Ambiente de simulación de barcos a vela para el entrenamiento de sistemas de navegación autónomos.

John Camilo Giraldo López

Trabajo de grado presentado para optar por el título de:
Ingeniero Electrónico

Asesor
Ricardo Andrés Velásquez Vélez, PhD

Universidad de Antioquia
Facultad de ingeniería, Departamento de ingeniería electrónica y de telecomunicaciones
Ingeniería electrónica
Medellín
2023

Cita	Giraldo L.
Referencia Estilo IEEE (2020)	[1] Giraldo, J. Camilo., “Ambiente de simulación de barcos a vela para el entrenamiento de sistemas de navegación autónomos”, Trabajo de grado, Ingeniería Electrónica, Universidad de Antioquia, Medellín, 2023.

SISTEMIC

Repositorio Institucional: http://bibliotecadigital.udea.edu.co

Universidad de Antioquia - www.udea.edu.co

Rector: John Jairo Arboleda Céspedes.

Decano/Director: Julio César Saldarriaga Molina.

Jefe departamento: Eduard Emiro Rodríguez Ramírez.

Dedicatoria

Dedico este proyecto a quienes no se rinden y no saben el significado de la palabra imposible a la hora de asumir un reto, a quienes prefieren decir “no pude ahora” a “no se puede”, a estos que dedican toda su vida a lograr las cosas que algunos dijeron que eran imposibles. Son ustedes los dueños del mundo y los capaces de lograr su progreso.

Agradecimientos

Agradezco a mis padres, compañeros y amigos que me ayudaron en todo mi proceso formativo a quienes alguna vez me dieron una palabra de aliento o compartieron una risa conmigo en una charla de pasillo, pues me inspiraron a seguir adelante aun cuando yo mismo no creía en mí. Agradezco al grupo SISTEMIC por invitarme a participar de su proyecto, a Open robotics y el subproyecto VRX por brindar las herramientas para desarrollarlo. Finalmente agradezco a la Universidad de Antioquia y la facultad de ingeniería por todo el conocimiento y apoyo brindado.

TABLA DE CONTENIDO

LISTA DE FIGURAS

SIGLAS, ACRÓNIMOS Y ABREVIATURAS

IEEE	Institute of Electrical and Electronics Engineers
ROS	Robot Operating System
MCU	Micro Controlling Unit
USV	Unmaned Surface Vehicle
HITL	Hardware in the Loop
OOP	Object Orientated Programming
DT	Digital Twin
URDF	Universal Robot Description File
SDF	Simulation Description File

RESUMEN

En el campo de los sistemas embebidos y bajo el marco del proyecto STINT, el cual se desarrolla en colaboración con la Universidad de Mälardalen en Suecia y la Universidad Tecnológica de Panamá, el grupo SISTEMIC trabaja en el subproyecto Autosail. Para el cual se plantea la realización de un velero completamente autónomo. Uno de los requerimientos de dicho proyecto consiste en la sintonización de controladores para el control de actuadores del vehículo no tripulado, sin embargo, con el objetivo de no arriesgar la integridad del vehículo y sus componentes electrónicos, se plantea la realización de un entorno de simulación en el que se consideren diversos escenarios, además de las condiciones propias del vehículo en cuestión para que se interactúe dinámicamente con el entorno virtual y real, mediante estrategias como el Hardware in the loop. A través del modelamiento detallados del vehículo, sus condiciones, actuadores, entorno y una versión básica del controlador para propósito de pruebas, se logra emular una navegación autónoma y como valor agregado para próximos desarrolladores y/o usuarios, se generó una Interfaz gráfica amigable para introducir modelos, controladores y comandos al entorno de simulación.

Palabras clave — **Sistemas embebidos, modelo, autonomía, interfaz gráfica.**

ABSTRACT

On the field of embed systems and framed on the STINT collaboration project, developed with Mälardalen's University and Panama's Technological University. SISTEMIC group Works on the subproject Autosail, with the objective of making a completely autonomous sailing boat. One of the requirements of the Project is to make the controllers syntonization for the unmanned vehicle, but rather than risking the vehicle integrity or it's electronical components, it's planned to make a full simulation environment that considers multiple sceneries, self-conditions of the sailing boat to dynamically interact with a virtual and real environment by means on strategies as hardware in the loop. Through the detailed modelling of the vehicle, its conditions, actuators, and environment and a simple version of the controller for testing purposes, we emulated an autonomous navigation and as an added value for next developers a friendly graphical interface to introduce models, controllers and commands to the simulation environment.

Keywords — **Embed systems, model, autonomy, graphical interface .**

I. INTRODUCCIÓN

Previo al planteamiento del presente proyecto, se hizo un análisis de las herramientas y avances que poseía el proyecto en marcha del grupo SISTEMIC: Autosail, y entre los problemas detectados se encontraba que las herramientas prexistentes del proyecto usaban versiones de software abierto que son obsoletas o que requieren versiones de sistemas operativos muy antiguas y específicas para funcionar, lo que dificultaba la actualización y/o modificación de controladores y modelos anteriores debido a la dificultad de intervenir dichos sistemas con nuevos paradigmas o herramientas.

En el apartado de modelamiento se visibilizó que el modelo utilizado previamente no se acomodaba a las especificaciones reales del proyecto, lo que lo hace poco fiable para representar la implementación de pruebas de seguimiento de trayectoria, sensores, peso, entre otros.

Además, al correr la simulación que previamente se había utilizado se evidenció, que los simuladores prexistentes no poseían una interfaz gráfica que facilitara la modificación de condiciones de simulación, la selección de la embarcación o el tipo de estrategia de navegación, entre otros parámetros que podrían ser básicos a la hora de realizar pruebas en un ambiente embebido.

Finalmente, con respecto a los requerimientos de colaboración internacional en el proyecto, se discutió que para homologar las herramientas de diseño de software se hace necesario que todos los sensores y/o actuadores puedan sostener una comunicación única la cual ya había sido planteada por los estudiantes de la Universidad de Mälardalen, Suecia; el sistema(interfaz) que utilizan los equipos de desarrollo en el mundo: ROS(Robot Operating System) que implementa librerías de software que simplifican la comunicación entre diversos tipos de Hardware mediante un sistema de mensajes asincrónicos. Esta tecnología, aunque conocida por los miembros del proyecto, no habría sido implementada dentro de los sistemas presentes.

Detectados todos estos problemas, se plantea entonces como propuesta al mejoramiento del proyecto Autosail, la realización de un nuevo y mejor documentado “ambiente de simulación de barcos a vela para el entrenamiento de sistemas de navegación autónomos” que subsanará las falencias que se observaran al interior del proyecto.

II. OBJETIVOS

A. Objetivo general

Implementar un ambiente de simulación de barcos a vela, que permita el entrenamiento de sistemas de navegación autónomo, utilizando técnicas de hardware-in-the-loop y digital-twin, y herramientas como ROS y Gazebo.

B. Objetivos específicos

- Implementar el ambiente de simulación que permita simular la navegación de barcos a vela, utilizando Gazebo y paquetes de modelado de variables físicas.
- Modelar un barco a vela dentro del ambiente de simulación, los actuadores y los sensores virtuales asociados con el control del barco utilizando ROS.
- Desarrollar una interfaz gráfica que permita cambiar las condiciones ambientales del ambiente de simulación, seleccionar el tipo de embarcación y la estrategia de control para facilitar el uso del simulador a personas no familiarizadas con Gazebo.
- Desarrollar una estrategia de simulación de Hardware-in-the-loop soportada en ROS que permita el uso de MCU reales para la evaluación de estrategias de navegación.
- Evaluar el ambiente de simulación con un caso de estudio en el que se sintonice un sistema de navegación basado en controladores PI.

III. MARCO TEÓRICO

Para el caso del proyecto en cuestión es particularmente importante el tener un modelo que asemeje las características del velero que se posee, dado que las condiciones de trabajo objetivo son difíciles de contemplar en un ambiente dentro de la universidad de Antioquia sede ciudadela y no existe un escenario de pruebas adecuado disponible, además de que el proceso mismo de pruebas implica posibles riesgos para los componentes electrónicos debido a la alta humedad y la probabilidad de inundación o hundimiento que existe en caso de fallas en los controladores.

A. Digital twin (DT o gemelo digital)

Es una tendencia emergente de la tecnología que está rodeada de promesas y potencial de reestructurar el futuro de las industrias. El gemelo digital es un sistema de sistemas que va más allá de la simulación por computadora tradicional. Consiste en la replicación de elementos, procesos, dinámicas, y soporte lógico en una contraparte digital. Ambas partes existen simultáneamente compartiendo sus entradas y operaciones usando datos en tiempo real y transferencia de información. Esta tendencia incorpora distintas tecnologías como: Internet of things (IoT), la inteligencia artificial (IA), los modelos 3D, las comunicaciones móviles y la realidad aumentada, entre otros. El DT ofrece una plataforma para probar y analizar los sistemas complejos, que sería imposible lograr para los simuladores tradicionales y las evaluaciones modulares.[2]

Hardware in the loop simulation (HITL)

Es una técnica ligada a la simulación de sistemas complejos que consiste en cerrar la brecha que existe entre la simulación y las condiciones reales por medio de la operación real de componentes de un sistema en conexión con componentes simulados. Esto debido a que la experimentación en los procesos controlados puede conllevar un costo elevado o grandes cantidades de tiempo de producción para concretarse.

