PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

ESCUELA DE POSGRADO

CONTROL DE APRENDIZAJE PROFUNDO BASADO EN UN CONTROL PREDICTIVO POR MODELO NO LINEAL PARA UN VEHÍCULO SUBMARINO NO TRIPULADO

Tesis para obtener el grado académico de Maestro en Ingeniería Mecatrónica que presenta:

Manuel Enrique Gallardo Rodríguez

Asesor:

MSc. Jhon Manuel Portella Delgado

Lima, 2024

Informe de Similitud

Yo, Jhon Manuel Portella Delgado, docente de la Escuela de Posgrado de la Pontificia Universidad Católica del Perú, asesor(a) de la tesis titulada(o) Control de Aprendizaje Profundo Basado en un Control Predictivo por Modelo no Lineal para un vehículo submarino no tripulado, de el autor Manuel Enrique Gallardo Rodríguez, dejo constancia de lo siguiente:

- El mencionado documento tiene un índice de puntuación de similitud de 19%. Así lo consigna el reporte de similitud emitido por el software *Turnitin* el 19/09/2024.
- He revisado con detalle dicho reporte y la Tesis o Trabajo de investigación, y no se advierte indicios de plagio.
- Las citas a otros autores y sus respectivas referencias cumplen con las pautas académicas.

Lugar y fecha:

Lima, 19 de septiembre del 2024.

<table>
<tr><td colspan="2">Apellidos y nombres del asesor / de la asesora:
Portella Delgado, Jhon Manuel</td></tr>
<tr><td>DNI:
47099341</td><td rowspan="2">Firma
  </td></tr>
<tr><td>ORCID:
https://orcid.org/0000-0003-2778-686X</td></tr>
</table>

RESUMEN

La presente tesis propone un algoritmo para el entrenamiento de una red neuronal con algoritmos de Redes Neuronales Profundas (*Deep Learning*) en combinación con un control predictivo basado en modelos no lineal (*NLMPC - Non Linear Model Predictive Control*) para el entrenamiento de vehículos submarinos.

El interés por el desarrollo de sistemas autónomos se incrementó considerablemente en los últimos años. Sectores como el industrial y militar han desarrollado esta tecnología con la finalidad de implementar futuros proyectos vinculados especialmente a la navegación autónoma. Este tipo de navegación se basa en sistemas de control en tiempo real que identifican el entorno que rodea al vehículo con la finalidad de tomar decisiones de desplazamiento en función de las restricciones mecánicas del vehículo y físicas del entorno.

Los sistemas de Control Predictivo en Base a Modelos o MPC por sus siglas en inglés Model Predictive Control, han sido usados de manera satisfactoria para el control de vehículos submarinos. Sin embargo, su aplicación requiere estimar los estados del sistema en todo momento, lo cual tiene un coste computacional muy alto en entornos complejos como el medio submarino.

Por otro lado, los sistemas de control con Redes Neuronales, no necesitan estimar de manera explícita la dinámica del modelo, pues obtienen una política que relaciona las entradas del sistema con las acciones finales, mejorando de esta manera el control de sistemas complejos a través de Redes Neuronales constituidas por muchas capas ocultas, sin caer en el error de acumulación de error durante la etapa de entrenamiento.

En este escenario, el presente trabajo tiene como objetivo el control de un robot submarino no tripulado, basado en una combinación del sistema de control no lineal MPC y una Red Neuronal entrenada con algoritmos de aprendizaje profundo (*Deep Learning*). El control no lineal MPC suministrará la información necesaria a la Red Neuronal durante la etapa de entrenamiento, posteriormente la Red Neuronal podrá controlar el movimiento del robot submarino sin la necesidad de conocer todos los estados del vehículo y a un coste computacional mucho menor.

ÍNDICE DE CONTENIDO

RESUMEN ii

ÍNDICE DE TABLAS v

ÍNDICE DE FIGURAS vi

INTRODUCCIÓN 1

1. ESTADO DEL ARTE 4

1.1. Vehículos Submarinos No Tripulados - UUVs 4

1.1.1. Clasificación de UUVs 5

1.1.2. Aplicaciones Industriales 9

1.2. Sistemas de Control de UUVs 11

1.2.1. Sistemas de Control de Vehículos Submarinos 11

1.2.2. Sistema de Control MPC en AUV 15

1.3. Control Predictivo por Modelos (MPC) 17

1.3.1. Principio del MPC 17

1.3.2. Enfoques de Solución MPC 20

1.3.3. Control Predictivo por Modelo Lineal 21

1.3.4. Control Predictivo por Modelo No Lineal 23

1.3.5. Métodos directos para resolver problemas de control óptimo 24

1.3.6. Selección del solucionador NLP 27

1.3.7. Ventajas y Desventajas del Control MPC 28

1.4. Redes Neuronales Profundas 30

1.4.1. Historia y surgimiento de las Redes Neuronales Profundas 30

1.4.2. Redes Neuronales 33

1.4.3. Proceso de aprendizaje 36

2. MODELADO DEL VEHÍCULO SUBMARINO AUTÓNOMO 38

2.1. Sistema de Coordenadas 38

2.2. Cinemática 40
2.3. Dinámica 41
2.4. Modelado de la planta 43

3. SISTEMA DE CONTROL DEL VEHÍCULO SUBMARINO AUTÓNOMO 48
3.1. Sistema de Control No Lineal MPC 49
3.1.1. Modelo Discreto de la Planta 50
3.1.2. Restricciones del Sistema 52
3.1.3. Control de Estabilización de Posición 53
3.1.4. Control de Seguimiento de Trayectoria 60
3.1.5. Análisis de Respuesta a Perturbaciones 65
3.2. Sistema de Control con Redes Neuronales Profundas 68
3.2.1. Generación de la Base de Datos 69
3.2.2. Creación de la Red Neuronal Profunda 71
3.2.3. Entrenamiento de la Red Neuronal Profunda 72

4. SIMULACIÓN Y EVALUACIÓN 78
4.1. Datos de Simulación 79
4.2. Simulación de Sistema 79
4.3. Evaluación de los Resultados 83

CONCLUSIONES Y TRABAJO FUTURO 85

BIBLIOGRAFÍA 87

ANEXOS 92
Anexo 1: Script `NLMPCdatabase.m`, Elaboración de Base de Datos 93
Anexo 2: Script `DNNevaluation.m`, Evaluación de Control con Deep Learning y Control No Lineal MPC 98

ÍNDICE DE TABLAS

Tabla 1.1: Métodos de control clásico y sus limitaciones 12
Tabla 1.2: Tipos principales de controladores usados en UUVs, Watson [2012] . 15
Tabla 1.3: Tipos de controladores alternativos usados en UUVs, Watson [2012] . 15

Tabla 2.1: Notaciones usadas en vehículos marinos 38
Tabla 2.2: Coeficientes Hidrodinámicos del ROV/AUV Falcon 46

Tabla 3.1: Resultados de simulaciones para ejemplo 60
Tabla 3.2: Limites de variables para generación de datos aleatorios 70
Tabla 3.3: Datos de la Red Neuronal Profunda 72
Tabla 3.4: Parámetros para entrenamiento 73

Tabla 4.1: Datos aleatorios para la Simulación 79
Tabla 4.2: Tiempos Ejecución de Simulaciones 84

ÍNDICE DE FIGURAS

Figura 1.1: Primeros vehículos submarinos desarrollados 5
Figura 1.2: Clasificación de Vehículos Submarinos, [Christ and Wernli, 2014] . . 6
Figura 1.3: Modelos de ROV desarrollados para trabajos y observaciones submarinas, [Ostensjo, 2011] 6
Figura 1.4: AUV Autosub 6000, UK Natural Environment Council (NERC), [Wynn et al., 2014] 7
Figura 1.5: Principales Centros de Investigación y Desarrollo de AUVs en el mundo, Gafurov and Klochkov [2015] 7
Figura 1.6: SAUVIM - Autonomous Systems Laboratory, Universidad de Hawaii, [Antonelli, 2006] 8
Figura 1.7: ROVs con Manipuladores incorporados 9
Figura 1.8: ROV para inspección en Instalaciones Petroleras Off-Shore, [Salgado et al., 2010] 10
Figura 1.9: Estructura de Control del Tipo - SIFLC 13
Figura 1.10: Esquema de Control con Redes Neuronales, [Ven et al., 2005] 14
Figura 1.11: Control de trayectoria implementado mediante MF-HOSMC *Model Free High Order Sliding Model Control* 14
Figura 1.12: Predicción de salida y señal de control postulada, [Ramos, 2007]. . . 19
Figura 1.13: Estructura básica MPC, [Camacho and Bordons, 2010]. 19
Figura 1.14: Analogía MPC con manejar un automóvil, Camacho and Bordons [2010]. 20
Figura 1.15: (Izquierda) Trayectorias individuales obtenidas a través de la solución de las EDO. (Derecha) Convergencia de los perfiles de estado y control para el método de disparo múltiple directo. Extraído de Correa [2016] 26
Figura 1.16: Neurona Artificial, Fuente: *https://commons.wikimedia.org/wiki* . . 31
Figura 1.17: Arquitectura de una red neuronal profunda 31
Figura 1.18: Historia de Deep Learning / Inteligencia Artificial. 33
Figura 1.19: Esquema Neurona Artificial 34
Figura 1.20: Señales de Activación 35

Figura 1.21: Efecto de diferentes tasas de aprendizaje 37

Figura 2.1: Sistema de coordenadas del vehículo submarino, Fossen [1994] . . . 39
Figura 2.2: AUV/ROV Falcon 43
Figura 2.3: Plano de evaluación de movimiento de control 47

Figura 3.1: Estrategia de implementación de control de trayectoria por redes profundas 49
Figura 3.2: Modelo de la Planta como Caja Negra 50
Figura 3.3: Lazo de Control MPC para el ROV/AUV Falcon 50
Figura 3.4: Restricciones de estados x y y 53
Figura 3.5: Objetivo de Control: Estabilización de Posición 53
Figura 3.6: Estimación inicial para t=0 54
Figura 3.7: Resultados de simulación con Horizonte $N = 3$ 58
Figura 3.8: Estados del Sistema para $N = 3$ 58
Figura 3.9: Resultados de simulación con Horizonte de Predicción $N = 100$ . . . 59
Figura 3.10: Estados del Sistema para $N = 50$ 59
Figura 3.11: Objetivo de Control: Seguimiento de Trayectoria 60
Figura 3.12: Resultados de simulación con Horizonte de Predicción $N = 100$ . . . 62
Figura 3.13: Estados del Sistema para $N = 50$ 63
Figura 3.14: Objetivo de Control: Seguimiento de Trayectoria 63
Figura 3.15: Resultados de simulación con Horizonte de Predicción $N = 100$ . . . 64
Figura 3.16: Estados del Sistema para $N = 50$ 65
Figura 3.17: Diagrama de Lazo de Control incluyendo perturbaciones 65
Figura 3.18: Resultados de simulación con perturbaciones entre $t = 10 : 15$ 67
Figura 3.19: Posición del AUV Falcon en el plano $X - Y$ 67
Figura 3.20: Estrategia de Control 68
Figura 3.21: Estructura de entradas y salidas de control NLMPC 69
Figura 3.22: Estructura de entradas y salidas de control DNN 70
Figura 3.23: Datos de ingreso y salida para una (01) simulación $run = 1$ 71
Figura 3.24: Movimiento en el plano X-Y para una (01) simulación $run = 1$ . . . 72
Figura 3.25: Base de Datos en Plano XY 73
Figura 3.26: Base de datos de señales de entrada y salida, 2.7 millones de datos . 75
Figura 3.27: Estructura de la Red Neuronal 76
Figura 3.28: Estructura gráfica de la Red Neuronal 76
Figura 3.29: Entrenamiento de la Red Neuronal Profunda 77

Figura 4.1: Modelos de control de Robot Submarino No Tripulado, UVV 78

Figura 4.2: Resultados de Simulación 1 80
Figura 4.3: Simulación 1 - Estados 80
Figura 4.4: Simulación 1 - Control 81
Figura 4.5: Resultados de Simulación 2 81
Figura 4.6: Resultados de Simulación 3 82
Figura 4.7: Resultados de Simulación 4 82
Figura 4.8: Resultados de Simulación 5 83

INTRODUCCIÓN

Durante los últimos años el uso de vehículos submarinos no tripulados o UUV por sus siglas en inglés *Unmanned Underwater Vehicles*, ha tomado un gran interés por parte de la industria de inspección e investigación submarina [Salgado et al., 2010; Hinton et al., 2006; McGinnis et al., 2014; Yuh et al., 1998; Tang et al., 2015]. Sin embargo, realizar el control de un UUV no es una tarea simple debido principalmente a las no linealidades presentes en su dinámica de movimiento tal como lo indica Shahrieel et al. [2015] ; por esta razón, se han venido desarrollando durante los últimos años diferentes tipos de control para este tipo de vehículos [Hassanein et al., 2011; Shi et al., 2006; Jordan and Bustamante, 2009; Caccia and Veruggio, 2000; Loebis et al., 2004; Liu et al., 2014; Ven et al., 2005; Lorentz and Yuh, 1996].

Los sistemas de control clásico no son adecuados para ser utilizados con vehículos submarinos debido a las no linealidades inherentes a este tipo de sistemas. Por esta razón los modelos de Control Predictivo Basados en Modelos o Control Predictivo por Modelos (MPC - *Model Predictive Control*), aparecen como una estrategia de control robusto, adecuado para el control de estos sistemas. El control MPC es una estrategia de optimización en línea que es más eficaz que los métodos tradicionales de control. La principal característica del MPC es que se pueden especificar el comportamiento deseado y las limitaciones en el sistema en la formulación del problema.

Sin embargo, la principal limitación del control MPC es el costo computacional para el cálculo de la solución en tiempo real del problema de optimización. En la mayoría de sistemas físicos, la dinámica del sistema está representada por un modelo no lineal, que implica un problema de optimización no lineal que puede ser no-convexa y, por tanto, computacionalmente muy costosa de resolver. Incluso si la dinámica está representada por un modelo lineal, en cuyo caso el problema de optimización es convexo, la obtención de una solución fiable dentro de un pequeño tiempo de cálculo es muy difícil. Estas deficiencias restringen el uso del control MPC solamente para sistemas dinámicos lentos, con tiempos de muestreo en el orden de segundos o minutos.

El Control Predictivo Basado en Modelos (MPC) es un método eficaz para el control

de sistemas robóticos, en particular de vehículos aéreos o submarinos autónomos, debido a su robustez frente a errores del modelo, su capacidad de utilizar objetivos de alto nivel y relativa simplicidad. Sin embargo, las aplicaciones del MPC pueden ser computacionalmente exigentes y típicamente requieren la estimación del estado del sistema en línea. El problema de la estimación de los estados puede ser bastante difícil en entornos complejos y no estructurados.

El Aprendizaje Profundo, también conocida como *Deep Learning*, es una subdisciplina del Aprendizaje Automático (*Machine Learning*) que se centra en el uso de redes neuronales artificiales con muchas capas llamadas **Redes Neuronales Profundas**. Este tipo de redes pueden incluir pocas capas hasta ciento de ellos con el objetivo de simular la complejidad del cerebro humano para reconocer complejos patrones como textos, imágenes, audio y otro tipo de datos que permitan extraer información y predicciones acertadas. Algunos ejemplos de aplicaciones de este tipo de redes están enfocados en la transcripción de audio, identificación de imágenes o textos, entre otros.

El *Deep Learning* tiene aplicaciones en diferentes sectores como el industrial enfocado en el vehicular, manufactura, espacial y de investigación sobre todo en la disciplina médica. Podemos ver algunas de estas aplicaciones en los vehículos autónomos que utilizan modelos de *Deep Learning* para detectar automáticamente las vías, semáforos y peatones. Así mismo, en el sector militar, los sistemas de defensa utilizan el *Deep Learning* para identificar automáticamente el enemigo en el registro de imágenes satelitales. Otra rama de aplicación es la medicina, donde se utiliza el *Deep Learning* para analizar imágenes médicas con el fin de detectar rastros de células cancerígenas que permitan mejorar el diagnóstico médico.

La motivación para desarrollar la presente tesis se basa en poder obtener las mayores ventajas del tipo de Control Predictivo Basado en Modelos (MPC) y los algoritmos de Aprendizaje Profundo (*Deep Learning*) para el control de un vehículo submarino no tripulado (UUV, *Unmanned Underwater Vehicle*), con una dinámica de sistema no linear. El objetivo principal es realizar el control MPC del modelo no lineal de vehículo submarino UVV para generar una base de datos compleja, esta base de datos servirá para el entrenamiento de una red neuronal profunda que posteriormente será reemplazada por el control MPC, generando de esta manera una reducción significativa en el costo computacional del control del vehículo submarino.

El contenido presentado en la presente tesis es el siguiente:

- El Capítulo 2 resume brevemente el Estado del Arte para los robot submarinos, tipos de control de UUV, control predictivo MPC lineal y no lineal, así como las

redes neuronales profundas.

- El Capítulo 3 realiza el modelado de la planta no lineal desde la descripción del Sistema de Coordenadas, Cinemática y Dinámica del robot submarino. Finalmente se realiza el modelado de la planta para el control sólo en el plano XY.
- El Capítulo 4 muestra la estrategia de control para el robot submarino. Se inicia con la definición del control no lineal MPC en Matlab para el desplazamiento del robot submarino. Luego, se configura el control por redes neuronales profundas, tomando en cuenta las etapas de generación de base de datos del control no lineal MPC y entrenamiento con redes profundas.
- El Capítulo 5 presenta los resultados de las simulaciones y la evaluación de cada una de ellas para verificar el rendimiento del control por redes profundas respecto al control no lineal MPC.
- El Capítulo 6 concluye la tesis con un resumen de los principales resultados y muestra los posibles trabajos futuros.

CAPÍTULO 1

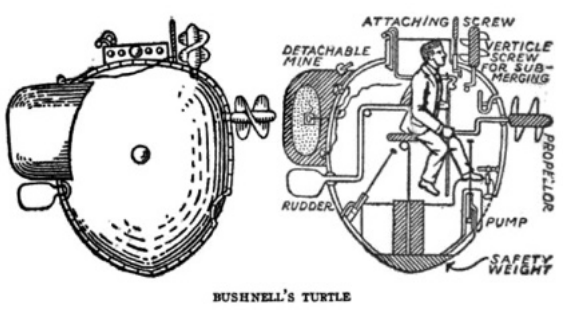
ESTADO DEL ARTE

Aproximadamente el 71 % de la superficie terrestre esta cubierta por agua [Hyakudome, 2011]. Este vasto territorio es utilizado por el hombre principalmente para el tránsito de embarcaciones comerciales por su superficie; sin embargo, bajo ella solo se ha explorado una pequeña área correspondiente a las zonas cercanas al zócalo continental. Con la finalidad de realizar actividades en estos territorios, en los últimos años se han desarrollado equipos submarinos con un concepto inicial de vehículos tripulados por pilotos. Sin embargo, el alto costo y riesgo inherente en las operaciones submarinas, han generado la necesidad de prescindir de pilotos, reemplazándolos por equipos guiados por un sistema de control autónomo o semi-autónomo, desarrollándose de esta manera un nuevo tipo de equipos conocidos como Vehículos Submarinos No Tripulados o UUV, por sus siglas en inglés: *Unmanned Underwater Vehicles*.

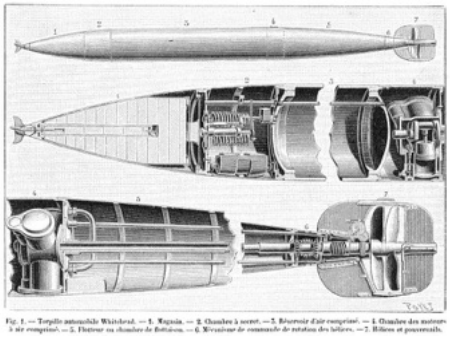
El diseño del primer vehículo submarino fue realizado por W. Bourne en 1578, sin embargo el primero en construirse estuvo a cargo de Cornelius Drebbel en 1620. En el ámbito militar, el primer vehículo submarino utilizado en combate fue *Turtle*, **Fig. 1.1** construido por David Bushnell en 1776. Sin embargo, la primera referencia del diseño de un vehículo submarino viene de la obra *Codice Atlantico* [1480 - 1518] de Leonardo Da Vinci [Antonelli, 2006]. El desarrollo de vehículos submarinos estuvo ligado en sus inicios a fines militares. El primer vehículo implementado fue el torpedo auto impulsado diseñado por Robert Whitehead en 1868 **Fig. 1.1**. Sin embargo, el auge de sectores productivos como el pesquero o hidrocarburos costa afuera (*Off Shore*), sumado al desarrollo tecnológico permitió diseñar equipos altamente especializados con fines comerciales.

1.1. Vehículos Submarinos No Tripulados - UUVs

Christ and Wernli [2014] indica que un UVV es definido como un equipo auto impulsado cuya operación es completamente autónoma (pre-programado o con control adaptado en tiempo real) o bajo mínimo control sin cables, excepto, para transmisión de datos como



(a) Vehículo Submarino *Turtle*, 1776, [Bryson, 1986]



(b) Torpedo Auto Impulsado, 1868, [Antonelli, 2006]

Figura 1.1: Primeros vehículos submarinos desarrollados

por ejemplo a través de fibra óptica. Los UUVs han revolucionado nuestra habilidad para mapear y monitorear el entorno marino, despertando un creciente interés en la investigación y la industria. Actualmente, es común el uso de UUVs para llevar a cabo misiones de inspección al fondo marino, inspección de tuberías, mantenimiento de cables, inspección y mantenimiento de estructuras costa afuera (*Off-Shore*), así como estudios biológicos de especies submarinas animales y vegetales.

1.1.1. Clasificación de UUVs

La **Fig. 1.2** muestra una clasificación de los UVVs. Como se puede observar el término *Unmanned Vehicles* engloba a los Vehículos Operados Remotamente (ROV - *Remote Operated Vehicle*) y los Vehículos Autónomos Submarinos (AUV - *Autonomous Underwater Vehicle*). En sus inicios los UUVs tenían un concepto de control desde superficie empleado actualmente por los ROVs, sin embargo el desarrollo de la tecnología permitió extender la clasificación enfocándola hacia equipos con una mayor autonomía, por lo cual aparecieron los AUVs. Actualmente, debido a la necesidad de realizar misiones en ambientes submarinos, se ha requerido la instalación de manipuladores acoplados al UUV, de esta manera se desarrollaron los UVMS - *Underwater Vehicle - Manipulator System*. A continuación se realiza una descripción de cada tipo de UUVs.

A. Vehículo Operado Remotamente - ROV

Los Vehículos Operados Remotamente ó ROV (*Remotely Operated Vehicle*), son un tipo de vehículos submarinos que están físicamente conectados con equipos en superficie a través de un cable umbilical conocido como *tether*, a través de este medio las señales de

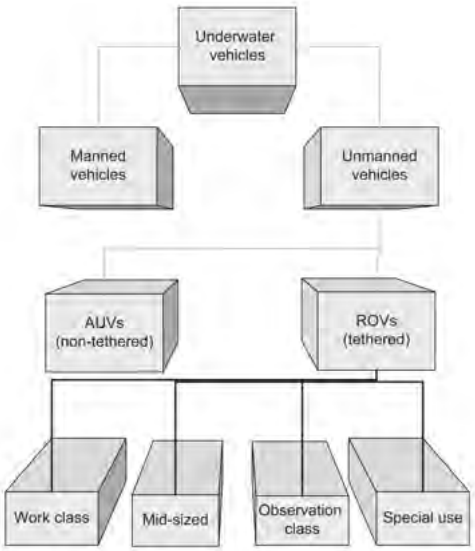


Figura 1.2: Clasificación de Vehículos Submarinos, [Christ and Wernli, 2014]

control y potencia son transmitidas desde superficie. Este tipo de vehículo es empleado en operaciones submarinas especiales dentro de un área limitada, pues tiene una restricción de movimiento debido a la conexión con superficie a través del *tether*. La **Fig. 1.3** presenta algunos modelos desarrollados en la actualidad.

(a) ROV Jason - Woods Hole Oceanographic Institute

(b) ROV VideoRay Pro 3 ROV - VideoRay LLC

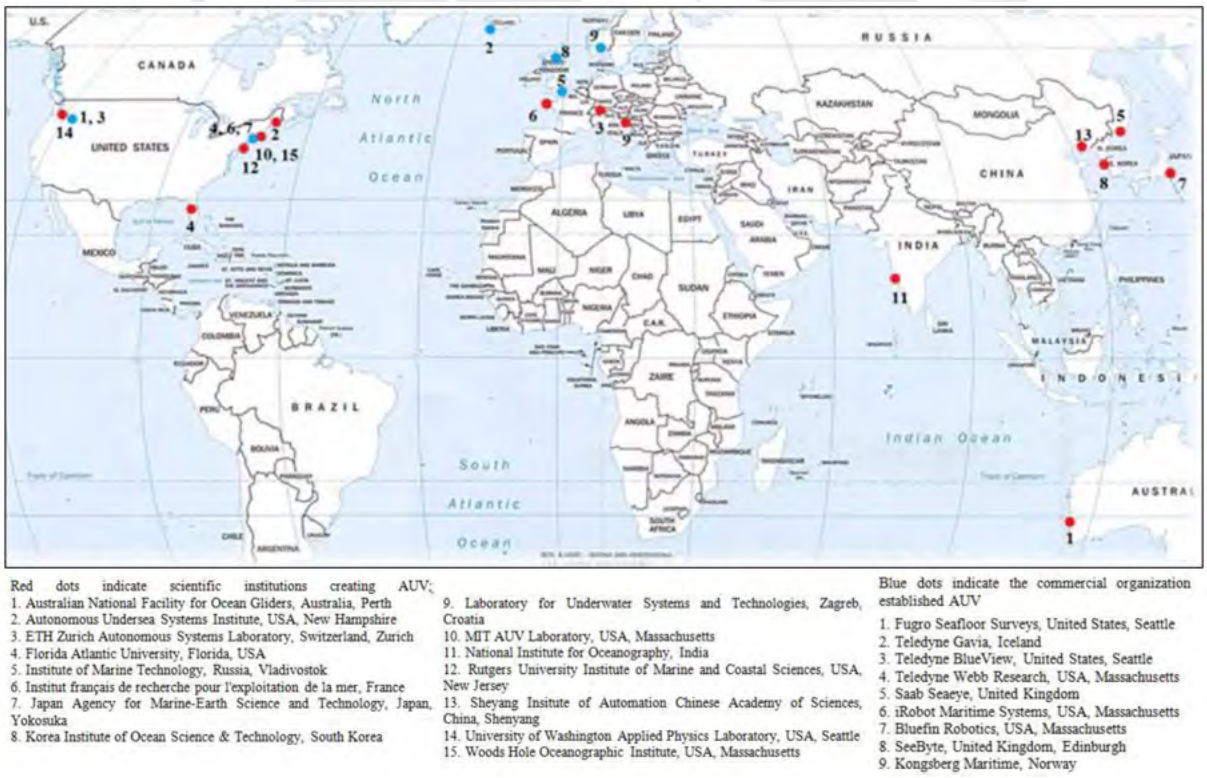
Figura 1.3: Modelos de ROV desarrollados para trabajos y observaciones submarinas, [Ostensjo, 2011]

B. Vehículo Sumbmarino Autónomo - AUV

Los Vehículos Submarinos Autónomos ó AUV (*Autonomous Underwater Vehicles*), son vehículos sumergibles que portan su propia fuente de energía, diversos sensores y una unidad de procesamiento ejecutando software y soluciones de control que les permiten desarrollar misiones sin la necesidad de intervención humana. Este tipo de equipos no

cuentan con un cable umbilical (*tether*) que limite su desplazamiento, por esta razón son utilizados principalmente para la exploración de extensas áreas de manera autónoma. El desarrollo de este tipo de vehículos ha tenido un constante crecimiento durante los últimos años, la **Fig. 1.5** muestra los principales centros de investigación y desarrollo en el mundo. Un ejemplo de este tipo de vehículos se muestra en la **Fig.1.4** el cual ha sido desarrollado por la *United Kingdom Natural Environment Research Council*, (NERC).

Figura 1.4: AUV Autosub 6000, UK Natural Environment Council (NERC), [Wynn et al., 2014]



Red dots indicate scientific institutions creating AUV;
1. Australian National Facility for Ocean Gliders, Australia, Perth
2. Autonomous Undersea Systems Institute, USA, New Hampshire
3. ETH Zurich Autonomous Systems Laboratory, Switzerland, Zurich
4. Florida Atlantic University, Florida, USA
5. Institute of Marine Technology, Russia, Vladivostok
6. Institut français de recherche pour l'exploitation de la mer, France
7. Japan Agency for Marine-Earth Science and Technology, Japan, Yokosuka
8. Korea Institute of Ocean Science & Technology, South Korea
9. Laboratory for Underwater Systems and Technologies, Zagreb, Croatia
10. MIT AUV Laboratory, USA, Massachusetts
11. National Institute for Oceanography, India
12. Rutgers University Institute of Marine and Coastal Sciences, USA, New Jersey
13. Sheyang Insitute of Automation Chinese Academy of Sciences, China, Shenyang
14. University of Washington Applied Physics Laboratory, USA, Seattle
15. Woods Hole Oceanographic Institute, USA, Massachusetts

Blue dots indicate the commercial organization established AUV
1. Fugro Seafloor Surveys, United States, Seattle
2. Teledyne Gavia, Iceland
3. Teledyne BlueView, United States, Seattle
4. Teledyne Webb Research, USA, Massachusetts
5. Saab Seaeye, United Kingdom
6. iRobot Maritime Systems, USA, Massachusetts
7. Bluefin Robotics, USA, Massachusetts
8. SeeByte, United Kingdom, Edinburgh
9. Kongsberg Maritime, Norway

Figura 1.5: Principales Centros de Investigación y Desarrollo de AUVs en el mundo, Gafurov and Klochkov [2015]

C. Sistema de Vehículo Manipulador Sumergible - UVMS

Existen situaciones donde los UUVs necesitan tener algún tipo de relación con el ambiente submarino para realizar operaciones mas especializadas que la simple inspección. Para este tipo de situaciones es necesario equipar el vehículo con uno o mas manipuladores que puedan ayudarlo en estas tareas. Esta situación es solucionada mediante la implementación de AUVs y ROVs con manipuladores con un alto grado de sofisticación para actuar en ambientes submarinos.

En el caso de los AUVs, al no contar con un cable umbilical, la implementación de manipuladores autónomos engloba muchos desafíos con respecto a la implementación de su sistema de control. Actualmente se han desarrollado equipos conocidos como Sistemas de Vehículo Manipulador Sumergible ó UVMS por sus siglas en inglés: *Underwater Vehicle-Manipulator System*. Un ejemplo de este tipo de vehículo es SAUVIM, proyecto desarrollado actualmente por la Universidad de Hawaii (**Fig. 1.6**).

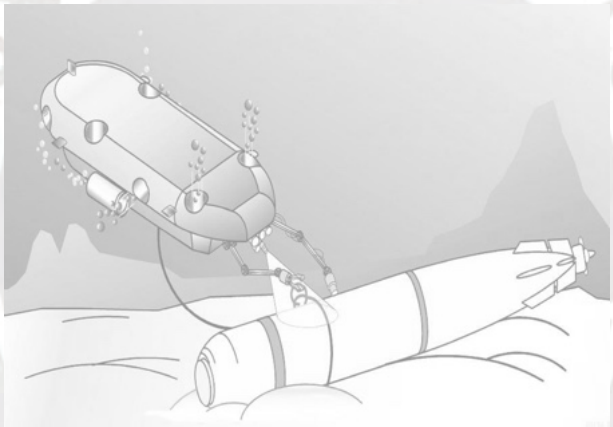
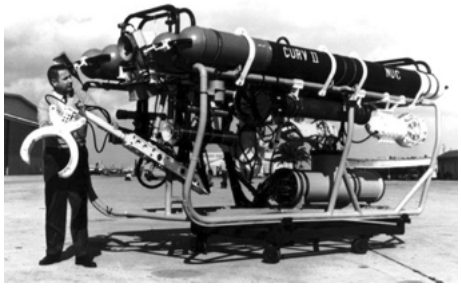


Figura 1.6: SAUVIM - Autonomous Systems Laboratory, Universidad de Hawaii, [Antonelli, 2006]

En el caso de los ROVs, la incorporación de manipuladores para operaciones complejas no ha sido un limitante desde los inicios de su desarrollo, esto debido a que las señales de control adicionales para el manipulador pueden enviarse también a través del cable umbilical. Actualmente, estos equipos incorporan dentro del *tether* cables de fibra óptica, los cuales gracias a su amplio ancho de banda, permiten la transmisión de señales de control y video a altas velocidades. Un ejemplo de este tipo de vehículos es el modelo CURV (*Cable-controlled Underwater Research Vehicle*) desarrollado con la finalidad de recuperar torpedos del fondo marino; uno de los grandes logros de este modelo fue la recuperación de una bomba atómica perdida en las costas de Palomares en España en 1966. La **Fig. 1.7** muestra el modelo hermano del CURV, el CURV II, y un modelo actual: FALCON, desarrollado para tareas de manipulación submarina.



(a) CURV II, VARE Industries, Christ and Wernli [2014]

(b) FALCON, SAAB Seaeye Ltd.

Figura 1.7: ROVs con Manipuladores incorporados

1.1.2. Aplicaciones Industriales

Actualmente los UUVs se utilizan en diferentes aplicaciones industriales, desde tareas simples como la inspección del entorno, hasta tareas mas sofisticadas que requieren un grado de sofisticación superior como la manipulación de objetos en el fondo marino. Dentro de las principales podemos detallar:

- **Ciencia**: En este sector la necesidad esta relacionada a la recopilación de información mediante sensores y toma de muestras para entender el entorno explorado. A menudo se utilizan exploraciones submarinas tripuladas para poder realizar estudios en entornos marinos, sin embargo los bajos costos de la implementación de UUVs hace que este tipo de equipos sean los mas usados para tele-operación submarina debido a su bajo costo comparado con exploraciones tripuladas, así como los bajos costos logísticos para su implementación. Las misiones típicas están relacionadas a la toma de muestras de entornos marinos.
- **Pesca y Acuacultura**: Debido al crecimiento de la población a nivel mundial y a la baja explotación de los recursos de los océanos, las granjas submarinas para crianza de peces son una opción que se ha desarrollado en los últimos años. El alcance de los trabajos de los UUV es la inspección del adecuado crecimiento de los peces, verificación de mortalidad y cualquier otra tarea relacionada a su adecuado crecimiento.
- **Militar**: La principal aplicación en el ámbito militar se enfoca en: desactivación de minas, recuperación de objetos y tareas de inspección. Para lograr esta misión se envía al UUV hacia objetivos previamente identificados. En el caso de las minas, una vez identificadas es necesario algún tipo de dispositivo adicional para poder

neutralizarla. En caso de recuperación de objetos, se necesita incluir equipos de izaje para poder extraer a la superficie los objetos encontrados. Y para el caso de las tareas de inspección solamente se incluye una cámara de video con sensores básicos.

