UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

ESCUELA DE INGENIERÍA EN MECATRÓNICA

TRABAJO DE GRADO PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN MECATRÓNICA

TEMA:

"MODELO DINÁMICO DEL ROBOT XPILOT-AI"

"XPILOT-AI BOT'S DYNAMIC MODEL"

AUTOR: SAIRY PAÚL DE LA TORRE MORALES

DIRECTOR: CARLOS XAVIER ROSERO

IBARRA-ECUADOR
2021

UNIVERSIDAD TÉCNICA DEL NORTE
BIBLIOTECA UNIVERSITARIA

AUTORIZACIÓN DE USO Y PUBLICACIÓN
A FAVOR DE LA UNIVERSIDAD TÉCNICA DEL NORTE

1. IDENTIFICACIÓN DE LA OBRA

En cumplimiento del Art. 144 de la Ley de Educación Superior, hago la entrega del presente trabajo a la Universidad Técnica del Norte para que sea publicado en el Repositorio Digital Institucional, para lo cual pongo a disposición la siguiente información:

DATOS DE CONTACTO			
CÉDULA DE IDENTIDAD:	**100395922-6**		
APELLIDOS Y NOMBRES:	**DE LA TORRE MORALES SAIRY PAÚL**		
DIRECCIÓN:	**SAN ROQUE, ATUNTAQUI**		
EMAIL:	**spdelatorrem@utn.edu.ec – sairys.dlt@gmail.com**		
TELÉFONO FIJO:		**TELÉFONO MÓVIL:**	**0988400479**

DATOS DE LA OBRA	
TÍTULO:	**“MODELO DINÁMICO DEL ROBOT XPILOT-AI”**
AUTOR (ES):	**SAIRY PAÚL DE LA TORRE MORALES**
FECHA: DD/MM/AAAA	**06/10/2021**
SOLO PARA TRABAJOS DE GRADO	
PROGRAMA:	**PREGRADO**
TITULO POR EL QUE OPTA:	**INGENIERO EN MECATRÓNICA**
ASESOR /DIRECTOR:	**CARLOS XAVIER ROSERO**

2. CONSTANCIAS

El autor manifiesta que la obra objeto de la presente autorización es original y se la desarrolló, sin violar derechos de autor de terceros, por lo tanto la obra es original y que es el titular de los derechos patrimoniales, por lo que asume la responsabilidad sobre el contenido de la misma y saldrá en defensa de la Universidad en caso de reclamación por parte de terceros.

Ibarra, a los 6 días del mes de Octubre de 2021

Sairy Paúl De La Torre Morales
C.I.: 100395922-6

UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

CERTIFICACIÓN

En calidad de director del trabajo de grado “MODELO DINÁMICO DEL ROBOT XPILOT-AI”, presentado por el egresado SAIRY PAÚL DE LA TORRE MORALES, para optar por el título de Ingeniero en Mecatrónica, certifico que el mencionado proyecto fue realizado bajo mi dirección.

Ibarra, 6 de Octubre de 2021

Firmado electrónicamente por:
CARLOS XAVIER ROSERO CHANDI

Carlos Xavier Rosero
DIRECTOR DE TESIS

Agradecimiento

En primer lugar a mis padres, Carlos Gustavo y Claudia Esthela, que siempre supieron darme el apoyo y la motivación necesaria a lo largo de todos estos años. Gracias por confiar en mí.

A mis hermanos, Stalin y Daky, que han estado a mi lado y ven en mí un ejemplo a seguir como hermano mayor.

A Ampi, gracias por ser parte de mi formación profesional a lo largo de todos estos años. Sin tí nada habría tenido sentido.

A Carlos Xavier Rosero por su conocimiento, paciencia y tiempo. Además, por su confianza depositada en mí desde el primer momento, permitiéndome realizar el presente trabajo.

A mis compañeros, especialmente a Anita y Daniel, gracias por todos los buenos y malos momentos vividos a lo largo de esta etapa.

Sairy Paúl De La Torre Morales

Dedicatoria

Especial dedicatoria a mis padres, Claudia Esthela y Carlos Gustavo, quienes han tenido mucha paciencia y me han brindado mucho cariño durante todo este tiempo en busca de mi título profesional.

A mis hermanos, Stalin y Daky, especialmente a Daky que es mi inspiración para seguir adelante, aunque no sea el mejor hermano mayor que podrían desear.

A Ampi, por todos estos años apoyándome día y noche incondicionalmente. Te mereces el cielo y las estrellas.

Ñuka María Dolores Mamitaman, ñuka Juan José Papitoman, ashtakata yupaychani kikinkunapa yachachishkakunamanta, kay kushikuyka kikinkunapashmi kapan.

Ñuka María Lucila Mamitaman, kikinpa kuyayta kuwashkamanta.

Sairy Paúl De La Torre Morales

Resumen

El control de robots es un gran campo de aplicación de la ingeniería. Estos pueden ser encontrados tanto en sistemas físicos como informáticos. Sin embargo, existen diferentes plataformas que no necesariamente son de código abierto, por lo que acceder a una licencia puede ser muy significativo.

Xpilot presenta una librería de código abierto que trabaja con diferentes lenguajes de programación de código abierto y una manera fácil de aprendizaje de diferentes tipos de habilidades para los estudiantes, especialmente para aquellos que desean aprender a programar diferentes estrategias de control. Por otra parte, Xpilot AI, permite programar bots de manera que los mismos puedan realizar diferentes tipos de acciones en el juego, en muchos de los casos se hace utilidad de IA (Inteligencia Artificial). Además, Xpilot-AI ofrece una amplia gama de bots para lograr comprender como funcionan los diferentes comandos.

El problema de moverse desde un punto A hasta un punto B, es el que se presenta en el actual trabajo. Para la resolución de este problema se hace utilidad de diferentes herramientas informáticas y matemáticas, así como también la información de muchas fuentes bibliográficas. Para que el performance del bot sea el óptimo, se considera una máquina de estados que contiene todos las condiciones para que el mismo se desempeñe en su entorno informático y pueda lograr con su cometido. Estas condiciones hacen que el robot actúe de diferente manera frente a las situaciones que existen dentro del juego.

Además, se utiliza la plataforma Raspberry Pi 3 modelo B, para montar un servidor al que el bot es capaz de conectarse de manera remota. Dicho servidor también es distribuido por XPilot y está disponible para su libre descarga desde su página web. Este servidor puede utilizarse en un mismo terminal de requerirse el caso.

Sairy Paúl De La Torre Morales

Abstract

Robot control is a huge branach of applied engineering. Robots can be found in both physical and computer systems. However, there are different platforms that are not necessarily open source, so accessing a license can be very significant.

Xpilot presents an open source library that works with different open source programming languages. Also, it presents an easy way of learning different types of skills for students, especially for those who want to learn to program different control strategies. On the other hand, Xpilot AI allows bots to be programmed so that they can perform different types of actions in the game, in many cases AI (Artificial Intelligence) is used. In addition, Xpilot-AI offers a wide range of bots to help you understand how the different commands work.

The problem of moving from point A to point B is the one presented in the current work. Different computer and mathematical tools are used to solve this problem, as well as information from many bibliographic sources. For the bot's performance to be optimal, it is considered a state machine that contains all the conditions for it, so that it works in the best way achieving its mission in its computing environment. These conditions make the robot act differently in the face of situations that exist within the game.

In addition, the Raspberry Pi 3 model B+ platform is used to mount a server to which the bot is able to connect remotely. This server is also distributed by XPilot and is available for free download from its website. This server can be used in the same terminal if the case is required.

