

Definición de una infraestructura cloud de alta disponibilidad en un entorno distribuido para el
despliegue de una plataforma IoT

Jose David Rojas Aguilar

Trabajo de Grado para optar al título de Ingeniero de Sistemas

Director

Gabriel Rodrigo Pedraza Ferreira

Doctorado en Ciencias de la computación

Universidad Industrial de Santander

Facultad de Ingenierías Fisicomecánicas

Escuela de Ingeniería de sistemas e informática

Bucaramanga

2019



UNIVERSIDAD INDUSTRIAL DE SANTANDER
SISTEMA DE TRABAJOS DE GRADO
ADMINISTRACIÓN DE TRABAJOS DE GRADO

Fecha Impresión:

08 junio 2018

Pág 1 de 1

Codigo:	17141	Fecha Presentacion:	31-may-2018
Titulo: VERIFICACIÓN DE CONJUNTOS ESTABILIZANTES PARA EL MÉTODO DE DISEÑO DE CONTROLADORES PI DE ZIEGLER & NICHOLS			
Nota Proyecto:	4.2	Fecha Registro Nota:	31-may-2018
Estado:	APROBADO		
Tipo Trabajo:	INVESTIGACION		
Estudiantes			
Código	Nombre	Programa Académico	
2112244	REY ARDILA EMERSON	26-INGENIERIA ELECTRONICA	
Directores			
Documento	Nombre	Clase	Firma
C-6383936	RICARDO ALZATE CASTAÑO	DIRECTOR	
Calificadores			
Documento	Nombre	Firma	
C-13513276	DANIEL ALFONSO SIERRA BUENO		
C-13513494	FRANKLIN ALEXANDER SEPULVEDA SEPULVEDA		



**ENTREGA DE TRABAJOS DE GRADO, TRABAJOS
DE INVESTIGACION O TESIS Y AUTORIZACIÓN
DE SU USO A FAVOR DE LA UIS**

Yo, EMERSON REY ARDILA, mayor de edad, vecino de Bucaramanga, identificado con la Cédula de Ciudadanía No.1.095.927.246 de Girón, actuando en nombre propio, en mi calidad de autor del trabajo de grado, del trabajo de investigación, o de la tesis denominada(o):

**VERIFICACIÓN DE CONJUNTOS ESTABILIZANTES PARA EL MÉTODO DE DISEÑO
DE CONTROLADORES PI DE ZIEGLER & NICHOLS.**

hago entrega del ejemplar respectivo y de sus anexos de ser el caso, en formato digital o electrónico (CD o DVD) y autorizo a LA UNIVERSIDAD INDUSTRIAL DE SANTANDER, para que en los términos establecidos en la Ley 23 de 1982, Ley 44 de 1993, decisión Andina 351 de 1993, Decreto 460 de 1995 y demás normas generales sobre la materia, utilice y use en todas sus formas, los derechos patrimoniales de reproducción, comunicación pública, transformación y distribución (alquiler, préstamo público e importación) que me corresponden como creador de la obra objeto del presente documento. PARÁGRAFO: La presente autorización se hace extensiva no sólo a las facultades y derechos de uso sobre la obra en formato o soporte material, sino también para formato virtual, electrónico, digital, óptico, uso en red, Internet, extranet, intranet, etc., y en general para cualquier formato conocido o por conocer.

EL AUTOR – ESTUDIANTE, manifiesta que la obra objeto de la presente autorización es original y la realizó sin violar o usurpar derechos de autor de terceros, por lo tanto la obra es de su exclusiva autoría y detenta la titularidad sobre la misma. PARÁGRAFO: En caso de presentarse cualquier reclamación o acción por parte de un tercero en cuanto a los derechos de autor sobre la obra en cuestión, EL AUTOR / ESTUDIANTE, asumirá toda la responsabilidad, y saldrá en defensa de los derechos aquí autorizados; para todos los efectos la Universidad actúa como un tercero de buena fe.

Para constancia se firma el presente documento en dos (02) ejemplares del mismo valor y tenor, en Bucaramanga, a los 12 días del mes de junio de Dos Mil dieciocho 2018.

EL AUTOR /ESTUDIANTE:

Emerson Rey A.
Emerson Rey Ardila

Dedicatoria

Este trabajo viene dedicado para todas aquellas personas que apoyaron el desarrollo y ejecución de este trabajo de grado.

En especial reconozco la permanente presencia de Dios en mi camino de vida.

Agradecimientos

Agradezco a mi familia por el apoyo económico y moral que tuvieron para conmigo durante el desarrollo de mi carrera. También agradezco a mis amigos y compañeros por las vivencias de estos inolvidables años de universidad.

Un reconocimiento y agradecimiento importante lo realizo a mi director de trabajo de grado, por dedicar su tiempo, experiencia y conocimiento en la guía de mi proyecto.

Tabla de Contenido

Introducción	15
1. Objetivos	16
2. Estado del arte	16
3. Marco de referencia	18
3.1. Computación en la nube	18
3.1.1. Características esenciales	19
3.1.2. Modelos de servicio	20
3.1.3. Modelos de despliegue	22
3.2. Alta disponibilidad	23
3.2.1. Escalabilidad	25
3.3. Virtualización	27
3.4. Contenedores	29
3.5. Orquestación de contenedores	32
3.6. Microservicios	32
4. Controladores PI y su conjunto estabilizante	33
4.1. Controladores PID	33

4.1.1. Reglas de sintonización	35
4.2. Análisis de estabilidad para un controlador PI	37
4.2.1. Sintonización PI por <i>Ziegler & Nichols</i>	37
4.2.2. Conjunto estabilizante para sistema ante control PI	40
4.3. Fragilidad de controladores PI	42
4.3.1. Geometría para márgenes de estabilidad en un controlador PI	43
4.3.2. Definición de métrica para calcular distancia a inestabilidad	48
5. Recomendaciones	53
6. Trabajo futuro	53
7. Conclusiones	53
Referencias Bibliográficas	57
Apéndices	63

Lista de Figuras

Figura 1. Arquitectura sugerida por AWS para una aplicación en WordPress. Adaptado de (Aws, 2018)	19
Figura 2. Arquitecturas sugeridas por Azure para una estación de Blockchain. Adaptado de (ALTIMORE, 2018)	20
Figura 3. Arquitecturas sugeridas por GCP para una aplicación basada en eventos. Adaptado de (Goo, 2018)	21
Figura 4. Responsabilidades según el modelo de servicio. Adaptado de (Gantenbein, 2017)	22
Figura 5. Modelo de un cluster activo-activo. Adaptado de (Villanueva, 2015)	24
Figura 6. Modelo de un cluster activo-pasivo. Adaptado de (Villanueva, 2015)	25
Figura 7. Escalabilidad vertical en contraste con escalabilidad horizontal.	26
Figura 8. Arquitectura tradicional en contraste con arquitectura con máquinas virtuales.	28
Figura 9. Máquinas virtuales vs contenedores. Adaptado de (Joy, 2015)	30
Figura 10. Solicitudes procesadas en 600 segundos. Adaptado de (Joy, 2015)	31
Figura 11. Arquitectura monolítica vs arquitectura de microservicios. Adaptado de (Kap-gantula, 2019)	33
Figura 12. Controlador PID en forma de realización paralela	35
Figura 13. Respuesta escalón en forma de s del método en lazo abierto	38
Figura 14. Respuesta escalón del sistema compensado	40

Figura 15. Conjunto estabilizante en el plano (k_P, k_I)	41
Figura 16. Intersección en el plano (k_P, k_I) para linea recta y elipse	47
Figura 17. Representación en (k_P, k_I) para márgenes de estabilidad	54
Figura 18. Distancia a la inestabilidad	55

Lista de TablasTabla 1. Cálculo controlador PI métodos de *Ziegler & Nichols*

39

Lista de Apéndices

	pág.
Apéndice A. Fundamentos de sólidos rígidos	63
Apéndice B. Función <i>ode45</i> de MATLAB	71
Apéndice C. Interfaz de animación de la dinámica del sistema	75

Glosario

Backend Capa software que se encarga de gestionar los datos de una o varias aplicaciones.

Continuous deployment Práctica que consiste en realizar el proceso de despliegue de forma automática al ambiente de producción.

Dataset Conjunto de datos generados o extraídos de una fuente para su posterior procesamiento.

Development & Operations Es una práctica de ingeniería de software que tiene como objetivo automatizar y monitorizar las etapas del ciclo de vida del software..

Infraestructura Conjunto de componentes informáticos que permiten mantener un conjunto de aplicaciones al servicio de usuarios.

Internet de las cosas Es un sistema de dispositivos de computación interrelacionados con la capacidad de transferir datos a través de una red, sin requerir intervención humana.

Sistema de alimentación ininterrumpida Dispositivo que almacena energía para suministrar a equipos en caso de alguna falla eléctrica

Resumen

Título: Definición de una infraestructura cloud de alta disponibilidad en un entorno distribuido para el despliegue de una plataforma IoT *

Autor: Jose David Rojas Aguilar **

Palabras Clave: Infraestructura, Cloud, Alta disponibilidad, Definición, Cluster.

Descripción: El internet de las cosas (IoT) ha tenido un gran impacto en los últimos años dentro de la sociedad en diferentes contextos. Una de sus aplicaciones mas importantes es la mejora de la calidad de vida y el apoyo en el proceso de aprendizaje de los estudiantes dentro de las diferentes universidades. Un reto dentro del IoT es tener una infraestructura capaz de soportar una cantidad masiva de dispositivos enviando información de forma constante.

Este trabajo de investigación se busca diseñar una arquitectura software para el despliegue de una plataforma IoT en una infraestructura cloud de alta disponibilidad en un entorno distribuido. El proyecto inició con una fase de exploración con el fin de identificar las herramientas disponibles en el mercado. Luego se definieron unos criterios de selección a nivel técnico, unos requisitos de los artefactos a desplegar, un conjunto de métricas y un conjunto de pruebas para evaluar el desempeño de la arquitectura plateada. Después se definió una arquitectura la cual fue evaluada, en una primera oportunidad, por las pruebas diseñadas durante el proyecto y, finalmente, en una prueba de integración en un caso de uso de la plataforma IoT.

* Trabajo de grado

** Facultad de Ingenierías Físico-Mecánicas. Escuela de Ingeniería de sistemas e informática. Director: Gabriel Rodrigo Pedraza Ferreria, Doctorado en Ciencias de la Computación.

Abstract

Title: Definition of a cloud infrastructure of high availability in a distributed environment for the deployment of an iot platform *

Author: Jose David Rojas Aguilar **

Keywords: Infrastructure, Cloud, High availability, Definition, Cluster.

Description: The Internet of Things (IoT) has had a strong impact in recent years within society in different contexts. One of its most important applications is the improvement of the quality of life and the support in the learning process of the students in different universities. A challenge in the IoT is to have an infrastructure capable of supporting a massive amount of devices sending information constantly.

This research work looks for design software architecture for the deployment of an IoT platform in a high availability cloud infrastructure in a distributed environment. The project began with an exploration phase in order to identify the tools available in the market. Then technical selection criteria were defined, requirements of the artifacts to be deployed, a set of metrics and a set of tests to evaluate the performance of the proposed architecture. An architecture was defined, which was evaluated, at a first opportunity, by the tests designed during the project and, finally, in an integration test in a case of use of the IoT platform.

* Bachelor Thesis

** Faculty of Physical-Mechanical Engineering. School of Systems Engineering and Informatics. Advisor: Gabriel Rodrigo Pedraza Ferreira.

Introducción

La expansión de internet y los grandes avances la comunicación entre diferentes dispositivos ha permitido el surgimiento de un nuevo paradigma tecnológico: Internet de las cosas (IoT). IoT se define como la infraestructura mundial para la sociedad de la información, que permite poner diferentes servicios a dispositivos interconectados entre sí, gracias a las tecnologías de la información y comunicación, permitiendo la extracción y procesamiento masivo de datos.

El internet de las cosas ha tenido un impacto significativo en contextos de diferentes magnitudes. Grandes empresas como Amazon, Microsoft, Google, Cisco, Philips e Intel han invertido millones en investigación, generando diferentes servicios enfocados al hogar, empresas e incluso ciudades. Barcelona es una ciudad pionera en el uso de IoT en sus calles, por ejemplo, la línea 9 del metro de Barcelona se ha actualizado con ascensores inteligentes que utilizan los datos en tiempo real para adaptarse a las necesidades de los viajeros lo cual ha logrado disminuir las aglomeraciones y el consumo de energía para 30 millones de pasajeros al año(BAR, 2018).

Uno de los campos de aplicación del IoT se encuentra en las universidades con el fin de mejorar la calidad de vida y apoyar el proceso de formación en los estudiantes. La implementación de una solución IoT suele ser una tarea muy compleja que requiere de una infraestructura hardware y software capaz de soportar una cantidad masiva de dispositivos enviando información de forma continua por lo cual una característica fundamental dentro de una infraestructura IoT es la alta

disponibilidad.

En este trabajo de investigación se presenta el diseño de una infraestructura para el despliegue de una plataforma IoT en una infraestructura cloud de alta disponibilidad en un entorno distribuido con el fin de proveer un entorno que pueda soportar una gran cantidad de dispositivos conectados enviando datos de forma masiva.

1. Objetivos

Objetivo general

Definir e implementar mecanismos para el despliegue de una infraestructura cloud que provea alta disponibilidad y escalabilidad para el caso de una plataforma IoT en un entorno distribuido.

Objetivos específicos

Identificar las herramientas en el mercado para el despliegue de una infraestructura cloud.

Diseño de la arquitectura y pruebas para validar la arquitectura planteada.

Despliegue de la arquitectura planteada en un entorno de pruebas y aplicación de las pruebas planteadas.

2. Estado del arte

Es importante tener en cuenta los aportes que han hecho otros colegas en trabajos relacionados con el presente trabajo de grado, y por lo tanto, se hará referencia a algunos de estos proyectos.

Administración de prototipo infraestructura de computación en la nube del conuss con énfasis en la implementación de nuevos módulos de Openstack. Este trabajo tenía como objetivo administrar, monitorear y mantener el funcionamiento de la infraestructura de computación en la nube modelo de alta disponibilidad del conuss, con énfasis en la exploración de nuevas tecnologías y otros módulos de Openstack en el conuss (Martínez, 2017).

Estudio de una alternativa de integración continua que soporte el desarrollo de una infraestructura TI de servicios de información del transporte público de pasajeros. Este trabajo tenía como objetivo diseñar y evaluar una alternativa de integración continua para el desarrollo de los componentes de una infraestructura TI que ofrece servicios de información al transporte público de pasajeros (Valbuena, 2017).

Implementación de una nueva infraestructura de computación en la nube con modelo de alta disponibilidad basada en Openstack y contenedores para el conuss. Este trabajo tenía como objetivo investigar y complementar el trabajo hecho previamente (Martínez, 2017) en conuss en la creación de una infraestructura de nube (Balaguera, 2018).