- **Sector Hidrocarburos**: La implementación de UUVs como apoyo a las operaciones de perforación y producción de petróleo se ha convertido en una necesidad a medida que se realizan exploraciones en aguas mas profundas. A estas profundidades, el cabezal y las tuberías de perforación se han trasladado al fondo marino, por lo cual todas las tareas tienen que realizarse roboticamente. La misión típica implica la observación del medio ambiente bajo el mar, el montaje de juntas, guía de herramientas y equipos de perforación.
- **Inspección, Reparación y Mantenimiento**: Este mercado conocido como IMR por sus siglas del inglés *Inspection, Repair and Maintenance*, comprende las infraestructuras submarinas fijas o flotantes de varias industrias. Dentro de estas industrias tenemos las granjas eólicas, muelles, vehículos marinos, etc. Las misiones típicas incluyen el monitoreo y evaluación de las estructuras submarinas mediante procesos no destructivos que garantices la integridad de la misma. La **Fig. 1.8** muestra el vehículo para IRM diseñado e implementado por la empresa PEMEX.

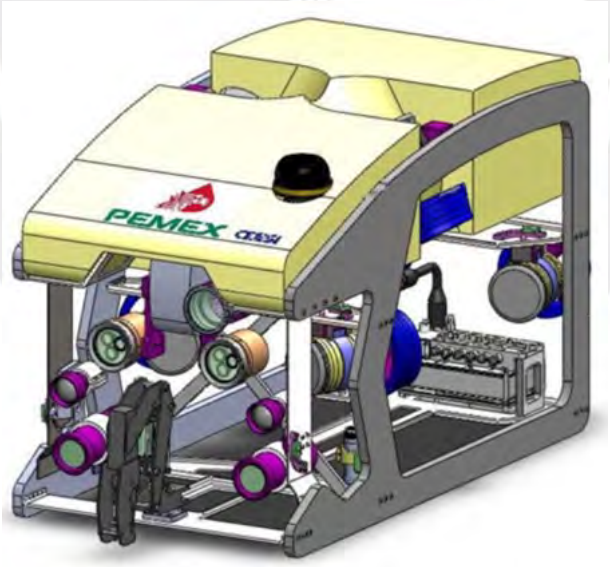


Figura 1.8: ROV para inspección en Instalaciones Petroleras Off-Shore, [Salgado et al., 2010]

1.2. Sistemas de Control de UUVs

Cuando se realiza una revisión de la literatura de los Vehículos Submarinos No Tripulados o UUVs, se puede observar que el término **Control** aborda una amplia gama de estudios de investigación. Özgür Yildiz et al. [2009] realizan la siguiente clasificación:

- Control de Movimiento: *Motion Control* - MC, estudios enfocados a la respuesta del vehículo a una entrada de control, así como a la estabilidad que presentan los UUVs frente a las perturbaciones.
- Control de Misiones: *Mission Control* - MiC, estudios enfocados en la ejecución de tareas determinadas de manera paramétrica.
- Control de Formación: *Formation Control* - FC, estudios enfocados en el comportamiento coordinado de múltiples UUVs.

La **estabilidad** de un UUV se define como la habilidad de retornar al estado de equilibrio después de una perturbación, a través del uso de propulsores o superficies de control, Fossen [1994]. Por otro lado, se define **maniobrabilidad** como la capacidad de un UVV para llevar a cabo ciertas maniobras predeterminadas. Una excesiva estabilidad implicaría un esfuerzo de control muy grande.

Se debe hacer una distinción entre la Estabilidad de Control Fijo (lazo abierto) y la Estabilidad de Control Libre (lazo cerrado), las diferencias principales se muestran a continuación Fossen [1994]:

- La estabilidad de Lazo Abierto implica la estabilidad del vehículo cuando las Superficies de Control son fijas y todos los propulsores tienen una potencia constante.
- La estabilidad en Lazo Cerrado implica La estabilidad del vehículo cuando las Superficies de Control y la potencia de los propulsores pueden variar. Esto implica que la dinámica del sistema de control debe ser considerada también en el análisis de estabilidad.

1.2.1. Sistemas de Control de Vehículos Submarinos

Shahrieel et al. [2015] enfatiza que el control de un UUV no es una tarea simple principalmente por las **no linealidades** presentes en su dinámica de movimiento y caracteres acoplados de las ecuaciones de la planta. Los UUVs son controlados mediante complejos algoritmos que usualmente requieren control automático de velocidad y posicionamiento, así como de sistemas para la ubicación adecuada de la profundidad

Tabla 1.1: Métodos de control clásico y sus limitaciones

Método de Control	**Limitaciones**
PID	No se puede realizar una compensación dinámica de las fuerzas hidrodinámicas del vehículo y de las perturbaciones desconocidas. La configuración de los parámetros es contradictoria entre el control y la velocidad de respuesta.
Control Deslizante (*Sliding Mode*)	Puede fácilmente llevar a una fluctuación en la precisión del efecto de control.
Control Difuso (*Fuzzy Logic*)	Las reglas difusas son de difícil sintonización. El tiempo de predicción debe ser suave.
Redes Neuronales (*Neural Network*)	No se pueden alcanzar los requerimientos de una respuesta rápida. Es de difícil implementación en aplicaciones de tiempo real.

y altitud sobre el lecho marino.

La dinámica de un UVV depende principalmente del ambiente que lo rodea. Las fuerzas del ambiente submarino afectan el movimiento del motor, los pequeños errores pueden afectan problemas de control ocasionando comportamientos indeseables del vehículo submarino; por esta razón, se deben tomar todas estas consideraciones en el diseño del control de un UVVs. Los sensores de presión y los sonares son los mas usados para el control de UVVs; en entornos con agua clara se puede utilizar cámaras para localización y navegación, sin embargo, en las profundidades del océano, estos sensores no se pueden utilizar debido a la baja iluminación y el ruido en la imagen producido por las partículas pequeñas.

Con la finalidad de cumplir con los requerimientos de control se han desarrollado diferentes técnicas que van desde control clásico, control avanzado, así como una combinación de las mismas con la finalidad de obtener una mejor *performance*. El sistema de control para un UUV tiene una alta complejidad debido a la hidrodinámica no lineal, inercia no lineal, y problemas relacionados al acoplamiento entre los grados de libertad (DoF). Muchos estudios ignoran algunas incertidumbres en los parámetros con el objetivo de reducir la dificultad en el diseño del controlador. Los métodos comunes de control usados en los UUVs así como sus limitaciones se muestran en la **Tabla 1.1**.

Autores como Tehrani et al. [2010] han desarrollado sistemas de control del tipo Proporcional/Integral/Derivativo - (PID), debido principalmente a la facilidad de su implementación.

Otros autores como Hassanein et al. [2011] han implementado controladores de Lógica

Difusa ó FLC (*Fuzzy Logic Controller*), los cuales representan a un sistema matemático que analiza los valores análogos de entrada en términos de variables lógicas entre 0 y 1 (falso o verdadero), este tipo de controlador es usado cuando no se cuenta con un modelo empírico o matemático del UUV; sin embargo, su principal inconveniente es la necesidad de procesadores rápidos y de alta performance. Por esta razón y con la finalidad de reducir el nivel de dificultad se han desarrollado controladores del tipo: Entrada Simple de Lógica Difusa (SIFLC - *Single Input Fuzzy Logic Controller*). El mayor beneficio del SIFLC es la reducción del sistema de MISO (*Multiple Input Single Output*) o MIMO (*Multiple Input Multiple Output*) a un sistema SISO (*Simple Input Simple Output*). Sin embargo, hasta la fecha, el SIFLC no ha sido investigado a profundidad en los UUV. La **Fig. 1.9** muestra un modelo de la estructura de control del SIFLC.

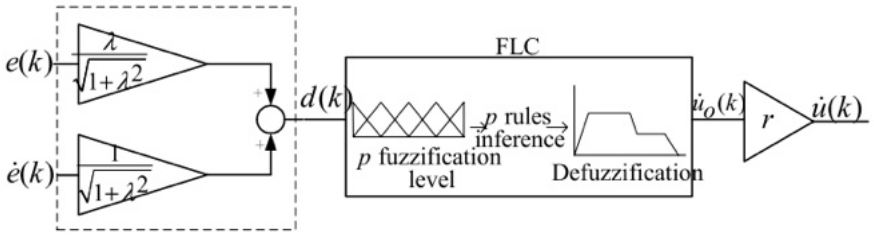


Figura 1.9: Estructura de Control del Tipo - SIFLC

Shi et al. [2006]; Ven et al. [2005] han usado sistemas de control de tipo Adaptativo para el control de UUVs. La ventaja de este tipo de control es su adaptabilidad a la variación de los parámetros dinámicos del UUV en el océano. En este caso, el tipo de controlador adaptativo puede adaptarse por si mismo a las perturbaciones del océano. Por ejemplo, mientras el UUV esta navegando hacia una mayor profundidad, la densidad del agua se incrementa lentamente, y el volumen del robot se reduce en cierta manera debido a la presión del agua; por lo cual se necesita una ley de control que se adapte por si misma a estas condiciones de cambio. Este tipo de controladores son también de gran utilidad debido a que los UUVs, especialmente los ROVs, son usualmente implementados con nuevos equipamientos como manipuladores o sistemas de visión y adaptados para diferentes misiones. La **Fig. 1.10** muestra el diagrama de bloques de un tipo de control por Redes Neuronales desarrollado por Ven et al. [2005].

Otra técnica que ha sido comúnmente usada para controlar los UUVs es el controlador del tipo Deslizante, SMC *Sliding Mode Controller* [Amer et al., 2012]. Este tipo de control ha sido ampliamente estudiado para controlar vehículos submarinos desde 1980. En Teoría de Control, el SMC es una forma de control de estructura variable VSC *Variable Structure Control* que altera la dinámica de un sistema no lineal con la aplicación de un control de conmutador de alta frecuencia *High-Frequency Switching Control*. Los resultados han

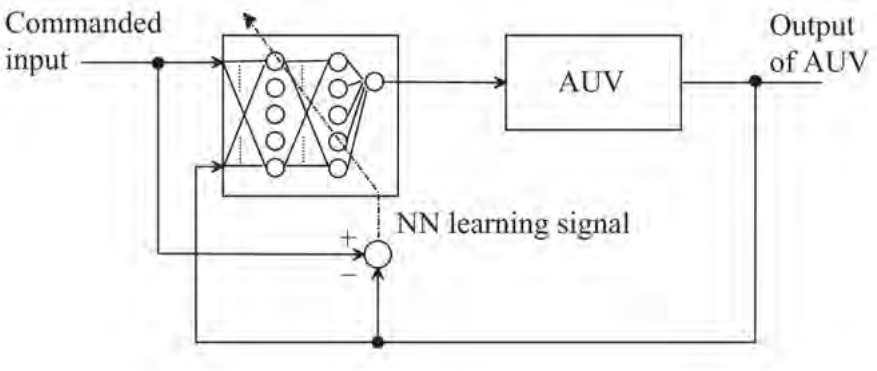


Figura 1.10: Esquema de Control con Redes Neuronales, [Ven et al., 2005]

mostrado que este tipo de control es práctico en términos de linealización, mejorando además el control de las características dinámicas del UUV. La **Fig. 1.11** muestra un tipo de control de trayectoria implementado en el ROV KAXAN.

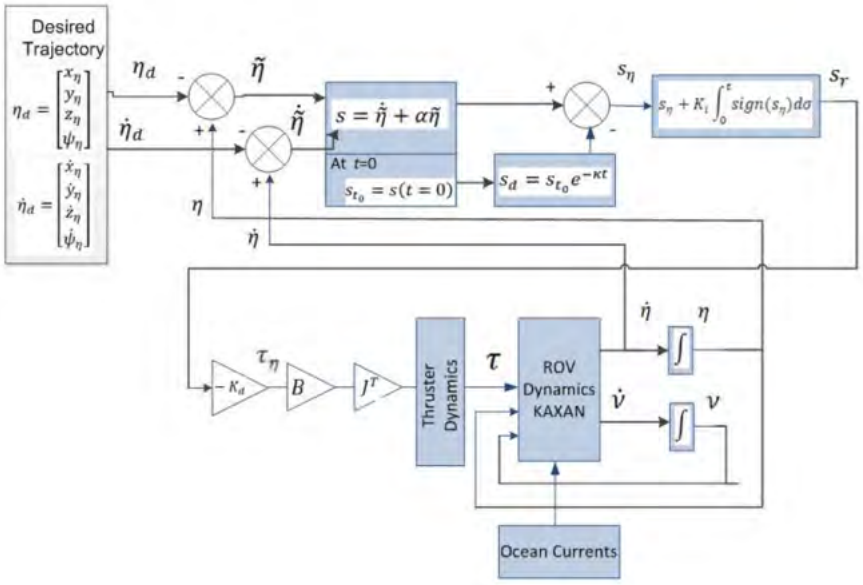


Figura 1.11: Control de trayectoria implementado mediante MF-HOSMC *Model Free High Order Sliding Model Control*

Actualmente, para el control de los UUV se utilizan las técnicas de control más frecuentes o una combinación de estas, ver **Tabla 1.2 y 1.3**. Un ejemplo son las técnicas que usan las Redes Neuronales combinadas con Lógica Difusa, llamadas Neuro-Fuzzy, o la combinación de la Lógica Difusa con un algoritmo de optimización de enjambre (PSO - *Particle Swarm Optimization Algorithm*, llamado Fuzzy-PMO.

Tabla 1.2: Tipos principales de controladores usados en UUVs, Watson [2012]

Tipo de Control	**UUV**
PID (y variaciones)	ARCS, ICTINEU, KwaZulu-Natal AUV, OBERON, ODIN, ORCA, REMUS, SPARUS, Subjugator, THETIS, ARIES, Phanton S2, UTM
Control Deslizante (y variaciones)	Benthos RPV-430, Hamburg ROV, Subjugator, OEX-C, EAVE, JASON, MUST, REMUS
Adaptativo (y variaciones)	ODIN, Manta-Ceresia, Taipan 2, R2D4

Tabla 1.3: Tipos de controladores alternativos usados en UUVs, Watson [2012]

Tipo de Control	**UUV**
Disturbance Compensation Scheme	NPS Phoenix
Nonlinear Gain Scheduling	INFANTE
Formation Control	SERAFINA
S-Surface/S-Plane	EAUV-XX, MAUV-II, OID-I
Lyapunov-based Tracking	Simulation only
HPSO-based Fuzzy Neural Network	National Key Lab AUV
Smith Control Scheme with LQG/LTR	ARGO
State-dependent Riccati Equation	REMUS
H2/H∞	Simulation only
Fuzzy	ARPA UUV
Robust Cascade	RRC
Cross-track Controller	C-SCOUT
Receding Horizon Tracking Control	Solo simulación
Multivariable Control using LQG/LTR	Solo simulación

1.2.2. Sistema de Control MPC en AUV

A diferencia del control SMC, los sistemas de control del tipo Modelo Predictivo Basado en Modelos - MPC (*Model Predictive Control*) y Control Óptimo, no han sido muy difundidos.

El algoritmo MPC, incluido dentro de los tipos de Control Óptimo, fue desarrollado en 1960 y posteriormente a partir de 1980 fue usado en procesos industriales como refinerías

de crudo y plantas químicas. Actualmente, el algoritmo MPC es el estándar de sistemas de control en procesos industriales debido a su capacidad para trabajar con sistemas MIMO de dimensiones superiores con restricciones de entrada, salida y estados dentro de la solución óptima.

La idea fundamental del MPC es utilizar un **modelo matemático** del sistema dinámico para predecir la trayectoria del sistema para entradas de control futuras. Usando esta propiedad, las entradas futuras óptimas pueden ser encontradas resolviendo un **problema de programación cuadrático** con restricciones de desigualdad. Una de las principales limitaciones del MPC, es el esfuerzo computacional requerido para encontrar la solución, especialmente durante los primeros años de su desarrollo, cuando no se contaba con una tecnología tan desarrollada. Sin embargo, debido al incremento de la eficiencia computacional, este problema ya no es tan relevante, y se ha logrado implementar este tipo de control para vehículos aéreos no tripulados (UAV - *Unmanned Aerial Vehicles*) con frecuencias de muestreo de 40 Hz.

Estos ejemplos de control de UUV no presentan el algoritmo MPC como un algoritmo viable para este trabajo. Sin embargo, el trabajo dentro de otras industrias similares presenta algunos resultados interesantes como la aeronáutica, donde los algoritmos MPC son utilizado para controlar la altitud de un helicóptero.

El control del tipo MPC tiene dos ventajas principales en comparación con otros algoritmos: la capacidad de "mirar-hacia adelante", que debería conducir a un control suave y óptima, y la posibilidad de encontrar una solución óptima dentro de las restricciones para la validez de la solución y mantener el sistema dentro de los límites. Este último punto es especialmente adecuado para vehículos submarinos no tripulados. En primer lugar, todos los actuadores mecánicos tienen limitaciones que se pueden manejar utilizando restricciones. En segundo lugar, debido a la amplia gama de condiciones de funcionamiento de este UUV, es probable que varíe significativamente el rendimiento de los diferentes actuadores y superficies de control.

1.3. Control Predictivo por Modelos (MPC)

El Control Predictivo por Modelos o Control Predictivo Basado en Modelos, en adelante MPC, por sus siglas en inglés *Model Predictive Control* es un tipo de control Óptimo. Este tipo de controladores se caracteriza porque las acciones de control se calculan mediante la optimización de una función de costo del modelo dinámico de la planta y que analiza el comportamiento del sistema en un estado futuro.

Camacho and Bordons [2010] presenta una serie de ventajas del control MPC sobre otros métodos, entre las principales se pueden citar las siguientes:

- Se usan para controlar procesos con dinámicas simples, complejas, con fase no mínima, inestables o multivariables.
- El modelo de predicción lo hace compensar los tiempos muertos del sistema.
- Se compensan las perturbaciones que pueden existir en el sistema debido a la operación del control predictivo.
- La ley del sistema de control es implementable de manera simple.
- Adecuado cuando el modelo permite conocer las referencias futuras, como en sistemas de robótica o procesos industriales en *batch*.
- Permite controlar las restricciones de una forma muy simple durante el diseño del sistema de control.

En base a lo indicado previamente, la implementación de este tipo de controladores significa un grado de dificultad elevado debido al análisis de las restricciones y cálculos del modelo dinámico en línea. Aún cuando la potencia que existe en los sistemas de control actuales permite desarrollar el sistema de control si problemas, esta restricción en cálculo computacional debe ser considerado.

A pesar del alto costo computacional, una de los beneficios mas importantes del control MPC es su apertura para considerar diferentes modelos de predicción y restricciones del sistema, esto permitió tener un gran impacto en el sector industrial. Es importante indicar que el MPC inició en el ámbito industrial, sin embargo la teoría fue desarrollada en el ámbito académico [Ramos, 2007].

1.3.1. Principio del MPC

Actualmente, el MPC se ha difundido progresivamente en el control de procesos industriales y una gama mucho mas amplia de aplicaciones, sobre todo gracias al aumento

constante de la velocidad y la potencia de cálculo, [Ramos, 2007]. Su principal ventaja sobre otras técnicas de es su capacidad para trabajar con restricciones en los estados y controlar las entradas del sistema, lo que permite una operación más eficiente.

Camacho and Bordons [2010] nos detalla a continuación la estrategia que siguen los controladores del tipo MPC, la cual es representada en la **Fig. 1.12**:

- Las salidas futuras se calculan para cada instante t, utilizando un modelo de la planta para un horizonte de predicción N. Las estimaciones de los valores de salida de la planta $y(t+k|t)$[1] para $k = 1...N$ están asociadas a los valores conocidos hasta el instante t (entradas y salidas conocidas) y a las señales de control $u(t+k|t), k = 0...N-1$, que se calculan y envían al sistema.
- Las señales de control futuras se estiman en base al criterio de reducir el error que existe entre el estado inicial y la trayectoria de referencia $w(t+k)$. Este criterio se expresa como una función cuadrática que mide el error entre la salida estimada y la trayectoria de referencias futura. Adicionalmente, se puede incluir el esfuerzo de control como parte de la función objetivo. La solución explícita se obtiene cuando el modelo es lineal y el criterio es cuadrático; caso contrario se utilizan métodos numéricos para determinar la solución.
- La señal de control calculada $u(t|t)$ se aplica a la planta mientras que las otras señales calculadas no se consideran, pues en el instante siguiente de muestreo $y(t+1)$ es conocida y los pasos previos se repiten con este nuevo valor. Por lo que la señal de control $u(t+1|t+1)$ se calcula con información diferente y en principio sería también diferente a $u(t+1|t)$.

La **Fig. 1.13** muestra el diagrama para el sistema de control MPC. Tal como se puede observar, se usa un modelo de la planta para estimar el comportamiento de la salida en base a las señales de entrada y salidas conocidas. La acciones de control en un estado futuro se calculan con el optimizador, que toma como datos de cálculo la función del coste y las posibles restricciones.

Tal como se puede observar, el modelo del proceso (o la planta) es un factor principal para el controlador. El modelo usado debe tener la dinámica precisa de la planta para que el controlador pueda predecir de manera precisa la evolución del sistema.

Otra parte fundamental del control MPC es el optimizador ya que permite estimar las acciones de control que se deben aplicar a la planta. Se puede obtener una solución

[1]Esta notación hace referencia al valor estimado de la variable y en el instante $t+k$ calculada en el instante t

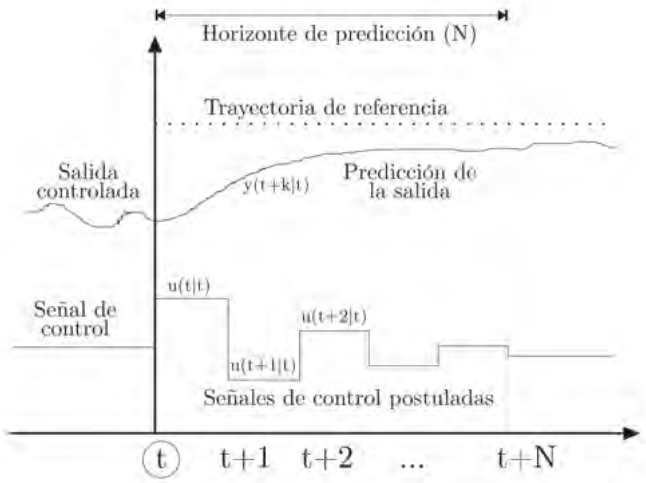


Figura 1.12: Predicción de salida y señal de control postulada, [Ramos, 2007]

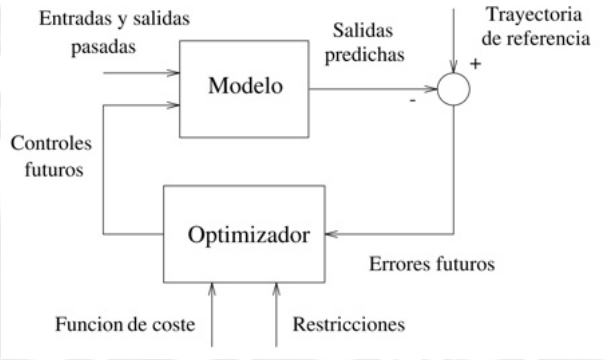


Figura 1.13: Estructura básica MPC, [Camacho and Bordons, 2010]

explícita si la función de coste es cuadrática, se tiene el modelo lineal de la planta y no existen restricciones asociadas al sistema. Si estas condiciones no se cumplen, se requeriría un algoritmo de optimización numérico que requiere mayor costo computacional.

La estrategia de control predictivo es similar a la estrategia al conducir un auto. El conductor tiene conocimiento de la trayectoria de referencia deseada para un horizonte de control finito. Tomando en consideración las características del automóvil (características físicas) decide generar una acción de control (frenos, acelerador, volante, cambios) para seguir la trayectoria deseada. El conductor sólo aplica la primera acción de control de la secuencia calculada mentalmente, y repite el procedimiento en los sucesivos instantes. El esquema se representa en la **Fig. 1.14**.

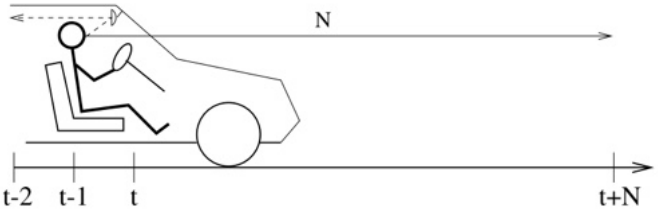


Figura 1.14: Analogía MPC con manejar un automóvil, Camacho and Bordons [2010]

1.3.2. Enfoques de Solución MPC

El uso del tipo de control MPC para sistemas dinámicos tiene dos diferentes enfoques de solución: las soluciones fuera de línea (*Offline*) y en línea (*Online*). Estos enfoques difieren en cómo y cuándo se resuelve el problema. A continuación, se presenta una breve explicación de ambos enfoques.

A. Offline MPC

La solución MPC fuera de línea (*Offline MPC*) también conocida como MPC Explícita (*Explicit MPC*) se caracteriza por realizar la optimización previo a la operación [Tondel et al., 2001]. En esta solución, el espacio de estados es dividido en un poliedro con áreas y la acción de control óptimo es computada como una función explícita para cada área. La ley de control explícita se almacena como una tabla de consulta, esto permite reducir el esfuerzo computacional en línea a solo localizar el estado inicial actual en el conjunto respectivo y evaluar una función explícita de este estado. La ubicación del estado (también llamada ubicación del punto) representa el esfuerzo principal y su complejidad depende de cómo se haya dividido el espacio de estados.

B. Online MPC

La solución MPC en línea (*Online MPC*) se ejecuta cuando el sistema está en operación, por lo cual el problema MPC se resuelve en tiempo real en cada tiempo de muestreo, esto implica que el tiempo computacional total para obtener la solución debe ser menor que el tiempo de muestreo del sistema ΔT. Debido a la complejidad del problema y al esfuerzo computacional involucrado, es necesario el uso de solucionadores eficientes, que puedan proporcionar una solución rápida y que permitan la aplicación de MPC a sistemas dinámicos rápidos. Los algoritmos de solución tradicionales para MPC en línea son los métodos de punto interior (*Interior Point*) y conjunto activo. Sin embargo, estos métodos son generales y no explotan las características inherentes de MPC para

obtener una solución más eficiente. Por lo tanto, el MPC en línea representa hoy en día un tema de investigación actual y se han propuesto una variedad de métodos de solución, especialmente para el MPC no lineal.

1.3.3. Control Predictivo por Modelo Lineal

En caso de que el sistema a controlar sea lineal, o se considere una linealización alrededor de un punto de operación, la función de costo es cuadrática y tanto los estados como las restricciones de las entradas de control son lineales, entonces el problema de MPC es un problema de LMPC (*Linear Model Predictive Control*).

Las dos últimas condiciones normalmente se satisfacen fácilmente ya que la función de costo se elige para que sea cuadrática y las restricciones del sistema suelen aparecer como restricciones de caja (límites simples). La ventaja de esta formulación es que el problema de optimización se puede formular como un problema de QP (*Quadratic Programming*), que es convexo y, por lo tanto, asegura que se puede encontrar un mínimo global.

Para analizar la formulación del LMPC debemos de considerar un sistema discreto invariante en el tiempo:

$$\begin{aligned} x_{k+1} &= A_k x_k + B_k u_k \\ y_k &= C_x k \end{aligned} \tag{1.1}$$

Donde $x_k \in R_n, u_k \in R_m$ y $y_k \in R_p$ representan los estados, entradas y salidas del sistema respectivamente. Adicionalmente, $x_k = x(k) = x(t = kT_s)$, para k = 0,1,2,... siendo T_s la representación del periodo de muestreo.

El algoritmo LMPC realiza el cómputo del control óptimo $u*$ sobre un horizonte de control predefinido (N_u) requerido para seguir una referencia y predecir los futuros estados sobre un horizonte de predicción (N_p). Con el objetivo de encontrar la solución óptima, se tiene que minimizar en cada instante k la siguiente función de costo:

$$J(k) = \frac{1}{2}\left(\sum_{i=1}^{N} ||x(k+i|k) - x_r(k+i|k)||^2_{Q_i} + \sum_{i=0}^{N-1} ||u(k+i|k) - u_r(k+i|k)||^2_{R_i}\right) \tag{1.2}$$

Donde x_r y u_r son los vectores de referencia para los estados de la planta y las entradas de control respectivamente. La expresión $||x||^2_Q$ representa la forma cuadrática de $x^T Qx$ y la expresión $x(k+i|k)$ denota los valores estimados para los estados en el instante $k+i$

usando la estimación en el instante k, lo mismo aplica para las entradas u. Teniendo en cuenta la dinámica del sistema y las restricciones, en cada nueva medición se resuelve el siguiente problema de optimización:

$$\begin{aligned} minimize_{u,x} J(k) = & \frac{1}{2}\sum_{i=1}^{N} ||x(k+i|k) - x_r(k+i|k)||^2_{Q_i} + \\ & \frac{1}{2}\sum_{i=0}^{N-1} ||u(k+i|k) - u_r(k+i|k)||^2_{R_i} \end{aligned} \tag{1.3}$$

Tomando en cuenta:

$$\begin{aligned} x(k|k) &= x_0, \\ x(k+i+1|k) &= A_i x(k+i|k) + B_i u(k+i|k), \qquad i = 0, 1, ..., N-1, \\ D_i x(k+i|k) &\leq d_i, \qquad i = 1, 2, ..., N, \\ F_i u(k+i|k) &\leq f_i, \qquad i = 0, 1, ..., N-1. \end{aligned} \tag{1.4}$$

Estas ecuaciones consideran el caso general de un sistema lineal variable en el tiempo, donde el subíndice en las matrices A y B indican su valor en el horizonte de predicción para el instante i, a partir del período de muestreo k. Así mismo se incluyen restricciones sobre los estados y las entradas de control.

Para el caso del LMPC el problema de optimización a resolver en cada intervalo de muestreo puede ser formulado como un problema de Programación Cuadrática (QP, *Quadratic Programming*) bajo la forma:

$$minimize_{\xi} J(k) = \frac{1}{2}\xi^T Q \xi + q^T \xi, \tag{1.5}$$

Considerando:

$$A\xi = b, C\xi \leq d. \tag{1.6}$$

Donde el vector ξ representa la optimización de variables y su estructura dependen del enfoque de la solución. Para que el problema sea convexo, la matriz Q debe ser positiva semidefinida.

1.3.4. Control Predictivo por Modelo No Lineal

La mayoría de los modelos de sistemas en la vida real son inherentemente no lineales, aunque se utilizan muchos modelos linealizados debido a su simplicidad. Sin embargo, los sistemas complejos como los vehículos presentan muchos desafíos, como restricciones cinemáticas no holonómicas, estados de control altamente acoplados y comportamiento no lineal. En esta situación, un modelo lineal claramente no es adecuado para describir el proceso.

Como se indicó previamente el control MPC es un tipo particular de Problema de Control Óptimo (OCP - *Optimal Control Problem*), que se puede definir como:

$$\min_{u(t)} \quad \Psi\left(\mathbf{x}\left(t_f\right), t_f\right) + \int_{t_0}^{t_f} L(\mathbf{x}(t), \mathbf{u}(t), t)dt \tag{1.7a}$$

$$\text{subject to} \quad \dot{\mathbf{x}} = f(\mathbf{x}(t), \mathbf{u}(t), t), \quad t \in [t_0, t_f] \tag{1.7b}$$

$$g(\mathbf{x}(t), \mathbf{u}(t), t) \leq \mathbf{0} \tag{1.7c}$$

$$\mathbf{x}\left(t_0\right) = \mathbf{x}_0 \tag{1.7d}$$

Donde $x(t)$ y $u(t)$ son las variables de estado y control, x_0 en (1.7d) son los valores de estado inicial, y el intervalo $[t_0, t_f]$ es el horizonte de predicción en el esquema MPC (normalmente fijo). En este problema, (1.7a) es la función objetivo que consiste en un término integral $L(x(t), u(t), t)$, llamado término de Lagrange, y un término terminal $(x(t_f), t_f)$, llamado término de Mayer. Las ecuaciones diferenciales en (1.7b) representan el modelo del vehículo en el intervalo $[t0, tf]$. Los estados y las restricciones de control se describen en (1.7c).

Diehl [2007]; Betts [2010] indican que los enfoques de solución para los OCP normalmente se dividen en tres grupos principales: programación dinámica, métodos indirectos y directos:

- **Programación dinámica**: convierte el OCP en una ecuación de Hamilton-Jacobi-Bellman mediante el cálculo recursivo del control de retroalimentación. Este enfoque está restringido a pequeñas dimensiones de estado debido a la dimensionalidad de Bellman.
- **Métodos indirectos**: hacen uso de conceptos del cálculo de variaciones como el principio máximo de Pontryagin y las ecuaciones diferenciales de Euler Lagrange para transformar el OCP en un problema de valor límite (BVP). Luego, este BVP se resuelve numéricamente. Por las razones explicadas anteriormente, este

método también se conoce como primero optimizar, luego discretizar. El principal inconveniente es la dificultad para derivar el BVP, que se vuelve aún más complejo cuando se introducen restricciones.

- **Métodos directos**: convierten el OCP de tiempo continuo en un problema de NLP de dimensión finita mediante el uso de una técnica de discretización. Este método también se describe como primero discretizar, luego optimizar. En la actualidad, los métodos directos son los más utilizados para resolver OCP restringidos; aunque el NLP resultante es grande, existen solucionadores de optimización de última generación que pueden resolver este tipo de configuraciones de manera muy eficiente, [Betts, 2010; Kirk, 2004]. En la presente tesis, se utilizó disparo múltiple, que es un método directo, para discretizar el problema NMPC en un NLP.

1.3.5. Métodos directos para resolver problemas de control óptimo

En la última sección, se explicó cómo los métodos directos convierten un problema de optimización dinámica en una programación no lineal (NLP) de dimensión finita, y por qué se prefieren sobre las otras metodologías.

La reformulación se logra a través de la discretización del problema de control óptimo, que se representa como un problema de valor inicial (IVP). Todos los métodos directos parametrizan primero la trayectoria de control (generalmente en un patrón por partes); sin embargo, varían en la forma en que se gestiona la parametrización de los estados.