Sairy Paúl De La Torre Morales

Índice general

Índice de figuras

Listings

Capítulo 1

Introducción

Este trabajo de grado ha sido realizado con el *Grupo de Investigación en Sistemas Inteligentes de la Universidad Técnica del Norte (GISI-UTN).*

1.1. Problema

X Pilot es un simulador de combate espacial bidimensional de código abierto en el que naves robots interactúan. Varios jugadores pueden conectarse a un servidor central de X Pilot y competir en muchas variedades de juegos, como el combate libre, la captura de la bandera o el combate en equipo (1).

El problema de la generación de un camino a seguir por un robot móvil para llegar de una posición inicial a otra final evitando los obstáculos del entorno ha sido tratado ampliamente en la literatura (2) . Por otro lado, ya se han desarrollado modelos dinámicos que describen la trayectoria de un robot en 2D, sin embargo, dichos modelos han sido utilizados para controlar trayectorias de robots en un entorno físico y no en uno computacional (3) .

En lo que respecta a la literatura de X Pilot AI, no existe suficiente información acerca de un modelo que describa el comportamiento de la nave en su entorno. Por lo tanto, es necesario modelar su dinámica para diseñar una estrategia que permita controlar su posición y orientación.

1.2. Alcance

El proyecto a realizarse consiste en establecer un modelo dinámico que permita describir el movimiento de la nave robot X Pilot AI en su entorno computacional, considerando las variables de estado importantes (aceleración, velocidad, distancia). Adicionalmente, se diseñará e implementará un algoritmo de control sobre una plataforma de software libre para su posterior utilización en la academia como plataforma para ensayo de técnicas de control.

1.3. Objetivos

1.3.1. Objetivo General

Determinar el modelo dinámico del robot X Pilot AI.

1.3.2. Objetivos Específicos

- Realizar el modelo dinámico considerando las características del robot implicadas en el movimiento y su entorno informático.
- Diseñar la acción de control para que el robot se desempeñe en su entorno.
- Implementar el algoritmo de control en un servidor con software libre.

1.4. Justificación

La relevancia del presente proyecto consiste en una investigación que permita utilizar la librería de X Pilot AI. La habilidad de modificar el código fuente permite a los usuarios e investigadores hacer grandes cambios en lo que respecta al entorno, que de otra manera no sería posible (4) .

La simulación se puede definir como la imitación de un sistema usando un modelo de computadora con el fin de evaluar y mejorar el desempeño de este (5) . Los sistemas dinámicos que se encuentran comúnmente como componentes de sistemas industriales tienen un comportamiento que debe representarse a través de modelos para obtener información sobre su funcionamiento (6) .

En X Pilot, las fuerzas físicas, como la inercia, la fricción y la gravedad, se modelan de manera realista. Las naves tienen masa, y se requiere habilidad para la aplicación de la dirección correcta y la cantidad de fuerza para navegar a través del entorno (4) .

Es necesario establecer un modelo que describa la trayectoria de la nave de un punto hacia otro, con el fin de utilizarlo como una herramienta de simulación para su utilización en un servidor de X Pilot. Existen modelos que permiten a robots físicos desplazarse en un entorno, sin embargo, dichos modelos no han sido implementados en la nave de X Pilot AI.

1.5. Contexto

El punto álgido del proyecto es encontrar las ecuaciones diferenciales que modelen el comportamiento de la nave. Sin embargo, existen algunos trabajos donde se sustentan diferentes soluciones para el control de robots en entornos 2D. Se puede utilizar dicha información para obtener un modelo que describa el comportamiento de la nave en su entorno.

(7) Presenta X Pilot como un entorno de aprendizaje que se puede usar para desarrollar comportamientos reactivos primitivos, pero puede ser lo suficientemente complejo como para requerir estrategias de combate y la cooperación del equipo. Además, utiliza el entorno con un algoritmo genético para conocer los pesos de un controlador de red neuronal artificial que proporciona control reactivo tanto ofensivo como defensivo para un agente autónomo.

En (8) se desarrolla un sistema de control para seguir una trayectoria determinada que es aplicado en un robot diferencial. Se utiliza la cinemática directa del robot para simular su comportamiento. Por otro lado, en (9) se considera el modelo dinámico de un dirigible y un análisis para el planteamiento de modelos de control. Para el suavizado de las trayectorias del dirigible se proponen dos modelos con relajación en la dinámica de la variable de control.

Gallardo D., Colomina o., Flórez F. y Rizo R., en (10) plantean una solución a la planificación de trayectorias subóptimas y robustas en el espacio de velocidades de un robot móvil. Se planifica una secuencia de velocidades, realizando entonces la búsqueda no en el espacio de configuraciones sino en el espacio de velocidades.

Capítulo 2

Revisión Literaria

En el presente capítulo se da una breve explicación de qué es XPilot y el funcionamiento del mismo. Además, en la sección 2.3, se realiza un estudio del estado del arte de los métodos de control existentes para bots de X-Pilot AI así como para el manejo de su entorno.

2.1. X-Pilot

Xpilot es un juego de combate espacial bidimensional, multijugador y de código abierto. Los jugadores navegan por naves triangulares a través de mapas personalizados, recolectando potenciadores, evitando trampas e intentando derribar a sus oponentes para ser la última nave viva (5).

Xpilot consta principalmente de dos componentes: el servidor y el cliente (Fig. 2.1). El servidor se utiliza para configurar los ajustes de un juego. Por ejemplo, puede cambiar el número de fotogramas por segundo (FPS) en un juego y el mapa que se utiliza. También es responsabilidad del servidor realizar un seguimiento de los jugadores que juegan el juego, sus puntuaciones y otra información. Toda la comunicación entre clientes se enruta a través del servidor. En lugar de que cada cliente se envíe información directamente entre sí, envían y reciben información desde y hacia el servidor. La naturaleza centralizada del juego ayuda a mejorar la confiabilidad y la eficiencia durante el juego que involucra a múltiples usuarios (4).

2.2. X-Pilot AI

Xpilot-AI es un entorno en el que los investigadores pueden probar programas de aprendizaje y control de agentes autónomos. Está basado en Xpilot, que es un juego de combate espacial 2d de código abierto. Xpilot-AI permite a un programador escribir scripts que controlan a un agente de juego de Xpilot (6).

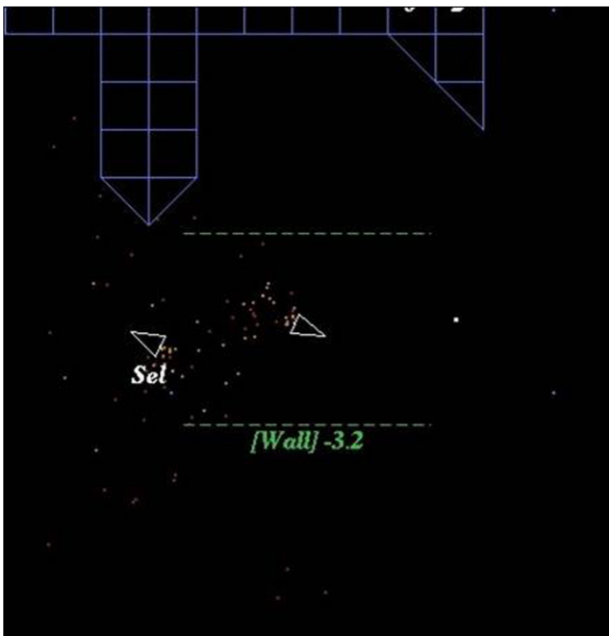

Figura 2.1: Entorno de juego de XPilot

2.2.1. Controles de Juego

Las funciones exportadas por la biblioteca Xpilot-AI realizan acciones simulando pulsaciones de teclas en el cliente Xpilot. La información sobre el mapa y los objetos circundantes se obtiene de las funciones de dibujo del cliente. También existen funciones para enviar y recibir mensajes a través del sistema de chat, lo que permite que los agentes independientes del juego se comuniquen entre sí y con el servidor Xpilot. Xpilot-AI también proporciona la función AI_main, una función vacía que se llama para cada marco de ejecución. Los programas de agentes se escriben redefiniendo esta función.
Además, Xpilot-AI proporciona varias funciones a través de las que los programas de control de agentes pueden obtener información sobre el entorno que los rodea, los barcos en el mapa y varios otros datos útiles. Existen enlaces externos para Java, Scheme y Python que permiten escribir programas de control de agentes en cualquiera de estos tres lenguajes (4).