La gran ventaja de probar con la técnica HITL reside en la posibilidad de realizar una prueba más realista para validar los sistemas de control utilizando hardware real e incluir efectos en tiempo de ejecución sin tener que usar el sistema real, lo que es particularmente útil cuando se requiere probar el sistema de control en condiciones extremas ayudando a diagnosticar fallas y/o malfuncionamientos sin dañar o destruir el sistema desarrollado. [3]

Programación Orientada a Objetos

La programación orientada a objetos OOP es un paradigma de la programación en el que se generan entidades (objetos) capaces de heredar, encapsular y procesar información de manera específica con el propósito de darle al usuario una serie de herramientas para la interacción con cada entidad, y modularizar el código fuente para la creación de múltiples entidades similares en estructura; a la vez que al desarrollador lo dota de la capacidad de restringir el acceso a variables y características para el acceso exclusivo de dichas entidades, sus descendientes y alcances definidos en el programa.

Para el modelamiento del proyecto se hizo necesario el modelamiento de entidades posibilitando modularizar el código y generalizar algunos de los apartados para que usuarios menos expertos pudiesen utilizar los desarrollos, volviendo más amigable el código.

Física del Velero

Al margen de que existe un modelo real sobre el cual trabajar, realizar modificaciones sobre el mismo implica cambiar la dinámica e interacciones puesto que afectará las variables del modelo como la distribución de masa y los momentos de inercia sobre los distintos ejes. Además, como sugiere la técnica de HITL probar condiciones extremas puede ser riesgoso para el modelo, así mismo emular dichas condiciones en un entorno realista puede ser cuando menos difícil o tal vez imposible con los recursos actuales que posee el grupo SISTEMIC. Por tanto, a la hora de digitalizar el modelo se hace necesario conocer a profundidad la dinámica estática general de los vehículos acuáticos superficiales como lo son: barcos, buques, veleros, catamaranes, entre otros.

La hidrostática: se refiere a la rama de la física que estudia el equilibrio de los fluidos. De esta rama se analizan conceptos como:

Centro de gravedad: es el punto en el que se ven representados todos los efectos de las fuerzas aplicadas a un cuerpo sólido debido a su masa.

Peso: es la fuerza de atracción debida a la masa a la que están sometidos todos los cuerpos, es directamente proporcional a la misma e inversamente proporcional al cuadrado de la distancia entre sus centros de gravedad.

Empuje: es la fuerza ejercida por un fluido desplazado sobre la superficie de un objeto debido a la diferencia de densidades.

Centro de carena: así como la fuerza que se ejerce sobre un sólido se ve reflejada sobre un punto al cual se le denomina centro de gravedad, también para el empuje se emplea el centro de carena, que es el punto donde se ven representados todos los efectos de la fuerza de empuje y coincide en los fluidos homogéneos con el centro geométrico del volumen sumergido.

Metacentro: se define como el eje de corte entre el plano diametral de la embarcación y el plano perpendicular al plano principal de flotación de la embarcación, que pasa por el centro de carena.

Estabilidad hidrostática: flotar se define como como la capacidad de un objeto de sostenerse encima o estar en suspensión en una sustancia liquida o gaseosa. Físicamente esta capacidad implica que:

- Para todo efecto el peso total del vehículo sea igual que la fuerza de sustentación (empuje) en su superficie, dada por el volumen de fluido desplazado. (principio de Arquímedes $masa = \gamma_{fluido} * Volumen\ deplazado$)
- Que el centro de carena y el centro de gravedad estén sobre el mismo plano vertical. (plano adrizante).
- Como condición absoluta en vehículos completamente sumergidos se debe cumplir que el centro de gravedad esté por debajo del centro de carena en condición estable. **

Nota:(**) En caso de que no esté completamente sumergido simplemente se toma como condición suficiente para ser estable.

Estas condiciones fuerzan a que se produzca el brazo adrizante que será el responsable de devolver al velero a su posición adrizante (verticalmente estable).

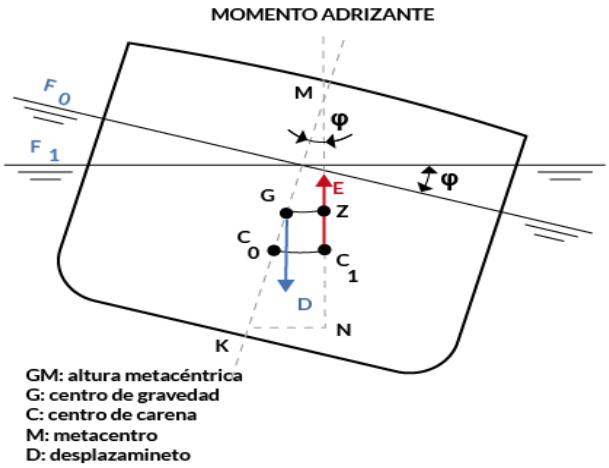


Figura 1. Fuerzas que influyen en la hidrostática y estabilidad transversal de un bote [12]

Dinámica del velero

La navegación de veleros depende no solamente de que el velero se mantenga a flote sino, además, de un conocimiento de las fuerzas que afectan el movimiento de la embarcación para seguir el curso deseado dentro del entorno acuático superficial. Esto implica entonces también que las fuerzas aerodinámicas que se encargan de empujar el velero mantengan un

equilibrio con las fuerzas hidrodinámicas para mantener la estabilidad del vehículo y aprovechar al máximo la energía recolectada. (Wilson, Ryan M., 2010)

Para el caso del velero es necesario conocer entonces la fuerza que lo propulsa (el viento), además de otras fuerzas que lo retienen y lo moderan a la hora de avanzar. En la dinámica de fluidos el contacto de una superficie con un fluido produce dos efectos:

Levantamiento(lift): se refiere al efecto que genera un fluido al pasar por un objeto en la componente perpendicular al flujo y está directamente relacionado con la forma del objeto en cuestión.

Arrastre(drag): se refiere al efecto que genera un fluido al pasar por un objeto y que se opone a la dirección del movimiento del objeto.

Estas fuerzas operan para fluidos tanto líquidos(agua) como gaseosos(aire) y dependen enteramente de la densidad del fluido y la forma de los objetos, normalmente deben ser medidos experimentalmente para ser implementados.

Presumiendo que se desea un movimiento relativamente uniforme entonces se considera que la condición para lograr este tipo de movimiento es:

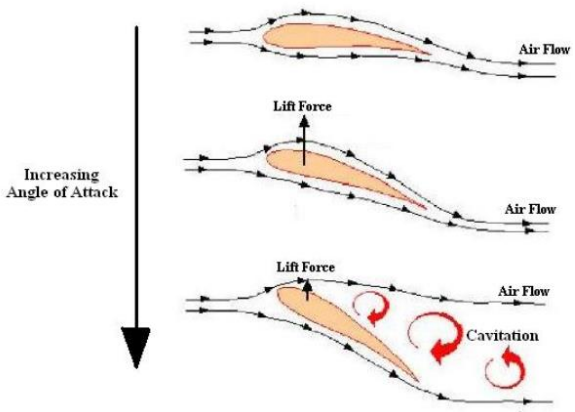


Figura 2. Efectos dinámicos del cruce por un fluido [8]

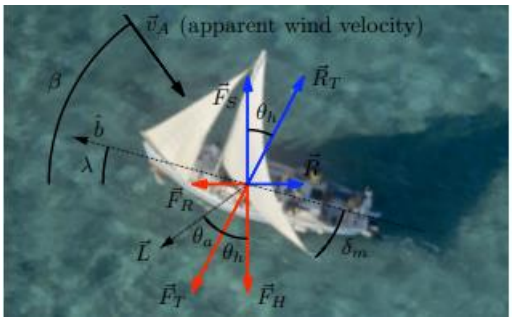


Figura 3. Fuerzas aerodinámicas e hidrodinámicas en un velero [4]

Que el levantamiento generado en la vela principal se contrarreste con el levantamiento generado en la quilla del velero lo cual anula el torque y la sumatoria de fuerzas en la superficie del velero en direcciones diferentes de la trayectoria aparente.

Herramientas de código abierto

Simuladores

Existen algunos simuladores de barcos de vela como eSail o Sailaway los cuales brindan una idea del manejo del barco, sin embargo, dichos simuladores solo consideran sus propios modelos, además de que su enfoque está dirigido al entrenamiento de personal para la navegación y el reconocimiento de la instrumentación de los botes reales, por lo que los entornos se encuentran controlados para realizar estas tareas y no admiten la implementación de código de fuentes externas.

También existen simuladores de robótica marítima que se enfocan al desarrollo de controladores para robots y entornos preestablecidos como lo son USV (Unmanned Surface Vehicle: Desarrollado a partir de gazebo classic), UWS, Stonefish, UUV (Unmanned Underwater Vehicle), entre otros; que permiten realizar distintos tipos de análisis en la cinemática de robots marítimos, pero que no cumplen con todos los requisitos del proyecto, como la posibilidad de integrar estrategias HITL, el modelamiento de ciertos comportamientos físicos, la integración de MCU’s y módulos externos al código fuente.