Existen dos enfoques diferentes para los métodos directos: enfoque secuencial y enfoque simultáneo. El enfoque secuencial es una integración secuencial de estados (simulación) y la optimización de la trayectoria del control se ejecuta en cada iteración. La trayectoria del estado $x(t)$ se define como una función implícita de los controles $u(t)$, lo que significa que el solucionador solo optimiza la trayectoria de los controles. Los métodos directos de disparo único (*single shooting*) pertenecen a esta categoría. Por otro lado, está el enfoque simultáneo, en el que los estados y los controles se introducen como variables de optimización. Las variantes más populares del enfoque simultáneo son la colocación global (*Global Allocation*) y el disparo múltiple (*Multiple Shooting*).

A. Disparo Único

Conocido como *single shooting*, este método genera inicialmente una cuadrícula de puntos de tiempo t_k, para $k = 0, ..., N$. Los controles $u(t)$ se discretizan a lo largo de los intervalos de tiempo generados, generalmente como valores constantes por partes. Luego, el problema

se resuelve integrando secuencialmente los estados (simulación) y optimizando para la trayectoria de los controles $u(t)$. La solución define una trayectoria $\bar{x} = x(t; \bar{u})$ para cada secuencia de control $\bar{u}$. Los beneficios de este método son su simplicidad y los pocos grados de libertad generados, incluso para grandes sistemas ODE o DAE, [Diehl, 2007]. Sin embargo, este enfoque presenta problemas con sistemas inestables, debido a la falta de información para los estados iniciales.

B. Colocación ortogonal en elementos finitos

También llamados métodos pseudoespectrales, [Rao, 2010], la colocación en elementos finitos es un caso particular de las ecuaciones implícitas de Runge-Kutta. Para un intervalo de tiempo dado ΔT (que en el caso específico de MPC, es la frecuencia de muestreo del sistema), las trayectorias de los estados se aproximan utilizando polinomios de colocación, como se ve en las ecuaciones (1.8). Generalmente, las trayectorias de los controles se suponen constantes por partes. Los puntos de colocación ortogonal, denominados t_k, para $k = 0, ..., M$ se pueden obtener a partir de las cuadraturas de Legendre, Radau o Lobatto, [Correa, 2016]. Para asegurar la continuidad de la EDO, se dan restricciones de igualdad entre los valores finales e iniciales de cada intervalo posterior.

$$\begin{aligned} x_i^{(l)}(t) &= \sum_{k=0}^{N} x_{ik}^{(l)} L_{lk}(t), \quad i = 1 \dots n \\ \text{Donde,} \quad L_{lk}(t) &= \prod_{\substack{\nu=0 \\ v \neq kl}}^{N} \left[\frac{t - t_\nu}{t_{lk} - t_\nu} \right] \end{aligned} \tag{1.8}$$

Después de la discretización, se obtiene un NLP muy grande, pero disperso. Una ventaja de este método es que el NLP disperso se puede resolver de manera eficiente mediante solucionadores numéricos que explotan estas estructuras particulares. Además, los métodos de colocación pueden lidiar con sistemas inestables y tienen una convergencia local rápida. Algunos trabajos han utilizado con éxito la colocación para resolver problemas NMPC, [Koutrik, 2015], Huang [2010].

C. Disparo múltiple directo

Conocido como *Multiple Shooting*), este método numérico fue desarrollado por Bock y Plitt alrededor de los años 80 [Bock and Plitt, 1984]. El principio de funcionamiento en el disparo múltiple es similar al disparo único. Se genera inicialmente una cuadrícula de puntos de tiempo tk, para $k = 0, ..., N$, y los controles $u(t)$ se discretizan a lo largo de los intervalos de tiempo. La diferencia con el disparo único se basa en cómo se realiza

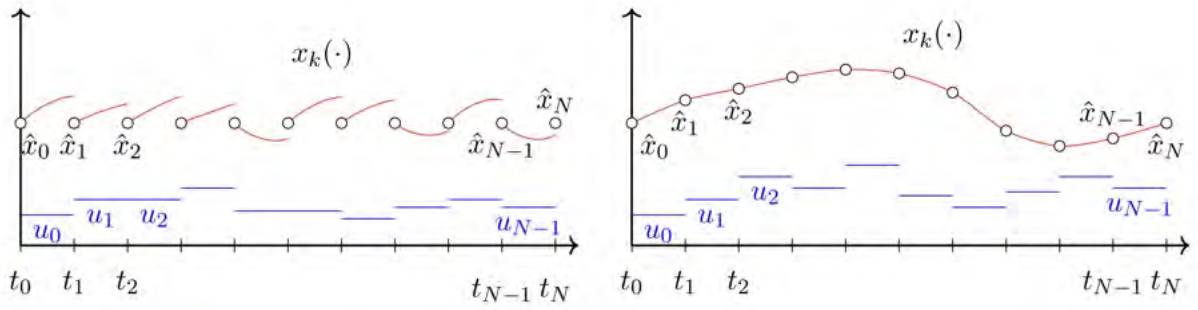


Figura 1.15: (Izquierda) Trayectorias individuales obtenidas a través de la solución de las EDO. (Derecha) Convergencia de los perfiles de estado y control para el método de disparo múltiple directo. Extraído de Correa [2016]

la integración en cada subintervalo. En el disparo múltiple, la EDO se integra (simula) independientemente en cada subintervalo con diferentes valores iniciales, designados por una nueva variable $\hat{x}_k$ (ver ecuación (1.9) y **Fig. 1.15** (izquierda)).

$$\begin{aligned} \dot{x}_k(t) &= f\left(x_k(t), u_k\right), \quad t \in [t_k, t_{k+1}], \\ x_k\left(t_k\right) &= \hat{x}_k \end{aligned} \tag{1.9}$$

De ahí la denominación de disparos "múltiples"(y aproximación simultánea), porque cada intervalo "disparaïndependientemente. La continuidad de la dinámica se satisface mediante las restricciones residuales en la igualdad, que conecta los intervalos discretizados, [Koutrik, 2015].

$$\hat{x}_{k+1} = x_k\left(t_{k+1}, \hat{x}_k, u_k\right) \tag{1.10}$$

Después de la convergencia del solucionador de optimización, las trayectorias de los estados pueden verse como en la **Fig. 1.15** (derecha).

Las ventajas de este método son el NLP disperso resultante y que se puede utilizar la información de los estados iniciales. Por lo tanto, este método puede manejar sistemas inestables con restricciones de ruta y terminal. En términos de velocidad, el disparo múltiple compite e incluso puede superar al enfoque de colocación, [Diehl, 2007]. En este trabajo, se aplicará el disparo múltiple para resolver el BVP de los modelos dinámicos.

1.3.6. Selección del solucionador NLP

Como se explicó en la sección anterior, un problema de optimización puede transformarse en un NLP para utilizar los solucionadores eficientes disponibles. Estos solucionadores utilizan diferentes algoritmos para abordar el problema, que pueden clasificarse en tres: programación cuadrática secuencial (SQP), métodos de punto interior (IPM) y métodos heurísticos, [Betts, 2010; Nocedal and Wright, 1999; Messac, 2015]. SQP e IPM se encuentran en la mayoría de las aplicaciones, mientras que los métodos heurísticos se utilizan para casos más particulares [2]. Dependiendo de las características particulares del problema, un enfoque puede ser más adecuado que el resto.

[Nocedal and Wright, 1999] indica que las principales consideraciones para la selección de un solucionador son: tamaño del problema, número de restricciones, grado de no linealidad y tiempo de CPU. Además, se debe tener en cuenta la disponibilidad y calidad de la documentación, así como su escalabilidad sin cambios importantes. Por último, una circunstancia particular en este proyecto fue que el solucionador tenía que ser compatible con el entorno del sistema operativo, de modo que pudiera interactuar con un software desarrollado previamente.

En base a la revisión de la información encontrada en Koutrik [2015]; Leyffer and Mahajan [2010], en este trabajo se eligió el solucionador Interior Point Optimizer (IpOpt) el cual es explicado en el trabajo de Wachter and Biegler [2006]. IpOpt es un solucionador de NLP conocido basado en el enfoque IPM. Puede resolver de manera eficiente problemas de optimización no lineal dispersos a gran escala. También está bien documentado y se puede integrar con diferentes solucionadores lineales para mejorar sus capacidades en trabajos futuros. Además, se distribuye como código abierto bajo la Licencia Pública Eclipse (EPL) de forma gratuita, incluso para aplicaciones comerciales.

IpOpt solicita al usuario una función objetivo, límites de variables, restricciones, gradiente de la función objetivo y el jacobiano de las restricciones. También se puede proporcionar el hessiano del lagrangiano, o se puede aproximar mediante el algoritmo Broyden–Fletcher–Goldfarb–Shanno (BFGS) integrado en IpOpt. En este trabajo, el gradiente y el jacobiano se generaron utilizando CasADI, [Andersson, 2013], que es un marco simbólico (*Framework* para la diferenciación automática y el control óptimo.

[2] Los métodos heurísticos implican enfoques no convencionales como redes neuronales y algoritmos evolutivos o genéticos

A. IpOpt

El optimizador de Punto Interior o conocido como IpOpt por sus siglas en inglés: Interior Point OPTimizer, es un paquete de software para optimización no lineal con grandes beneficios en problemas de gran escala, Wachter and Biegler [2006]. El solucionador fue desarrollado inicialmente por Andreas Wächter como parte de su investigación de tesis doctoral, pero ha sido mejorado a lo largo de los años, gracias a su licencia de código abierto. En el núcleo del solucionador, se utiliza un algoritmo de punto interior primal-dual acoplado con un método de filtro de búsqueda de línea. IpOpt ha sido escrito en C++ y ha demostrado un buen rendimiento para problemas de optimización en línea (Correa [2016], Verschueren et al. [2016]). Como se ha mencionado anteriormente, las ventajas de IpOpt son su compatibilidad con múltiples solucionadores lineales, buena documentación, distribución gratuita, integración con plataformas como MATLAB®y Modelica®, y su disponibilidad para diferentes sistemas operativos. IpOpt también ha sido utilizado por otros investigadores para aplicaciones de conducción autónoma, con resultados bastante buenos (Correa [2016], Verschueren et al. [2016], Koutrik [2015], Verschueren et al. [2016], Rosolia et al. [2017]).

B. CasADI

CasADi es un marco simbólico (*Symbolic Framework* para la diferenciación algorítmica y la optimización numérica, desarrollado por Joel Andersson y Joris Gillis, [Andersson, 2013; Andersson et al., 2011]. El usuario puede definir fácilmente expresiones simbólicas utilizando el sistema algebraico computacional (CAS, *Computer Algebra System*); luego, sus derivadas se pueden obtener utilizando la diferenciación algorítmica de última generación en modos directo e inverso y técnicas de coloración de gráficos como jacobianos y hessianos dispersos. CasASI proporciona interfaces en Python y MATLAB® y también se puede integrar con solucionadores de última generación como Sundials (CVODES, IDAS y KINSOL), IpOpt, WORHP, SNOPT y KNITRO. También vale la pena mencionar que CasADI es de código abierto, está escrito en código C++ autónomo y está bien documentado con información en línea.

1.3.7. Ventajas y Desventajas del Control MPC

El MPC ha demostrado ser muy eficiente para controlar sistemas muy complejos y ha superado a las estrategias de control típicas que se han utilizado durante muchos años en la industria. En comparación con las técnicas de control tradicionales, como los controladores PID, el tipo de control MPC ofrece las siguientes ventajas:

- Es posible especificar las limitaciones deseadas en el proceso (considerando restricciones de control y de estado), así como el comportamiento deseado a través de la función objetivo empleada en la formulación del problema. Esta característica facilita el diseño y ajuste del controlador.
- El control MPC puede manejar problemas de control a gran escala de sistemas dinámicos con múltiples entradas y salidas (sistemas MIMO).
- Para el control de seguimiento de referencia, el control MPC minimiza el error de seguimiento al cambiar la entrada de control antes de un cambio de punto de ajuste debido a su característica predictiva.
- Se reduce la propagación del ruido de medición a través de la señal de control.También se logra la compensación de perturbaciones debido a la característica de retroalimentación proporcionada por la técnica de horizonte deslizante.

Además de todas estas características, el control MPC se basa en una teoría bien establecida. Diferentes estudios sobre estabilidad y robustez respaldan el uso de esta técnica de control. Además, los estudios extensivos sobre herramientas de optimización matemática utilizadas para resolver problemas de control óptimo han aumentado en las últimas décadas, dando como resultado diferentes solucionadores robustos que se pueden utilizar en aplicaciones generales de MPC. Sin embargo, todas estas ventajas se producen a expensas de la complejidad en la resolución del problema de control óptimo resultante, lo que hace que el uso de MPC sea muy desafiante en áreas donde el tiempo es un factor crítico, como la navegación autónoma. En particular, MPC se vuelve desafiante debido a las siguientes razones:

- Cuando la dinámica es no lineal (NMPC), el problema de optimización es generalmente no convexo. Por lo tanto, en algunos casos, puede que no sea posible obtener una solución global óptima, sino muchas soluciones locales subóptimas. Esto hace que el problema sea muy difícil de resolver, lo que implica un mayor esfuerzo computacional y, por lo tanto, más tiempo para obtener la solución óptima.
- Los sistemas con dinámicas rápidas (lineales o no lineales) requieren la solución del problema de optimización en tiempo real, es decir, dentro de intervalos de tiempo en el rango de milisegundos o incluso microsegundos. Por lo tanto, es necesario calcular la solución del problema en el tiempo t_k lo más rápido posible (dentro del tiempo de muestreo ΔT) para obtener la entrada de control óptima u en el tiempo t_{k+1}.

1.4. Redes Neuronales Profundas

El sueño de crear ciertas formas de inteligencia que nos imiten a nosotros mismos existe desde hace mucho tiempo. Si bien la mayoría de ellos aparecen en la ciencia ficción, en las últimas décadas hemos ido progresando gradualmente en la construcción de máquinas inteligentes que pueden realizar ciertas tareas como un humano. Esta es un área llamada inteligencia artificial (IA) y hoy en día ya se considera parte de una nueva revolución industrial.

La inteligencia artificial utiliza redes neuronales para poder implementar tareas complejas, estas redes tratan de imitar el comportamiento del cerebro humano. Dentro de la disciplina de la inteligencia artificial, el desarrollo de redes profundas han ido incrementándose durante los últimos años, sobre todo por su capacidad para aprender automáticamente la representación de características en información con diferentes niveles de abstracción. Esto permite que las redes neuronales profundas aprendan funciones con un elevado grado de complejidad sin muchas dependencias de características creadas por humanos. Además, proporciona la capacidad de entrenamiento preliminar, que consiste en el aprendizaje de la representación en un conjunto de datos disponibles y luego aplicar las relaciones aprendidas a otros dominios. Esto puede tener algunas limitaciones, como la posibilidad de adquirir datos de calidad suficientemente buena para el aprendizaje.

El modelo de aprendizaje profundo, es decir, la red neuronal profunda a menudo consta de múltiples capas, que en conjunto trabajan para construir un espacio de funciones mejorado. La primera capa tiene como función el aprendizaje de las características de primer orden, alguna de ellas tan lógicas como el color, bordes o siluetas. La segunda capa aprende características de orden superior, como esquinas, atenuaciones u otras. La tercera capa aprende sobre características mas complejas como pequeños parches, texturas, relaciones, etc.

Con la aparición de procesadores de mayor potencia y conjuntos de datos a gran escala (Big Data), el aprendizaje profundo se ha convertido en un pilar del mundo tecnológico actual y se ha aplicado en una amplia gama de campos.

1.4.1. Historia y surgimiento de las Redes Neuronales Profundas

La primera red neuronal se desarrolló en la década de 1940, poco después de los albores de la investigación de la IA. En 1943, se publicó un artículo fundamental llamado Ün cálculo lógico de ideas inmanentes en la actividad nerviosa", que proponía las características del primer modelamiento matemático de una red neuronal. La unidad mas simple de este

modelo es una neurona formalizada simple, denominada neurona McCulloch-Pitts. Esta red neuronal es expresada como una función matemática concebida como un modelo de neuronas biológicas.

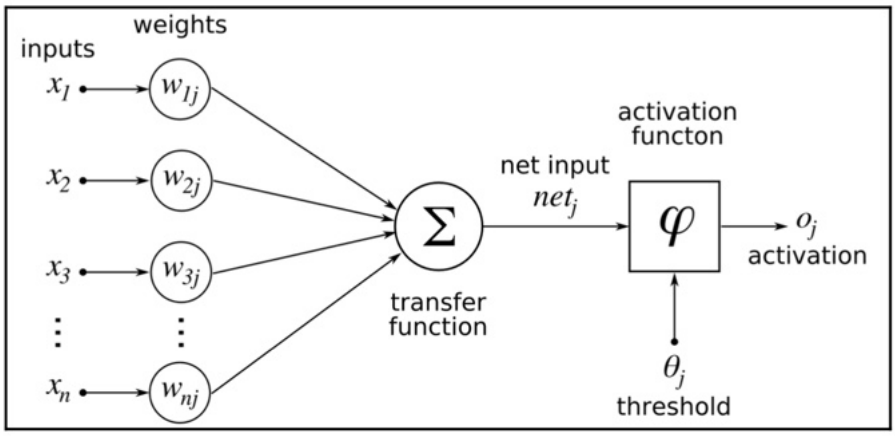


Figura 1.16: Neurona Artificial, Fuente: *https://commons.wikimedia.org/wiki*

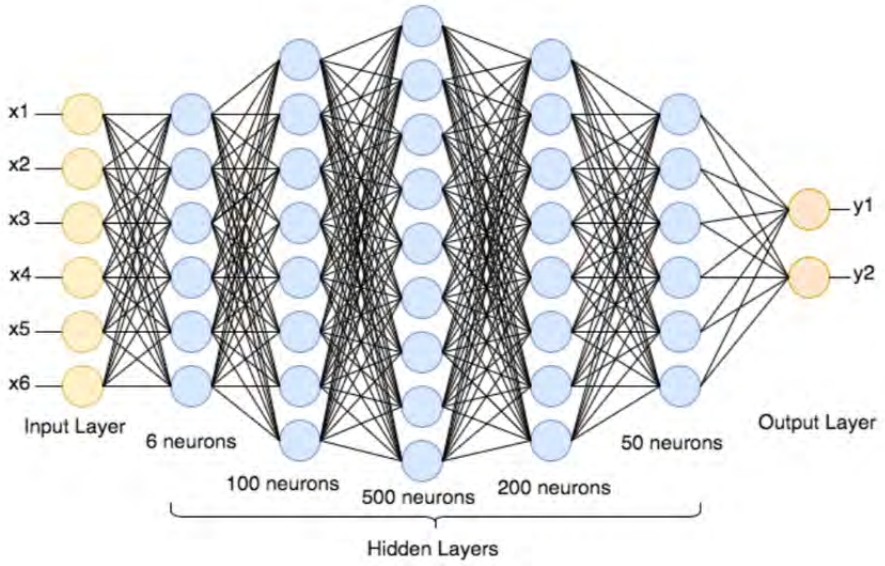


Figura 1.17: Arquitectura de una red neuronal profunda

Estos primeros modelos consisten en solo un conjunto muy pequeño de neuronas virtuales y se usa un número aleatorio llamado pesos para conectarlas. Estos pesos determinan cómo cada neurona simulada transfiere información entre ellas, es decir, cómo responde cada neurona, con un valor entre 0 y 1.

En la década de 1960 se propusieron por primera vez los conceptos de retropropagación y el uso de errores en el entrenamiento de modelos de aprendizaje profundo. A mediados de la década de 1980, Hinton y otros ayudaron a despertar el interés en las redes neuronales con

los llamados modelos profundos que hicieron un mejor uso de muchas capas de neuronas, es decir, con más de dos capas ocultas. Hinton y sus coautores demostraron que la retropropagación en un la red neuronal podría resultar en una distribución representativa interesante. En 1989, Yann LeCun demostró el primer uso práctico de la retropropagación en Bell Labs. Llevó la propagación hacia atrás a las redes neuronales convolucionales para comprender los dígitos escritos a mano, y su idea finalmente se convirtió en un sistema que lee el número de cheques escritos a mano.

En la última década, muchos investigadores lograron algunos avances conceptuales fundamentales, y hubo un repentino estallido de interés en el aprendizaje profundo, no solo del lado académico sino también de la industria. En 2006, el profesor Hinton de la Universidad de Toronto en Canadá y otros desarrollaron una forma más eficiente de enseñar capas individuales de neuronas, llamada Un algoritmo de aprendizaje rápido para redes profundas. En su artículo, presentó Deep Belief Networks (DBN), con un algoritmo de aprendizaje que entrena con avidez una capa a la vez mediante la explotación de un algoritmo de aprendizaje no supervisado para cada capa, una máquina de Boltzmann restringida (RBM).

El DBN propuesto se probó utilizando la base de datos MNIST, la base de datos estándar para comparar la precisión y exactitud de cada método de reconocimiento de imágenes. Esta base de datos incluye 70,000, 28 x 28 píxeles, imágenes de caracteres escritas a mano de números del 0 al 9 (60,000 es para entrenamiento y 10,000 es para prueba). El objetivo es responder correctamente qué número del 0 al 9 está escrito en el caso de prueba. Aunque el artículo no atrajo mucha atención en ese momento, los resultados de DBM tuvieron una precisión considerablemente mayor que un enfoque de aprendizaje automático convencional.

Desde entonces, el aprendizaje profundo ha despegado, y hoy vemos muchas aplicaciones exitosas no solo en clasificación de imágenes, sino también en regresión, reducción de dimensionalidad, modelado de texturas, reconocimiento de acciones, modelado de movimiento, segmentación de objetos, recuperación de información, robótica, procesamiento de lenguaje natural, reconocimiento de voz, campos biomédicos, generación de música, arte, filtrado colaborativo, etc.

Es interesante señalar que la mayoría de los avances teóricos ya se habían logrado en las décadas de 1980 y 1990, sin embargo, el éxito del aprendizaje profundo en la actualidad es en gran parte un éxito de la ingeniería. De hecho, el procesamiento mas rápido, con imágenes de procesamiento de GPU, aumentó la velocidad de cálculo 1000 veces en un lapso de 10 años.

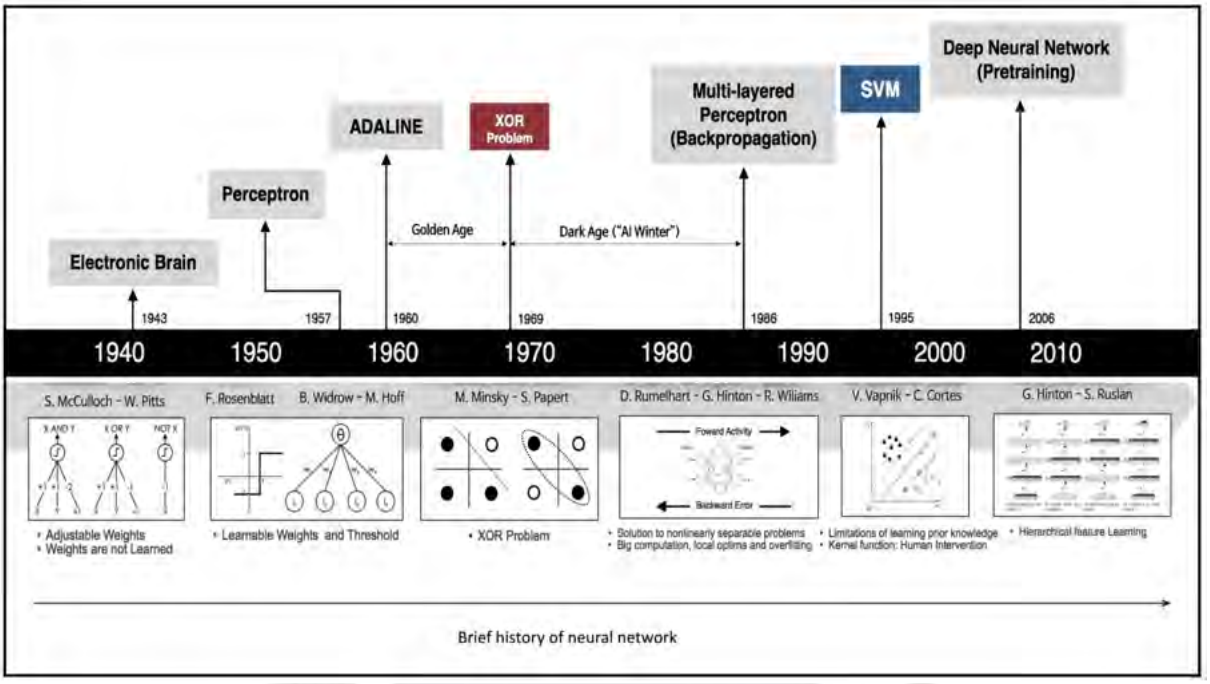


Figura 1.18: Historia de Deep Learning / Inteligencia Artificial

Casi al mismo tiempo, llegó la era del Big Data. Todos los días se recopilan millones, miles de millones o incluso billones de bytes de datos. Los líderes de la industria también están haciendo un esfuerzo en el aprendizaje profundo para aprovechar las enormes cantidades de datos que han recopilado. Con suficientes datos de entrenamiento y una velocidad computacional mas rápida, las redes neuronales ahora pueden extenderse a una arquitectura profunda, algo que nunca antes se había realizado. Por un lado, la aparición de nuevos enfoques teóricos, datos masivos y computación rápida han impulsado el progreso en el aprendizaje profundo. Por otro lado, la creación de nuevas herramientas, plataformas y aplicaciones impulsó el desarrollo académico, el uso de GPU mas rápidas y potentes y la recopilación de Big Data.

1.4.2. Redes Neuronales

Una red neuronal está conformada por un conjunto de neuronas simples conectadas por nodos y cuya funcionalidad esta basada en el comportamiento de las neuronas animales. La capacidad de procesamiento de estas redes está asociada a las fuerzas que conectan estas unidades, también conocidas como peso, y que son obtenidas mediante un proceso de aprendizaje sobre un conjunto de patrones de información como parte de un entrenamiento, Aleksander [1990].

Las neuronas artificiales imitan el comportamiento de las neuronas biológicas mediante las

entradas y salidas, aunque su funcionamiento es diferente. La estructura de una neurona artificial j está conformado por los siguientes componentes:

- La función de entrada in_j
- La función de activación f
- El valor θ_j umbral
- La función de salida f_{out}

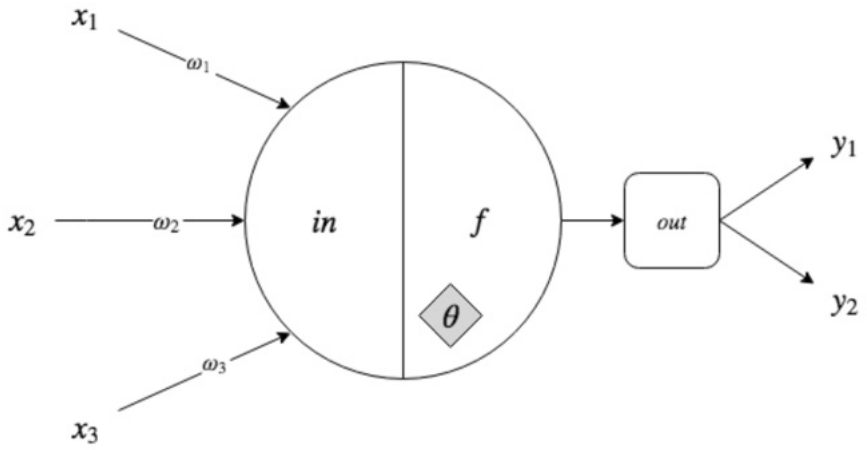


Figura 1.19: Esquema Neurona Artificial

Tal como se muestra en la Figura 1.19, la función de entrada in_j realiza el cálculo de las entradas x_n multiplicadas por un peso w_n y realiza la operación para obtener una única entrada. Estas operaciones son $\sum_{i=1}^{n} x_i w_i$ y $\prod_{i=1}^{n} x_i w_i$ o bien el máximo de las entradas $Max(x_i w_i)$.

La función de activación, representada por f, calcula el valor de salida de la neurona tomando en cuenta el resultado de la función de entrada menos el valor de umbral θ, los estados de activación obtenidos están en un rango entre (-1, 1) o (0, 1). Algunos ejemplos de función de activación se pueden observar en la Figura 1.20.

La función de salida `out` determina el valor obtenido en la salida de la neurona y que entrará en las siguientes conexiones. Es usual que tome la función identidad, sin embargo en algunas ocasiones se usa para acotar la salida dentro de un rango de valores.

Las neuronas ubicadas en la capa de entrada no tienen predecesoras, pues actúan sólo como una interfaz de entrada de la red. Así mismo, las neuronas de salida no tienen sucesoras, actuando sólo como interfaz de las neuronas que están en la capa de salida de la red.

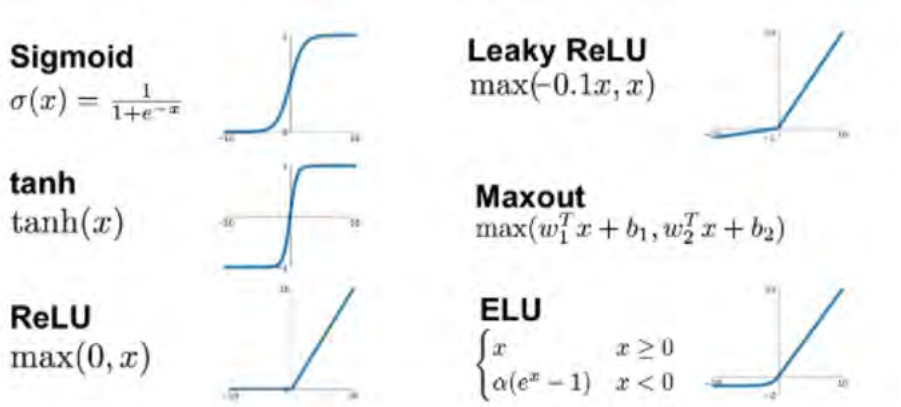


Figura 1.20: Señales de Activación

La red neuronal está conformada por un número determinado de neuronas que se distribuyen en campas formando una red en capas o niveles, tal como se puede ver en la Figura 1.17. Estas capas se clasifican en tres tipos:

- Capa de Entrada: esta capa recibe la información de ingreso que viene de la zona externa de la red.
- Capas Ocultas: son las capas ubicadas en la zona interna de la red y que no tiene contacto con el entorno exterior.
- Capa de Salida: toma la información que fue generada por las capas predecesoras y transmite la información de salida al exterior de la red.

El aprendizaje de la red se realiza en la etapa de entrenamiento. En esta etapa la red aprende mediante la modificación de los pesos de las conexiones entre neuronas. Para la etapa de aprendizaje se requiere definir una función de costo $C : F \to \mathbb{R}$ de manera que para la solución óptima f^* tal que $C(f^*) \leq C(f) \forall f \in F$, es decir, ninguna solución puede tener un valor de la funcieon de costo superior a la solución óptima.

El proceso de aprendizaje se realiza en tres fases:

- Forward propagation: la capa de entrada es alimentada con datos que tienen etiquetas y los resultados son proporcionados por la capa de salida.
- Calculo de la función de costo: se realiza el cálculo del error entre el valor entregado a la red y el valor objetivo.
- Backpropagation: a partir de los resultados del análisis de la función de costo, se ejecuta el algoritmo de backpropagation [Rumelhart et al., 1986] el cual tiene como objetivo determinar las neuronas que han generado una repuesta incorrecta para

ajustar sus pesos con la finalidad de reducir el error.

1.4.3. Proceso de aprendizaje

La clave de la etapa de entrenamiento de redes neuronales está en utilizar una adecuada función de pérdida o `loss function`, [Goodfellow et al., 2016]. Esta función permite medir el error de los valores de las predicciones $\hat{y}$ respecto de los valores verdaderos y. En la etapa de entrenamiento se realiza la minimización de este valor en base a la actualización de los parámetros w con el objetivo principal de mejorar la precisión.

La función de pérdida utilizada para el entrenamiento de redes neuronales es la *Cross Entropy Loss*, debido a su utilidad en problemas de clasificación con problemas de clases excluyentes. La función viene dada por:

$$H(y, \hat{y}) = \sum_x y_x log \frac{1}{\hat{y_x}} = -\sum_x y_x log \hat{y_x} \tag{1.11}$$

El proceso de entrenamiento, indicado previamente, es en realidad un problema de optimización, el cual utiliza el método del descenso del gradiente estocástico o *Stochastic Gradient Descent*, indicado por [Mei et al., 2018], un método iterativo para optimizar una función objetivo diferenciable. Se denomina estocástico debido a lo aleatorio en la toma de muestras, dentro de las técnicas de descenso del gradiente estocástico se usa el algoritmo de optimización de Adam (*Adaptative Moment Estimation*) [Kingma and Ba, 2014].

Uno de los parámetros importantes de la red neuronal es el *learning rate* o también conocido como la tasa de aprendizaje. Este parámetro determina el descenso del gradiente para obtener el siguiente punto, siendo w_j un parámetro, f la función de pérdidas a minimizar y α el learning rate.

$$w_j = w_j - \alpha \frac{\partial f(w_j)}{\partial w_j} \tag{1.12}$$

Si la tasa de aprendizaje es pequeña, la convergencia del gradiente se realizaría de manera muy lenta, y si es demasiado grande, nunca se podría llegar a converger, e incluso diverger del dato deseado, como se puede observar en la **Fig. 1.21**.

El descenso del gradiente o *Gradient Descent* es un proceso que se realiza de manera iterativa con el objetivo de actualizar los pesos de la red en cada una de las iteraciones. Debido a las iteraciones se vuelve necesario procesar el dataset en mas de una ocasión. Para realizar esto, se divide todo el dataset en partes llamadas `batch`, las cuales tienen

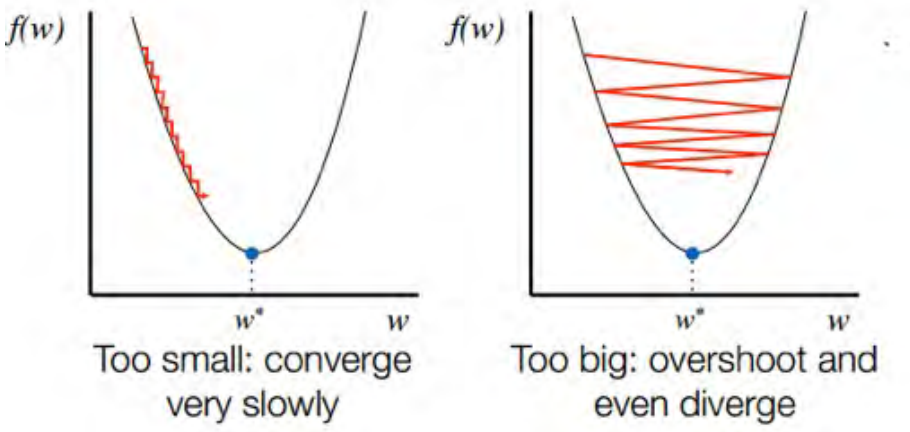


Figura 1.21: Efecto de diferentes tasas de aprendizaje

un determinado tamaño que permita optimizar el tamaño de memoria. Cada vez que se procesa todo el dataset de manera completa se llama época (*epoch*). Como se puede observar en la **Fig. 1.21** una función de dos parámetros converge más rápidamente, al punto de menor coste, cuanto mayor es el tamaño del batch.