2.3. Modelos X-Pilot AI

2.3.1. Algoritmo Genético de Cola

La estructura de la QGA (Algoritmo Genético de Cola por sus siglas en Inglés) es una cola en la que se forman nuevos individuos a partir de cruces de individuos seleccionados estocásticamente (rueda de ruleta) de la población, mientras que se eliminan los individuos de mayor edad. La cola es del tamaño de la población deseada y está compuesta en su totalidad por individuos que ya han sido evaluados para determinar su aptitud. Cuando un cliente está disponible, dos individuos se eligen estocásticamente de acuerdo con la aptitud de la población, sus cromosomas se cruzan para formar el nuevo hijo. El niño es enviado al cliente, donde se prueba su condición física y se coloca al comienzo de la cola. Se elimina al individuo de mayor edad, y cada individuo se mueve hacia abajo un espacio más cerca del final donde eventualmente será eliminado. Si el tamaño de la población es p, cada individuo tiene p posibilidades de ser elegido como padre de un nuevo hijo. Debido a que cada individuo tiene p posibilidades de reproducirse, el QGA es muy similar en comportamiento evolutivo a un RGA, que realiza p recombinaciones para formar una nueva población después de evaluar a cada individuo en la generación (11).

2.3.2. Estrategias Evolutivas

Las estrategias de evolución implican el uso de una población inicial, una mutación y una función de aptitud. En el caso general, las estrategias de evolución toman la población inicial, mutan a sus miembros para crear al menos la misma cantidad de niños y luego prueban la aptitud de los niños utilizando la función de aptitud. En las estrategias de evolución 1 a 1, la población inicial es exactamente un individuo, y en cada época se crea y prueba un niño, y solo un niño exitoso reemplaza al padre. Esta técnica, si bien es eficaz en términos de garantizar el progreso salvo tretas en la función de aptitud, no es necesariamente tan rápida como otros tipos de estrategias de evolución cuando la aptitud se puede alcanzar rápidamente. Sin embargo, dado que el objetivo del aprendizaje en tiempo real es tener siempre la mejor versión en uso, utilizando una población, que no siempre mostrará al individuo con la mayor aptitud, cuando se puede evitar, parecía ser una complicación innecesaria. a este 1 a 1 se consideró la opción más adecuada (12).

2.3.3. Aprendizaje por Refuerzo

Muchas técnicas de RL (Aprendizaje por refuerzo por sus siglas en Inglés) emplean formas de abstracción de estados. En muchos de estos enfoques, se utiliza un conjunto reducido de variables de estado para generalizar sobre un gran número de estados, lo que reduce la dimensionalidad general del problema para permitir una mayor repetición y una convergencia más rápida. En parte, este enfoque es natural, ya que parece que el comportamiento inteligente a menudo implica ignorar las características irrelevantes del entorno y concentrarse en las cosas que importan. En la práctica, sin embargo, para hacer que la RL sea más práctica, las abstrac-

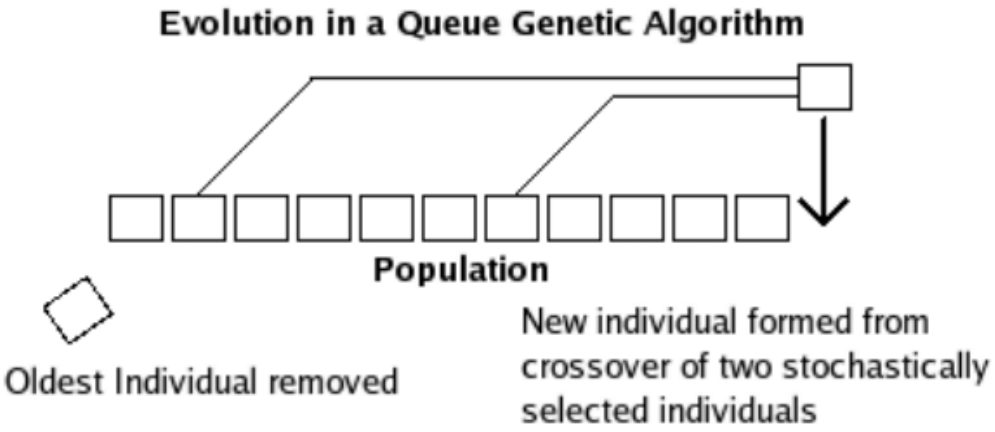

Figura 2.2: Algoritmo Genético de Cola

ciones a menudo ignoran por completo un gran número de variables de estado para reducir el tamaño del espacio de estados y arriesgarse a la simplificación (13).

2.3.4. Codificación Adaptativa de Kanerva

Este método emplea un pequeño subconjunto de los estados originales como proxy del entorno completo, actualizando los valores sobre los estados del prototipo representativos abstractos de una manera análoga a Q-learning. Con el tiempo, el conjunto de prototipos se ajusta para proporcionar una cobertura y una abstracción más efectivas, nuevamente de forma automática. Nuestros resultados muestran que esta técnica permite que un simple agente de aprendizaje duplique su tiempo de supervivencia cuando navega por el entorno de Xpilot, utilizando solo una pequeña fracción del espacio de estado completo como sustituto y aumentando en gran medida el potencial para un aprendizaje más rápido (13).

2.3.5. Controlador de Red Neuronal

El término red neuronal se refiere a un modelo computacional cuyo diseño está motivado por la estructura y los aspectos funcionales de las redes neuronales biológicas que se encuentran en un cerebro. Las redes neuronales se componen de neuronas artificiales que se han conectado entre sí para promover la comunicación con el fin de resolver un problema específico. Se utilizan para determinar patrones en grandes cantidades de datos, así como para encontrar formas de representar datos complejos. Hay varios tipos de redes neuronales, algunas considerablemente más complejas que otras. Las redes neuronales simples a menudo se representan con capas: la red tendrá n-nodos en la capa de entrada y m-nodos en la capa de salida. Los nodos en la capa de entrada aceptan datos y luego envían los datos a la capa de salida, realizando transformaciones en el camino para generar la salida deseada. Estas transformaciones ocurren multiplicando las

entradas por pesos específicos, asociados con cada neurona. Para representar datos más complejos, a menudo se utilizan capas adicionales. Estas capas adicionales se denominan “capas ocultas”, dado que son utilizadas por el sistema pero no interactúan directamente con el usuario, ya que no producen una salida visible propia y solo reciben información de otras neuronas del sistema (14).

2.4. Propuesta

En la literatura existen varios modelos de control para bots de XPilot, muchos de ellos utilizan IA para su propósito. Sin embargo, también existen bots que no requieren de de IA, si no que más bien, optan por utilizar técnicas más sencillas.
El algoritmo propuesto utiliza la segunda Ley de Newton, es decir, se propone resolver el problema aplicando las formulas del Movimiento Rectilíneo Uniformemente Variado (M.R.U.V.).
El script del bot será ensayado en un servidor remoto, montado en la plataforma Raspberry Pi modelo 3 B+.