ROS

Robot Operanting System (ROS): son un conjunto de librerías y herramientas que ayudan en la construcción de aplicaciones en robótica. Desde drivers hasta algoritmos que lindan con el

estado del arte, y herramientas de desarrollo en lenguajes como C++ y Python, disponibles para que cualquier desarrollador las use libremente. (Open Robotics, 2023).

ROS es un sistema que busca implementar un estándar para la simplificación de la codificación y la modularización de código fuente a través de un sistema asíncrono de comunicación en red de los dispositivos conectados.

Gazebo Simulator

Gazebo Simulator: Mientras que ROS permite desarrollar proyectos de robótica desde el punto de vista del software, Gazebo es un simulador Open-Source que permite la integración de software ROS, además modelamiento de mayas visuales en aplicaciones de diseño como: Inventor, OpenSCAD, Blender, Solidworks, entre otras; motores de física que soportan interacciones: gravitacionales, colisiones, hidrostática, hidrodinámica, aerostática, aerodinámica, modelamiento de articulaciones, cinemática directa e inversa, y más características. (Open Robotics,2023)

Gazebo es una plataforma de simulación de código abierta diseñada y mantenida por la misma empresa responsable de ROS, que busca ofrecer a los usuarios de ROS una implementación realista de sus algoritmos de control en condiciones controladas para facilitar el desarrollo y producción de nuevos automatismos robóticos.

IV. METODOLOGÍA

1) Implementación del ambiente de simulación que permita simular la navegación de barcos a vela, utilizando Gazebo y paquetes de modelado de variables físicas.

a. Investigación estado del arte.

Se realizó una investigación del estado del arte de las herramientas disponibles para la implementación de simuladores que soporten la física hidrostática e hidrodinámica dentro de sus entornos, encontrando en el proceso varias herramientas de distintos indoles como lo pueden ser:

- UUVSim (Unmanned Underwater Vehicles)
- USV Sim (Unmanned Surface Vehicles Simulator)
- Stonefish simulator
- UWSim(Under Water Simulator).
- Gazebo

Dado que ROS ya era un requisito dentro de los parámetros del proyecto y Gazebo Garden es la última versión finalizada del paquete provisto por Open Robotics se optó por ella para evitar posibles incompatibilidades entre el código desarrollado y el usado por los pares académicos.

b. Revisión documental del proyecto.

Se revisaron y estudiaron todos los archivos del proyecto Autosail para determinar el alcance del mismo y los cambios a realizar llegando a las siguientes conclusiones:

- Todos los controladores debían ser actualizados puesto que, aunque generaban una respuesta en teoría correcta a los estímulos físicos detectados por los sensores, no compartían una interfaz de comunicación estandarizada con los sensores que usan actualmente los pares académicos de la universidad Mälardalen, Suecia; ni con la herramienta de simulación que presentaba el proyecto previamente por lo que se debieron crear nuevas interfaces para dicha comunicación. Lo que dificultaba el proceso de desarrollo para todos los involucrados.
- Rediseñar las interfaces de los MCUs (Rasphberry pi pico) aunque compatibles con las librerías de código abierto de micro-ROS, serían un retraso importante al proyecto puesto que sería muy costoso rediseñar toda la lógica que ya se tenía implementada para estos desde los registros, solo para hacerla compatible con el nuevo sistema. Se optó entonces, por redactar un traductor que convirtiera los datos recibidos a través de la interfaz UART previamente implementada, en nuevos mensajes y estructuras dentro del sistema de ROS.
- El sistema actualmente implementado era complejo y carecía de los módulos que la implementación de la arquitectura de ROS provee a los sistemas embebidos. Para cumplir con los objetivos de cooperación internacional, se hizo necesaria la modificación del apartado de control del código fuente del proyecto para generar una estrategia que posteriormente posibilitara las estrategias HITL.

c. Comprensión del funcionamiento de las herramientas

Conocimiento de las nuevas herramientas a implementar, cómo manipularlas, modelarlas y modificarlas dentro del entorno de trabajo.

Para comprender las herramientas se inició por una búsqueda del estado del arte que comprendió el análisis profundo de la documentación de ROS haciendo especial énfasis en los siguientes apartados:

Instalación

Para la instalación previendo el uso de herramientas de simulación en el futuro cercano se realizó (tal y como se recomienda) una instalación desde la fuente. Los pasos se documentaron en un script de consola que será anexado al final (ANEXO 1).

Nodos (Nodes)

Son las células inteligentes dentro del sistema de ROS, responsables de configurar y administrar las interacciones dentro del sistema, al interior de ellos se crean: publicadores, funciones, suscriptores, servicios, acciones, entre otras interacciones que se modelan. Además, son en sí, estructuras que configuran a un tipo de arquitectura de software que puede ser repetida indefinidamente en el sistema y ser diferenciada mediante una identificación adicional, es decir, módulos u objetos de código.

Tópicos (Topics)

Se entiende por tópicos el nombre público que le damos a cada una de las interfaces (colas) de estructuras de mensajes que queremos mandar a través del sistema de ROS. Cada mensaje que se envía en el sistema tiene una estructura fija que debe ser decodificada por el otro extremo de la comunicación, y es identificada mediante el nombre del tópico y al ser pública varios nodos pueden obtener información de ella.

Servicios (Services)

Son un tipo de interfaz que implementa estructuras de requisitos y estructuras de respuesta simultáneamente, por tanto, cada que son invocadas, estas generan una petición a través de un nodo cumpliendo los requisitos y esperan una respuesta que se acomode a los datos de la estructura por parte del nodo receptor de la petición. Varios nodos pueden solicitar el mismo servicio a otro y obtener una respuesta individual según los requisitos enviados.

Archivos de lanzamiento (Launch Files)

Los archivos de lanzamiento son aquellos encargados de que se activen simultáneamente configuraciones de sistemas predefinidos, es decir, el entorno completo de trabajo para una aplicación específica: nodos, tópicos, servicios, acciones y demás interacciones.

Particularmente y para el proyecto fueron indispensables dada la complejidad que implica el sistema implementado.

2)Modelado de un barco a vela dentro del ambiente de simulación, los actuadores y los sensores virtuales asociados con el control del barco utilizando ROS.

a. Implementación en Hardware.

Se implementó dentro del hardware de control la lógica del set de librerías de código abierto ROS, más específicamente en su segunda versión y la distribución más reciente: Humble Hawks Bill.

Para ello, se implementó un nodo con el objetivo de codificar y decodificar los comandos, desde y hacia los MCU's para introducirlos dentro del sistema de ROS de manera que se pudiera desarrollar la modularidad del código en pro de avanzar en los objetivos colaborativos para con las universidades extranjeras.

El nodo decoder mediante la librería pySerial toma información de una secuencia separada por caracteres 'x' a través de un puerto COM codificada según la siguiente tabla:

TABLA I
ETIQUETAS DE IDENTIFICACION DE SECUENCIA

ID	**Dato**
S1	Medida de latitud dada por el GPS
S2	Medida de longitud dada por el GPS
S3	Velocidad del velero
S4	Dirección de la velocidad del velero
S5	Velocidad del viento aparente
S6	Ángulo pitch del velero
S7	Curso actual del velero dado por el magnetómetro
S8	Dirección del viento aparente
A1	Acción de control del timón en grados (-45,45)°
A2	Acción de control de la vela principal (-90,90)°
A3	Acción de control de la vela secundaria (-90,90)°

Para luego extraer la información relevante en el sistema ROS, se plantearon entonces estructuras de mensajes personalizadas para almacenar todas las variables como:

- Viento: Velocidad del viento(float), dirección del viento(float).

- IMU: Aceleración lineal (vector 3D), Aceleración angular (vector 3D).
- GPS: Latitud(float), Longitud(float).
- Magnetómetro: Magnitud del campo magnético (vector 3D).
- Timón: Posición timón(float).
- Vela Principal: Posición de la vela principal(float).

Cada estructura fue asociada a un tópico para ser identificada por ROS, luego de tener esta información cada tópico podía ser consultado por diversas entidades o nodos conectados a la red.

b. Medición del velero.

Se realizaron las medidas de las dimensiones físicas del velero de referencia, para igualar sus dimensiones y modificar así la interacción del modelo previo del velero con el entorno virtual de pruebas. (Véase la figura 4 y 5)

c. Modelación visual del velero.

Posteriormente, se modelaron con ayuda de programas de diseño asistido por computadora (CAD) en 3D componentes simplificados del velero de referencia.