Para realizar un entrenamiento completo se realiza un entrenamiento con varias de estas épocas, y al final de de ellas se ejecuta una etapa de validación. La etapa de validación tiene como objetivo presentar un conjunto de datos que no han sido validados previamente. A partir de estos datos se calcula la función de pérdidas `loss` y la precisión del aprendizaje. Si no existe una no mejora luego de ejecutar cierto numero de épocas, el entrenamiento se concluye.

CAPÍTULO 2

MODELADO DEL VEHÍCULO SUBMARINO AUTÓNOMO

El modelamdo de un UUV se puede dividir en dos pasos: Cinemática, que se enfoca en todos los aspectos geométricos del movimiento, y Dinámica, que analiza las fuerzas y los momentos que causan el movimiento. En este capítulo elaboramos las ecuaciones cinemáticas y dinámicas del movimiento del AUV, y en base a las cuales establecemos el modelo de la planta a la cual se le aplicará el modelo de control

2.1. Sistema de Coordenadas

El movimiento de un UVV puede ser descrito con seis grados de libertad - 6 DoF (*Degrees of Freedom*), debido a que se necesitan 6 coordenadas independientes para determinar la posición y orientación de un cuerpo rígido submarino. Los 6 componentes de movimiento son definidos como avance ($surge$), deriva ($sway$), arfada ($heave$), balanceo ($roll$), cabeceo ($pitch$) y guiñado (yaw), como se puede observar en la **Tabla 2.1** y en la **Fig. 2.1**.

Tabla 2.1: Notaciones usadas en vehículos marinos

DoF	Descripción	Fuerzas y Momentos	Velocidad angular y lineal	Posición y ángulos de Euler
1	Movimiento en dirección x, avance ($surge$)	X	u	x
2	Movimiento en dirección y, deriva ($sway$)	Y	v	y
3	Movimiento en dirección z, arfada ($heave$)	Z	w	z
4	Rotación en eje x, balanceo ($roll$)	K	p	ϕ
5	Rotación en eje y, cabeceo ($pitch$)	M	q	θ
6	Rotación en eje z, guiñada (yaw)	N	r	ψ

Cuando se analiza el movimiento de un AUV de 6 DoF, es conveniente definir los ejes de coordenadas. El eje de coordenadas $X_b\ Y_b\ Z_b$ está fijado al AUV (Body-fixed Frame) y tiene el origen 0,0,0 coincidente con el Centro de Gravedad (CG) del vehículo. El movimiento del

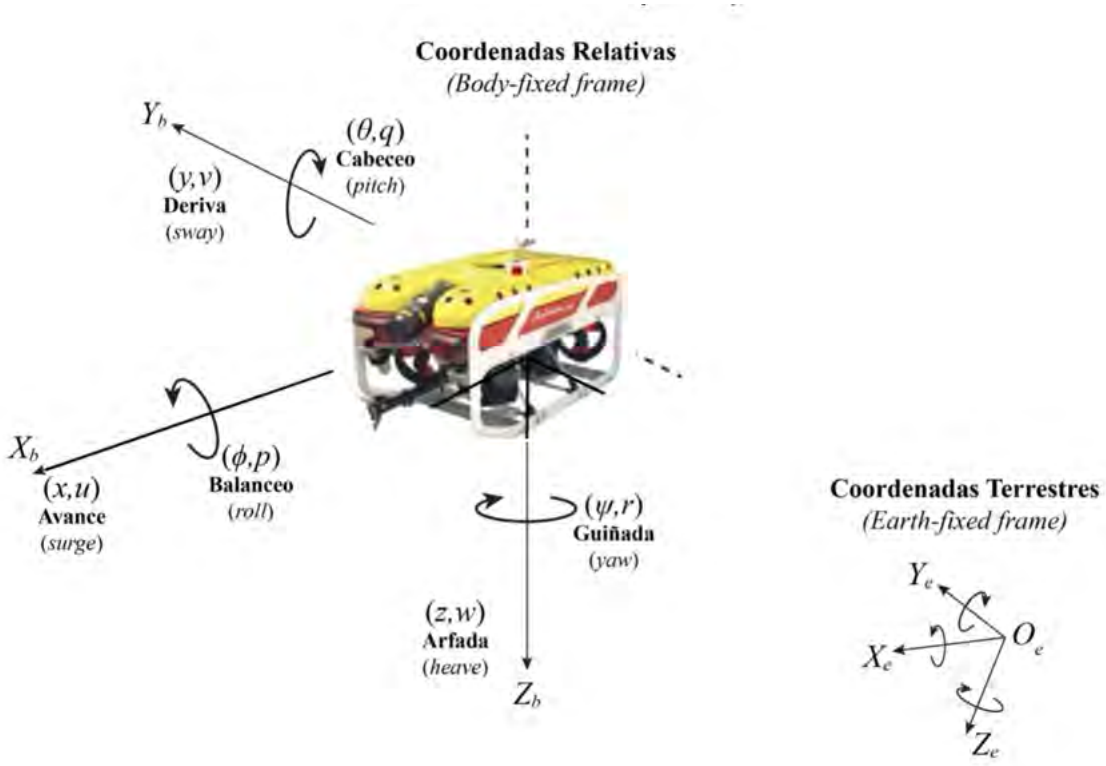


Figura 2.1: Sistema de coordenadas del vehículo submarino, Fossen [1994]

eje fijo es relativo a la estructura de referencia inercial (Earth-fixed Frame). Para vehículos submarinos se puede considerar que la aceleración de un punto en la superficie de la tierra es despreciable debido sus bajas velocidades. Como resultado, el eje de coordenadas X_e Y_e Z_e puede considerarse inercial.

Fossen [1994] describe el movimiento de los vehículos submarinos de 6 DoF según los vectores indicados en la **Tabla 2.1**, :

$$\begin{aligned}
&\eta = [\ \eta_1^T\ ,\ \eta_2^T\ ]^T; \quad \eta_1 = [\ x,\ y,\ z\ ]^T; \quad \eta_2 = [\ \phi,\ \theta,\ \psi\ ]^T \\
&\nu = [\ \nu_1^T\ ,\ \nu_2^T\ ]^T; \quad \nu_1 = [\ u,\ v,\ w\ ]^T; \quad \nu_2 = [\ p,\ q,\ r\ ]^T \\
&\tau = [\ \tau_1^T\ ,\ \tau_2^T\ ]^T; \quad \tau_1 = [\ X,\ Y,\ Z\ ]^T; \quad \tau_2 = [\ K,\ M,\ N\ ]^T
\end{aligned} \tag{2.1}$$

De la ecuación 2.1, η indica el vector de posición y orientación, ν indica el vector de velocidad angular y lineal y τ describe el vector de fuerzas y momentos actuando en el vehículo, todos estos parámetros en base al eje de coordenadas fijo ($Earth-fixed\ Frame$).

2.2. Cinemática

La primera transformación de coordenadas relaciona las **velocidades traslacionales** del eje del cuerpo fijo del vehículo ($Body - fixes\ Frame$) y las coordenadas del eje inercial de la tierra ($Earth - fixed\ Frame$), se muestra a continuación:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = J_1(\eta_2) \begin{bmatrix} u \\ v \\ w \end{bmatrix} \tag{2.2}$$

Donde:

$$J_1(\eta_2) = \begin{bmatrix} cos\psi cos\theta & -sen\psi cos\phi + cos\psi sen\theta sen\phi & sen\psi sen\phi + cos\psi sen\theta cos\phi \\ sen\psi cos\theta & cos\psi cos\phi + sen\psi sen\theta sen\phi & -cos\psi sen\phi + sen\psi sen\theta cos\phi \\ -sen\theta & cos\theta sen\phi & cos\theta cos\phi \end{bmatrix} \tag{2.3}$$

Notar que $J_1(\eta_2)$ es ortogonal:

$$(J_1(\eta_2))^{-1} \ = \ (J_1(\eta_2))^T \tag{2.4}$$

La segunda transformación de coordenadas relaciona las velocidades entre el cuerpo del vehículo y las coordenadas fijas de la tierra:

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \ = \ J_2(\eta_2) \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{2.5}$$

Donde:

$$J_2(\eta_2) \ = \ \begin{bmatrix} 1 & sen\phi tan\theta & cos\phi tan\theta \\ 0 & cos\phi & -sen\phi \\ 0 & sen\phi/cos\theta & cos\phi/cos\theta \end{bmatrix} \tag{2.6}$$

Notar que $J_2(\eta_2)$ no está definida en el ángulo de cabeceo ($pitch$) $\theta = \pm 90°$. Esto no es problema, debido a que el movimiento del vehículo no alcanza esta singularidad de manera natural u ordinaria.

2.3. Dinámica

Fossen [1994] realiza una amplia deducción de las ecuaciones de movimiento para un vehículo marino, llegando finalmente a la expresión general determinada por la ecuación:

$$M\ \dot{\nu} + C(\nu)\ \nu + D(\nu)\ \nu + g(\eta) = \tau_E + \tau \tag{2.7}$$

A su vez, los componentes de esta ecuación se describen mediante las siguientes ecuaciones:

$$M = M_{RB} + M_A \tag{2.8}$$

$$C(\nu) = C_{RB}(\nu) + C_A(\nu) \tag{2.9}$$

$$D(\nu) = Dp(\nu) + Ds(\nu) + Dw(\nu) + D_M(\nu) \tag{2.10}$$

Donde:

M	:	Matriz de Inercia Total
M_{RB}	:	Matriz de Inercia del Cuerpo Rígido (*Rigib Body Inertial Matrix*)
M_A	:	Matriz de Inercia de Masa Agregada (*Added Mass Inertial Matrix*)
$C(\nu)$	:	Matriz de Coriolis y Centrípeta Total
$C_{RB}(\nu)$	:	Matriz de Coriolis y Centrípeta del Cuerpo Rígido
$C_A(\nu)$	:	Matriz Hidrodinámica de Coriolis y Centrípeta
$D(\nu)$	:	Matriz de Atenuación Hidrodinámica (*Damping matrix*)
$D_P(\nu)$	:	Matriz de Atenuación
$D_S(\nu)$	:	Fricción de superficie
$D_W(\nu)$	:	Atenuación por ondas
$D_M(\nu)$	:	Atenuación por efecto vortex
$g(\eta)$	:	Fuerzas y momentos gravitacionales
τ	:	Fuerzas de control
τ_ξ	:	Fuerzas del entorno

Cada una de los parámetros se define a continuación:

Matriz de Inercia del Cuerpo Rígido

$$M_{RB} = \begin{bmatrix} mI_{3x3} & -mS(r_G) \\ mS(r_G) & I_o \end{bmatrix} = \begin{bmatrix} m & 0 & 0 & 0 & mz_G & -my_G \\ 0 & m & 0 & -mz_G & 0 & mx_G \\ 0 & 0 & m & my_G & -mx_G & 0 \\ 0 & -mz_G & my_G & I_x & -I_{xy} & -I_{xz} \\ mz_G & 0 & -mx_G & -I_{yx} & I_y & -I_{yz} \\ -my_G & mx_G & 0 & -I_{zx} & I_{zy} & I_z \end{bmatrix} \tag{2.11}$$

Matriz de Coriolis y Centrípeta del Cuerpo Rígido

$$C_{RB}(\nu) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ -m(y_G q + z_G r) & m(y_G p + w) & m(z_G p - v) \\ m(x_G q - w) & -m(z_G r + x_G p) & m(z_G q + u) \\ m(x_G r + v) & m(y_G r - u) & -m(x_G p + y_G q) \end{bmatrix} \begin{bmatrix} m(y_G q + z_G r) & -m(x_G q - w) & -m(x_G r + v) \\ -m(y_G p + w) & m(z_G r + x_G p) & -m(y_G r - u) \\ -m(z_G p - v) & -m(z_G q + u) & m(x_G p + y_G q) \\ 0 & -I_{yz} q - I_{xz} + I_z r & I_{yz} r + I_{xy} p - I_y q \\ I_{yz} q + I_{xz} - I_z r & 0 & -I_{xz} r - I_{xy} q + I_x p \\ -I_{yz} r - I_{xy} p + I_y q & I_{xz} r + I_{xy} q - I_x p & 0 \end{bmatrix} \tag{2.12}$$

Matriz de Inercia de Masa Agregada

$$M_A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = -\begin{bmatrix} X_{\dot{u}} & X_{\dot{\nu}} & X_{\dot{w}} & X_{\dot{p}} & X_{\dot{q}} & X_{\dot{r}} \\ Y_{\dot{u}} & Y_{\dot{\nu}} & Y_{\dot{w}} & Y_{\dot{p}} & Y_{\dot{q}} & Y_{\dot{r}} \\ Z_{\dot{u}} & Z_{\dot{\nu}} & Z_{\dot{w}} & Z_{\dot{p}} & Z_{\dot{q}} & Z_{\dot{r}} \\ K_{\dot{u}} & K_{\dot{\nu}} & K_{\dot{w}} & K_{\dot{p}} & K_{\dot{q}} & K_{\dot{r}} \\ M_{\dot{u}} & M_{\dot{\nu}} & M_{\dot{w}} & M_{\dot{p}} & M_{\dot{q}} & M_{\dot{r}} \\ N_{\dot{u}} & N_{\dot{\nu}} & N_{\dot{w}} & N_{\dot{p}} & N_{\dot{q}} & N_{\dot{r}} \end{bmatrix} \tag{2.13}$$

Matriz Hidrodinámica de Coriolis y Centrípeta

$$C_A(\nu) = \begin{bmatrix} 0 & 0 & 0 & 0 & -a_3 & a_2 \\ 0 & 0 & 0 & a_3 & 0 & -a_1 \\ 0 & 0 & 0 & -a_2 & a_1 & 0 \\ 0 & -a_3 & a_2 & 0 & -b_3 & b_2 \\ a_3 & 0 & -a_1 & b_3 & 0 & -b_1 \\ -a_2 & a_1 & 0 & -b_2 & b_1 & 0 \end{bmatrix} \tag{2.14}$$

Donde:

$$\begin{aligned} a1 &= X_{\dot{u}} u + X_{\dot{\nu}} \nu + X_{\dot{w}} w + X_{\dot{p}} p + X_{\dot{q}} q + X_{\dot{r}} r \\ a2 &= X_{\dot{\nu}} u + Y_{\dot{\nu}} \nu + Y_{\dot{w}} w + Y_{\dot{p}} p + Y_{\dot{q}} q + Y_{\dot{r}} r \\ a3 &= X_{\dot{w}} u + Y_{\dot{w}} \nu + Z_{\dot{w}} w + Z_{\dot{p}} p + Z_{\dot{q}} q + Z_{\dot{r}} r \\ b1 &= X_{\dot{p}} u + Y_{\dot{p}} \nu + Z_{\dot{p}} w + K_{\dot{p}} p + K_{\dot{q}} q + K_{\dot{r}} r \\ b2 &= X_{\dot{q}} u + Y_{\dot{q}} \nu + Z_{\dot{q}} w + K_{\dot{q}} p + M_{\dot{q}} q + M_{\dot{r}} r \\ b3 &= X_{\dot{r}} u + Y_{\dot{r}} \nu + Z_{\dot{r}} w + K_{\dot{r}} p + M_{\dot{r}} q + N_{\dot{r}} r \end{aligned} \tag{2.15}$$

Matriz de Atenuación Hidrodinámica

$$D(\nu) = -diagX_u, Y_v, Z_w, K_p, M_q, N_r \\ - diagX_{u|u}|u|, Y_{v|v}|v|, Z_{w|w}|w|, K_{p|p}|p|, M_{q|q}|q|, N_{r|r}|r| \quad (2.16)$$

Fuerzas y momentos gravitacionales

$$g(\eta) = \begin{bmatrix} (W-B)s\theta \\ -(W-B)c\theta s\phi \\ -(W-B)c\theta c\phi \\ -(y_G W - y_B B)c\theta c\phi + (z_G W - z_B B)c\theta s\phi \\ (z_G W - z_B B)s\theta + (x_G W - x_B B)c\theta c\phi \\ -(x_G W - x_B B)c\theta s\phi - (y_G W - y_B B)s\theta \end{bmatrix} \quad (2.17)$$

2.4. Modelado de la planta

El presente trabajo tomó como referencia al ROV/AUV Saab SeaEye Falcon (`https://www.saabseaeye.com/solutions/underwater-vehicles/falcon`), para el diseño de la planta trabajaremos en el plano XY, por lo cual se considerarán sólo cuatro propulsores y las posiciones y velocidades en los ejes X y Y. En base a esta consideración, las ecuaciones de movimiento indicadas en el capítulo preliminar se reducen a las indicadas a continuación:

(a) Vista general (b) Distribución de actuadores

Figura 2.2: AUV/ROV Falcon

La Matriz de Inercia se reduce:

$$\mathbf{M} = \begin{bmatrix} M_{\dot{u}} & 0 & 0 \\ 0 & M_{\dot{v}} & 0 \\ 0 & 0 & M_{\dot{r}} \end{bmatrix} \quad (2.18)$$

Donde $M_{\dot{u}} = m - X_{\dot{u}}$, $M_{\dot{v}} = m - _{\dot{v}}$ y $M_{\dot{r}} = I_z - N_{\dot{r}}$, son los términos de la matriz de inercia y masa agregada. La fuerza de restauración no es considerada $g(\eta) = 0$, y la matriz de amortiguación es:

$$\mathbf{D}(\mathbf{v}) = \begin{bmatrix} X_u + D_u|u| & 0 & 0 \\ 0 & Y_v + D_v|v| & 0 \\ 0 & 0 & N_r + D_r|r| \end{bmatrix} \tag{2.19}$$

Donde X_u, Y_v y N_r son los coeficientes de arrastre lineal (linear drag coefficients), y D_u, D_v y D_r son los coeficientes de arrastre cuadrático (quadratic drag coefficients). Las matrices de Coriolis y Centripetal son:

$$\mathbf{C}(\mathbf{v}) = \begin{bmatrix} 0 & 0 & -M_{\dot{v}}v \\ 0 & 0 & M_{\dot{u}}u \\ M_{\dot{v}}v & -M_{\dot{u}}u & 0 \end{bmatrix} \tag{2.20}$$

En el plano XY, el vector de velocidad se representa como $\mathbf{v}=[u, v, r]^T$ que agrupa las velocidades en eje X (surge), eje Y(sway) y angular sobre el eje Z (yaw), y el vector de posición y orientación es $\eta = [x, y, \psi]^T$.

En el diseño del control de movimiento del AUV, se asumen que las perturbaciones son pequeñas, por lo cual $\mathbf{w}\approx 0$. Por lo cual las ecuaciones dinámicas de movimiento se representan como:

$$\mathbf{M}\dot{\mathbf{v}} + \mathbf{C}(\mathbf{v})\mathbf{v} + \mathbf{D}(\mathbf{v})\mathbf{v} + \mathbf{g}(\eta) = \tau \tag{2.21}$$

Donde $\tau = [F_u, F_v, F_r]^T$ representa las fuerzas y momentos. Así mismo, expandiendo los componentes indicados en la ecuación 2.21 obtenemos:

$$\dot{u} = \frac{M_{\dot{v}}}{M_{\dot{u}}}vr - \frac{X_u}{M_{\dot{u}}}u - \frac{D_u}{M_{\dot{u}}}u|u| + \frac{F_u}{M_{\dot{u}}} \tag{2.22}$$

$$\dot{v} = \frac{M_{\dot{u}}}{M_{\dot{v}}}ur - \frac{Y_v}{M_{\dot{v}}}v - \frac{D_v}{M_{\dot{v}}}v|v| + \frac{F_v}{M_{\dot{v}}} \tag{2.23}$$

$$\dot{r} = \frac{M_{\dot{u}} - M_{\dot{v}}}{M_{\dot{r}}}uv - \frac{N_r}{M_{\dot{r}}}r - \frac{D_r}{M_{\dot{r}}}r|r| + \frac{F_r}{M_{\dot{r}}} \tag{2.24}$$

Las ecuaciones cinemáticas pueden ser simplificadas como se muestra a continuación:

$$\dot{\eta} = \begin{bmatrix} cos\psi & -sen\psi & 0 \\ sen\psi & cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ r \end{bmatrix} = \mathbf{R}(\psi)\mathbf{v} \tag{2.25}$$

Adicionalmente, expandiendo las ecuaciones cinemáticas indicadas en 2.25 obtenemos las siguientes ecuaciones:

$$\dot{x} = ucos\psi - vsin\psi \tag{2.26}$$

$$\dot{y} = usin\psi + vcos\psi \tag{2.27}$$

$$\dot{\psi} = r \tag{2.28}$$

Definiendo los estados como $\mathbf{x} = [\eta^T, v^T]^T$, teniendo como τ como las entradas de control. En base a las ecuaciones mostradas podemos llegar a la forma general del modelo del AUV:

$$\dot{\mathrm{x}} = \begin{bmatrix} \mathbf{R}(\psi)\mathbf{v} \\ \mathbf{M^{-1}}(\tau - \mathbf{C}(\mathbf{v})\mathbf{v} - \mathbf{D}(\mathbf{v})\mathbf{v} - \mathbf{g}(\eta)) \end{bmatrix} = \bar{f}(\mathrm{x}, \tau) \tag{2.29}$$

El vector de entrada τ es la fuerza de los propulsores. Para el movimiento en el plano XY del ROV/AUV Falcon se utilizarán cuatro propulsores. La relación entre el vector de entrada y el torque de los propulsores se indica a continuación:

$$\tau = \mathbf{Bu} \tag{2.30}$$

Donde $\mathbf{u} = [u_1, u_2, u_3, u_4]^T$ muestra la fuerza que proviene de cada propulsor y $\mathbf{B}$ es la matriz de entradas. Por lo tanto, el modelo de control de la planta puede ser resumido en una ecuación que agrupa la parte cinemática, dinámica y control, tal como se muestra a continuación:

$$\dot{\mathrm{x}} = \begin{bmatrix} \mathbf{R}(\psi)\mathbf{v} \\ \mathbf{M^{-1}}(\mathbf{Bu} - \mathbf{C}(\mathbf{v})\mathbf{v} - \mathbf{D}(\mathbf{v})\mathbf{v} - \mathbf{g}(\eta)) \end{bmatrix} = \mathbf{f}(\mathrm{x}, \tau) \tag{2.31}$$

Los coeficientes hidrodinámicos para el ROV/AUV Falcon se resumen en la Tabla siguiente. La matriz de entrada es:

$$\mathbf{B} = \begin{bmatrix} 0{,}7974 & 0{,}8643 & 0{,}8127 & 0{,}8270 \\ 0{,}6032 & 0{,}5029 & -0{,}5824 & -0{,}5610 \\ 0{,}2945 & -0{,}3302 & -0{,}2847 & 0{,}3505 \end{bmatrix} \tag{2.32}$$

Tabla 2.2: Coeficientes Hidrodinámicos del ROV/AUV Falcon

Términos de Inercia	Arrastre Lineal	Arrastre Cuadrático
$M_{\dot{u}} = 283.6kg$	X_u= $26.9kg/s$	$D_u = 241.3kg/m$
$M_{\dot{v}} = 593.2kg$	Y_v= $35.8kg/s$	$D_v = 503.8kg/m$
$M_{\dot{r}} = 29.0kgm^2$	N_r= $3.5kgm^2/s$	$D_r = 76.9\ kgm^2$

Luego de desarrollar las diferentes ecuaciones relacionadas a la cinemática y dinámica del movimiento del ROV/AUV Falcon, procedemos a trabajar en el modelo matemático en base a las ecuaciones cinemáticas de movimiento, las ecuaciones dinámicas de movimiento y la función de distribución de empuje. Reemplazando las ecuaciones de (2.25), (2.18), (2.20), (2.19) en (2.31) obtenemos:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\psi} \\ \dot{u} \\ \dot{v} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & cos\psi & -sin\psi & 0 \\ 0 & 0 & 0 & sin\psi & cos\psi & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -0{,}094 - 0{,}84|u| & 0 & 2{,}078v \\ 0 & 0 & 0 & 0 & -0{,}061 - 0{,}85|v| & -1{,}010u \\ 0 & 0 & 0 & -20{,}47v & 9{,}784u & -0{,}12 - 2{,}65|r| \end{bmatrix} \begin{bmatrix} x \\ y \\ \psi \\ u \\ v \\ r \end{bmatrix} +$$

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0{,}0028 & 0{,}0030 & 0{,}0029 & 0{,}0029 \\ 0{,}0010 & 0{,}0008 & -0{,}0010 & -0{,}0009 \\ 0{,}0102 & -0{,}0114 & -0{,}0098 & 0{,}0121 \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \end{bmatrix}$$

Tal como se muestra en la ecuación (2.33) y en la **Fig. 2.3**, la cantidad de estados se limita al plano X-Y por lo cual sólo se tienen seis estados. Adicionalmente, el sistema mantiene las no linealidades en su estructura, por lo cual es necesario aplicar controles robustos para poder tener resultados adecuados en la estrategia de control.

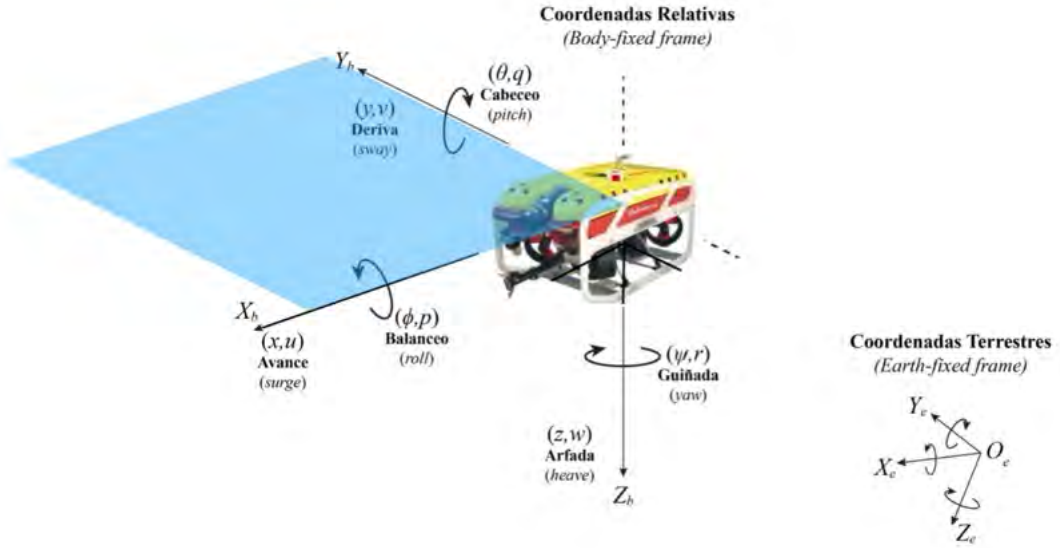


Figura 2.3: Plano de evaluación de movimiento de control

CAPÍTULO 3

SISTEMA DE CONTROL DEL VEHÍCULO SUBMARINO AUTÓNOMO

En el presente Capítulo se desarrollará el sistema de control del Vehículo Submarino no Tripulado *ROV/AUV Falcon*, tomando como base el modelo del vehículo submarino discutido en el capítulo previo. El sistema de control tiene como objetivo el control estabilización de posición en el plano X-Y.

Como primera etapa se diseñó el sistema de control no lineal MPC o NLMPC (*Non Lineal Model Predictive Control*) para la planta representada por las ecuaciones (2.33). El control NLMPC es un problema de Control Óptimo OCP (*Optimal Control Problem*) que se resolverá mediante algoritmos de optimización aplicados a problemas de programación no lineal NLP (*Nonlineal Programming Problem*), para el presente trabajo se utilizó el software CasADi [Andersson et al., 2011] (`https://web.casadi.org`), una herramienta de fuente abierta para la optimización no lineal y diferenciación algorítmica para aplicaciones de control NLMPC. A pesar que los algoritmos NLP son potentes para resolver los problemas de optimización presentes en el NLMPC, su costo computacional representa una gran desventaja en el momento de su implementación en modelos reales con controladores empaquetados en pequeños robots submarinos, por esta razón se determinó los tiempos de procesamiento de los algoritmos para evaluar su optimización en la siguiente etapa.

Como segunda etapa se implementó un sistema de control con algoritmos de redes neuronales profundas DNN (*Deep Neural Network*). Las redes DNN se caracterizan por tener una gran cantidad de capas intermedias de neuronas con el objetivo de mejorar el aprendizaje para procesos complejos. Se generó una base de datos extensa con las respuestas de las señales de control NLMPC a diferentes señales aleatorias de entrada. La base de datos generada se utilizó para realizar un entrenamiento supervisado del control con DNN, se consideró un ochenta por ciento para aprendizaje y un veinte por ciento para validación. Posterior a la etapa de entrenamiento se realizó una verificación de los resultados de control entre los tipos de control NLMPC y DNN, demostrando que los resultados son satisfactorios por lo cual se puede reemplazar el tipo de control NLMPC por

el DNN. A diferencia del control NLMPC, el control DNN no realiza una optimización en línea de un modelo, por lo cual los tiempos de ejecución son muy reducidos en comparación al NLMPC. El resumen de la estrategia seguida se muestra en la **Fig. 3.1**.

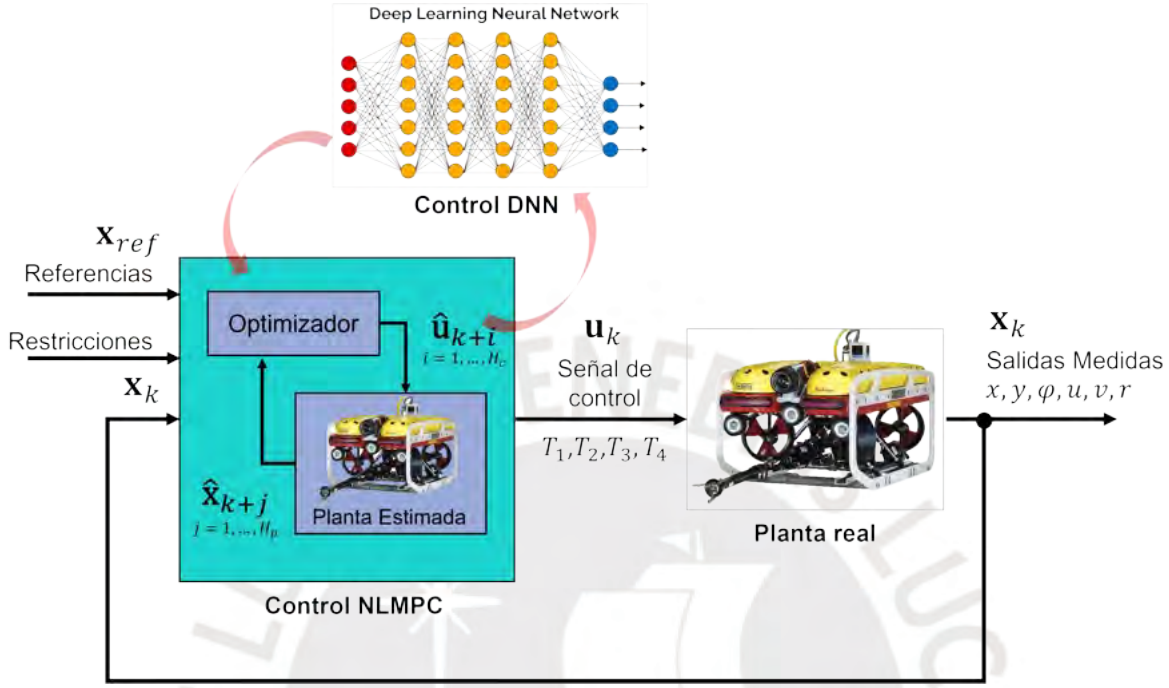


Figura 3.1: Estrategia de implementación de control de trayectoria por redes profundas

3.1. Sistema de Control No Lineal MPC

Tomando en cuenta el modelo dinámico del ROV/AUV: Falcon obtenido en el capítulo anterior, el mismo que está expresado mediante ecuaciones del sistema de control dinámico, procedemos a representar la planta como una caja negra. Dado que nuestro interés está centrado en los estados de posición en el plano X-Y, se seleccionan los estados $\mathbf{x} = [x, y, \psi, u, v, r]$ y la señal de control representada por los propulsores $\mathbf{u} = [T1, T2, T3, T4]$, tal como se muestra en la **Fig. 3.2**

En base al modelo de caja negra implementamos el sistema de control MPC en lazo cerrado mostrado en la **Fig. 3.3**. Tal como se indicó en la teoría, el control MPC está conformado por un modelo de la planta (Planta Estimada) y un optimizador que evalúa la mejor acción de control para que el error con la referencia sea el menor considerando las restricciones consideradas para la planta. Esta señal de control óptima es la que se aplica a la planta y las señales de salida son medidos para ser tomados en cuenta en el siguiente periodo. Considerando que la optimización se realiza casi en tiempo real, los costos computacionales dependen de factores como el tiempo de muestreo, el horizonte de

Figura 3.2: Modelo de la Planta como Caja Negra

predicción y las restricciones.

