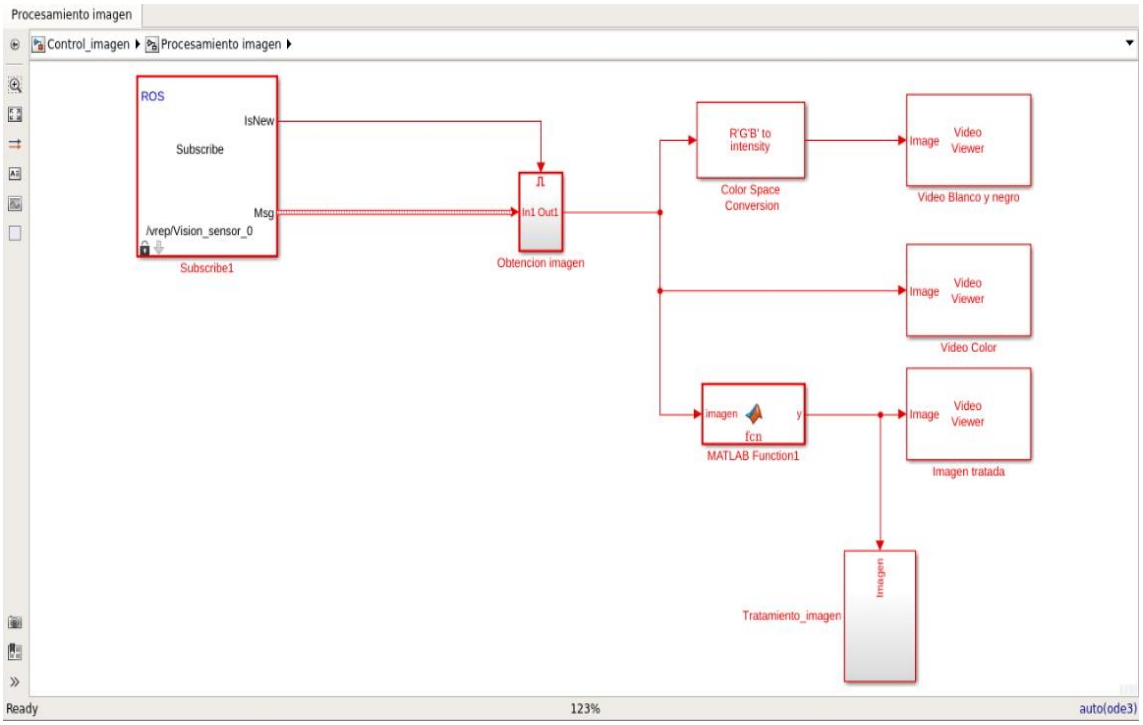
El subsistema **Topics** se muestra a continuación:



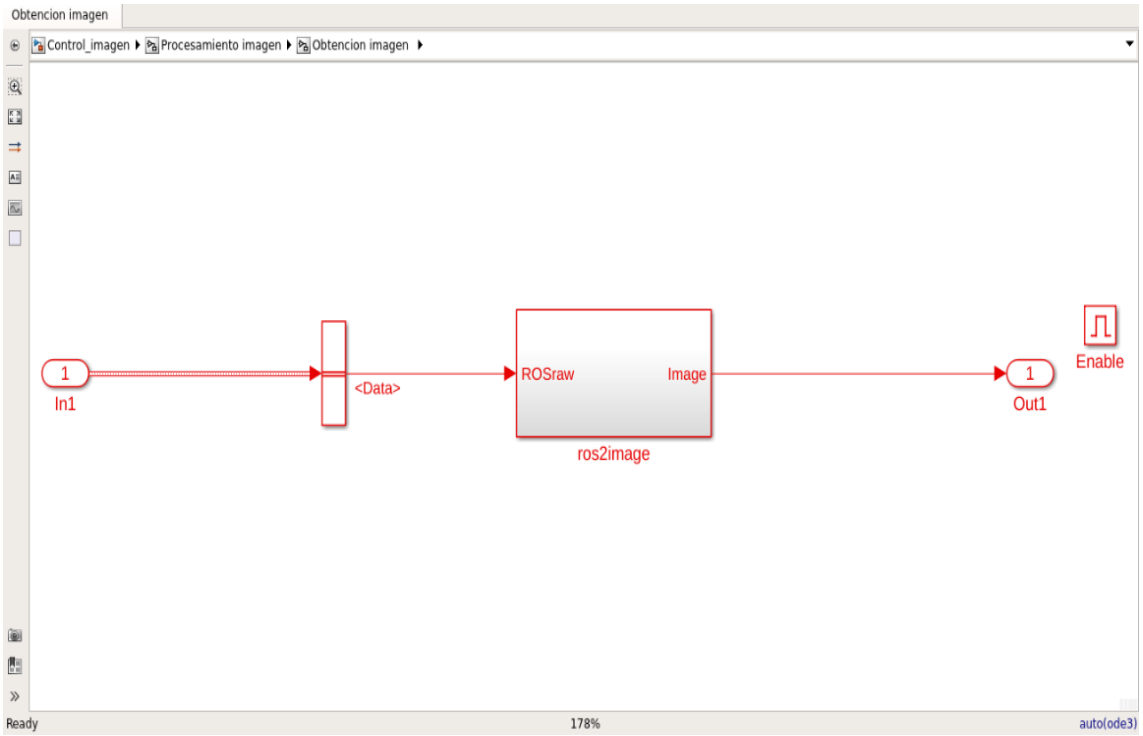
El bloque **Obtener mensaje** contenido en el subsistema **Topics** está formado de la siguiente manera:



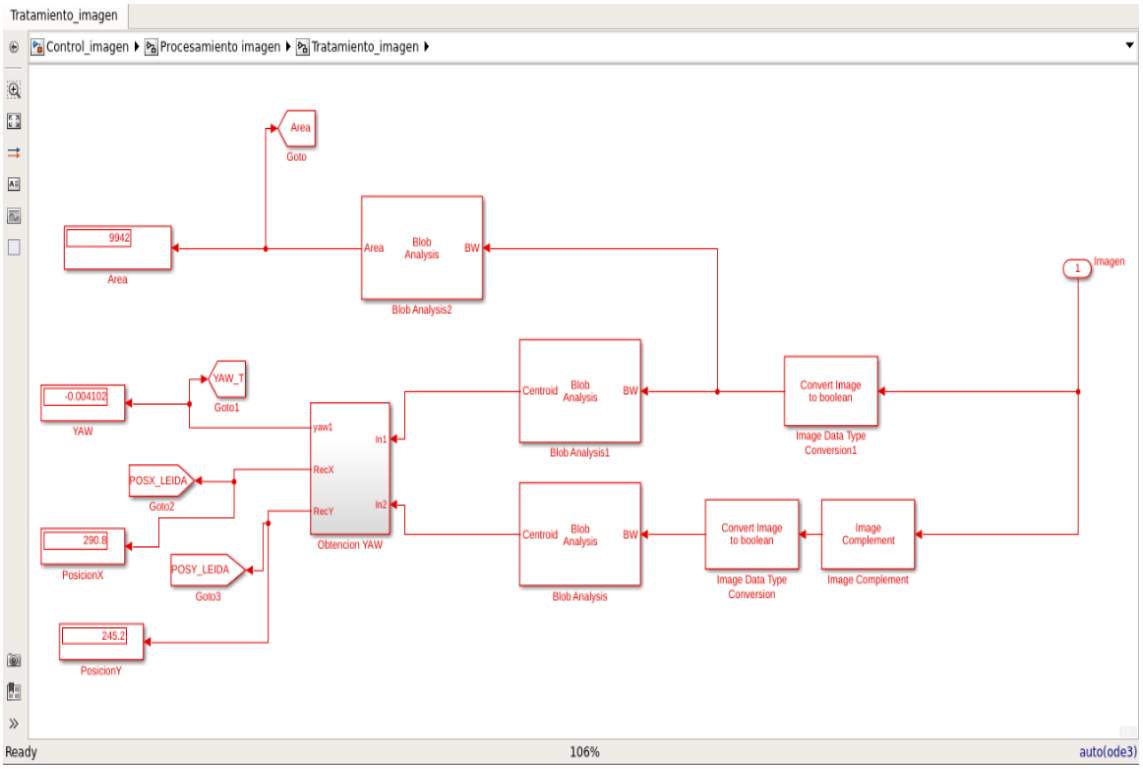
El bloque general **Procesamiento imagen** se forma con los siguientes bloques:



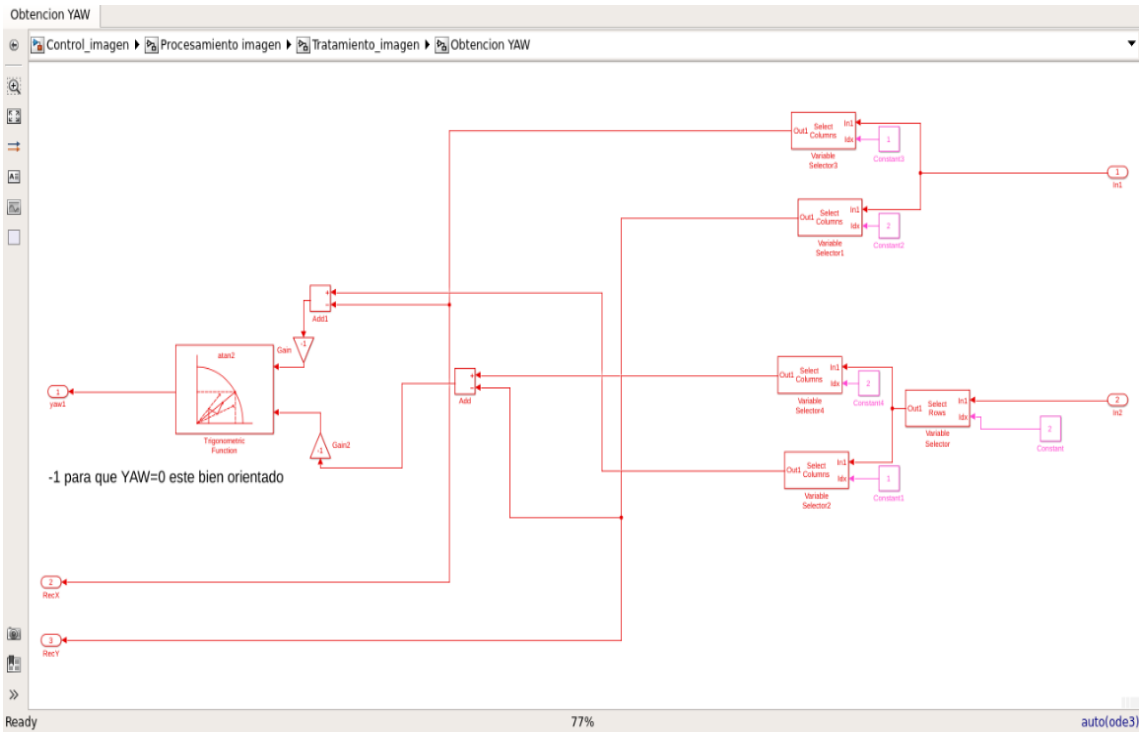
El bloque **Obtención imagen** contenido en el bloque anterior tiene la siguiente forma:



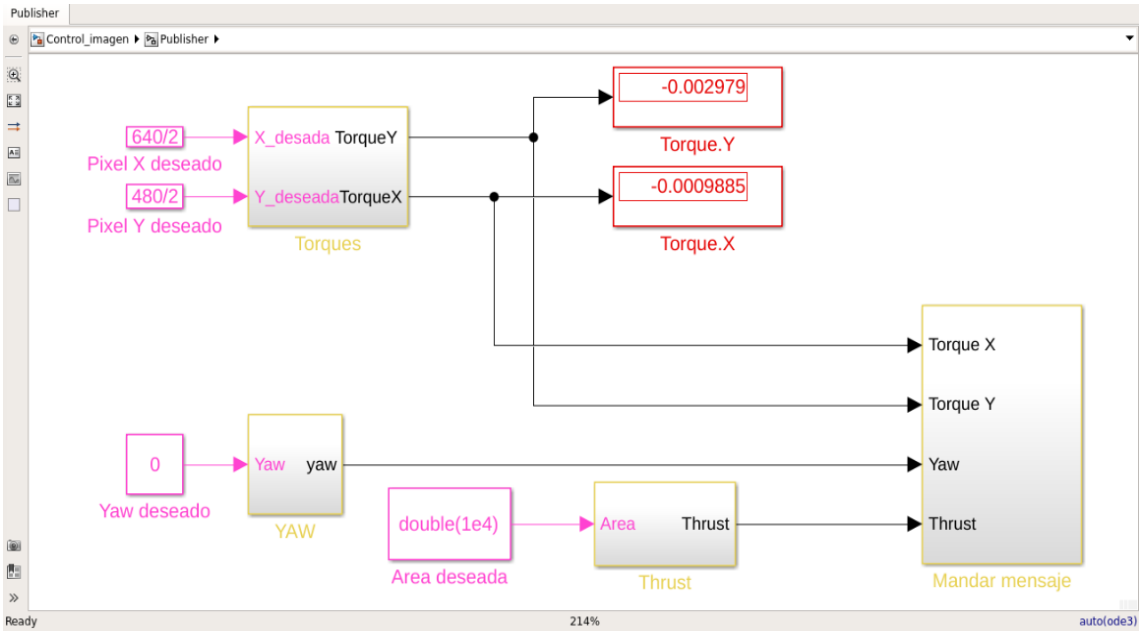
Otro de los bloques que forman el subsistema **Procesamiento imagen** es el llamado **Tratamiento_imagen**, el cual está representado por el siguiente diagrama de bloques:



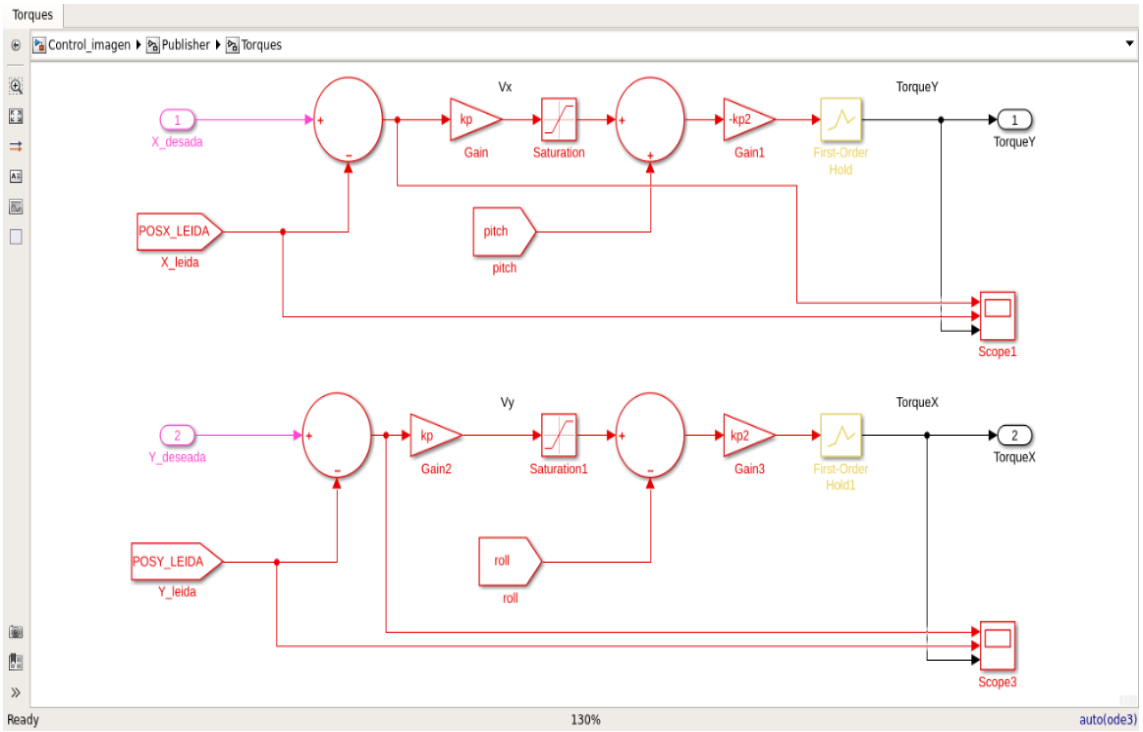
El diagrama del bloque **Obtención YAW** contenido en el subsistema anterior es de la siguiente forma:



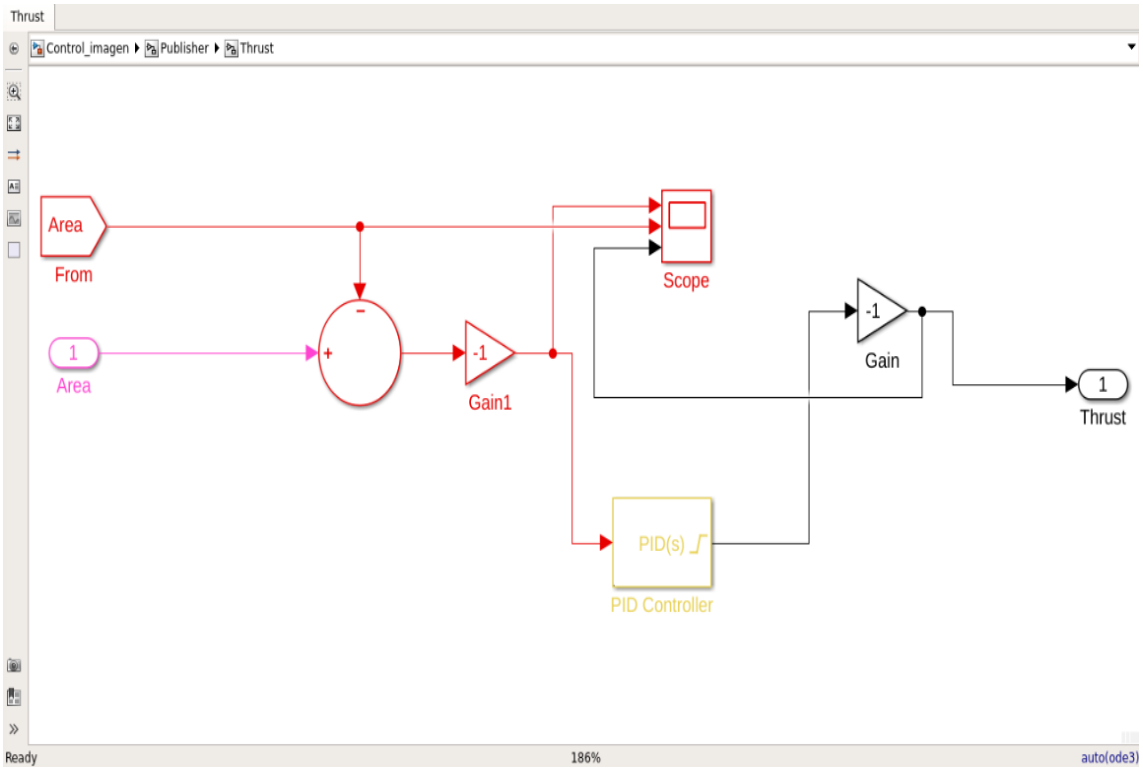
El siguiente modelo que se muestra es el del subsistema general **Publisher**, el cual contiene bastantes bloques:



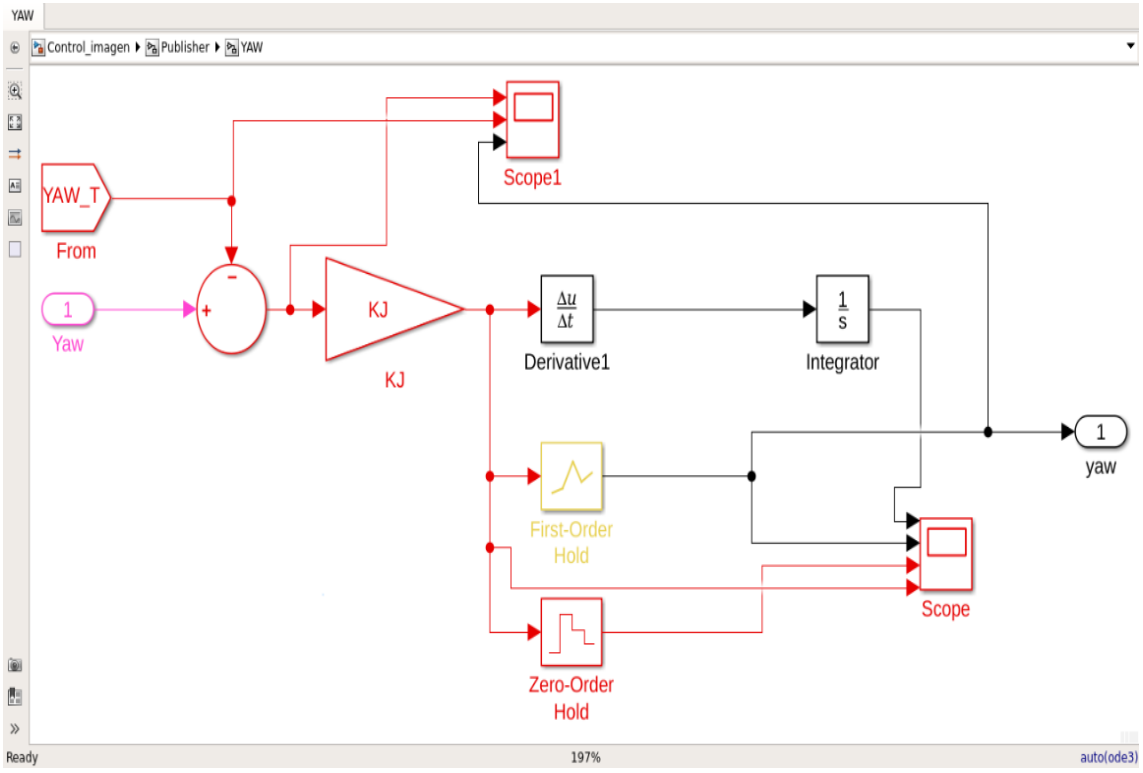
Uno de esos bloques se llama **Torques**:



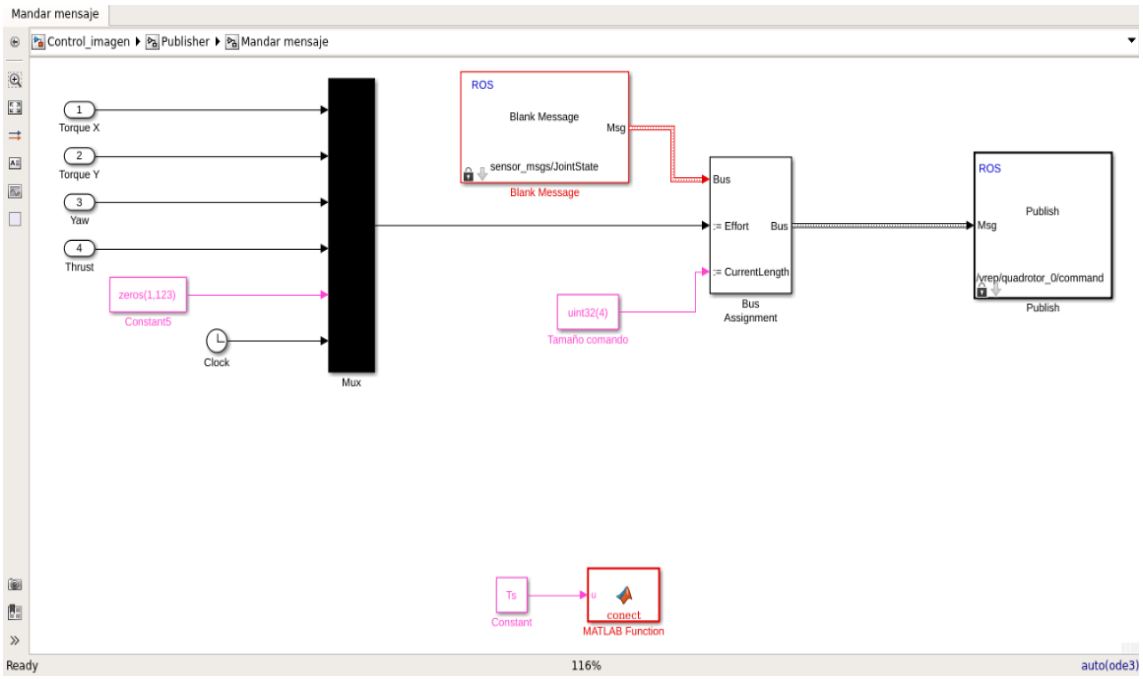
También contiene un bloque llamado **Thrust**:



El bloque **YAW** se muestra a continuación:



Por último, bloque **Mandar mensaje** contenido en el subsistema **Publisher**:



PLIEGO DE CONDICIONES

PLIEGO DE CONDICIONES

Requisitos de Hardware.

El hardware utilizado para este proyecto ha sido el siguiente:

- Ordenador compatible con PC. Memoria RAM mínima de 4GB y almacenamiento mínimo de 300GB.

Requisitos de Software.

El software utilizado para este proyecto ha sido:

- Sistema operativo GNU/Linux con distribución Ubuntu 14.05 LTS. Se debe utilizar esta distribución en concreto obligatoriamente. En él se han realizado todas las aplicaciones.

- Sistema operativo Windows 10. Ha sido utilizado para la confección del proyecto.

- Entorno de desarrollo robótico ROS (distribución INDIGO).

- Software matemático MATLAN/SIMULINK.

- Simulador robótico V-REP (Versión Educativa).

PRESUPUESTO

PRESUPUESTO

En esta sección se muestra el presupuesto de ejecución del proyecto y forma de las siguientes partes:

- Coste de equipos y software.
- Coste de personal.
- Costes de materiales e impresión.

Presupuesto de ejecución material.

En estas partidas se incluyen el coste por uso de equipos y software, coste de personal y coste de materiales.

- Ordenador Asus Intel Core i7-4510U, up to 3.1Gz, 8GB de memoria RAM y 1TB de memoria ROM.
- Sistema operativo Windows 10.
- Softwares de libre distribución (tanto el sistema operativo GNU/LINUX como el entorno de desarrollo ROS y el simulador V-REP).
- Software matemático Matlab/Simulink.

Equipos	Precio unitario (€)	Amortización	Uso	Total (€)
Ordenador ASUS	750	4 años	5 meses	78,125
SO: Windows 10	51	Para siempre	5 meses	51
Software: Matlab	150	4 años	5 meses	15,625
TOTAL				144,75

Tabla 1.1. Presupuesto de costes y equipos y software.

En este punto se detalla el sueldo base por hora trabajada de un ingeniero trabajando en jornada parcial de 15 horas a la semana durante los 5 meses que se han necesitado para la realización de este proyecto. Se estima que el mes tiene una media de 22 jornadas laborables.

Concepto	Meses	Sueldo al mes (€)	Total (€)
Ingeniero	5	310	1550

Tabla 1.2. Coste de personal.

A continuación, se muestra el total de los costes de ejecución.

Concepto	Precio unitario (€)	Total (€)
Coste de equipos y software	144,75	144,75
Coste de personal	1550	1550
TOTAL		1874,75

Tabla 1.3. Presupuesto de ejecución de material.

