INFRAESTRUCTURA DE SIMULACIÓN PARA VEHÍCULOS NO TRIPULADOS

Carlos Alejandro Zuluaga Toro, Luís Benigno Gutiérrez

Universidad Pontificia Bolivariana, Medellín, Colombia
Circular 1 #70-01
caz@upb.edu.co

Abstract: El propósito de este artículo es mostrar el proceso de diseño e implementación de una infraestructura de simulación para sistemas de control de vehículos no tripulados. La infraestructura permite la implementación de modelos de simulación como *software in the loop and hardware in the loop simulations* para verificar la correcta operación de la arquitectura de control propuesta.

Keywords: infraestructuras de simulación, *software in the loop simulation*, *hardware in the loop simulation*, vehículos no tripulados.

1. INTRODUCCIÓN.

En el estudio del desarrollo de vehículos no tripulados existen áreas de especial interés como la verificación de modelos y pruebas sobre los algoritmos de control. Para cumplir con estas demandas se requiere de programas especializados en los que se puedan implementar las ecuaciones completas que modelan el vehículo, los algoritmos de control y cada uno de los elementos necesarios en la arquitectura de control básica. Adicionalmente la infraestructura debe contar con herramientas de comunicación acordes con la aplicación.

En este artículo se pretende mostrar el proceso de diseño e implementación de una infraestructura que permite de forma fácil e intuitiva la implementación de un modelo de simulación básico. Lo que hace interesante esta infraestructura es la flexibilidad para conectar los elementos que conforman el modelo de simulación. En general el modelo se puede ejecutar en una misma máquina y en un mismo proceso, o en diferentes procesos y una misma máquina, o en diferentes máquinas. Las conexiones remotas (entre máquinas) se realizan usando ethernet.

2. ANTECEDENTES.

Para enmarcar este proyecto hay que hacer un poco de historia que inicia a finales del año de 1993 cuando un grupo de profesores y estudiantes de la Universidad Pontificia Bolivariana se reunieron estimulados por el Dr Reynolds, y decidieron construir una plataforma sumergible para la investigación subacuática, dicha plataforma sirvió de antesala a la propuesta de diseñar y construir un minisubmarino con mas prestaciones que la plataforma ya implementada.

Así se dio inicio al proyecto VISOR presentado a la UPB en 1997, hasta llegar al prototipo actual, el minisubmarino VISOR. El proyecto VISOR tenía como objetivo principal el diseño e implementación de un vehículo subacuático para la exploración subacuática y que además tuviera una forma hidrodinámica para reducir las fuerzas de arrastre en el agua. VISOR fue desarrollado entre los años 1998 y 2000. Mas adelante, en enero de 2006 el grupo de investigación A+D de la Universidad Pontificia Bolivariana en conjunto con el grupo de Investigación en Ingeniería Naval de la Escuela Naval Admirante Padilla inician el proyecto titulado "Desarrollo de un vehículo sumergible operado remotamente ROV, para la inspección subacuática", con el apoyo de Colciencias. En este proyecto se pretende desarrollar no sólo un prototipo si no también, crear un producto comercializable en el futuro.

Una de las falencias del proyecto VISOR fue la falta de una infraestructura de simulación adecuada para realizar las pruebas de comportamiento del sistema y así realizar los ajustes y correcciones necesarias sobre el vehículo en tiempo de diseño. Esto obliga a buscar una solución que dé la posibilidad de probar los algoritmos de control sin necesidad de tener el prototipo operando. Sistemas como este disminuyen tiempos de ejecución y los costos del proyecto.

En la actualidad se usan diferentes infraestructuras desarrolladas para ser usadas con vehículos no tripulados, pero una de las más reconocidas es el modelo OCP (*Open Control Platform*). OCP es una infraestructura de *software* basada en CORBA orientada a objetos que permite la integración de componentes de *software*, habilitando la implementación de sistemas de control avanzado.

Está diseñada para facilitar la implementación y pruebas de los algoritmos del control en una infraestructura de cómputo distribuido. Actualmente, OCP se enfoca especialmente en modelos de vehículos aéreos no tripulados (UAVs). (Paunicka, J., Mendel, B., and Corman, D., 2001; Wills L., et al, 2001).