Despliegue y monitorización de un clúster Mesos. Este trabajo tenía como finalidad utilizar un sistema de gestión de recursos en un conjunto de nodos computacionales (Mesos) para el despliegue y evaluación del rendimiento de un cluster aplicaciones con diferentes complejidades algorítmica (LÓPEZ, 2016).

Despliegue de una aplicación usando Docker y Kubernetes. Este trabajo tenía como finalidad de optimizar el proceso de despliegue de una aplicación desarrollada por Sparta Consulting Ltd en un servidor con Minikube (MOILANEN, 2018).

Diseño de una arquitectura cloud para una aplicación con varios usuarios. Este trabajo tenía como finalidad diseñar una arquitectura cloud que pudiera soportar una cantidad masiva de usuarios de una aplicación de pagos móvil (SCHUCHMANN, 2018).

A su vez, existen arquitecturas cloud empresariales que pueden ser adaptadas según los requerimientos del proyecto a implementar.

AWS provee un conjunto de servicios y cada arquitecto arma su arquitectura según los requerimientos del proyecto y los servicios que proporciona AWS como podemos ver en la figura 1. Este modelo es también utilizado por Microsoft Azure (Figura 2) y GCP(Figura 3).

3. Marco de referencia

En el presente Capítulo se presentan los conceptos básicos que serán abordados en Capítulos posteriores.

3.1. Computación en la nube

Es un modelo para permitir el acceso, de manera extensa, conveniente y bajo demanda, a un grupo compartido de recursos informáticos configurables (Por ejemplo redes, servidores, al-

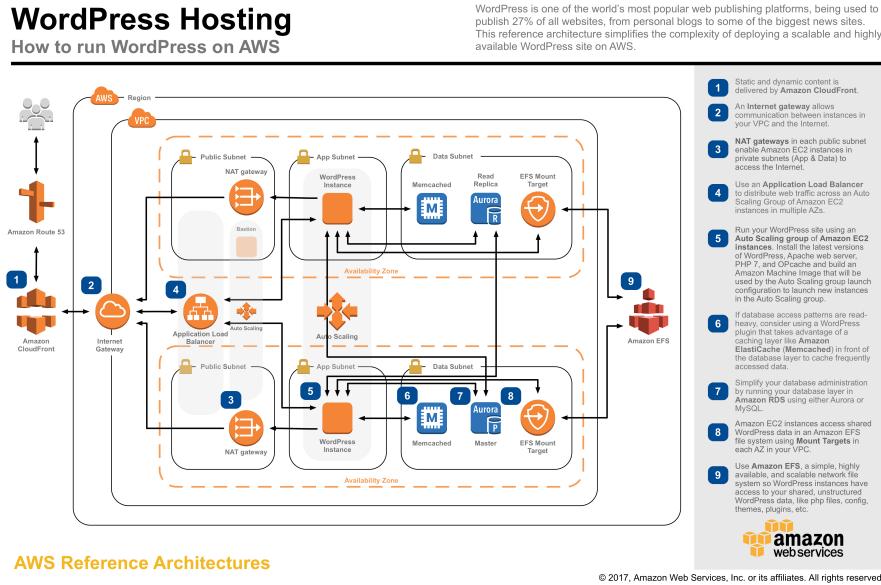


Figura 1. Arquitectura sugerida por AWS para una aplicación en WordPress. Adaptado de (Aws, 2018)

macenamiento, aplicaciones y servicios) que pueden provisionarse y lanzarse rápidamente con un mínimo esfuerzo administrativo o la interacción del proveedor de servicios (Mell, 2011).

3.1.1. Características esenciales.

- **Autoservicio sobre demanda:** Los usuarios tienen acceso a recursos en la nube, por ejemplo capacidad de cómputo o almacenamiento, bajo demanda siempre que sean necesarios.
- **Amplio acceso a la red:** Los recursos están disponibles a través de la red y se accede por medio del mecanismo estándar que promueve el uso de la plataforma por un grupo variado de dispositivos cliente (Teléfonos móviles, tablets, laptops y estaciones de trabajo).
- **Grupación de recursos:** Es una abstracción sobre la manera en la cual se separa la manera en la cual se encuentran los recursos físicamente distribuidos y la asignación de los mismo

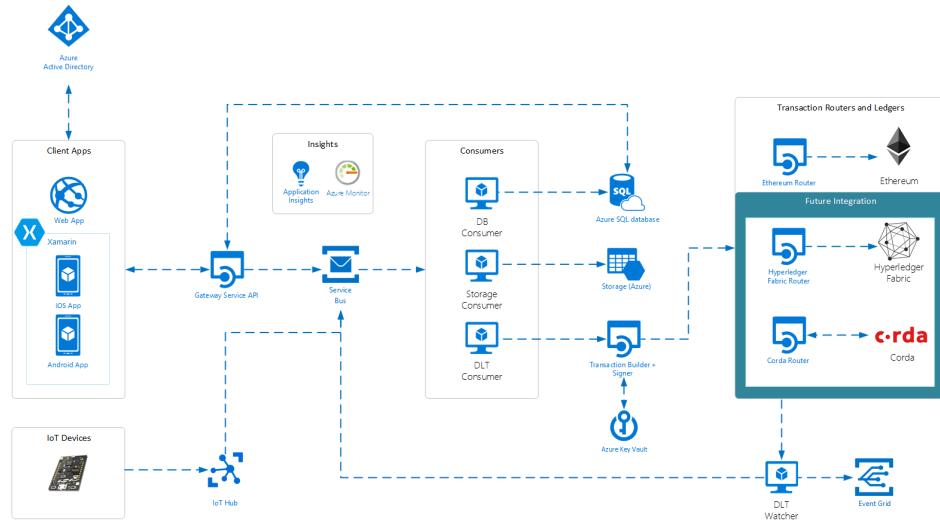


Figura 2. Arquitecturas sugeridas por Azure para una estación de Blockchain. Adaptado de (ALTIMORE, 2018)

para los diferentes clientes. Los clientes suelen tener especificar la ubicación de los recursos a un nivel alto de abstracción (por ejemplo, país, estado o datacenter).

- **Elasticidad:** La capacidad de aumentar o disminuir los recursos asignados para poder escalar de la manera mas óptima una aplicación. Esto puede ser manual o automático.
- **Servicio medido:** Los sistemas cloud poseen diferentes herramientas para poder medir el uso que se le da a los recursos asignados. Estas interfaces pueden ser gráficas o por línea de comando.

3.1.2. Modelos de servicio.

- **Software como servicio:** El cliente tiene acceso a una o varias aplicaciones que se encuentran ejecutando en la infraestructura cloud. El cliente se encuentra limitado al alcance que le provea la aplicación.

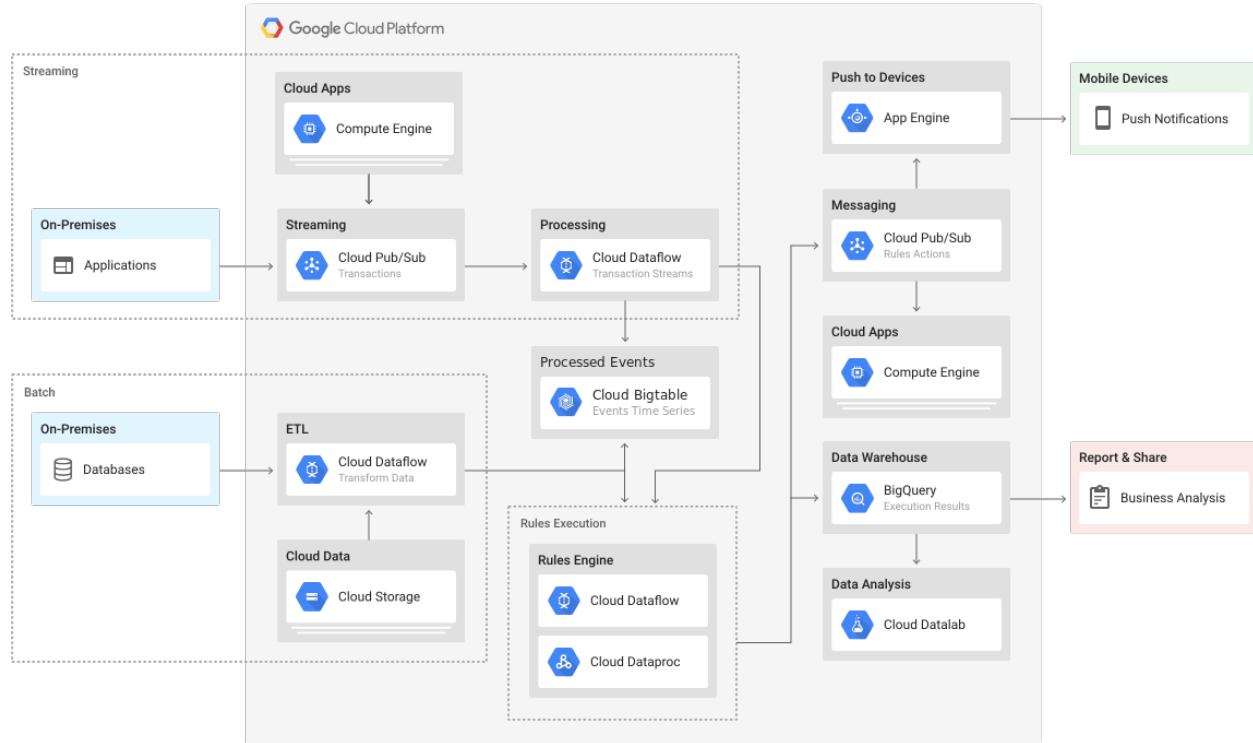


Figura 3. Arquitecturas sugeridas por GCP para una aplicación basada en eventos. Adaptado de (Goo, 2018)

- **Plataforma como servicio:** El cliente puede desplegar aplicaciones en la infraestructura cloud siempre que se usen tecnologías, como lenguajes de programación, librerías, servicios y herramientas, soportadas por el proveedor del servicio.
- **Infraestructura como servicio:** El cliente puede desplegar y correr software de forma arbitraria, esto incluye sistema operativo y aplicaciones, por lo cual no se ve limitado a ninguna configuración preliminar a nivel software.

En la figura 4 se puede ver las responsabilidades del cliente y proveedor según el modelo de servicio.

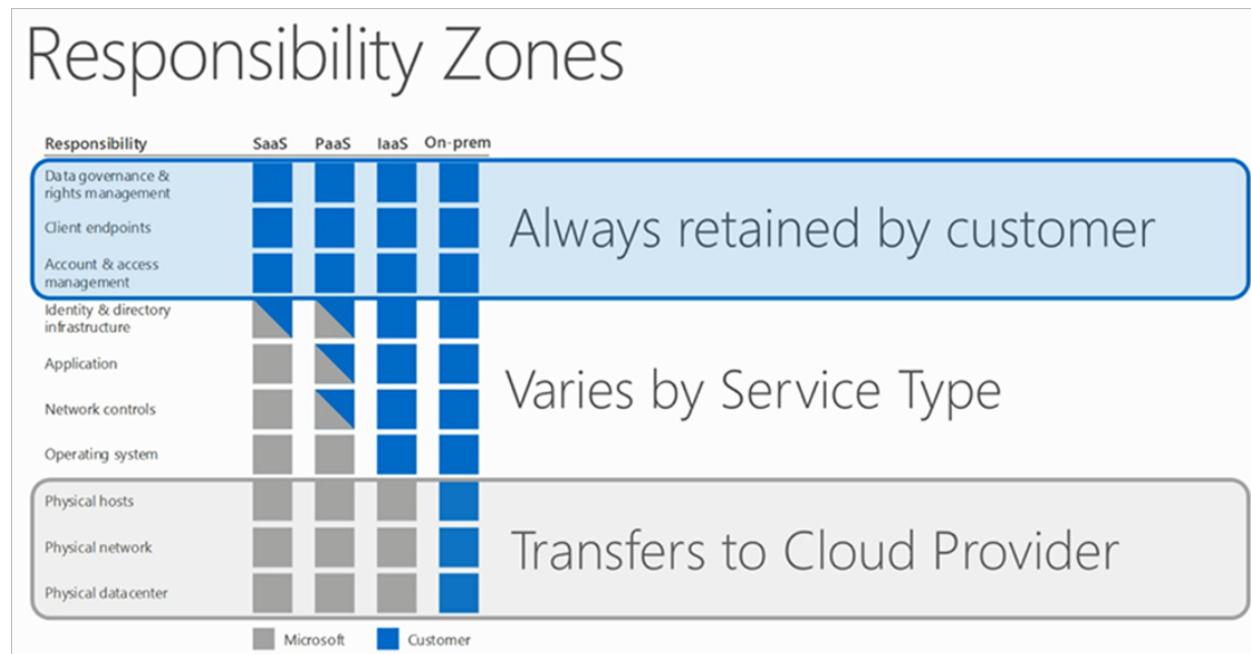


Figura 4. Responsabilidades según el modelo de servicio. Adaptado de (Gantenbein, 2017)

3.1.3. Modelos de despliegue.

- **Nube privada:** La infraestructura cloud es proveída para el uso exclusivo de una organización. Puede ser propiedad, administrada y operada por la organización, un tercero o una combinación de estos dos.
- **Nube comunitaria:** La infraestructura cloud es proveída por una organización para el uso exclusivo de una comunidad de consumidores que tienen unas necesidades comunes.
- **Nube pública:** La infraestructura cloud es proveída para el uso abierto de un público general. Está es administrada, operada y poseída por una empresa, académicos, una organización del gobierno o una combinación de los anteriores.

- **Nube híbrida:** La infraestructura cloud es una mezcla de dos o más tipos de infraestructura(Privada, comunitaria o pública).

3.2. Alta disponibilidad

En computación, disponibilidad es la capacidad de un módulo para ejecutar una función cuando se es requerido. La disponibilidad se expresa de la siguiente manera:

$$\text{Disponibilidad} = \frac{\text{Tiempodeservicio} - \text{Tiemppodeinactividad}}{\text{Tiempodeservicio}} * 100 \quad (1)$$

Cuando hablamos de ?Alta disponibilidad?, hacemos referencia a que cumple el máximo estándar: la disponibilidad medida es de 99.999 % (Benz, 2013).

El objetivo de la alta disponibilidad es eliminar los puntos de fallo potencial en la infraestructura. Un punto de fallo potencial es un componente del stack tecnológico que puede producir una interrupción del sistema. Al igual, todo componente que sea necesario para el sistema y no tenga redundancia, también se considera un punto de falla (Heidi, 2016).

