SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLOGICO NACIONAL DE MEXICO
Instituto Tecnológico de La Paz

INSTITUTO TECNOLÓGICO DE LA PAZ

DESARROLLO DE UN SOFTWARE PARA CÁLCULO DE RESISTENCIA DE EMBARCACIONES

TITULACIÓN INTEGRAL (INFORME TÉCNICO DE RESIDENCIA PROFESIONAL)

que para obtener el título de

INGENIERO EN SISTEMAS COMPUTACIONALES

Presenta

JORGE ALBERTO ARMENTA ATAMOROS

La Paz, Baja California Sur, Abril 2016

SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO
Instituto Tecnológico de La Paz

Departamento: DIV.EST.PROF.
No. de Oficio: DEP-T-355-2016
Asunto: Se autoriza impresión.

La Paz, B.C.S., 18 de marzo del 2016.

CJORGE ALBERTO ARMENTA ATAMOROS,
PASANTES DE LA CARRERA DE INGENIERIA EN SISTEMAS COMPUTACIONALES.
P R E S E N T E.

Con base en el dictamen de aprobación emitido por la Comisión Revisora del trabajo denominado **"DESARROLLO DE UN SOFTWARE PARA CALCULO DE RESISTENCIA DE EMBARCACIONES".** Entregado por usted para su análisis, le informo que se **AUTORIZA** la impresión.

A T E N T A M E N T E
"Ciencia es Verdad, Técnica es Libertad"

S.E.P. D.G.E.S.T. S.E.S.
INSTITUTO TECNOLÓGICO DE LA PAZ
DIVISIÓN DE ESTUDIOS PROFESIONALES

L.C. EDGAR GUADALUPE CASTRO MIRANDA,
JEFE DE LA DIVISIÓN DE ESTUDIOS PROFESIONALES.

EGCM/nda

SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO
Instituto Tecnológico de La Paz

MEMORANDUM

PARA: MSC. MARTIN AGUNDEZ AMADOR,
JEFE DEL DEPTO. DE SISTEMAS Y COMPUTACIÓN.

DE: COMISIÓN REVISORA DE TITULACIÓN.

FECHA: 17 de Marzo del 2016.

ASUNTO: Autorización de Impresión.

Por medio del presente le informamos que una vez revisado el trabajo denominado **"DESARROLLO DE UN SOFTWARE PARA CALCULO DE RESISTENCIA DE EMBARCACIONES"**, como opción de Titulación Integral (Informe Técnico de Residencia Profesional), que presenta el pasante **Jorge Alberto Armenta Atamoros**, de la carrera de **INGENIERIA EN SISTEMAS COMPUTACIONALES**, le informamos que esta comisión revisora ha dictaminado que el trabajo esta:

ACEPTADO (X) SUJETO A MODIFICACIONES ( ) RECHAZADO ( )

Por lo cual solicitamos se continúe con el trámite de sustentación del examen profesional.

A T E N T A M E N T E

ING. ALBERTO IBARRA BERBER

MSC. MARTIN AGUNDEZ AMADOR

MC. JORGE ENRIQUE LUNA TAYLOR

c.c.p. División de Estudios Profesionales.
c.c.p. Archivo.
vmhr*

RESUMEN

En el presente documento se reportan las actividades realizadas durante el desarrollo del proyecto de residencias denominado “Desarrollo de un software para cálculo de resistencia de embarcaciones”, así como los resultados obtenidos durante las diversas etapas de análisis, diseño y funcionamiento del proyecto.

La problemática atendida es que el actual diseño del casco de una embarcación requiere de la elaboración de un modelo a escala, como también, de la realización de pruebas de arrastre en un canal que cuente con las instalaciones apropiadas para obtener una medición precisa de la fuerza de resistencia.

Anteriormente se desarrolló un software para calcular las fuerzas de resistencia y fricción de una embarcación, basado en el método propuesto por el Ing. Wolff, desarrollado en un trabajo de investigación, y el cual se utiliza en el presente proyecto. Este software era muy limitado tanto en funcionalidad y realismo, ya que no modelaba embarcaciones curvas, solamente figuras prismáticas; se acercaba mediante cortes, donde se debía modificar cada uno de ellos para poder darle un aspecto curvo, esto hacía el proceso de diseño muy complicado, lento y tedioso.

Como resultado de este proyecto se obtuvo un software para graficar modelos tridimensionales con trazos curvos de una manera amigable, para poder aplicar así los cálculos de resistencia y fricción, con los cálculos propuestos por el Ing. Wolff.

Índice

1. Informe final de Residencia Profesional

1.1. Introducción

El diseño del casco de un barco requiere, actualmente, de la elaboración de un modelo a escala y pruebas de arrastre en un canal que cuente con las instalaciones correspondientes para una medición precisa de la fuerza de resistencia sobre éste. Además, requiere de un proceso de cálculo con un software especial, que tarda hasta media hora para arrojar datos sobre la potencia que requiere el prototipo para moverse a una determinada velocidad. En este proceso de cálculo se utiliza la teoría del flujo potencial, es decir se basa en un patrón de olas generado por el modelo arrastrado en el canal. La fricción entre casco y agua, tanto como la fuerza de inercia que genera el agua desplazada, no se toman en cuenta directamente.

En el presente proyecto se utiliza un método de cálculo basado en la fuerza de inercia, desarrollado por el Ing. Eberhard Wolff Krautter, combinado con valores de fricción que se conocen para determinadas condiciones de flujo; de manera que los modelos de embarcaciones, que se elaboren mediante el programa diseñado en éste proyecto, se transfieran al programa de cálculo y se obtenga directamente la información sobre la potencia que requiere el casco dibujado para moverse a cierta velocidad sobre el agua.

El proyecto incluye el desarrollo de un software que permite diseñar gráficamente la forma tridimensional del casco de una embarcación, y a través de la integración de cortes horizontales del diseño, realice el cálculo de la resistencia y fricción.

1.2. Datos generales de la empresa

1.2.1. Nombre de la empresa y/o razón social.

Instituto Tecnológico de La Paz.

1.2.2. Domicilio de la empresa

Boulevard Forjadores de Baja California Sur No.4720, apdo. postal 43-B, C.P 23080
La Paz, B. C. S., México.

Tels. (612) 12-140-24, 12-104-26, 12-107-05
Fax. (612) 12-112-95.

1.2.3. Giro

Educativo.

SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO
Instituto Tecnológico de La Paz

ORGANIGRAMA

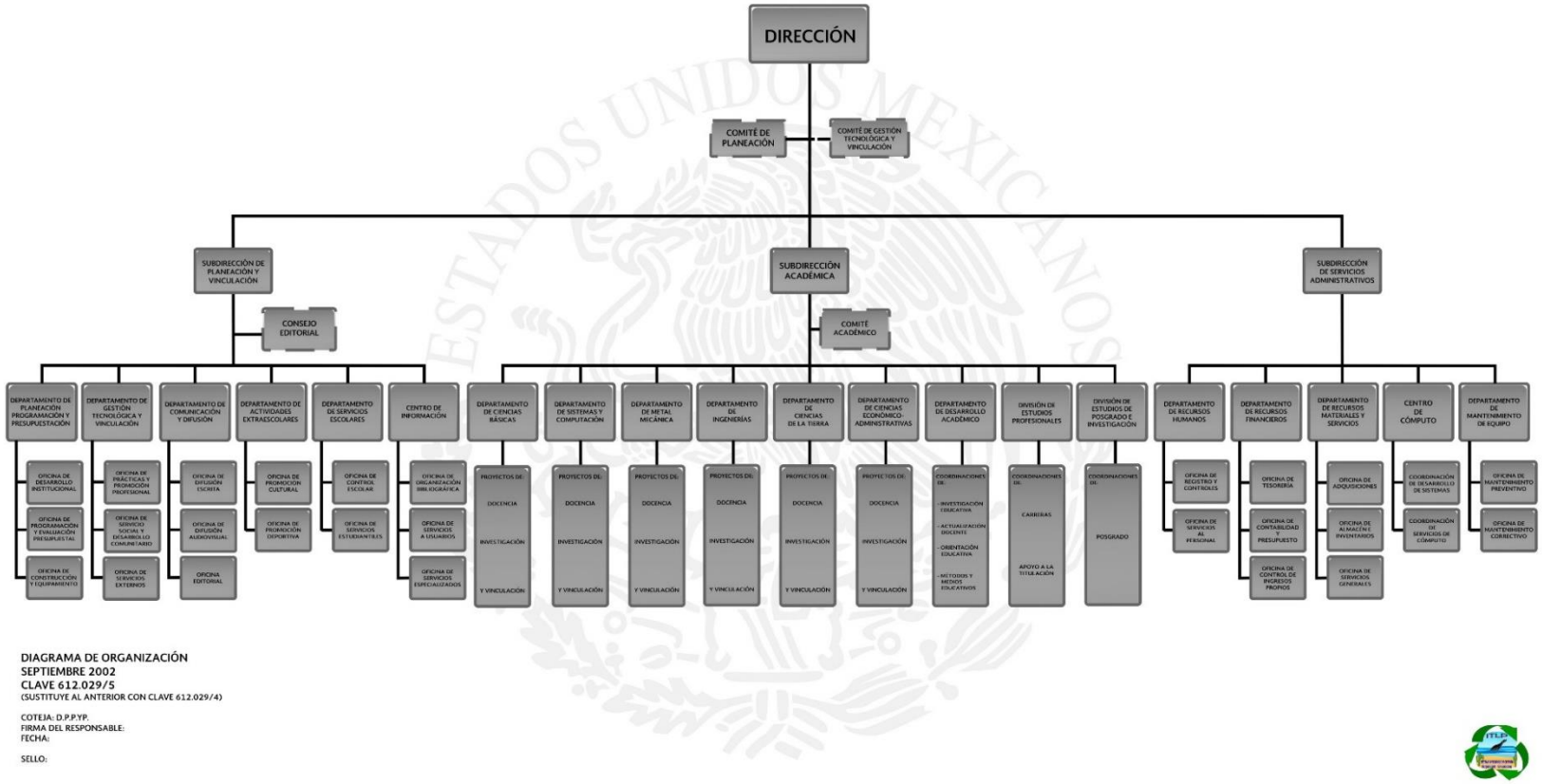


DIAGRAMA DE ORGANIZACIÓN
SEPTIEMBRE 2002
CLAVE 612.029/5
(SUSTITUYE AL ANTERIOR CON CLAVE 612.029/4)

COTEJA: D.P.P.YP.
FIRMA DEL RESPONSABLE:
FECHA:

SELLO:

Figura 1. Organigrama de la empresa.

1.2.5. Descripción de la empresa.

El Instituto Tecnológico de La Paz, es una institución pública de educación superior localizada en La Paz, Baja California Sur. Es considerada como la institución educativa de mayor nivel académico en todo el estado. El Instituto Tecnológico de La Paz imparte 9 carreras a nivel licenciatura y 2 a nivel posgrado en las áreas de ciencias sociales y administrativas, e ingeniería. Forma parte de la Dirección General de Educación Superior Tecnológica (DGEST), de la Secretaría de Educación Pública de México.

Ésta es una institución educativa que cumple con la sociedad sudcaliforniana entregando nuevos profesionistas que se incorporan a las actividades productivas del estado.

1.2.5.1. Visión del I. T. L. P.

Ser un instituto reconocido a nivel nacional e internacional por su calidad y pertinencia, sustento en el desarrollo del ser humano y su interacción con el medio ambiente.

1.2.5.2. Misión del I. T. L. P.

Contribuir al desarrollo sustentable, científico y tecnológico del país, a través de la formación integral de profesionistas competentes y comprometidos con una sociedad más humana.

1.3. Justificación del proyecto

Hasta la fecha no existe en el mercado internacional un programa computacional, que permita dibujar un casco de una embarcación de cualquier tipo y tamaño, y obtenga directamente los cálculos, de acuerdo al método desarrollado por el Ingeniero Eberhard Wolff Krautter, de fuerza de resistencia y fricción de las que la embarcación requiere para moverse en el agua con determinada velocidad.

1.4. Objetivos generales y específicos

1.4.1. Objetivo general

Desarrollar un software que permita modelar gráficamente la forma tridimensional curvas de diversos cascos de embarcaciones, y a través de la integración de cortes horizontales se realice el cálculo de fricción y resistencia.

1.4.2. Objetivos específicos

- Diseñar y elaborar un programa que permita dibujar un cuerpo tridimensional con las mayores facilidades de manejo posibles.
- Generar modelos de cuña prismática, para realizar los cálculos de fuerza de desplazamiento en el fluido.
- Generar modelos de cuña curva, aproximando la curva por medio de rectas para calcular la fuerza de fricción y resistencia.
- Implementar el método de cálculo desarrollado por el Ing. Eberhard Wolff Krautter, para la obtención del cálculo de la fuerza de fricción y resistencia de los diseños generados.

1.5. Características del área en que participó

El proyecto se desarrolló en el campo de la investigación e ingeniería, involucrando principalmente dos ramas de la misma, Ingeniería Industrial e Ingeniería de Software.

La Ingeniería Industrial es la rama de la ingeniería que se ocupa del desarrollo, mejora implantación y evaluación de sistemas integrados de gente, dinero, conocimientos, información, equipamiento, energía, materiales y procesos. También trata con el diseño de nuevos prototipos para ahorrar dinero y hacerlos mejores.

La Ingeniería Industrial está construida sobre los principios y métodos del análisis y síntesis de la ingeniería y el diseño para especificar, predecir y evaluar los resultados obtenidos de tales sistemas. En la manufactura, los ingenieros industriales trabajan para eliminar desperdicios, así, logrando la optimización de todos los recursos.

La Ingeniería de Software es la disciplina o área de la informática que ofrece métodos y técnicas para desarrollar y mantener software de calidad. Esta ingeniería trata con áreas muy diversas de la informática y de las ciencias de la computación, tales como la construcción de compiladores, sistemas operativos, o desarrollos intranet/internet.

La ingeniería de Software aborda todas las fases del ciclo de vida del desarrollo de cualquier tipo de sistemas de información a diversas áreas: negocios, investigación científica, medicina, producción logística, banca, control de tráfico, meteorología, derecho, internet, etcétera.

1.6. Alcances y limitaciones.

1.6.1. Alcances

- Generación de modelos para cascos de embarcaciones en forma curva y prismática.
- Cálculo automático de la fuerza de fricción y fuerza de resistencia del casco en el agua.
- Generación de modelo con varias caras del casco.
- Modelado de la embarcación en 3D.
- Variación de los parámetros de experimentación.
- Guardar y recuperar modelos.

1.6.2. Limitaciones.

El software se limita a sólo hacer diseños de la parte sumergida de la embarcación, por lo tanto, no se pueden hacer diseños completos de embarcaciones.

1.7. Marco teórico

1.7.1. Lenguaje unificado de modelado

El lenguaje unificado de modelado (Unified Modeling Language, UML) es un lenguaje estándar para escribir planos de software. UML puede utilizarse para visualizar, especificar, construir y documentar los artefactos de un sistema que involucra una gran cantidad de software.

El UML es apropiado para modelar desde sistemas de información en empresas hasta aplicaciones distribuidas basadas en la Web e incluso para sistemas empotrados de tiempo real muy exigentes. Es un lenguaje muy expresivo, que cubre todas las vistas necesarias para desarrollar y luego desplegar tales sistemas. Por ser expresivo, el UML no es difícil de aprender, ni de utilizar. Aprender a aplicar el UML, de modo eficaz, comienza por crear un modelo conceptual del lenguaje, de lo cual se requiere aprender tres elementos principales: los bloques básicos de construcción del UML, las reglas que dictan cómo pueden combinarse dichos bloques y algunos mecanismos comunes que se aplican en el lenguaje.

El UML es sólo un lenguaje y, por lo tanto, es parte de un método de desarrollo de software. El UML es independiente del proceso, aunque para utilizarlo óptimamente se debería usar en un proceso que fuese dirigido por los casos de uso, centrado en la arquitectura, iterativa e incremental (Boosh,1999).

1.7.1.1. Diagrama en UML

Un diagrama es la representación gráfica de un conjunto de elementos, visualizado la mayoría de las veces como un grafo conexo de nodos (elementos) y arcos (relaciones).

Los diagramas se dibujan para visualizar un sistema desde diferentes perspectivas, de forma que un diagrama es una proyección de un sistema. Para todos los sistemas, excepto los más triviales, un diagrama representa una vista resumida de los elementos que constituyen un sistema. El mismo elemento puede aparecer en todos los

diagramas, sólo en unos pocos diagramas (el caso más común), o en ningún diagrama (un caso muy raro).

En la teoría, un diagrama puede contener cualquier combinación de elementos y relaciones. En la práctica, sin embargo, sólo surge un pequeño número de combinaciones, las cuales son consistentes con las cinco vistas más útiles que comprenden la arquitectura de un sistema con gran cantidad de software. Por esta razón, el UML incluye nueve de estos diagramas:

1. Diagrama de Clases.
2. Diagrama de Objetos.
3. Diagrama de Casos de Uso.
4. Diagrama de Secuencia.
5. Diagrama de Colaboración.
6. Diagrama de Estados.
7. Diagrama de Actividades.
8. Diagrama de Componentes.
9. Diagrama de Despleguе.

(Boosh,1990).

1.7.2. Lenguaje C++

C++ es un lenguaje de programación diseñado a mediados de los años 80's, por Bjarne Stroustrup. La intención de su creación fue el extender al lenguaje de programación "*C"* con mecanismos que permitan la manipulación de objetos. En ese sentido, desde el punto de vista de los lenguajes orientados a objetos, el *C*++ es un lenguaje híbrido y más popular de todos.

1.7.3. XML

XML es un derivado del SGML. Intenta recoger lo mejor de HTML y de SGML. Los creadores de XML han pretendido recoger tan solo los elementos necesarios para un uso en internet. Una vez realizados los recortes solo han quedado 35 páginas de

especificaciones frente a las 155 del SGML, siendo el número de páginas directamente proporcional a la complejidad del lenguaje.

XML fue desarrollado inicialmente para internet, pero gracias a las distintas DTD (Definición de tipo de documento) a las que puede asociarse, puede servir en la actualidad para distintas aplicaciones como formato de intercambio o como sistema de almacenamiento de datos.

El hecho es que es un lenguaje más simple de escritura que su predecesor el HTML. Es ciertamente más fácil crear los marcadores cuando éstos se necesitan; en HTML es necesario conocer los marcadores de que disponemos, así como saber a qué corresponden. Esto tiene el objetivo de hacer comprensible el texto del documento XML por todo el mundo.

XML, al igual que SGML y HTML, es lo que se llama un lenguaje de marcas. El marcado es un procedimiento que se remonta a las artes gráficas, en donde se anotaban textos o párrafos completos con el fin de aportar informaciones complementarias a las personas que trabajaban en las correcciones de los documentos. En el caso de los documentos electrónicos, utilizamos un marcado electrónico que permite a las aplicaciones tratar los datos que reciben. El objeto de este marcado es, a diferencia de las artes gráficas, permitir el formato del documento. Uno de los tipos de formato más conocidos para el tratamiento de texto es el RTF (Rich Text Format).

Estructura de un documento XML

Un documento XML está formado por elementos, marcadores, varios bloques, comentarios y atributos, en la *Figura 2* se ejemplifica la estructura de un documento XML.

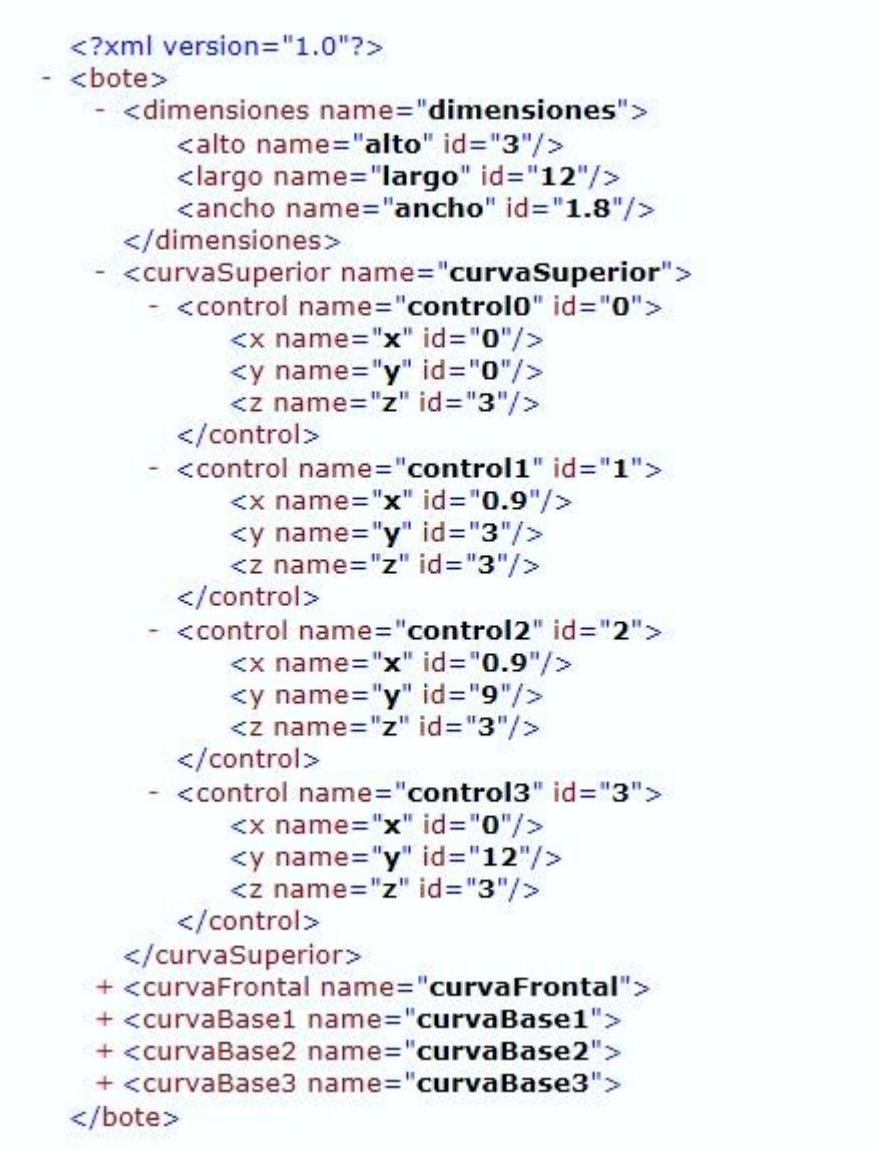


Figura 2. Ejemplificación la estructura de un documento XML.