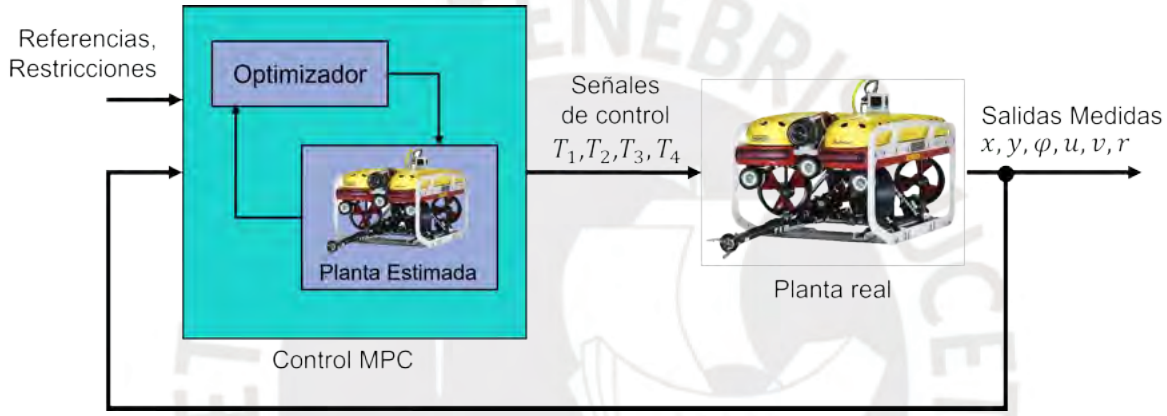


Figura 3.3: Lazo de Control MPC para el ROV/AUV Falcon

3.1.1. Modelo Discreto de la Planta

Para realizar el control necesitamos el modelo discreto de la planta. Para lograr esto utilizaremos el método de Euler, el cual consiste en encontrar la solución de una ecuación diferencial de primer orden mediante iteraciones partiendo de un valor inicial X_0 y avanzando en un periodo $\triangle T$, tal como se muestra en la ecuación (3.1).

$$Y_{k+1} = Y_k + \triangle T \cdot f(X_k, Y_k) \tag{3.1}$$

Partimos de definir el sistema de ecuaciones tomando en cuenta lo indicado en la ecuación (2.33). La representación de las ecuaciones del sistema se muestra a continuación:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\psi} \\ \dot{u} \\ \dot{v} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} ucos\psi - vsin\psi \\ usin\psi + vcos\psi \\ r \\ -0{,}094u - 0{,}84|u|u + 2{,}078vr + 0{,}0028T_1 + 0{,}0030T_2 + 0{,}0029T3 + 0{,}0029T_4 \\ -0{,}061v - 0{,}85|v|v - 1{,}010ur + 0{,}0010T_1 + 0{,}0008T_2 - 0{,}0010T3 + 0{,}0009T_4 \\ -10{,}686uv - 0{,}12r - 2{,}65|r|r + 0{,}0102T_1 - 0{,}0114T_2 - 0{,}0098T_3 + 0{,}0121T_4 \end{bmatrix} \tag{3.2}$$

En base a lo indicado en la ecuación (3.1), procedemos a aplicar el método de Euler en la ecuación (3.2). Los resultados de la discretización para cada estado se muestran a continuación:

$$x(k+1) = x(k) + \Delta T[u(k)cos\psi(k) - v(k)sin\psi(k)] \tag{3.3}$$

$$y(k+1) = y(k) + \Delta T[u(k)sin\psi(k) + v(k)cos\psi(k)] \tag{3.4}$$

$$\psi(k+1) = \psi(k) + \Delta T[r(k)] \tag{3.5}$$

$$\begin{aligned} u(k+1) = u(k) + \Delta T[&- 0{,}094u(k) - 0{,}84|u(k)|u(k) + 2{,}078v(k)r(k) + ... \\ &+ 0{,}0028T_1(k) + 0{,}0030T_2(k) + 0{,}0029T_3(k) + 0{,}0029T_4(k)] \end{aligned} \tag{3.6}$$

$$\begin{aligned} v(k+1) = v(k) + \Delta T[&- 0{,}061v(k) - 0{,}85|v(k)|v(k) - 1{,}010u(k)r(k) + ... \\ &+ 0{,}0010T_1(k) + 0{,}0008T_2(k) - 0{,}0010T_3(k) + 0{,}0009T_4(k)] \end{aligned} \tag{3.7}$$

$$\begin{aligned} r(k+1) = r(k) + \Delta T[&- 10{,}686u(k)v(k) - 0{,}12r(k) - 2{,}65|r(k)|r(k) + ... \\ &+ 0{,}0102T_1(k) - 0{,}0114T_2(k) - 0{,}0098T_3(k) + 0{,}0121T_4(k)] \end{aligned} \tag{3.8}$$

Las ecuaciones discretas de la planta representadas previamente pueden expresarse en la ecuación general (3.9). La cual representa los estados de la planta en el instante $k+1$ en base a los estados de la planta y las señales de control en el estado k.

$$\mathbf{x}(k+1) = \mathbf{f}(\mathbf{x}(k), \mathbf{u}(k)) \tag{3.9}$$

Donde:

$$\mathbf{x}(k+1) = \begin{bmatrix} x(k+1) \\ y(k+1) \\ \psi(k+1) \\ u(k+1) \\ v(k+1) \\ r(k+1) \end{bmatrix}, \mathbf{x}(k) = \begin{bmatrix} x(k) \\ y(k) \\ \psi(k) \\ u(k) \\ v(k) \\ r(k) \end{bmatrix}, \mathbf{u}(k) = \begin{bmatrix} T_1(k) \\ T_2(k) \\ T_3(k) \\ T_4(k) \end{bmatrix}$$

3.1.2. Restricciones del Sistema

El control de tipo NLMPC permite incorporar restricciones físicas que pueden existir en el ambiente de control. Estas restricciones pueden estar asociadas tanto a las señales de control como a los estados de la planta. Para nuestro caso se han determinado dos tipos de restricciones:

- **Señales de Control:** Para el modelo evaluado se tomó como máximo valor de empuje y arrastre el valor de $1kgf$.

$$T_{1_{max}} = T_{2_{max}} = T_{3_{max}} = T_{4_{max}} = 1kgf \tag{3.10}$$

$$T_{1_{min}} = T_{2_{min}} = T_{3_{min}} = T_{4_{min}} = -1kgf \tag{3.11}$$

- **Estados de la Planta:** De todos los estados de la planta vamos a crear restricciones del espacio de desplazamiento en los ejes X y Y. En este caso hemos determinado un espacio de operación definido por $[x_{max}, x_{min}] = [-5, 5]$ y $[y_{max}, y_{min}] = [-5, 5]$, tal como se puede mostrar en la **Fig.3.4**, respecto al estado definido por el ángulo de guiñada ψ, no se define restricción pues puede ser infinito dado que cualquier ángulo tiene un equivalente entre 0 y 2π.

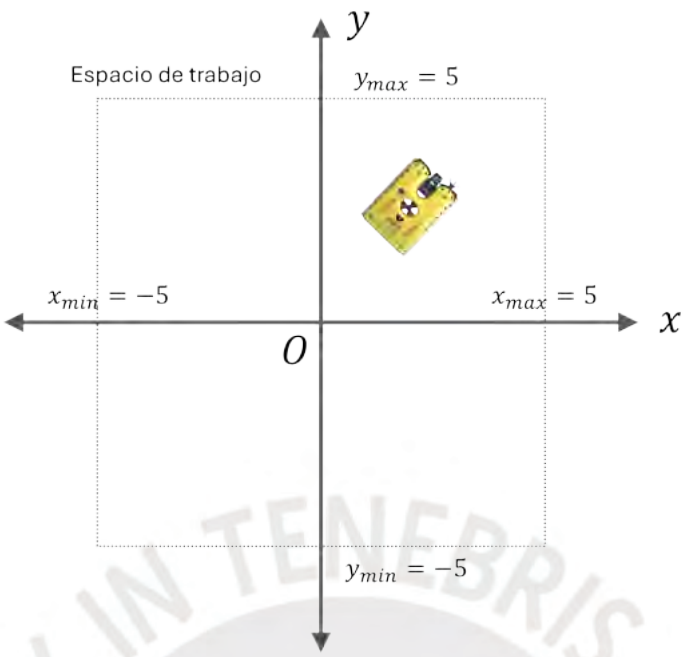


Figura 3.4: Restricciones de estados x y y

3.1.3. Control de Estabilización de Posición

El control de Estabilización de Posición o conocido en inglés como *Point Stabilization Control*, tiene como objetivo desplazar el robot desde un estado inicial $[x_0, y_0, \psi_0, u_0, v_0, r_0]$ hasta un estado final $[x_f, y_f, \psi_f, u_f, v_f, r_f]$. Esta definición del problema se muestra en la **Fig.3.5**.

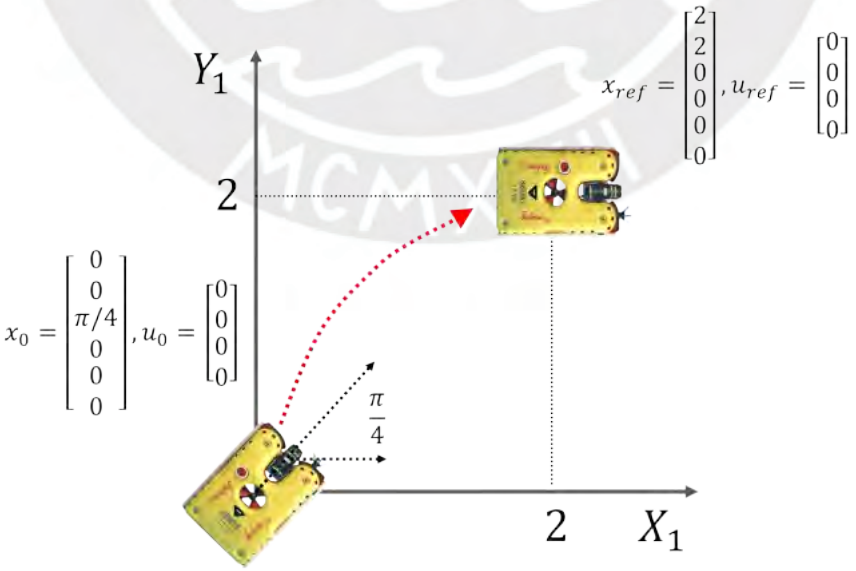


Figura 3.5: Objetivo de Control: Estabilización de Posición

Procederemos a plantear la solución matemática del problema. Tomando en cuenta lo indicado previamente, y considerando un horizonte de predicción de $N = 3$, generamos la formulación matemática para la solución del problema de optimización.

Debido a que la solución del problema de optimización se realiza de manera iterativa iniciando de $t = 0$ y avanzando en cada paso de Δt, se procederá a ejemplificar el caso en el instante inicial:

Para t = 0:

Para el momento inicial se tienen se tienen los valores de los estados y señales de control indicadas en la **Fig. 3.6**. Como se puede apreciar se planteó un horizonte de predicción pequeño (N=3) a manera de ejemplo.

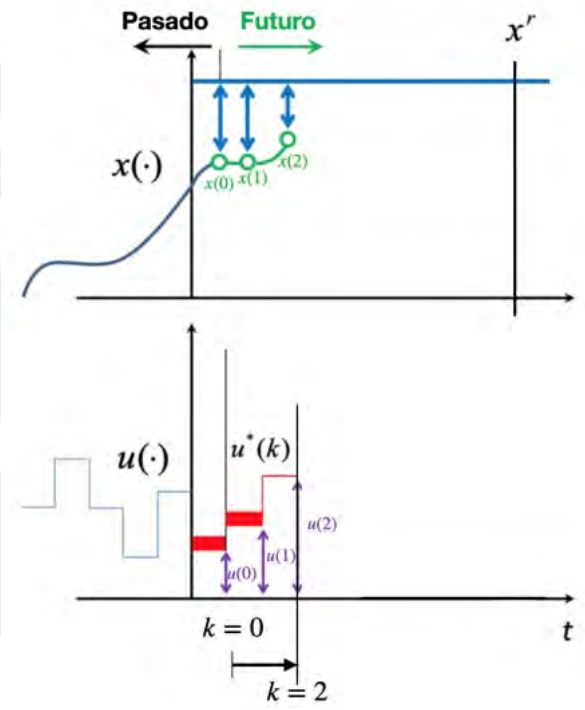


Figura 3.6: Estimación inicial para t=0

Los valores iniciales de los estados y señales de control iniciales son:

$$x_{k=0} = \begin{bmatrix} 0 \\ 0 \\ \pi/4 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad u_{k=0} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{3.12}$$

Procedemos a resolver matemáticamente el proceso de optimización no lineal para $k+0$ siguiendo los pasos detallados a continuación:

Paso 1. **Costos de Operación:** en este paso se define la función que caracteriza el objetivo de control, la ecuación encontrada se indica en la ecuación (3.13)

$$\boldsymbol{\ell}(\mathbf{x},\mathbf{u}) = \left\| \mathbf{x}_u - \begin{bmatrix} 2 \\ 2 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \right\|_Q^2 + \left\| \mathbf{u} - \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \right\|_R^2 \tag{3.13}$$

Paso 2. **Función de Costos:** función que permite la evaluación de la función de costos de operación a lo largo de todo el horizonte de predicción, se muestra en la ecuación (3.14)

$$J_N(\mathbf{x},\mathbf{u}) = \sum_{k=0}^{N-1} \boldsymbol{\ell}(\mathbf{x}_\mathbf{u}(k), \mathbf{u}(k))) = \sum_{k=0}^{N-1} \left\| \mathbf{x}_u - \begin{bmatrix} 2 \\ 2 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \right\|_Q^2 + \left\| \mathbf{u} - \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \right\|_R^2 \tag{3.14}$$

Paso 3. **Problema de Control Óptimo:** con la función encontrada en la ecuación (3.15) y tomando en consideración un horizonte de predicción de $N = 3$, procedemos a resolver el problema de optimización planteado en la ecuación (3.15)

$$\text{minimize}_u J_N(\mathbf{x},\mathbf{u}) = \left\| \mathbf{x}_u(0) - \begin{bmatrix} 2 \\ 2 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \right\|_Q^2 + \left\| \mathbf{u}(0) - \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \right\|_R^2$$
$$+ \left\| \mathbf{x}_u(1) - \begin{bmatrix} 2 \\ 2 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \right\|_Q^2 + \left\| \mathbf{u}(1) - \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \right\|_R^2$$
$$+ \left\| \mathbf{x}_u(2) - \begin{bmatrix} 2 \\ 2 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \right\|_Q^2 + \left\| \mathbf{u}(2) - \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \right\|_R^2$$

Considerando:

$$\mathbf{x}(k+1) = \mathbf{f}(\mathbf{x}(k), \mathbf{u}(k)) \tag{3.15}$$

$$\mathbf{x}_u(0) = \begin{bmatrix} 2 \\ 2 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \mathbf{u}_0 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$
$$\mathbf{u}(k) \in U, \forall k \in [0, 2]$$
$$\mathbf{x_u}(k) \in X, \forall k \in [0, 2] \tag{3.16}$$

Paso 4. **Solución del problema de Control Óptimo:** la ecuación (3.15) se resuelve con el método de Punto Interior (*Interior Point*). En nuestro caso resolvemos el problema de optimización no lineal utilizando el paquete CaSAdi. La solución a esta optimización nos da los siguientes valores de control:

$$T1_0 = \begin{bmatrix} 0{,}0044 \\ 0{,}0044 \\ 0{,}0044 \end{bmatrix}, T2_0 = \begin{bmatrix} 0{,}0059 \\ 0{,}0059 \\ 0{,}0059 \end{bmatrix}, T3_0 = \begin{bmatrix} 0{,}0056 \\ 0{,}0056 \\ 0{,}0056 \end{bmatrix}, T4_0 = \begin{bmatrix} 0{,}0045 \\ 0{,}0045 \\ 0{,}0045 \end{bmatrix} \quad (3.17)$$

Estas señales de control aplicadas a los estados generaron los valores:

$$x_1 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, x_2 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, x_3 = \begin{bmatrix} 0{,}7854 \\ 0{,}7854 \\ 0{,}7854 \end{bmatrix} x_1 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, x_2 = \begin{bmatrix} 0 \\ 0 \\ 00 \end{bmatrix}, x_3 = \begin{bmatrix} 0 \\ 0 \\ 00 \end{bmatrix} \quad (3.18)$$

Finalmente, tomamos los primeros valores y actualizamos los valores de los estados y señales de control, estos estados serán los iniciales para la siguiente iteración:

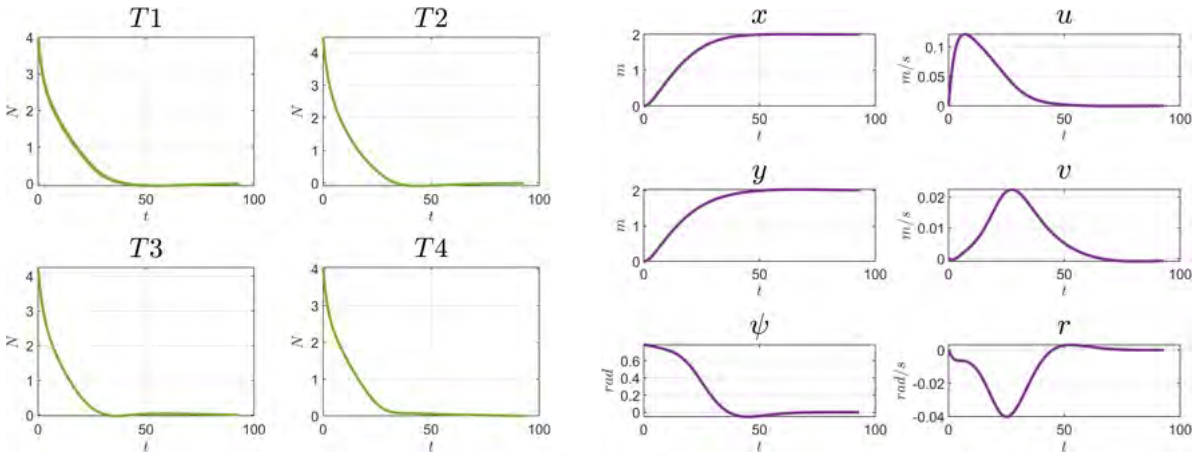
$$u_0 = \begin{bmatrix} 0{,}0044 \\ 0{,}0059 \\ 0{,}0056 \\ 0{,}0045 \end{bmatrix}, \; x_0 = \begin{bmatrix} 0 \\ 0 \\ 0{,}7854 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad (3.19)$$

Tal como se puede apreciar, las señales de control aplicadas a la planta no representaron una modificación considerable de los estados de control, esto se debe a que el horizonte de predicción considerado: $N = 3$ es muy pequeño, sin embargo, es adecuado para fines de explicación de los pasos a seguir.

Para t desde $t_0 + \Delta T$ hasta t_{sim}

Para la siguiente iteración tomamos los valores finales indicados en la ecuación (3.19). Este proceso se repite hasta completar el tiempo de simulación o el error de estado estacionario cumple con el margen seleccionado. Los resultados finales de esta simulación se muestran en la **Fig. 3.8** y **Fig. 3.7**.

El análisis realizado se llevó a cabo con un horizonte de predicción $N = 3$, sin embargo, es posible reducir el número de iteraciones (tiempo de solución) incrementando el horizonte de predicción. Tomando en cuenta que el horizonte de predicción es de $N = 3$, los vectores de control y los estados estimados tienen una extensión de 3 periodos, y la solución de la optimización se realiza con estos valores. Sin embargo, con el objetivo de mejorar el control se puede extender el horizonte de predicción de $N = 3$ a $N = 100$, lo cual mejora



(a) Señales de Control (b) Estados del Sistema

Figura 3.7: Resultados de simulación con Horizonte $N = 3$

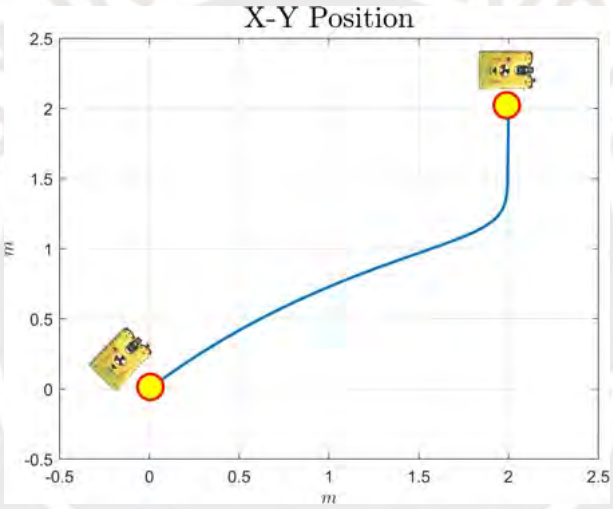
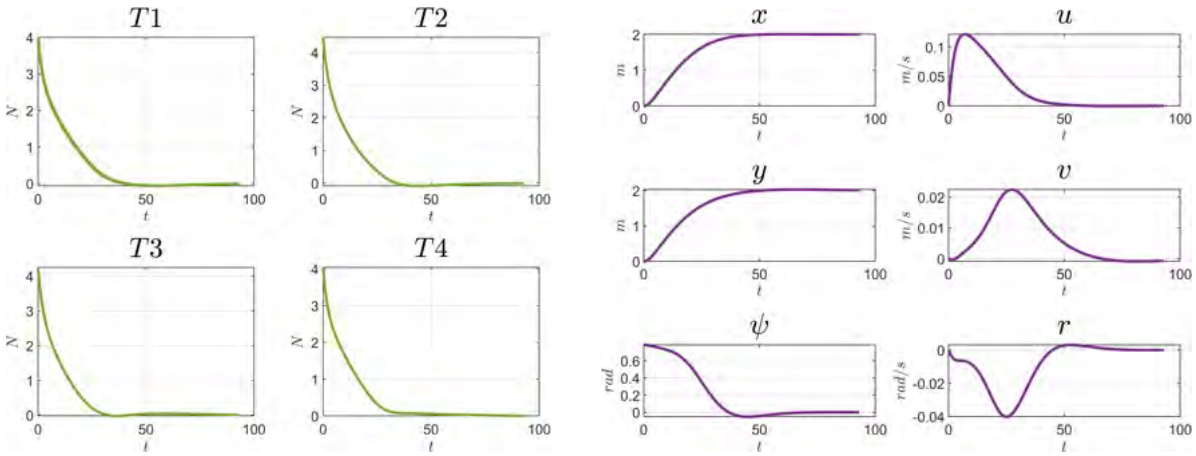


Figura 3.8: Estados del Sistema para $N = 3$

el control pero genera un costo computacional elevado, debido a que se trabajará con matrices de estados y de control que tienen una extensión de 100 periodos.

Los resultados para un horizonte de control de N=100 se muestran en la **Fig. 3.10** y **Fig. 3.9**. Tal como se puede apreciar las señales de control y los estados tiene una mayor variación debido a la estimación con un horizonte mucho mayor, esto permite menos iteraciones y tiempo de cómputo.



(a) Señales de Control (b) Estados del Sistema

Figura 3.9: Resultados de simulación con Horizonte de Predicción $N = 100$

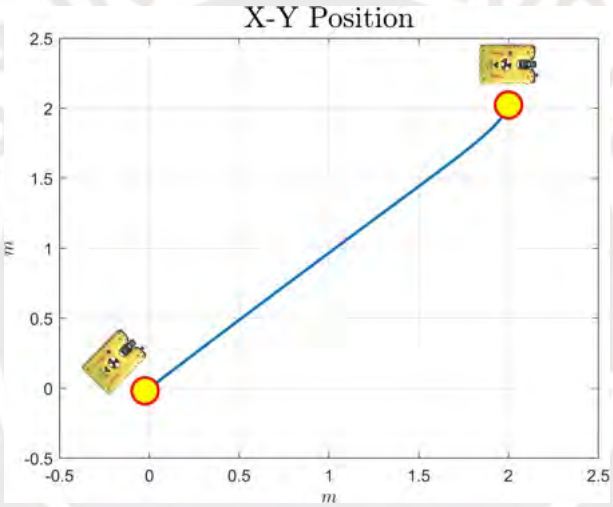


Figura 3.10: Estados del Sistema para $N = 50$

La variación del Horizonte de Predicción N genera una solución en menor tiempo, sin embargo, a un costo computacional mas alto. Para el caso de Estabilización de Posición (*Point Stabilization*) hemos realizado un muestreo de diferentes escenarios modificando en Horizonte de Predicción N, el resumen de las iteraciones, tiempos de procesamiento y error se muestran en la Tabla (3.1).

Tabla 3.1: Resultados de simulaciones para ejemplo

N	Iteraciones	Tiempo procesamiento (seg)	Error
3	365,450	334,800.0000	0.0602
20	3308	52.3820	0.0100
50	401	5.9667	0.0097
100	372	8.5846	0.0099
150	337	10.6014	0.0100
200	331	13.3020	0.0099
300	330	18.7943	0.0099
500	330	31.1743	0.0099

3.1.4. Control de Seguimiento de Trayectoria

El control de Seguimiento de Trayectoria o conocido en inglés como *Trajectory Tracking Control* tiene como objetivo el seguimiento de una trayectoria predefinida. En la **Fig. 3.11** se puede observar que se definió una línea recta para el seguimiento por parte de la planta.

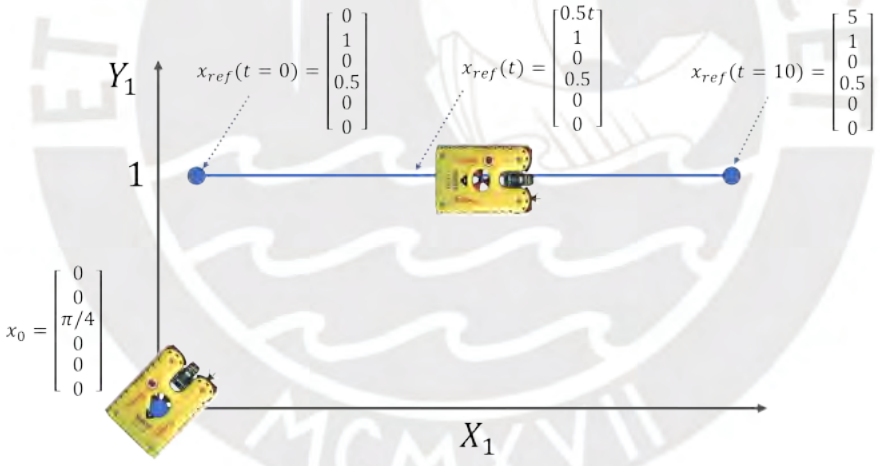


Figura 3.11: Objetivo de Control: Seguimiento de Trayectoria

En base a la línea recta seleccionada en la **Fig. 3.11** se plantean los estados de referencia en la ecuación (3.20).

$$\mathbf{X_{ref}} = \begin{bmatrix} 0{,}5t \\ 0 \\ 0 \\ 0{,}5 \\ 0 \\ 0 \end{bmatrix} \tag{3.20}$$

Con los resultados de la ecuación (3.20) de los estados de referencia, procedemos a calcular la señal de control de referencia u_{ref}. Debido a que se desea que el vehículo pare completamente al finalizar el seguimiento de la trayectoria, se plantean estados de control de referencia iguales a cero, tal como se muestra a continuación:

$$\mathbf{u_{ref}} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{3.21}$$

Tal como se detalló en el control de Estabilización de Posición, llevamos todas las ecuaciones a un problema de programación lineal, planteamos las ecuaciones del problema de control óptimo:

$$\min_u J_N(\mathbf{x}_0, \mathbf{u}) = \sum_{k=0}^{N-1} \boldsymbol{\ell}(\mathbf{x_u}(k), \mathbf{u}(k))$$

Considerando:

$$\begin{aligned} &\mathbf{x}_u(k+1) = \mathbf{f}(\mathbf{x}_u(k), \mathbf{u}(k))) \\ &\mathbf{x}_u(0) = \mathbf{x}_0, \\ &\mathbf{u}(k) \in U, \forall k \in [0, N-1] \\ &\mathbf{x_u}(k) \in X, \forall k \in [0, N] \end{aligned} \tag{3.22}$$

Tomando en cuenta que la función de costos de valor se representa por:

$$\boldsymbol{\ell}(\mathbf{x},\mathbf{u}) = \|\mathbf{x}_u - \mathbf{x}_{ref}\|_Q^2 + \|\mathbf{u} - \mathbf{u}_{ref}\|_R^2 \tag{3.23}$$

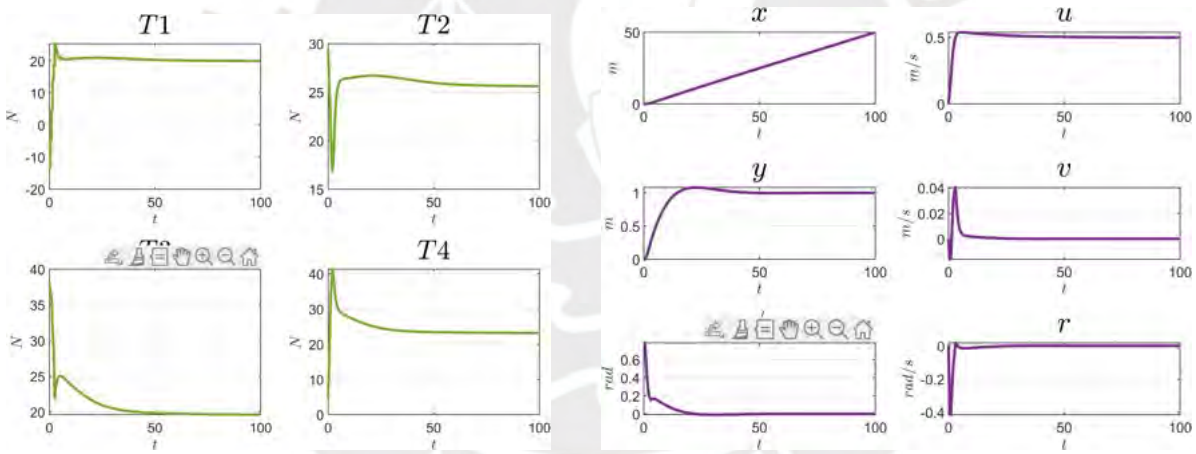
Y la función que relaciona los estados y señales de control se representa por las ecuaciones de los estados discretizados por el método de Euler y representados en las ecuaciones (3.3), (3.4), (3.5), (3.6), (3.7) y (3.8), y que se relacionan por la siguiente ecuación:

$$\mathbf{x}(k+1) = \mathbf{f}(\mathbf{x}(k), \mathbf{u}(k)) \tag{3.24}$$

Donde:

$$\mathbf{x}(k+1) = \begin{bmatrix} x(k+1) \\ y(k+1) \\ \psi(k+1) \\ u(k+1) \\ v(k+1) \\ r(k+1) \end{bmatrix}, \mathbf{x}(k) = \begin{bmatrix} x(k) \\ y(k) \\ \psi(k) \\ u(k) \\ v(k) \\ r(k) \end{bmatrix}, \mathbf{u}(k) = \begin{bmatrix} T_1(k) \\ T_2(k) \\ T_3(k) \\ T_4(k) \end{bmatrix}$$

Analizando el control de Estabilización de Posición (*Point Stabilization*) discutido en el capítulo anterior, vamos a tomar un horizonte de control de N=100 para el análisis. Los resultados para el seguimiento de la trayectoria de línea recta se muestran en la **Fig. 3.12** y **Fig. 3.13**.



(a) Señales de Control (b) Estados del Sistema

Figura 3.12: Resultados de simulación con Horizonte de Predicción $N = 100$

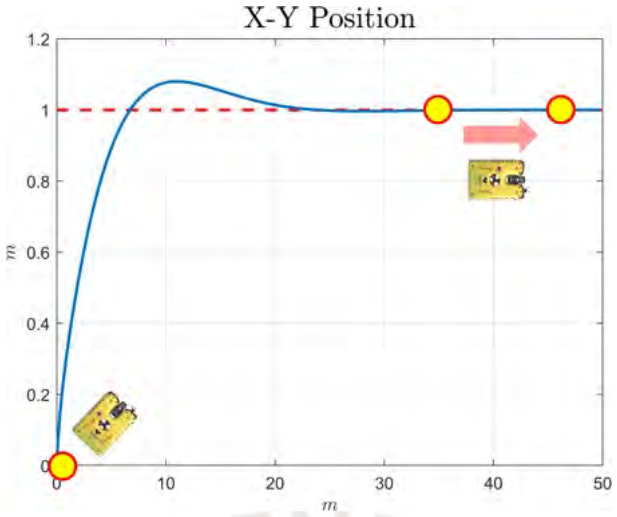


Figura 3.13: Estados del Sistema para $N = 50$

De igual forma podemos determinar el seguimiento de una línea que cuente con una pendiente. Para el caso de estudio utilizaremos la recta con función: $x = y$, tal como se muestra en la **Fig. 3.14**.

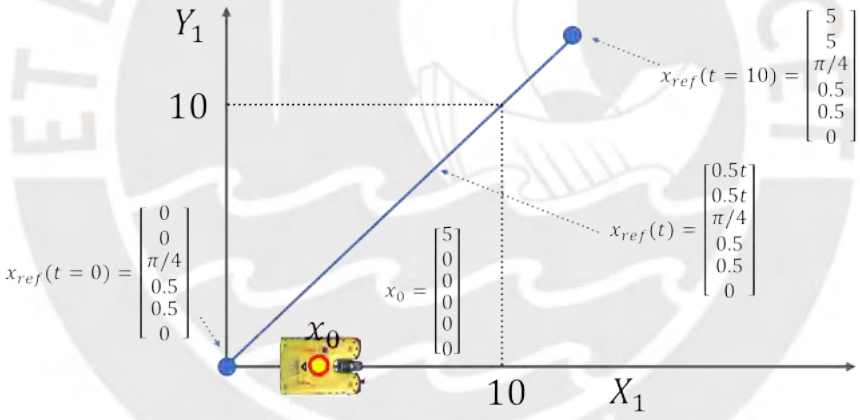


Figura 3.14: Objetivo de Control: Seguimiento de Trayectoria

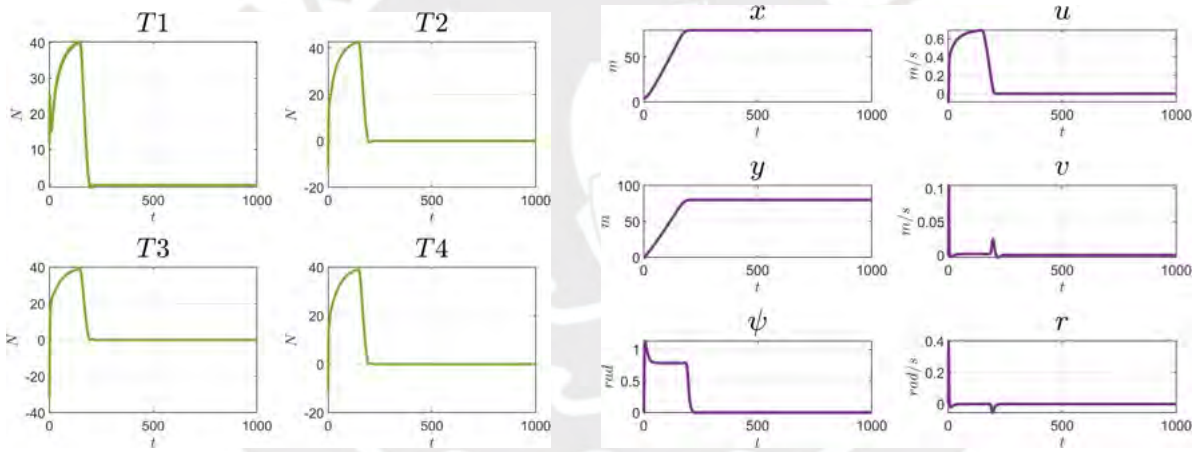
En base a la línea recta con pendiente seleccionada en la **Fig. 3.14** se plantean los estados de referencia en la ecuación (3.25).

$$\mathbf{X_{ref}} = \begin{bmatrix} 0{,}5t \\ 0{,}5t \\ \pi/4 \\ 0{,}5 \\ 0{,}5 \\ 0 \end{bmatrix} \tag{3.25}$$

Con los resultados de la ecuación (3.20) de los estados de referencia, procedemos a calcular la señal de control de referencia u_{ref}. Debido a que se desea que el vehículo pare completamente al finalizar el seguimiento de la trayectoria, se plantean estados de control de referencia iguales a cero, tal como se muestra a continuación:

$$\mathbf{u_{ref}} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{3.26}$$

Analizando el control de Estabilización de Posición (*Point Stabilization*) discutido en el capítulo anterior, vamos a tomar un horizonte de control de N=100 para el análisis. Los resultados para el seguimiento de la trayectoria de línea recta se muestran en la **Fig. 3.15** y **Fig. 3.16**.