Existen 2 tipos de cluster de alta disponibilidad: (Villanueva, 2015)

- **Activo-Activo:** Se suelen tener al menos 2 nodos, ambos se encuentran corriendo la misma clase de servicios de manera simultánea. El propósito principal de un cluster con alta disponibilidad de tipo activo-activo es lograr un buen balanceo de carga. El balanceo de cargas distribuye las solicitudes sobre todos los nodos con el fin de evitar que un solo nodo se sobrecargue. La configuración más sencilla es presentada en la figura 5

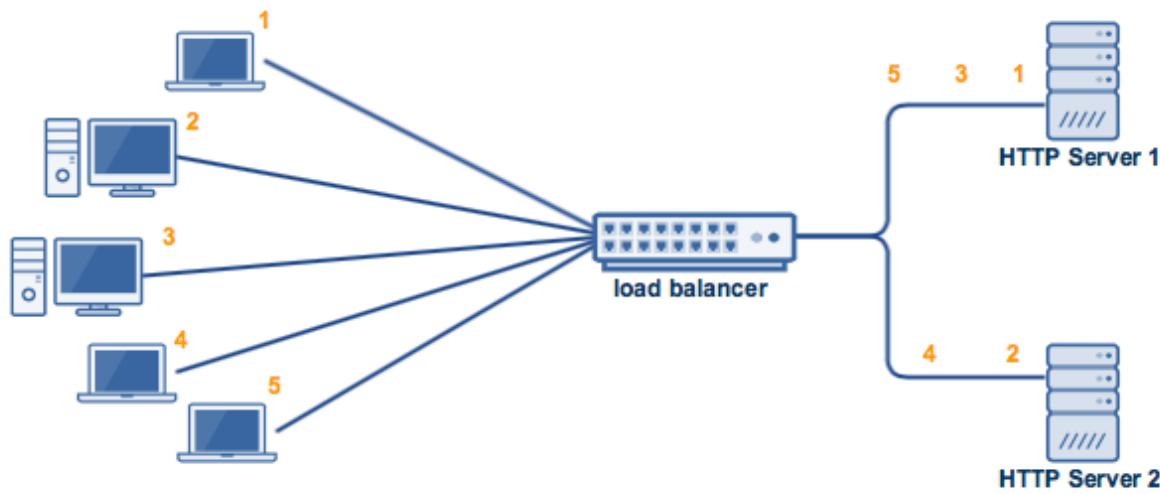


Figura 5. Modelo de un cluster activo-activo. Adaptado de (Villanueva, 2015)

Este modelo tiene un balanceador de carga y 2 servidores http. Los clientes no se conectan directamente a los servidores http, sino que las solicitudes pasan por medio de un balanceador de carga, el cual usa un algoritmo para determinar a qué servidor se direccionan las solicitudes.

Este modelo se recomienda para aplicaciones que van a tener una cantidad masiva de conexiones durante un tiempo prolongado.

- **Activo-Pasivo:** Al igual que la configuración activo-activo, la configuración activo-pasivo requiere al menos de 2 nodos. Se conoce como activo-pasivo por que no todos los nodos empiezan activos.

El nodo pasivo es un respaldo cuya función es recibir las solicitudes de los clientes tan pronto como el nodo activo se desconecte o quede inhabilitado para poder responder a las

solicitudes de los usuarios. En la figura 6 se presenta este modelo.

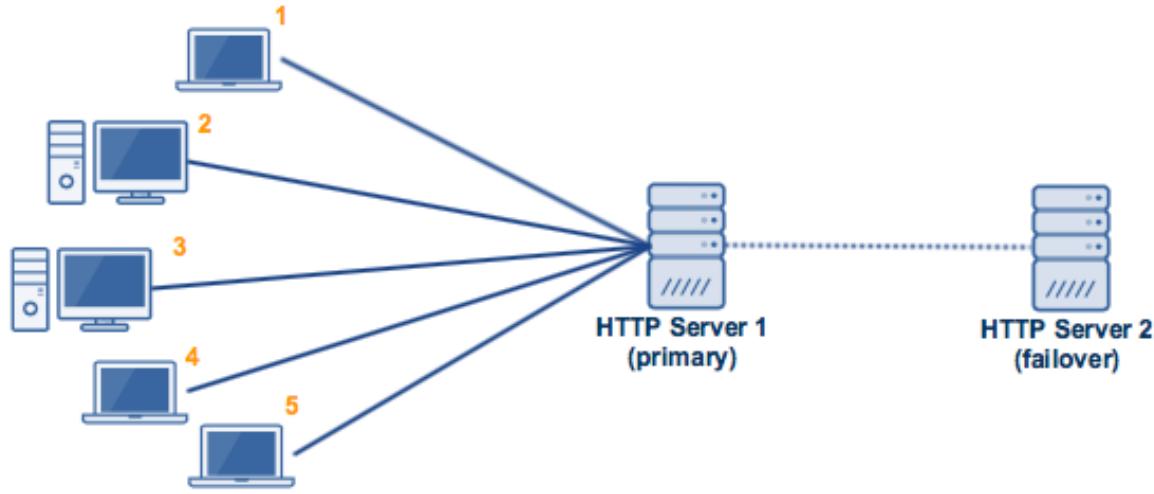


Figura 6. Modelo de un cluster activo-pasivo. Adaptado de (Villanueva, 2015)

Esta configuración se recomienda cuando no se va a tener que lidiar con una cantidad masiva de solicitudes durante un tiempo prolongado pero existen escenarios en los cuales la cantidad de peticiones puede aumentar de manera considerable.

3.2.1. Escalabilidad. Escalabilidad es la capacidad que tiene una solución para poder adaptarse al crecimiento en la demanda.

Imaginemos que tenemos una aplicación corriendo en un servidor con ciertas características, de repente la cantidad de usuarios de nuestra aplicación aumenta por lo cual se hace necesario escalar nuestra infraestructura con el fin de poder seguir brindando el mejor servicio. Existen 2 maneras de escalar:

- **Escalabilidad vertical:** Era la forma de escalar convencional hace unos años y es básica-

mente aumentar las capacidades del equipo de computo o en su defecto, comprar uno nuevo con más capacidad. Es la forma más sencilla de escalar ya que ya que la arquitectura de la infraestructura no cambia.

- **Escalabilidad horizontal:** Consiste en distribuir la carga en un conjunto de equipos de cómputo, de tal manera que si necesitamos soportar una mayor carga, en lugar de cambiar todo el hardware por uno de mayor capacidad, lo que usualmente es muy costoso, simplemente añadimos un nuevo equipo al conjunto lo cual implica una redistribución de la carga y aumenta la cantidad que el conjunto puede llegar a soportar.

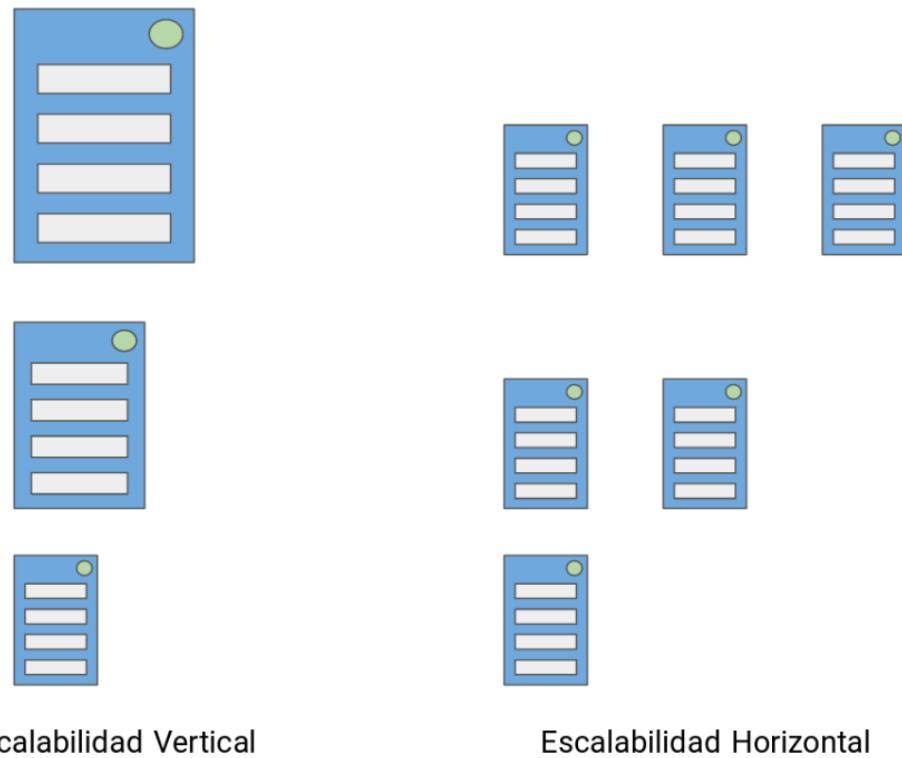


Figura 7. Escalabilidad vertical en contraste con escalabilidad horizontal.

La escalabilidad vertical tiene un problema: La ley de Moore, la cual dicta: ?Cada dos años, aunque en principio dijo que sería cada 18 meses, se duplica el número de transistores. Según la sia, aunque es físicamente posible que los fabricantes de microprocesadores lleguen a crear algunos chips más de lo estipulado por Moore, no sería práctico a nivel financiero, debido a los altos costos que implica.

"Y, siendo optimistas, la fecha límite ?de acuerdo con el presidente y CEO de la sia John Neuffer? sería, como mucho, 2030? (BBC, 2018). En otras palabras, va a llegar un punto en el cual, aunque tengamos dinero ilimitado para poder comprar el equipo de computo mas poderoso en el mercado, va llegar un punto donde el equipo que pueda satisfacer nuestra demanda no exista, lo cual convierte la escalabilidad vertical en una solución inviable para aplicaciones que visionan un crecimiento masivo.

Por otra parte, la escalabilidad horizontal, a pesar de ser más compleja en términos de arquitectura, puede escalar sin estar limitada por la ley de Moore además que, al tener la carga distribuida en varios nodos, podemos proveer un servicio de alta disponibilidad y los costos para escalar

3.3. Virtualización

La virtualización es una capa intermedia a nivel de sistema operativo que provee una abstracción de los recursos del sistema. Es tal el papel de la virtualización dentro del cloud computing que grandes compañías, como Amazon, Google y Microsoft, basan sus servicios cloud en la virtualización.

Las máquinas virtuales son impulsadas por los hipervisores. El hipervisor es un software

que provee un entorno aislado para cada máquina virtual y es responsable de correr diferentes kernels a nivel del sistema operativo anfitrión.

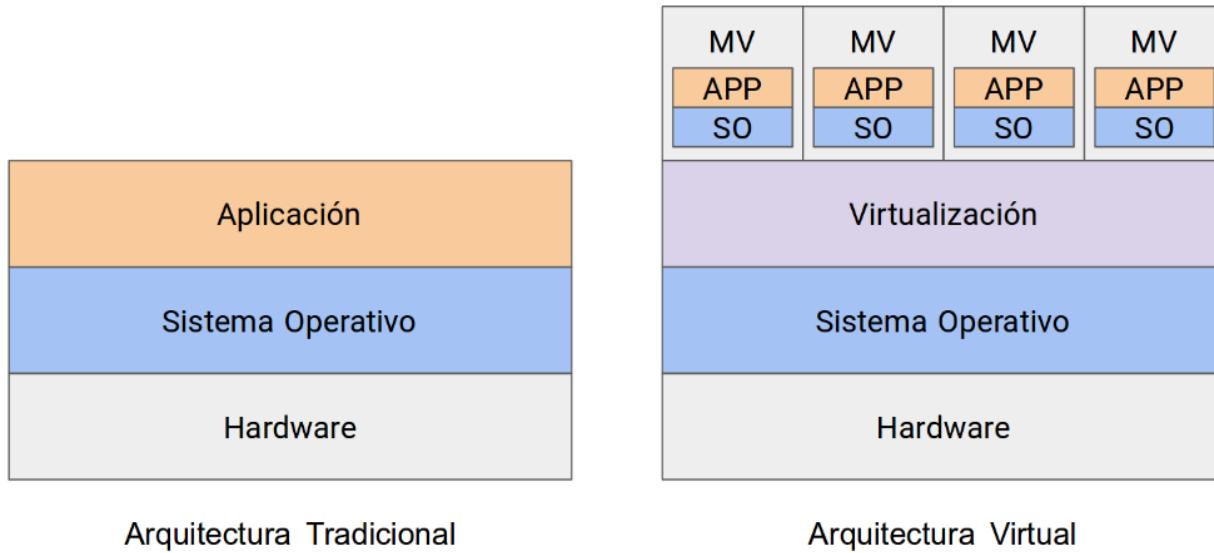


Figura 8. Arquitectura tradicional en contraste con arquitectura con máquinas virtuales.

Como podemos ver en la figura 8, en la arquitectura tradicional hay un acceso más directo al hardware pero posee las siguientes falencias:

- Solo se pueden ejecutar aplicaciones de un sistema operativo.
- Las aplicaciones no se corren en un ambiente aislado por lo que un fallo en el sistema afecta directamente a todas las aplicaciones.
- No hay portabilidad en las aplicaciones.

Las máquinas virtuales resuelven los problemas anteriormente mencionados usando el hipervisor, el cual nos permite crear máquinas virtuales con diferentes sistemas operativos corriendo

al tiempo por lo que podemos tener corriendo, en una misma máquina, aplicaciones para Linux y para Windows al tiempo, adicionalmente cada aplicación puede estar corriendo en ambientes aislados por lo que un fallo en una aplicación no va a afectar a las demás aplicaciones. Sin embargo, las máquinas virtuales tienen los siguientes problemas:

- Los recursos necesarios es significativamente mayor al enfoque tradicional debido a que se deben correr los servicios de cada sistema operativo adicional por cada máquina virtual.
- El rendimiento de las aplicaciones dentro de las máquinas virtuales se ve afectado.
- Los tiempos de encendido y apagado de las máquinas virtuales pueden llegar a ser del orden de minutos.

Para solucionar estos problemas aparecen los contenedores.

3.4. Contenedores

Las aplicaciones software suelen desplegarse como un conjunto de librerías y archivos de configuración en un entorno, por ejemplo, un servidor. Estas se despliegan en un sistema operativo con un conjunto de servicios corriendo, como puede ser un servidor de base de datos o un servidor http, sin embargo estos servicios pueden ser desplegados en cualquier ambiente que pueda proveer los mismos servicios, ya sea una máquina virtual o una máquina física.

Sin embargo, esta metodología tiene un problema relacionado con la actualización o parches ya que estos pueden, por problemas de compatibilidad, dejar una aplicación fuera de servicio. Otro escenario es el cual tenemos 2 aplicaciones en un mismo sistema operativo anfitrión las cuales comparten librerías, luego para solucionar un problema con la aplicación 1, surge la necesidad

de actualizar una de las librerías, en cuyo caso se corre el riesgo de afectar el funcionamiento de la aplicación 2. Para poder evitar cualquier inconveniente durante el despliegue, las compañías de software suelen hacer pruebas antes de realizar el despliegue en el sistema de producción, sin embargo, según la complejidad de la aplicación, estas pruebas pueden llegar a ser una tarea tediosa.