Capítulo 3

Metodología

En este apartado se realiza una descripción de la metodología utilizada para llevar a cabo la programación del bot en su entorno informático.

Previo a la utilización de XPilot-AI y su entorno, se debe instalar todas las librerías para que su funcionamiento sea el óptimo. En www.xpilot-ai.org se pueden encontrar los archivos necesarios para su instalación. Sin embargo, en la sección 3.3 se puede encontrar una explicación detallada para la preparación del sistema, instalar XPilot-AI y todos los componentes necesarios para que el programa funcione correctamente.

3.1. Descripción del Algoritmo

En la figura 3.1 se muestra el diagrama de bloques que se ha utilizado para la programación del algoritmo que hace posible el funcionamiento del bot de la manera requerida. El código se presenta en los anexos.

En primera instancia, se procede a iniciar un servidor de XPilot y conectarse a él en la red. Una vez conectado al servidor de XPilot se procede al cálculo del angulo al que debe girar la nave. Posteriormente se realiza el cálculo de las coordenadas X e Y, que serán los puntos de destino al que la nave debe llegar aplicando las ecuaciones del modelado dinámico. Finalmente se impulsa la nave para que realice su recorrido hasta el punto X,Y anteriormente calculado.

Más adelante se explica más a detalle cada uno de los pasos que se pueden observar en el diagrama de flujo.

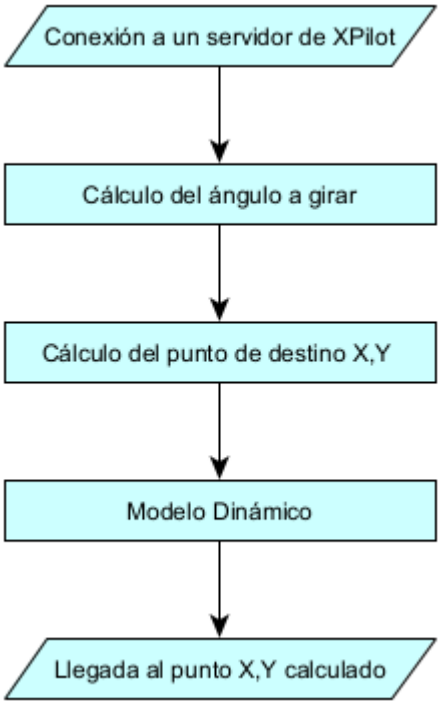

Figura 3.1: Diagrama de Flujo

3.2. Máquina de Estados

Se denomina máquina de estados a un modelo de comportamiento de un sistema con entradas y salidas, en donde las salidas dependen no sólo de las señales de entradas actuales sino también de las anteriores. Las máquinas de estados se definen como un conjunto de estados que sirve de intermediario en esta relación de entradas y salidas, haciendo que el historial de señales de entrada determine, para cada instante, un estado para la máquina, de forma tal que la salida depende únicamente del estado y las entradas actuales (15). En la Fig. 3.2 se puede observar el criterio con el que fue programado el algoritmo.

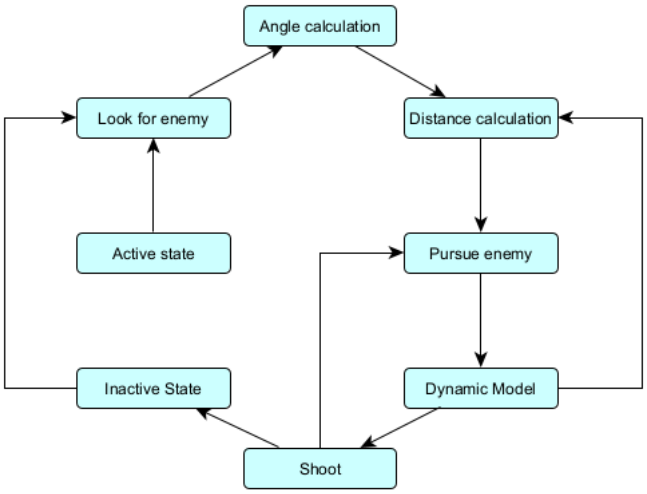

Figura 3.2: Máquina de estados

3.3. Requerimientos del Sistema

3.3.1. Sistema Operativo

Para poder llevar a cabo la realización de este proyecto es necesario contar con el sistema operativo (O.S. por sus siglas en inglés) Ubuntu 12.04 LTS. De otra manera, no será posible ensayar los bots ya que en los O.S. lanzados posteriormente únicamente se puede ejectuar XPilot y jugar pero sin la libertad de poder programar los mismos.

3.3.2. Python 3 y sus componentes

XPilot AI funciona con Python 3, por lo que en primera instancia es necesario instalar dicho lenguaje de programación. Así mismo, se necesita la librería NumPy, para poder utilizar determinadas operaciones matemáticas y poder ensayar las naves robot. Para instalar los 2 compo-

nentes principales, se debe abrir la terminal de Ubuntu 12.04 LTS y proceder con los siguientes comandos:

- Instalación de Python 3: sudo apt install python3.
- Instalación de NumPy: sudo apt install python-numpy.

3.3.3. Librerías Necesarias

Las librerías que se mencionan a continuación deben ser instaladas antes de instalar XPilot, ya que de lo contrario el mismo no se podrá ejecutar.

Actualmente existen 2 versiones de XPilot-AI que tienen soporte:

- Xpilot-AI 1.0 basado en XPilot Classic 4.5.5
- Xpilot-ng-AI 0.9 basado en XPilot NG 4.7.3

En este caso se utilizó Xpilot-ng-AI 0.9 ya que incluye un cliente SDL, lo que brinda un entrono gráfico mucho más desarrollado. Para compilar Xpilot-ng-AI 0.9 se necesitan las siguientes librerías:

- Expat XML versión 1.1
- Librería de compresión zlib versión 1.1.3
- Compilador ANSI C
- X11

Los requisitos adicionales para el cliente SDL son:

- SDL 1.2
- SDL_ttf 2.0
- SDL_image 1.2
- OpenGL (acelerado por hardware)

El proceso para la instalación de XPilot AI se describe en el siguiente apartado.

3.3.4. Instalación de XPilot AI

Una vez instaladas todas las librerías necesarias, se procede a instalar Xpilot-ng-AI 0.9. Para ello se puede utilizar los siguientes comandos desde la terminal de Ubuntu 12.04 LTS.

1. wget http://xpilot-ai.org/downloads/xpilot-ng-ai-0.9/xpilot-ng-ai-0.9.sh
2. chmod +x xpilot-ng-ai-0.9.sh
3. ./xpilot-ng-ai-0.9.sh

De esta manera XPilot AI queda instalado y listo para su utilización en ensayo de naves robot.

3.4. Configuración de Raspberry Pi 3

En el presente trabajo se hace utilidad de la plataforma Raspberry Pi, por lo que es necesaria su configuración para su posterior utilización como servidor de juego de Xpilot. En este caso se optó por utilizar una Raspberry Pi 3 donde se procede a montar el sistema operativo Raspbian. El O.S. pertinente para el manejo de Raspberry Pi 3 está disponible en `https://www.raspberrypi.org/software/`. Es necesario configurar el sistema para tener actualizados todos los repositorios. Para este cometido, se hace utilidad del comando **sudo apt-get update** seguido de **sudo apt-get upgrade**. En la Fig. 3.3 se muestra dicha configuración.

Figura 3.3: Configuración Raspberry Pi

3.5. Algoritmo Propuesto

3.5.1. Calculo del Ángulo de giro

El cálculo del ángulo a girar se realiza con el objetivo de mantener la nave espacial en dirección a un enemigo detectado. De esta manera se mantiene el error de la orientación en el mínimo posible. El comando utilizado para sensar la orientación de la nave es ai.selfHeadingDeg() en caso de requerirlo en grados, o ai.selfHeadingRad() en caso de requerirlo en radianes.