(a) Señales de Control (b) Estados del Sistema

Figura 3.15: Resultados de simulación con Horizonte de Predicción $N = 100$

Tal como se pudo revisar, el controlador No Lineal MPC o NLMPC cumple con los requerimientos para el control de estabilización de posición (*Point Stabilization*) y Control de Seguimiento de Trayectoria (*Trajectory Tracking*).

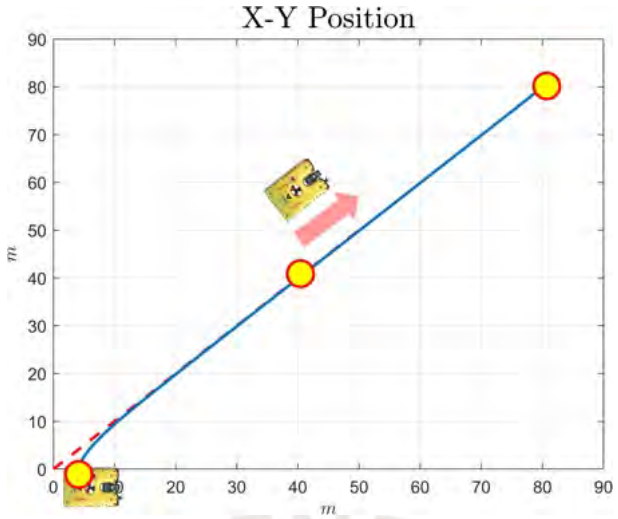


Figura 3.16: Estados del Sistema para $N = 50$

3.1.5. Análisis de Respuesta a Perturbaciones

Luego de definir el controlador no lineal MPC, procedemos a analizar la respuesta del mismo a perturbaciones que puedan generarse sobre la planta real, tal como se muestra en la **Fig. 3.17**.

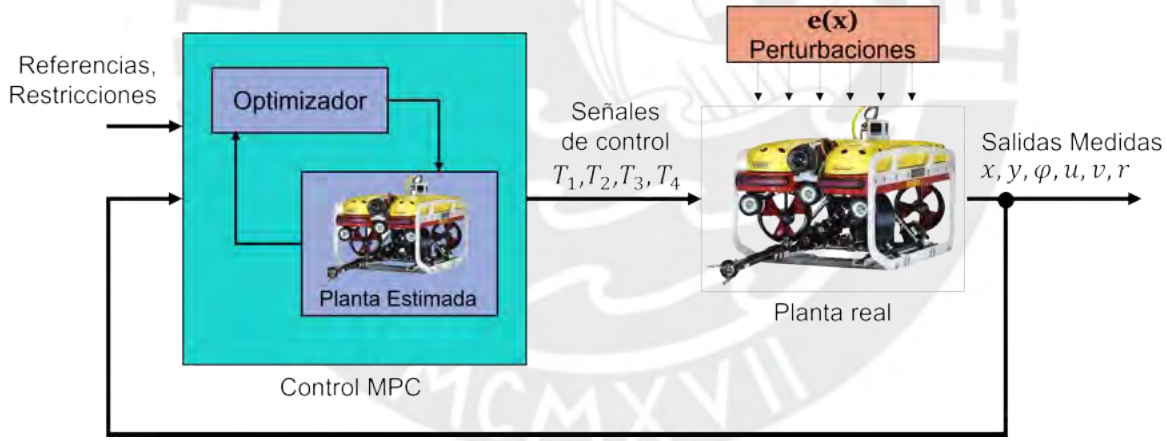


Figura 3.17: Diagrama de Lazo de Control incluyendo perturbaciones

Para el análisis procedemos a incluir un vector de perturbaciones $e(x)$ que tiene componentes de fuerza en los ejes X y Y, tal como se indica a continuación:

$$\mathbf{e}(\mathbf{x}) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ -0{,}05 \\ -0{,}005 \\ 0 \end{bmatrix} \tag{3.27}$$

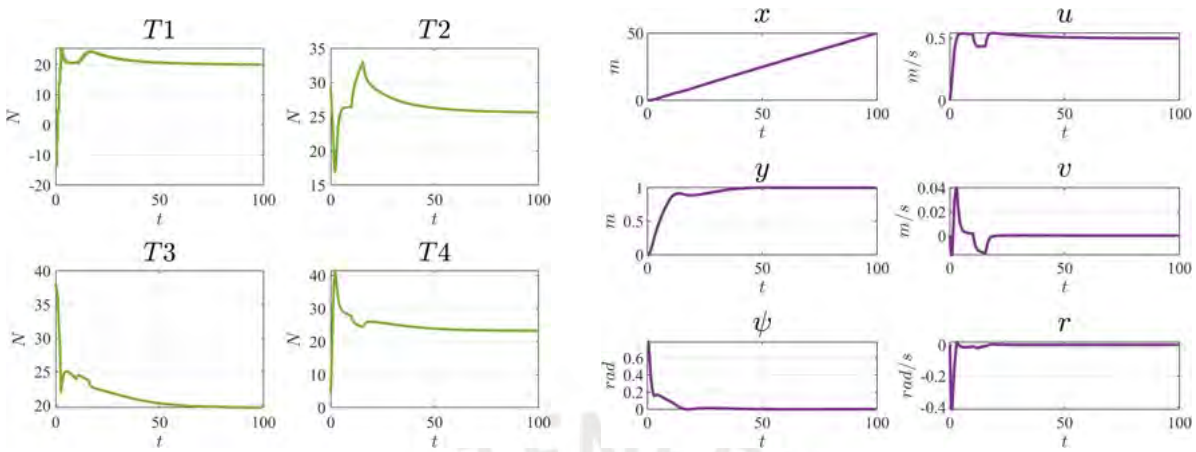
Tomando en cuenta el modelo de la planta indicado en la ecuación (2.33) y el vector de perturbación indicado en la ecuación (3.27), procedemos a representar en la ecuación (3.28) el modelo general de la planta con las perturbaciones.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\psi} \\ \dot{u} \\ \dot{v} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & cos\psi & -sin\psi & 0 \\ 0 & 0 & 0 & sin\psi & cos\psi & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -0{,}094 - 0{,}84|u| & 0 & 2{,}078v \\ 0 & 0 & 0 & 0 & -0{,}061 - 0{,}85|v| & -1{,}010u \\ 0 & 0 & 0 & -20{,}47v & 9{,}784u & -0{,}12 - 2{,}65|r| \end{bmatrix} \begin{bmatrix} x \\ y \\ \psi \\ u \\ v \\ r \end{bmatrix} +$$

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0{,}0028 & 0{,}0030 & 0{,}0029 & 0{,}0029 \\ 0{,}0010 & 0{,}0008 & -0{,}0010 & -0{,}0009 \\ 0{,}0102 & -0{,}0114 & -0{,}0098 & 0{,}0121 \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ -0{,}05 \\ -0{,}005 \\ 0 \end{bmatrix}$$

Para verificar la eficiencia del controlador con perturbaciones se analizó el seguimiento de la trayectoria para una línea en $X = 1$, tal como se revisó previamente. Para el análisis se aplicó la perturbación entre el segundo 10 y 15 de la simulación. Los resultados obtenidos se muestran en las **Fig. 3.18** y **Fig. 3.19**.

Los resultados de las simulaciones muestran que el controlador no lineal MPC realiza el seguimiento de la trayectoria a pesar de la perturbación incluida en los estados u y v. Esto ratifica el buen funcionamiento del controlador incluso bajo escenarios de perturbaciones que podrían estar asociadas a fuerzas externas como corrientes marinas, las mismas que afectan los estados analizados.



(a) Señales de Control (b) Estados del Sistema

Figura 3.18: Resultados de simulación con perturbaciones entre $t = 10 : 15$

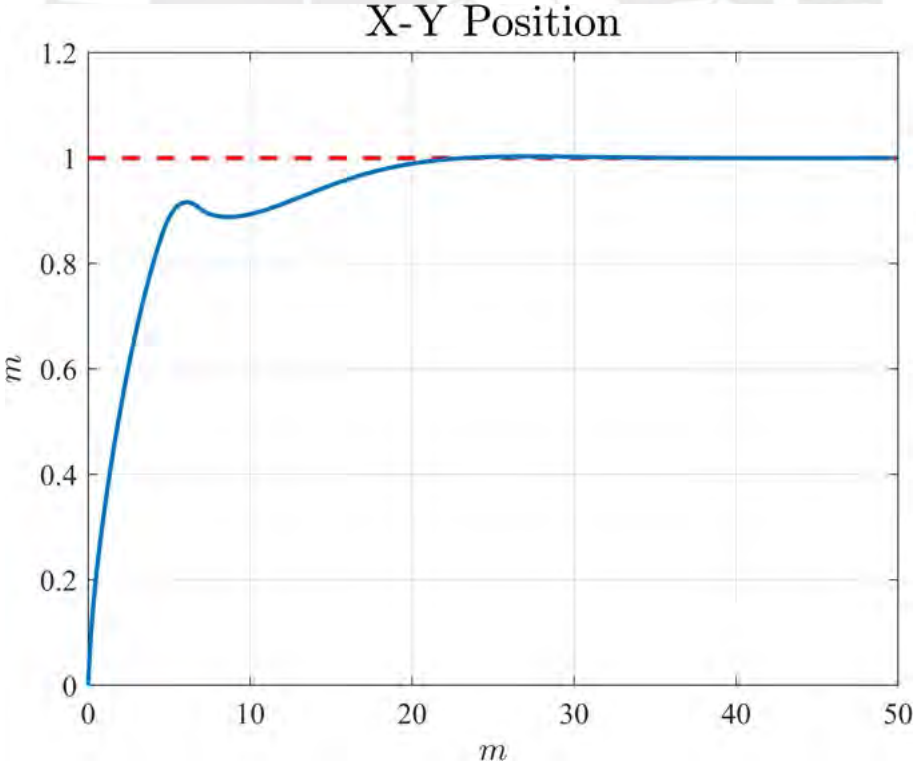


Figura 3.19: Posición del AUV Falcon en el plano $X - Y$

3.2. Sistema de Control con Redes Neuronales Profundas

En el presente capítulo se definirá el controlador neuronal basado en una red neuronal profunda DNN (*Deep Neural Network*). El objetivo principal es realizar un entrenamiento supervisado de la DNN con la base de datos generada por el control NLMPC, para posteriormente reemplazar a la red neuronal por el control NLMPC, tal como se muestra en la **Fig. 3.20**.

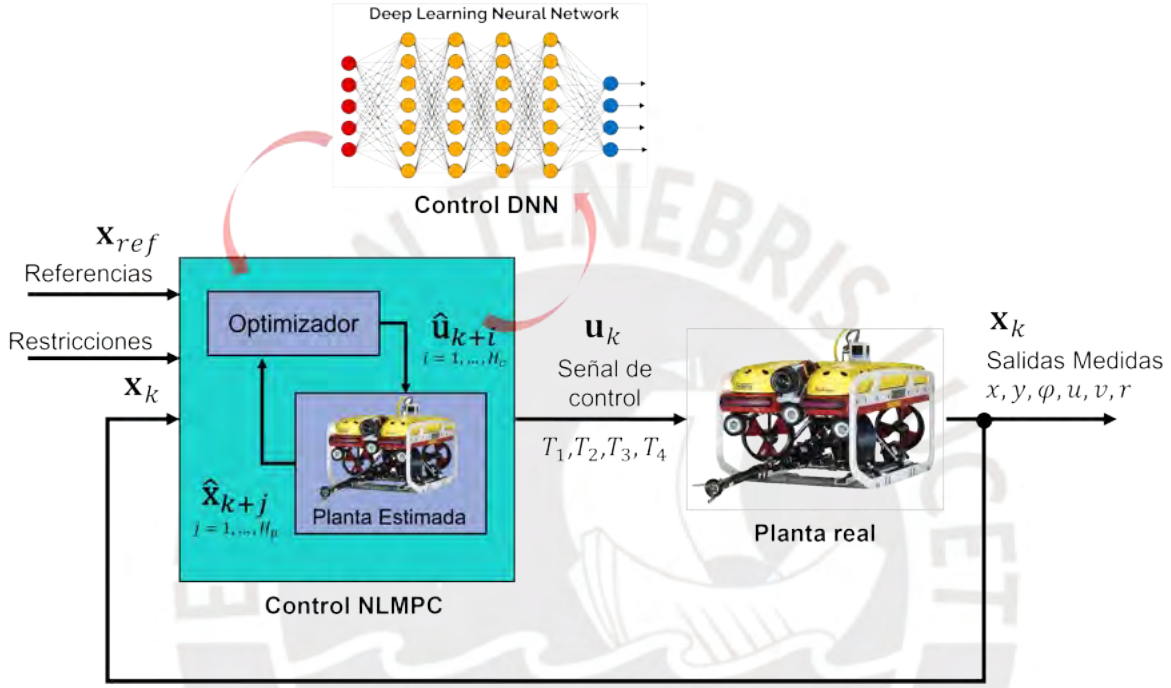


Figura 3.20: Estrategia de Control

La definición del controlador se realizará siguiendo las siguientes etapas:

- **Generación de la base de datos**: se genera en base a los resultados obtenidos por el controlador no lineal MPC (NLMPC), se estiman valores de estados iniciales y finales aleatorios y se almacena en la base de datos las acciones de control u_k, los estados x_k y estados de referencia x_ref.
- **Creación de la red neuronal profunda**: seleccionamos una red neuronal profunda con gran cantidad de neuronas en las capas ocultas, así mismo las funciones de activación entre ellas. Así mismo, se seleccionan los parámetros principales para el entrenamiento de la red neuronal.
- **Entrenamiento de la Red Neuronal Profunda**: Se realiza el entrenamiento de la DNN con la base de datos generada inicialmente.

3.2.1. Generación de la Base de Datos

El objetivo del presente capítulo es generar una base de datos con una cantidad extensa de información para entrenar la red neuronal profunda. La base de datos se elaboró mediante el Script de `Matlab` denominado `NLMPCDatabase.m`. Este script genera estados iniciales y estados de referencia aleatorios, posteriormente realiza la simulación de la respuesta del sistema de control NLMPC para cada uno de los casos determinados, para finalmente almacenar todas las respuestas dentro de la base de datos. Para elaborar el script se tomó en cuenta el trabajo de Mehrez [2019] y Mathworks [2023].

Para generar la base de datos tenemos que identificar las entradas y salidas que correspondían al controlador NLMPC y adecuarlas a las requeridas por el controlador de DNN. En la **Fig. 3.21** podemos observar que el controlador NLMPC tiene como entradas a los estados, la referencia y las restricciones, y como señal de salida a la acciones de control u_k.

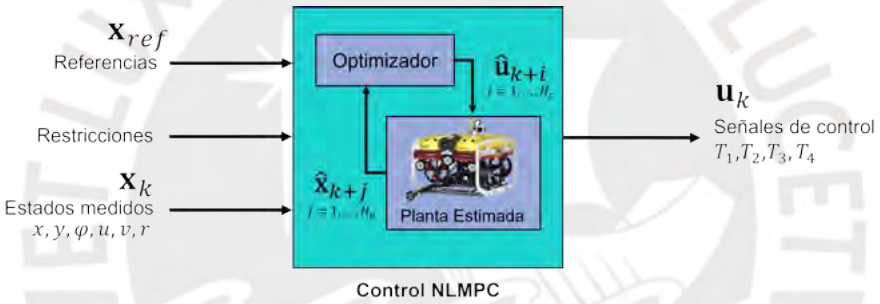


Figura 3.21: Estructura de entradas y salidas de control NLMPC

La configuración de entradas y salidas del control NLMPC, mostrado en la **Fig. 3.21**, debe ser traducida a una configuración de red neuronal profunda DNN. Para esto vamos a considerar como señales de entrada los estados de la planta y los estados referencia, sin considerar las restricciones ya que el control NLMPC ya genera una base de datos con las restricciones incluidas en su respuesta. Tomando en cuenta estas modificaciones, podemos plantear las señales de entrada y salida de la base de datos tal como se muestra en la **Fig. 3.22**.

Los resultados preliminares del control NLMPC para una trayectoria desde un estado aleatorio inicial a un estado de referencia mostraron resultados de alrededor de diez (10) segundos por simulación. Por lo cual, el generar una base de datos demandará un tiempo extenso dependiendo de la cantidad de variables que se desea almacenar. La cantidad de datos que obtenemos mejorará el aprendizaje de la red, por lo cual esta etapa es crucial dentro del proceso.

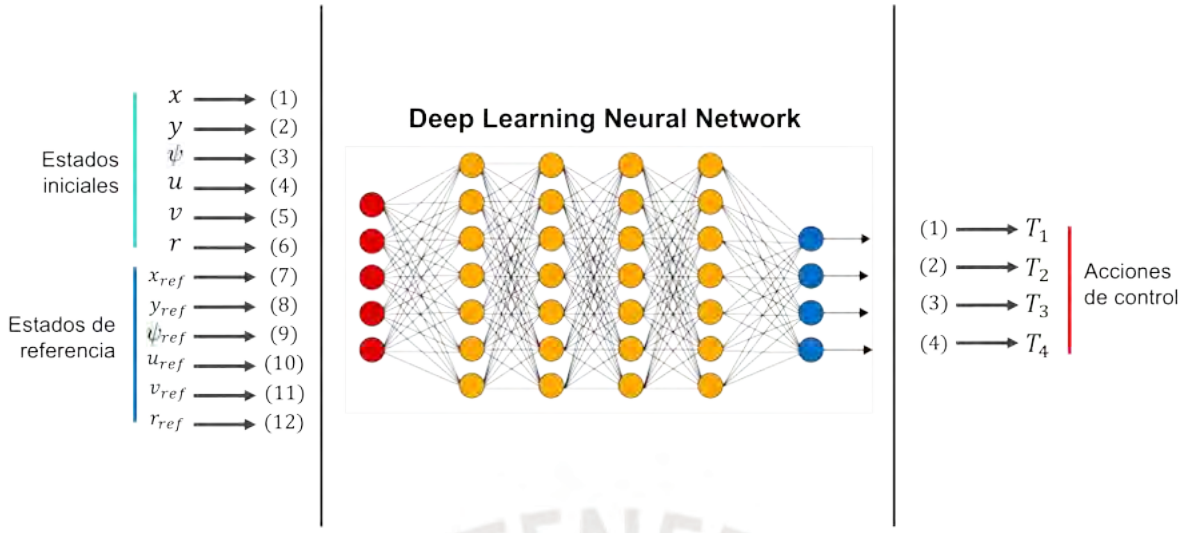


Figura 3.22: Estructura de entradas y salidas de control DNN

La **Fig. 3.23** muestra la información generada para una corrida (*run*). Esta información se almacenará en una base de datos. Así mismo, la **Fig. 3.24** muestra el desplazamiento del AUV desde un punto inicial a uno final, ambos aleatorios.

Con la estructura determinada según la **Fig. 3.22** podemos utilizar el scritp `NLMPCdatabase.m` para colectar toda los valores de la respuesta del control NLMPC y almacenarlos en una base de datos. Para este ejercicio se generó una base de datos de 1,000 datos para cada variable, por lo cual tenemos en la entrada un total de 800,000 datos y en la salida un total de 320,000 datos para el entrenamiento. La totalidad de datos se puede observar en la **Fig. 3.25** y la base de datos por estados se muestra en la **Fig. 3.26**.

Debido a que la trayectoria óptima del robot submarino se realizará en el plano $X - Y$, tenemos que definir límites de movimiento para evitar sobrecargar la cantidad de datos necesarios para el entrenamiento. Con ese objetivo, se redujo el horizonte de generación de data según la Tabla 3.2.

Tabla 3.2: Limites de variables para generación de datos aleatorios

Límite	x	y	ψ	u	v	w	$T1_0$	$T2_0$	$T3_0$	$T4_0$
Superior	10	10	0.5	0.5	0.5	0.5	1	1	1	1
Inferior	-10	-10	-0.5	-0.5	-0.5	-0.5	-1	-1	-1	-1

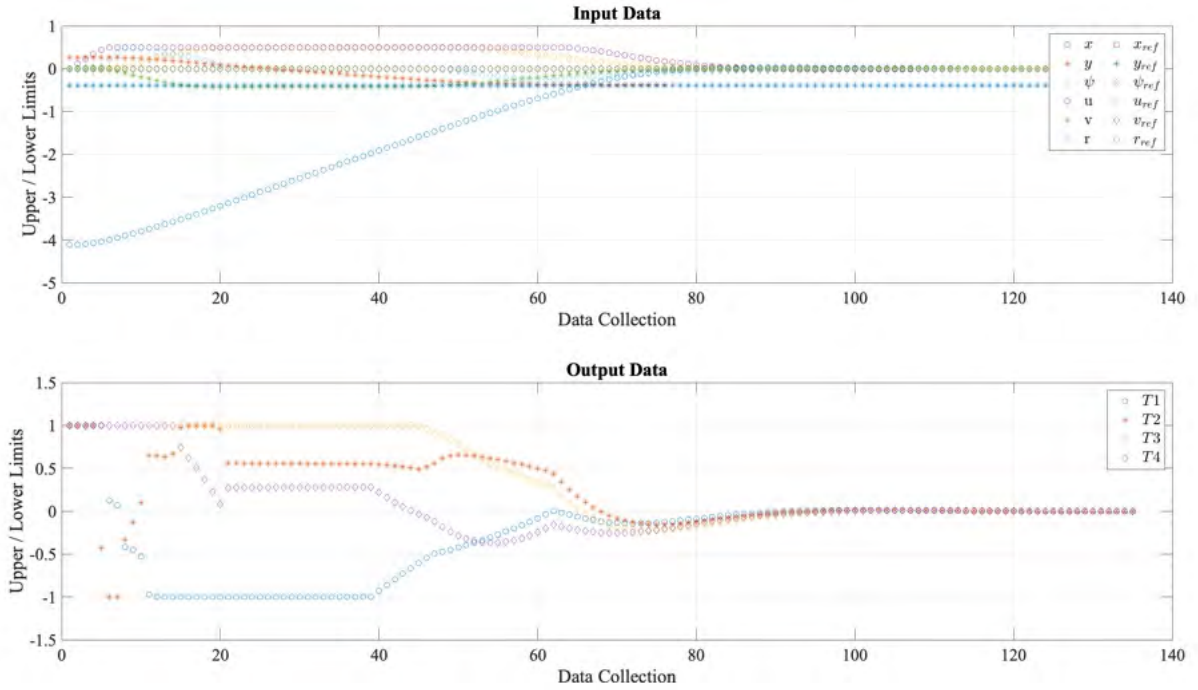


Figura 3.23: Datos de ingreso y salida para una (01) simulación $run = 1$

3.2.2. Creación de la Red Neuronal Profunda

El objetivo de la presente sección es generar la Red Neuronal Profunda, este tipo de redes está compuesta por una cantidad de capas ocultas superiores a seis. Para esta tesis se consideró una cantidad de 14 capas, tal como se muestra en la Figura 3.27, la descripción de cada una de las capas que conforma la DNN se indica a continuación:

- `Observation`: capa de entrada de la red neuronal.
- `FullyConnectedLayer`: capa oculta de 256 neuronas que multiplica la entrada por una matriz de ponderación y luego agrega un vector de sesgo.
- `ReluLayer`: función de activación de la red neuronal.
- `fcLast`: capa de salida de la red neuronal.

Con la información presentada preliminarmente procedemos a crear la red neuronal profunda. Para la generación de la capa utilizamos el script `DNNevaluation.m` adjunto en el Apéndice del presente documento, el cual fue elaborado tomado en cuenta el trabajo de Mathworks [2024].

Los principales valores que debemos considerar se indican en la Tabla 3.3.

El siguiente paso es incluir los parámetros de entrenamiento para la DNN. Para la presente tesis se ha considerado utilizar el optimizador `Adam` (Adaptive Moment Estimation). Los

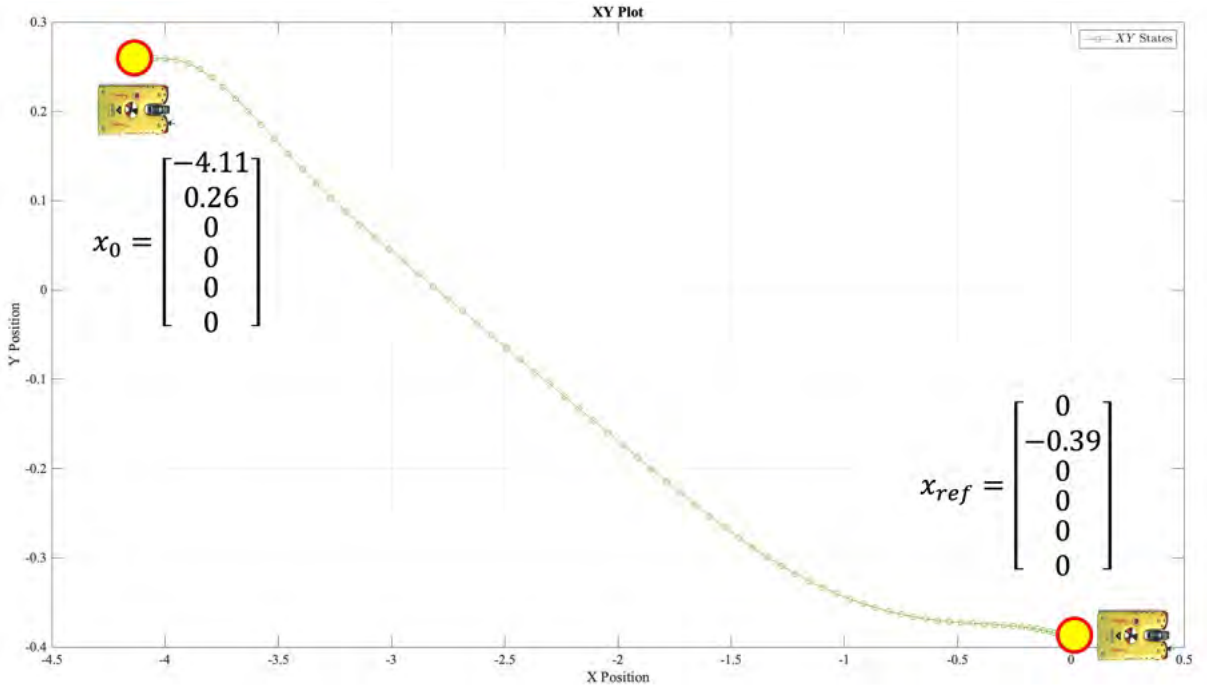


Figura 3.24: Movimiento en el plano X-Y para una (01) simulación $run = 1$

Tabla 3.3: Datos de la Red Neuronal Profunda

Dato	Valor	Descripción
`numObservations`	12	Número de datos de entrada
`numActions`	4	Número de señales de control
`hiddenLayerSize`	14	Número de capas ocultas
`actionFunction`	ReLU	Función de activación

parámetros que debemos de considerar para el optimizador se indican en la Tabla 3.4.

3.2.3. Entrenamiento de la Red Neuronal Profunda

Como último paso procedemos a entrenar la DNN utilizando el script `DNNevaluation.m` incluido en el Anexo del presente documento, el cual se basa en el trabajo de

Debido a la gran cantidad de datos que se generó, el proceso de entrenamiento puede prolongarse por un largo periodo dependiendo de las características del equipo que lo proceso.

Para este caso se ha utilizado un Equipo: MacBook Pro, con Procesador M2 Pro, Memoria: 16 GB y Tarjeta gráfica: Apple M2 de 16 núcleos. Los resultados del entrenamiento se muestran en la **Fig. 3.29**, con estas prestaciones de máquina se realizó el entrenamiento

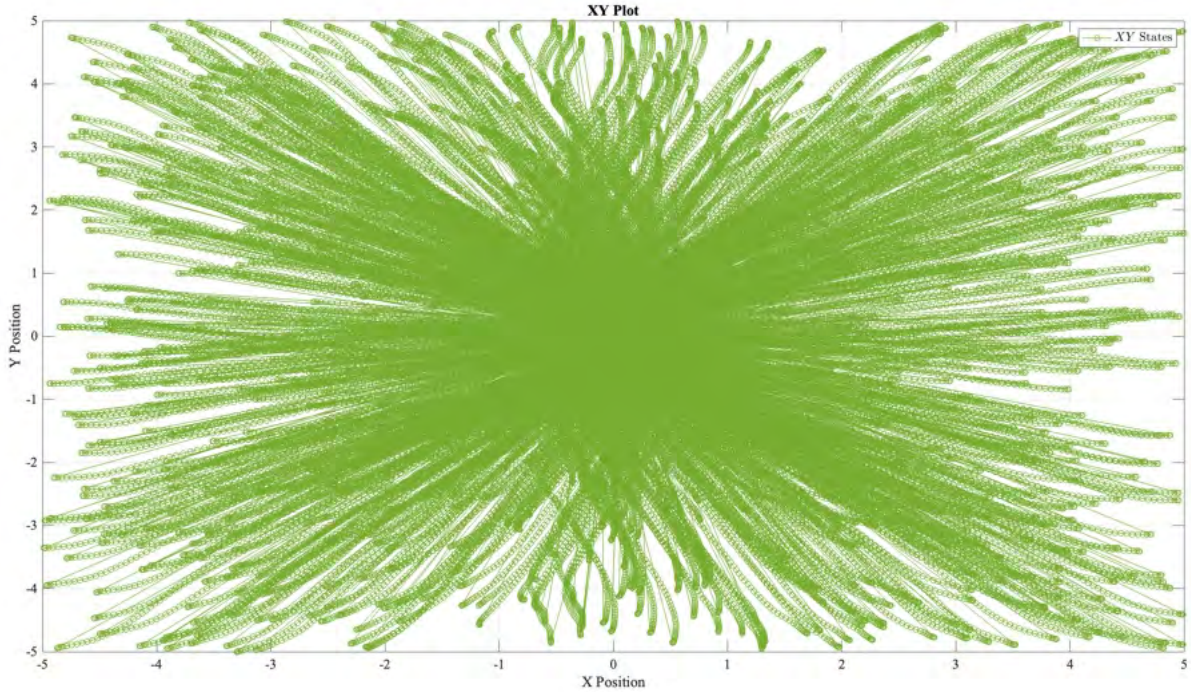


Figura 3.25: Base de Datos en Plano XY

Tabla 3.4: Parámetros para entrenamiento

Dato	Valor	Descripción
`solverName`	Adam	Optimizador
`Verbose`	false	Indicador que muestra progreso de entrenamiento
`Plots`	training-progress	Muestra progreso de entrenamiento
`Shuffle`	every-epoch	Mezcla de datos de entrenamiento y validación
`MiniBatchSize`	8192	Tamaño del miniBatch usado en cada iteración de entrenamiento
`ValidationData`	ValidationCellArray	Data usada para validación durante entrenamiento
`InitialLearnRate`	1e-3	Ratio inicial de entrenamiento
`ExecutionEnvironment`	cpu	Uso de CPU
`GradientThreshold`	10	Umbral positivo para el gradiente
`MaxEpochs`	500	Máximo número de épocas por entrenamiento

en 21 minutos y 35 segundos.

La performance de entrenamiento se evalúa en función del RMSE (`Root Mean Square`

`Error - Raíz del Error Cuadrático Medio` , el cual representa la raíz cuadrada del segundo momento de la muestra de las diferencias entre los valores previstos y los valores observados o la media cuadrática de estas diferencias, mientras mas próximo es el valor a 0, podemos deducir que la red tiene un buen desempeño para los datos de entrenamiento y validación.

Adicionalmente, el factor de pérdida o `Loss`, el cual indica qué tan mala fue la predicción del modelo en un solo ejemplo. Si la predicción del modelo es perfecta, la pérdida es cero; de lo contrario, la pérdida es mayor. Tal como se observa en **Fig. 3.29**, la pérdida es cercana a cero, por lo cual podemos concluir que el entrenamiento se realizó de manera correcta.