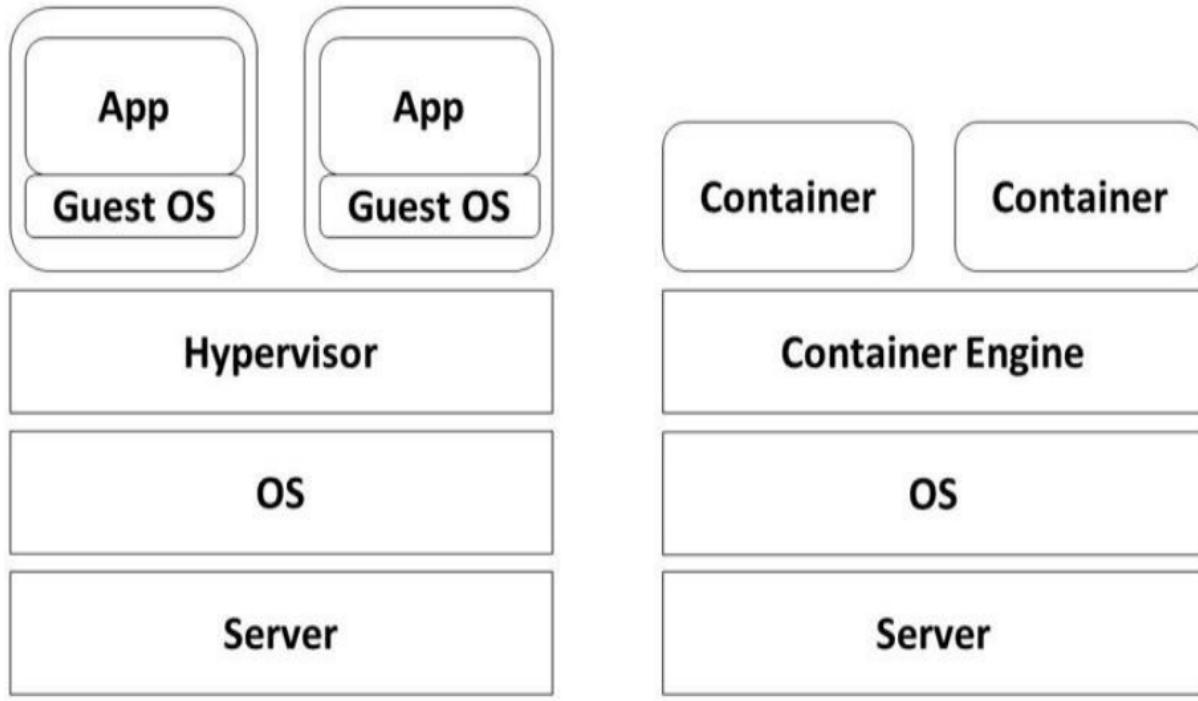


Figura 9. Máquinas virtuales vs contenedores. Adaptado de (Joy, 2015)

Como alternativa, aparecen los contenedores los cuales son un ambiente aislado dentro de un sistema operativo. Los contenedores toman ciertos beneficios de las máquinas virtuales, como la seguridad, el almacenamiento y el aislamiento de red, mientras que consumen muchos menos recursos que las máquinas virtuales. Adicionalmente, los contenedores nos proveen un rendimiento y escalabilidad mayor a las máquinas virtuales como se muestra en la figura 10

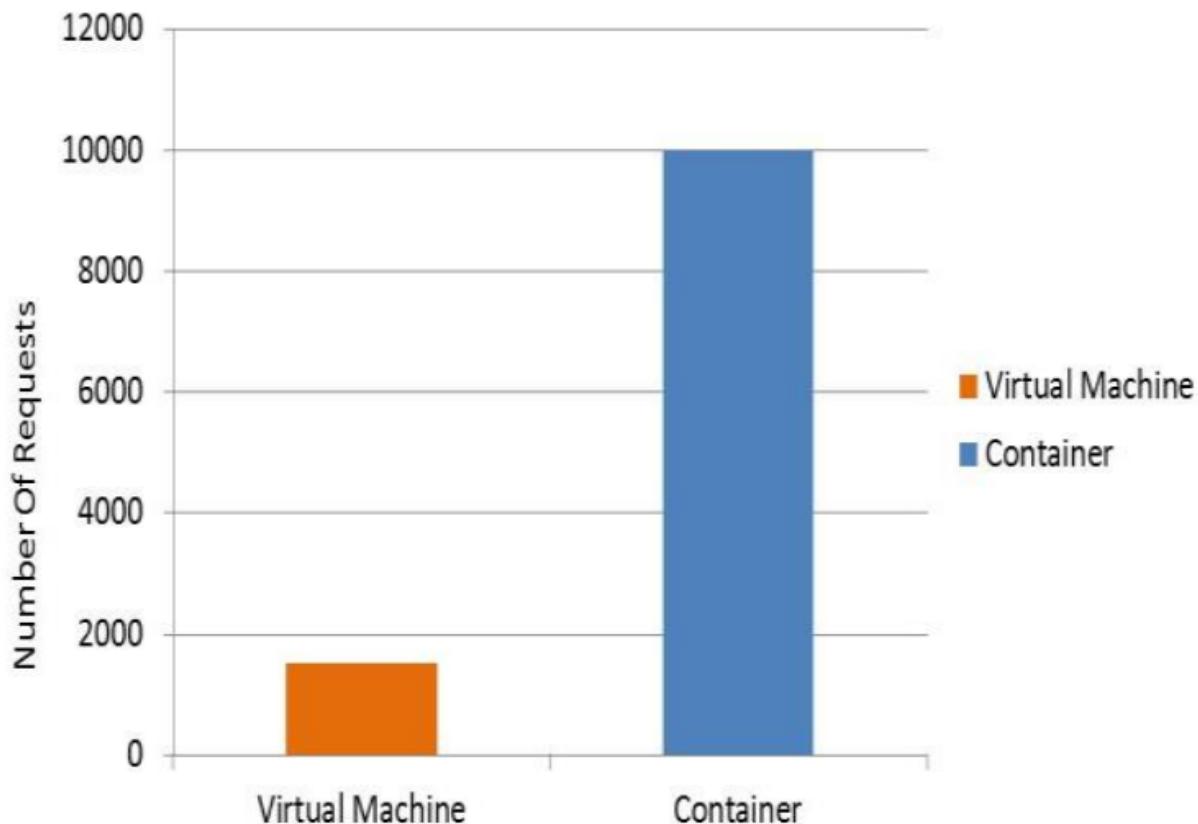


Figura 10. Solicitudes procesadas en 600 segundos. Adaptado de (Joy, 2015)

Los contenedores poseen las siguientes ventajas:

- Poco impacto sobre los recursos
- Ambiente aislado
- Despliegue rápido
- Portabilidad

3.5. Orquestación de contenedores

Supongamos que tenemos una aplicación desplegada usando contenedores y por alguna razón, ya sea un ataque por denegación de servicios o un simple error en el código de la aplicación, el contenedor que la contiene falla. En un sistema de disponibilidad alta debemos asegurar de alguna manera que la aplicación siempre va a estar disponible, por lo tanto debemos implementar una arquitectura que nos permita tolerar fallos, es aquí donde aparece el siguiente concepto: ?Orquestación de contenedores?.

La orquestación de contenedores nos permite desplegar un nuevo contenedor en caso de que otro falle con el fin de mantener la consistencia dentro del cluster, gestionar la manera en que los diferentes contenedores interactúan entre sí. Existen varias tecnologías que implementan una arquitectura para orquestar contenedores: Kubernetes, Docker Swarm o Fleet.

3.6. Microservicios

Los microservicios son una arquitectura y un enfoque sobre la escritura de software en el que las aplicaciones se dividen en componentes más pequeños e independientes entre sí. A diferencia de un enfoque tradicional y monolítico sobre las aplicaciones, en el que todo se crea en una única pieza, los microservicios están separados y funcionan conjuntamente para llevar a cabo las mismas tareas como podemos ver en la figura 11. Cada uno de estos componentes, o procesos, son los microservicios. Este enfoque sobre el desarrollo de software valora la granularidad por ser liviana y la capacidad de compartir un proceso similar en varias aplicaciones. (Hat, 2019)

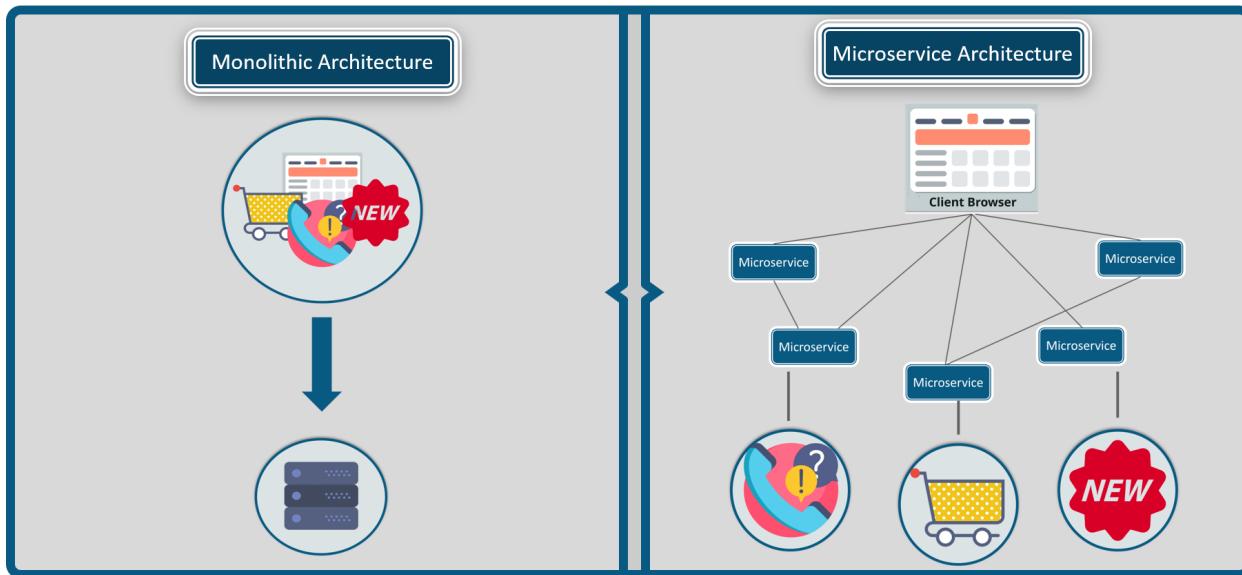


Figura 11. Arquitectura monolítica vs arquitectura de microservicios. Adaptado de (Kappagantula, 2019)

4. Controladores PI y su conjunto estabilizante

Como complemento a los desarrollos presentados en el capítulo anterior, se analiza a continuación la incidencia de conjuntos estabilizantes en controladores clásicos del tipo proporcional/integral (más conocidos como PI), sintonizados empleando las reglas de *Ziegler & Nichols*. La manera de abordar el problema involucra una revisión general de conceptos, el cálculo de \mathcal{S} para un caso de estudio y la definición de una métrica para valorar la *fragilidad* del controlador diseñado.

4.1. Controladores PID

La acción de control proporcional/integral/derivativo (o simplemente PID), constituye la estrategia de control más empleada en automatización de procesos industriales (Astrom and Hagglund, 1995).

Entre las razones por las cuales se prefiere el uso de controladores PID, se incluye la simplicidad de su estructura que con tan sólo 3 términos permite asegurar rechazo ante perturbaciones, velocidades de respuesta apropiadas y la eliminación de errores en estado estacionario. Lo anterior, facilita el cálculo de parámetros de control al igual que su operación y mantenimiento (Díaz and Bhattacharyya, 2016) (Díaz, 2017) (Méndez et al., 2008).

Fundamentalmente, la estructura de una acción PID está constituida de una parte *proporcional al error*:

$$u_P = k_P e(t),$$

siendo $e(t)$ el error de medida y k_P la ganancia proporcional; una parte *proporcional a la historia del error* (a partir del operador de memoria integral en el tiempo):

$$u_I = k_I \int_0^t e(t) dt,$$

con ganancia integral k_I y finalmente, una parte *proporcional al cambio reciente del error* (a partir del operador anticipativo derivada temporal):

$$u_D = k_D \frac{d}{dt} e(t),$$

con ganancia derivativa k_D . La superposición de las tres acciones anteriores permite constituir la

siguiente expresión para el esfuerzo de control:

$$u_{PID}(t) = u_P + u_I + u_D \quad (2)$$

$$= k_P e(t) + k_I \int_0^t e(t) dt + k_D \frac{d}{dt} e(t) \quad (3)$$

$$= k_P \left(e(t) + \frac{1}{T_I} \int_0^t e(t) dt + T_D \frac{d}{dt} e(t) \right), \quad (4)$$

siendo T_I el tiempo integral y T_D el tiempo derivativo. El esquema general para la realización de un controlador PID en forma paralela, se presenta en la Fig. 12.

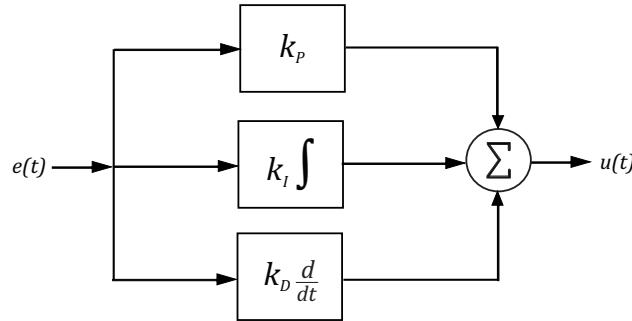


Figura 12. Controlador PID en forma de realización paralela

4.1.1. Reglas de sintonización. La determinación de los parámetros del controlador (k_P , T_I y T_D), que satisfagan condiciones deseadas de desempeño para el sistema controlado, se denomina *procedimiento de sintonización*.

Existen fundamentalmente dos grandes tipos de métodos de diseño o sintonización de controladores PID: 1) los *métodos analíticos* basados en el modelo y 2) los *métodos empíricos* o

experimentales basados en datos reales del proceso.

Dentro del primer grupo, se encuentran los métodos clásicos en el dominio de la frecuencia como el lugar de las raíces o la respuesta en frecuencia mediante diagramas de *Bode* y de *Nyquist*. Sin embargo, estos métodos requieren el conocimiento de un modelo matemático suficientemente apropiado para la dinámica de la planta.

En ocasiones sin embargo este modelo de la planta no se encuentra disponible o simplemente su determinación es inviable, por ejemplo por falta de información de la constitución interna del sistema. Ante esta situación los métodos de control basados en datos (*data-driven control* (Safonov and Tsao, 1997)) adquieren particular relevancia.

A nivel de técnicas de sintonización de controladores PID basadas en datos se destaca el trabajo clásico desarrollado por *Ziegler & Nichols* en 1942 (Ziegler and Nichlos, 1942), el cuál ha sido la base hasta nuestros días de métodos de sintonización para controladores que operan en aplicaciones industriales de diferente naturaleza.