Elemento raíz

Un documento XML contiene un único elemento raíz (Document Element). Este elemento engloba todos los anteriores, sean del tipo que fueren: texto, elementos, atributos o entidades. En nuestro ejemplo anterior *(Figura 2)*. El elemento raíz es:

```
<bote> … </bote>
```

Elementos

Los elementos son los principales componentes del contenido del documento. Pueden contener texto u otros elementos. Cuando un elemento contiene otros elementos, estos son llamados elementos hijos. En la *Figura 2*, los elementos son, “dimensiones” como padre y “alto”, “largo”, “ancho” como hijos

```
<dimensiones name="dimensiones">
    <alto name="alto" id="0.5"/>
    <largo name="largo" id="2"/>
    <ancho name="ancho" id="0.688896"/>
</dimensiones>
```

Atributos

Los elementos pueden contener uno o varios atributos. Cada elemento solo puede contener un mismo atributo una única vez. De lo contrario, el documento se denomina “no bien formado”. En el ejemplo de la *Figura 2* se tiene el siguiente atributo:

```
<alto name="alto" id="0.5"/>
```

1.7.4. *Qt*

Qt es un marco de desarrollo de aplicaciones multiplataforma para escritorio, incrustado y móvil. Plataformas compatibles incluyen Linux, OS X, Windows, VxWorks, QNX, Android , iOS , BlackBerry , Sailfish OS y otros.

Qt no es un lenguaje de programación por su propia cuenta. Es un marco escrito en *C*++. Un preprocesador, el MOC (compilador Meta- Object), se utiliza para extender el lenguaje *C*++ con funciones como señales y slots. Antes de la etapa de compilación, el MOC analiza los archivos de código fuente escrito en extendido *Qt* y *C*++, genera fuentes estándar compatible con C ++ de ellos. Así, el propio marco y aplicaciones/librerías que utilizan puede ser compilado por cualquier compilador de *C*++ compatible estándar como Clang, GCC, ICC, MinGW y MSVC.

1.7.4.1. Licencia.

Qt está disponible bajo varias licencias, El *Qt* Compañía vende licencias comerciales, pero *Qt* también está disponible como software libre bajo varias versiones de la GPL y la LGPL.

1.7.4.2. Sistema de construcción (Build System).

Aunque cualquier sistema de construcción se puede utilizar con *Qt*, *Qt* trae su propio *qmake*. es un frontend multiplataforma para sistemas de construcción de plataforma nativa, como *GNU Make*, *Visual Studio* y *Xcode* .

CMake es también una alternativa popular para construir proyectos de *Qt*, se ha integrado hace años soporte para *Qt* 4 y 5.

1.7.4.3. Entorno de desarrollo integrado (IDE).

Qt viene con su propio entorno de desarrollo integrado (IDE), llamado *Qt Creator*. Se ejecuta en *Linux* , *OS X* y *Windows* y ofrece terminación inteligente de código, resaltado de sintaxis, un sistema de ayuda integrado, depurador y la integración de perfiles y también de integración para todos los principales sistemas de control de versiones (por ejemplo, git , Bazar).

Por estas razones se eligió, como entorno de desarrollo, utilizar *Qt*, al ser de código abierto y tener un entorno muy amigable, como Visual Studio, hace que el trabajo se facilite de mejor manera.

1.7.5. *OpenGL*

OpenGL es el principal entorno de desarrollo portátil, con aplicaciones graficas interactivas en 2D y 3D. Desde su introducción, en 1992, *OpenGL* se ha convertido en la interfaz de programación de aplicaciones (API) en gráficos 2D y 3D más utilizada en las industrias, trayendo miles de aplicaciones a una amplia variedad de plataformas informáticas.

OpenGL fomenta la innovación y acelera el desarrollo de aplicaciones mediante la incorporación de un amplio conjunto de renderizado, mapeado de textura, efectos especiales, y otras funciones de visualización de gran alcance. Los desarrolladores pueden aprovechar la potencia de *OpenGL* a través de todas las plataformas de escritorio y estaciones de trabajo populares asegurando la implementación de aplicaciones.

1.7.5.1. Funcionamiento.

El funcionamiento básico de *OpenGL* consiste en aceptar primitivas tales como puntos, líneas y polígonos, y convertirlas en píxeles. Este proceso es realizado por una *pipeline* gráfica conocida como *Máquina de estados de OpenGL*. La mayor parte de los comandos de *OpenGL* emiten primitivas a la pipeline gráfica o bien configuran cómo la pipeline procesa dichas primitivas. Hasta la aparición de la versión 2.0, cada etapa de la pipeline ejecutaba una función prefijada, resultando poco configurable. A partir de la versión 2.0, algunas etapas son programables usando un lenguaje de programación llamado *GLSL(Lenguaje de sombreado de OpenGL)*.

OpenGL es una API basada en procedimientos de bajo nivel que requiere que el programador dicte los pasos exactos necesarios para renderizar una escena. Esto contrasta con las API's descriptivas, donde un programador sólo debe describir la escena y puede dejar que la biblioteca controle los detalles para representarla. El diseño de bajo nivel de *OpenGL* requiere que los programadores conozcan en profundidad la pipeline gráfica, a cambio de darles libertad para implementar algoritmos gráficos novedosos.

OpenGL tiene dos propósitos esenciales:

- Ocultar la complejidad de la interfaz con las diferentes tarjetas gráficas, presentando al programador una API única y uniforme.
- Ocultar las diferentes capacidades de las diversas plataformas hardware, requiriendo que todas las implementaciones soporten la funcionalidad completa de OpenGL (utilizando emulación software si fuese necesario).

1.7.5.2. Proceso de dibujo de OpenGL.

En la *Figura 3* se puede apreciar el orden de operaciones que se sigue para el dibujado en OpenGL. Por un lado, tenemos el “Vertex data”, que describe los objetos de nuestra escena, y por otro, el “pixel data”, que describe las propiedades de la escena que se aplican sobre la imagen tal y como se representa en el buffer. Ambas se pueden guardar en una “displaylist”, que es un conjunto de operaciones que se guardan para ser ejecutadas en cualquier momento (Shereiner,2009).

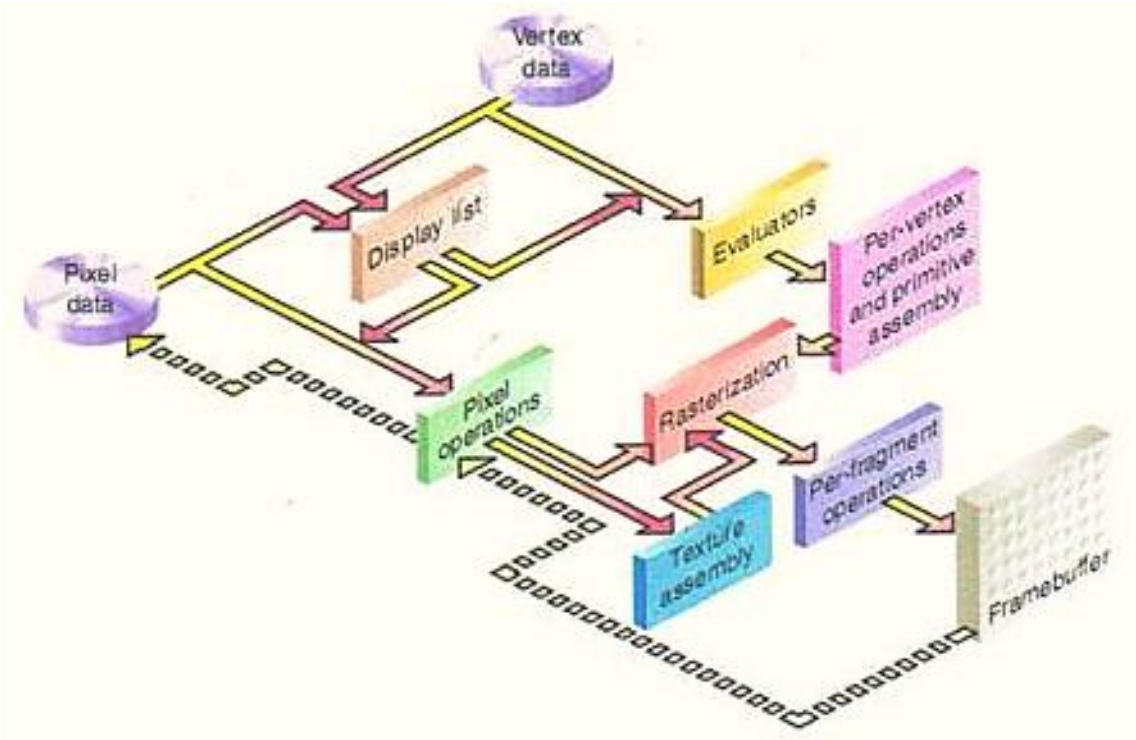


Figura 3. Orden de operaciones de OpenGL.

1.7.5.3. Licencia.

OpenGL es de código abierto y su código fuente está disponible para los proveedores de plataformas de hardware con licencia. Los usuarios finales, proveedores de software independientes, y otros que escriben código basado en la API de *OpenGL* están libres de los requisitos de licencia.

1.7.5.4. Funciones de OpenGL utilizadas.

Las funciones utilizadas son:

- gluPerspective(…),
- gluLookAt (...);
- glViewport (…);
- glBegin (…) glEnd;
- glClearColor
- glClear
- glMatrixMode (…);
- glLoadIdentity

1.7.5.4.1. Perspectiva.

La perspectiva se refiere a la proyección de los objetos que dan la ilusión de verlos en tres dimensiones, en *OpenGL* se utiliza la función *gluPerspective* que especifica un tronco de visualización en el sistema mundial de coordenadas. En general, la relación de aspecto en *gluPerspective* debe coincidir con la relación de aspecto de la ventana gráfica asociada. Por ejemplo, $aspect = 2{,}0$ significa que el ángulo del espectador de vista es el doble de ancho en el eje X en relación al eje Y. Si la ventana es el doble de ancho como alto, se muestra la imagen sin distorsión, ver *Figura 4*.

Función

```
Void gluPerspective(double fovy, double aspect, double Near, double Far);
```

Parámetros.

Los parámetros *fovy, aspect, Near y Far* se visualizan en la *Figura 4*.

- *fovy:* Especifica el campo de visión de ángulo, en grados, en la dirección del eje y.
- *aspect:* Especifica la relación de aspecto que determina el campo de visión en la dirección x. La relación de aspecto es la relación de w (anchura) a h (altura).
- *Near:* Específica la distancia del espectador al plano de delimitación cercano (siempre positivo).

- *Far:* Específica la distancia del espectador al plano de delimitación lejano (siempre positivo).

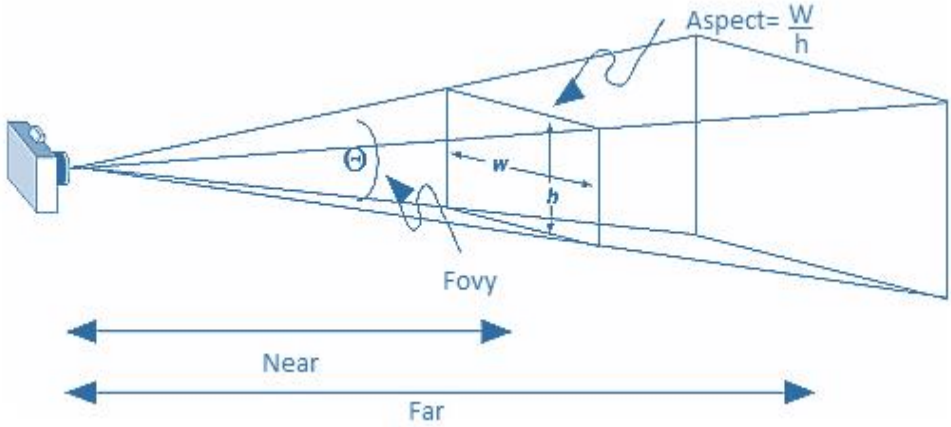


Figura 4. Parámetros (fovy, aspect, Near, Far)

1.7.5.4.2. Cámara de visualización.

Como observador debemos estar en una posición determinada dentro de la escena, con la visualización dirigida, también, a cierta dirección, en una respectiva inclinación. Esto en *OpenGL* se logra con la función ***gluLookAt***, esta crea una matriz de visualización derivado de un punto de posición a un punto de referencia que indica el centro de la escena, y un vector de inclinación llamado "up", ver *Figura 5*.

Función:

```
Void gluLookAt (double eyeX, double eyeY, double eyeZ, double centerX, double centerY, double centerZ, double upX, double upY, double upZ,)
```

Parámetros:

- *eyeX*, *eyeY*, *eyeZ:* Especifica la posición del punto del observador.
- *centerX*, *centerY*, *centerZ:* Especifica la posición del punto de referencia (hacía donde mira el observador).
- *upX*, *upY*, *upZ:* Especifica la dirección el Vector Up (inclinación del observador)

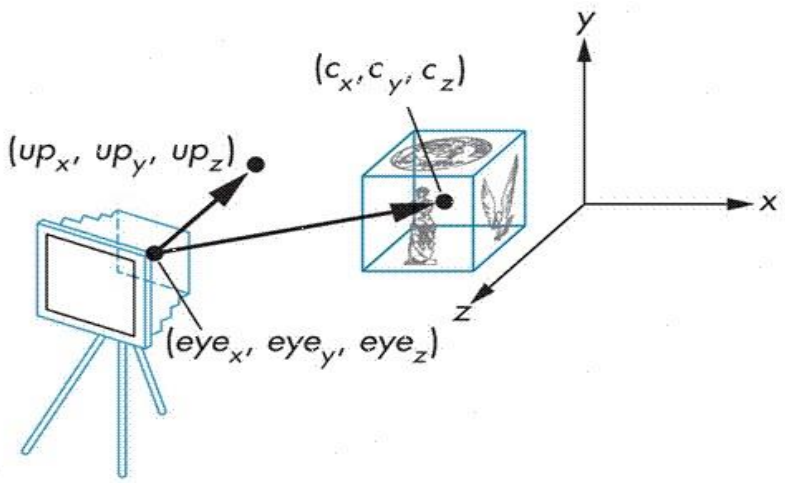


Figura 5. Parámetros de la función gluLookAt (eyeX, eyeY, eyeZ, centerX, centerY, centerZ, upX, upY, upZ)

1.7.5.4.3. Visualización.

Para poder visualizar todo lo almacenado en la computadora, ya sea 3D o 2D, se necesita un puerto de visión, si la figura es en 3D, se mostrará en la pantalla, por lo tanto tendrá que ser proyectado. Para esto, se utiliza la función glViewport, esta especifica la transformación afín de x y y de las coordenadas normalizadas de dispositivo a la ventana de coordenadas, ver *Figura 6*.

Función:

```
Void glViewport (int x, int y, int width, int height);
```

Parámetros:

- *x, y* : Especifica la esquina inferior izquierda del rectángulo de puerto de visión.
- *width*: Especifica el ancho del puerto de visión.
- *height:* Especifica el alto del puerto de visión.

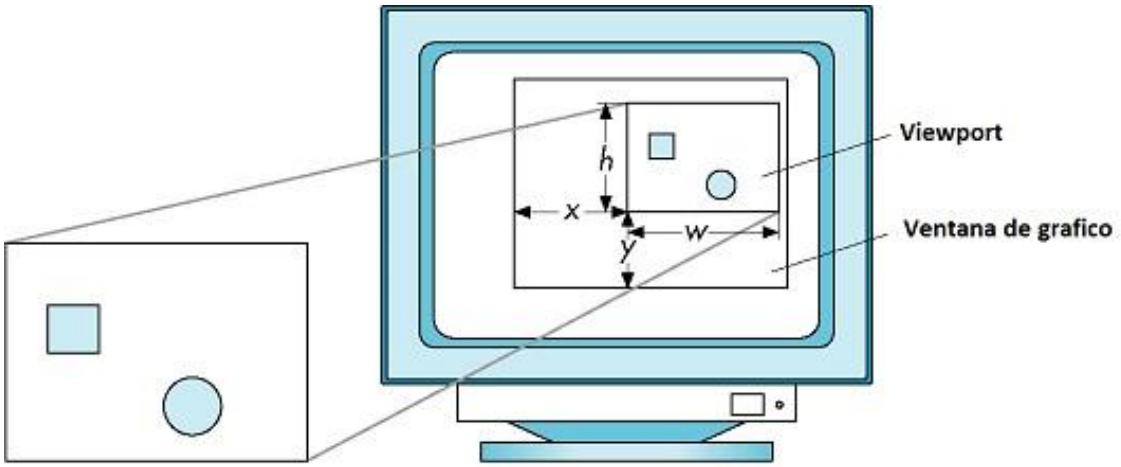


*Figura 6. Parámetros de la función **glViewport** (x, y, width, height)*

1.7.5.4.4. Figuras geométricas.

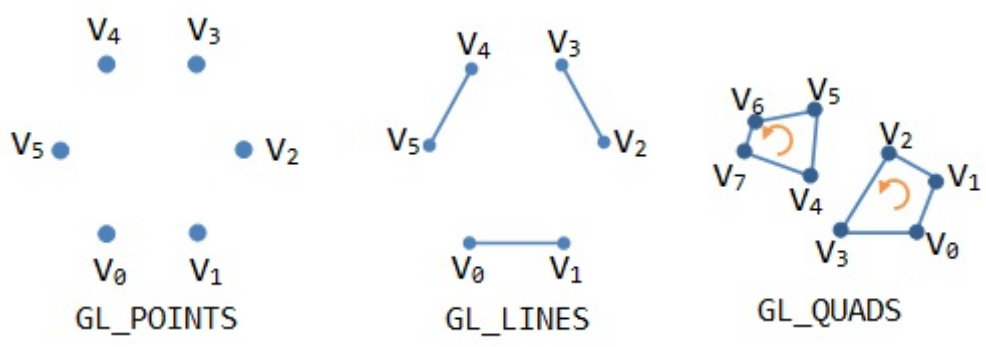
Para cualquier programa de gráficos las figuras geométricas son fundamentales ya que con esta se crea cualquier figura tridimensional, ya sea en base a rectángulos, triángulos, líneas o puntos. Esto, en OpenGL, se genera con la función glBegin, glEnd que delimitan los vértices que definen una primitiva o primitivas que serán creados, hay diez constantes simbólicas que son aceptadas.

Función:

```
glBegin (PARÁMETRO);
        …
glEnd();
```

Parámetros.

- GL_POINTS: Se usa para tratar cada vértice como un único punto. *(Figura 7.a)*
- GL_LINES: Se usa para tratar cada par de vértices como un segmento de línea independiente. *(Figura 7.b)*
- GL_QUADS: Se usa para tratar cada grupo de cuatro vértices como un cuadrilátero independiente. *(Figura 7.c)*



a. **b.** **c.**

Figura 7. Parámetros de la función **glBegin** (..)**glEnd**().
a. *GL_POINTS*, b. *GL_LINES*, c. *GL_QUADS*

1.7.5.4.5. Color de la escena.

El color de fondo es especificado con la función glClearColor, esta especifica los valores rojo, verde, azul y alfa. Utilizando la función glClear para limpiar el color del buffer. Los valores especificados en la función glClearColor varían en 0,1.

Función.

```
Void glClearColor (float Red, float Green, float Blue, float alpha);
```

Parámetros.

- Red, Green, Blue, alpha: Especificar los valores rojo, verde, azul y alfa utilizados cuando se borran los buffers de color. Los valores iniciales son todos iguales a cero.

1.7.5.4.6. Limpiar ventana.

En gráficos por computadora la pantalla constantemente se está refrescando, la función que se encarga de limpiar el color del buffer es *glClear*, esta establece el área de la ventana al valor anteriormente seleccionado, dicha función utiliza una máscara que indica el tipo de buffer a ser limpiado.