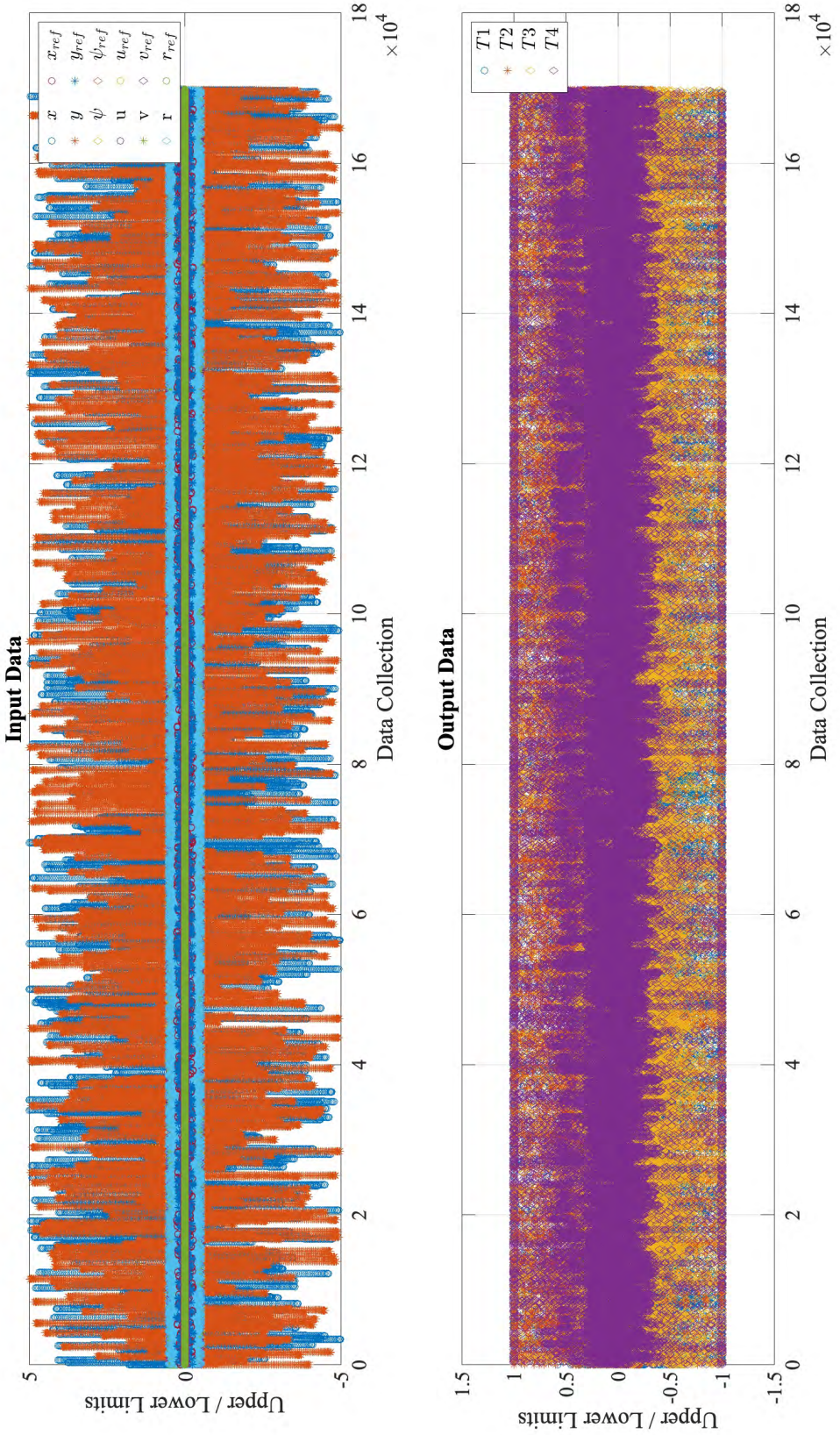
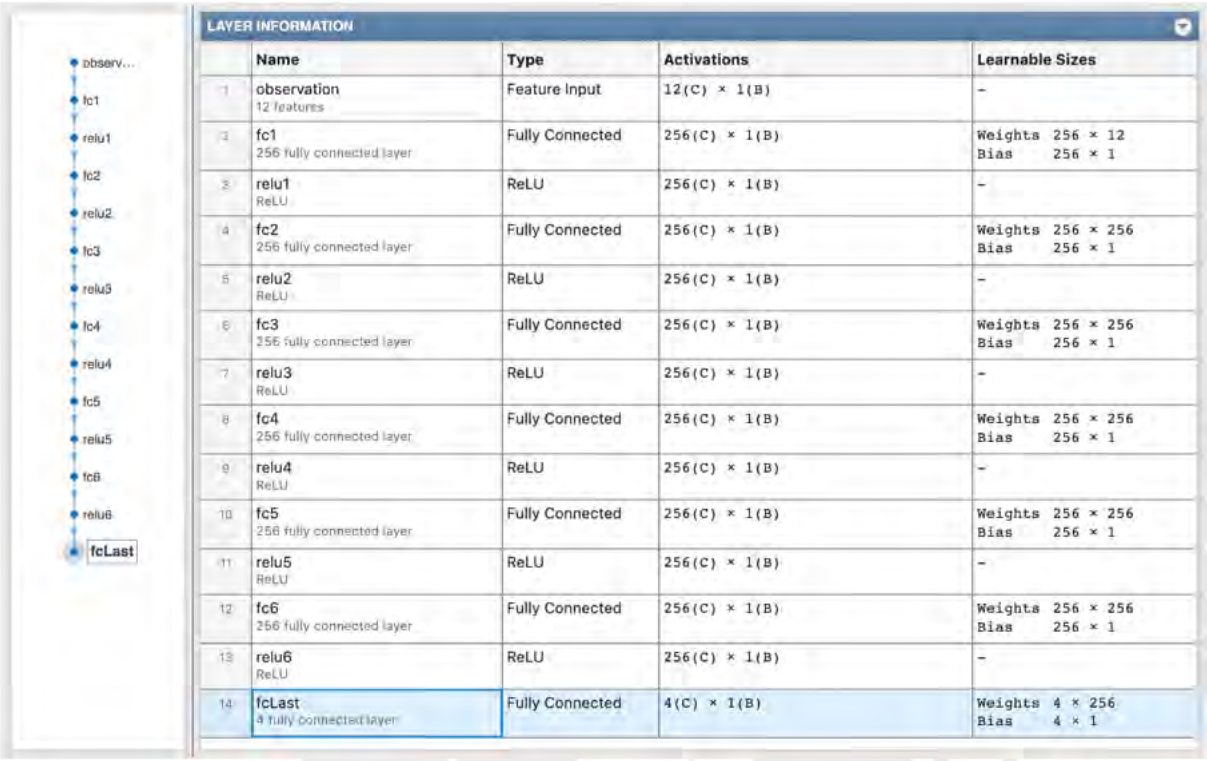


Figura 3.26: Base de datos de señales de entrada y salida, 2.7 millones de datos



LAYER INFORMATION

	Name	Type	Activations	Learnable Sizes
1	observation 12 features	Feature Input	12(C) × 1(B)	-
2	fc1 256 fully connected layer	Fully Connected	256(C) × 1(B)	Weights 256 × 12 Bias 256 × 1
3	relu1 ReLU	ReLU	256(C) × 1(B)	-
4	fc2 256 fully connected layer	Fully Connected	256(C) × 1(B)	Weights 256 × 256 Bias 256 × 1
5	relu2 ReLU	ReLU	256(C) × 1(B)	-
6	fc3 256 fully connected layer	Fully Connected	256(C) × 1(B)	Weights 256 × 256 Bias 256 × 1
7	relu3 ReLU	ReLU	256(C) × 1(B)	-
8	fc4 256 fully connected layer	Fully Connected	256(C) × 1(B)	Weights 256 × 256 Bias 256 × 1
9	relu4 ReLU	ReLU	256(C) × 1(B)	-
10	fc5 256 fully connected layer	Fully Connected	256(C) × 1(B)	Weights 256 × 256 Bias 256 × 1
11	relu5 ReLU	ReLU	256(C) × 1(B)	-
12	fc6 256 fully connected layer	Fully Connected	256(C) × 1(B)	Weights 256 × 256 Bias 256 × 1
13	relu6 ReLU	ReLU	256(C) × 1(B)	-
14	fcLast 4 fully connected layer	Fully Connected	4(C) × 1(B)	Weights 4 × 256 Bias 4 × 1

Figura 3.27: Estructura de la Red Neuronal

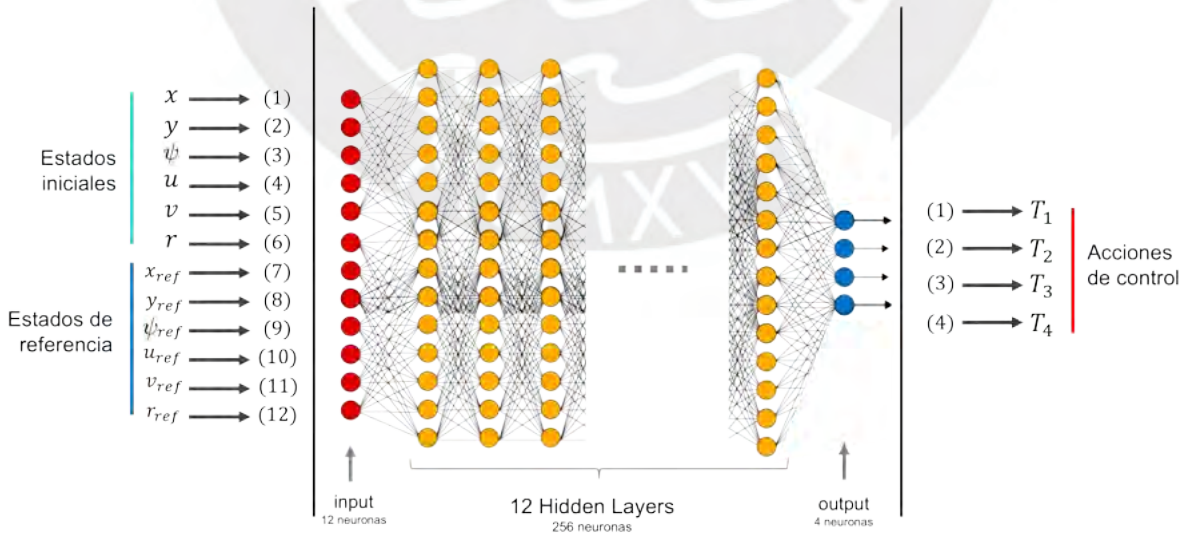


Figura 3.28: Estructura gráfica de la Red Neuronal

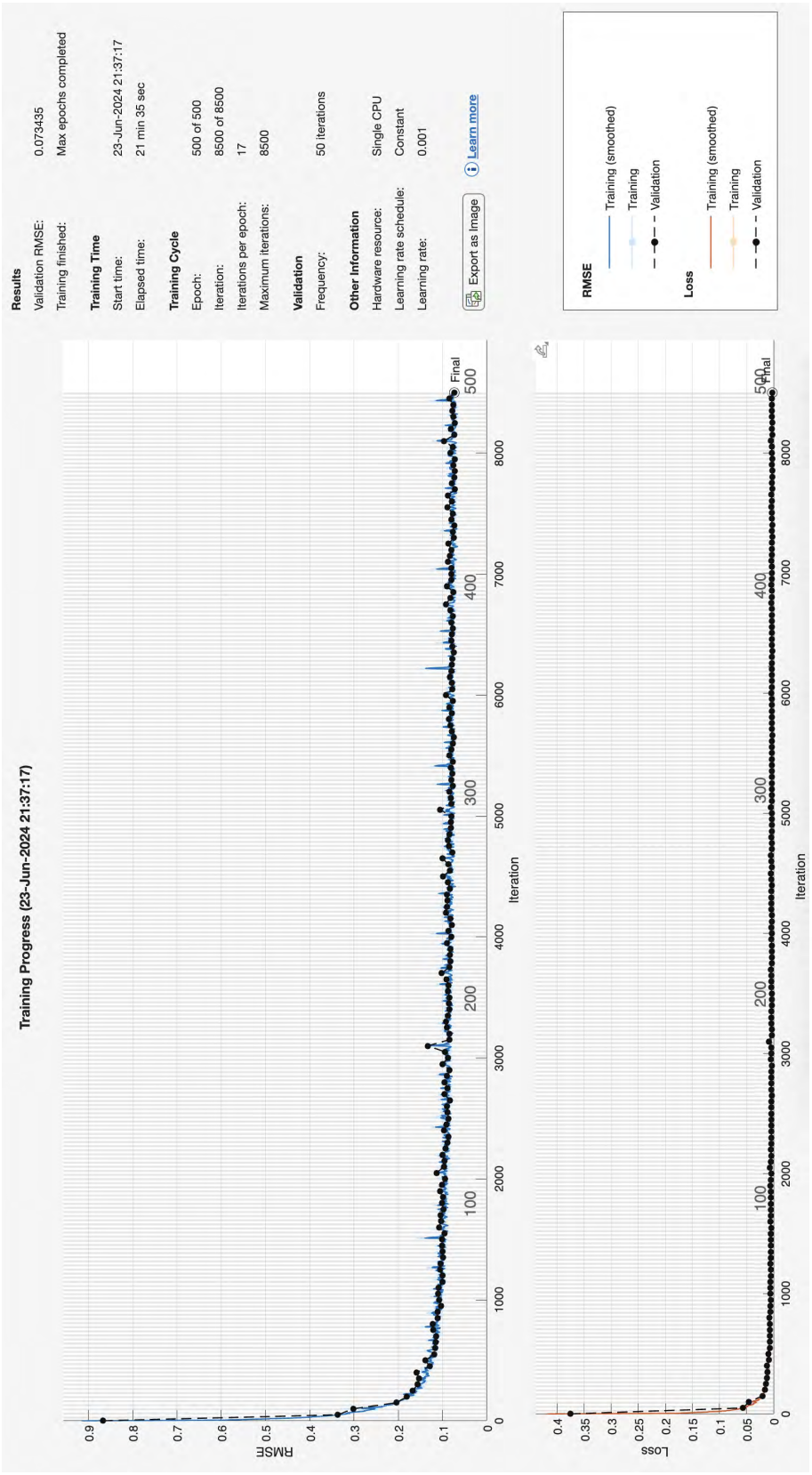
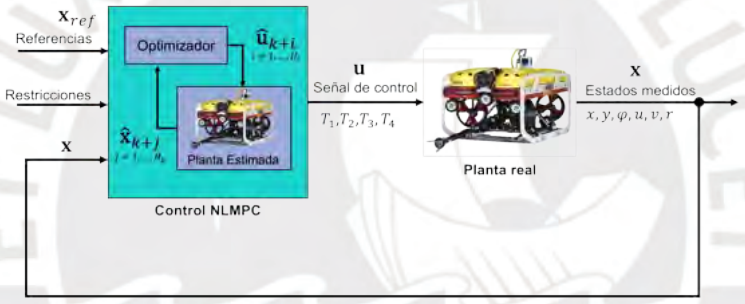


Figura 3.29: Entrenamiento de la Red Neuronal Profunda

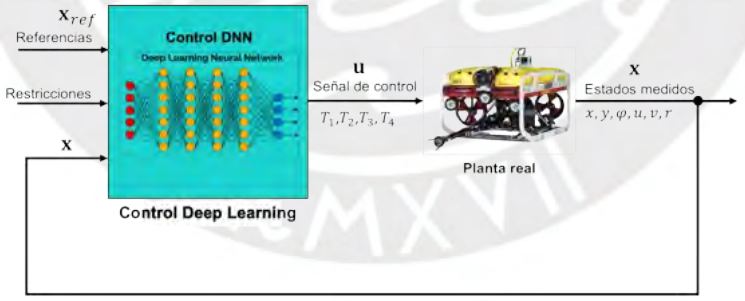
CAPÍTULO 4

SIMULACIÓN Y EVALUACIÓN

El presente capítulo tiene como objetivo comparar los resultados obtenidos mediante la simulación con el Control Predictivo Basado en Modelos No Linean (NLMPC) y el control por Redes Neuronales Profundas (DL, *Deep Learning*), ambos modelos representados en la **Fig. 4.1**.



(a) Control Predictivo Basado en Modelos



(b) Control por Redes Profundas

Figura 4.1: Modelos de control de Robot Submarino No Tripulado, UVV

Para realizar la validación de los resultados, procedemos a generar valores aleatorios de estados iniciales y estados objetivo, conocidos como estados referenciales. El objetivo es verificar el comportamiento de los estados de la planta con el control NLMPC y DL para cumplir con el objetivo de control.

4.1. Datos de Simulación

Para realizar la simulaciones en el plano XY se utilizarán los valores aleatorios de los estados iniciales, representados por x, y, ψ, u, v, w, y los estados de referencia representados por $x_r, y_r, \psi_r, u_r, v_r, w_r$. Todos los valores de los estados son indicados en la Tabla 4.1.

Tabla 4.1: Datos aleatorios para la Simulación

Simulación	x	y	ψ	u	v	w	x_r	y_r	ψ_r	u_r	v_r	w_r
1	4.0	3.0	0.1	0.0	0.0	0.0	0.5	-0.5	0	0	0	0
2	-2.0	5.0	0.2	0.0	0.0	0.0	0.0	0.25	0	0	0	0
3	2.5	3.0	0.2	0.0	0.0	0.0	0.2	0.1	0	0	0	0
4	5.0	3.5	0.1	0.0	0.0	0.0	-0.2	-0.1	0	0	0	0
5	-4.5	-2.5	-0.1	0.0	0.0	0.0	0.1	0.15	0	0	0	0

En base a los datos indicados en la Tabla 4.1 se realizaron las diferentes simulaciones del robot submarino UUV con el control con redes neuronales profundas (*Deep Learning*). Los resultados obtenidos se analizarán con el comportamiento de los estados con el Control Predictivo por Modelos no Lineal.

4.2. Simulación de Sistema

Todas las simulaciones fueron realizadas utilizando el método de optimización `Interior Point` suministrada por IPOPT [Wachter and Biegler, 2006] asociada con `Matlab` a través del `Framework` CasADi, [Andersson, 2013].

La **Fig. 4.2** muestra el resultado de la simulación 1. Adicionalmente, las **Fig. 4.3 y 4.4** muestran los estados de la planta y señales de control respectivamente para la simulación 1, tanto para el control no lineal MPC como para el control por DNN. Como se puede observar los valores de todos los estados de la planta y señales de control tienen un alto grado de semejanza, esto demuestra que el grado de aprendizaje del controlador DNN es muy alto y se asemeja al control no lineal MPC pero con tiempos de respuesta mas cortos y costos menores de cómputo.

Los resultados de todas las simulaciones se muestran en las **Fig. 4.5, 4.6, 4.7** y **4.8**.

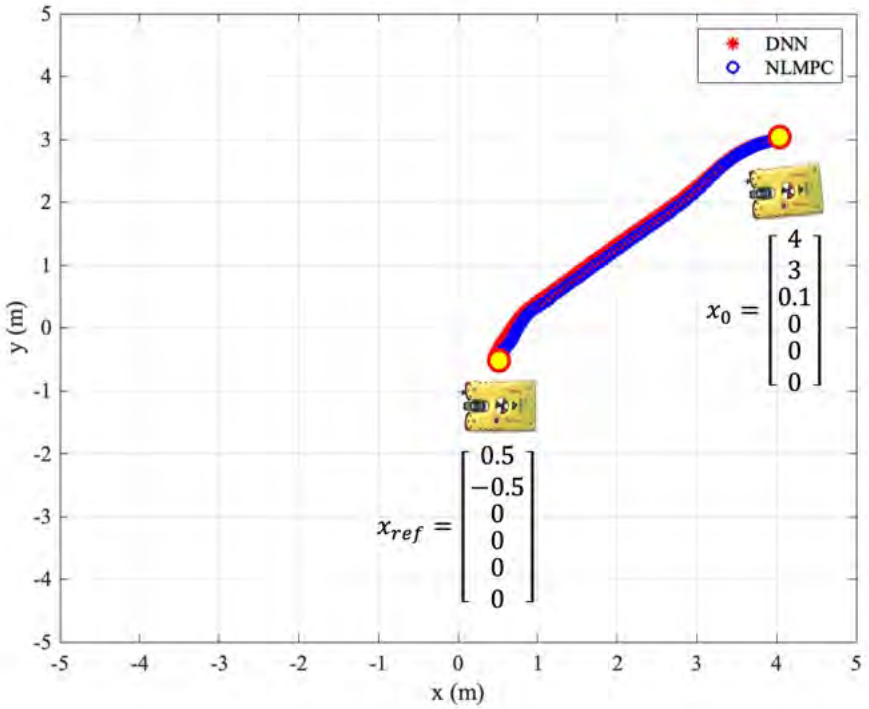


Figura 4.2: Resultados de Simulación 1

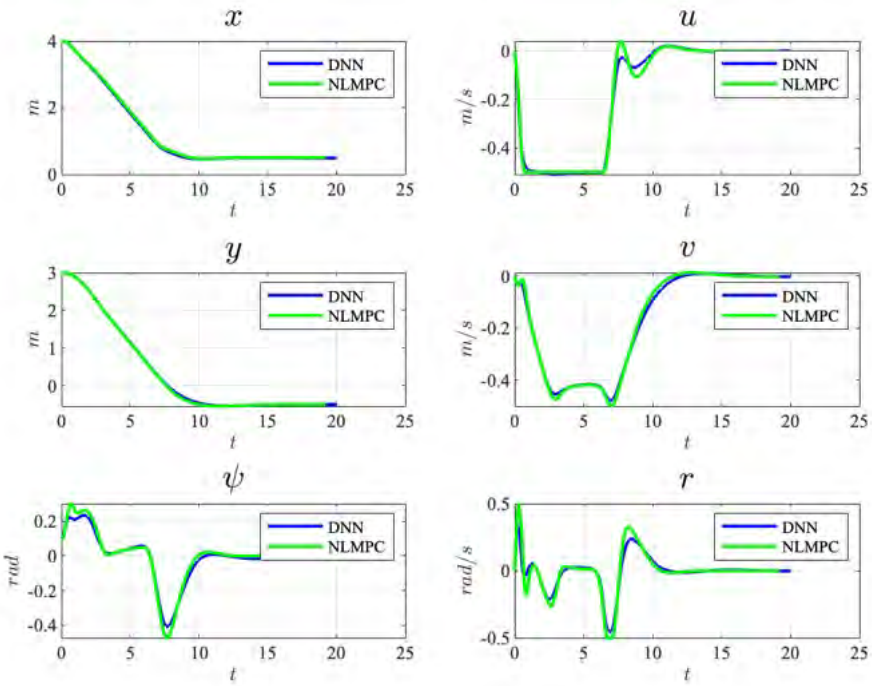


Figura 4.3: Simulación 1 - Estados

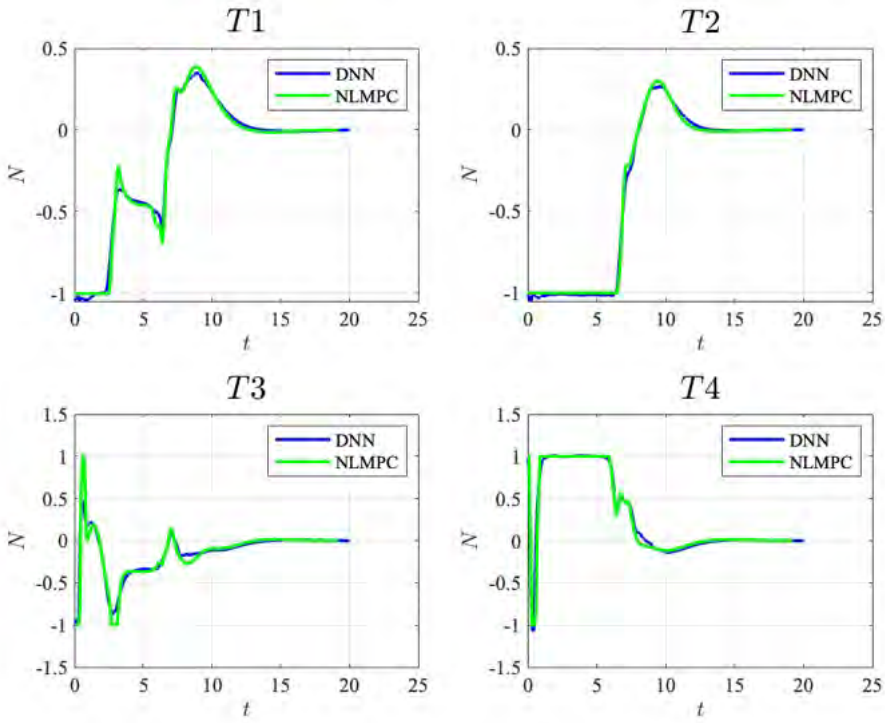


Figura 4.4: Simulación 1 - Control

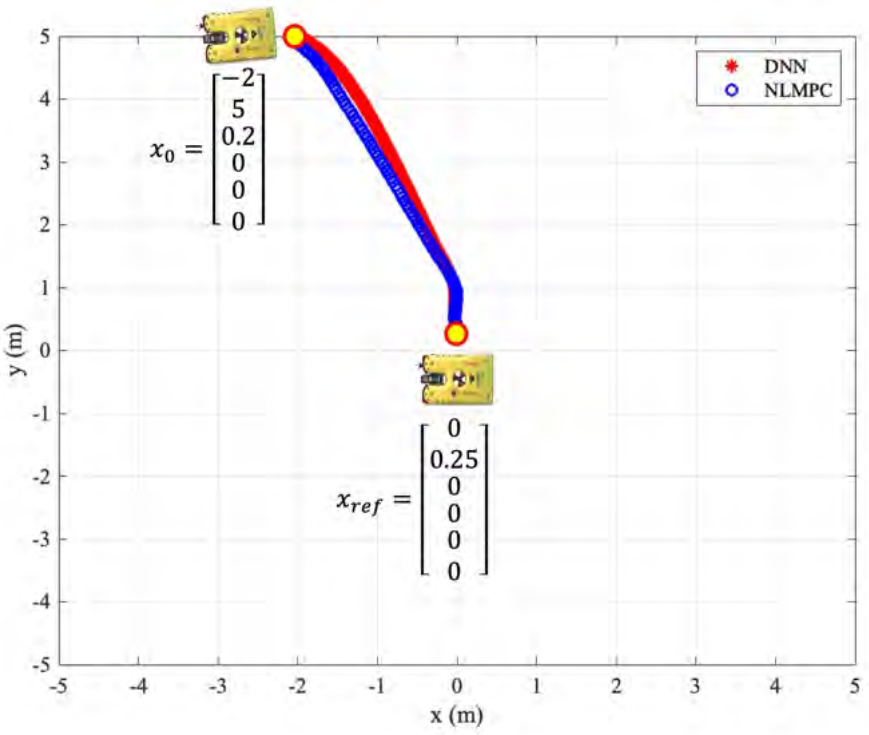


Figura 4.5: Resultados de Simulación 2

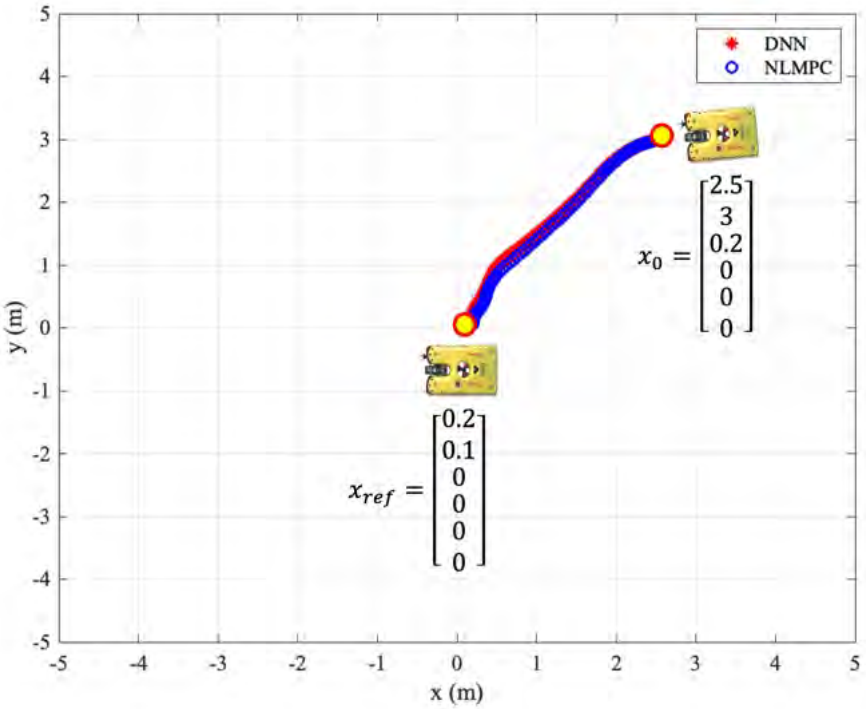


Figura 4.6: Resultados de Simulación 3

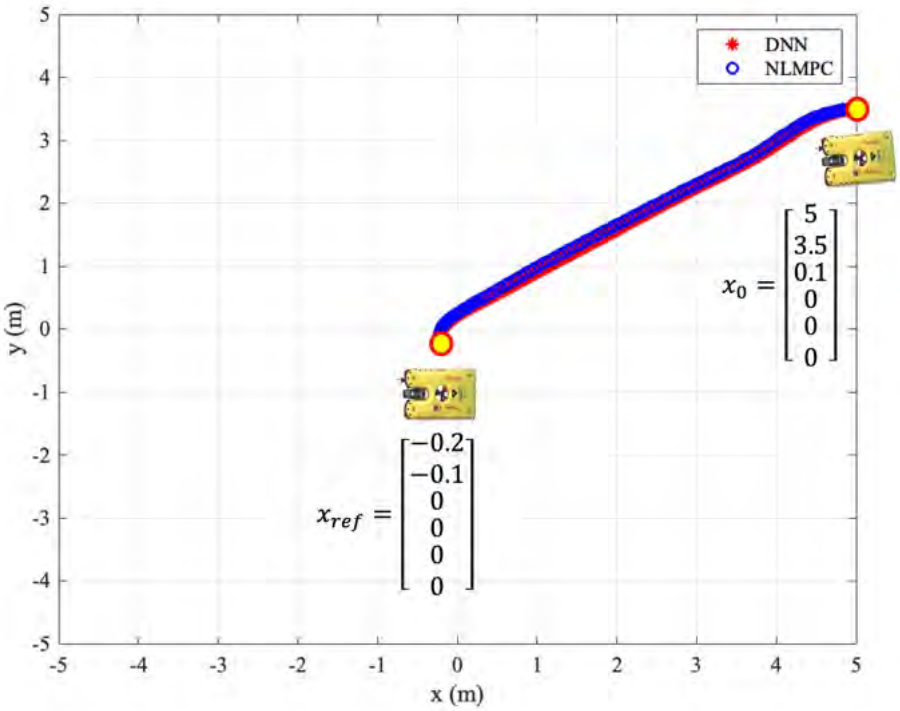


Figura 4.7: Resultados de Simulación 4

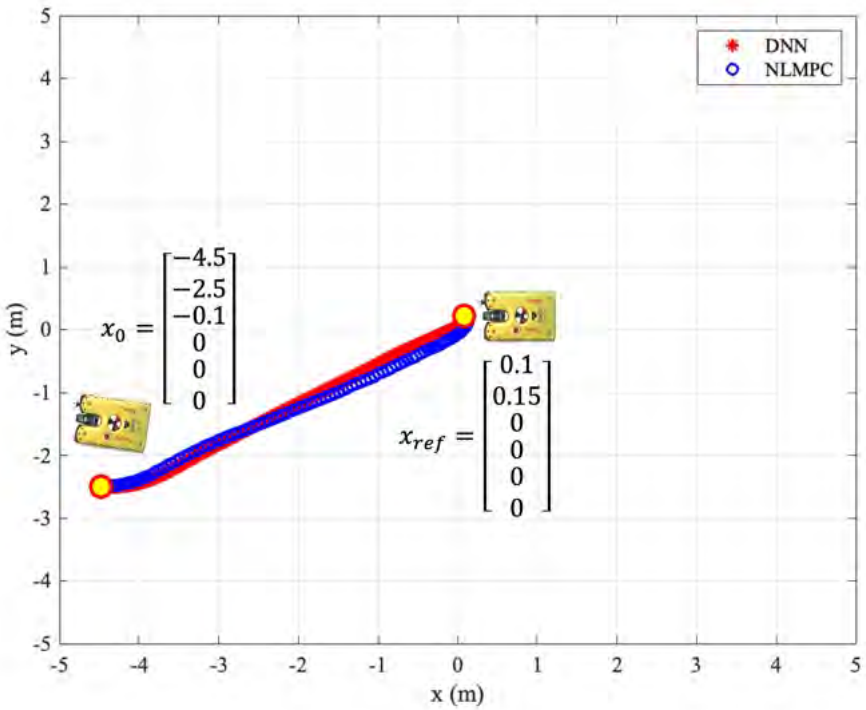


Figura 4.8: Resultados de Simulación 5

4.3. Evaluación de los Resultados

De la revisión de los resultados, podemos determinar que la Red Neuronal Profunda (*Deep Learnign*) tuvo un adecuado aprendizaje, esto se puede apreciar en las gráficas de las trayectorias para ámbos controles. Como se puede ver el control por Redes Profundas pudo seguir las trayectorias óptimas generadas por el control no lineal MPC. Esto también se puede corroborar en los resultados mostrados en la **Fig. 4.4** y **Fig. 4.3**, donde se puede apreciar la similitud que existe en los estados de la planta y las señales de control para ambos tipos de controladores. Lo mismo se replica en todas las otras simulaciones, donde se puede apreciar que el control DNN puede imitar de manera adecuada el comportamiento del control NLMPC.

Un dato importante que podemos notar de todas las simulaciones es el tiempo de procesamiento de cada una de ellas. De los resultados de simulación mostrados en las **Fig. 4.5**, **Fig. 4.6**, **Fig. 4.7** y **Fig. 4.8**, podemos notar que los tiempos de ejecución de ambos controles varían significativamente debido al requerimiento de memoria que necesita el control NLMPC para poder realizar la optimización en línea y obtener el estado de control futuro; a diferencia del control por Redes Neuronales Profundas, donde sólo se resume a una multiplicación de las señales de entrada por la red neuronal para

obtener las señales de control.

El resumen de los tiempos de ejecución se muestran en la Tabla 4.2. Como podemos notar en una comparación entre el tiempo de procesamiento, el control MPC No Lineal resultó ser entre 5.2 y 9.5 veces mas lento que el control por Redes Neuronales Profundas, esto se traduce en una variación de tiempo entre 417 % a 850 % a favor del control por Redes Neuronales Profundas.

Tabla 4.2: Tiempos Ejecución de Simulaciones

Ítem	t_{NLMPC}, [seg]	t_{DNN}, [seg]	$\frac{t_{NLMPC}}{t_{DNN}}$	$\frac{\Delta t_{NLMPC-DNN}}{t_{DNN}}$
Simulación 1	11.320970	1.191308	9.5	850 %
Simulación 2	8.602545	1.148033	7.5	649 %
Simulación 3	6.054583	1.171043	5.2	417 %
Simulación 4	8.222687	1.138356	7.2	622 %
Simulación 5	7.254374	1.147181	6.3	532 %

CONCLUSIONES Y TRABAJO FUTURO

1. Las ecuaciones que gobiernan el modelamiento de un robot submarino no tripulado o UUV presentan no linealidades que hacen muy desafiante la selección de controladores adecuados para su operación. Usualmente, los UUVs son controlados mediante complejos algoritmos que a menudo requieren la implementación de un control automático de velocidad y posicionamiento, así como de sistemas de control para determinar la ubicación del UVV según la profundidad y altitud sobre el lecho marino.

2. El presente trabajo demostró que el Control Predictivo por Modelo no Lineal - NLMPC (*No Linear Model Predictive Control*), puede ser usado para el control de robots submarinos no tripulados con dinámica del sistema no lineal. Este tipo de control nos permite incluir en el modelo las restricciones físicas que pueden existir en las señales de control y estados físicos de la planta. Estas restricciones pueden ser la potencia de los impulsores, las restricciones en la velocidad y aceleración máxima, así como las limitaciones de espacio en el plano X-Y.

3. El procedimiento seguido en el presente trabajo para realizar el control del robot submarino fue representar al control no lineal MPC como un problema de programación no lineal (NLP - *Non Lineal Programming*). Para esto se utilizó el *Framework* CasADi y el algoritmo de optimización de Punto Interior IPOPT (*Interior Point OPTimizer*). De las pruebas ejecutadas con el MPC no lineal en la planta, se concluye que este controlador cumple con el control de movimiento en el plano XY. Sin embargo, el principal problema que tiene el control MPC no lineal es el alto costo computacional para su implementación debido a la necesidad de aplicar herramientas de optimización en linea.