Adicionalmente, OCP fue la columna vertebral del programa de investigación SEC (*Software Enabled Control*) financiado por DARPA (*Defense Advanced Research Projects Agency* de los E.E.U.U). La meta del programa SEC era desarrollar nuevas tecnologías de *software* que habilitaran la implementación de algoritmos de control avanzado en vehículos aéreos no tripulados. OCP está siendo desarrollado por el *Embedded Systems Research Team* de la organización *Boeing Phantom Works Open Systems* con el apoyo de Georgia Institute of Technology, los laboratorios de Honeywell y la Universidad de California en Berkeley. (Paunicka, Mendel and Corman, 2001).

Antes de hacer pruebas reales en un vehículo deben realizarse dos tipos de simulación que facilitan la depuración final del código: Simulación con software en el lazo de control (*Software-In-The-Loop Simulation*) *y* con hardware en lazo de control (*Hardware-In-The-Loop Simulation*). La primera permite simular todo el *software* sobre uno o varios procesadores sin utilizar el *hardware* real, mientras que la segunda permite simular el *software* interactuando con el *hardware* real. (Pruett, H., Slutz, G., Paunicka J. and Portilla, E., 2003).

3. ARQUITECTURA DE CONTROL

El sistema de control básico para un vehículo no tripulado mostrado en la figura 1, cuenta con varios bloques que serán tratados de aquí en adelante como componentes. Un componente es una abstracción de un módulo u objeto creado a partir de código y que define el funcionamiento de un elemento específico.

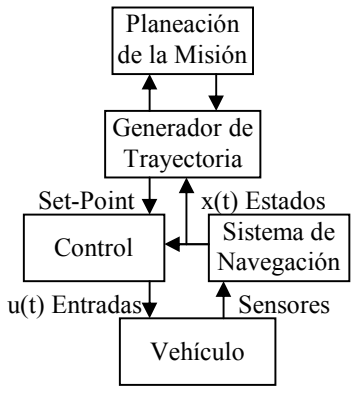

Fig. 1. Diagrama de bloque que muestra la arquitectura de control usada.

El componente de planeación de misión permite convertir la información de la misión que se quiere desarrollar en un conjunto de puntos de vía (*way-*

points) que luego son utilizados por el componente generador de trayectoria para producir los puntos de consigna (*set-points*)requeridos por el control de bajo nivel.

El sistema de navegación permite estimar el estado del vehículo para que el componente de control pueda calcular las señales de los actuadores basado en los puntos de consigna suministrados por el generador de trayectoria. El componente vehículo incluye un modelo matemático que simula su comportamiento dinámico.

Para la simulación del vehículo subacuático operado remotamente (ROV), se requiere de una arquitectura de control conformada sólo por tres componentes: el modelo, el control y el sistema de navegación. Aunque el vehículo es operado remotamente el operario no manipula directamente los actuadores. Lo que realmente ocurre es que desde una palanca de control en la estación de superficie se generan los puntos de consigna del controlador de modo que el vehículo se desplace con una orientación y velocidad determinada. Por esta razón, en este caso no se requiere de los componentes "generador de trayectorias" y "planeación de misión". Sin embargo, esta infraestructura está pensada para que pueda implementarse la cantidad de componentes que se necesite dependiendo de la aplicación.

4. FILOSOFÍA DE LA INFRAESTRUCTURA

La infraestructura debe proveer la funcionalidad básica de los componentes y permitir la interconexión entre los mismos cumpliendo con las condiciones mencionadas a continuación. La primera condición es que la implementación debe ser lo mas intuitiva posible. Debe ser portable y, aunque no necesariamente se ejecuta sobre un sistema operativo en tiempo real, debe facilitar la migración a un sistema operativo con estas características. Las conexiones y tiempos de ejecución deben ser configurables. Por último, la más importante es que se debe garantizar la estabilidad de la ejecución de los componentes especialmente cuando se están usando conexiones remotas.

4.1. Implementación

La implementación de una simulación sólo requiere que el usuario inserte el código que le da la funcionabilidad a cada componente y defina en el programa principal el orden de ejecución de las funciones de los componentes, los tiempos de ejecución de los mismos, y la forma como se van a interconectar los diferentes componentes. En conclusión el usuario no tiene que ser un experto del lenguaje de programación. Más adelante se detalla el procedimiento seguido para implementar un modelo de simulación sencillo.