Función.

```
Void glClear (PARÁMETRO);
```

Parámetros:

- GL_COLOR_BUFFER_BIT: Indica el buffer actualmente habilitado para la escritura de color.
- GL_DEPTH_BUFFER_BIT: Indica el buffer de profundidad.
- GL_ACCUM_BUFFER_BIT: Indica el buffer de acumulación.
- GL_STENCIL_BUFFER_BIT: Indica el buffer de plantilla.

1.7.5.4.7. Colores.

Para poder dar color a un programa con *OpenGL* es necesario utilizar sus funciones.

Funciones:

```
Void glColor3d (double Red, double Green, double Blue);
Void glColor3f (float Red, float Green, float Blue);
Void glColor3d (int Red, int Green, int Blue);
Void glColor4d (double Red, double Green, double Blue, double Alpha);
Void glColor4f (float Red, float Green, float Blue, float Alpha);
Void glColor4d (int Red, int Green, int Blue, int Alpha);
```

Parámetros:

- Red, Green, Blue: Especifique nuevos valores de rojo, verde y azul para el color actual.
- Alpha: Especifica un nuevo valor alfa para el color actual. Incluido sólo en los cuatro argumentos comandos *glColor4.*

1.7.5.4.8. Matrices.

Para que un programa de gráficos sea bueno se necesita tener muchas vistas, interacción o animación, para eso son necesarias transformaciones que se aplicación

con operaciones matriciales, para elegir que matriz será el objetivo de dichas operaciones se utiliza la función glMatrixMode.

Función:

```
Void glMatrixMode (PARÁMETRO);
```

Parámetros:

- GL_MODELVIEW: Aplica la matriz subsecuente a la pila de la matriz de modelado.
- GL_PROJECTION: Aplica la operación de la matriz subsecuente en la matriz de proyección almacenada en la pila.
- GL_TEXTURE: Aplica la operación de la matriz subsecuente en la matriz de textura almacenada en la pila.
- GL_COLOR: Aplica la operación de la matriz subsecuente en la matriz almacenada en la pila.

1.7.5.4.9. Inicializar matriz.

Como OpenGL utiliza matrices predefinidas es necesario limpiar dicha matriz para utilizarla de nuevo. La función glLoadIdentity remplaza la matriz actual por una matriz identidad, mostrada a continuación:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Función:

```
Void glLoadIdentity ();
```

1.7.5.4.10. Rotación.

Una de las Transformaciones con mayor relevancia es la rotación ya que hace rotar los objetos en muchas maneras, girarlas sobre un eje en específico, su propio eje o en

base canónica. La función de OpenGL para esta transformación es glRotated o glRotatef.

Función:

```
Void glRotated (double Angulo, double x, double y, double z);
Void glRotatef (float Angulo, float x, float y, float z);
```

Parámetros:

- *Angulo:* Especifica el ángulo de rotación en grados.
- *x*, *y*, *z:* Especifican las coordenadas *x, y, z* de un vector, respectivamente.

1.7.5.4.11. Traslación.

Otra transformación muy importante es la traslación, con esta podemos mover cualquier objeto a diferentes lugares en la escena ya sea para aplicarle otra transformación o una visualización en diferente perspectiva. La función de OpenGL para esta trasformación es glTraslate.

Funciones:

```
Void glTraslated (double x, double y, double z);
Void glTraslatef (float x, float y, float z);
```

Parámetros:

- *x*, *y*, *z:* Especificar las coordenadas x, y, z del vector de traslación.

1.7.5.4.12. Vértice.

Un vértice es el punto donde se encuentra dos o más semirrectas o segmentos que conforman un ángulo. Para definir cualquier figura en OpenGL es necesario utilizar la función glVertex, los comandos de dicha función se usan en glBegin/glEnd para especificar pares de puntos, líneas, polígonos o vértices.

Funciones:

```
Void glVertexd (double x, double y, double z);
Void glVertexf (float x, float y, float z);
Void glVertexi (int x, int y, int z);
```

Parámetros:

- *x*, *y*, *z:* Especifica las coordenadas del vértice.

1.7.6. Propuesta de cálculo para la fuerza de resistencia de cuerpos flotantes

1.7.6.1. Proa prismática de cuña.

La forma geométrica más sencilla de una proa de un barco con fondo plano (como son los viejos barcos de guerra) es la cuña, ver *Figura 8*.

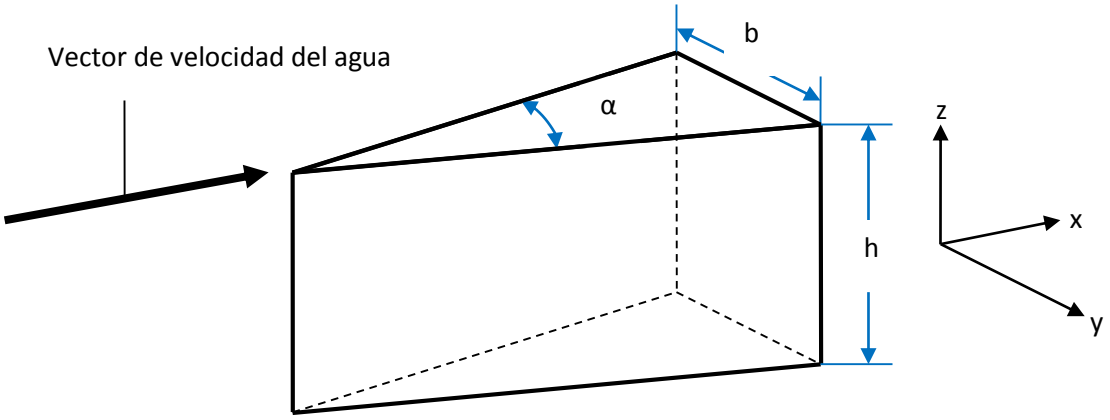


Figura 8. Proa prismática de cuña.

Esta forma nos permite considerar la proa como bidimensional, ya que en dirección de "z" la forma no cambia. Con estas condiciones se puede presentar la corriente como se muestra en la *Figura 9*.

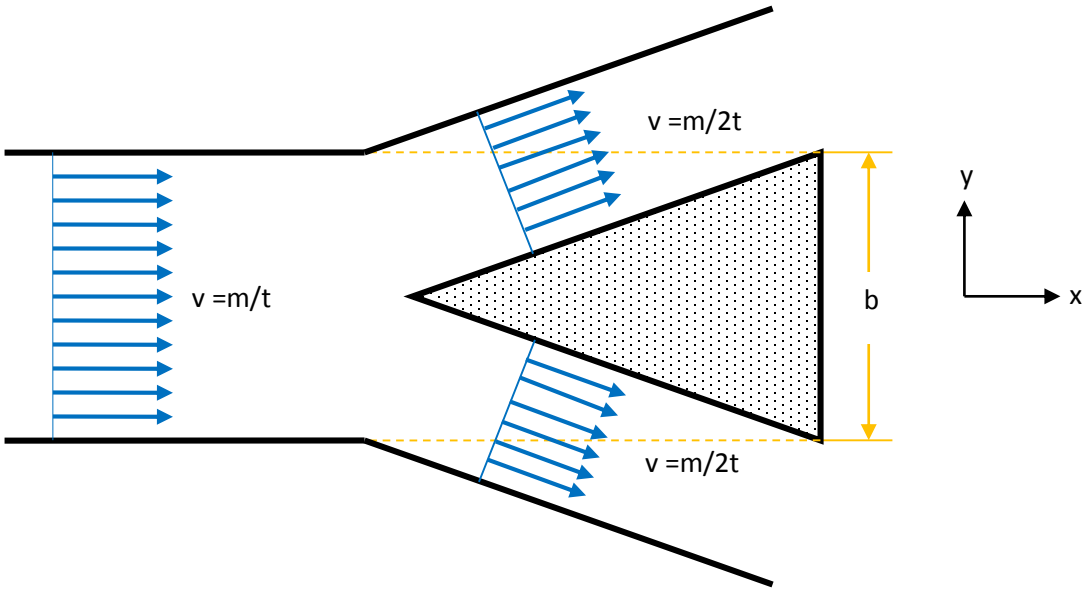


Figura 9. División del flujo de agua por la proa prismática de cuña.

Para poder calcular con parámetros tridimensionales como la masa *m*, se establece un grueso de la cuña en dirección de *z* de la unidad (1). Además se considera la corriente de agua que pega de frente a la cuña, como del mismo ancho *b* como el de la cuña.

Por lo tanto, esto representa que llega un chorro de agua de sección rectangular hacia la cuña, del ancho *b* y del grueso *1*, la cual divide este chorro en dos partes iguales, desviando cada mitad de chorro por el ángulo $\frac{\alpha}{2}$.

La velocidad del agua en los dos chorros desviados no difiere de la velocidad del chorro de entrada, no existe razón para un cambio de velocidad. La justificación se apoya en la presión estática en cada uno de los chorros, la cual debe ser igual a la presión atmosférica, ya que se trata de chorros libres.

La pregunta es, si se comete un error al transferir estas condiciones a la proa de un barco, la que normalmente tiene una extensión considerable en dirección de *z* debajo del agua, incrementándose considerablemente la presión estática; en estos casos se tendría que tomar en cuenta la ecuación de *Bernoulli*.

La respuesta es, que en un fluido incompresible, como es el agua, la presión estática, geodésica o la que sea, no tiene influencia sobre la masa; el cálculo presente se basa exclusivamente en la inercia de la masa desalojada de agua. Si dividimos el chorro rectangular de agua en capas horizontales de un grueso Δz, deben resultar para cada una de estas capas la misma fuerza para desviar el chorro, no obstante si es la capa pegada a la superficie del agua, o la pegada al fondo del barco. Por lo tanto, se puede establecer el siguiente equilibrio de fuerzas mostrado en la *Figura 10*:

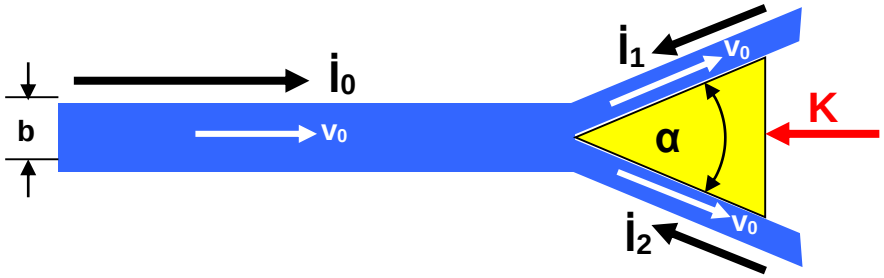


Figura 10. Vectores de impuso.

De este equilibrio de fuerzas se desarrolla el siguiente diagrama vectorial mostrado en la *Figura 11*:

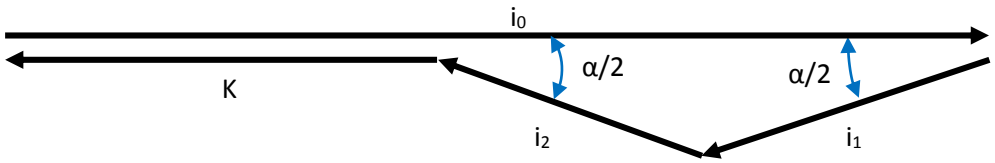


Figura 11. Suma vectorial de fuerzas.

Con la suma vectorial de las fuerzas que intervienen en el desplazamiento:

$$\vec{\imath_0} + \vec{\imath_1} + \vec{\imath_2} + \vec{K} = 0 \qquad (1)$$

De las relaciones geométricas resulta

$$\vec{\imath_0} = \vec{K} + \vec{\imath_1}\cos\left(\frac{\alpha}{2}\right) + \vec{\imath_2}\cos\left(\frac{\alpha}{2}\right) = \vec{K} + (\vec{\imath_1} + \vec{\imath_2})\cos\left(\frac{\alpha}{2}\right) \qquad (2)$$

Recordando que $\vec{\imath_1}$=$\vec{\imath_2}$, sustituyendo en (2):

$$\vec{\imath_0} = \vec{K} + 2\,\vec{\imath_1}\cos\left(\frac{\alpha}{2}\right) \qquad (3)$$

Despejando la fuerza de reacción $\vec{K}$ de (3):

$$\vec{K} = \overrightarrow{\iota_0} - 2\overrightarrow{\iota_1}\cos\left(\frac{\alpha}{2}\right) \quad (4)$$

El impulso $\vec{\iota}$, depende del caudal másico $\dot{m}$ por la velocidad de desplazamiento v.

$$\vec{\iota} = \dot{m}v = \left(\frac{m}{t}\right)v \quad (5)$$

Donde:

m: masa. t: tiempo. v: velocidad.

Sustituyendo (5) en (4).

$$\vec{K} = \left(\frac{m}{t}\right)(v) - 2\left(\frac{1}{2}\right)\left(\frac{m}{t}\right)(v)\,cos\left(\frac{\alpha}{2}\right) \quad (5)$$

Simplificando (5)

$$\vec{K} = \left(\frac{m}{t}\right)(v)\left(1 - cos\left(\frac{\alpha}{2}\right)\right) \quad (6)$$

El caudal másico $\dot{m}$, en términos de densidad ρ:

$$\dot{m} = \rho V/t = \rho A v \quad (7)$$

Siendo ρ la densidad del fluido, A el área de sección transversal del chorro de fluido, y v su velocidad. Se recuerda, que esta velocidad no cambia con la desviación del chorro, como se explicó anteriormente.

Introduciendo las medidas de altura y ancho de la cuña, b y h, para obtener el área A en la fórmula (7):

$$\dot{m} = \frac{m}{t} = \rho \cdot b \cdot h \cdot v \quad (8)$$

La fuerza de resistencia por el desplazamiento del agua para una cuña $\vec{K}$, sustituyendo (8) en (6):

$$\vec{K} = (\rho \cdot b \cdot h \cdot v\,)(v)\left(1 - cos\left(\frac{\alpha}{2}\right)\right) = \rho \cdot b \cdot h \cdot v^2\left(1 - cos\left(\frac{\alpha}{2}\right)\right) \quad (9)$$

Esta fuerza tiene un máximo para un ángulo de $\frac{\alpha}{2} = 90º$, es decir que la cuña se convierte en una placa plana atravesada.

Para $\alpha = 0$, también la fuerza $\vec{K}$ es cero, lo que corresponde a una placa infinitamente delgada, paralela a la corriente de agua. Con eso se comprueba, que no hay errores en las consideraciones anteriores.

1.7.6.2. Proa prismática curva.

La mínima resistencia la tendrá una proa prismática de cantos rectos, no curvos. (Por el momento, se sugiere que la esquina donde termina la proa, en el máximo ancho del casco, tenga un radio tal, que ni se produzcan remolinos ni cavitación).

Por fines de simplificación, se divide la curvatura convexa de un casco real en 2 partes rectilíneas, pero de diferente ángulo, como en la *Figura 12*.

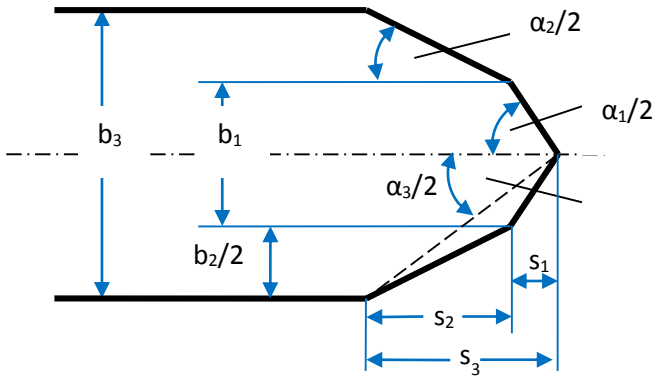


Figura 12. Proa poligonal.

En este caso, la cuña con el ángulo α_3 y con la longitud s_3 es sustituida por la cuña con el ángulo α_1 y con la longitud s_1; y por la cuña con el ángulo α_2 y con la longitud s_2, donde además vale:

$$s_3 = s_1 + s_2 \tag{10}$$

De la *Figura 12*, determinamos que:

$$b_3 = b_1 + b_2 \tag{11}$$

Para un polígono aproximado (convexo), los ángulos valen:

$$\alpha_1 > \alpha_2 > \alpha_3$$

La fuerza de resistencia se define para este caso en:

$$\overrightarrow{K_{1+2}} = \overrightarrow{K_1} + \overrightarrow{K_2} \tag{12}$$

Donde:

$$\overrightarrow{K_1} = \rho b_1 h v^2 \left(1 - cos\left(\frac{\alpha_1}{2}\right)\right) \tag{13}$$

Y

$$\overrightarrow{K_2} = \rho b_2 h v^2 \left(1 - cos\left(\frac{\alpha_2}{2}\right)\right) \tag{14}$$

Sustituyendo (13) y (14) en (12):

$$\overrightarrow{K_{1+2}} = \rho b_1 h v^2 \left(1 - cos\left(\frac{\alpha_1}{2}\right)\right) + \rho b_2 h v^2 \left(1 - cos\left(\frac{\alpha_2}{2}\right)\right) \tag{15}$$

Simplificando (15):

$$\overrightarrow{K_{1+2}} = \rho h v^2 \left[b_1 \left(1 - cos\left(\frac{\alpha_1}{2}\right)\right) + b_2 \left(1 - cos\left(\frac{\alpha_2}{2}\right)\right)\right] \tag{16}$$

En la *Figura 13* es un acercamiento de la proa poligonal (*Figura 12*). De acuerdo al análisis de la Figura 13, se hace obvio que las longitudes e, tienen valores relacionados:

$$e_3 < e_1 + e_2$$

Por lo que las fuerzas de resistencia se definen para este caso en:

$$\overrightarrow{K_3} < \overrightarrow{K_1} + \overrightarrow{K_2} \tag{17}$$

ó

$$\overrightarrow{K_3} < \overrightarrow{K_{1+2}} \tag{17.a}$$

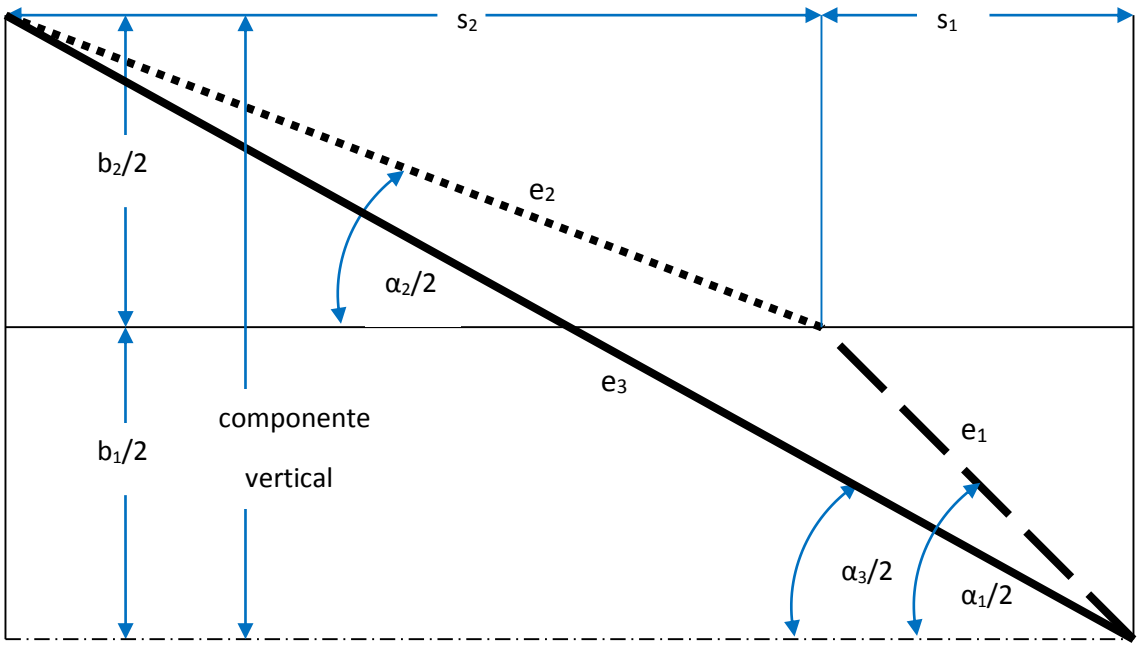


Figura 13. Proa prismática con dos ángulos.