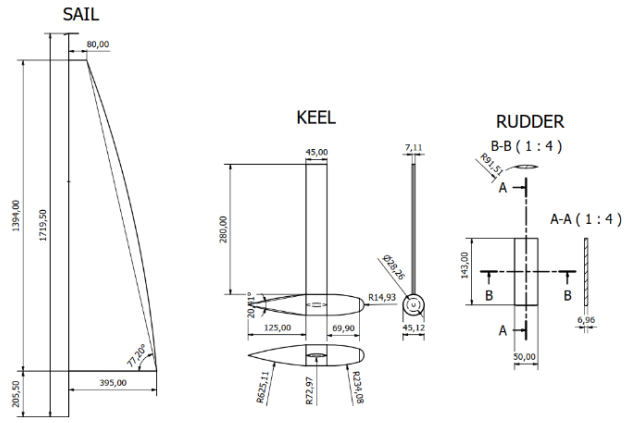


Figura 4. Medidas del velero de referencia para la vela, la quilla y el timón

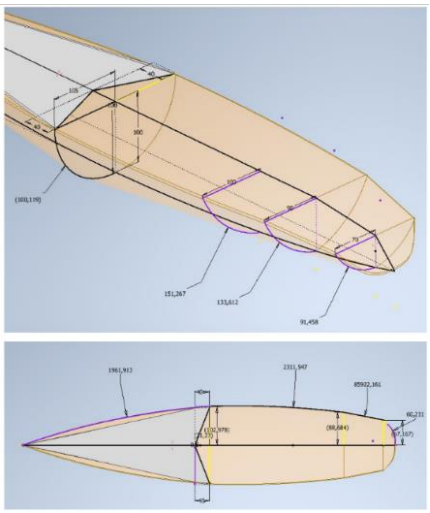


Figura 5. Medidas del bote del velero de referencia

d. Ensamblaje.

Se modeló entonces el ensamblaje del velero de referencia como un 'robot' definiendo sus límites articulares por medio de un archivo URDF. Para lograr este objetivo se hace necesario entonces conocer la cinemática básica del robot descomponiéndolo en partes o piezas, articulaciones, conociendo el tipo de movimientos e interacciones a las que se vería sometido. Como primera instancia se decidió dividir el modelo en partes(links) mínimas como se describe a continuación:

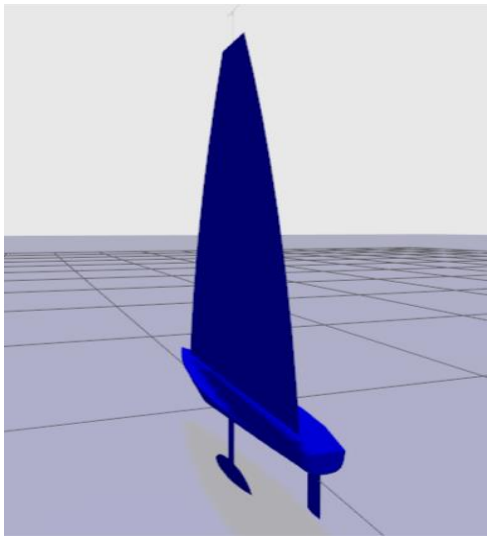


Figura 6. Ensamblaje del velero

Partes del velero como robot

- Bote (base_link): para propósitos prácticos el bote es la parte principal porque es a partir de el que se definen las posiciones relativas de los demás componentes. Es objeto de interacciones como: la flotabilidad, la estabilidad y es el componente más susceptible a los cambios súbitos del barco
- Vela principal (main_sail_link): Es la fuente de energía cinética del velero, pero también debido a la presión que se genera sobre ella podría causar el volcamiento de la embarcación, es objeto de interacciones como: colisión con el viento, levantamiento y arrastre.
- Timón (rudder_link): es parte encargada de dirigir el bote dentro del agua, es susceptible de las mismas interacciones que la vela principal, sin embargo, el fluido de contacto para su caso es el agua, lo que hace que, aunque el área de contacto se mucho menor las interacciones que generan sean mucho notorias.
- Quilla (keel_link): Es una pieza que provee estabilidad hidrostática e hidrodinámica al bote debido a que dirige el centro de masa hacia abajo forzando a que se cumpla la condición de estabilidad #3 y facilitando la existencia del brazo adrizante, a la vez que al estar sumergido en el mismo fluido que el timón, sufre también los efectos de levantamiento y arrastre, pero son estos mismos efectos los que contrarrestan a las fuerzas de la vela generando un par contrario.
- Sensores: Son partes adicionales que dotan al modelo de capacidad para capturar variables del entorno simulado y transmitirlos a sistemas fuera del sistema de simulación.
- Vela secundaria (secondary_sail_link): al igual que la principal genera(captura) parte de la energía cinética que usa el barco para impulsarse, sin embargo, en el momento permanece inhabilitada.

Como modelar las partes (Links)

En el formato URDF se modelaron los links a partir de tres características principales que define la herramienta de simulación:

- Colisión: es la maya de puntos con la que se interactúa con los demás objetos de la simulación con respecto al contacto, puede ser creada a partir de figuras simples como: esferas, cilindros o prismas rectos; o ser importada a partir de una maya preexistente modelada en formato *.dae(collada) o *.stl. Su pose (vector de posición: denota la posición relativa en los 3 ejes cartesianos; y rotación: la rotación relativa sobre los mismos) es independiente a las demás

características, y puede modificar las interacciones con otros objetos como la flotabilidad, el choque y los efectos de viento

- Inercia: en esta característica se toma la matriz de sus tensores de inercia para definir el modelamiento de los ejes sobre los que puede girar un objeto y su interacción con fuerzas externas relacionadas a la masa; la posición que defina esta característica será el centro de masa para el objeto, incluso si en ese espacio no existiera maya de colisión o visual para el mismo.
- Visual: tal como lo expresa su nombre es la capa donde se aloja la escena visual del objeto, esta es la capa que interactuara con luces, sombras, y reflejos, es la capa más difícil de modelar e implica al igual que para la colisión crear o importar una figura esta vez aseverando no solamente en la dimensión de la misma sino también su color y textura.

Articulaciones o juntas (Joints)

Para cada link existe una articulación asociada para relacionar la cinemática de este con el robot, e interpretarlo como una entidad compuesta dentro del entorno de simulación.

Se modelaron entonces las articulaciones a partir de las siguientes características:

- El tipo de articulación(movimiento): en el formato URDF se contemplan articulaciones para realizar movimientos de revolución, prismáticos, universales, de bola y fijos.
- Limites articulares: modelan los límites de desplazamiento, velocidad, esfuerzo, el coeficiente de fricción, el coeficiente de amortiguamiento, entre otros. Que ayudan a equiparar el comportamiento físico real con el simulado.
- Ejes: define por medio de un vector director el eje (o los ejes) de acción de la articulación, puesto que no es obligatorio usar uno de los ejes principales cartesianos.

Interacciones y sensores

Para dar respuesta a este requerimiento y dado que las interacciones son de múltiples naturalezas, se hizo un análisis detallado de las condiciones de cada link y articulación, las usadas para modelar el velero van desde:

- El empuje o fuerza de flotación: por medio del plugin del sistema de flotación gz::sim::systems::Buoyancy se definen los links que son afectados por la fuerza de flotación, el plugin toma las mayas de collision del link (solo figuras primitivas) para generar la flotabilidad del objeto.

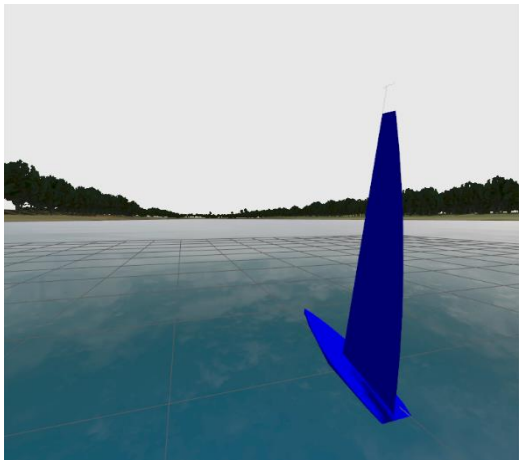


Figura 7. Velero dentro de entorno de pruebas modelado

- Los efectos hidrodinámicos: de manera similar se cargan los efectos hidrodinámicos para el link con el sistema de sustentación y arrastre gz::sim::systems::LiftDrag

- Los efectos de viento: gz::sim::systems::WindEffects

- El control de las articulaciones: gz::sim::systems::JointPositionController

- La publicación del estado de las entidades: gz::sim::systems::JointStatePublisher y gz::sim::systems::PosePublisher

- El control de características luminosas a partir de la inserción de luces en la escena
- La creación de sensores simulados: IMU, GPS, Cámara y Magnetómetro dentro de cada modelo (Estas características deben ir contenidas dentro de un Link)

Se importan estas características a cada entidad que las requiera a través de los plugins que no son más que fragmentos de código con propósito específico para implementar un nuevo comportamiento o característica dentro de la simulación.

Para cada una de las partes se modelan a partir de las estructuras descritas de los archivos URDF según el estándar descrito a detalle en http://wiki.ros.org/urdf/XML/model y que luego sería traducido para comodidad de la herramienta de simulación (Gazebo Simulator) en un archivo SDF (Simulation Description Format).

e. Modelado del entorno de pruebas.