4.2. Portabilidad

La portabilidad del la infraestructura se garantiza ya que sólo se utilizan librerías ANSI y POSIX. Adicionalmente cuando se decida migrar la infraestructura a un sistema operativo en tiempo real, no debe existir ningún tipo de incompatibilidad ya que las librerías usadas y la forma como está implementado el código no entra en conflicto con la filosofía de dichos sistemas operativos (Canosa, J. M., 2000).

4.3. Configuración.

Los componentes se conectan usando puertos, los cuales abstraen el mecanismo de interconexión. Dependiendo de la aplicación, en cada componente se pueden configurar puertos remotos o puertos locales con respecto a los procesos donde se implementan.

Los puertos locales se basan en un simple intercambio de estructuras de datos entre los componentes, cuando los componentes interconectados se están ejecutando en el mismo proceso.

Los puertos remotos permiten conectar componentes que se ejecutan en diferentes procesos. Los procesos que implementan a los componentes se pueden ejecutar en uno o varios procesadores. Estos puertos realizan la conexión usando sockets. La infraestructura cuenta con la posibilidad de cambiar los tiempos en que se ejecutan los diferentes componentes, en el apartado 7 se explica la forma como se manejan estos tiempos de ejecución.

4.4 Estabilidad de ejecución.

En los casos en que se requiere realizar conexiones

remotas, esta aplicación usa puertos UDP sobre una red Ethernet. Una característica especial de los sockets UPD es que las funciones usadas no generan bloqueos de ejecución y por tal motivo los componentes remotos quedan desacoplados, es decir en términos de estabilidad, la ejecución de uno no depende de la del otro. En el apartado 6 se profundiza más sobre el funcionamiento de las conexiones remotas.

5. ESTRUCTURA DE LOS COMPONENTES

Cada uno de los componentes que lleva el modelo de simulación cuenta con una estructura de datos conformada por las entradas, las salidas, los estados, los parámetros y las constantes. La estructura de las entradas, las salidas, los estados, los parámetros y las constantes debe ser definida para cada componente pero no tiene restricciones. Adicionalmente, cada componente cuenta con un archivo de macros que facilita la el análisis del código. El uso de este archivo es opcional, simplemente está hecho para facilitar el trabajo de implementación. Además de la estructura de datos, cada componente tiene seis funciones, de las cuales dos son opcionales. Estas funciones se describen a continuación.

5.1. Función de inicialización.

Esta función sólo se ejecuta una vez y busca dar un valor inicial a cada elemento de la estructura de datos del componente. Este proceso se puede realizar de tres formas. Una forma es usando parámetros por defecto definidos en el código fuente y que luego de compilar el programa no se pueden cambiar. Otra forma es usar valores por defecto para algunos de las variables y el resto de variables se inicializan usando los argumentos del programa a la hora de ejecutarse, en este caso hay que tener en cuenta la cantidad y el formato de los argumentos. Si se presenta una incompatibilidad el programa no los tiene en cuenta y la inicialización se hace por defecto. La tercera forma es usando un archivo de texto organizado de tal forma que facilita identificar el sitio correspondiente para cada variable, este archivo se puede manipular con cualquier editor de texto.

5.2. Función para actualizar entradas.

Con esta función se toman los valores de cada entrada que se encuentran disponibles en el buffer de entrada. El buffer de entrada del componente es un espacio de memoria idéntico al de las entradas, pero que se actualiza a una rata diferente.

5.3. Función para calcular salidas.

En esta función se encuentran las ecuaciones para calcular las salidas de cada componente, estas salidas se almacenan en un buffer de salidas una vez que se han calculado.

5.4. Función para calcular los estados.

Esta función calcula los estados siguientes del sistema, guardándolos para usarlos en el siguiente periodo de ejecución.

5.5. Función para guardar.

Función encargada para almacenar los datos que se generas durante la simulación.

5.6. Función de inicialización.

Esta función limpia la información almacenada en los archivos en la simulación anterior.