De acuerdo con la ecuación (9), la fuerza $\overrightarrow{K_3}$ para e_3:

$$\overrightarrow{K_3} = \rho b h v^2 \left(1 - cos\left(\frac{\alpha_3}{2}\right)\right) \tag{18}$$

Comprobando (17.a), sustituyendo con las ecuaciones (16) y (18):

$$\overrightarrow{K_3} < \overrightarrow{K_{1+2}} \tag{17.a}$$

$$\rho b h v^2 \left(1 - cos\left(\frac{\alpha_3}{2}\right)\right) < \rho h v^2 \left[b_1 \left(1 - cos\left(\frac{\alpha_1}{2}\right)\right) + b_2 \left(1 - cos\left(\frac{\alpha_2}{2}\right)\right)\right]$$

Simplificando sabiendo que $b_1 = b_2 = \frac{b}{2}$:

$$\rho b h v^2 \left(1 - cos\left(\frac{\alpha_3}{2}\right)\right) < \rho h v^2 \left(\frac{b}{2}\right) \left[\left(1 - cos\left(\frac{\alpha_1}{2}\right)\right) + \left(1 - cos\left(\frac{\alpha_2}{2}\right)\right)\right]$$

$$\left(1 - cos\left(\frac{\alpha_3}{2}\right)\right) < \frac{1}{2}\left[1 - cos\left(\frac{\alpha_1}{2}\right) + 1 - cos\left(\frac{\alpha_2}{2}\right)\right]$$

$$\left(1 - cos\left(\frac{\alpha_3}{2}\right)\right) < \frac{1}{2}\left[2 - cos\left(\frac{\alpha_1}{2}\right) - cos\left(\frac{\alpha_2}{2}\right)\right]$$

$$\left(1 - cos\left(\frac{\alpha_3}{2}\right)\right) < \left[1 - \frac{1}{2}cos\left(\frac{\alpha_1}{2}\right) - \frac{1}{2}cos\left(\frac{\alpha_2}{2}\right)\right]$$

$$-cos\left(\frac{\alpha_3}{2}\right) < -\frac{1}{2}cos\left(\frac{\alpha_1}{2}\right) - \frac{1}{2}cos\left(\frac{\alpha_2}{2}\right)$$

Invirtiendo los signos, resulta:

$$cos\left(\frac{\alpha_3}{2}\right) > \frac{1}{2}cos\left(\frac{\alpha_1}{2}\right) + \frac{1}{2}cos\left(\frac{\alpha_2}{2}\right) \quad (19)$$

Demostrando que (17.a) es cierta, los ángulos de la *Figura 13* son:

$$\frac{\alpha_3}{2} = 29° \qquad \frac{\alpha_1}{2} = 45° \qquad \frac{\alpha_2}{2} = 21°$$

La cuña sigue siendo la forma con la mínima resistencia, aun cuando invertimos los ángulos α_1 y α_2, lo que se puede verificar con la ecuación (19); es decir que no importa si primero ponemos la parte más picuda (línea rayada) o la más chata (línea punteada), ver *Figura 14*.

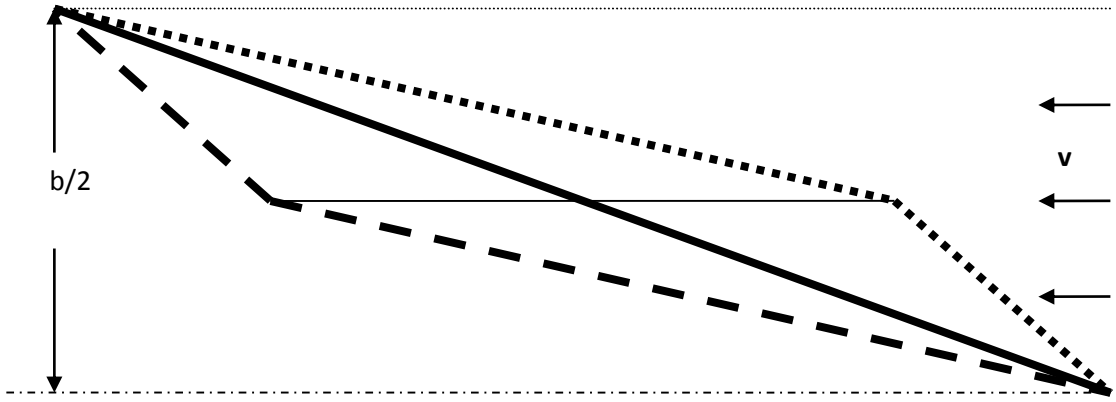
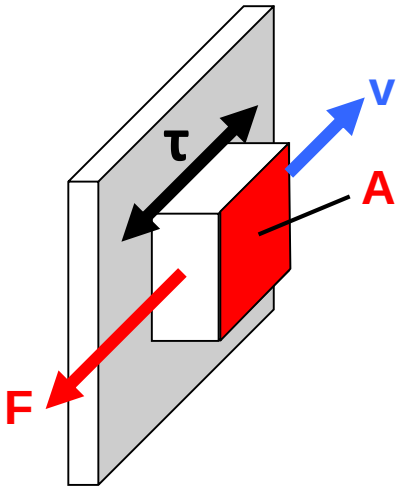


Figura 14. Intercambio de ángulos.

1.7.7. Propuesta de cálculo para la Fricción.

El esfuerzo cortante τ, se define como F/A, donde F es la fuerza del fluido que interactúa con cierta área A del casco. Este esfuerzo es el que define la fuerza de resistencia debido a la fricción. *(Figura 15).*



$\tau = F/A$, es el esfuerzo cortante que se genera entre el agua y la superficie sólida debido a la fricción.

El vector de velocidad relativa v tiene la dirección opuesta que la fuerza F.

Figura 15. Esfuerzo cortante.

Conociendo la superficie mojada se puede, entonces, calcular para cada velocidad relativa agua y superficie sólida, la fuerza de fricción.

En el trabajo de investigación realizado por el Ing. Wolf se obtuvo empíricamente como resultado la siguiente fórmula para calcular el esfuerzo cortante τ:

$$\boldsymbol{\tau = \frac{F}{A} = 0,15v^3 + 3,9v^2 + 1,1v} \quad (20)$$

Despejando la fuerza *F*, queda:

$$\boldsymbol{F = (0,15v^3 + 3,9v^2 + 1,1v)(A)} \quad (21)$$

1.8. Procedimientos y descripción de las actividades realizadas

Las actividades realizadas en este proyecto iniciaron a partir del análisis de un trabajo que le precede, "*Digitalización de gráficas y su conversión en un programa de cálculo*", en donde se manejó un diseño prismático del modelando de un bote para aplicarle el cálculo del Ingeniero Eberhard Wolff Krautter.

Para el diseño de nuestro software se emplearon las técnicas de UML, así como el paradigma orientado a objetos.

1.8.1. Investigación preliminar

La investigación preliminar consistió en una entrevista con el Ingeniero Eberhard Wolff Krautter, con el fin de conocer sus inquietudes en cuanto a la mejora del software realizado anteriormente. Así mismo, se estudió el software "D*igitalización de gráficas y su conversión en un programa de cálculo*" y el documento producto de su trabajo de año sabático denominado "*Propuesta de cálculo para la fuerza de resistencia de cuerpos flotantes en el agua*" (Wolff,2006)

1.8.2. Definición de requerimientos

1.8.2.1. Problemática

El cálculo de la fuerza de resistencia de un barco que se mueve en el agua, y el cálculo de la potencia que este requiere para el movimiento, es sumamente complicado y requiere además de la elaboración de un modelo a escala y de las pruebas de arrastre para las cuales, en México, no existen instalaciones. Resulta, que sin estas instalaciones y sin el software especial para los cálculos que se requieren, actualmente no es posible diseñar el casco de una embarcación del tipo y tamaño que se desea, si se quiere conocer la potencia que este requiere para su movimiento a determinada velocidad.

Anteriormente, se desarrolló en el Instituto Tecnológico de La Paz un software que permite el cálculo de la resistencia para cascos con forma prismática, sin embargo, los cascos de las embarcaciones reales no tienen, por lo general, esta forma, de tal manera que se requiere el desarrollo de un nuevo software que permita reconocer formas con cortes horizontales y curvos, cuya integración de estos cortes lleve al resultado final.

1.8.2.2. Diagrama de casos de uso.

En el diagrama de casos de uso de la *Figura 16* se muestra la descripción de la interacción entre el usuario y el sistema.

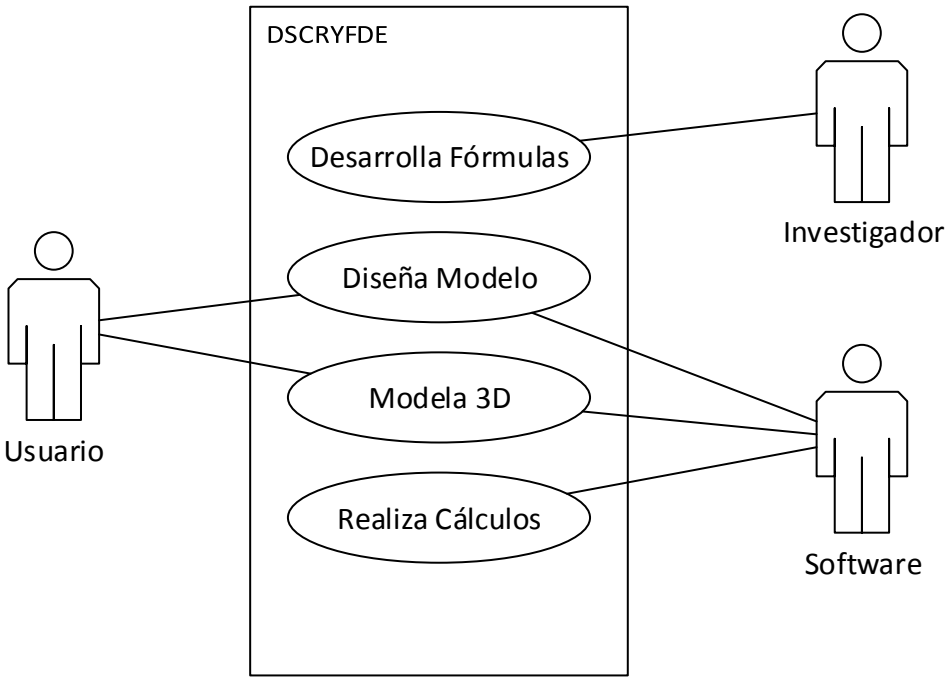


Figura 16. Diagrama de casos de usos.

1.8.2.3. Requerimientos

Basados en la investigación preliminar y la asesoría del Ing. Wolff, se formularon los siguientes requerimientos.

1.8.2.3.1. Requerimientos funcionales

- Diseño gráfico del modelo del casco de una embarcación en 3D y con forma Curva.
- Cálculo de fuerza de resistencia basada en la aceleración de la masa del fluido.
- Cálculo de la fuerza de fricción al desplazarse sobre el agua a una determinada velocidad.
- Tener las 3 vistas correspondientes superior, lateral y frontal.
- Poder modificar la forma en cualquiera de las 3 vistas.
- Variar los parámetros para los cálculos de las fuerzas.

1.8.2.3.2. Requerimientos no funcionales

- Ver diseño en formato con color relleno o en líneas.
- Almacenar los modelos diseñados por el usuario.
- Cargar modelos anteriormente guardados.

Generación de prototipo no funcional 1.

El prototipo no funcional 1 fue diseñado a partir de los requerimientos iniciales del usuario, y con base en la interpretación de éstos, se diseñó la pantalla mostrada en la *Figura 17*.

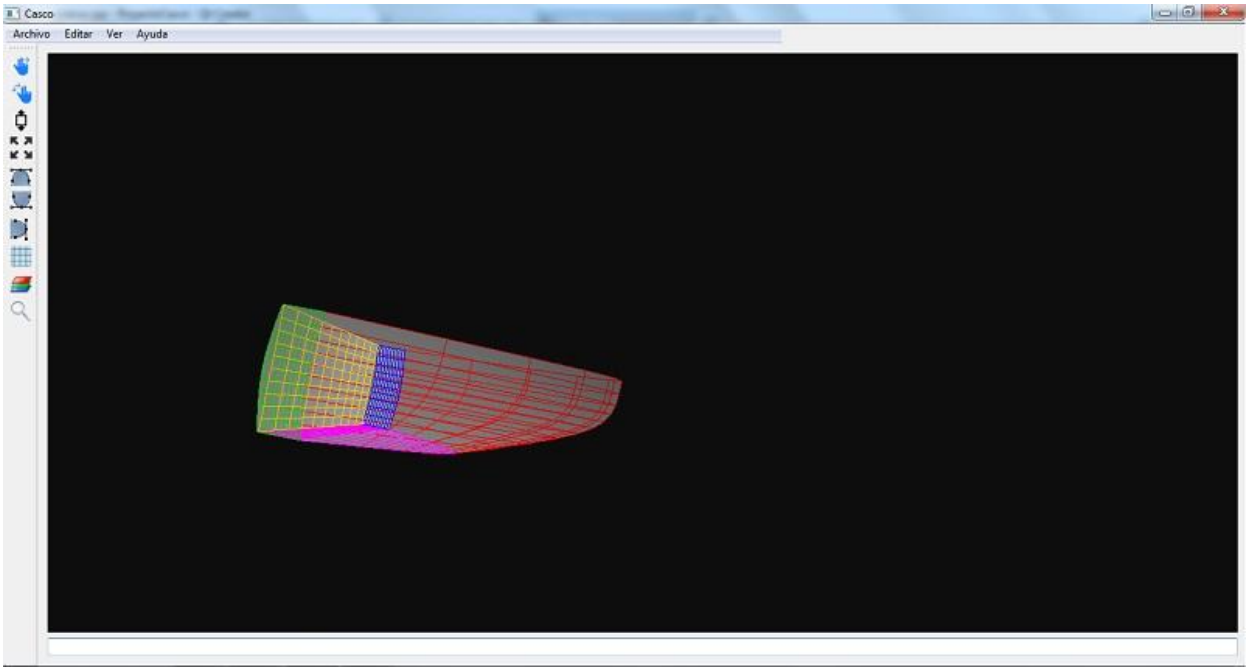


Figura 17. Captura de pantalla del software desarrollado. Prototipo no funcional del tipo 1.

Descripción de la captura de pantalla:

En esta primera aproximación del software, las opciones de diseño se encontraban en una barra en la lateral a la izquierda y tenía una opción de comandos para poder agilizar el desarrollo del diseño del casco, el tener dicho cuadro de comandos haría al usuario dejar de depender del mouse con sólo teclear la acción deseada.

Validación con el usuario:

El primer prototipo permitió refinar la estructura inicial del programa, así como analizar otras características que el software debía contemplar para poder tener un diseño más acercado a la realidad y contemplando la mayor parte de formas posibles. Este primer acercamiento también sirvió para ajustar las características de diseño del funcionamiento del software.

Observaciones:

Se planteó agregar 3 vistas, lateral, frontal y superior para que el usuario pudiera ver al mismo tiempo la embarcación y modificarla acorde a la vista deseada, esto facilitaría la forma de diseñar y modificar la embarcación.

Prototipo no funcional 2:

La barra de menú se colocó en la parte izquierda y se eliminó el control de comandos. Al agregar las tres vistas queda un espacio para poder agregar los formularios donde se mostraría la información y la captura de datos deseada para su análisis. Ver *Figura 18*.

Validación con el usuario:

Con este prototipo se abarcó, casi por completo, los requisitos deseados, agregando opciones como nuevo, guardar y abrir. Es necesario que cada vista sea estática y todas las funciones planteadas anteriormente como mover, acercar o alejar no funcionen para las vistas ya que estas serán exclusivamente para modificar la forma de la embarcación.

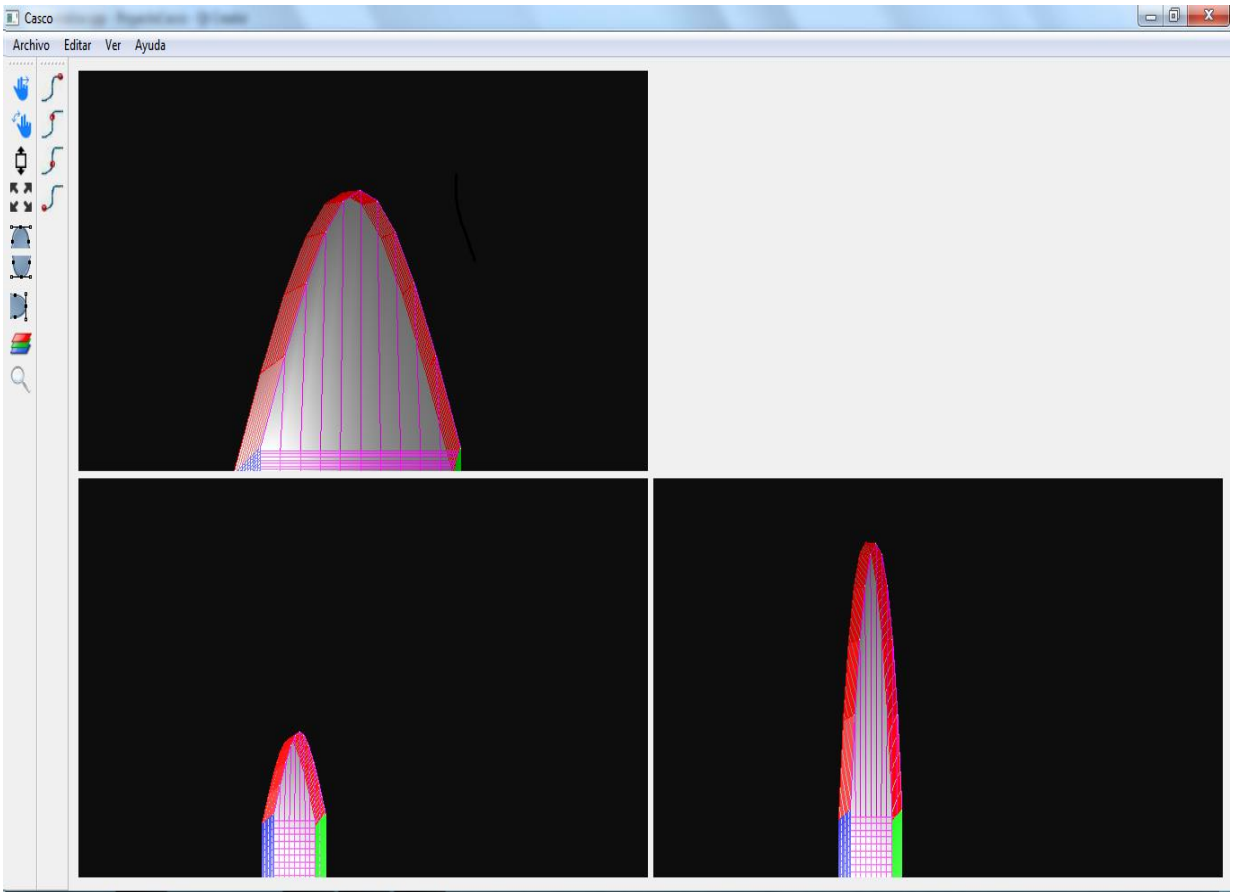


Figura 18. Captura de pantalla del software desarrollado. Prototipo no funcional del tipo 2.

Observaciones:

Se debe agregar una cuarta vista para respetar la esencia inicial del programa en 3D, donde se deberá apreciar la embarcación desde cualquier vista y ángulo, reacomodando los datos de entrada y salida en una barra colocada en la parte derecha de la pantalla.

Prototipo no funcional 3:

El prototipo final cuenta con todos los requisitos que el cliente necesita, incorporando cuatro vistas, donde tres de ellas son estrictamente para la edición y la cuarta para visualización en cualquier ángulo. Se eliminó el control de comandos y se agregó en la parte derecha los controles de entrada y salida de datos.

Validación con el usuario:

Se realizó una demostración con el usuario para validar los requisitos y mostrar la funcionalidad del sistema. Este prototipo cumple finalmente con los requisitos del usuario.

Observaciones:

Se deben de agregar puntos de control para que el usuario pueda ver lo que está modificando. También, cuando el usuario desea modificar el tamaño de la embarcación ya sea alto o largo, los demás valores se deben ajustar proporcionalmente.