Métodos adicionales de sintonización para controladores PID incluyen: el de *sintonización de relé* (Díaz, 2017); el *Cohen-Coon* (Alfaro, 2002) y otros más modernos involucrando optimización de márgenes de estabilidad a partir de herramientas computacionales (Alzate and Oliveira, 2016) (Díaz and Bhattacharyya, 2016) (Fung et al., 1998) (Ho et al., 1995) (Toscano, 2005).

Para una revisión detallada de métodos de sintonización para controladores PID, se recomienda al lector interesado consultar (Correa et al., 2008).

4.2. Análisis de estabilidad para un controlador PI

A pesar que un controlador PID concentra en una misma estructura las acciones de control necesarias para asegurar una forma de onda adecuada en la respuesta del sistema controlado, la acción derivativa se considera nociva en términos de amplificación de ruidos.

Por esta razón, el controlador proporcional/integral o simplemente PI es una estructura todavía más simple, que concentra los mayores beneficios de simpleza y utilidad práctica en aplicaciones. La función de transferencia para un controlador PI (o PID con acción derivativa nula) viene dada por:

$$\begin{aligned} C(s) &= k_P \left(1 + \frac{1}{T_I s} \right) \\ &= \frac{k_P s + k_I}{s}. \end{aligned} \tag{5}$$

4.2.1. Sintonización PI por Ziegler & Nichols. Considere el sistema dado por:

$$P(s) = \frac{N(s)}{D(s)} = \frac{1}{s(s+1)(s+5)}, \tag{6}$$

y proceda a calcular los parámetros de un controlador PI para el mismo empleando las reglas de *Ziegler & Nichols*.

Inicialmente, se recuerda que existen dos posibles métodos (Ogata, 2010):

- *Lazo cerrado*: donde para una acción proporcional pura en el sistema realimentado, se aplican cambios de tipo escalón en la referencia buscando identificar el valor de k_P para el cual el sistema oscila con amplitud sostenida. Este valor de ganancia se denomina ganancia crítica k_{cr} y al periodo de oscilación correspondiente periodo crítico P_{cr} ;
- *Lazo abierto*: cuando no existe un valor k_{cr} que produzca oscilaciones sostenidas, se recurre a aplicar un estímulo de tipo escalón al sistema en lazo abierto buscando obtener una respuesta en forma de s tal y como la ilustrada en la Fig. 13, siendo T y L las cantidades a ser tomadas en cuenta.

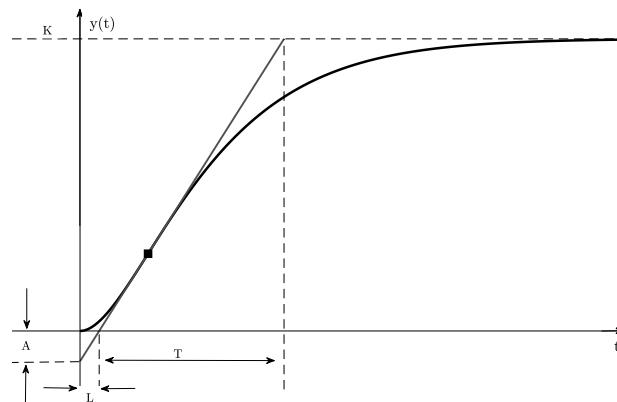


Figura 13. Respuesta escalón en forma de s del método en lazo abierto

Una vez obtenidos los valores importantes para cada caso, los parámetros del controlador PI (i.e.

ganancia proporcional k_P y tiempo integral T_I) se determinan con base en las equivalencias presentadas en la Tabla 1.

Tabla 1

Equivalencias cálculo controlador PI para métodos de Ziegler & Nichols

Método	k_P	T_I
Lazo abierto	$0.45k_{cr}$	$\frac{1}{1.2}P_{cr}$
Lazo cerrado	$0.9\frac{T}{L}$	$\frac{L}{0.3}$

Nota: Adaptado de (Ogata, 2010).

En el caso particular de una configuración realimentada como la presentada en la Fig. ?? para la combinación de planta y controlador dada respectivamente por las expresiones (6) y (5), es posible mostrar que el lugar geométrico de las raíces para el sistema realimentado cruza el eje imaginario cuando $k_P = k_{cr} = 30$, con oscilaciones sostenidas de periodo $P_{cr} = 2.81 [s]$.

De esta manera, el controlador diseñado corresponde con $k_P = 13.50$; $T_I = 2.34$; es decir:

$$\begin{aligned} C(s) &= 13.50 \left(1 + \frac{1}{2.34s} \right) \\ &= \frac{13.50s + 5.76}{s}. \end{aligned} \quad (7)$$

Los parámetros de respuesta para el sistema compensado empleando dicho controlador, corresponden con:

$$Mp = 104.05 [\%]; \quad t_s = 249.54 [s],$$

según ilustrado en la respuesta escalón de la Fig. 14.

Esta respuesta dinámica a pesar de ser estable, no representa un resultado satisfactorio en términos de velocidad de convergencia hacia el valor de estado estacionario, dadas las evidentes oscilaciones del régimen transitorio y el prolongado tiempo de establecimiento. Dicha condición es susceptible de mejora a través de un *ajuste fino*. Nótese sin embargo, que la acción de control es simple (PI) y la planta es de un orden significativo (tercero).

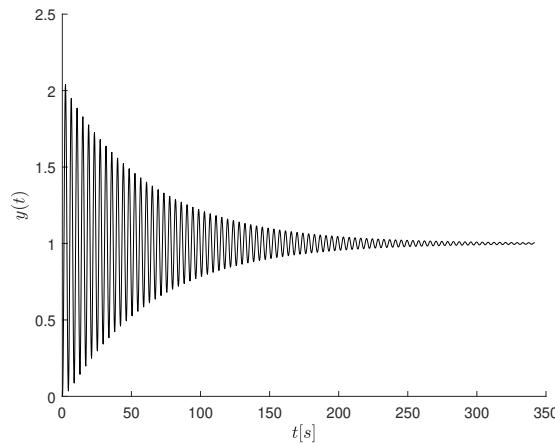


Figura 14. Respuesta escalón del sistema compensado

4.2.2. Conjunto estabilizante para sistema ante control PI. Empleando un tratamiento similar al utilizado para el análisis del conjunto estabilizante de la Sección ??, es posible deducir que la combinación de planta + controlador definida en (6) y (5), permite delimitar una región de

estabilidad en el plano (k_P, k_I) dada por:

$$0 < k_P < 30; \quad (8)$$

$$0 < k_I < \frac{-k_P^2 + 30k_P}{36}, \quad (9)$$

definiendo a su vez el siguiente conjunto estabilizante:

$$\mathcal{S} = \{(k_P, k_I) : (8) \text{ y } (9) \text{ se satisfagan simultáneamente}\}. \quad (10)$$

La Fig. 15 ilustra el conjunto estabilizante (10), resaltando en su interior mediante un asterisco el punto correspondiente al controlador diseñado y definido en (7).



Figura 15. Conjunto estabilizante en el plano (k_P, k_I)

Como se observa, el controlador analizado en la Fig.14 se encuentra cerca de los límites de estabilidad, justificando la alta oscilación de su respuesta dinámica.

La siguiente *Sección* realizará un análisis alrededor de la *fragilidad* del controlador PI.

4.3. Fragilidad de controladores PI

Los métodos de *Ziegler & Nichols* para la sintonización de controladores PI y PID han sido la referencia empleada por ingenieros en diversos campos de aplicación desde su aparición en 1942. Sin embargo, como se observó en el ejemplo presentado en la *Sección* anterior, no siempre se logra una respuesta dinámica adecuada a las exigencias de una respuesta deseada, aunque la misma sea estable.

La forma tradicional de corregir esta situación, es considerar que los parámetros del controlador PI o PID obtenidos por el método de *Ziegler & Nichols* son el valor inicial de un proceso iterativo denominado *ajuste fino*, que permitirá eventualmente obtener una respuesta mejorada en términos de *nuevos valores sintonizados*.

El objetivo de la presente *Sección* no es discutir el proceso de sintonía fina de parámetros en los métodos de *Ziegler & Nichols*, sino evaluar la *fragilidad del controlador* diseñado con dicho método, en términos de su conjunto estabilizante.

En (Keel and Bhattacharyya, 1997) Bhattacharyya define la *fragilidad de un controlador* como aquel fenómeno que implica para el mismo márgenes de estabilidad extremadamente pequeños. Otra manera de entender el concepto es a través de la más pequeña perturbación admisible en

los parámetros de un controlador tal que el sistema realimentado pierda su estabilidad. La fragilidad es un concepto muy cercano a la robustez, y por tanto conviene enfatizar en que la primera estudia la manera en que alteraciones leves en los valores de parámetro de un controlador afectan la estabilidad del sistema realimentado, mientras que la segunda realiza el estudio independiente-mente de donde hayan ocurrido las variaciones paramétricas.

4.3.1. Geometría para márgenes de estabilidad en un controlador PI. Una interpretación geométrica del margen de fase para un sistema realimentado ante un control PI, se propone en (Alzate and Oliveira, 2016). El desarrollo presentado en el presente numeral se basa en el trabajo referenciado y sugiere la manera de aplicar el mismo resultado en términos del margen de ganancia del sistema.

Para ello, asuma $P(j\omega)$ y $C(j\omega)$ como la respuesta frecuencial de la planta y el controlador PI definidos en (6) y (5), respectivamente. En el sistema realimentado estable que se obtiene a partir de esta combinación *planta + controlador*, los márgenes de ganancia A_m y fase θ_m se pueden determinar analíticamente a partir de las condiciones de magnitud y fase:

$$|P(j\omega_g)| |C(j\omega_g)| = 1, \quad (11)$$

$$\angle P(j\omega_\theta) + \angle C(j\omega_\theta) = \pi n; \quad n = \pm 1, 3, 5, \dots \quad (12)$$

de manera que:

$$A_m = \frac{1}{|P(j\omega_\theta)| |C(j\omega_\theta)|}, \quad (13)$$

$$\theta_m = \angle P(j\omega_g) + \angle C(j\omega_g) - \pi, \quad (14)$$

siendo w_g y w_θ las frecuencias de cruce de ganancia y fase, respectivamente.

Alternativamente, estos margenes de estabilidad pueden obtenerse a partir de una relación geométrica en el plano de parámetros (k_P, k_I) de un controlador PI, con respecto al conjunto estabilizante \mathcal{S} de la planta bajo esta acción de control.

Así entonces, tomando en cuenta que:

$$\begin{aligned} C(j\omega) &= k_P + \frac{k_I}{j\omega} \\ &= k_P - \frac{k_I}{\omega} j, \end{aligned} \quad (15)$$

la fase del controlador PI puede expresarse como:

$$\angle C(j\omega) = \arctan \left(\frac{-k_I}{\omega k_P} \right), \quad (16)$$

o equivalentemente:

$$-\omega k_P \tan(\angle C(j\omega)) = k_I. \quad (17)$$

Si en la expresión anterior se asume $\omega = \omega_g$, la fase del controlador $\angle C(j\omega_g)$ debe satisfacer un margen de fase θ_m ante una respuesta frecuencial $P(j\omega)$ conocida para la planta, según definido en (14). Por tanto, ante esta situación la expresión (17) define una linea recta en el plano (k_P, k_I) con pendiente $-\omega \tan(\angle C(j\omega))$, cuyos puntos satisfacen dichas restricciones.

Asimismo, a partir de la condición de magnitud del controlador se tiene:

$$|C(j\omega)|^2 = k_P^2 + \frac{k_I^2}{\omega^2}, \quad (18)$$

o equivalentemente:

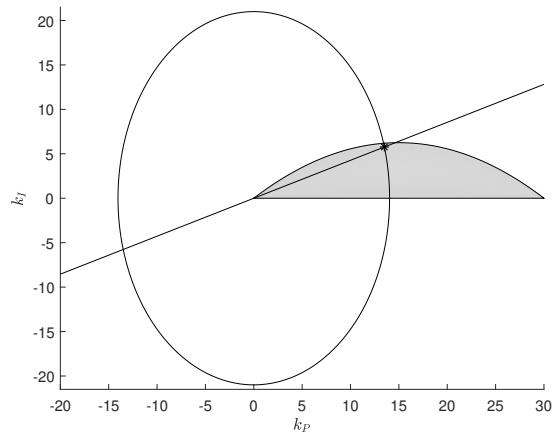
$$\frac{k_P^2}{|C(j\omega)|^2} + \frac{k_I^2}{|C(j\omega)|^2 \omega^2} = 1. \quad (19)$$

Bajo la misma suposición de $\omega = \omega_g$, la expresión anterior representará una elipse en el plano (k_P, k_I) , que intersecta a la recta (17) según ilustrado en el diagrama de la Fig. 16.

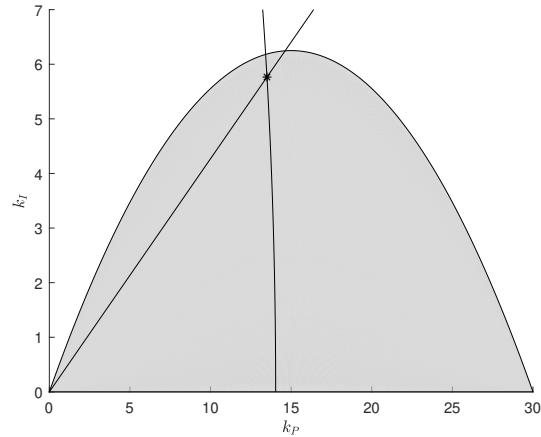
El punto de intersección, dará la coordenada (k_P, k_I) del controlador PI que satisface θ_m y ω_g , representando un método gráfico para el diseño de controladores.

Al respecto del método se resaltan los siguientes detalles adicionales:

- La intersección entre la elipse y la linea recta se da para dos puntos del plano. Sin embargo, puede observarse de la Fig. 16 que sólo uno de ellos tiene sentido práctico por encontrarse al interior del conjunto estabilizante \mathcal{S} ;
- La magnitud para $C(j\omega_g)$ se deja como un parámetro libre de diseño y de este depende la amplitud (i.e. radio mayor y radio menor) de la elipse en el plano;
- El margen de ganancia no se involucra explícitamente en el método sugerido, aunque se entiende que la elección arbitraria para $|C(j\omega_g)|$ influencia dicho valor. La Sección siguiente abordará con mayor detalle este aspecto;
- El procedimiento sugerido puede abordarse explícitamente desde el margen de ganancia, seleccionando $\omega = \omega_0$ y forzando de (13) a la magnitud del controlador $|C(j\omega_0)|$ a satisfacer un margen de ganancia A_m ante una respuesta frecuencial $P(j\omega)$ conocida para la planta. Ante esta situación, el punto de intersección en el plano (k_P, k_I) para la elipse y la linea recta representará la coordenada (k_P, k_I) del controlador PI que satisface A_m y ω_0 , dejando como parámetro libre a la pendiente de la recta y a través de ella, al margen de fase para el sistema controlado.