Se modeló en 3D el entorno de prueba, similar al anterior, con la distinción de que para el entorno de simulación se hizo mediante la importación de modelos preexistentes en un mundo en formato SDF (playa, muelles, estaciones, edificios, arboles, entre otros). Modelando así, un entorno complejo mediante módulos(modelos) pequeños y escalables a mundos cada vez más grandes.

f. Modelación de interacciones.

Se modelaron las interacciones entre el entorno de prueba y el velero de referencia. Para el mundo se plantearon características como: la ubicación geográfica, el nivel del agua, el viento incidente en la superficie, la luz solar, entre otras.

Mientras que, en el barco, teniendo como referencia el marco teórico y las limitaciones que provee la herramienta de simulación, se realizaron 2 capas:

*C*apa visual.

Es una capa concebida para que asemejara la forma que se observa del velero e interactuara con las luces y las demás formas a partir de las medidas tomadas del modelo de referencia y acogidas en la característica visual del modelo importando los esquemas previamente realizados.

La implementación de esta capa fue sencilla dado que ya se poseían medidas y modelos CAD del velero de referencia, por tanto, bastó con realizar la importación de dichos modelos por medio de BLENDER para ajustar las dimensiones de los archivos *.stl en un archivo *.dae que portase además de la matriz de puntos que proporciona la maya de la figura, la ubicación y pose del origen cinemático de la misma.

*C*apa física.

Se diseño una capa que implicara las interacciones mediante un modelo minimalista de los efectos físicos en figuras menos complejas como cilindros, esferas y prismas rectangulares, esta capa se acoge las características de colisión e inercia del modelo que uniéndose podrían asimilar la complejidad del velero para las interacciones físicas. Obteniendo el siguiente modelo:

Para la implementación de esta capa se tuvieron muy en cuenta las consideraciones del marco teórico acerca de la estabilidad al ubicar las mayas de colisión sobre el centro masa, la distribución de peso al ubicar el centro de masa en el mismo plano que el centro de carena, la inercia de los ejes al escoger las figuras simples que reemplazacen los efectos que representaran al barco, entre otros.

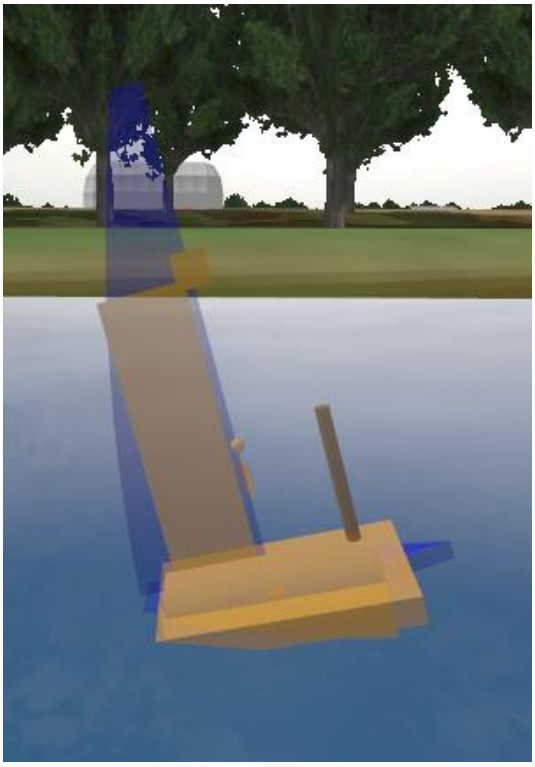


Figura 8. Abstracción de las mayas de colisión en figuras primitivas.

Durante la ejecución de pruebas se hizo necesaria la creación de un Link adicional para marcar la dirección del viento, este link no tiene componente visual puesto que su propósito es el de implementar un sensor simulado de dirección mediante el gz::system::jointStatePublisher. que indique la dirección relativa del viento durante la navegación. Para esta implementación se tuvo en cuenta el principio de una veleta mecánica.

g. Codificación de puentes.

Se codificó la interfaz de controladores con el entorno de prueba. Ya que la herramienta de simulación es un sistema completamente ajeno a los controladores previamente configurados y codificados por ROS, se hizo necesaria la codificación de puentes para conectar las partes con ayuda de un sistema previamente establecido por los diseñadores de ambas herramientas instalando así, en la carpeta de compilación, los paquetes necesarios para la creación de estos puentes. (ANEXO 2)

h. Reporte parcial.

Para este punto, se reportaron los avances en el desarrollo del DT en el entorno de pruebas realizando una navegación en el entorno de pruebas simplificado mediante control por comandos y controles en pantalla propios de las herramientas de simulación (gazebo: joint position controller) obteniendo una navegación satisfactoria por parte de los veleros de prueba.

3) Desarrollo de una interfaz gráfica que permita cambiar las condiciones ambientales del ambiente de simulación, seleccionar el tipo de embarcación y la estrategia de control para facilitar el uso del simulador a personas no familiarizadas con Gazebo.

a. Definición de requerimientos

Se definieron en conjunto con el asesor los requerimientos necesarios para la GUI que se deseaba, se llegó a la conclusión de que la GUI requería de modelar:

- Una interfaz sencilla e intuitiva que iniciara requiriendo los parámetros de la simulación, como: nombre del robot, ubicación del modelo, mundo a lanzar, tipo de simulación, tipo de conexión, controlador, ubicación del controlador.
- Se planteo entonces una nueva pestaña para realizar el control de los aspectos propios de la simulación y observar su progreso en tiempo real: La dirección del barco, dirección del viento percibido o la dirección de la vela.
- Finalmente, y como un valor agregado se adicionó una pestaña para realizar el rastreo de la telemetría del bote, y la planeación de rutas al interior del simulador o la zona de pruebas.

b. Interfaz de usuario

Dentro de la interfaz se plantearon objetos dinámicos para el usuario que le ayudasen a interactuar y luego se codificaron las funciones para modificar los parámetros de simulación. Con

los respectivos comandos de modificación asociadas a eventos como el click, deslizamiento de sliders y la modificación de ciertos parámetros se generaron ordenes que modificaran la simulación.

c. Instalación y uso de interfaz

Para el uso de la interfaz gráfica serán necesarios los siguientes pasos en un entorno de trabajo de Linux Ubuntu 22.04(jammy):

i. Crear el espacio de trabajo y descargar el repositorio el repositorio

sudo mkdir -p ~/autosail/src

git clone –b main https://github.com/JohnkaGL/Autosail_ws.git

ii. Corriendo el script bash para confirmar que todos los requisitos se cumplen para correr el ambiente

cd ~/autosail/src

sudo ./requirements.sh

iii. Construir el entorno y disponer como directorio fuente el espacio de trabajo creado:

cd autosail_ws

colcon builld –merge-install

source install/setup.bash

iv. Finalmente, para correr la configuración de la GUI se debe correr el comando:

ros2 run GUI GUI_start

La interfaz

Pantalla de bienvenida: En esta pantalla se configura la simulación

En esta ventana el usuario debe configurar la simulación:

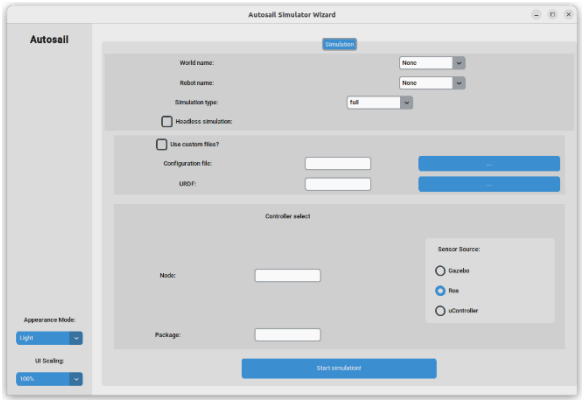


Figura 9. GUI página de inicio

-Elegir el mundo en el cual se desea simularse: Se proporciona una lista desplegable con mundos previamente diseñados, sin embargo, dado que es una casilla editable si se desea se puede introducir el nombre de un mundo personalizado ubicando previamente el archivo SDF en la carpeta descargada autosail_ws/src/autosail_gz/worlds.

- El modelo de robot que desea crear: Este será el nombre con el que se identificará los modelos a crear controlado y al igual que el anterior hay una lista de modelos previamente creados, y de la misma manera se podrían introducir paquetes de modelos para realizar un spawn (aparición).
- El tipo de simulación que desea generar: existen 3 tipos de simulaciones sim, bridge, full. Para la primera simulación se genera la aparición del modelo en el mundo sin llegar a generar los puentes para enlazar el modelo con el sistema de ROS, independientemente de que sea con o sin GUI el modelo existe para el sistema del simulador solamente. Para el segundo existen los puentes entre ROS y GAZEBO, sin embargo, se entiende que el modelo no requiere de ser creado, pues preexiste dentro del archivo del mundo y los archivos de configuración (personalizados); Para el tercero se crean tanto los puentes como el archivo de modelo que va a aparecer en el mundo.

-En la simulación puede elegirse si desea o no el despliegue gráfico (para sistemas con baja capacidad).