En este apartado se calcula el error en la orientación para posteriormente calcular las velocidades tanto en el eje X, como en el eje Y. Para evitar un resultado no deseado en la resta del ángulo actual y deseado, en función de su signo, se restringe el error del ángulo de rotación entre 0 a 180 grados y 0 a -180 grados. Se utiliza la función arctan2 (mt.atan2) de la librería **math** para un cálculo preciso y así evitar una orientación errónea. Esto se debe a que la función arctan2 es una función de cuatro cuadrantes, que produce el ángulo dentro de los cuatro cuadrantes del plano. Ayuda a determinar el signo y la ubicación del cuadrante derecho del ángulo resultante, dentro del rango de (]. Los ángulos en sentido antihorario se consideran positivos y los ángulos en sentido horario se consideran negativos. El código utilizado en Python se muestra a continuación. Ver Fig. 3.4.

```
theta = ai.selfHeadingDeg() * np.pi / 180

deltaX = xGoal - xNow
deltaY = yGoal - yNow
angg = mt.atan2(deltaY, deltaX)
```

Figura 3.4: Cálculo del ángulo theta

3.5.2. Cálculo del punto de destino X,Y

Para conocer el punto X, Y al que la nave espacial debe dirigirse debemos en primera instancia sensar la posición y orientación de la nave. Esto se puede lograr utilizando los comandos ai.selfX() y ai.selfY(), de esta manera se obtiene la lectura de la posición de la nave en los ejes X e Y respectivamente. La orientación es la misma que se sensó en la anterior sección. Sin embargo, para la navegación en combate, es necesario conocer la localización de las naves robot enemigas, esto se logra con la función look4enemy que se presenta en la Fig. 3.5donde el punto de destino se calcula cuando se detecta a un enemigo, esto se puede lograr gracias al comendo ai.closestShipId().. En este caso, el vector [x_goal,x_goal] será el punto de llegada y el vector [x_now,y_now] será el punto de partida. Se utiliza ai.selfVelX() y ai.selfVelY() para conocer las velocidades en X e Y. En este caso el vector velocidad **V** está compuesto por las velocidades Vx y Vy, su módulo se representa con la ecuación 3.1. El punto de destino se calcula cuando se detecta a un enemigo, esto se puede lograr gracias al comendo ai.closestShipId().

```
def look4Enemy():
    Id = ai.closestShipId()  # look for the closest ship

    if Id != -1:
        Dist = ai.enemyDistanceId(Id)
        Name = ai.enemyNameId(Id)
        X = ai.screenEnemyXId(Id)
        Y = ai.screenEnemyYId(Id)
        print("Id:", Id, Name, "D", Dist, "X", X, "Y", Y)
        return [Id, Name, Dist, X, Y]
    else:
        #print("Closest target not found!")
        return [-1, '', -1, -1, -1]

def wallDetection():
    wall_feeler1 = ai.wallFeeler(500, int(ai.selfHeadingDeg()), 0, 0)
    print("The wall is", wall_feeler1, "pixels away")

    return
```

Figura 3.5: Función para localización de enemigos en el mapa

Por tanto, la distancia a recorrer se calcula mediante el teorema de pitágoras con la ecuación 3.2.

$$V = \sqrt{v_x^2 + v_y^2} \tag{3.1}$$

$$Distancia = \sqrt{(x_{goal} - x_{now})^2 + (y_{goal} - y_{now})^2} \tag{3.2}$$

3.5.3. Modelo Dinámico

La animación emula una nave espacial que se mueve en un espacio (bidimensional) (16). Es decir, la nave se moviliza en el plano XY, por lo que, tanto como su posición y velocidad son calculadas en este plano. De esta manera su posición queda descrita por las ecuaciones **3.1** y **3.2**.

Eje X:

$$x_i = x_{i-1} + v_x \Delta t + \frac{1}{2} a x_i \Delta t^2 \tag{3.3}$$

Eje Y:

$$y_i = y_{i-1} + v_y \Delta t + \frac{1}{2} a y_i \Delta t^2 \tag{3.4}$$

A su vez, la velocidad queda descrita por las ecuaciones **3.3** y **3.4**

Velocidad en X:

$$vx_i = vx_{i-1} + a x_i \Delta t \tag{3.5}$$

Velocidad en Y:

$$vy_i = vy_{i-1} + a y_i \Delta t \tag{3.6}$$

Capítulo 4

Resultados

En este capítulo se muestran la prueba de la plataforma Xpilot, el servidor y los resultados de la simulación.

4.1. Implementación

4.1.1. Xpilot

Para inciar Xpilot, se abre la terminal de Ubuntu 12.04 LTS presionando las teclas **Ctrl+Alt+T**. En la terminal se digita el comando **xpilot-ng-server** seguido de **Enter** tal y como se muestra en la Fig. 4.1.

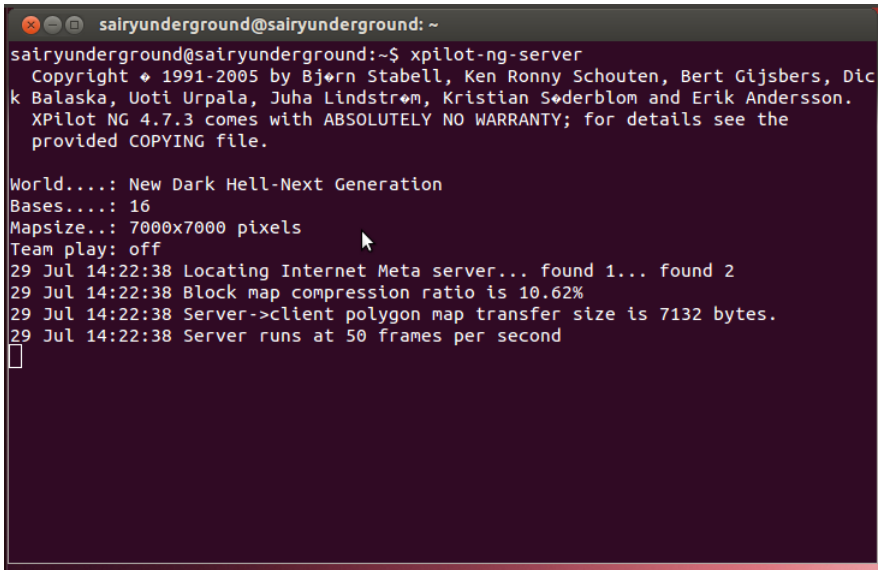

Figura 4.1: Prueba de Xpilot: Servidor

Una vez inicializado el servidor, presionar **Ctrl+Shift+T** para abrir una nueva pestaña de la terminal. Con el comando **xpilot-ng-x11 -join** se inicia el entorno de Xpilot (ver Fig. 4.2). También se puede utilizar el comando **xpilot-ng-sdl -join** en caso de querer utilizar el cliente SDL. Los entornos de Xpilot X11 y SDL se muestran en las Fig. 4.3 y Fig. 4.4 respectivamente.