(a)



(b)

Figura 16. Intersección en el plano (k_P, k_I) para linea recta y elipse, dados $\omega = \omega_g$ y θ_m , ilustrando: (a) panorámica general de la intersección y (b) detalle del punto de intersección

4.3.2. Definición de métrica para calcular distancia a inestabilidad.

Como ya definido al principio de la presente *Sección*, un buen diseño para un controlador debe ir más allá de la simple respuesta dinámica del sistema controlado y por tanto, debe además garantizar la estabilidad para el lazo de control aún ante leves variaciones en sus parámetros.

Complementando las ideas de *Bhattacharyya* y *Keel* en (Keel and Bhattacharyya, 1997), se empleará la visión geométrica propuesta por *Morarescu et al.* adaptada en el presente documento para el caso de lazos PI sin retardo (Méndez et al., 2008) (Morarescu et al., 2010).

Para ello, se define la distancia euclídea:

$$d = \sqrt{(k_P^* - k_P)^2 + (k_I^* - k_I)^2}, \quad (20)$$

siendo (k_P^*, k_I^*) las coordenadas para un controlador en el límite del conjunto estabilizante \mathcal{S} y ortogonal a (k_P, k_I) , que define el radio para una circunferencia de puntos que representan un rango o margen de estabilidad.

A partir de lo anterior, dado un controlador PI al interior de \mathcal{S} es posible cuantificar su *fragilidad* a través de esta métrica, al menos en un modo relativo; es decir, dados dos controladores estables será más *frágil* aquel para el cual se obtenga el menor d .

Sin embargo, no es claro el significado de *fragilidad* para un valor d en un contexto absoluto.

En cualquier caso, el cálculo para d en (20) implica conocer las coordenadas del punto (k_P^*, k_I^*) . Dichas coordenadas representan un valor en la frontera de \mathcal{S} , que conecta con el punto de análisis (k_P, k_I) a través de una línea recta y por tanto, en teoría habrá una solución para (k_P^*, k_I^*) en cada dirección posible de proyección para el vector $(k_P^* - k_P, k_I^* - k_I)$ en el plano. En (Méndez et al., 2008) (Morarescu et al., 2010) por ejemplo, los resultados presentados se realizan a partir de una proyección sobre la vertical. Una solución más general implica el mínimo valor para d en un barrido de 360° lo cual no es trivial, al menos analíticamente, si se piensa en que la descripción para la frontera de \mathcal{S} corresponde con funciones definidas por tramos (es decir, con transiciones condicionadas).

Como alternativa, se presenta en el presente apartado una forma de calcular dicho punto límite a partir del enfoque geométrico discutido en la Sección 4.3.1. Para ello, considere el problema de cuantificar la fragilidad del controlador PI calculado en (7).

Dicho controlador representa un punto en el conjunto estabilizante \mathcal{S} en (10) y a su vez, la intersección entre una línea recta y una elipse dadas respectivamente por (17) y (19), con los siguientes parámetros: $\omega_g = 1.49 \text{ rad/s}$ y $\theta_m = 1.18^\circ$ (el cálculo para dichos parámetros fue realizado empleando la función *allmargins(.)* de MATLAB).

En términos prácticos, la regla de diseño indica que un buen margen de fase es alrededor de 60° (Ho et al., 1995). Como se observa el margen de fase θ_m obtenido por *Ziegler & Nichols* es muy cercano a la inestabilidad y por tanto sugiere *fragilidad*.

Para determinar los valores límite (k_P^*, k_I^*) , se mantiene constante la pendiente de la recta (17) a la misma frecuencia $\omega = \omega_g = 1.49 \text{ rad/s}$ y se incrementa la amplitud de la elipse (19) a partir del parámetro $|C(j\omega_g)|$ (ver Fig.17(a)).

Ante estas condiciones, el margen de ganancia A_m del sistema puede determinarse mediante el cociente entre las distancias euclídeas correspondientes a los puntos (k_P, k_I) y (k_P^*, k_I^*) ; es decir (Díaz and Bhattacharyya, 2016):

$$\begin{aligned}
 A_m &= \frac{\sqrt{(k_P^*)^2 + (k_I^*)^2}}{\sqrt{(k_P)^2 + (k_I)^2}} \\
 &= \frac{\sqrt{(k_P^*)^2 + (-\omega k_P^* \tan(\angle C(j\omega)))^2}}{\sqrt{(k_P)^2 + (-\omega k_P \tan(\angle C(j\omega)))^2}} \\
 &= \frac{k_P^* \sqrt{1 + (-\omega \tan(\angle C(j\omega)))^2}}{k_P \sqrt{1 + (-\omega \tan(\angle C(j\omega)))^2}} \\
 &= \frac{k_P^*}{k_P} \\
 &= \frac{k_I^*}{k_I}.
 \end{aligned} \tag{21}$$

De este resultado se observa que si (k_P, k_I) se encuentra en la frontera de \mathcal{S} entonces $A_m = 1$; si (k_P, k_I) se encuentra fuera de \mathcal{S} entonces $A_m < 1$ y si (k_P, k_I) se encuentra dentro de \mathcal{S} entonces $A_m > 1$, lo cual coincide con el comportamiento esperado según la teoría para dicho margen en términos de la estabilidad del sistema.

De esta manera, igualando (17) para k_I con la condición de frontera dada por (9), se obtiene:

$$\begin{aligned}
 -\omega_g k_P^* \tan(\angle C(j\omega_g)) &= \frac{-(k_P^*)^2 + 30k_P^*}{36} \\
 (k_P^*)^2 - 30k_P^* - 36\omega_g k_P^* \tan(\angle C(j\omega_g)) &= 0 \\
 (k_P^*)^2 - 30k_P^* - 53.64k_P^* \tan(-0.2789) &= 0 \\
 (k_P^*)^2 - 30k_P^* + 15.36k_P^* &= 0 \\
 k_P^*(k_P^* - 14.64) &= 0,
 \end{aligned} \tag{22}$$

tras reemplazar los valores conocidos para ω_g , k_P y k_I . A partir de ello, $k_P^* = 14.64$ corresponde con la solución válida en el conjunto estabilizante \mathcal{S} mostrado previamente en la Fig.15. Dicho valor evaluado en:

$$\begin{aligned}
 k_I^* &= -\omega_g k_P^* \tan(\angle C(j\omega_g)) \\
 &= 6.24,
 \end{aligned} \tag{23}$$

permite obtener como coordenada de frontera: $(k_P^*, k_I^*) = (14.64, 6.24)$ y por ende, un margen de ganancia:

$$\begin{aligned} A_m &= \frac{14.64}{13.50} \\ &= \frac{6.24}{5.76} \\ &= 1.08, \end{aligned}$$

para el controlador (7), que a su vez representa una distancia (ver Fig.18):

$$\begin{aligned} d &= \sqrt{(14.64 - 13.50)^2 + (6.24 - 5.76)^2} \\ &= 1.2369. \end{aligned}$$

Como se observa, el valor de d por sí solo no es tan diciente como los márgenes de estabilidad A_m y θ_m obtenidos, ambos de valor muy pequeño.

Una formulación similar podría haberse realizado calculando a partir de (7) el valor de A_m y ω_0 , y con base en ello determinar los valores límite (k_P^*, k_I^*) tras mantener constante la amplitud de la elipse modificando la pendiente de la recta hasta alcanzar la frontera del conjunto estabilizante \mathcal{S} , y a través de ello el margen de fase θ_m para el sistema (ver Fig.17(b)).

Finalmente, se debe mencionar que la interfaz desarrollada en la Sección ?? fue comple-

mentada incorporando el cálculo gráfico para controladores PI, junto con una determinación para sus márgenes de estabilidad y para la distancia d , como medida de su *fragilidad*.

5. Recomendaciones

Al momento de ejecutar la interfaz desarrollada es importante que el usuario tenga una noción respecto al rango de valores que desea visualizar, pues esto hace más fino el detalle de los puntos sobre los cuales se calcula el conjunto estabilizante y por ende, la exploración de los valores admisibles para el control en la práctica.

6. Trabajo futuro

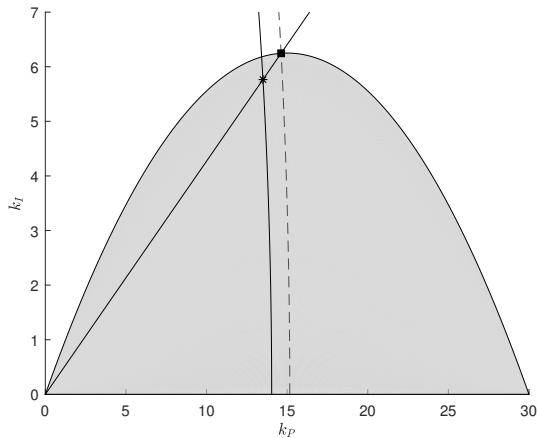
Actividades complementarias a los desarrollos presentados, incluyen el cálculo automático para conjuntos estabilizantes en plantas arbitrarias empleando el *método de la signatura* desarrollado por Keel y Bhattacharyya en (Keel and Bhattacharyya, 2008).

Asimismo es importante explorar otras topologías de compensador y controladores PID, en sus versiones de tiempo continuo y discreto.

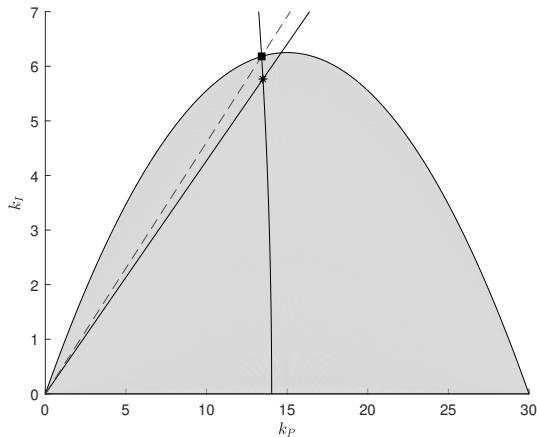
7. Conclusiones

A partir de los desarrollos presentados y los resultados obtenidos en el presente trabajo de grado, es posible enunciar las siguientes conclusiones:

Se interpretaron las tablas de diseño de parámetros PI de *Ziegler & Nichols* en términos de conjuntos estabilizantes. Tal y como fue abordado en la Sección 4.2, se ilustró el diseño de un compensador PI para una planta y posteriormente se analizó la posición de dicho punto en el plano (k_P, k_I) de controladores factibles con base en su conjunto estabilizante. A partir

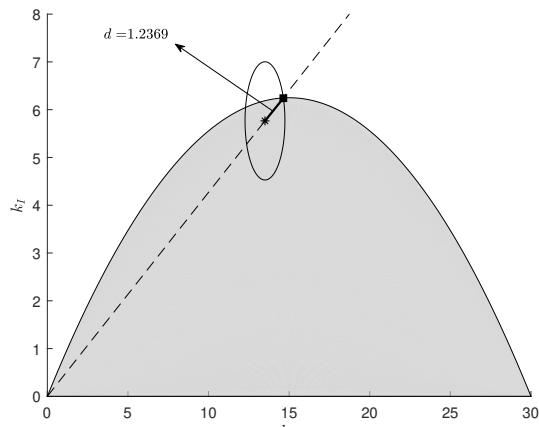


(a)

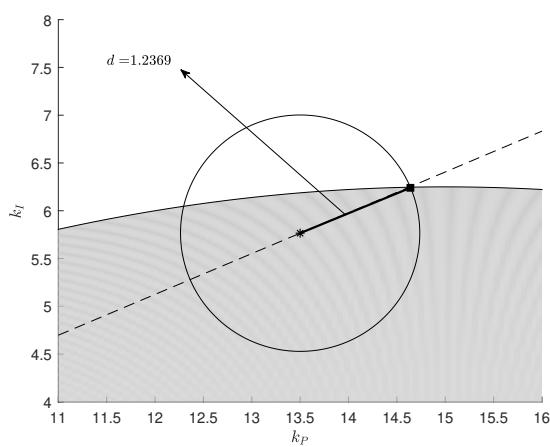


(b)

Figura 17. Representación geométrica en el plano (k_P, k_I) para márgenes de estabilidad: (a) A_m manteniendo fija la recta y variando la elipse; (b) θ_m manteniendo fija la elipse y variando la recta



(a)



(b)

Figura 18. Distancia a la inestabilidad ilustrando: (a) panorámica general de la métrica y (b) detalle de la métrica

de ello, es claro que el método de *Ziegler & Nichols* siempre dará como resultado un controlador estable, tomando en cuenta su carácter empírico. Sin embargo, a partir de la definición de una métrica en la *Sección 4.3.2*, fue posible mostrar a través de una cuantificación para su *fragilidad* que no necesariamente el controlador calculado es estable ante ligeras variaciones en sus valores de parámetro. De otro lado, la definición de conjunto estabilizante fue ampliamente abordada en la *Sección ??* y posteriormente aplicada al caso PI en la *Sección 4.2.*

Se desarrolló un algoritmo que permitió verificar las condiciones de estabilidad para controladores PI diseñados mediante el método de *Ziegler & Nichols*. Inicialmente, se realizó una discusión general de conjuntos estabilizantes en la *Sección ??*, posteriormente complementada en la *Sección 4.2.2* con medidas de inestabilidad a través de una métrica basada en la interpretación geométrica para márgenes de estabilidad en un lazo de control sometido a control PI. El método (o algoritmo) consistió fundamentalmente en calcular el conjunto estabilizante en el plano de parámetros del controlador, para posteriormente transformar las especificaciones de controladores viables a cantidades igualmente viables en el dominio del tiempo. Posteriormente un usuario podría seleccionar el controlador deseado a partir de un punto en el conjunto de parámetros admisible, para el cual se provee además indicación de sus márgenes de estabilidad como medida de *fragilidad*. El procedimiento anterior se desarrolló para los casos de un compensador de 3 parámetros (uno de ellos conocido) y un controlador PI.

Se implementó una interfaz para cálculo de controladores PI a partir de selección de parámetros en el dominio del tiempo, admisibles respecto al conjunto estabilizante correspondiente.

El algoritmo descrito en el ítem anterior fue codificado en una interfaz en MATLAB según se describe en la *Sección ??*, empleando una metodología de diseño del tipo *top-down*.

Referencias Bibliográficas

- (2018). Architecture: Complex event processing. *Google Cloud Platform*. Recuperado de <https://cloud.google.com/solutions/architecture/complex-event-processing>.
- (2018). ?el mundo se está quedando sin potencia computacional?: ¿ qué es la ley de moore y por qué le preocupa al ceo de microsoft? *BBC Mundo*.
- (2018). Hosting wordpress on aws. *Amazon Web Services*. Recuperado de <https://github.com/aws-samples/aws-refarch-wordpress>.
- (2018). Smart city 3.0: pregunte a barcelona por la siguiente generación de ciudades inteligentes. *URBAN HUB*. Recuperado de <http://www.urban-hub.com/es/cities/la-ciudad-de-barcelona-gana-en-inteligencia/>.
- Alfaro, V. M. (2002). Métodos de sintonización de controladores PID que operan como reguladores. *Ingeniería*, 12(1,2):21–36.
- ALTIMORE, P. (2018). Azure blockchain workbench architecture. *Microsoft Azure*. Recuperado de <https://docs.microsoft.com/en-us/azure/blockchain/workbench/architecture>.
- Alzate, R. and Oliveira, V. A. (2016). Multiobjective design of PI controllers with applications. *Control Applications (CCA), IEEE Conference on*, pages 203–214.