Presupuesto total.

Costes de ejecución	IVA	Total con IVA
1874,75	21%	2268,4475

Tabla 1.3. Costes totales con IVA.

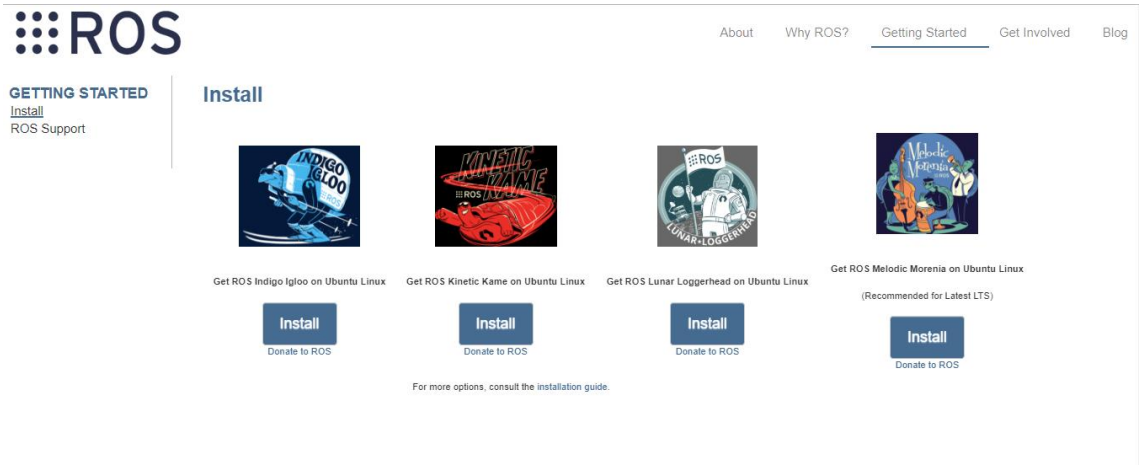
MANUAL

MANUAL

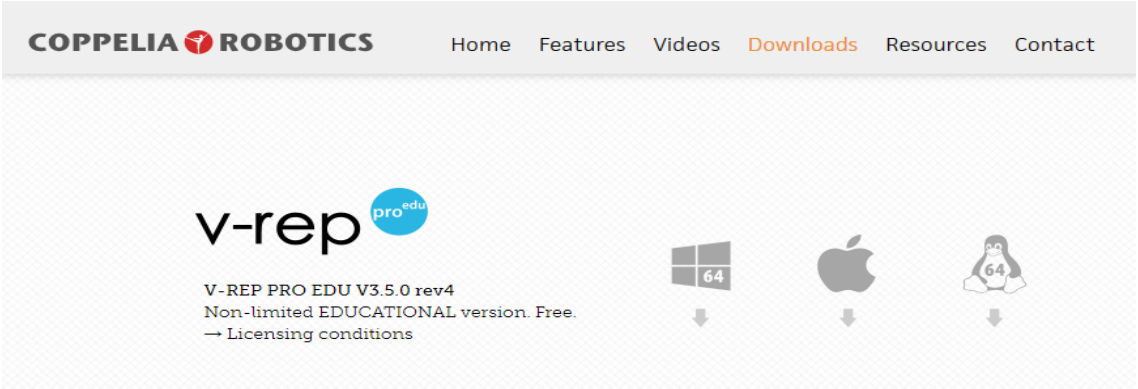
En este capítulo se explica cómo se debe proceder para poder ejecutar las aplicaciones creadas en este proyecto.

INSTALACIÓN.

Lo primero que se debe realizar es la instalación del entorno de desarrollo ROS, el cual se encuentra disponible en la página web http://www.ros.org/. En esta página se encuentra información de cómo funciona el entorno y cómo se debe proceder para su instalación en el sistema Linux. La instalación, en concreto se encuentra en el link *http://wiki.ros.org/ROS/Installation.*