Elegir una configuración personalizada mediante archivos de modelo, un modelo descriptivo personalizado para que sea sintetizado, mediante los botones se elige la ubicación de los archivos de configuración de modelo (donde se guarda la configuración de puentes personalizados a implementar para el modo de simulación ‘sim’), además del modelo URDF que se desea sintetizar para aparecer en la simulación.

Finalmente, se elige el controlador mediado por ROS que debe ser ubicado dentro de la carpeta autosail_ws/src previamente y construido con el comando mencionado anteriormente.

Posterior al inicio de la simulación una nueva ventana es desplegada para el control de las condiciones ambientales y el monitoreo de las variables percibidas por el bote durante la simulación(dirección del viento, dirección del bote, coordenadas)

El controlador

El controlador contemplado para el desarrollo del presente trabajo está direccionado con un enfoque minimalista que proporciona a nuevos usuarios una idea de cómo se debe construir un

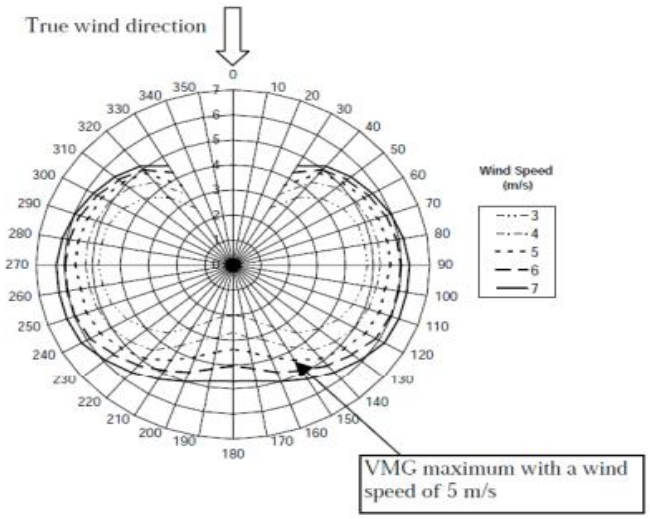


Figura 10. Diagrama polar de máxima velocidad desarrollable por un velero según la dirección del viento y su rumbo respecto a ella.[4]

nuevo controlador, las variables indispensables, y las funciones básicas que este debe desempeñar para generar una estrategia de navegación autónoma. Bajo la anterior premisa, el controlador desarrollado contempla la lectura de 3 sensores básicos:

1. GPS: para una navegación autónoma se debe conocer periódicamente la ubicación del velero por lo que este parámetro es básico para planear la ruta a seguir.

2. Magnetómetro: El anterior sensor informa acerca de la ubicación del velero, sin embargo, con este único dato (sumado a la falta de precisión) es imposible conocer el rumbo, para despejar dicha incógnita se hace necesario usar la magnitud de los campos magnéticos para conocer la dirección en la que se encamina el barco y así corregir la trayectoria.

3. Dirección del viento: El viento juega un papel principal ya que para el caso de un velero es la fuente de energía que este usará para su desplazamiento, por tanto, aprovechar dicha fuente al máximo también es crítico por parte del controlador para la realización de las rutas con mínimo esfuerzo.

Así las cosas, las dos variables a controlar por parte del sistema son claras el ángulo de ataque de la vela respecto al viento, y la trayectoria del velero. Ambas variables influidas entonces

por actuadores rotativos (servos motores), y cada una controlada por un algoritmo independiente integrado al programa del controlador:

El proceso para controlar el ángulo de la vela contempla un lazo cerrado simple en el que se evalúa un set point definido por la diferencia entre la dirección actual del viento y el rumbo del velero para fijar la vela en el ángulo más cercano al optimo más notorio de la gráfica polar de un velero.

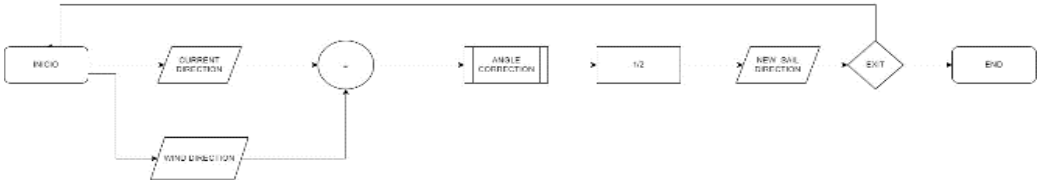


Figura 11. Gráfico de secuencia del controlador de la dirección

Mientras que para el rumbo se contempla un controlador PI cuyo set point es el ángulo de trayectoria deseado hacia el siguiente punto de la ruta y se evalúa mediante el sensor GPS y el magnetómetro para la toma del error actuante y ejercer control sobre el actuador del timón.

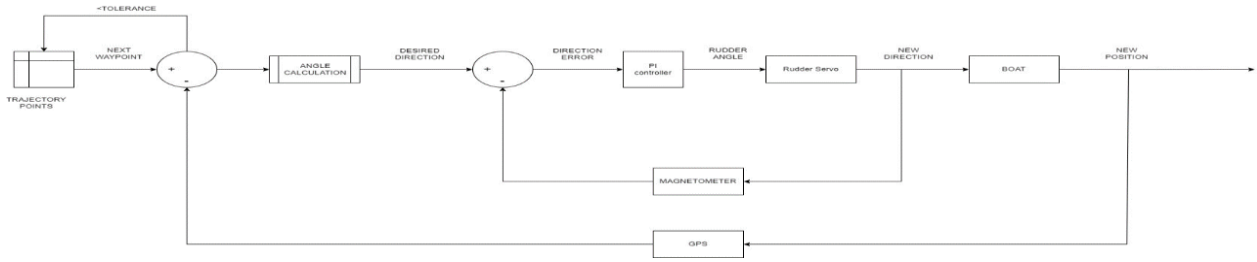


Figura 12. Diagrama de flujo de controlador de angulo de la vela

4)Desarrollo una estrategia de simulación de Hardware-in-the-loop soportada en ROS que permita el uso de MCU reales para la evaluación de estrategias de navegación.

a. Codificación de una interfaz mediada por MCU's.

Se desarrollo un puente serial compatible con los MCU's comerciales para la operación de los sensores y actuadores reales. Para esta etapa se creó un nuevo nodo que sintetizara la información aportada por los puentes desde la simulación que escribiese en los tópicos previamente definidos para los sensores físicos. Además, se incluyó un parámetro en la configuración para

desactivar la escritura por parte de estos sensores en dichos tópicos sin desactivar la escritura del controlador en los mismos.

b. Lanzamiento.

Dentro del lanzamiento se integró la secuencia de los distintos nodos para realizar la sinterización del ambiente general de simulación, se comprobó la estructura final del envío de mensajes obteniendo el siguiente grafo de los nodos y sus interfaces.

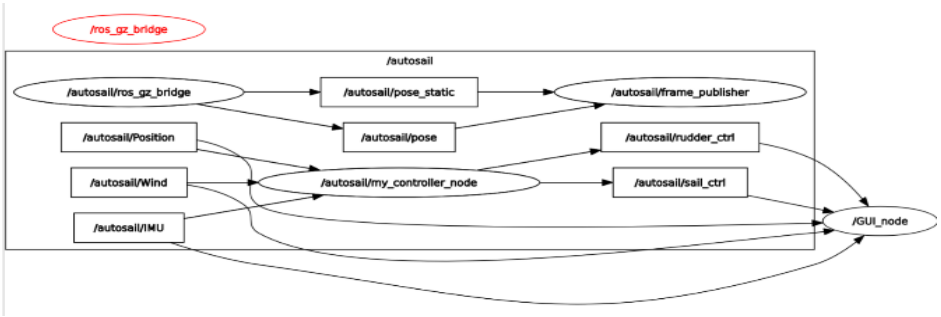


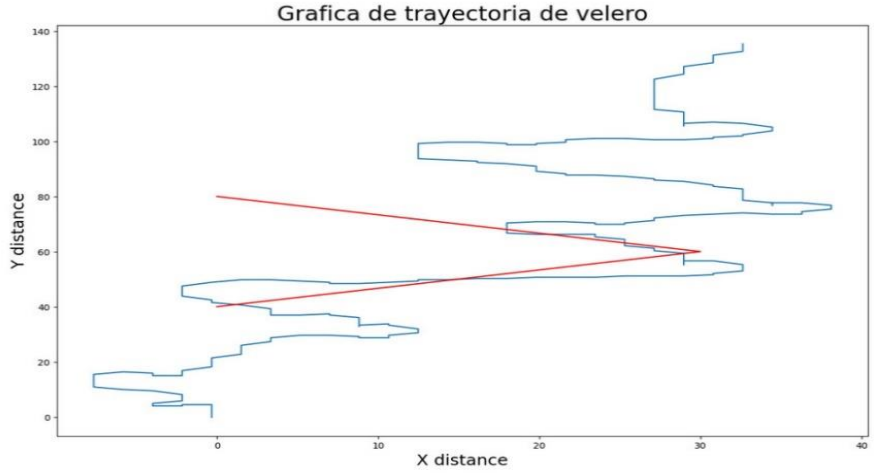
Figura 13. Esquema de comunicación entre las distintas instancias generado por rqt_graph

V. RESULTADOS

1. Se desarrollaron las partes, componentes y ensambles de un modelo virtual suficiente y acorde a la cinemática real del velero de referencia del grupo SISTEMIC, capaz de aparecer dentro del entorno de simulación, simular la navegación autónoma, interactuar con el entorno de prueba y otros entes similares dentro de la simulación, emitir y recibir información; sin que esto implicase un riesgo de inundación o hundimiento del modelo real, que además permite la observación por parte de los usuarios en tiempo real por medio de una interfaz gráfica.
2. Se desarrollo un entorno de pruebas simulado a partir de modelos preexistentes capaces de interactuar entre ellos y con los demás, e influir en sus sensores y actuadores para el entrenamiento de controladores.
3. Se desarrolló un controlador que será la base para la generación de nuevos y más complejos controladores de navegación autónoma mediada por el sistema de ROS.