Astrom, K. and Hagglund, T. (1995). *PID Controllers: Theory, Design, and Tuning*. ISA: The Instrumentation, Systems, and Automation Society.

Balaguera, F. y López, A. (2018). *Implementación de una nueva infraestructura de computación en la nube con modelo de alta disponibilidad basada en Openstack y contenedores para el grupo de investigación GID-CONUSS*. Universidad Industrial de Santander.

Benz, K. y Bohnert, T. (2013). Dependability modeling framework: A test procedure for high availability in cloud operating systems. *Vehicular Technology Conference*. Recuperado de <https://ieeexplore.ieee.org/document/6692157>.

Chapra, S. C. and Canale, R. P. (2007). *Métodos Numéricos para Ingenieros*. Mc-Graw Hill.

Correa, R., Villamizar, R., and Quiroz, J. (2008). *De La Sintonización De Controladores*. División de Publicaciones UIS.

Díaz, I. (2017). *Modern design of classical controllers*. PhD thesis, Texas A&M University, USA.

Díaz, I. and Bhattacharyya, S. P. (2016). PI controller design in the achievable gain-phase margin plane. In *Decision and Control (CDC), 2016 IEEE 55th Conference on*, pages 4919–4924.

Fung, H. W., Wang, Q. G., and Lee, T. H. (1998). PI tuning in terms of gain and phase margin. *Automática*, 34(9):1145–1149.

Gantenbein, H. y Neira, B. y. L. T. (2017). Securing paas deployments. *Microsoft Azure*

- Documentation.* Recuperado de <https://docs.microsoft.com/en-us/azure/security/security-paas-deployments>.
- Grossman, S. I. (1993). *Algebra Lineal con aplicaciones*. McGraw-Hill.
- Hat, R. (2019). Understanding microservices.
- Heidi, E. (2016). What is high availability. *DigitalOcean*. Recuperado de <https://www.digitalocean.com/community/tutorials/what-is-high-availability>.
- Ho, W., Hang, C., and Cao, L. (1995). Tuning of PI controller based on gain and phase margin specifications. *Automatica*, 31(3):497–502.
- Joy, A. (2015). Performance comparison between linux containers and virtual machines. *ICACEA*.
- Kappagantula, S. (2019). Microservice architecture ? learn, build and deploy microservices.
- Keel, L. H. and Bhattacharyya, S. P. (1997). Robust, fragile or optimal? *IEEE Transactions on Automatic Control*, 42(8):1098–1105.
- Keel, L. H. and Bhattacharyya, S. P. (2008). Controller synthesis free of analytical models: Three term controllers. *IEEE Transactions on Automatic Control*, 53(6):1353–136.
- LÓPEZ, S. (2016). Despliegue y monitorización de un clúster mesos. Master's thesis, Escola Tècnica Superior d'Enginyeria Informàtica Universitat Politècnica de València.
- Martínez, J. y Vásquez, C. (2017). *Administración de prototipo infraestructura de computación*

en la nube del grupo GID-CONUSS con énfasis en la implementación de nuevos módulos de Openstack. Universidad Industrial de Santander.

Mell, P. y Grance, T. (2011). The nist definition of cloud computing. recommendations of the national institute of standards and technology. *National Institute of Standards and Technology.* Recuperado de <https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf>.

Méndez, C., Niculescu, S., Morarescu, C., and Gu, K. (2008). On the fragility of PI controllers for time-delay SISO systems. In *Mediterranean Conference on control and automation*, pages 529–534.

MOILANEN, M. (2018). *Deploying an application using Docker and Kubernetes.* Oulu University of Applied Sciences.

Morarescu, I. C., Niculescu, S. I., and Gu, K. (2010). The geometry of stability crossing curves of PI controllers for SISO systems I/O delays. *Romanian Journal of Pure and Applied Mathematics*, 55(4):297–313.

Ogata, K. (2010). *Ingeniería de control Moderna.* Pearson Education.

Safonov, M. G. and Tsao, T.-C. (1997). The unfalsified control concept and learning. *IEEE Transactions on Automatic Control*, 42(6):843–847.

SCHUCHMANN, M. (2018). Designing a cloud architecture for an application with many users. Master's thesis, University of Jyväskylä.

Sears, F. W., Ford, A. L., and Freedman, R. A. (2005). *Física universitaria: con física moderna*, volume 2. Pearson Educación.

Toscano, R. (2005). A simple robust PI/PID controller design via numerical optimization approach. *Journal of process control*, 15:81–88.

Valbuena, A. (2017). *Estudio de una alternativa de integración continua que soporte el desarrollo de una infraestructura ti de servicios de información del transporte publico de pasajeros*. Universidad Industrial de Santander.

Villanueva, J. (2015). Active-active vs active-passive high availability cluster. *Jscape*. Recuperado de <http://www.jspape.com/blog/active-active-vs-active-passive-high-availability-cluster>.

Ziegler, J. G. and Nichlos, N. B. (1942). Optimun settings for automatic controllers. *Transactions of the ASME*, 64:759–768.

Apéndices

Apéndice A. Fundamentos de sólidos rígidos

Un sólido rígido, es un cuerpo formado por varias partículas puntuales que guardan distancias constantes entre sí (Sears et al., 2005).

Una operación fundamental para definir cantidades en el espacio de movimiento de un sólido rígido es el producto vectorial (también denominado producto cruz (Grossman, 1993)), el cual produce un vector perpendicular (normal) al plano formado por otros dos vectores que se multiplican.

Sean dos vectores \vec{a} y \vec{b} definidos en \mathbb{R}^3 . El producto vectorial entre \vec{a} y \vec{b} (denotado $\vec{a} \times \vec{b}$) es otro vector (digamos $\vec{c} \in \mathbb{R}^3$) cuyo cálculo puede ser efectuado a través de determinantes como sigue:

$$\vec{c} = \vec{a} \times \vec{b} = \begin{vmatrix} i & j & k \\ a_i & a_j & a_k \\ b_i & b_j & b_k \end{vmatrix} = \begin{vmatrix} a_j & a_k \\ b_j & b_k \end{vmatrix} i - \begin{vmatrix} a_i & a_k \\ b_i & b_k \end{vmatrix} j + \begin{vmatrix} a_i & a_j \\ b_i & b_j \end{vmatrix} k \quad (24)$$

De esta manera, siendo $\vec{a} = (1, -1, 2)$ y $\vec{b} = (3, -4, 5)$ se obtiene:

$$\vec{a} \times \vec{b} = \begin{vmatrix} i & j & k \\ 1 & -1 & 2 \\ 3 & -4 & 5 \end{vmatrix} = \begin{vmatrix} -1 & 2 \\ -4 & 5 \end{vmatrix} i - \begin{vmatrix} 1 & 2 \\ 3 & 5 \end{vmatrix} j + \begin{vmatrix} 1 & -1 \\ 3 & 4 \end{vmatrix} k = 3i - j - k \quad (25)$$

La Fig. 19 ilustra esta operación gráficamente en el espacio tridimensional.

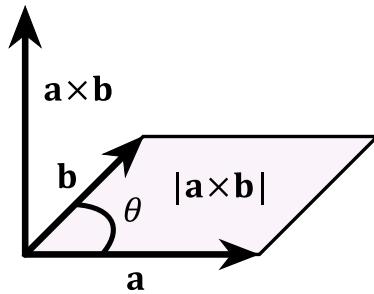


Figura 19. Ilustración gráfica para producto vectorial

Condición de rigidez

Considere el sólido rígido presentado en la Fig. 20. Para cada pareja de puntos (P_i, P_j) pertenecientes al sólido, se cumple:

$$\frac{d}{dt}[|r_i - r_j|] = \frac{d}{dt}[|r_{ij}|] = 0, \quad (26)$$

lo cual significa que la distancia entre puntos de un sólido rígido se mantiene invariante. Esto último se conoce como la *condición de rigidez*.

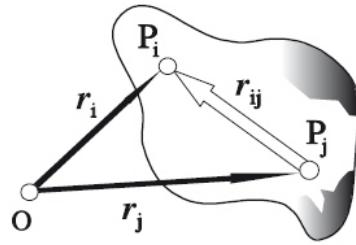


Figura 20. Sólido rígido

Asimismo, a partir de (26) se obtiene:

$$\frac{d}{dt}[|\vec{r}_i - \vec{r}_j|] = |\dot{\vec{r}}_i - \dot{\vec{r}}_j| = 0, \quad (27)$$

y por tanto, sabiendo que \vec{r} es el vector velocidad \vec{v} para un punto del sólido visto desde el observador, es posible escribir:

$$|v_i| = |v_j|, \quad (28)$$

con lo cual la velocidad de translación para cualquier punto del sólido será la misma, y así, una vez definido el movimiento de un punto cualquiera del cuerpo rígido que se traslada en el espacio, es posible definir la totalidad de su movimiento.

Movimiento de rotación

En la Fig. 21 se ilustra un punto que rota alrededor de un eje fijo, localizado en el cuerpo del sólido.

A partir de ello, es posible definir la velocidad angular que experimenta el punto P alrededor

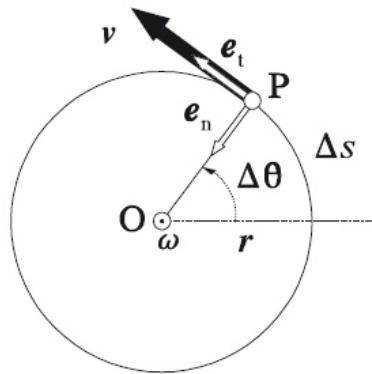


Figura 21. Rotación de un punto del sólido alrededor de un eje fijo

del eje de rotación, en el modo siguiente:

$$\omega = \frac{d}{dt} \theta \quad (29)$$

Tambien, puede escribirse del diagrama la velocidad tangencial v del punto mediante:

$$\vec{v} = \vec{r} \times \vec{\omega}, \quad (30)$$

siendo \vec{r} el vector que marca la distancia del punto P al eje de rotación O .

Por tanto, el vector de aceleración puede ser formulado como:

$$\begin{aligned}\frac{d}{dt}\vec{v} &= \frac{d}{dt}[\vec{r} \times \vec{\omega}] \\ &= \left(\frac{d}{dt}\vec{r} \times \vec{\omega} \right) + \left(\vec{r} \times \frac{d}{dt}\vec{\omega} \right) \\ \vec{a} &= \vec{r} \times \vec{\alpha},\end{aligned}\tag{31}$$

con \vec{a} y $\vec{\alpha}$ representando, respectivamente, los vectores de aceleración lineal y angular. Note que se asume $\frac{d}{dt}\vec{r} = 0$ debido a que el eje de rotación es fijo.

Conservación del momento angular

En un movimiento translacional, el principio de conservación del momento lineal establece:

$$\frac{d}{dt}\vec{p} = \frac{d}{dt}m\vec{v} = 0,\tag{32}$$

a partir de lo cual el momento \vec{p} será constante en ausencia de fuerzas externas.

De manera similar, es posible definir el momento angular \vec{L} de una partícula de masa puntual que rota alrededor de un eje fijo, en el modo siguiente:

$$\vec{L} = \vec{r} \times \vec{p},\tag{33}$$

siendo \vec{r} el vector de distancia a la masa desde el centro de rotación.

Por tanto, el principio de conservación del momento angular puede establecerse como sigue:

$$\begin{aligned}
 \frac{d}{dt} \vec{\mathbf{L}} &= \frac{d}{dt} [\vec{r} \times \vec{p}] \\
 &= \frac{d}{dt} [\vec{r} \times m\vec{v}] \\
 &= m \frac{d}{dt} [\vec{r} \times \vec{v}] \\
 &= m \left([\vec{r} \times \frac{d}{dt} \vec{v}] + [\frac{d}{dt} \vec{r} \times \vec{v}] \right) \\
 &= m ([\vec{r} \times \vec{a}] + [\vec{v} \times \vec{v}]) \\
 &= \vec{r} \times m\vec{a} \\
 &= \vec{r} \times \vec{F} \\
 &= \tau,
 \end{aligned}$$

siendo τ el torque neto aplicado.

Empleando (31) puede relacionarse este torque con la aceleración angular $\vec{\alpha}$, a partir de:

$$\begin{aligned}
 \tau &= \vec{r} \times m\vec{a} \\
 &= \vec{r} \times m(\vec{r} \times \vec{\alpha}) \\
 &= m(\vec{r} \times (\vec{r} \times \vec{\alpha}))
 \end{aligned}$$

donde, si \vec{r} es perpendicular a $\vec{\alpha}$, entonces el producto vectorial se reduce al producto de las mag-

nitudes:

$$\begin{aligned}\tau &= mr^2\alpha \\ &= I\alpha,\end{aligned}\tag{34}$$

siendo I el momento de inercia de las partes rotativas del cuerpo rígido.

La expresión (34) es la segunda ley de Newton de rotación, y podrá ser definida siempre que sea válido un I constante. Dicha situación no siempre es posible, principalmente si se asume que el eje de rotación puede variar en el tiempo. En tal caso, \vec{r} en la Fig. 21 no es constante y por tanto no es válida la solución propuesta para \vec{a} en (31), resultando en la siguiente definición alternativa para τ :

$$\begin{aligned}\tau &= \vec{r} \times m\vec{a} \\ &= \vec{r} \times m \left(\left(\frac{d}{dt} \vec{r} \times \vec{\omega} \right) + \left(\vec{r} \times \frac{d}{dt} \vec{\omega} \right) \right) \\ &= m \left(\left[\vec{r} \times \left(\frac{d}{dt} \vec{r} \times \vec{\omega} \right) \right] + [\vec{r} \times (\vec{r} \times \vec{\alpha})] \right) \\ &= m \left(\left[\vec{r} \times \left(\frac{d}{dt} \vec{r} \times \vec{\omega} \right) \right] \right) + I\alpha.\end{aligned}$$

El término

$$m \left(\left[\vec{r} \times \left(\frac{d}{dt} \vec{r} \times \vec{\omega} \right) \right] \right),$$

representa los efectos (torques) debidos a las variaciones del eje de rotación, que evidentemente también representan variaciones del vector de momento angular \vec{L} . Dichos efectos se denominan *fuerzas iniciales*, puesto que tienen sentido en un marco de referencia de un cuerpo en rotación. Los tipos más representativos de fuerza inercial son los efectos giroscópicos y la fuerza de Coriolis (Sears et al., 2005).