1.8.3. Diseño del sistema

1.8.3.2 Diagrama de clases.

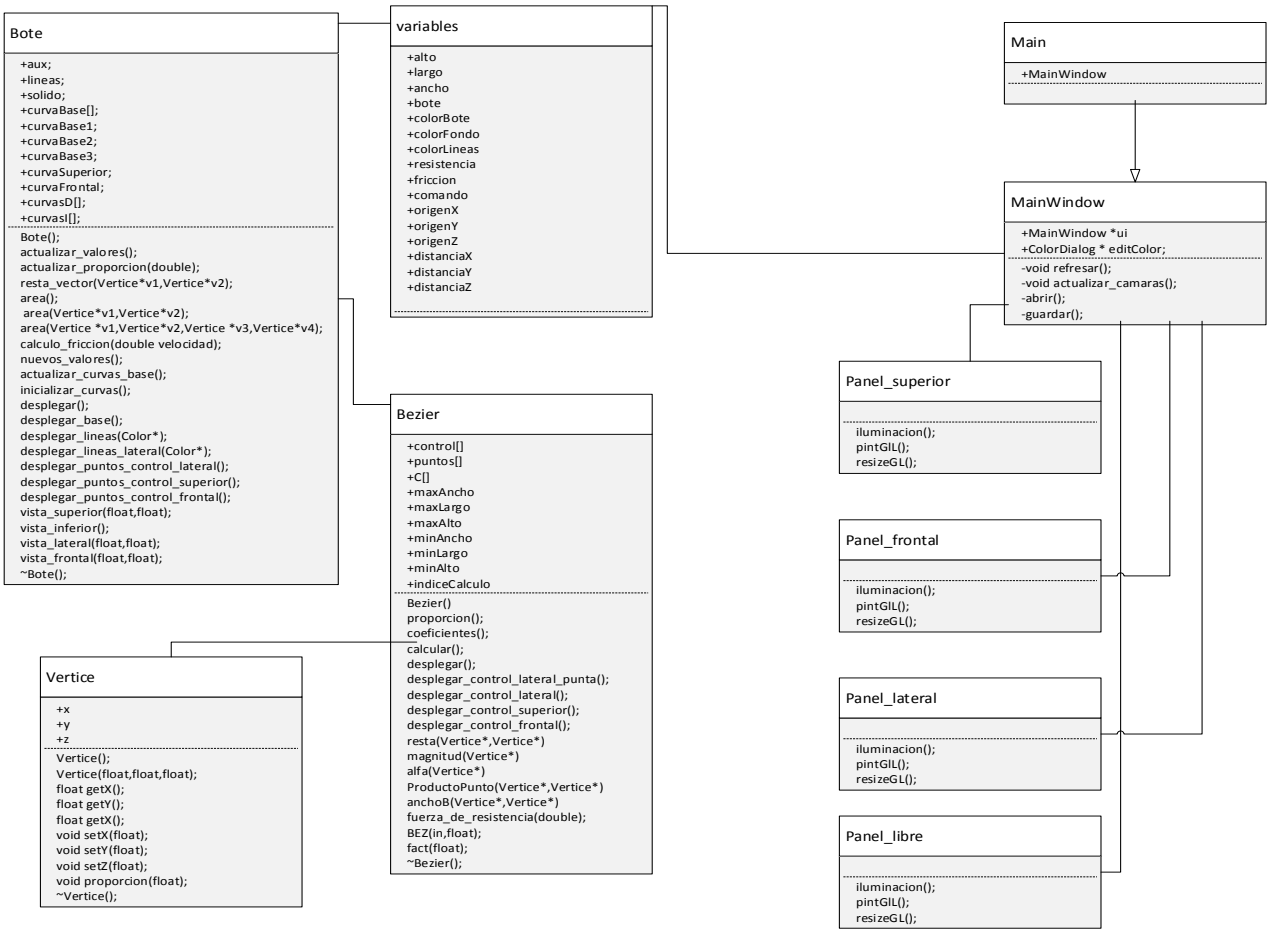


*Figura 19. M*uestra el diagrama de clases del sistema diseñado.

El diagrama de secuencia “Guardar Archivo”, se muestra en la *Figura 20.*

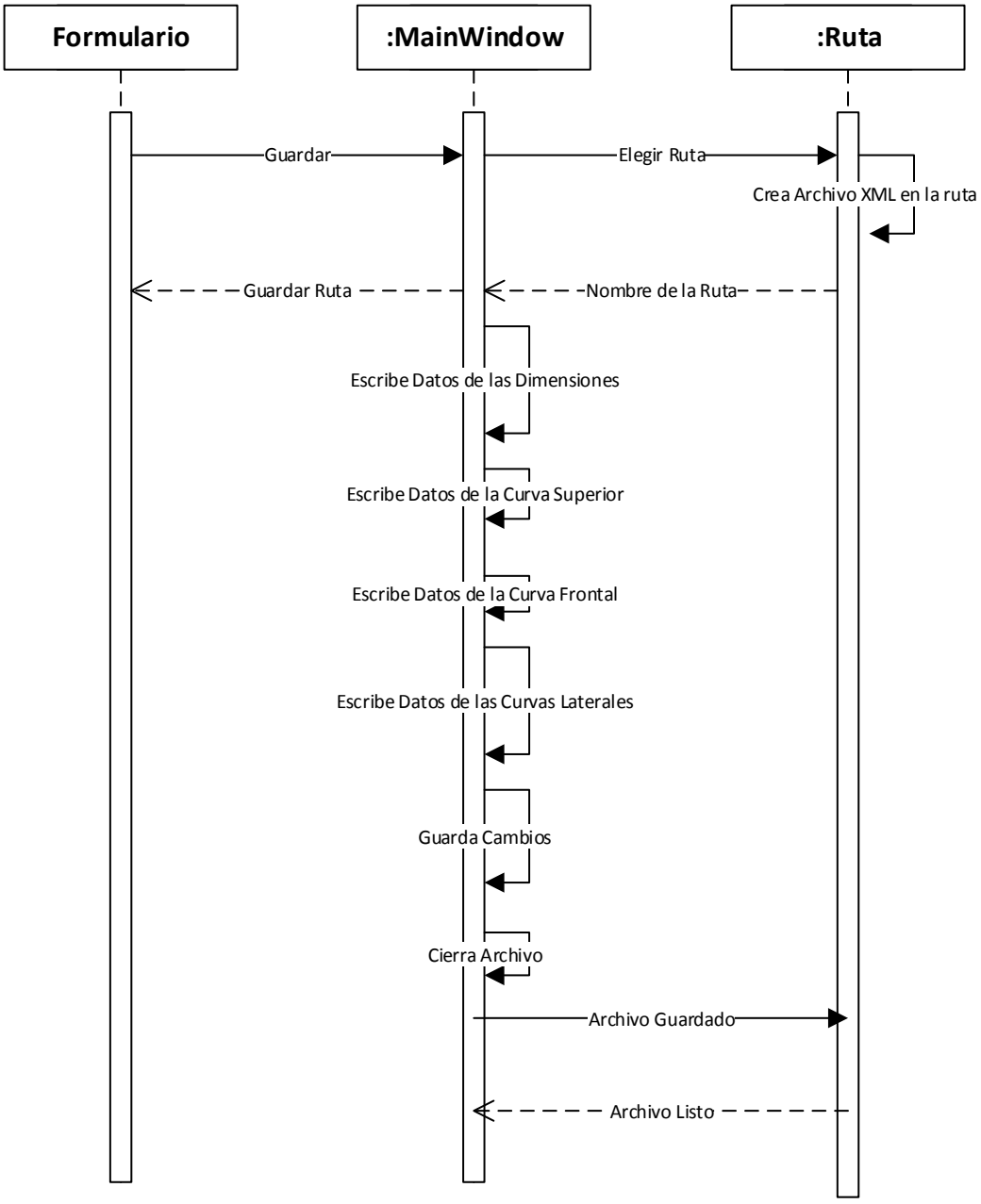


Figura 20. Diagrama de secuencia Guardar Archivo.

El diagrama de secuencia “Abrir Archivo”, se muestra en la *Figura 21.*

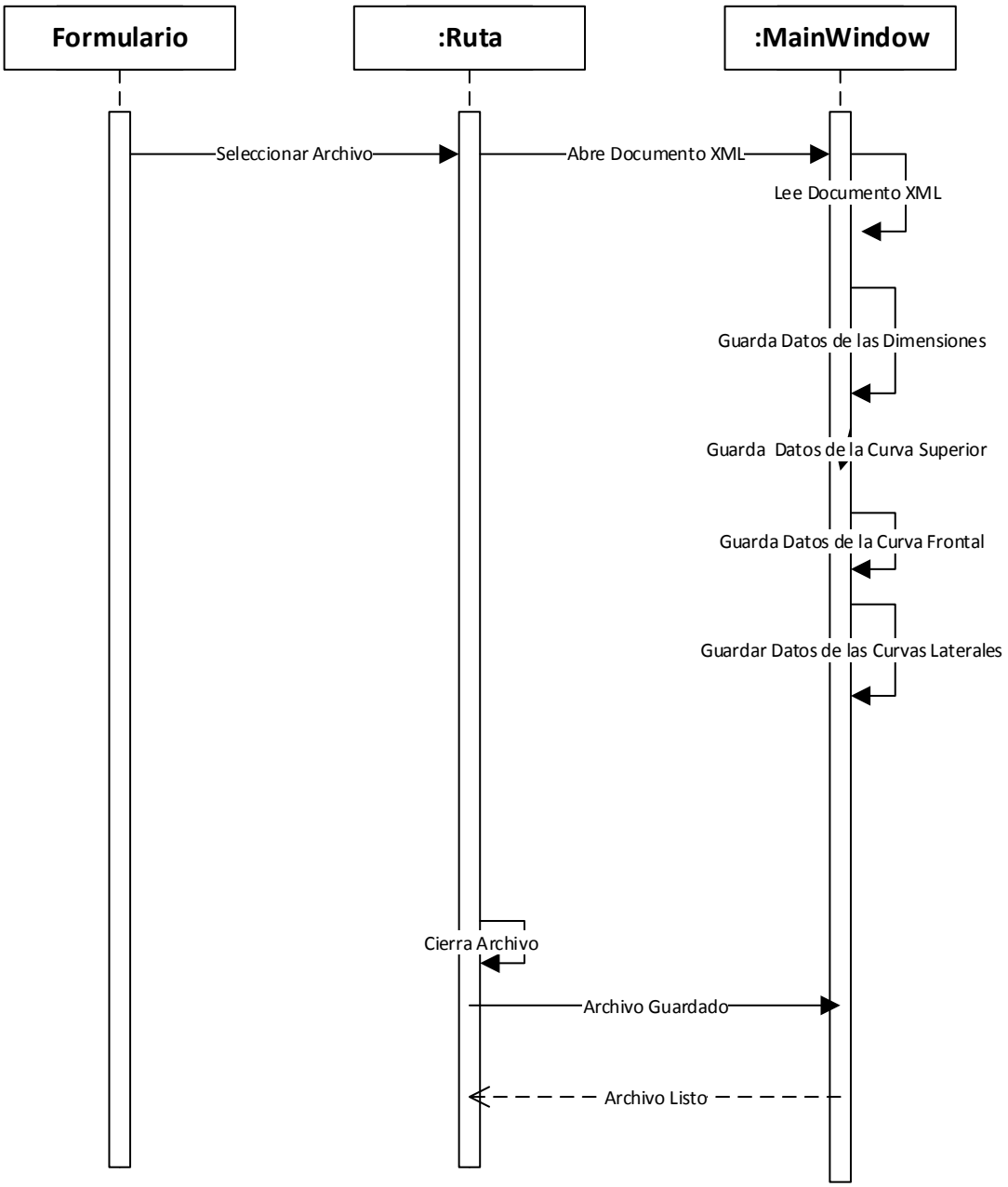


Figura 21. Diagrama de secuencia Abrir Archivo.

1.8.3.3 Funcionamiento.

1.8.3.3.1 Herramientas.

Las herramientas colocadas en el software.


Modificar Dimensiones: esta opción modifica el alto y ancho dependiendo de la vista.


Modificar Vertical: modifica verticalmente las dimensiones dependiendo la vista.


Modificar Horizontal: modifica horizantalmente las dimensiones dependiendo la vista


Habilita la edición de los puntos de control: activa el modo edición de los puntos de control, al darle clic al punto de control rojo se puede mover libremente.


Nuevo: reinicia el archivo y le da los valores predeterminados.


Guardar: guarda el archivo en un formato XML.

Abrir: abre un archivo de dicho proyecto y carga diseño de embarcación.


Visualizar Líneas: se visualiza la embarcación en líneas.


Editar Colores: se abre una ventana para editar los colores de la embarcación, líneas o fondo de pantalla.


Rotar: opción que sirve para rotar la embarcación. Solo funciona en la vista libre (vista superior derecha).


Zoom: opción que sirve para acercar o alejar la embarcación.

1.8.3.3.2 Vistas.

1.8.3.3.2.1 Vista superior.

En la vista presentada en la *Figura 22*, los cuadrados de color rojo son los puntos de control utilizados por el usuario para modificar libremente la forma de la embarcación. Los cuadros de color azul son estáticos, no se pueden modificar.

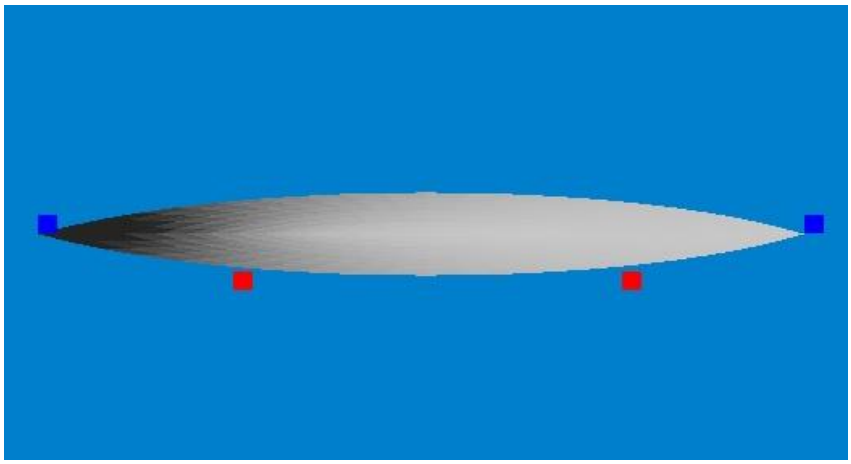


Figura 22. Vista superior.

Para modificar el ancho de la embarcación existen dos formas de hacerlo. Una de ellas es utilizando la herramienta *modificar dimensiones* y otra es la herramienta *modificar vertical* Véase *Figura 23*.

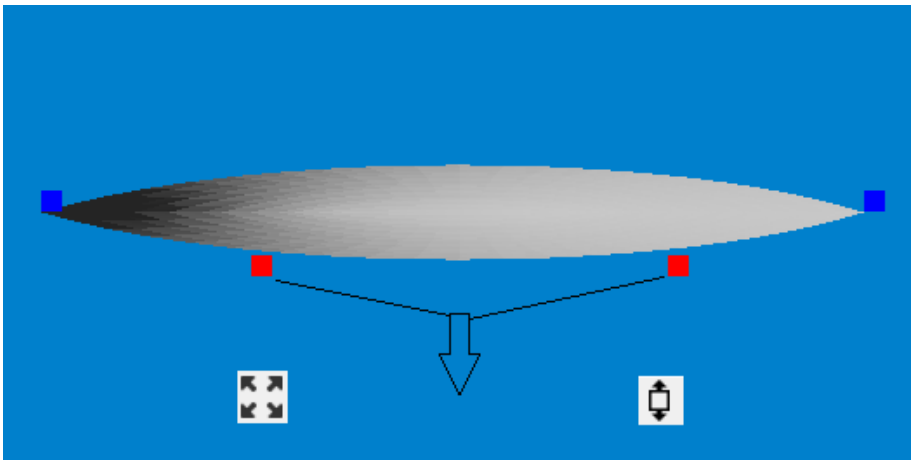


Figura 23. Modificación de dimensiones (ancho) desde la vista superior.

Para generar la forma puntiaguda a la embarcación, se tiene que unir los puntos de control (cuadros rojos), para eso, se utiliza la herramienta *modificar horizontal* o modificar directamente los puntos de control, véase *Figura 24*.

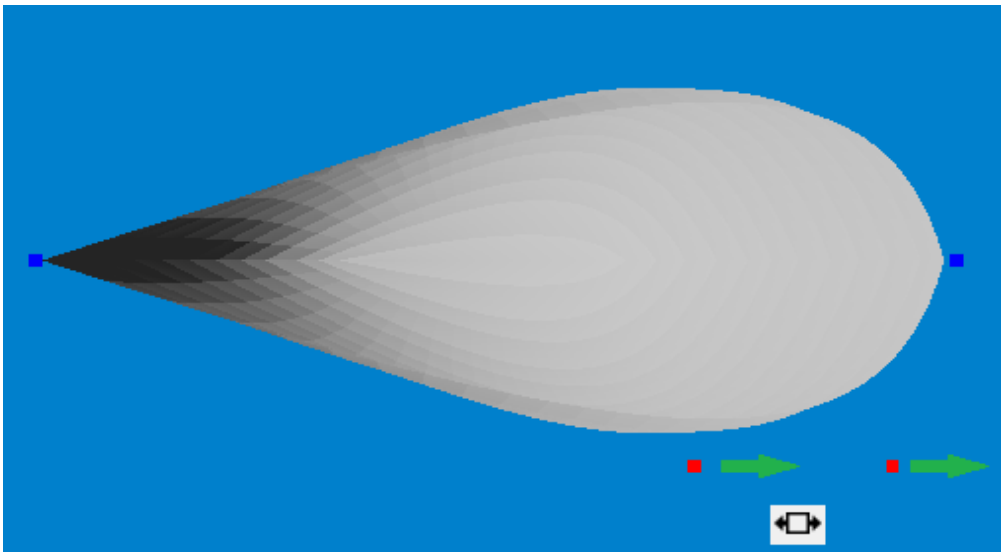


Figura 24. Modificación de dimensiones (forma de la punta) desde la vista superior.

Para darle forma chata, los puntos de control tienen que separarse. Para este método se utiliza la herramienta modificar dimensiones o modificar directamente los puntos de control, véase Figura 25.

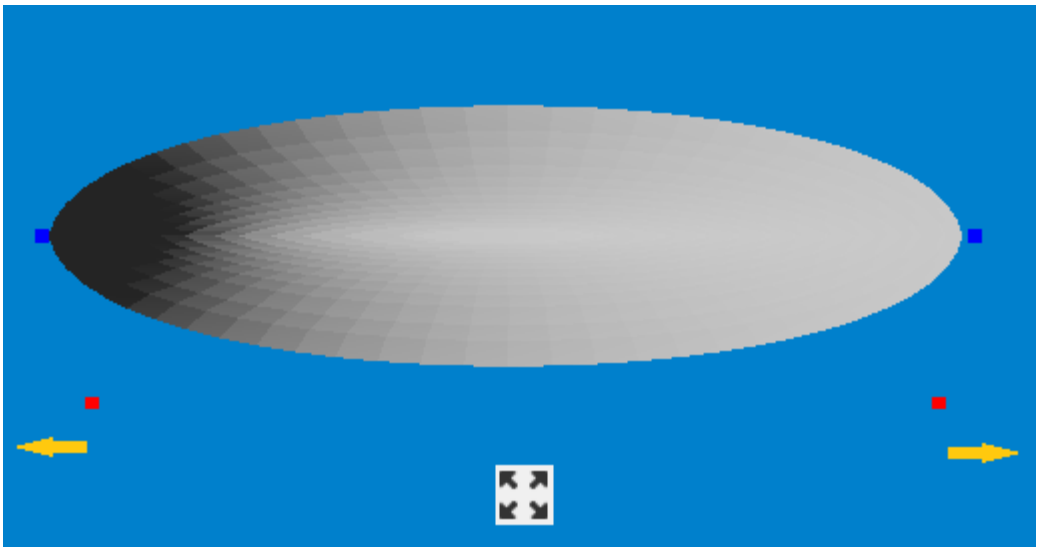


Figura 25. Modificación de dimensiones (forma chata) desde la vista superior.

1.8.3.3.2.2 Vista frontal.

En esta vista se puede modificar el ancho y la forma frontal de la embarcación, ya sea en forma prismática o curva, utilizando las herramientas modificar dimensiones,

modificar horizontal, *modificar vertical* o con los puntos de control individualmente. (*Figura 26*).

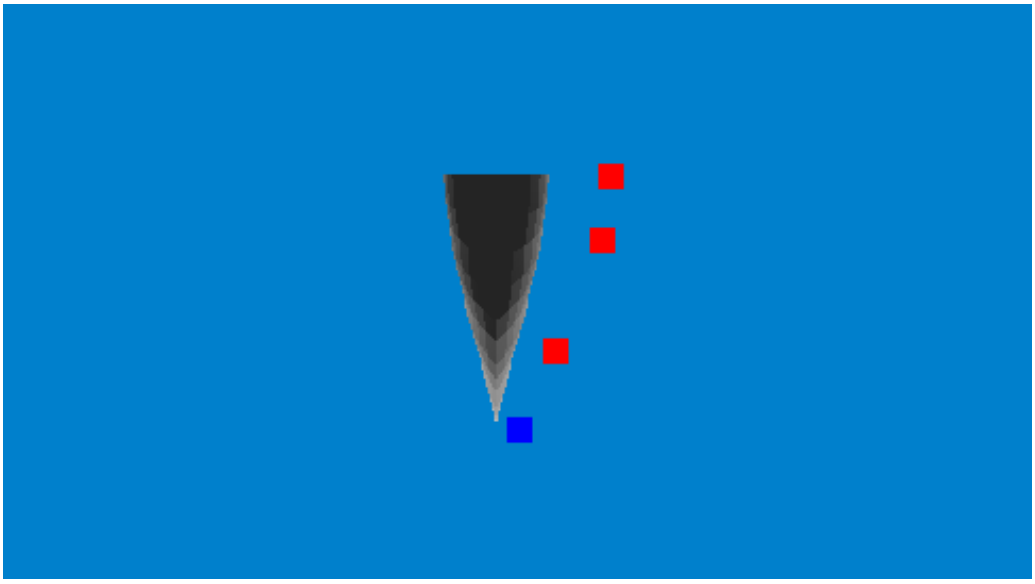


Figura 26. Modificación de dimensiones (forma de la punta) desde la vista frontal.

Si se desea dar una forma *curva* a las partes laterales de la embarcación, los puntos de control del medio (puntos rojos) deben moverse con la herramienta *modificar dimensiones*, *modificar horizontal* o de manera individual. Véase *Figura 27*.