4. Se desarrollaron interfaces múltiples para la recolección de datos desde diversas fuentes mediados por ROS y Gazebo.
5. Se crearon guías para el desarrollo de modelos, entornos, controladores e interfaces que favorecerán el crecimiento de este y otros proyectos de colaboración internacional.
6. Se creo la arquitectura de código necesaria para realizar el lanzamiento de todas las instancias de código (nodos, eventos, ventanas), la creación y traducción de modelos a los formatos requeridos, la ubicación de paquetes de código y en general la: compilación, construcción, instalación y el funcionamiento de todas las instancias de código empleadas para la ejecución del simulador.
7. Todo el desarrollo se documentó para ayudar a los nuevos investigadores a mejorar con su trabajo el proyecto de colaboración internacional con la Universidad tecnológica de panamá y la Universidad de Malärdalen, además se documentaron guías que ayudarán al mejor entendimiento de todo el sistema, la anexión de código.
8. Se hizo comprobación de la navegación del velero obtenido la siguiente gráfica

VII. ANÁLISIS



1. Se reconoce el potencial de crecimiento tanto para la interfaz, los controladores y los modelos, apuntando entre otras cosas a las limitaciones encontradas en cuanto a la capacidad de las plataformas de simulación de código abierto para interactuar

fuera de los límites de sus figuras geométricas básicas predefinidas para la modelación de comportamientos como la flotabilidad;

2. También se apunta a la necesidad de crear nuevos controladores especializados realizar operaciones de maximización y optimización para las variables como el viento cuyo comportamiento depende de múltiples factores durante la navegación.
3. Finalmente, se aplaude la capacidad que posibilitan las mismas plataformas de código para seguir creando herramientas cada vez más especializadas en proyectos de desarrollo que cumplan propósitos cada vez más específicos, mediante la codificación de nuevas instancias de código(plugins) compatibles.

VIII. CONCLUSIONES

1. La creación de nuevas plataformas de simulación implica el conocimiento completo del modelo que se desea simular, sus condiciones, interfaces, interacciones, entre otros, además de la investigación exhaustiva de las herramientas a utilizar durante la ejecución de las tareas para las cuales es creada.
2. La compatibilidad es código es fundamental a la hora de realizar trabajo colaborativo, por lo que es fundamentar acordar y compartir la información acerca de nuevos parámetros de desarrollo y/o herramientas a utilizar previo al desarrollo final.
3. La navegación completamente autónoma requiere de una gran carga de sensores y un controlador robusto que interprete de forma efectiva los cambios en el ambiente para aprovechar los recursos a su disposición de la mejor manera posible, es a mi pensar necesario que más investigadores se involucren en el desarrollo de estas tecnologías para la futura exploración de terrenos inhóspitos.
4. La digitalización se vuelve entonces una manera de proteger y optimizar los escasos recursos disponibles a la hora de comprobar el funcionamiento de desarrollos en distintas ramas como lo pueden ser: la robótica, la automatización y los sistemas embebidos, entre otros. Lo que plantea para las pequeñas empresas una alternativa viable para la comprobación de sistemas y algoritmos previo a su adquisición, y para las grandes empresas plantea una alternativa para ofertar, evaluar y observar sus

desarrollos sin incurrir en gastos de desplazamiento, instalación y/o puesta a punto de los equipos previo a su implementación; y solo conociendo con anterioridad a profundidad las interacciones a las que pueda verse sometido y probando las situaciones límite sin exposición alguna al mismo, lo que implica un ahorro significativo en el proceso de Investigación y desarrollo, y posiblemente una mayor satisfacción entre las partes.

5. El uso de plataformas como github, Docker, IEEE para la creación y publicación de nuevos modelos, paradigmas y plataformas de código abierto posibilita el desarrollo de más y mejores herramientas para la comunidad, la cooperación, la colaboración internacional, la revisión, el desarrollo de nuevos proyectos, la integración de desarrollos, las nuevas perspectivas y en general la integración con el estado del arte del desarrollo de proyectos de software.
6. El presente trabajo se desarrolló para mejorar el proceso de desarrollo para el proyecto Autosail, sin embargo, los aprendizajes implicados pueden ser llevados y aplicados en casi cualquier ámbito de desarrollo de sistemas embebidos.

REFERENCIAS

[1] Bingham, Brian et Al. “Proceedings of MTS/IEEE OCEANS Conference: Toward Maritime Robotic Simulation in Gazebo”. Seattle, WA: October, 2019. Disponible en: https://github.com/osrf/vrx

[2] Mihai, Stefan et Al. IEEE COMUNICATIONS SURVEYS & TUTORIALS, Vol. 24, No. 4 (2022); pg. 2255.

[3] Castanheira de Farias, A. Bernardes;Silva R.,Reurison; Murilo,André;Viela L., Renato & Avila, Suzana. Low-Cost Hardware-in-the-Loop Platform for Embedded Control Strategies Simulation. IEEE/ACCESS DOI 10.1109/ACCESS.2019.2934420(August,2019)

[4] Wilson, Ryan M.: “The Physics of Sailing”. JILA and Department of Physics, University of Colorado, Feb. 7 2010. Disponible en: http://grizzly.colorado.edu/~rmw/files/papers/PhysicsofSailing.pdf

[5] Open Robotics, “ROS 2: Humble [Software].” Open Robotics, 2023. Disponible en: https://docs.ros.org/en/humble/Installation.html

[6] Open Robotics, “Gazebo Simulator (Garden version 7.1) [Software].” Open Robotics, 2023. Disponible en: https://gazebosim.org/docs/garden/getstarted

[7] N. S. Giraldo Bustamante, “Sailboat navigation control system based on spiking neural networks” [tesis de maestría]. Medellin (Colombia), Universidad de Antioquia, 2022.

[8] Open Robotics. Why ROS?. https://www.ros.org/blog/why-ros/. Tomada el 15/03/2023.

[9] Open Robotics. Features and Benefits: showcase for inspiration. https://gazebosim.org/features . Tomada el 15/06/2023 Más información en https://gazebosim.org/api/sim/7/namespacegz_1_1sim_1_1systems.html.

[10] Open Robotics. Gazebo: Tutorial: Aerodynamics. https://classic.gazebosim.org/tutorials?tut=aerodynamics&cat=physics, 2014.

[11] Open Robotics. Gazebo Sim: Underwater vehicles. https://gazebosim.org/api/sim/7/underwater_vehicles.html. Tomada el: 15/06/2023