Apéndice B. Función *ode45* de MATLAB

La función *ode45* está basada en un algoritmo de tipo Runge-Kutta, que se desarrolló a partir del método de Euler mejorado (Chapra and Canale, 2007). La función recibe tres parámetros esenciales: $f(t)$ dentro de un *script* en el que se define la ecuación diferencial acompañado por un simbolo @, el vector de límites de tiempo $[t_0 \quad t_f]$ y el vector de condiciones iniciales y_0 . En otras palabras el prototipo básico para usar *ode45* es el siguiente:

$$[t, y] = \text{ode45}(@f(t), [t_0 \quad t_f], y_0); \quad (35)$$

En este caso la solución numérica se almacenará en el vector y para cada uno de los instantes de tiempo presentes en el vector t .

La función *ode45*, resuelve ecuaciones del tipo $\dot{y} = f(t, y)$, por tanto si se desea resolver ecuaciones de orden superior estas deben escribirse como un sistema de ecuaciones diferenciales de primer orden.

A manera de ejemplo, se ilustrará la forma de resolver la ecuación diferencial de segundo orden

$$\ddot{x} - \mu (1 - x^2) \dot{x} + x = 0, \quad (36)$$

donde $\mu > 0$ es un parámetro escalar.

Por tanto, definiendo

$$y_1 = x; \quad y_2 = \dot{x}$$

la expresión (36) puede ser reescrita como

$$\dot{y}_2 = \mu (1 - y_1^2) y_2 + y_1$$

es decir, transformando la ecuación diferencial original de segundo orden y una variable, en una ecuación diferencial equivalente de primer orden y dos variables. Así entonces, es posible construir el vector

$$y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$

cuya dinámica viene representada por

$$\dot{y} = \begin{bmatrix} f_1(t, y) \\ f_2(t, y) \end{bmatrix}$$

siendo

$$f_1(t, y) = y_2; \quad f_2(t, y) = \mu (1 - y_1^2) y_2 + y_1$$

De esta manera, evaluar la expresión (35) permite obtener una matriz de salida y con filas representando los vectores solución para y_1 e y_2 como función de t .

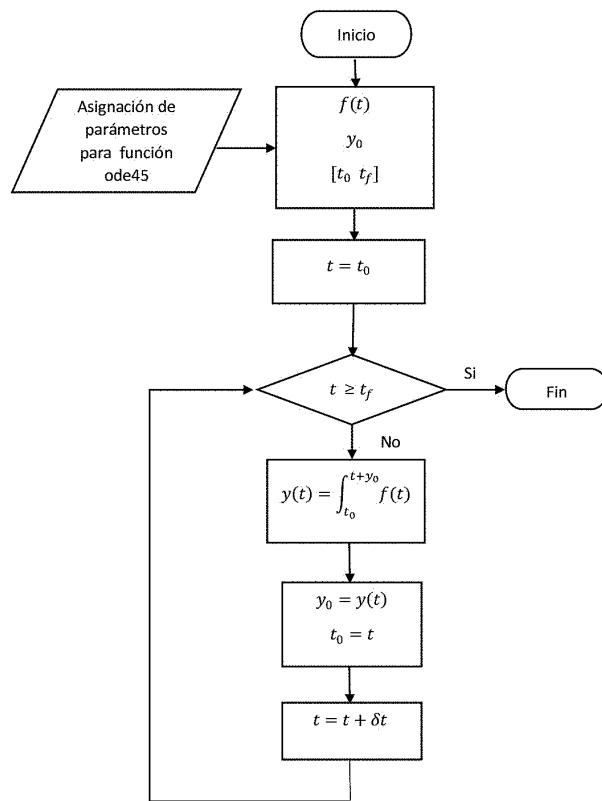


Figura 22. Diagrama de flujo para algoritmo de integración numérica de la función *ode45* de MATLAB

En la Fig. 22 se ilustra el diagrama de flujo del algoritmo empleado para hallar la solución de una ecuación diferencial mediante integración numérica empleando la función *ode45* de MATLAB.

Inicialmente, se deben asignar los parámetros definidos en la ecuación (35).

Posteriormente, un bucle interno hace llamado iterativo a la función $f(t)$ evaluada para valores de tiempo entre t_0 y t_f a partir de las condiciones iniciales y_0 . Para cada ciclo la condición inicial se recalcula siendo la condición final del ciclo anterior. El tiempo se incrementa en un tamaño de paso δt de forma adaptativa, si no se especifica lo contrario. Tras alcanzarse el tiempo final t_f , el bucle interno termina y entrega como resultado el vector de puntos de la trayectoria solución $y(t)$ al igual que el vector de tiempos t .

Apéndice C. Interfaz de animación de la dinámica del sistema

En ausencia de un prototipo real para verificar el comportamiento dinámico del sistema controlado, se optó por construir una animación que permitiera recrear el movimiento del *dron* de una manera cercana al comportamiento físico real. A continuación se presentan las etapas importantes para este desarrollo.

Descripción general de requerimientos

Se requiere construir una interfaz de software que permita visualizar el comportamiento del dron en el espacio de movimiento, como aproximación a la operación real del sistema para diferentes condiciones de simulación (lazo abierto, lazo cerrado controlado PID y por realimentación de estados) ante la presencia de perturbaciones. La interfaz deberá permitir modificar parámetros del sistema y los parámetros de simulación, así como también entregar información de las trayectorias de variables con respecto al tiempo.

Nivel superior de detalle. Posterior a tener una descripción, en palabras, acerca de los requerimientos del sistema (interfaz) a desarrollar, el paso siguiente es crear un diagrama general de entradas y salidas a manera de nivel superior de detalle. Dicho diagrama se presenta en la Fig. 23.



Figura 23. Representación de nivel superior de detalle para desarrollo de interfaz

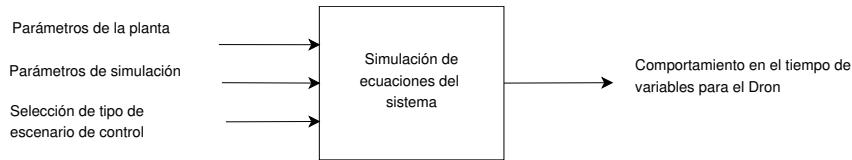


Figura 24. Representación de primer nivel de partición para subproceso de simulación

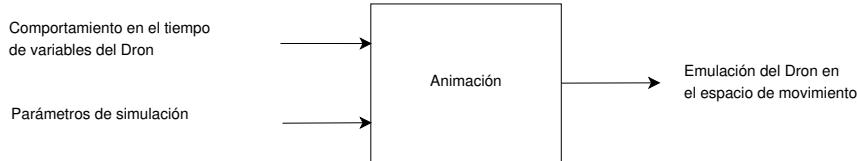


Figura 25. Representación de primer nivel de partición para subproceso de animación

Partición de primer nivel. Una primera partición se logra incorporando el bloque que realiza la solución numérica de las ecuaciones del sistema, a partir de los parámetros de entrada en el modelo y los valores que configuran la simulación, según se muestra en la Fig. 24.

Asimismo, los resultados de este simulador serán la entrada de un nuevo bloque encargado de construir una animación para emular el comportamiento del *dron* en el espacio de movimiento. Una ilustración para este segundo bloque se presenta en la Fig. 25 donde se observa también que será necesario configurar algunas opciones de simulación incorporadas como señal de entrada.

Finalmente, este primer nivel de partición se completa uniendo los dos subprocesos tal y como se ilustra en la Fig. 26.

Partición de segundo nivel. A su vez, es posible abrir el bloque correspondiente a la simulación de las ecuaciones del sistema (según se observa en la Fig. 27) para permitir incorporar la selección del escenario de control que define una configuración importante como lo es la forzante

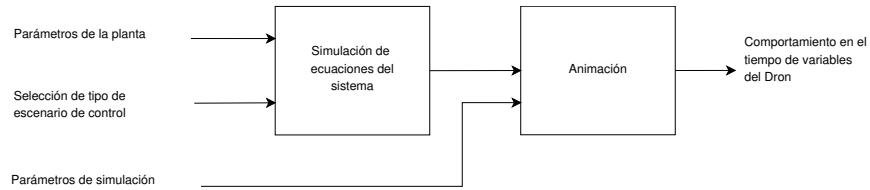


Figura 26. Representación de primer nivel de partición para desarrollo de interfaz

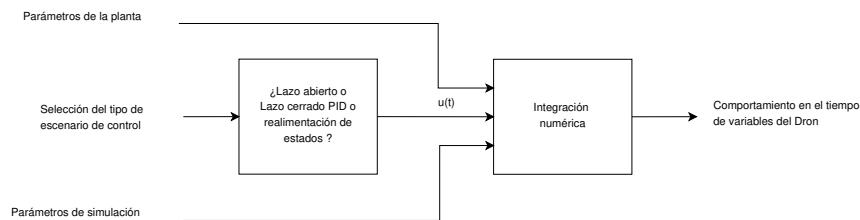


Figura 27. Representación de segundo nivel de partición para subprocesso de simulación

de entrada $\Delta\tau(t)$ en el modelo. De manera similar, el bloque que realiza la solución en el tiempo para las ecuaciones del sistema es un integrador numérico.

Por tanto, en la Fig. 28 se muestra el diagrama de bloques resultante para este segundo y definitivo nivel de detalle.

Selección de herramienta para implementación. A partir del diagrama obtenido en la Fig. 28, es claro que el corazón de la interfaz a ser diseñada es el integrador numérico que resuelve las ecuaciones del sistema. Como ya ilustrado en el Anexo 2, este integrador numérico ha sido codificado empleando la función *ode45* de MATLAB. Por tanto, con el objetivo de facilitar la utilización de los desarrollos numéricos a disposición, se presenta a MATLAB como la primera opción para desarrollar la herramienta de software requerida.

Ahora bien, el segundo elemento importante de la interfaz es la animación que permite emu-

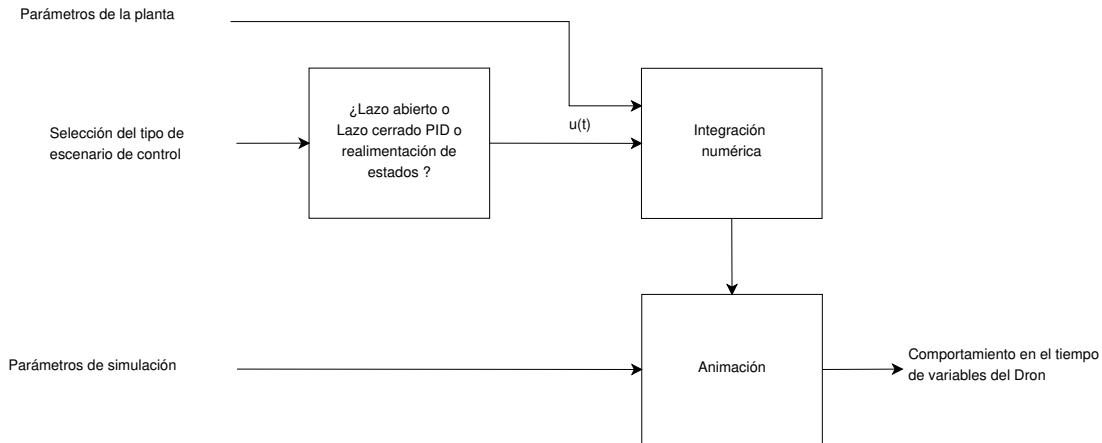


Figura 28. Diagrama de interconexión de susbsistemas que conforman la interfaz de animación de la planta

lar el comportamiento del *dron* en el espacio de movimiento. Por tanto, aunque no es restricción que ambas componentes de la interfaz (simulador y bloque de animación) sean desarrollados en el mismo lenguaje de programación, sí se considera conveniente esta opción por motivos ligados principalmente a la reducción en tiempos de desarrollo y a una mayor compatibilidad entre componentes.

Adicional a esto, se recuerda que MATLAB posee además de la consola de comandos y el entorno de programación gráfico SIMULINK, un entorno para el desarrollo de interfaces de usuario denominado GUIDE (Graphical User Interface Development Environment).

Tomando en consideración todo lo anterior, se selecciona MATLAB vR2014a para construir la interfaz de usuario que satisface los requerimientos de diseño ilustrados en el diagrama de nivel de partición presentado en la Fig. 28.

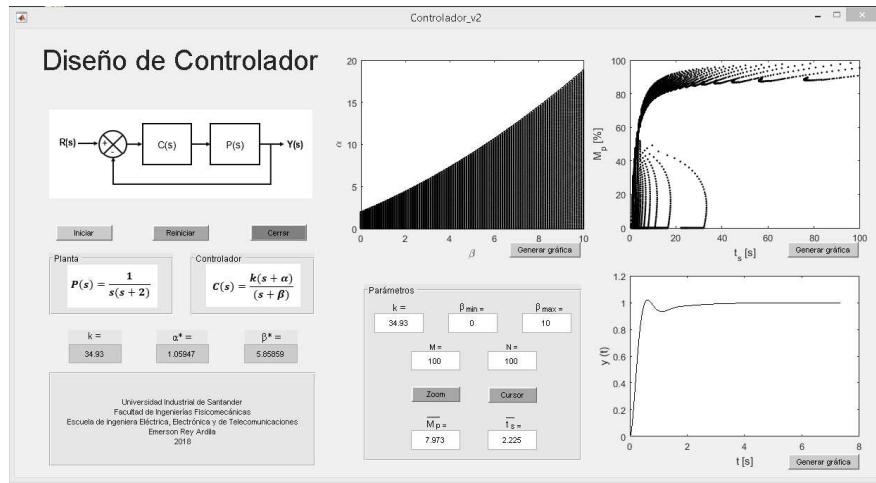


Figura 29. Presentación final para interfaz desarrollada

Descripción de interfaz diseñada. Procediendo con el diseño, se realiza codificación en MATLAB para el diagrama de bloques de la Fig. 28, asumiendo las siguientes variables de entrada:

- Parámetros de la planta: $[g, m, l, k_\tau, b, J_G, I_{xx}, I_{yy}, I_{zz}, A_z]$;
- Selección del tipo de escenario de control: [lazo abierto, lazo cerrado, PID, regulado espacio de estados, seguimiento espacio de estados];
- Parámetros de simulación: [tiempo de simulación, tiempo de perturbación, amplitud de perturbación],

y de salida:

- Comportamiento en el tiempo de variables para sistema *dron*: [posición en eje z ; ángulos de balanceo ϕ , cabeceo θ y guiñada ψ ; vector de velocidades de traslación \mathbf{v} ; vector de velocidades angulares $\boldsymbol{\eta}$; vector de forzantes de control $\Delta\tau$].

Asimismo, se requieren los comandos de control de interfaz siguientes:

- Reset: para reiniciar las variables del programa;
- Tipo de visualización: para seleccionar el gráfico de salida a visualizar;
- Simular: para llamar el inicio de una simulación;
- Salida: para terminar el programa.

Todo lo anterior fue adecuado como se presenta en la Fig. 29, ilustrando la presentación final de la interfaz desarrollada.