6. LOS PUERTOS

Para realizar la conexión entre los componentes, se definen puertos de entrada y de salida. Cuando se usan las conexiones locales la operación de estos puertos es trivial ya que sólo deben pasar información de un buffer a otro.

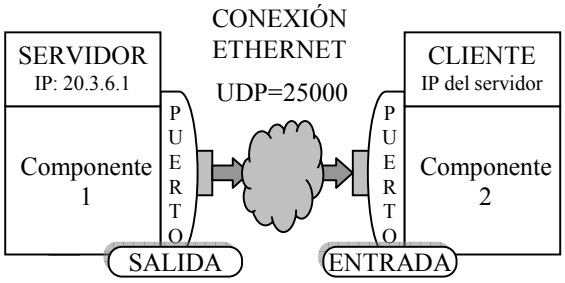

Fig. 2 Conexión cliente servidor entre componentes remotos.

En los casos en que las conexiones son remotas se crea un cliente en la entrada y un servidor en la salida

de dicha conexión. La figura 2 muestra gráficamente esta conexión.

El cliente remoto en el puerto de entrada debe tener la información del puerto UDP que se usará en la conexión y adicionalmente la dirección IP del servidor a que se va a conectar. El servidor remoto en el puerto de salida sólo requiere el número del puerto UDP. El puerto de salida envía constantemente datos a una rata configurable.

7. PROGRAMADOR DE TAREAS

Una de los requerimientos de la infraestructura es que los tiempos de ejecución de los componentes debe ser configurable. Esta necesidad trae consigo muchas implicaciones, donde el peor de los casos es cuando se ejecutan todos los componentes en un mismo proceso y cada uno de ellos a una rata diferente. Adicionalmente los puertos de entrada también se ejecutan de forma independiente y a ratas diferentes para garantizar disponibilidad en la información.

Los tiempos de ejecución de los puertos deben ser menores o iguales que los tiempos de los componentes. Para solucionar el reto planteado se hace uso de hilos POSIX y se sigue la siguiente regla: cada puerto de entrada y cada componente corren en hilos independientes.

Con un sistema multihilo ejecutándose en el mismo proceso hay que tener cuidado con la lectura y escritura de la memoria compartida, para evitar este tipo de conflictos se usan semáforos (Stones, R. and Matthew, N. 1999).

8. IMPLEMENTACIÓN DE UN SISTEMA DE SIMULACIÓN

Uno de los procedimientos que se pueden realizar con esta infraestructura es la ejecución de componentes en máquinas independientes. El siguiente es un caso típico para probar los algoritmos de control para un vehículo no tripulado.
Se implementará una simulación de tres componentes, el primero es el modelo de un submarino, el segundo es el sistema de control y el tercero es el sistema de navegación.
El modelo se ejecuta en el computador 1 representando la dinámica simulada del vehículo y los modelos de los sensores usados. El control y el sistema de navegación se ejecutan en el Computador 2 representando el procesador a bordo del vehículo (Figura 3).

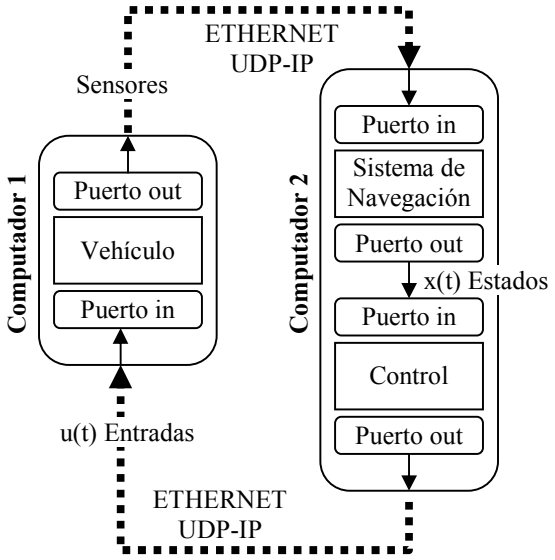

Fig. 3 Implementación del sistema de simulación.