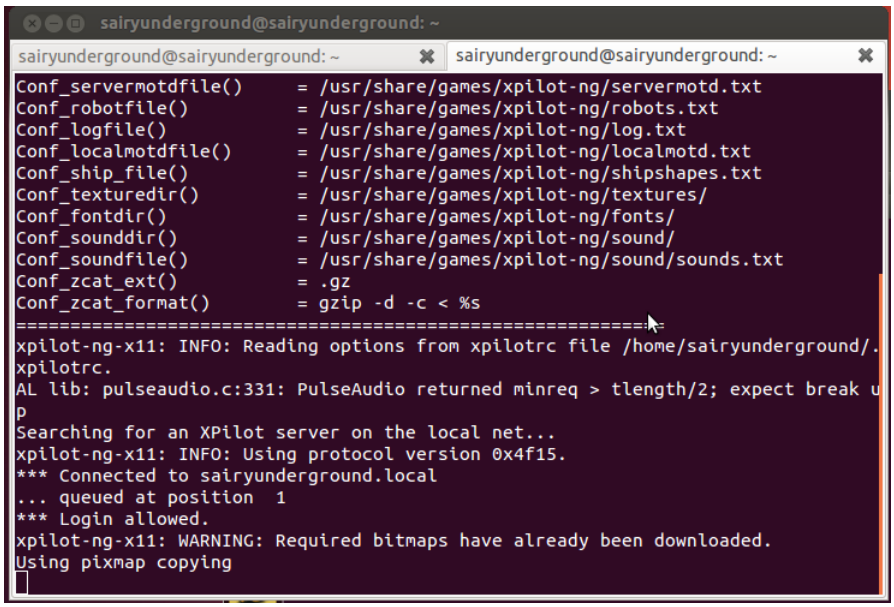

Figura 4.2: Prueba de Xpilot: Inicialización de Xpilot

Figura 4.3: Xpilot Entorno X11

Figura 4.4: Xpilot Entorno SDL

4.1.2. Servidor

En la Raspberry Pi 3, se abre la terminal LXTerminal, y se procede a iniciar el servidor con el comando **xpserver** (ver Fig. 4.5)

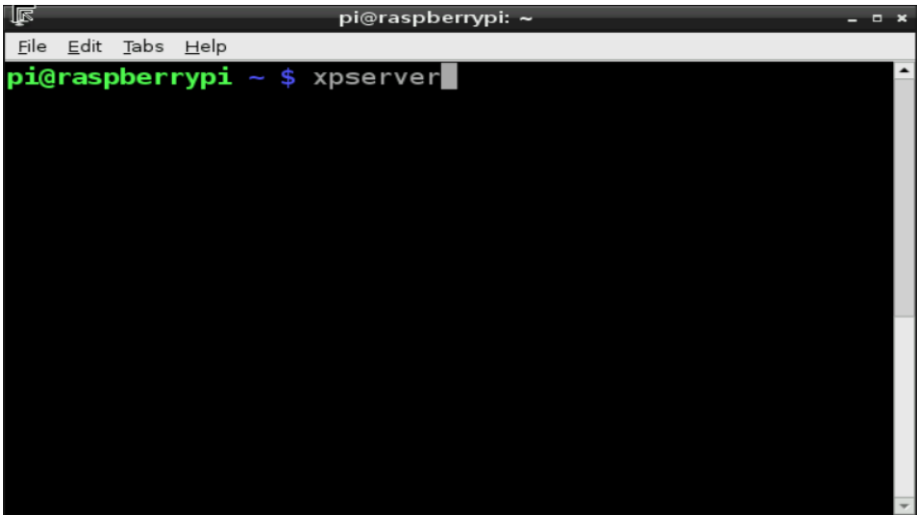

Figura 4.5: Comando xpserver

En la Fig. 4.5 se puede observar una captura de LXTerminal, una vez que se ha iniciado el servidor.

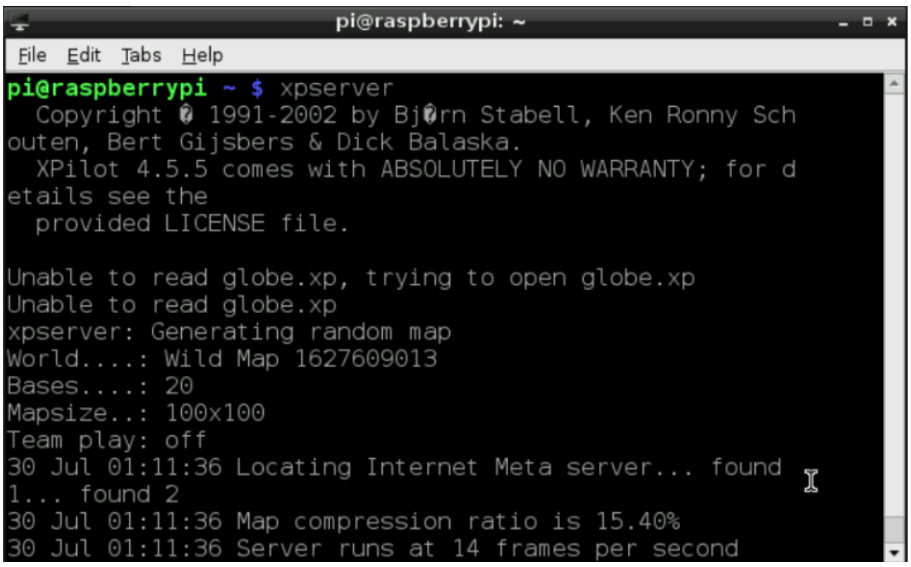

Figura 4.6: Servidor iniciado

4.2. Pruebas

En el caso de este proyecto, los scripts de las naves robot, están almacenadas en la carpeta Bots de la carpeta Documentos (ver Fig. 4.7).