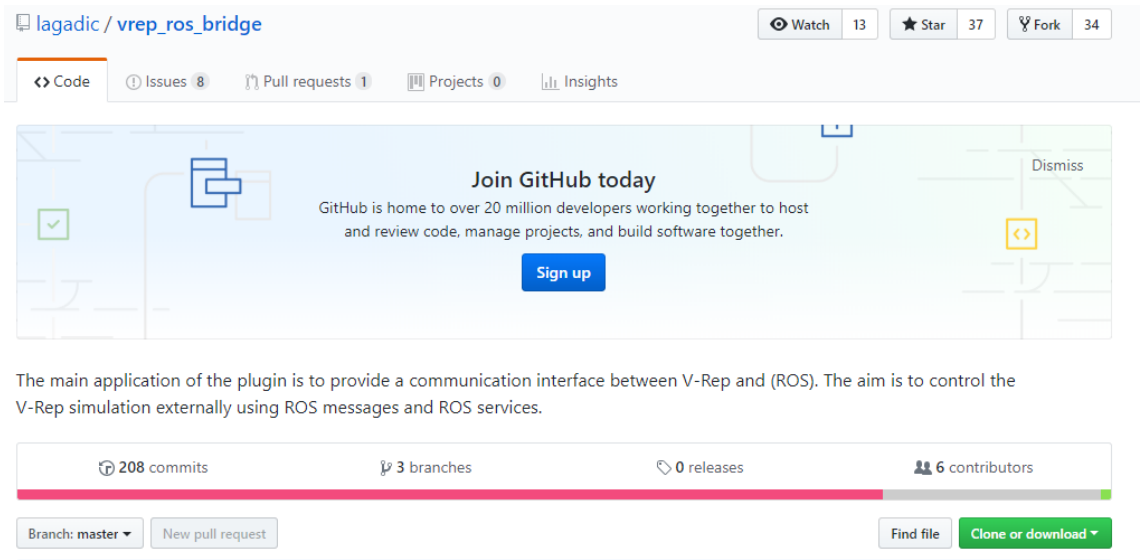


Una vez instalado ROS, lo siguiente a instalar es el simulador V-REP, siendo la versión para instalar X.
En la página web *http://www.coppeliarobotics.com/* se encuentra toda la información suminstrada por el desarrollador. La versión a instalar se encuentra en el link *http://www.coppeliarobotics.com/downloads.html*.
Para poder manejar bien este simulador, se recomienda visitar la página *http://www.coppeliarobotics.com/helpFiles/*.



El último software que se debe instalar es MATLAB/SIMULINK, para el cual es necesaria una licencia. En este caso, se ha utilizado una licencia proporcionada por la Universidad de Alcalá. La descarga del software se encuentra en el link *https://es.mathworks.com/* así como toda la información necesaria para su ejecución. La versión que se ha utilizado es Matlab Rb2016b, aunque puede utilizarse cualquier versión más reciente.

Lo siguiente que hay que realizar es instalar el paquete de ROS *vrep_ros_bridge* disponible en el link *http://wiki.ros.org/vrep_ros_bridge* y en el repositorio de Lagadic *https://github.com/lagadic/vrep_ros_bridge* en el cual se encuentra un guion a seguir para que el funcionamiento del paquete se produzca de forma correcta. Este es un proceso crítico, ya que, si no se realiza de la forma adecuada, la escena del dron creada en V-REP no se laza correctamente.



LANZAMIENTO APLICACIONES

Para el lanzamiento de las aplicaciones primero se debe tener claro qué escena corresponde a cada modelo de simulink, esto se explica a continuación:

- Control vía posición: V-REP→ GPS_CONTROL_scene || Simulink→ CONTROL_GPS
- Control vía imagen: V-REP→ QD_platform || Simulink→ Control_imagen
- Control vía imagen robot móvil: V-REP→ QD_RBM_plat || Simulink→ Control_XYZW_imagen_robot

Una vez se tiene esto claro, se deben abrir tres terminales diferentes en el sistema operativo Linux y seguir los siguientes pasos:

1. En un primer terminal, lanzar un roscore.

```
oscarcanre10@oscarcanre10-X555LD:~$ roscore
... logging to /home/oscarcanre10/.ros/log/c1efaff4-706d-11e8-ab68-14dda91145b0/roslaunch-oscarcanre10-X555LD-2511.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://oscarcanre10-X555LD:41944/
ros_comm version 1.11.21


SUMMARY
========

PARAMETERS
 * /rosdistro: indigo
 * /rosversion: 1.11.21

NODES

auto-starting new master
process[master]: started with pid [2523]
ROS_MASTER_URI=http://oscarcanre10-X555LD:11311/

setting /run_id to c1efaff4-706d-11e8-ab68-14dda91145b0
process[rosout-1]: started with pid [2536]
started core service [/rosout]
```

2. En un segundo terminal, una vez que el roscore sea lanzado, se debe abrir la carpeta en la que se ha instalado V-REP y lanzar el simulador.

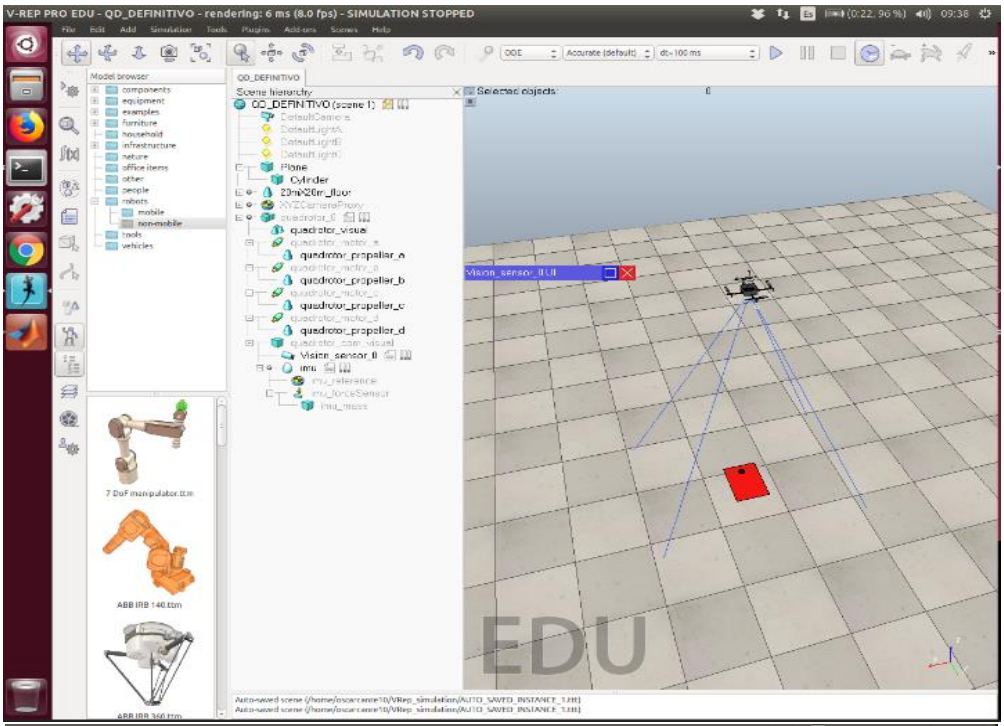
```
oscarcanre10@oscarcanre10-X555LD:~$ cd VRep_simulation/
oscarcanre10@oscarcanre10-X555LD:~/VRep_simulation$ ./vrep.sh
Using the default Lua library.
Loaded the video compression library.
Add-on script 'vrepAddOnScript-addOnScriptDemo.lua' was loaded.
QSplitter::setCollapsible: Index 0 out of range
Simulator launched.
Plugin 'MeshCalc': loading...
Plugin 'MeshCalc': load succeeded.
Plugin 'BubbleRob': loading...
Plugin 'BubbleRob': load succeeded.
```