Para construir la simulación se siguen los siguientes pasos:
- En el computador 1 se crea la estructura de archivos necesarios, que son: el código fuente principal que define la infraestructura del proceso que corre en el computador 1, el código fuente para el componente del modelo, el archivo de inicialización del modelo, el archivo de definiciones del modelo.
- Se insertan las ecuaciones del modelo.
- Se define el tiempo de ejecución del modelo.
- Se define el puerto de salida del modelo como remoto. Como el puerto es de salida el programa crea un socket servidor en el puerto UDP definido.
- Se define el puerto de entrada del modelo como remoto. El programa crea un socket cliente en el puerto UDP definido, esta dirección debe ser diferente que la del puerto de salida, ya que son conexiones independientes. Este puerto requiere la dirección IP del servidor a que se va a conectar.
- Se define el tiempo de actualización del puerto de entrada.
- En el computador 2 se crea la estructura de archivos, pero en este caso se crean los archivos necesarios para dos componentes.

- Se insertan las ecuaciones que definen a los componentes (control y sistema de navegación).
- Se define el puerto de entrada al componente de control como local.
- Se define el puerto de salida del componte de control como remoto y se le da el número del puerto UDP, recuerde que el puerto queda automáticamente definido como servidor.
- Se define el puerto de entrada del componente de sistema de navegación como remoto y se le da el número del puerto UDP y la dirección IP del servidor (la del computador 1).
- Se define el puerto de salida del componente de sistema de navegación como local.
- Se definen el orden y los tiempos de ejecución de cada componente.
- Se definen los tiempos de actualización de los puertos. Se recomienda, aunque no es necesario, que se usen los mismos tiempos para los puertos del computador 1.

Finalmente se ejecuta el programa en cada procesador e inmediatamente el proceso de intercambio de información se inicia. No importa cual de los dos programas se ejecuta primero, simplemente el intercambio de la información se hace de acuerdo a la disponibilidad de la misma. De manera que si se ejecuta sólo el sistema de control y navegación, el algoritmo se continúa ejecutando aunque las salidas se saturen, pero la ejecución de la simulación se garantiza. Algo parecido sucede si sólo se ejecuta el modelo.

9. CONCLUSIONES

La infraestructura de software presentada en este artículo permite implementar y simular sistemas de control que requieren la validación de los modelos de los vehículos y los algoritmos de control y navegación usando el mismo software que correrá en la aplicación final. Las simulaciones se pueden hacer con software en el lazo de control (*Software-In–The-Loop Simulation*) o con hardware en el lazo de control (*Hardware-In-The-Loop Simulation*).

Esta infraestructura podrá ser usada para simular sistemas de control en otras aplicaciones a pesar de haber sido desarrollada para vehículos no tripulados.
El desarrollo de esta infraestructura continua y uno de las metas a seguir en la adecuación del programa para que pueda ser ejecutado en sistemas operativos en tiempo real como es el caso de RTLinux.

REFERENCIAS

Bhattacharya, R. and Balas, G. J. (2004). Implementation of Online Control Customization within the Open Control Platform. In: *Software Enabled Control: Information Technologies for Dynamical Systems, IEEE Press Publication.*

Canosa, J. M. (2000). *Programación avazada en UNIX*. McGraw-Hill, Madrid.

Hassani, M. and Stewart, D. A. (1997).Mechanism for Communicating in Dynamically Reconfigurable Embedded Systems. In: *Proc. of High AssuranceSystems Engineering Workshop*.

Liu, C.L. and Layland, J.W. (1973) Scheduling Algorithms for Multiprogramming in a Hard Real Time Environment. In: *Journal of the ACM.*

Paunicka, J., Mendel, B., and Corman, D., (2001). The OCP – An Open Middleware Solution for Embedded Systems. In: *American Control Conference.*

Pruett, H., Slutz, G., Paunicka J. and Portilla, E., (2003). Hardware-in-the-loop simulation using open control platform. In: *AIAA Modeling and Simulation Technologies Conference and Exhibit*.

Stewart, D. B. and Arora, G. (1996). Dynamically reconfigurable embedded software - does it make sense?. In: *Proc. of IEEE Real-Time Applications Workshop (RTAW)*.

Stones, R. and Matthew, N. (1999). *Beginning Linux programming*. Wrox Press Ltd, New York.

Wills L., et al, (2001). An open platform for reconfigurable control. In: *IEEE Control Systems Magazine*.