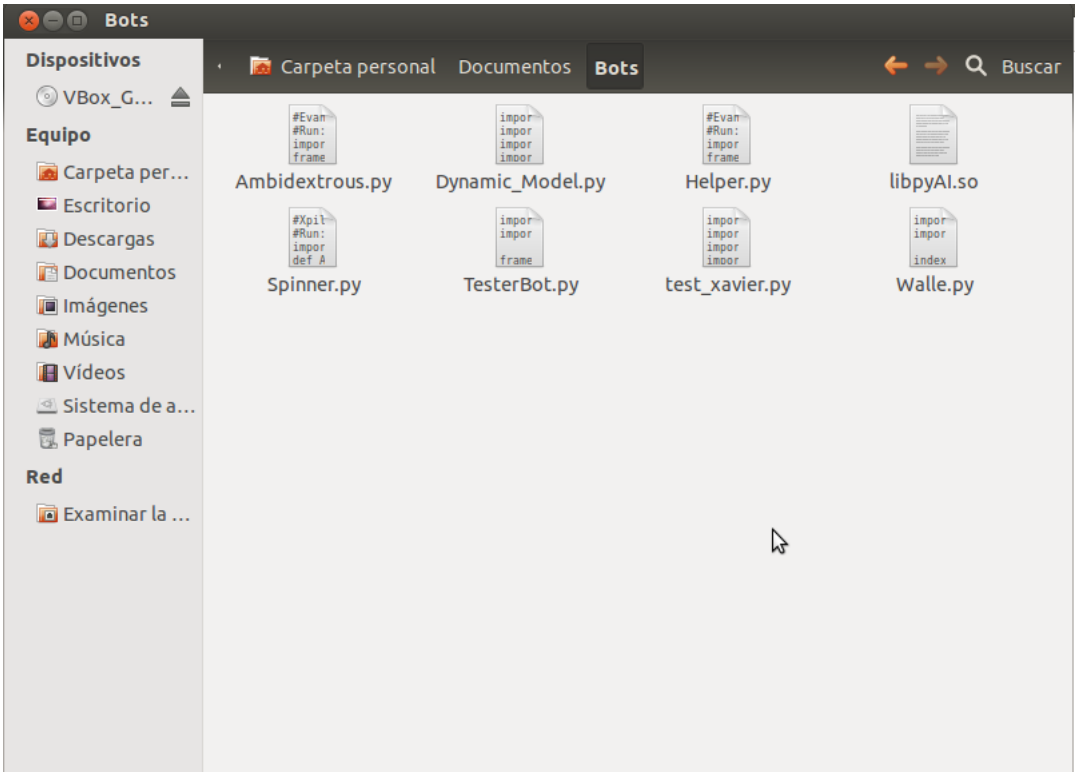

Figura 4.7: Dirección de documentos

Para poder ensayar el script de la nave robot, se abre una terminal con **Ctrl+Alt+T**, a continuación se digita **cd Documentos** y una vez en la carpeta Documentos se digita **cd Bots**. De esta manera se accede al contenido de la carpeta Bots.
El ensayo de la nave robot que en este caso lleva por nombre *Dynamic_model.py*, se realiza utilizando el comando **python3 Dynamic_model.py**. Todo lo anteriormente mencionado se puede observar en la Fig. 4.8. A continuación, el entorno Xpilot abre una ventana con la opción de conexión a los posibles servidores que existen en la red local. En este caso, como ya se mostró en la Fig. 4.6 existe un servidor de nombre **pi** montado en la Raspberry Pi 3. Esto se puede evidenciar, en la Fig. 4.9. Por lo que se debe conectarse a dicho servidor.

En la Fig. 4.10 se puede visualizar el funcionamiento de la nave robot *Dynamic_Model.py* desempeñándose en su entorno. En este caso se muestra a la nave realizando un giro al detectar un muro en su trayectoria.

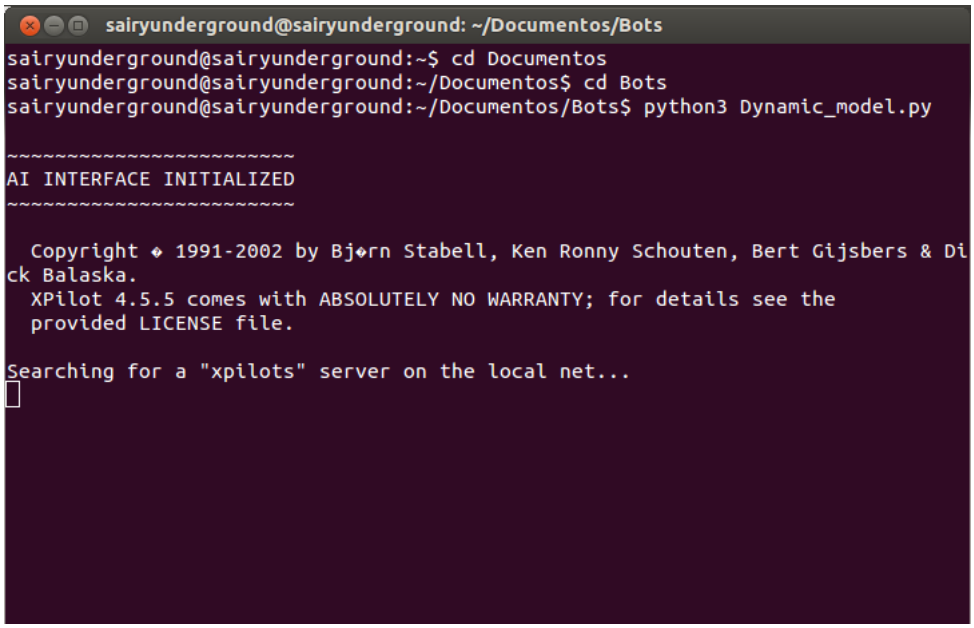

Figura 4.8: Acceso a la dirección de los Scripts

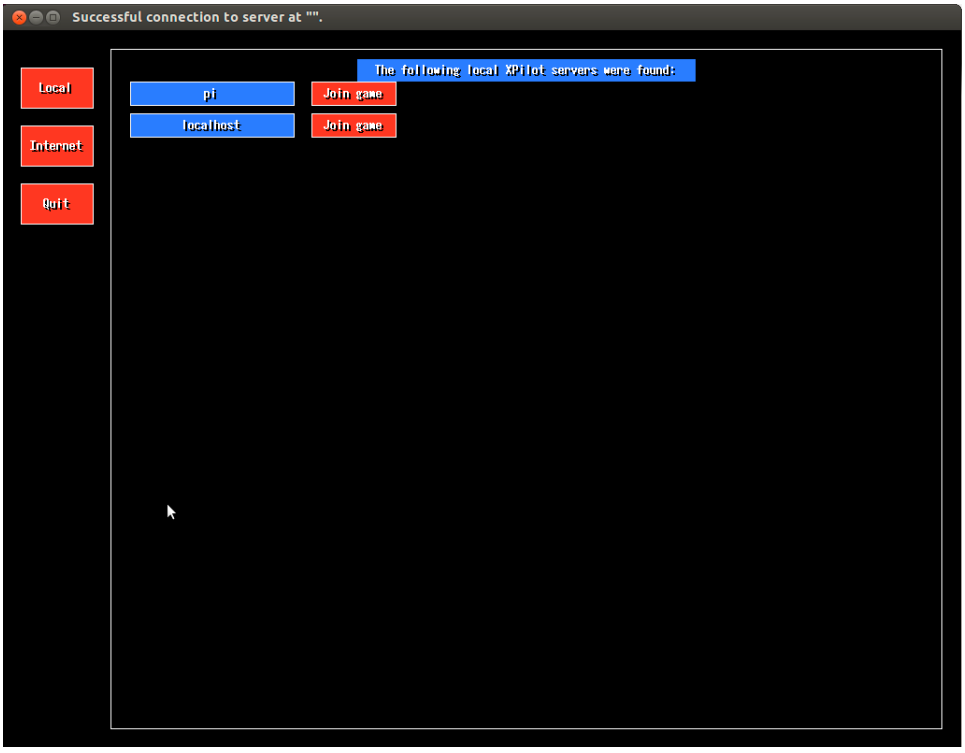

Figura 4.9: Servidores Disponibles

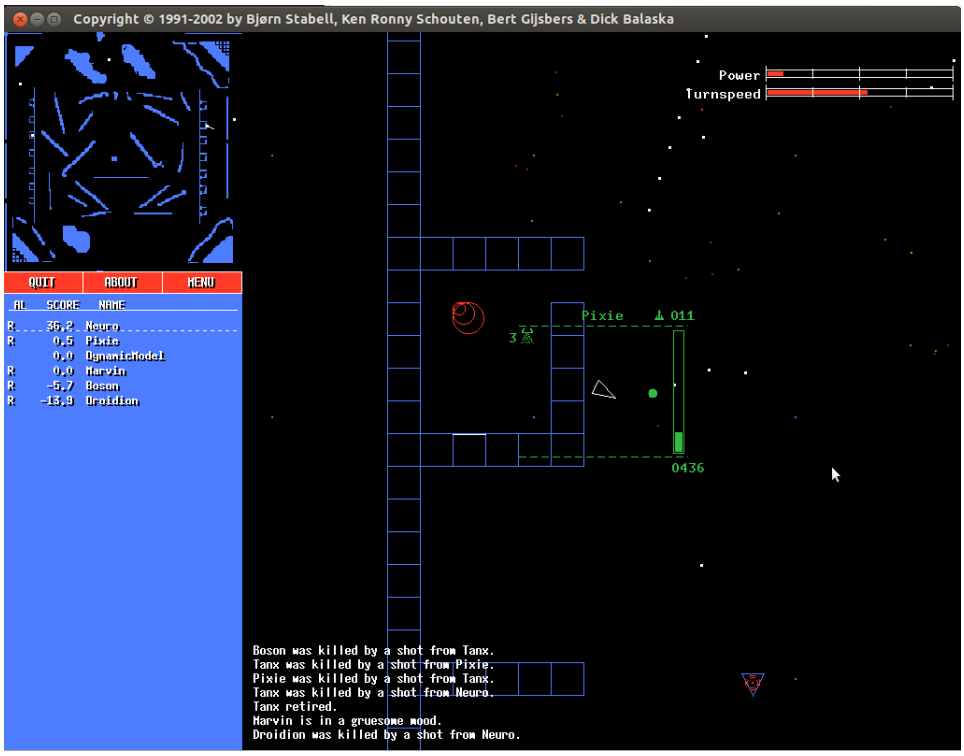

Figura 4.10: Desempeño Nave Robot: Dynamic_model.py

En las Fig. 4.11 y Fig. 4.12 se puede observar a la nave robot antes y después de detectar un enemigo. Una vez detectado el enemigo, la orientación de la nave cambia completamente y se dirige a atacarlo.