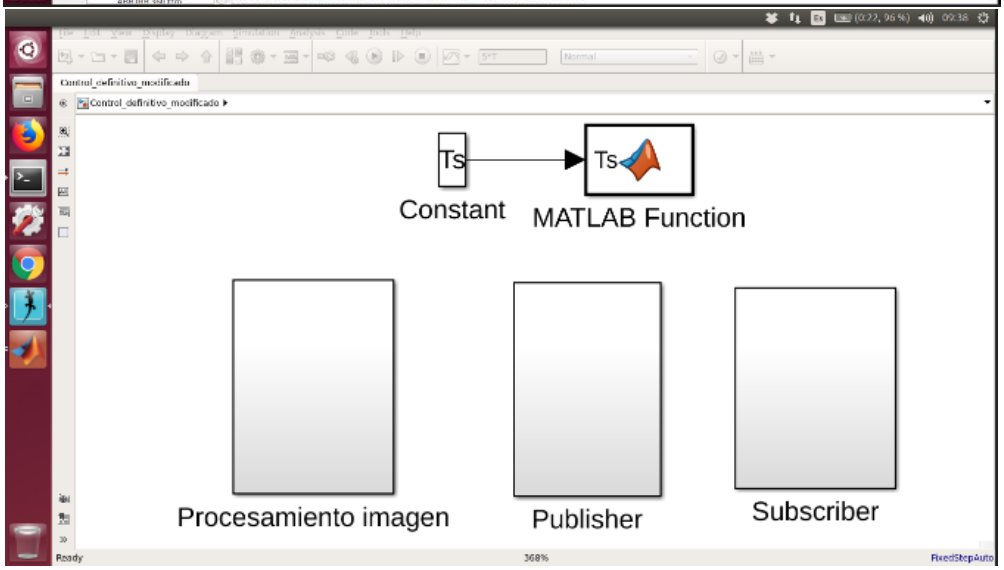
3. En el último terminal, se abre el Matlab.

```
oscarcanre10@oscarcanre10-X555LD:~$ ./matlab
```

4. En V-REP, se selecciona la escena que se desea abrir y en Matlab su modelo correspondiente.

5. Una vez que todo lo anterior se haya realizado, se debe lanzar primero la simulación en V-REP, y seguidamente se debe lanzar el modelo de Simulink.





BIBLIOGRAFÍA

BIBLIOGRAFÍA

[1] Dr. Hoam Chung. "Coordination and Control of Unmanned Aerial and Ground Vehicle (UAV/UGV)".

[2] Ethan Hoerr, Dakota Mahan and Alex Vallejo. "Multiple UAV Coordination". 2014.

[3] Markus Quaritsch, Saeed Yahyanejad, Bernhard Rinner, Hermann Hellwagner and Christian Bettstetter. "Communication and Coordination for Drone Networks". 2016.

[4] Óscar Díaz Cantos. "Drones y su aplicación en materia de seguridad y salud en el trabajo". 2015.

[5] Ricardo SPica, Giovanni Claudio, Fabien Spindler and Paolo Robuffo Giordano. "Interfacing Matlab/Simulink with V-REP for an Easy Development of Sensor-Based Control Algorithms for Robotic Platforms". 2014.

[6] Iván F. Mondragón, Pascual Campoy, Miguel A. OLivares-Mendez and Carol Martínez. "3D Object Following Based on Visual Information for unmanned Aerial Vehicles". 2011.

[7] Información extraída de "Phantom1 Especificaciones". (https://www.dji.com/es/phantom/info#specs).

[8] Información extraída de "Parrot AR.DRONE 2.0 Elite Edition". (https://www.parrot.com/es/drones/parrot-ardrone-20-elite-edition#parrot-ardrone-20-elite-edition-details).

[9] Información extraída de "AscTec Hummingbird". (http://www.asctec.de/en/uav-uas-drones-rpas-roav/asctec-hummingbird/#pane-0-1).

[10] Información extraída de "WikiRos" (http://wiki.ros.org/).

[11] Información extraída de "vrep_ros_bridge" (http://wiki.ros.org/vrep_ros_bridge) y de "V-Rep ROS Bridge" (https://github.com/lagadic/vrep_ros_bridge).

[12] José Luis Sánchez de la Rosa. "Matlab/Octave". (http://nereida.deioc.ull.es/~pcgull/ihiu01/cdrom/matlab/contenido/node2.html).

[13] Información extraída de "Design and test algorithms for robotics applications" (https://es.mathworks.com/products/robotics.html).

[14] Información extraída de "Procesamiento, análisis y desarrollo de algorítmos de imágenes" (https://es.mathworks.com/products/image.html).

[15] Información extraída de "Virtual Robot Experimentation Platform USER MANUAL". (http://www.coppeliarobotics.com/helpFiles/).

Universidad de Alcalá
Escuela Politécnica Superior

ESCUELA POLITECNICA
SUPERIOR