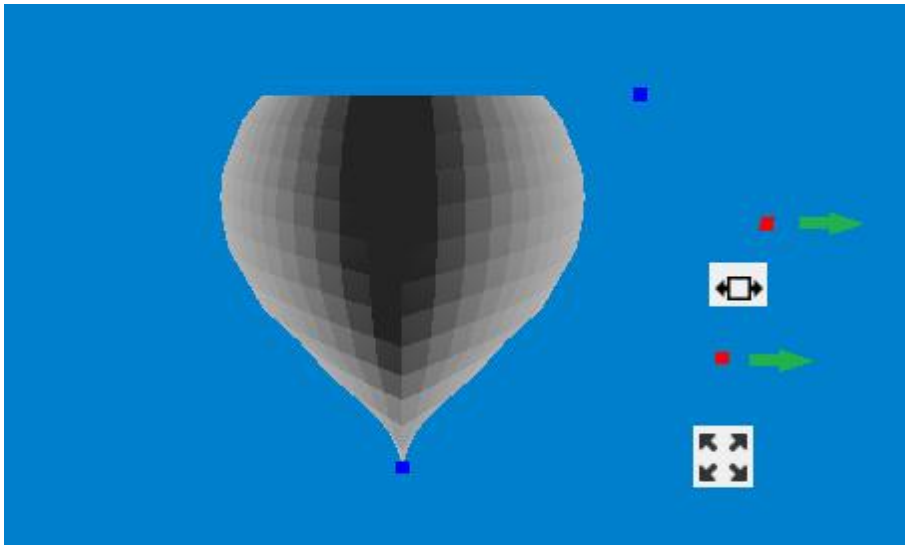


Figura 27. Modificación de dimensiones (ancho) desde la vista Frontal.

Se desea hacer puntiaguda la parte de abajo los puntos de control deben juntarse en la parte superior, es recomendable usar la herramienta modificar vertical o modificar individualmente los puntos de control (*Figura 28*).

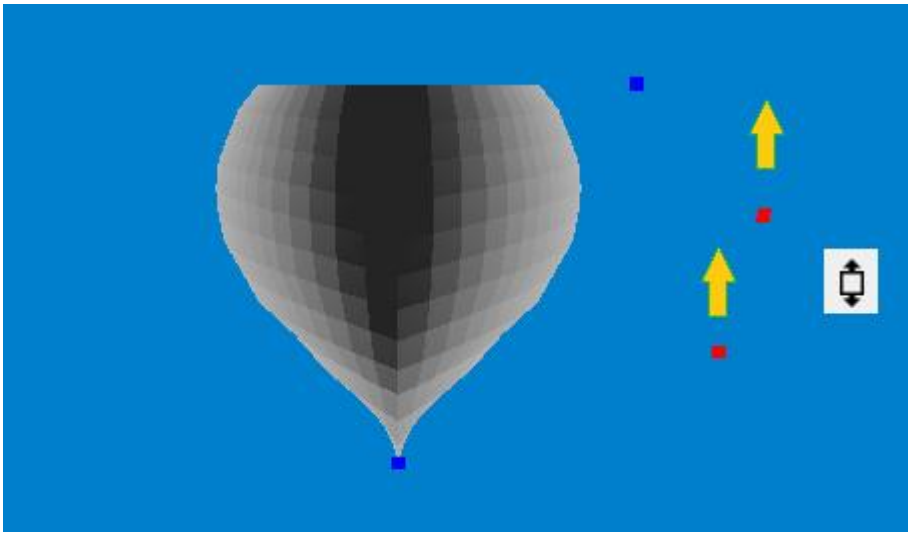


Figura 28. Modificación de dimensiones (forma de la punta) desde la vista frontal.

De la misma manera que en las otras vistas, la herramienta m*odificar dimensiones* separa o junta respectivamente los puntos de control, en la vista frontal los separa de manera vertical moviendo el ratón (mouse) hacía abajo y se une hacía arriba. *Figura 29*

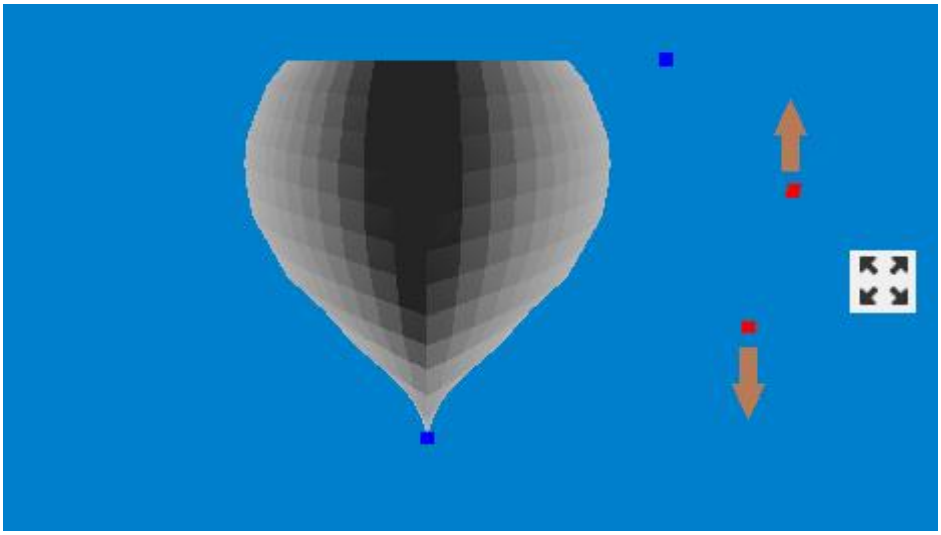


Figura 29. Modificación de dimensiones desde la vista frontal.

1.8.3.3.2.3 Vista lateral.

La vista lateral, es la vista más compleja de todas ya que manejan varios puntos de control, se muestra en la Figura 30.

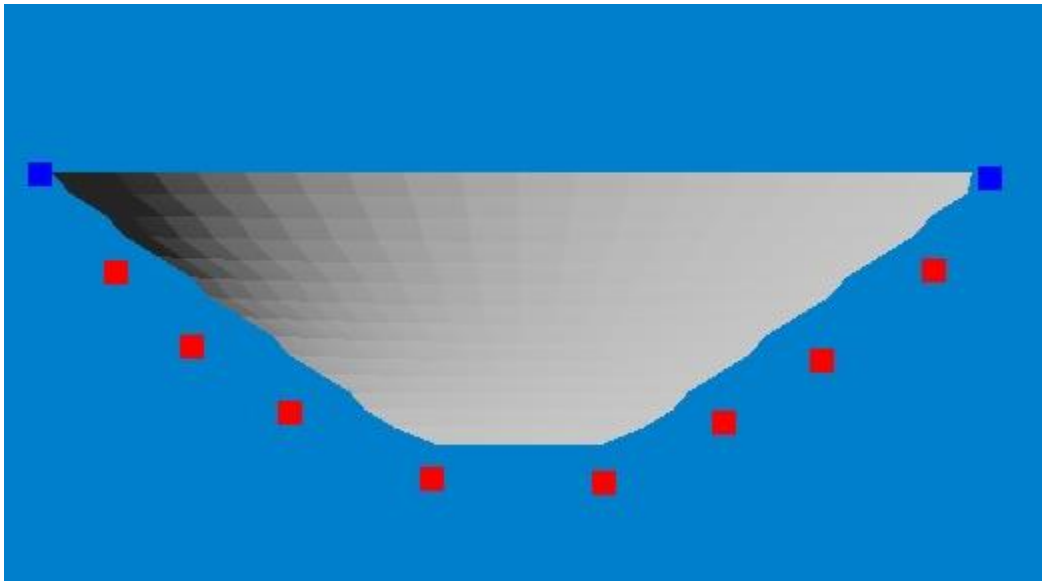


Figura 30. Vista lateral.

Para modificar la altura de la vista lateral se debe utilizar la herramienta *Modificar Vertical*; esta mueve todos los puntos de control ya sea arriba o abajo. (*Figura 31)*.

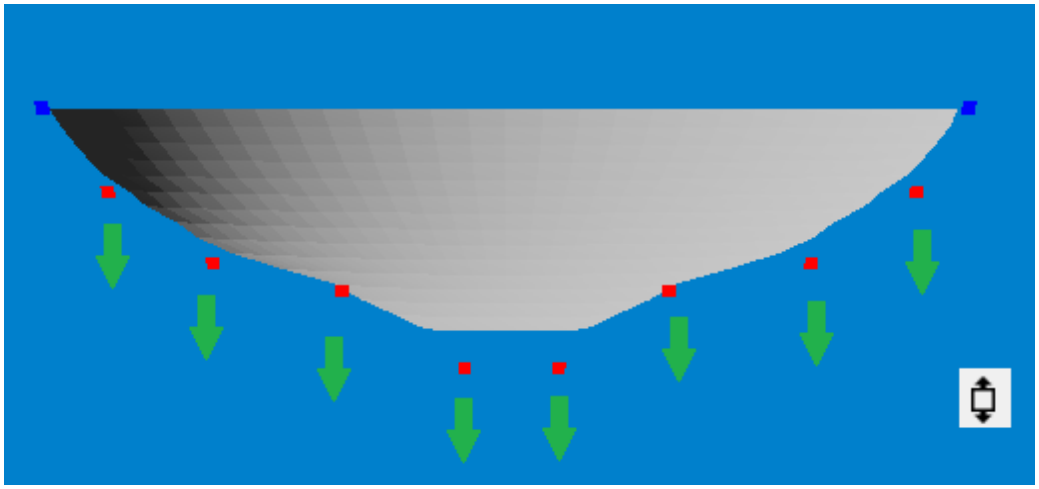


Figura 31. Vista lateral, Modificar Altura.

Para curvear la punta (Proa) y la parte trasera (Popa) se hace uso de la herramienta *Modificar Horizontal,* con ella se mueve los puntos de control marcados en la *Figura 32*, hacía afuera o adentro dependiendo del movimiento del ratón.

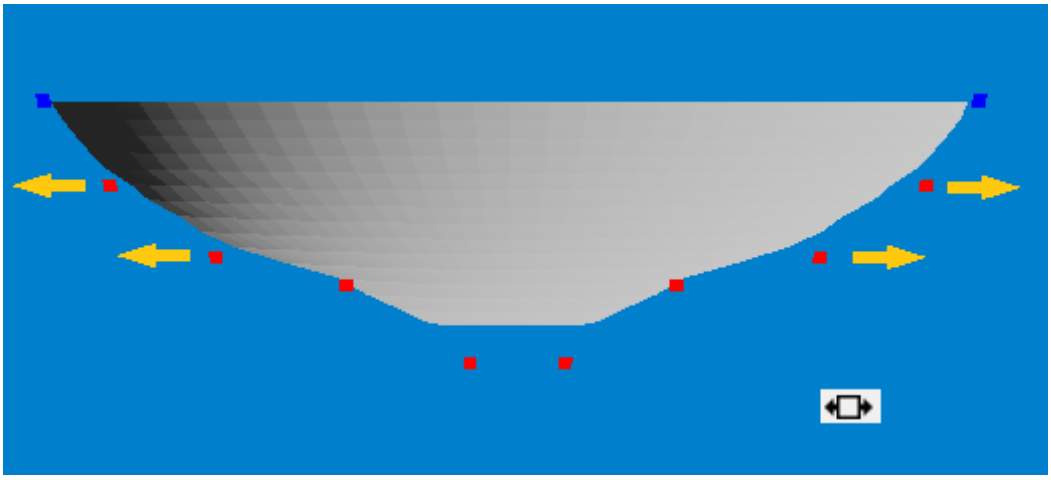


Figura 32. Vista lateral, modificar curvatura.

La Herramienta *Modificar dimensiones, es* la combinación de las dos herramientas anteriores, la única diferencia es que al modificar la altura solo se mueven los puntos de control del centro (*ver Figura 33)*. Esto hace que la embarcación tenga una punta en la parte inferior la cual se puede personalizar moviendo individualmente los puntos de control.

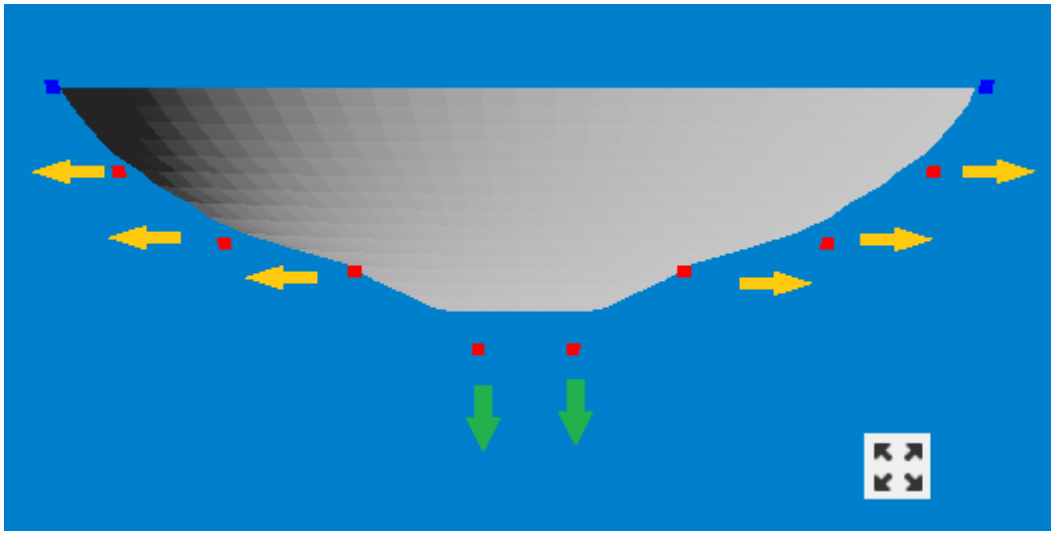


Figura 33. Vista lateral, modificar dimensiones.

1.8.3.3.2.4 Vista libre.

La vista libre fue diseñada para no hacer ninguna modificación al diseño, es simplemente una visualización. En esta vista se puede rotar, acercar y alejar la embarcación, para su mejor apreciación *(Figura 34).*

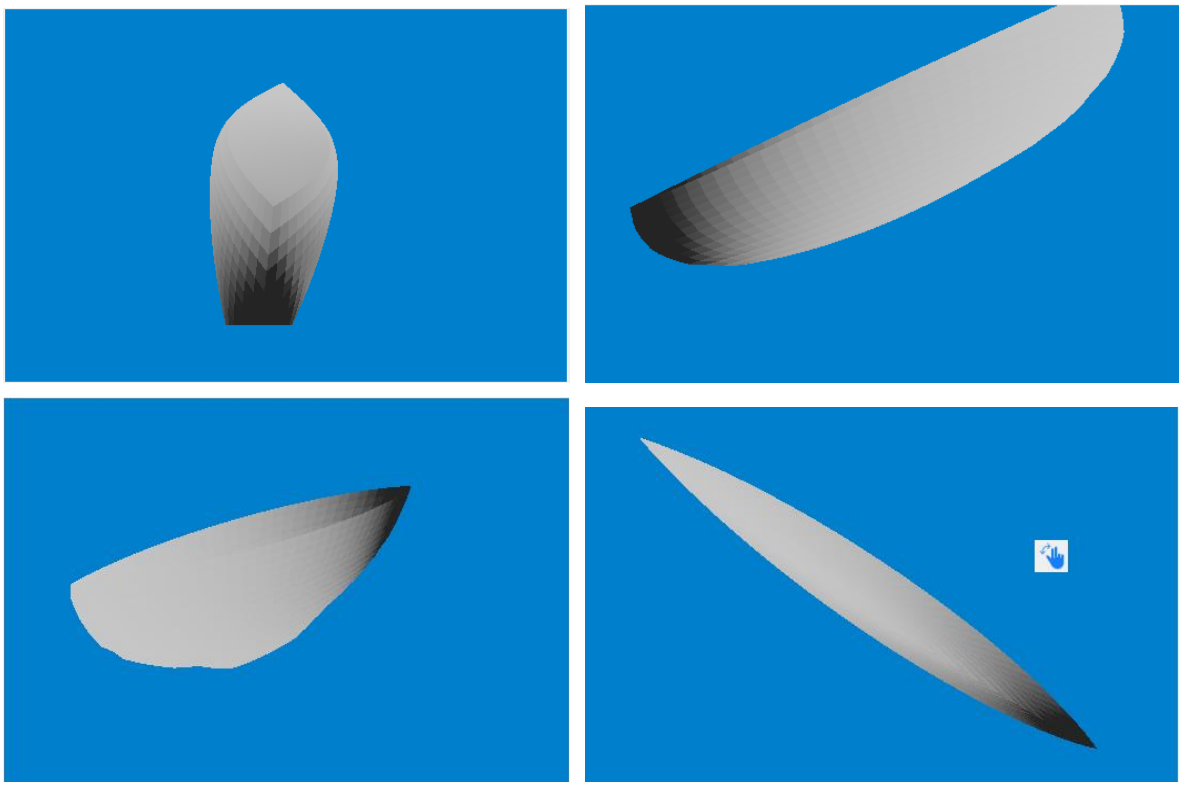


Figura 12. Vista libre, rotando el bote.

1.8.3.3.3 Datos.

1.8.3.3.3.1 Dimensiones.

Otra forma de modificar las dimensiones es escribiendo directamente los datos en los controles correspondientes al largo, alto y ancho. Al modificar cualquiera de estos, automáticamente se modificarán los demás proporcionalmente, ver *Figura 35*.

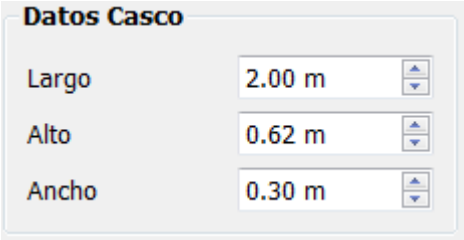


Figura 35. Cuadro de dimensiones

1.8.3.3.3.2 Datos para el cálculo.

Los datos necesarios para hacer el cálculo de ficción y resistencia, son la velocidad y densidad, de los cuales solo variara la velocidad, ya que la densidad es una constante, ver *Figura 36.*

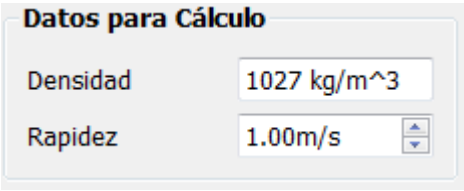


Figura 36. Cuadro de datos para cálculo.

1.8.3.3.3.3 Resultados.

Los resultados de las fuerzas de fricción y resistencia se muestran en dos controles de solo lectura, ya que son valores que solo se visualizarán, no serán editables. Estos cambiarán automáticamente cualquier modificación que se le haga a las dimensiones de la embarcación. *Figura 37*.

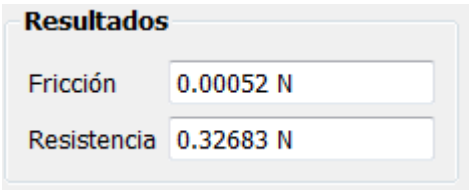


Figura 37. Cuadro de datos para cálculo.

1.8.3.3.4 Capacitación.

La estrategia de capacitación está basada en entrenamiento directo con el usuario. Una vez terminada la capacitación, se le entregará un manual de uso, en el cual pueda consultar todo lo aprendido en dicho entrenamiento.