ANEXOS

ANEXO 1: instalación

Script de instalación de ROS y GAZEBO

```
#!/bin/bash
 ## Bash script for setting up ROS2 Humble for Ubuntu 20.04||22.04.
## It installs the common dependencies for all targets (including Qt Creator)
## Installs:
## - Common dependencies libraries and tools as defined in `ubuntu_sim_common_deps.sh`
## - ROS2 Humble (including Gazebo11)
 cd ~
 if [[ $(lsb_release -sc) == *"bionic"* ]]; then
  echo "OS version detected as $(lsb_release -sc) (18.04)."
  echo "ROS2 Humble requires at least Ubuntu 20.04."
  echo "Exiting ...."
  return 1;
fi
if echo ${LANG} == en_US.UTF-8; then
echo "UTF-8 supported";
else
echo "UTF-8 not supported installing necesary dependencies"
sudo apt update && sudo apt install locales
sudo locale-gen en_US en_US.UTF-8
sudo update-locale LC_ALL=en_US.UTF-8 LANG=en_US.UTF-8
export LANG=en_US.UTF-8
locale;
fi
  ##Agregar la apt Ros 2 al sistema.
 ##Primero asegurarse de que el repositorio de Ubuntu Universe está habilitado.
sudo apt install software-properties-common
```

```
sudo add-apt-repository universe
 ##Agregar la llave GPG con la apt a Ros2.
 sudo apt update && sudo apt install curl
sudo curl -sSL https://raw.githubusercontent.com/ros/rosdistro/master/ros.key -o /usr/share/keyrings/ros-archive-keyring.gpg
 ##Luego agregar el repositorio a la lista de recursos.
 echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/ros-archive-keyring.gpg] http://packages.ros.org/ros2/ubuntu $(. /etc/os-release && echo $UBUNTU_CODENAME) main" | sudo tee /etc/apt/sources.list.d/ros2.list > /dev/null
 ##Intalar paquetes comunes
 sudo apt update && sudo apt install -y \
  python3-flake8-docstrings \
  python3-pip \
  python3-pytest-cov \
  ros-dev-tools
  ##Install packages according to our Ubuntu version.
 if [[ $(lsb_release -sc) == *"focal"* ]]; then
python3 -m pip install -U \
flake8-blind-except \
flake8-builtins \
flake8-class-newline \
flake8-comprehensions \
flake8-deprecated \
flake8-import-order \
flake8-quotes \
pytest-repeat \
pytest-rerunfailures;
else
sudo apt install -y \
python3-flake8-blind-except \
python3-flake8-builtins \
```

```
python3-flake8-class-newline \
python3-flake8-comprehensions \
python3-flake8-deprecated \
python3-flake8-import-order \
python3-flake8-quotes \
python3-pytest-repeat \
python3-pytest-rerunfailures;
fi
 ## Preparar la carpeta fuente de humble
 mkdir -p ~/ros2_humble/src
cd ~/ros2_humble
vcs import --input https://raw.githubusercontent.com/ros2/ros2/humble/ros2.repos src
 sudo apt-get upgrade
 if $(printenv | grep -i ROS)=="";then
echo "No previuous ROS sourced...OK";
else
sudo apt-get purge --auto-remove ros-${ROS_DISTRO}-desktop-full
cd ~
cd ..
cd ..
sudo rm /etc/ros/rosdep/sources.list.d/20-default.list;
cd ~/ros2_humble;
fi
 sudo rosdep init
rosdep update
rosdep install --from-paths src --ignore-src -y --skip-keys "fastcdr rti-connext-dds-6.0.1 urdfdom_headers"
 cd ~/ros2_humble
colcon build --symlink-install
 . ~/ros2_humble/install/local_setup.bash
https://gazebosim.org/docs/garden/install_ubuntu_src
```

ANEXO 2: Instalación del paquete ros_gz

Para la codificación de puentes Open Robotics (Organización responsable del soporte de ROS y GAZEBO) provee una serie de paquetes que agrupan distintos tipos de integración entre ROS y GAZEBO. Para instalar estos recursos dentro de la máquina de Linux se debe seguir la siguiente secuencia de pasos (Previa instalación de ROS y GAZEBO):

1. Abrir una terminal de linux y situarse en la raíz con el comando:

cd ~

2. Establecer una variable de entorno con el nombre de la distribución de gazebo:

export GZ_VERSION=garden

3. Crear una carpeta de espacio de trabajo y guardar los archivos de fuente (o usar otro espacio de trabajo de ROS previamente creado) y copiar el codigo fuente:

mkdir -p ~/ws/src #Crear nuevo espacio de trabajo

cd ~/ws/src #Ir a la carpeta del espacio de trabajo

git clone https://github.com/gazebosim/ros_gz.git -b humble

4. Instalar dependencias

cd ~/ws

rosdep install -r --from-paths src -i -y --rosdistro humble

5. Construir el espacio de trabajo no olvidando poner como fuente la ubicación de la carpeta fuente de nuestra distribución de ROS (en este caso ‘humble’).

Archivo de configuración fuente la distribución de ROS

source /opt/ros/<distro>/setup.bash

Construir e instalar el espacio de trabajo

cd ~/ws

colcon build

Uso de ros_gz_bridge

Para el uso de los puentes se debe tener en cuenta la siguiente configuración, debido a que ros_gz son un conjunto de paquetes al igual que cualquier otro paquete se debe invocar mediante el comando ros2 run [paquete] [script], su función consiste en hacer una copia de los temas redactados en gazebo dentro del sistema de ROS, sin embargo, para que este configure el puente se debe tener en cuenta que los argumentos deben tener la siguiente estructura:

[gazebo_topic]@[ros_topic_type]{BRG_DIR}[gazebo_topic_type]

La dirección se determina mediante los caracteres ‘]’: Gazebo a ROS, ’[‘: ROS a GAZEBO y ‘@’ Bidireccional, es decir la dirección define si es publicador, escuchante o ambos roles simultáneamente del tema.

Para realizar el mapeo se debe cumplir que exista la combinación entre tipos de temas dentro de la siguiente tabla

ROS type	Gazebo type
builtin_interfaces/msg/Time	ignition::msgs::Time
std_msgs/msg/Bool	ignition::msgs::Boolean
std_msgs/msg/ColorRGBA	ignition::msgs::Color
std_msgs/msg/Empty	ignition::msgs::Empty
std_msgs/msg/Float32	ignition::msgs::Float
std_msgs/msg/Float64	ignition::msgs::Double
std_msgs/msg/Header	ignition::msgs::Header
std_msgs/msg/Int32	ignition::msgs::Int32
std_msgs/msg/UInt32	ignition::msgs::UInt32
std_msgs/msg/String	ignition::msgs::StringMsg
geometry_msgs/msg/Wrench	ignition::msgs::Wrench
geometry_msgs/msg/WrenchStamped	ignition::msgs::Wrench
geometry_msgs/msg/Quaternion	ignition::msgs::Quaternion
geometry_msgs/msg/Vector3	ignition::msgs::Vector3d
geometry_msgs/msg/Point	ignition::msgs::Vector3d
geometry_msgs/msg/Pose	ignition::msgs::Pose
geometry_msgs/msg/PoseArray	ignition::msgs::Pose_V
geometry_msgs/msg/PoseWithCovariance	ignition::msgs::PoseWithCovariance
geometry_msgs/msg/PoseStamped	ignition::msgs::Pose
geometry_msgs/msg/Transform	ignition::msgs::Pose
geometry_msgs/msg/TransformStamped	ignition::msgs::Pose
geometry_msgs/msg/Twist	ignition::msgs::Twist
geometry_msgs/msg/TwistWithCovariance	ignition::msgs::TwistWithCovariance
nav_msgs/msg/Odometry	ignition::msgs::Odometry
nav_msgs/msg/Odometry	ignition::msgs::OdometryWithCovariance
rcl_interfaces/msg/ParameterValue	ignition::msgs::Any

ros_gz_interfaces/msg/Altimeter	ignition::msgs::Altimeter
ros_gz_interfaces/msg/Contact	ignition::msgs::Contact
ros_gz_interfaces/msg/Contacts	ignition::msgs::Contacts
ros_gz_interfaces/msg/Dataframe	ignition::msgs::Dataframe
ros_gz_interfaces/msg/Entity	ignition::msgs::Entity
ros_gz_interfaces/msg/Float32Array	ignition::msgs::Float_V
ros_gz_interfaces/msg/GuiCamera	ignition::msgs::GUICamera
ros_gz_interfaces/msg/JointWrench	ignition::msgs::JointWrench
ros_gz_interfaces/msg/Light	ignition::msgs::Light
ros_gz_interfaces/msg/SensorNoise	ignition::msgs::SensorNoise
ros_gz_interfaces/msg/StringVec	ignition::msgs::StringMsg_V
ros_gz_interfaces/msg/TrackVisual	ignition::msgs::TrackVisual
ros_gz_interfaces/msg/VideoRecord	ignition::msgs::VideoRecord
ros_gz_interfaces/msg/WorldControl	ignition::msgs::WorldControl
rosgraph_msgs/msg/Clock	ignition::msgs::Clock
sensor_msgs/msg/BatteryState	ignition::msgs::BatteryState
sensor_msgs/msg/CameraInfo	ignition::msgs::CameraInfo
sensor_msgs/msg/FluidPressure	ignition::msgs::FluidPressure
sensor_msgs/msg/Imu	ignition::msgs::IMU
sensor_msgs/msg/Image	ignition::msgs::Image
sensor_msgs/msg/JointState	ignition::msgs::Model
sensor_msgs/msg/Joy	ignition::msgs::Joy
sensor_msgs/msg/LaserScan	ignition::msgs::LaserScan
sensor_msgs/msg/MagneticField	ignition::msgs::Magnetometer
sensor_msgs/msg/NavSatFix	ignition::msgs::NavSat
sensor_msgs/msg/PointCloud2	ignition::msgs::PointCloudPacked
tf2_msgs/msg/TFMessage	ignition::msgs::Pose_V
trajectory_msgs/msg/JointTrajectory	ignition::msgs::JointTrajectory

NOTA: El namespace ‘ignition’ está siendo derogado por lo que es recomendable buscar y realizar los cambios para usar el namespace ‘gz’ en su lugar.

Uso de ros_gz_sim: Herramienta para la creación de entidades nuevas en la herramienta de simulación

Una tercera herramienta es el paquete de creación de entidades nuevas dentro de la simulación, que permite realizar la creación de una nueva entidad(modelo) en tiempo de ejecución simplemente utilizando como argumento la pose que va a adoptar el elemento y el stream SDF de la entidad que se desea incluir dentro de la simulación. Esta herramienta se convoca mediante el comando

ros2 run ros_gz_sim create –stream [sdf_stream] -name [nombre] -allow_renaming [true/false] -x [x] -y [y] -z [z] -R [roll] -P [pitch] -Y [yaw]