4. En el presente trabajo se demostró que los algoritmos de *Deep Learning* permiten entrenar una red neuronal profunda para que pueda aprender los patrones generados por el Control Predictivo por Modelo no Lineal. Esta característica se debe a que estos algoritmos son capaces de reconocer patrones complejos en una gran cantidad de datos. En base a los buenos resultados del control no lineal MPC, se generó un

script para poder generar una base de 1.8 millones de datos que sirvió como fuente de entrenamiento y validación para el aprendizaje de las redes neuronales profundas.

5. Luego del proceso de entrenamiento y validación de la red neuronal profunda con la bases de datos, se procedió a la verificación con datos aleatorios. Los resultados de las simulaciones demostraron que el tiempo computacional para el control del UUV se reduce entre cinco a siete veces cuando se trabaja con el controlador de redes neuronales profundas en comparación con los tiempos del control no lineal MPC.

6. El presente trabajo abordó el control de estabilización de estado (*Stabilization Point*) del robot submarino no tripulado, el cual tiene como objetivo realizar el control para el movimiento del robot desde un estado a otro. Sin embargo, se sugiere ampliar el estudio para el tipo de control de trayectoria empleando el mismo procedimiento de recopilación de datos y entrenamiento de una red neuronal con una gran cantidad de datos.

7. Como parte de un trabajo futuro, se sugiere ampliar el rango de simulación del plano XY al plano XYZ. Para esto, el modelo de la planta debe ser ampliado para que pueda mostrar los estados en los tres ejes, pasando de los seis estados del plano XY a los doce estados en el plano XYZ. Adicionalmente, se deberá generar una mayor cantidad de data de entrenamiento para que el control por redes neuronales profundas pueda ser entrenado para controlar una mayor cantidad de estados.

8. Adicionalmente a lo indicado previamente, se sugiere evaluar un mayor número de datos en base a rangos de evaluación del plano XY más amplios en todos los estados y salidas de la planta. El presente trabajo se limitó a considerar límites de los estados x, y de plano XY entre -5 a 5 para poder demostrar la correcta aplicación de redes neuronales profundas para el control de la planta. Sin embargo, se podría extender los límites del plano, para esto es importante poder trabajar con procesadores más potentes que sean capaces de generar la data con el procesador MPC no lineal y entrenar a la red neuronal profunda con toda la base de datos generada.

BIBLIOGRAFÍA

[Aleksander, 1990] I Aleksander. An introduction to neural computing, 1990.

[Amer et al., 2012] Ahmed F. Amer, Elsayed A. Sallam, and Wael M. Elawady. Quasi sliding mode-based single input fuzzy self-tuning decoupled fuzzy pi control for robot manipulators with uncertainty. *International Journal of Robust and Nonlinear Control*, 22:2026–2054, 12 2012. ISSN 10498923. doi: 10.1002/rnc.1805.

[Andersson, 2013] Joel Andersson. A general-purpose software framework for dynamic optimization. 10 2013.

[Andersson et al., 2011] Joel Andersson, Johan Åkesson, Francesco Casella, and Moritz Diehl. Integration of casadi and jmodelica.org. In *Proceedings of the 8th International Modelica Conference*, 2011.

[Antonelli, 2006] Gianluca Antonelli. *Underwater Robots: Motion and Force Control of Vehicle-Manipulator Systems*. Springer, second edition, 2006. ISBN 139783540317524.

[Betts, 2010] John T Betts. *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming*. Society for Industrial and Applied Mathematics, second edition edition, 2010. ISBN 9780898716887.

[Bock and Plitt, 1984] H. G. Bock and K. J. Plitt. A multiple shooting algorithm for direct solution of optimal control problems. In *Proceedings of the 9th IFAC World Congress*, pages 242–247, 1984.

[Bryson, 1986] Lindsay Bryson. All at sea. *IEE Proceedings*, 133, 1986.

[Caccia and Veruggio, 2000] M Caccia and G Veruggio. Guidance and control of a reconfigurable unmanned underwater vehicle. *Control Engineering Practice*, 8, 2000.

[Camacho and Bordons, 2010] Eduardo Camacho and Carlos Bordons. Control predictivo: Pasado, presente y futuro. *Revista Iberoamericana de Automática e Informática Industrial (RIAI)*, 2010. doi: 10.4995/riai.v1i3.10587.

[Christ and Wernli, 2014] Robert D.. Christ and Robert L.. Wernli. *The ROV Manual, A*

user guide for remotely operated vehicles. Elseiver, second edition, 2014. ISBN 9780080982885.

[Correa, 2016] Max Correa. High performance implementation of mpc schemes for fast systems. 3 2016.

[Diehl, 2007] Moritz Diehl. Fast nonlinear model predictive control algorithms and applications in process engineering, 2007.

[Fossen, 1994] Thor I Fossen. *Guidance and Control of Ocean Vehicles*. 1994.

[Gafurov and Klochkov, 2015] Salimzhan A. Gafurov and Evgeniy V. Klochkov. Autonomous unmanned underwater vehicles development tendencies. In *Procedia Engineering*, volume 106, pages 141–148. Elsevier Ltd, 2015. doi: 10.1016/j.proeng.2015.06.017.

[Goodfellow et al., 2016] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[Hassanein et al., 2011] O Hassanein, Sreenatha Anavatti, and Tapabrata Ray. Fuzzy modeling and control for autonomous underwater vehicle. In *Proceedings of the 5th International Conference on Automation, Robotics and Applications*. IEEE, 2011. ISBN 978145770330011.

[Hinton et al., 2006] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation, MIT*, 2006.

[Huang, 2010] Rui Huang. Nonlinear model predictive control and dynamic real time optimization for large-scale processes, 12 2010.

[Hyakudome, 2011] Tadahiro Hyakudome. Design of autonomous underwater vehicle, 2011. URL `www.intechopen.com`.

[Jordan and Bustamante, 2009] Mario Alberto Jordan and Jorge Luis Bustamante. Adaptive control for guidance of underwater vehicles. *Underwater Vehicles*, 2009.

[Kingma and Ba, 2014] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. 12 2014. URL `http://arxiv.org/abs/1412.6980`.

[Kirk, 2004] Donald Kirk. *Optimal Control Theory, An Introduction*. Dover Publications, dover publications edition, 2004.

[Koutrik, 2015] S Van Koutrik. Optimal control for race car minimum time maneuvering. 3 2015.

[Leyffer and Mahajan, 2010] Sven Leyffer and Ashutosh Mahajan. Nonlinear constrained optimization: Methods and software. 2010.

[Liu et al., 2014] Yuhong Liu, Pingping Fang, Dongdong Bian, Hongwei Zhang, and Shuxin Wang. Fuzzy comprehensive evaluation for the motion performance of autonomous underwater vehicles. *Ocean Engineering*, 88:568–577, 9 2014. ISSN 00298018. doi: 10.1016/j.oceaneng.2014.03.013.

[Loebis et al., 2004] D. Loebis, R. Sutton, J. Chudley, and W. Naeem. Adaptive tuning of a kalman filter via fuzzy logic for an intelligent auv navigation system. *Control Engineering Practice*, 12:1531–1539, 2004. ISSN 09670661. doi: 10.1016/j.conengprac.2003.11.008.

[Lorentz and Yuh, 1996] Jorgen Lorentz and J Yuh. A survey and experimental study of neural network auv control, 1996.

[Mathworks, 2023] Mathworks. Control of quadrotor using nonlinear model predictive control, 2023. URL `https://www.mathworks.com/help/mpc/ug/control-of-quadrotor-using-nonlinear-model-predictive-control.html`. Accessed December 12, 2023.

[Mathworks, 2024] Mathworks. Imitate nonlinear mpc controller for flying robot, 2024. URL `https://www.mathworks.com/help/reinforcement-learning/ug/imitate-nonlinear-mpc-controller-for-flying-robot.html`. Accessed June 12, 2024.

[McGinnis et al., 2014] Tim McGinnis, Nick Michel-Hart, Michael Mathewson, and Tim Shanahan. Deep profiler for the ocean observatories initiative regional scale nodes: Rechargable, adaptive, rov servicable. 2014.

[Mehrez, 2019] Mohamed Mehrez. Optimization based solutions for control and state estimation in dynamical systems (implementation to mobile robots) a workshop. 01 2019. doi: 10.13140/RG.2.2.21613.23521.

[Mei et al., 2018] Song Mei, Andrea Montanari, and Phan Minh Nguyen. A mean field view of the landscape of two-layer neural networks. *Proceedings of the National Academy of Sciences of the United States of America*, 115:E7665–E7671, 8 2018. ISSN 10916490. doi: 10.1073/pnas.1806579115.

[Messac, 2015] Achille Messac. *Optimization in practice with Matlab® for engineering students and professionals*. Cambridge University Press, 2015.

[Nocedal and Wright, 1999] Jorge Nocedal and Stephen J Wright. *Numerical Optimization*. Springer, 1999.

[Ostensjo, 2011] Steffen Ostensjo. Guidance and control strategies for uuvs, 6 2011.

[Ramos, 2007] César Ramos. Control predictivo basado en modelos (cpbm) robusto con bdu. 1 2007.

[Rao, 2010] Anil V Rao. A survey of numerical methods for optimal control. *Advances in the Astronautical Sciences*, 2010. URL `https://www.researchgate.net/publication/268042868`.

[Rosolia et al., 2017] Ugo Rosolia, Stijn De Bruyne, and Andrew G. Alleyne. Autonomous vehicle control: A nonconvex approach for obstacle avoidance. *IEEE Transactions on Control Systems Technology*, 25:469–484, 3 2017. ISSN 10636536. doi: 10.1109/TCST.2016.2569468.

[Rumelhart et al., 1986] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986. URL `https://api.semanticscholar.org/CorpusID:205001834`.

[Salgado et al., 2010] T Salgado, L Gonzalez, F Martínez, E Olguín, A Resendiz, and M Bandala. Deep water rov design for the mexican oil industry. 2010.

[Shahrieel et al., 2015] Mohd Shahrieel, Shahrum Shah Abdullah, and F. A. Azis. Review on auto-depth control system for an unmanned underwater remotely operated vehicle (rov) using intelligent controller. *Journal of Telecommunication, Electronic and Computer Engineering*, 7:47–55, 2015. ISSN 21801843.

[Shi et al., 2006] Yang Shi, Weiqi Qian, Weisheng Yan, and Jun Li. Adaptive depth control for autonomous underwater vehicles based on feedforward neural networks. *Intelligent Control and Automation*, pages 207–218, 2006.

[Tang et al., 2015] Jianzhong Tang, Youfang Yu, and Yong Nie. An autonomous underwater vehicle docking system based on optical guidance. *Ocean Engineering*, 104:639–648, 6 2015. ISSN 00298018. doi: 10.1016/j.oceaneng.2015.06.001.

[Tehrani et al., 2010] N. H. Tehrani, Mahdi Heidari, Yadollah Zakeri, and Jafar Ghaisari. Development, depth control and stability analysis of an underwater remotely operated vehicle (rov). *8th IEEE International Conference on Control and Automation*, pages 814–819, 2010.

[Tondel et al., 2001] P Tondel, T A Johansen, and A Bemporad. An algorithm for

multi-parametric quadratic programming and explicit mpc solutions. In *Proceedings of the 40th IEEE Conference on Decision and Control*, pages 1199–1204, 2001.

[Ven et al., 2005] Pepijn W.J. Van De Ven, Colin Flanagan, and Daniel Toal. Neural network control of underwater vehicles. *Engineering Applications of Artificial Intelligence*, 18: 533–547, 8 2005. ISSN 09521976. doi: 10.1016/j.engappai.2004.12.004.

[Verschueren et al., 2016] Robin Verschueren, Mario Zanon, Rien Quirynen, and Moritz Diehl. *Time-optimal Race Car Driving using an Online Exact Hessian based Nonlinear MPC Algorithm*. 2016. ISBN 9781509025916. doi: 10.0/Linux-x86_64.

[Wachter and Biegler, 2006] A Wachter and L Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106:25–57, 5 2006. ISSN 00255610.

[Watson, 2012] Simon Andrew Watson. *Mobile Platforms for Underwater Sensor Networks*. The University of Manchester (United Kingdom), 2012.

[Wynn et al., 2014] Russell B. Wynn, Veerle A.I. Huvenne, Timothy P. Le Bas, Bramley J. Murton, Douglas P. Connelly, Brian J. Bett, Henry A. Ruhl, Kirsty J. Morris, Jeffrey Peakall, Daniel R. Parsons, Esther J. Sumner, Stephen E. Darby, Robert M. Dorrell, and James E. Hunt. Autonomous underwater vehicles (auvs): Their past, present and future contributions to the advancement of marine geoscience. *Marine Geology*, 352: 451–468, 6 2014. ISSN 00253227. doi: 10.1016/j.margeo.2014.03.012.

[Yuh et al., 1998] J Yuh, SK Choi, GH Kim, G McMurtry, M Ghasemi-Nejhad, N Sarkar, K Sugihara, and Holmes Hall. Design of a semi-autonomous underwater vehicle for intervention missions (sauvim). In *Proceedings of the International Symposium of Underwater Technology*, pages 63–68, 1998.

[Özgür Yildiz et al., 2009] Özgür Yildiz, R Bülent Gökalp, and A Egemen Yilmaz. A review on motion control of the underwater vehicles, 2009.

ANEXOS

Anexo 1: Script `NLMPCdatabase.m`: Elaboración de Base de Datos

Contents

- STEP 1.0: NLMPC CONFIGURATION
- STEP 2.0: NLMPC DATABASE
- STEP 3.0: PLOT INPUT DATA
- STEP 4.0: PLOT TRACKING DATA


```
% ************************************************************************
% Universidad: Pontificia Universidad Católica del Perú
% Tesis:       Control de Aprendizaje Profundo basado en un Control Predictivo por Modelo No Lineal
% Curso:       Maestría en Ingeniería Mecatrónica
% Alumno:      Manuel Enrique Gallardo Rodriguez
% ************************************************************************
```

STEP 1.0: NLMPC CONFIGURATION

```
--------------------------------------
Method: Multiple Shooting
Control Strategy: Point Stabilization
Framework: CaSAdi
Optimization: IPOPT
```

```
clear all, close all, clc
addpath('/Users/3nrique/Desktop/Thesis/2024/10 Casadi')
import casadi.*

T = 0.1;            % sampling time [s]
N = 50;             % prediction horizon
rob_diam = 1;       % for drawing

% STEP 1.1: Create "f" Function
% ------------------------------
    % STATES Symbology
    x = SX.sym('x'); y = SX.sym('y'); psi = SX.sym('psi');
    u = SX.sym('u'); v = SX.sym('v');   r = SX.sym('r');
    states = [x;y;psi;u;v;r];
    l_states = length(states);

    % CONTROL Symbology
    T1 = SX.sym('T1'); T2 = SX.sym('T2'); T3 = SX.sym('T3'); T4 = SX.sym('T4');
    controls = [T1;T2;T3;T4];
    l_controls = length(controls);

    % RHS - Right Hand Side
    rhs = [u*cos(psi)-v*sin(psi);
           u*sin(psi)+v*cos(psi);
           r;
           -0.094*u-0.84*abs(u)*u+2.078*v*r + 0.28*T1 + 0.30*T2 + 0.29*T3 + 0.29*T4;
           -0.061*v-0.85*abs(v)*v-1.01*u*r  + 0.10*T1 + 0.08*T2 - 0.10*T3 - 0.09*T4;
           -10.686*u*v-0.12*r-2.65*abs(r)*r + 1.02*T1 - 1.14*T2 - 0.98*T3 + 1.21*T4]; % system r.h.s
    f = Function('f',{states,controls},{rhs}); % nonlinear mapping function f(x,u)

% STEP 1.2: Objective Function
% ------------------------------

    objF = 0; % Objective function
    g = [];  % constraints vector

    Q = zeros(6,6);
    Q(1,1) = 10;   Q(2,2) = 10;   Q(3,3) = 1;     % weight matrix (position states)
    Q(4,4) = 0.1;  Q(5,5) = 0.1;  Q(6,6) = 0.1;   % weight matrix (velocity states)

    R = zeros(4,4);
    R(1,1) = 1; R(2,2) = 1; R(3,3) = 1; R(4,4) = 1; % weight matrix (controls)

    U = SX.sym('U',l_controls,N);           % Control States
    P = SX.sym('P',l_states + l_states);    % Parameters (initial state + reference state)
    X = SX.sym('X',l_states,(N+1));         % States at the optimization problem

    i_st  = X(:,1);       % initial state
    g = [g;i_st-P(1:6)];  % initial constraints
```

```
    for k = 1:N
        i_st = X(:,k);
        con = U(:,k);
        objF = objF+(i_st-P(7:12))'*Q*(i_st-P(7:12)) + con'*R*con;
        nexy_st = X(:,k+1);
        f_value = f(i_st,con);
        next_st_euler = i_st+ (T*f_value);
        % Constrains
        g = [g;nexy_st-next_st_euler];
    end

    % make the decision variable one column vector
    OPT_variables = [reshape(X,l_states*(N+1),1);reshape(U,l_controls*N,1)];

% Step 1.3: NLP problem with x and the function
% ---------------------------------------------
    nlp_prob = struct('f', objF, 'x', OPT_variables, 'g', g, 'p', P);

    % Optimization options
      opts = struct;
      opts.ipopt.max_iter = 5000;
      opts.ipopt.print_level =0; %0,3
      opts.print_time = 0;
      opts.ipopt.acceptable_tol =1e-8;
    % Optimal convergence tolerance:
      opts.ipopt.acceptable_obj_change_tol = 1e-6;
    % Create NLP solver:
      solver = nlpsol('solver','ipopt', nlp_prob,opts);

% Step 1.4: Constrains and initial parameters and values
% ------------------------------------------------------
    % Control Constrains
    T1_max = 1; T1_min = -T1_max;
    T2_max = 1; T2_min = -T2_max;
    T3_max = 1; T3_min = -T3_max;
    T4_max = 1; T4_min = -T4_max;

    % States Constrains
    x_max = 10; x_min = -x_max;
    y_max = 10; y_min = -y_max;
    psi_max = 0.5; psi_min = -psi_max;
    u_max = 0.5; u_min = -u_max;
    v_max = 0.5; v_min = -v_max;
    r_max = 0.5; r_min = -r_max;

    % Equality Constrains
    args = struct;
    args.lbg(1:6*(N+1)) = 0;
    args.ubg(1:6*(N+1)) = 0;

    args.lbx(1:6:6*(N+1),1) = x_min; %state x lower bound
    args.ubx(1:6:6*(N+1),1) = x_max; %state x upper bound
    args.lbx(2:6:6*(N+1),1) = y_min; %state y lower bound
    args.ubx(2:6:6*(N+1),1) = y_max; %state y upper bound
    args.lbx(3:6:6*(N+1),1) = psi_min; %state psi lower bound
    args.ubx(3:6:6*(N+1),1) = psi_max; %state psi upper bound
    args.lbx(4:6:6*(N+1),1) = u_min;  %state u lower bound
    args.ubx(4:6:6*(N+1),1) = u_max; %state u upper bound
    args.lbx(5:6:6*(N+1),1) = v_min; %state v lower bound
    args.ubx(5:6:6*(N+1),1) = v_max; %state v upper bound
    args.lbx(6:6:6*(N+1),1) = r_min; %state r lower bound
    args.ubx(6:6:6*(N+1),1) = r_max; %state r upper bound

    args.lbx(6*(N+1)+1:4:6*(N+1)+4*N,1) = T1_min; %u lower bound
    args.ubx(6*(N+1)+1:4:6*(N+1)+4*N,1) = T1_max; %u upper bound
    args.lbx(6*(N+1)+2:4:6*(N+1)+4*N,1) = T2_min; %v lower bound
    args.ubx(6*(N+1)+2:4:6*(N+1)+4*N,1) = T2_max; %v upper bound
    args.lbx(6*(N+1)+3:4:6*(N+1)+4*N,1) = T3_min; %r lower bound
    args.ubx(6*(N+1)+3:4:6*(N+1)+4*N,1) = T3_max; %r upper bound
    args.lbx(6*(N+1)+4:4:6*(N+1)+4*N,1) = T4_min; %r lower bound
    args.ubx(6*(N+1)+4:4:6*(N+1)+4*N,1) = T4_max; %r upper bound
```

STEP 2.0: NLMPC DATABASE

```
%-----------------------------------------
run = 1;            % # of simulations
Data = [];          % NLMPC Database
Data_ext = [];
x0h = [];

for n = 1:run
    t0 = 0;
    u0 = zeros(N,l_controls);            % N,control inputs

    % Reference and initial states
    bx = [10,10,0,0,0,0]';               % state limits should be the same that restrictions
    x0 = getRandomInputAUVRobot(bx, []); % initial random states inside limits
    x0h = [x0h,x0];
    bs = [1,1,0,0,0,0]';                 % state reference inside limits
    xs = getRandomInputAUVRobot(bs, []); % reference random state inside limits

    t(1) = t0;                           % initial time
    xx(:,1) = x0;                        % xx contains the history of states
    sim_tim = 40;                        % maximum simulation time
    X0 = repmat(x0,1,N+1)';              % initialization of the states decision variables

    % Start NLMPC
    mpciter = 0;                         % iteraciones
    xx1 = [];                            % xx1 contains the estimated states
    u_cl= [];                            % u_cl contains control actions u, v & w

    main_loop = tic;
    while(norm((x0-xs),2) > 1e-3 && mpciter < sim_tim / T)
        args.p   = [x0;xs]; % set the values of the parameters vector
        % initial value of the optimization variables
        args.x0  = [reshape(X0',l_states*(N+1),1);reshape(u0',l_controls*N,1)];

        sol = solver('x0', args.x0, 'lbx', args.lbx, 'ubx', args.ubx,...
            'lbg', args.lbg, 'ubg', args.ubg,'p',args.p);

        u = reshape(full(sol.x(l_states*(N+1)+1:end))',l_controls,N)'; % get controls only from the solution
        xx1(:,1:6,mpciter+1)= reshape(full(sol.x(1:6*(N+1)))',6,N+1)'; % get solution TRAJECTORY
        u_cl= [u_cl ; u(1,:)];
        t(mpciter+1) = t0;
        % Apply the control and shift the solution
        [t0, x0, u0] = shift(T, t0, x0, u,f);
        xx(:,mpciter+2) = x0;
        X0 = reshape(full(sol.x(1:6*(N+1)))',6,N+1)'; % get solution TRAJECTORY
        % Shift trajectory to initialize the next step
        X0 = [X0(2:end,:);X0(end,:)];
        mpciter;
        mpciter = mpciter + 1;
    end
    main_loop_time = toc(main_loop);
    ss_error = norm((x0-xs),2)

    % Taking data for DNN
    states= xx';
    data_ref= repmat(xs',length(u_cl)+1,1);
    data_nn = [states(1:mpciter+1,:) [data_ref] [u_cl;[0 0 0 0]]];
    Data = [Data; data_nn];
    n = n+1
end

save('CloningInputDataAUV','Data','x0h');
```

STEP 3.0: PLOT INPUT DATA

```
%-----------------------------------------

figure('Position',[1025 173 900 600])
subplot(2,1,1),
plot(Data(:,1),'o'), hold on,
plot(Data(:,2),'*'), hold on,
plot(Data(:,3),'d'), hold on,
plot(Data(:,4),'o'), hold on,
plot(Data(:,5),'*'), hold on,
plot(Data(:,6),'d'), hold on,
```

```
plot(Data(:,7),'o'), hold on,
plot(Data(:,8),'*'), hold on,
plot(Data(:,9),'d'), hold on,
plot(Data(:,10),'o'), hold on,
plot(Data(:,11),'d'), hold on,
plot(Data(:,12),'o'), hold on,
title('Input Data'),xlabel('Data Collection'),ylabel('Upper / Lower Limits')
legend({'$x$','$y$','$\psi$','u','v','r','$x_{ref}$','$y_{ref}$','$\psi_{ref}$','$u_{ref}$','$v_{ref}$','$r_{ref}$'},...
        'Interpreter','latex','Location','northeast','NumColumns',2)
set(gca, 'FontSize', 18)
grid on

% Plot the output data
subplot(2,1,2),
plot(Data(:,13),'o'), hold on,
plot(Data(:,14),'*'), hold on,
plot(Data(:,15),'d'), hold on,
plot(Data(:,16),'d'), hold on,
title('Output Data'), xlabel('Data Collection'), ylabel('Upper / Lower Limits')
legend({'$T1$','$T2$','$T3$','$T4$'},'Interpreter','latex',...
        'Location','northeast','NumColumns',1)
set(gca, 'FontSize', 18)
grid on

% Plot the output data
figure()
plot(Data(:,1),Data(:,2),'o','Color','#77AC30','LineWidth',0.75,'LineStyle','-'), hold on
%plot(Data(:,7),Data(:,8),'*','Color','r','LineWidth',5,'LineStyle','-'), hold on
title('XY Plot'), xlabel('X Position'), ylabel('Y Position')
legend({'$XY$ States'},'Interpreter','latex',...
        'Location','northeast','NumColumns',1)
set(gca, 'FontSize', 14)
grid on
```

STEP 4.0: PLOT TRACKING DATA

```
%-----------------------------------------

% MPC_tracking (t,xx,xx1,u_cl,xs,N,rob_diam)
```

Published with MATLAB® R2024a

Anexo 2: Script `DNNevaluation.m`: Evaluación de Control con Deep Learning y Control No Lineal MPC

Contents

- STEP 1.0: DNN Configuration
- STEP 2.0: Training the DNN
- STEP 3.0: NLMPC vs DNN Evaluation
- STEP 4.0: Plotting the results


```
% ************************************************************************
% Universidad: Pontificia Universidad Católica del Perú
% Tesis:       Control de Aprendizaje Profundo basado en un Control Predictivo por Modelo No Lineal
% Curso:       Maestría en Ingeniería Mecatrónica
% Alumno:      Manuel Enrique Gallardo Rodriguez
% ************************************************************************
```

STEP 1.0: DNN Configuration

```
clear all, close all,  clc
umax = 1;               % Scale the data
mpcverbosity off;       % Disable MPC Toolbox messages

fileName = 'CloningInputDataAUV.mat';
AUVData = load(fileName);
data = AUVData.Data;
x0h = AUVData.x0h;
existingData = data;
numCol = size(data,2);

numObservations = numCol-4; % Input Data
numActions = 4;             % Output Data
hiddenLayerSize = 256;      % Hidden Layers

imitateMPCNetwork = [
    featureInputLayer(numObservations,'Normalization','none','Name','observation')
    fullyConnectedLayer(hiddenLayerSize,'Name','fc1')
    reluLayer('Name','relu1')
    fullyConnectedLayer(hiddenLayerSize,'Name','fc2')
    reluLayer('Name','relu2')
    fullyConnectedLayer(hiddenLayerSize,'Name','fc3')
    reluLayer('Name','relu3')
    fullyConnectedLayer(hiddenLayerSize,'Name','fc4')
    reluLayer('Name','relu4')
    fullyConnectedLayer(hiddenLayerSize,'Name','fc5')
    reluLayer('Name','relu5')
    fullyConnectedLayer(hiddenLayerSize,'Name','fc6')
    reluLayer('Name','relu6')
    fullyConnectedLayer(numActions,'Name','fcLast')
    regressionLayer('Name','routput')];

% Plot the DNN
figure()
plot(layerGraph(imitateMPCNetwork))

% Intialize validation cell array
validationCellArray = {0,0};

% Training options
```

```
options = trainingOptions('adam', ...
                          'Verbose', false, ...
                          'Plots', 'training-progress', ...
                          'Shuffle', 'every-epoch', ...
                          'MiniBatchSize', 8192, ...
                          'ValidationData', validationCellArray, ...
                          'InitialLearnRate', 1e-3, ...
                          'ExecutionEnvironment', 'cpu', ...
                          'GradientThreshold', 10, ...
                          'MaxEpochs', 500 ...
                          );
```

STEP 2.0: Training the DNN

```
doTraining = true; % change 'true' if you perform the training

if doTraining
    behaviorCloningNNObj = behaviorCloningTrainNetwork(imitateMPCNetwork, options);
    DNN = behaviorCloningNNObj.imitateMPCNetObj;
else
    load('behaviorCloningMPCImDNNObject.mat');
    DNN = behaviorCloningNNObj.imitateMPCNetObj;
end
```

STEP 3.0: NLMPC vs DNN Evaluation

```
x0 = [-8 8 -0.1 0 0 0]';
u0 = [0 0 0 0]';
x_r = [0.1,0.15,0,0,0,0];
x_0 = x0';

% Duration
Tf = 15;
% Sample time
nlobj.Ts = 0.1;
Ts = nlobj.Ts;
% Simulation steps
Tsteps = Tf/Ts+1;

tic
[tHistoryDNN,xHistoryDNN,uHistoryDNN] = simModelDNNImAUVRobot(x0,x_r,DNN,Ts,Tf);
toc

tic
[tHistoryNLMPC,xHistoryNLMPC,uHistoryNLMPC] = simModelNLMPCImAUVRobot(x_0,x_r);
toc
```

STEP 4.0: Plotting the results

```
figure()
plot(xHistoryDNN(:,1),xHistoryDNN(:,2),'r*','linewidth',1.5)
hold on
plot(xHistoryNLMPC(:,1),xHistoryNLMPC(:,2),'bo','linewidth',1.5)
```

```
axis([-5 5 -5 5])
legend('DNN','NLMPC')
xlabel('x (m)')
ylabel('y (m)')
grid on

%
%States
largo = size(tHistoryNLMPC,2);
figure()
subplot(321)
plot(tHistoryDNN,xHistoryDNN(:,1),'Color','b','LineWidth',1.75,'LineStyle','-')
hold on
plot(tHistoryNLMPC',xHistoryNLMPC(1:largo,1),'Color','g','LineWidth',1.75,'LineStyle','-')
xlabel('$t$','Interpreter','latex')
ylabel('$m$','Interpreter','latex')
legend('DNN','NLMPC')
title({'$x$'},'Interpreter','latex','FontSize',18,'FontWeight','bold')
grid on

subplot(323)
plot(tHistoryDNN,xHistoryDNN(:,2),'Color','b','LineWidth',1.75,'LineStyle','-')
hold on
plot(tHistoryNLMPC',xHistoryNLMPC(1:largo,2),'Color','g','LineWidth',1.75,'LineStyle','-')
xlabel('$t$','Interpreter','latex')
ylabel('$m$','Interpreter','latex')
legend('DNN','NLMPC')
title({'$y$'},'Interpreter','latex','FontSize',18,'FontWeight','bold')
grid on

subplot(322)
plot(tHistoryDNN,xHistoryDNN(:,4),'Color','b','LineWidth',1.75,'LineStyle','-')
hold on
plot(tHistoryNLMPC',xHistoryNLMPC(1:largo,4),'Color','g','LineWidth',1.75,'LineStyle','-')
xlabel('$t$','Interpreter','latex')
ylabel('$m/s$','Interpreter','latex')
legend('DNN','NLMPC')
title({'$u$'},'Interpreter','latex','FontSize',18,'FontWeight','bold')
grid on

subplot(324)
plot(tHistoryDNN,xHistoryDNN(:,5),'Color','b','LineWidth',1.75,'LineStyle','-')
hold on
plot(tHistoryNLMPC',xHistoryNLMPC(1:largo,5),'Color','g','LineWidth',1.75,'LineStyle','-')
xlabel('$t$','Interpreter','latex')
ylabel('$m/s$','Interpreter','latex')
legend('DNN','NLMPC')
title({'$v$'},'Interpreter','latex','FontSize',18,'FontWeight','bold')
grid on

subplot(325)
plot(tHistoryDNN,xHistoryDNN(:,3),'Color','b','LineWidth',1.75,'LineStyle','-')
hold on
plot(tHistoryNLMPC',xHistoryNLMPC(1:largo,3),'Color','g','LineWidth',1.75,'LineStyle','-')
xlabel('$t$','Interpreter','latex')
ylabel('$rad$','Interpreter','latex')
legend('DNN','NLMPC')
title({'$\psi$'},'Interpreter','latex','FontSize',18,'FontWeight','bold')
grid on

subplot(326)
plot(tHistoryDNN,xHistoryDNN(:,6),'Color','b','LineWidth',1.75,'LineStyle','-')
hold on
```

```
plot(tHistoryNLMPC',xHistoryNLMPC(1:largo,6),'Color','g','LineWidth',1.75,'LineStyle','-')
xlabel('$t$','Interpreter','latex')
ylabel('$rad/s$','Interpreter','latex')
legend('DNN','NLMPC')
title({'$r$'},'Interpreter','latex','FontSize',18,'FontWeight','bold')
grid on

%
largoT=size(uHistoryDNN,1);
figure()
subplot(221)
plot(tHistoryDNN(1:largoT),uHistoryDNN(:,1),'Color','b','LineWidth',1.75,'LineStyle','-')
hold on
plot(tHistoryNLMPC',uHistoryNLMPC(1:largo,1),'Color','g','LineWidth',1.75,'LineStyle','-')
xlabel('$t$','Interpreter','latex')
ylabel('$N$','Interpreter','latex')
legend('DNN','NLMPC')
title({'$T1$'},'Interpreter','latex','FontSize',18,'FontWeight','bold')
grid on

subplot(222)
plot(tHistoryDNN(1:largoT),uHistoryDNN(:,2),'Color','b','LineWidth',1.75,'LineStyle','-')
hold on
plot(tHistoryNLMPC',uHistoryNLMPC(1:largo,2),'Color','g','LineWidth',1.75,'LineStyle','-')
xlabel('$t$','Interpreter','latex')
ylabel('$N$','Interpreter','latex')
legend('DNN','NLMPC')
title({'$T2$'},'Interpreter','latex','FontSize',18,'FontWeight','bold')
grid on

subplot(223)
plot(tHistoryDNN(1:largoT),uHistoryDNN(:,3),'Color','b','LineWidth',1.75,'LineStyle','-')
hold on
plot(tHistoryNLMPC',uHistoryNLMPC(1:largo,3),'Color','g','LineWidth',1.75,'LineStyle','-')
xlabel('$t$','Interpreter','latex')
ylabel('$N$','Interpreter','latex')
legend('DNN','NLMPC')
title({'$T3$'},'Interpreter','latex','FontSize',18,'FontWeight','bold')
grid on

subplot(224)
plot(tHistoryDNN(1:largoT),uHistoryDNN(:,4),'Color','b','LineWidth',1.75,'LineStyle','-')
hold on
plot(tHistoryNLMPC',uHistoryNLMPC(1:largo,4),'Color','g','LineWidth',1.75,'LineStyle','-')
xlabel('$t$','Interpreter','latex')
ylabel('$N$','Interpreter','latex')
legend('DNN','NLMPC')
title({'$T4$'},'Interpreter','latex','FontSize',18,'FontWeight','bold')
grid on
```

Published with MATLAB® R2024a