1.8.3.3.5 Estructura de datos.

1.8.3.3.5.1 Vértice.

La clase Vértice, como su nombre lo indica, se encarga de almacenar los datos de los vértices creados para utilizar en el programa.

```
class Vertice
{
private:
    float x,y,z;
public:
    Vertice();
    Vertice(float,float,float);
    int bandCambio;
    float getX();
    float getY();
    float getZ();
    void setX(float);
    void setY(float);
    void setZ(float);
    void proporcion(float);
    void apagaBanderaCambio();
    ~Vertice();

};
```

1.8.3.3.5.2 Color.

La clase *Color* se encarga de almacenar los colores RGB utilizados, ya que el objeto estando en color solido o trazado en líneas, puede cambiar de color al gusto del usuario.

```
class Color
{
private:
    float r,g,b,alpha;
public:
    Color();
    Color(float,float,float);
    Color(float,float,float,float);
    Color(float);
    float getR();
    float getG();
    float getB();
    float getA();
    void setColor(float,float,float);
    void setR(float);
    void setG(float);
    void setB(float);
    void setA(float);
    ~Color();
};
```

1.8.3.3.5.3 Bezier.

Es la clase encargada de generar curvas, utilizada para darle la forma curva a la embarcación.

```
class Bezier4
{
```

```
public:
    Bezier4();
    Vertice* control[NCONTROL];
    Vertice* puntos[NSUPERFICIES+1];
    Vertice*resta(Vertice*,Vertice*);
    int indiceCalculo;
    int valida_ancho();
    int valida_altura();
    int valida_izquierda();
    int valida_derecha();
    int valida_altura_inicial();
    int valida_altura_inversa();
    float C[NCONTROL];
    float maxAncho,maxLargo,maxAlto;
    float minAncho,minLargo,minAlto;
    float largoControl;
    float BEZ(int,float);
    float fact(float);
    double magnitud(Vertice*);
    double alfa(Vertice*,Vertice*);
    double productoPunto(Vertice*,Vertice*);
    double anchoB(Vertice*,Vertice*);
    void proporcion(float);
    void coeficientes();
    void calcular();
    void desplegar();
    void desplegar_control_lateral_punta();
    void desplegar_control_lateral();
    void desplegar_control_superior();
    void desplegar_control_frontal();
    void fuerza_de_resistencia(double);
    void apagaBanderasCambio();
```

```
    void setL(float);
    ~Bezier4();
};
```

1.8.3.3.5.4 Bote.

La clase bote es el objeto donde guarda toda la información respecto a la figura, está conformado por cinco curvas Bézier principales y dos arreglos de curvas Bézier.

```
class Bote
{
public:

    Bote();
    Color*lineas;
    Color*solido;
    Vertice*curvaBase[NSUPERFICIES*3];
    Vertice*resta_vector(Vertice*v1,Vertice*v2);
    Vertice * normal(Vertice *, Vertice *, Vertice *);
    Bezier4 *curvaBase1;
    Bezier4 *curvaBase2;
    Bezier4 *curvaBase3;
    Bezier4 *curvaSuperior;
    Bezier4 *curvasD[NCURVAS];
    Bezier4 *curvasI[NCURVAS];
    Bezier4 *curvaFrontal;
    double area();
    double area(Vertice*v1,Vertice*v2);
    double area(Vertice *v1,Vertice*v2,Vertice *v3,Vertice*v4);
    void actualizar_valores();
    void actualizar_proporcion(double);
    void calculo_friccion(double velocidad);
    void nuevos_valores();
```

```
    void actualizar_curvas_base();
    void inicializar_curvas();
    void desplegar();
    void desplegar_base();
    void desplegar_lineas(Color*);
    void desplegar_lineas_lateral(Color*);
    void desplegar_puntos_control_lateral();
    void desplegar_puntos_control_superior();
    void desplegar_puntos_control_frontal();
    void vista_superior(float,float);
    void vista_inferior();
    void vista_lateral(float,float);
    void vista_frontal(float,float);
    void apagaBanderasCambio();
    ~Bote();
};
```

1.8.4 Implementación

La implementación surgió con base en los diagramas elaborados en la etapa de diseño, por lo cual se decidió desarrollar el sistema a partir de tres módulos.

- Módulo de construcción.
- Módulo de modelado en 3D.
- Módulo de cálculos.

1.8.4.2 Módulo de construcción.

Al abrir el programa las variables globales se inicializan. Las variables que se incluyen son: un objeto *Bote*, y las variables *largo*, *alto* y *ancho*.

```
            Bote * bote=new Bote();
            float ancho=0.3;    //metros
            float largo=2;            //metros
            float alto=0.55;    //metros
```

El constructor del barco inicializa todas sus respectivas curvas y variables.

```
    this->lineas=colorLineas;
    this->solido=colorBote;
    this->curvaBase1=new Bezier4();
    this->curvaBase2=new Bezier4();
    this->curvaBase3=new Bezier4();
    this->curvaSuperior=new Bezier4();
   this->curvaFrontal=new Bezier4();
```

Después de la inicialización, se establecen los respectivos valores de los puntos de control, en base a las dimensiones *largo*, *alto* y *ancho*.

```
for(int i=0;i<NCURVAS;i++){
   this->curvasD[i]=new Bezier4();
   this->curvasD[i]->control[0]=new Vertice(0,0,alto);
   this->curvasD[i]->control[1]=new
Vertice(ancho/2,largo/4,alto);
   this->curvasD[i]->control[2]=new
Vertice(ancho/2,(largo/4)*3,alto);
   this->curvasD[i]->control[3]=new Vertice(0,largo,alto);

   this->curvasI[i]=new Bezier4();
   this->curvasI[i]->control[0]=new Vertice(0,0,alto);
   this->curvasI[i]->control[1]=new Vertice(-
ancho/2,(largo/4),alto);
```

```
   this->curvasI[i]->control[2]=new Vertice(-
ancho/2,(largo/4)*3,alto);
   this->curvasI[i]->control[3]=new Vertice(0,largo,alto);
  }

float deltaAlto=alto/3;
float deltaLargo=largo/9;
float deltaAncho=ancho/4;



this->curvaBase1->control[0]=new Vertice(ancho/3,0,alto);
this->curvaBase1->control[1]=new
Vertice(ancho/3,deltaLargo,deltaAlto*2);
this->curvaBase1->control[2]=new
Vertice(ancho/3,deltaLargo*2,deltaAlto);
this->curvaBase1->control[3]=new
Vertice(ancho/3,(deltaLargo)*3,0);

this->curvaSuperior->control[0]=this->curvaBase1->control[0];

this->curvaBase2->control[0]=this->curvaBase1->control[3];
this->curvaBase2->control[1]=new Vertice(ancho/3,deltaLargo*4,-
deltaAlto);
this->curvaBase2->control[2]=new Vertice(ancho/3,deltaLargo*5,-
deltaAlto);
this->curvaBase2->control[3]=new
Vertice(ancho/3,deltaLargo*6,base);
this->curvaBase3->control[0]=this->curvaBase2->control[3];
this->curvaBase3->control[1]=new
Vertice(ancho/3,deltaLargo*7,deltaAlto); this->curvaBase3-
>control[2]=newVertice(ancho/3,deltaLargo*8,deltaAlto*2);
this->curvaBase3->control[3]=new Vertice(ancho/3,largo,alto);
```

```
this->curvaSuperior->control[1]=new
Vertice(ancho/2,largo/4,alto);
this->curvaSuperior->control[2]=new
Vertice(ancho/2,(largo/4)*3,alto);
this->curvaSuperior->control[3]=this->curvaBase3->control[3];
this->curvaSuperior->calcular();
this->curvaFrontal->control[0]=new Vertice(0,largo/3,0);
this->curvaFrontal->control[1]=new
Vertice(deltaAncho,largo/3,deltaAlto);
this->curvaFrontal->control[2]=new
Vertice(ancho/2,largo/3,deltaAlto*2);
this->curvaFrontal->control[3]=new
Vertice(ancho/2,largo/3,alto);
this->curvaFrontal->calcular();
```

Una vez que se tienen todas las variables inicializadas, se invoca a la función *actualizar_curva_base();*

```
     void Bote::actualizar_curvas_base(){

float incremento=(this->curvaBase1->maxAlto-this->curvaBase2-
>minAlto)/NCURVAS;
float incrementoY=(this->curvaSuperior->control[1]->getY()-this-
>curvaBase2->control[1]->getY())/NCURVAS;
float incrementoY2=(this->curvaSuperior->control[2]->getY()-
this->curvaBase2->control[2]->getY())/NCURVAS;


   int j=0;
   int z=NSUPERFICIES;
   int k=NSUPERFICIES*3-1;
   for(int i=0;i<NCURVAS;i++){
```

```
this->curvasD[i]->control[0]->setZ(this->curvaBase[0]->getZ()-
i*incremento);
this->curvasD[i]->control[1]->setZ(this->curvasD[i]->control[0]-
>getZ());
this->curvasD[i]->control[2]->setZ(this->curvasD[i]->control[0]-
>getZ());
this->curvasD[i]->control[3]->setZ(this->curvasD[i]->control[0]-
>getZ());

this->curvasI[i]->control[0]->setZ(this->curvaBase[0]->getZ()-
i*incremento);
this->curvasI[i]->control[1]->setZ(this->curvasI[i]->control[0]-
>getZ());
this->curvasI[i]->control[2]->setZ(this->curvasI[i]->control[0]-
>getZ());
this->curvasI[i]->control[3]->setZ(this->curvasI[i]->control[0]-
>getZ());


if(i==0){
   this->curvasD[0]->control[0]->setY(curvaBase[0]->getY());
   this->curvasD[0]->control[3]->setY(curvaBase[NSUPERFICIES*3-
1]->getY());
   this->curvasD[0]->control[1]->setY(curvaSuperior->control[1]-
>getY());
   this->curvasD[0]->control[2]->setY(curvaSuperior->control[2]-
>getY());
   this->curvasI[0]->control[0]->setY(curvaBase[0]->getY());
   this->curvasI[0]->control[3]->setY(curvaBase[NSUPERFICIES*3-
1]->getY());
```

```
   this->curvasI[0]->control[1]->setY(curvaSuperior->control[1]->getY());
   this->curvasI[0]->control[2]->setY(curvaSuperior->control[2]->getY());



   aux=this->curvaFrontal->control[3]->getX();
   this->curvasD[0]->control[1]->setX(aux);
   this->curvasD[0]->control[2]->setX(aux);
   this->curvasI[0]->control[1]->setX(-aux);
   this->curvasI[0]->control[2]->setX(-aux);

    }else{
  while(curvaBase[j]->getZ()>curvasD[i]->control[0]->getZ()){j++;}
  while(curvaBase[k]->getZ()>curvasD[i]->control[0]->getZ()){k--;}
  while(curvaFrontal->puntos[z]->getZ()>curvasD[i]->control[0]->getZ()&&z>0){z-;}

  this->curvasD[i]->control[0]->setY(curvaBase[j-1]->getY());
  this->curvasD[i]->control[1]->setY(this->curvaSuperior->control[1]->getY()-i*incrementoY);
  this->curvasD[i]->control[2]->setY(this->curvaSuperior->control[2]->getY()-i*incrementoY2);
 this->curvasD[i]->control[3]->setY(curvaBase[k+1]->getY());
 this->curvasI[i]->control[0]->setY(curvaBase[j-1]->getY());
 this->curvasI[i]->control[1]->setY(this->curvaSuperior->control[1]->getY()-i*incrementoY);
 this->curvasI[i]->control[2]->setY(this->curvaSuperior->control[2]->getY()-i*incrementoY2);
 this->curvasI[i]->control[3]->setY(curvaBase[k+1]->getY());
```

```
 this->curvasD[i]->control[1]->setX(this->curvaFrontal-
>puntos[z]->getX());
 this->curvasD[i]->control[2]->setX(this->curvaFrontal-
>puntos[z]->getX());
this->curvasI[i]->control[1]->setX(-this->curvaFrontal-
>puntos[z]->getX());
this->curvasI[i]->control[2]->setX(-this->curvaFrontal-
>puntos[z]->getX());}

this->curvasD[i]->calcular();
this->curvasI[i]->calcular();
}
}
```

1.8.4.3 Módulo de modelado en 3D

Para el despliegue de la figura en los diferentes puertos, se utilizó el mismo método para validar el desplegado, iluminación, y el cambio de tamaño de la pantalla.

El siguiente es el evento que dibuja los objetos en el puerto, éste se invoca constantemente mientras se esté interactuando con el formulario. Es un método de extensión de OpenGL.

```
void panel::paintGL(){
    glClearColor(colorFondo->getR(),colorFondo-
>getG(),colorFondo->getB(), 1.0f);

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    iluminacion();

    glViewport(0, 0, ancho_widget, alto_widget);
    glMatrixMode(GL_PROJECTION);
```

```
    glLoadIdentity();

    gluPerspective(25.0f, (float)ancho_widget/alto_widget, 0.2f,
1000.0f);

    gluLookAt(distanciaX, distanciaY, distanciaZ, origenX,
origenY,
      origenZ,0,0,1);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

    if (comando=="r"||comando=="rotar")
    {

      glTranslatef(-origenX,-origenY,-origenZ);
      glRotatef(alpha, 1.0f, 0.0f, 0.0f);
      glRotatef(beta,  0.0f, 0.0f, 1.0f);
      glTranslatef(origenX,origenY,origenZ);

    }
    glTranslatef(-origenX,-origenY,-origenZ);
    glRotatef(25, 1.0f, 0.0f, 0.0f);
    glRotatef(-15,  0.0f, 0.0f, 1.0f);
    glTranslatef(origenX,origenY,origenZ);
```

En esta parte del método se pregunta qué visualización será la que se mostrará. La variable relleno es la que decide si se mostrará la embarcación con dibujo relleno en colores o solo las líneas de los contornos.

```
       if(relleno=='s'){bote->desplegar();bote-
>desplegar_base();}
      else{bote->desplegar_lineas(colorLineas);}
```

```
        bote->actualizar_curvas_base();
    }
}
```

El evento que redimensiona los objetos en el puerto, ya sea porque cambiaron de tamaño la ventana o la maximizaron es *resizeGL(float, float).*

```
void panel::resizeGL(int width,int height){

    if(height == 0)
        height = 1;
    glViewport(0, 0, width, height);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(60.0f, 1.0f, 1.0f,100.0f);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

}
```

Para la iluminación se utilizaron los métodos predefinidos de OpenGL, donde se definen las variables para dar la luz ambiente.

```
void panel::iluminacion ()
{
    GLfloat mat_diffuse [] = {0.3, 0.3, 0.3, 1.0};
    GLfloat mat_specular [] = {0.3, 0.3, 0.3, 1.0};
    GLfloat mat_shininess [] = {50};
    GLfloat light_position0[] = {0, -largo/2, largo/2, 1.0};
    glLightfv(GL_LIGHT0,GL_POSITION, light_position0);
```

```
    GLfloat light_position1[] = {0, -largo/2, -(alto+largo/2),
1.0};
    glLightfv(GL_LIGHT1,GL_POSITION, light_position1);
    glEnable(GL_NORMALIZE);
    glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
    glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
    glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);

    glEnable (GL_LIGHTING);
    glEnable (GL_LIGHT0);
    glEnable (GL_LIGHT1);
    glEnable(GL_COLOR_MATERIAL);
    glEnable(GL_CULL_FACE);
    glCullFace(GL_BACK);
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
    glEnable(GL_DEPTH_TEST);

}
```

Para desplegar el bote en la pantalla se utiliza el arreglo de curvas, y se recorren tomando los vértices para formar las superficies rectangulares, para posteriormente desplegarlas con el método GL_QUADS de OpenGL.

```
void Bote::desplegar(){
    glShadeModel(GL_SMOOTH);
    glBegin(GL_QUADS);

    for(int nivel=0;nivel<NCURVAS-1;nivel++){
        for(int numPuntos=0;numPuntos<NSUPERFICIES;numPuntos++){

           glColor3f(this->solido->getR(),this->solido-
>getG(),this->solido->getB());
```

```
Vertice *n=normal(this->curvasD[nivel]
>puntos[numPuntos+1],this->curvasD[nivel]-
>puntos[numPuntos],this->curvasD[nivel+1]->puntos[numPuntos]);
            glNormal3f(n->getX(),n->getY(),n->getZ());
            delete n;

            glNormal3f(-this->curvasD[nivel]-
        >puntos[numPuntos+1]->getX(),-this->curvasD[nivel]-
        >puntos[numPuntos+1]->getY(),-this->curvasD[nivel]-
        >puntos[numPuntos+1]->getZ());

        glVertex3f(this->curvasD[nivel]->puntos[numPuntos+1]-
>getX(),this->curvasD[nivel]->puntos[numPuntos+1]->getY(),this-
>curvasD[nivel]->puntos[numPuntos+1]->getZ());

        glVertex3f(this->curvasD[nivel]->puntos[numPuntos]-
>getX(),this->curvasD[nivel]->puntos[numPuntos]->getY(),this-
>curvasD[nivel]->puntos[numPuntos]->getZ());

        glVertex3f(this->curvasD[nivel+1]->puntos[numPuntos]-
>getX(),this->curvasD[nivel+1]->puntos[numPuntos]->getY(),this-
>curvasD[nivel+1]->puntos[numPuntos]->getZ());

        glVertex3f(this->curvasD[nivel+1]->puntos[numPuntos+1]-
>getX(),this->curvasD[nivel+1]->puntos[numPuntos+1]-
>getY(),this->curvasD[nivel+1]->puntos[numPuntos+1]->getZ());

        n=normal(this->curvasD[nivel+1]->puntos[numPuntos+1],this-
>curvasD[nivel+1]->puntos[numPuntos],this->curvasD[nivel]-
>puntos[numPuntos]);
```

```
        glNormal3f(n->getX(),n->getY(),n->getZ());
       delete n;

      glNormal3f(-this->curvasD[nivel+1]->puntos[numPuntos+1]-
>getX(),-this->curvasD[nivel+1]->puntos[numPuntos+1]->getY(),-
this->curvasD[nivel+1]->puntos[numPuntos+1]->getZ());

      glVertex3f(this->curvasD[nivel+1]->puntos[numPuntos+1]-
>getX(),this->curvasD[nivel+1]->puntos[numPuntos+1]-
>getY(),this->curvasD[nivel+1]->puntos[numPuntos+1]->getZ());

      glVertex3f(this->curvasD[nivel+1]->puntos[numPuntos]-
>getX(),this->curvasD[nivel+1]->puntos[numPuntos]->getY(),this-
>curvasD[nivel+1]->puntos[numPuntos]->getZ());

      glVertex3f(this->curvasD[nivel]->puntos[numPuntos]-
>getX(),this->curvasD[nivel]->puntos[numPuntos]->getY(),this-
>curvasD[nivel]->puntos[numPuntos]->getZ());

      glVertex3f(this->curvasD[nivel]->puntos[numPuntos+1]-
>getX(),this->curvasD[nivel]->puntos[numPuntos+1]->getY(),this-
>curvasD[nivel]->puntos[numPuntos+1]->getZ());
    }
    glEnd();
}
```

1.8.4.4 Módulo de cálculos

1.8.4.4.1 Resistencia

La fuerza de resistencia se calcula por cortes verticales que se realizan sobre el bote. Los resultados de cada corte se suman, hasta llegar al último nivel. A continuación se presenta el método que recorre el arreglo de curvas que corresponden a cada corte.

```
for(int i=0;i<NCURVAS;i++){
     bote->curvasD[i]->fuerza_de_resistencia(ui->rapidezTxt-
>value());
   }
```

En cada iteración del ciclo anterior, se recorren todos los puntos de la curva correspondiente, para realizar el cálculo mostrado en la sección 1.7.6.2.

```
void Bezier4::fuerza_de_resistencia(double rapidez){
    double resis=DENSIDAD*rapidez*rapidez*(alto/NCURVAS);
    double suma=0;
    for(int i=0;i<indiceCalculo;i++){
    suma=suma+anchoB(puntos[i],puntos[i+1])*2*(1-
            cos(alfa(puntos[i],puntos[i+1])/2));
    }
    resistencia=resistencia+resis*suma;
}
```

1.8.4.4.2 Fricción

Para el cálculo de fricción se aplicó la ecuación (5).

```
void Bote::calculo_friccion(double velocidad){

float factor=pow(10,-4);

friccion=(2.9*pow(velocidad,3)+8.03*pow(velocidad,2)+0.01*veloci
dad)*area()*factor;
}
```

Como su nombre lo indica, el método área obtiene el área del recuadro de cada superficie que conforma la embarcación. Se realiza la suma de todas las superficies hasta conseguir el área perimetral total.

```
double Bote::area(){
    double ar=0;
    for(int i=0;i<NCURVAS-1;i++){
        for(int j=0;j<NSUPERFICIES;j++){
            ar=ar+area(curvasD[i]->puntos[j],curvasD[i]-
             >puntos[j+1],curvasD[i+1]-
>puntos[j+1],curvasD[i+1]->puntos[j]);
        }
    }
    return ar;
}
```

1.8.5 Pruebas de unidad

La técnica utilizada para las pruebas del software, es la prueba unitaria, donde se prueba cada uno de los módulos del sistema.

Dependiendo de los resultados obtenidos en las pruebas del software, se realizan las correcciones que sean pertinentes. Después de estas correcciones se vuelve a probar el mismo caso, esperando obtener un resultado satisfactorio.

1.8.5.2 Pruebas del módulo de construcción

Las pruebas del módulo de construcción se describen en la tabla que se muestra a continuación.

Caso de prueba	Resultado	Acción correctiva
Cambiar las medidas de alto, largo y ancho.	se redimensiona el bote correctamente con los nuevos valores de *largo*, *alto* y *ancho*.	Ninguna, funcionamiento correcto.
Guardar dibujo.	Almacena, en alguna dirección dada por el usuario, de manera gráfica un archivo de	Ninguna, funcionamiento

	tipo XML la forma diseñada por el usuario.	correcto.
Cargar archivo de dibujo.	Se carga el archivo señalado por el usuario y este se muestra en todas las vistas.	Ninguna, Funcionamiento correcto.
Nuevo.	Regresan las dimensiones a los valores originales.	Ninguna, funcionamiento correcto.

1.8.5.3 Pruebas al módulo de modelado 3D

Las pruebas al módulo de modelado 3D, se muestran en la tabla siguiente:

Caso de prueba	Resultado	Acción Correctiva
Rotar	El objeto rota correctamente.	Ninguna, funcionamiento correcto.
Zoom	La cámara se acerca y aleja correctamente.	Ninguna, funcionamiento correcto.
Cambiar color	La embarcación, líneas y fondo cambiar de color correctamente.	Ninguna, funcionamiento correcto.
Modificar Dimensiones	Se modifica correctamente la altura, ancho y largo.	Ninguna, funcionamiento correcto.

1.8.5.4 Pruebas al módulo de cálculo

Las pruebas al módulo de cálculo, se describen en la tabla siguiente:

Caso de prueba	Resultado	Acción correctiva
Cálculo de fricción con rapidez 1 m/s.	El resultado de la fuerza de fricción es la que corresponde con las pruebas registradas en la investigación.	Ninguna, funcionamiento correcto.
Cálculo de resistencia con rapidez en 1 m/s.	El resultado de la fuerza de resistencia es la correspondiente con las pruebas registradas en la investigación.	Ninguna, funcionamiento correctamente.

1.9. Evaluación o impacto económico, social o tecnológico

Actualmente el diseño y la fabricación de embarcaciones requieren de métodos costosos, debido a que se ocupan de modelos a escala, e instalaciones especiales. Considerando este hecho, el impacto económico de este trabajo puede ser muy importante, ya que elimina la necesidad de fabricar modelos físicos para realizar pruebas de arrastre, evitando tanto la construcción de estos modelos, así como la construcción y adaptación de instalaciones para realizar tales pruebas, cuyas instalaciones resultan normalmente ser muy grandes y costosas.

El impacto tecnológico es uno de los más importantes, ya que se implementó un modelo matemático innovador propuesto por el Ing. Wolff, para el cálculo de fuerza de resistencia y fricción de una embarcación, combinado con el desarrollo de un nuevo software para el diseño gráfico tridimensional de embarcaciones con formas curvas.

1.10. Resultados obtenidos

Los resultados obtenidos fueron satisfactorios, ya que se obtuvo un software para modelar una embarcación en tres dimensiones, que resultó ser sencillo, amigable y eficiente.

Los cálculos realizados por el software desarrollado corresponden con los valores de las pruebas registradas en la investigación del Ing. Wolff, y con nuevos cálculos realizados manualmente para efecto de la evaluación.

1.11. Conclusiones y recomendaciones

La elaboración de este proyecto de software requirió de muchos de los conocimientos adquiridos a lo largo de la carrera, como lo fueron la programación orientada a objetos, graficación, muchas ramas de la ingeniería de software, como diagramas UML y ciclos de vida del software, los cuales fueron aplicados al proyecto el cual se concluyó satisfactoriamente.

Al final del proyecto se obtuvo un software que cumple con los objetivos propuestos. Además de que las herramientas de lenguaje de modelado unificado y el paradigma orientado a objetos, le dan al sistema la posibilidad de un fácil mantenimiento y crecimiento.

El resultado de los cálculos en el software concordó con las pruebas que se realizaron en la investigación del Ing. Wolff, sin embargo, debido a que no se tienen pruebas físicas, sólo teóricas con los modelos curvos, queda como recomendación realizar los experimentos correspondientes.

Los cálculos generados por el software servirán de apoyo para validar las mediciones que se obtengan en las pruebas de campo.

1.12. Bibliografía

Baker, D. H. (2006). *Gráficos por computadora con OpenGL.*

Booch, G., Jacobson, I., & Rumbaugh, J. (1999). *El lenguaje Unificado de Modelado.* Addison Wesley Iberoamericana.

Wolff Krauter, E. (2006). *Propuesta de cálculo para la fuerza de cuerpos flotantes en el agua.* La Paz.

Pressman, R. S. (2005). *Ingenieria del Software* (Sexta ed.). México : Pearson Education.

Qt. (22 de mayo de 2015). *About Qt*. Obtenido de http://wiki.qt.io/About_Qt

Shreiner, D. (2009). *OpenGL Programming Guide* (Septima ed.). Addison Wesley.

Sommerville, I. (2002). *Ingenieria de Software* (sexta ed.). Mexico : pearson education.

2. Documentos administrativos

DEPENDENCIA: DEPTO. DE SERVS. ESCS.
OFICIO NO.: 152568
CLAVE : 03DIT0002X
ASUNTO : CONSTANCIA

A quien corresponda:

El suscrito, Jefe del Departamento de Servicios Escolares de este Instituto, hace constar que el C. JORGE ALBERTO ARMENTA ATAMOROS con No. de control: 10310451 no esta inscrito en el semestre de la carrera de INGENIERIA EN SISTEMAS COMPUTACIONALES en el periodo comprendido del 17 DE AGOSTO DE 2015 AL 4 DE DICIEMBRE DE 2015
A continuacion se relacionan las calificaciones obtenidas en los siguientes semestres.

MATERIA	CALIFICACION	OPCION
CALCULO DIFERENCIAL	70	ORDINARIO
FUNDAMENTOS DE INVESTIGACION	92	ORDINARIO
FUNDAMENTOS DE PROGRAMACION	80	REGULARIZACION
MATEMATICAS DISCRETAS	70	ORDINARIO
TALLER DE ADMINISTRACION	81	EXTRAORDINARIO
TALLER DE ETICA	82	ORDINARIO
TUTORIA	100	ORDINARIO
CALCULO INTEGRAL	75	ORDINARIO
CONTABILIDAD FINANCIERA	78	EXTRAORDINARIO
PROBABILIDAD Y ESTADISTICA	85	ORDINARIO
PROGRAMACION ORIENTADA A OBJETOS	83	REGULARIZACION
QUIMICA	72	REGULARIZACION
TUTORIA	100	ORDINARIO
ALGEBRA LINEAL	75	EXTRAORDINARIO
CALCULO VECTORIAL	84	REGULARIZACION
CULTURA EMPRESARIAL	78	ORDINARIO
ESTRUCTURA DE DATOS	80	ORDINARIO
FISICA GENERAL	81	ORDINARIO
INVESTIGACION DE OPERACIONES	70	EXTRAORDINARIO
PINTURA	100	ORDINARIO
SISTEMAS OPERATIVOS	80	EXTRAORDINARIO
TUTORIA	100	ORDINARIO
ECUACIONES DIFERENCIALES	80	ORDINARIO
FUNDAMENTOS DE BASE DE DATOS	91	ORDINARIO
METODOS NUMERICOS	88	ORDINARIO
PRINCIPIOS ELECTRICOS Y APLICACIONES DIGITALES	81	REGULARIZACION
TALLER DE SISTEMAS OPERATIVOS	82	ORDINARIO
TOPICOS AVANZADOS DE PROGRAMACION	85	ORDINARIO
ARQUITECTURA DE COMPUTADORAS	83	REGULARIZACION
DESAROLLO SUSTENTABLE	95	ORDINARIO
FUNDAMENTOS DE INGENIERIA DE SOFTWARE	80	REGULARIZACION

FUNDAMENTOS DE TELECOMUNICACIONES	87	EXTRAORDINARIO
PINTURA	100	ORDINARIO
SIMULACION	90	ORDINARIO
TALLER DE BASE DE DATOS	81	REGULARIZACION
ACTIVIDADES COMPLEMENTARIAS	100	ORDINARIO
ADMINISTRACION DE BASE DE DATOS	95	ORDINARIO
GRAFICACION	81	ORDINARIO
INGENIERIA DE SOFTWARE	78	ORDINARIO
LENGUAJES Y AUTOMATAS I	78	REGULARIZACION
REDES DE COMPUTADORAS	90	ORDINARIO
CONMUTACION Y ENRUTAMIENTO DE REDES DE DATOS	94	REGULARIZACION
GESTION DE PROYECTOS DE SOFTWARE	74	ORDINARIO
LENGUAJES Y AUTOMATAS II	82	REGULARIZACION
SISTEMAS PROGRAMABLES	95	REGULARIZACION
TALLER DE INVESTIGACION I	88	ORDINARIO
ADMINISTRACION DE REDES	79	ORDINARIO
AUDITORIA INFORMATICA	83	
LENGUAJES DE INTERFAZ	96	REPETICION
PROGRAMACION LOGICA Y FUNCIONAL	80	ORDINARIO
PROGRAMACION WEB	86	REGULARIZACION
SEGURIDAD EN REDES	80	REGULARIZACION
SERVICIO SOCIAL	100	ORDINARIO
TALLER DE INVESTIGACION II	89	ORDINARIO
ALGORITMOS HEURISTICOS	97	ORDINARIO
INTELIGENCIA ARTIFICIAL	86	ORDINARIO
PROGRAMACION DE DISPOSITIVOS MOVILES	88	REGULARIZACION
PROGRAMACION PARALELA	91	ORDINARIO
RESIDENCIA PROFESIONAL	100	ORDINARIO

Promedio de los semestres listados: 83,94 Avance: 100%

Creditos Aprobados: 260 de un total de 260

A petición del interesado y para los fines legales que mejor le convengan, se extiende la presente en la ciudad de La Paz, B. C. S. a los 19 días del mes de agosto del año 2015

A T E N T A M E N T E

LIC. MARIA JESUS VILLEGAS BARBA
JEFE DEL DEPARTAMENTO DE SERVICIOS ESCOLARES

SECRETARÍA DE EDUCACIÓN PÚBLICA
TECNOLÓGICO NACIONAL DE MÉXICO
INSTITUTO TECNOLÓGICO DE LA PAZ
SERVICIOS ESCOLARES

SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO
Instituto Tecnológico de La Paz

"2015, Año del Generalísimo José María Morelos y Pavón"

INSTITUTO TECNOLÓGICO DE LA PAZ
SISTEMAS Y COMPUTACIÓN
17 FEB 2015
RECIBIDO

La Paz, B.C.S. a 09/FEBRERO/2015

Departamento: GESTION TEC. Y VINC.
No. de Oficio: DGTV/131/2015.

ASUNTO: PRESENTACIÓN DEL ESTUDIANTE Y AGRADECIMIENTO.

C. ING. JESUS DAVID ESTRADA RUÍZ,
DIRECTOR DEL INSTITUTO TECNOLOGICO DE LA PAZ,
PRESENTE

El Instituto Tecnológico de La Paz, tiene a bien presentar a sus finas atenciones al (la) C. **JORGE ALBERTO ARMENTA ATAMOROS** con número de control **10310451** de la carrera de **INGENIERÍA EN SISTEMAS COMPUTACIONALES**, quien desea desarrollar el proyecto de Residencias Profesionales, denominado **"DESARROLLO DE UN SOFTWARE PARA CÁLCULO DE RESISTENCIA DE EMBARCACIONES"** cubriendo un total de **500** horas, en un período de cuatro a seis meses.

Es importante hacer de su conocimiento que todos los estudiantes que se encuentran inscritos en esta institución cuentan con un seguro contra accidentes personales con la empresa Royal & Sunalliance Seguros México, S.A. de C.V., según póliza No. 0000005560 e inscripción en el IMSS.

Así mismo, hacemos patente nuestro sincero agradecimiento por su buena disposición y colaboración para que nuestros estudiantes, aún estando en proceso de formación, desarrollen un proyecto de trabajo profesional, donde puedan aplicar el conocimiento y el trabajo en el campo de acción en el que se desenvolverán como futuros profesionistas.

Al vernos favorecidos con su participación en nuestro objetivo, sólo nos resta manifestarle la seguridad de nuestra más atenta y distinguida consideración.

A T E N T A M E N T E
"Ciencia es Verdad, Técnica es Libertad"

LIC. MIRENCHU BENILDE ALCEDO MARQUEZ,
JEFA DEL DEPARTAMENTO DE GESTIÓN TECNOLÓGICA Y VINCULACIÓN

TECNOLÓGICO NACIONAL DE MÉXICO
INSTITUTO TECNOLÓGICO DE LA PAZ
GESTIÓN TECNOLÓGICA Y VINCULACIÓN

Jorge Armenta
17/marzo/2015

MBAM'asr*

ACUERDO DE TRABAJO QUE CELEBRAN POR UNA PARTE EL **INSTITUTO TECNOLÓGICO DE LA PAZ,** REPRESENTADO POR LA **SUBDIRECCIÓN DE PLANEACIÓN Y VINCULACIÓN** POR ACUERDO DEL DIRECTOR **ING. JESÚS DAVID ESTRADA RUÍZ,** POR LA OTRA, EL (LA) **C. JORGE ALBERTO ARMENTA ATAMOROS,** A QUIENES EN LO SUCESIVO SE LES DENOMINARÁ **"EL TECNOLÓGICO", Y "EL ESTUDIANTE"** RESPECTIVAMENTE, SUJETÁNDOSE A LAS SIGUIENTES DECLARACIONES Y CLÁUSULAS.

I.- DECLARACIONES:

Declara: **"EL TECNOLÓGICO"**

1.1 Ser una institución dedicada a impartir educación superior tecnológica de acuerdo a las necesidades que demanden los sectores productivos regionales y nacionales, adscrita a la Dirección General Educación Superior Tecnológica dependiente de la Secretaría de Educación Pública y dotado de plena capacidad para suscribir el presente convenio de colaboración.

1.2 Que derivado de la Reforma de la Educación Superior Tecnológica, se incorporó a los planes de estudio de las carreras que se imparten en el Sistema Nacional de Educación Superior Tecnológica, el programa de Residencias Profesionales, y dado el carácter curricular de este programa se hace necesario fortalecer la vinculación entre la educación superior y los sectores productivos de bienes y servicios.

1.3 Que tiene establecido su domicilio en Boulevard Forjadores de Baja California Sur No. 4720, en la Ciudad de La Paz, B.C.S. y que su R.F.C. es SEP-210905-778.

1.4 Que su misión es: Ser una Institución de Educación Superior Tecnológica de calidad, que forme profesionistas competentes y con valores, que se integren activamente en el desarrollo sustentable, económico social y cultural de nuestro País y de la humanidad.

Declara **"EL ESTUDIANTE"**

1.5 Ser estudiante (a) regular de **"EL TECNOLÓGICO"** inscrito en la carrera de **INGENIERÍA EN SISTEMAS COMPUTACIONALES,** con número de control **10310451.**

1.6 Haber acreditado como mínimo el **80%** del total de los créditos del Plan de Estudios de la carrera a que se hace referencia en la declaración 1.5.

1.7 Que desea realizar su Residencia Profesional en el "**INSTITUTO TECNOLÓGICO DE LA PAZ".**

1.8 Que el proyecto denominado **"DESARROLLO DE UN SOFTWARE PARA CÁLCULO DE RESISTENCIA DE EMBARCACIONES"** es producto de la investigación preliminar realizada y que previo aval de la Academia de **INGENIERÍA EN SISTEMAS COMPUTACIONALES,** ha sido autorizado para su desarrollo por el Departamento de **SISTEMAS COMPUTACIONALES.**

II.- CLÁUSULAS:

PRIMERA. "EL TECNOLÓGICO", se compromete a brindar todas las facilidades necesarias tanto al asesor del proyecto como a "**EL ESTUDIANTE**" para el desarrollo del proyecto de Residencias Profesionales.

SEGUNDA. "EL ESTUDIANTE" se compromete a cumplir en tiempo y forma con todas las condiciones, características y requisitos que se mencionan en el proyecto a que se hace referencia en la declaración 1.8 y que es parte integrante del presente Acuerdo de Trabajo.

TERCERA. "**EL ESTUDIANTE**" se compromete a sujetarse al manual de procedimientos para la planeación, operación y acreditación de las Residencias Profesionales vigentes.

CUARTA. "EL ESTUDIANTE" cuenta con un seguro contra accidentes que pudieran ocurrir con motivo del desarrollo del proyecto que se menciona en la declaración 1.8 para lo cual "**EL TECNOLOGICO"** a través de la Dirección General de Educación Superior Tecnológica, tiene contratado seguro contra accidentes personales con la empresa Royal & Sunalliance Seguros México, S.A. de C.V., con la póliza No. 0000005560.

Así mismo "**EL TECNOLÓGICO**" se compromete a verificar que "**EL ESTUDIANTE**" tenga actualizado sus servicios médicos.

QUINTA. Con motivo del desarrollo de algún proyecto de Residencias Profesionales y debido al carácter académico que tienen "**EL ESTUDIANTE**" acepta que su participación no originará ninguna relación laboral ni de seguridad social con "**EL TECNOLÓGICO**"

SEXTA. Los apoyos que **"EL TECNOLÓGICO"** pueda otorgar a **"EL ESTUDIANTE"** serán de acuerdo a su normatividad y disponibilidad presupuestal.

SÉPTIMA. "EL ESTUDIANTE" se compromete con **"EL TECNOLÓGICO"** a resguardar la confidencialidad de la información que surja con motivo de la implementación del presente convenio y que se aplique única y exclusivamente al desarrollo del proyecto a que se refiere la declaración 1.8.

Leído que fue el presente Acuerdo y enteradas las partes de su contenido, alcance y fuerza legal, se firma al margen y al calce en la ciudad de La Paz, Baja California Sur a los **Nueve días del mes de Febrero del Año Dos mil Quince**.

POR EL TECNOLÓGICO

M.C. ANTONIO HERNÁNDEZ SÁNCHEZ
SUBDIRECTOR DE PLANEACIÓN Y VINCULACIÓN

EL ASESOR EXTERNO

ING. EBERHARD WOLFF KRAUTTER
RESPONSABLE TÉCNICO

EL ESTUDIANTE

Jorge Armenta

JORGE ALBERTO ARMENTA ATAMOROS
No. DE CONTROL 10310451
INGENIERÍA EN SISTEMAS COMPUTACIONALES

	Nombre del documento: Evaluación del proyecto de residencias profesionales	Código:ITLP-AC-PO-007-07
		Revisión: 12
	Referencia a la Norma ISO 9001:2008 7.5.1	Página 1 de 1

EVALUACIÓN DEL PROYECTO DE RESIDENCIAS PROFESIONALES

NOMBRE DEL RESIDENTE Jorge Alberto Armenta Atamoros
CARRERA: Ingeniería en Sistemas Computacionales No. DE CONTROL 10310451
NOMBRE DE LA EMPRESA: Instituto Tecnológico de La Paz
DEPARTAMENTO ASIGNADO: Departamento de Sistemas y Computación.
PERÍODO DE LA RESIDENCIA: DEL 26 DE Enero AL 7 DE Agosto DEL 2015
FECHA DE EVALUACIÓN: 7 de Agosto del 2015
INSTRUCCIONES: Marcar con una X para cada criterio de evaluación, su percepción sobre el desempeño del estudiante residente en el cumplimiento de los objetivos de su proyecto.

CRITERIO DE EVALUACIÓN	D	A	B	E
CUMPLIMIENTO DE LOS OBJETIVOS				X
CALIDAD EN SUS ACTIVIDADES			X	
DISCIPLINA				X
COOPERACIÓN			X	
RESPONSABILIDAD				X
INICIATIVA			X	
SOCIABILIDAD				X

CÓDIGO
D: DEFICIENTE
A: ACEPTABLE
B: BUENO
E: EXCELENTE

VALUACION (ESCALA 0 A 100 %) 100

COMENTARIOS:

Eberhard Wolff Krauter

ASESOR EXTERNO (NOMBRE Y FIRMA)

ESTADOS UNIDOS MEXICANOS
SECRETARÍA DE EDUCACIÓN PÚBLICA
TECNOLÓGICO NACIONAL DE MÉXICO
INSTITUTO TECNOLÓGICO DE LA PAZ
DIRECCIÓN

Sello de la Empresa

ORIGINAL: DEPARTAMENTO ACADÉMICO
COPIA: ASESOR INTERNO.
COPIA: DEPTO. DE GESTIÓN TECN. Y VINCULACIÓN.

EVALUACIÓN INTERNA DEL PROYECTO DE RESIDENCIAS PROFESIONALES

Esta evaluación es realizada por el Asesor Interno

NOMBRE DEL RESIDENTE: Jorge Alberto Armenta Atamoros
CARRERA: Ingeniería en Sistemas Computacionales No. De control 10310451
NOMBRE DE LA EMPRESA: Instituto Tecnológico de La Paz
NOMBRE DEL PROYECTO: Desarrollo De Un Software Para Cálculo De Resistencia De Embarcaciones
DEPARTAMENTO ASIGNADO: Departamento de Sistemas y Computación
PERIODO DE LA RESIDENCIA: Del: 26 de enero al 7 de agosto del 2015
FECHA DE LA EVALUACIÓN: 7 de agosto del 2015

INSTRUCCIONES: Marcar con una X para cada criterio de evaluación, su percepción sobre el desempeño del estudiante residente en el cumplimiento de los objetivos de su proyecto.

CRITERIO DE EVALUACIÓN	D	A	B	E
CUMPLIMIENTO DE LOS OBJETIVOS				X
CALIDAD EN SUS ACTIVIDADES			X	
DISCIPLINA				X
COOPERACIÓN			X	
RESPONSABILIDAD				X
INICIATIVA			X	
SOCIABILIDAD				X

CÓDIGO
D: DEFICIENTE
A: ACEPTABLE
B: BUENO
E: EXCELENTE

EVALUACION (ESCALA 0 A 100 %) 100

COMENTARIOS:

Jorge Enrique Luna Taylor

ASESOR INTERNO (NOMBRE Y FIRMA)

ESTADOS UNIDOS MEXICANOS
SECRETARÍA DE EDUCACIÓN PÚBLICA
TECNOLÓGICO NACIONAL DE MÉXICO
INSTITUTO TECNOLÓGICO DE LA PAZ
DIRECCIÓN

ORIGINAL: DEPARTAMENTO ACADÉMICO
COPIA: ASESOR INTERNO.
COPIA: DEPTO. DE GESTIÓN TECN. Y VINCULACIÓN.