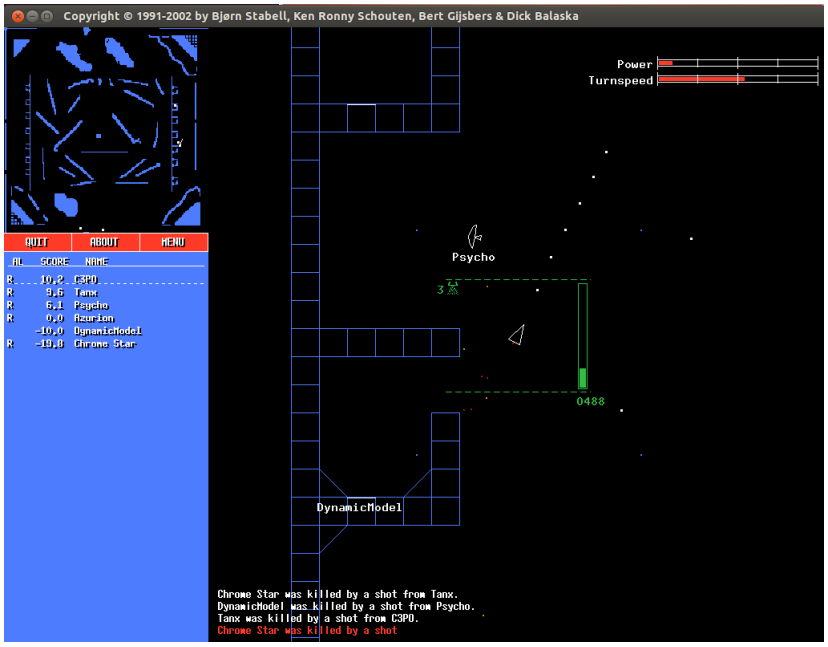

Figura 4.11: Pruebas nave robot: Detección del enemigo

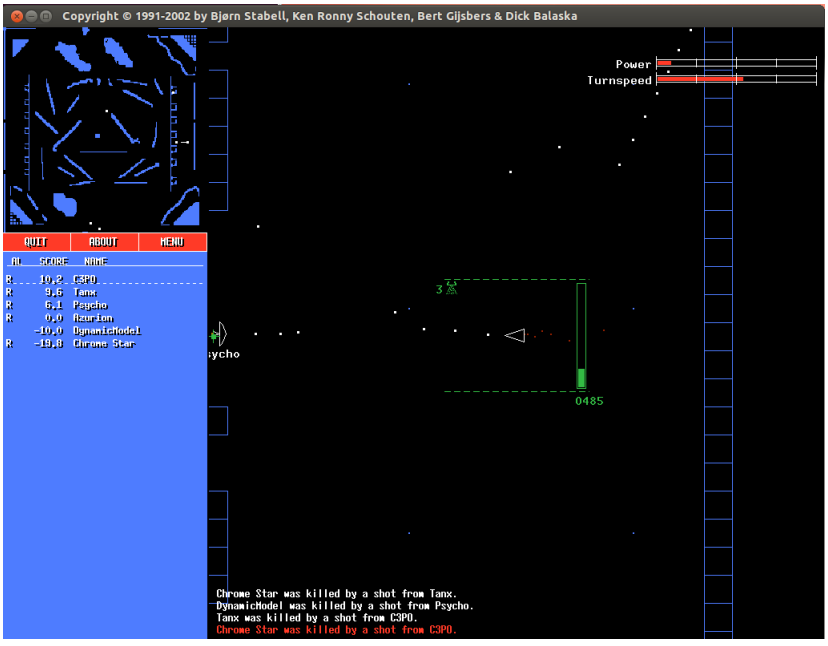

Figura 4.12: Pruebas nave robot: Seguimiento del enemigo, modelo dinámico.

Capítulo 5

Conclusiones y trabajo futuro

Este capítulo muestra las conclusiones del presente proyecto y esboza algunas líneas posibles de trabajo futuro.

5.1. Conclusiones

- Se realizó el modelado dinámico del robot tomando en cuenta las variables de posición angular, posición en el plano (coordenadas X,Y) y velocidad.
- En este caso, la variable que se controló fue la velocidad para ello fue necesario conocer las coordenadas de las naves enemigas, mismas que fueron detectadas con la función look4enemy misma que se puede apreciar en 6.1.1.
- Se logró implementar un servidor remoto con la ayuda de Xpserver, mismo que es distribuido por XPlitot, en la plataforma Raspberry Pi 3 B.

5.2. Trabajo futuro

La planeación de trayectorias queda como un pendiente para trabajos futuros. Dicho trabajo ayudaría a describir un movimiento deseado al robot haciéndolo mucho más autónomo. De esta manera, el robot podría evadir cualquier tipo de obstáculos, incluso los proyectiles lanzados por las naves robot enemigas.

Capítulo 6

Listado de códigos

Este apéndice incluye el listado de código(s) desarrollado(s) en el proyecto. Sólo se ha anexado el archivo más importante, en este caso, el script de bot ensayado.

6.1. XPilot AI

6.1.1. Script de la Nave Robot (Dynamic_Model.py)

Listing 6.1: Script Modelo Dinámico

```
import numpy as np
import math as mt
import libpyAI as ai
import os

#global constants and definitions
class robot:
    mainEnemy = 'Uzi' #the name of the most important target
    speedLimitHi = 100  #if we overcome it, the speed limiter will brake the ship
    speedLimitLo = 80
    distMin = 300
    point1X = 1200
    point1Y = 1200
    point2X = 1800
    point2Y = 1800
    point3X = 1200
    point3Y = 1800
    point4X = 1800
    point4Y = 1200


# Global variables
tickCount = 0
mode = "wait"
targetCount = 0
targetDist = 0
targetName = ' '
shieldFlag = 1
```

```
counterRunAway = 0
calcAngle = 0

actionLimit = 0
currentObjective = 1

def shield(command):

    global shieldFlag

    if command == 1 & shieldFlag == 0:
        shieldFlag = 1

    elif command == 0 & shieldFlag == 1:
        shieldFlag = 0

    ai.shield()

def avoidSpeedLimit():

    global actionLimit

    if ai.selfSpeed() >= robot.speedLimitHi:
        print("Speed limit overcame, reducing!!!")
        ang = mt.atan2(ai.selfVelY(), ai.selfVelX())*180/np.pi
        if ang < 0:
            ang = ang + 360
        ang = ang - 180 #changes the direction
        if ang < 0:
            ang = ang + 360
        print(ang)
        ai.turnToDeg(int(ang))  #angle correction
        ai.setPower(20)
        ai.thrust(1)
        actionLimit = 1
    elif (ai.selfSpeed() <= robot.speedLimitLo):
        if actionLimit == 1:
            print("Speed reduced!!!")
            actionLimit = 0
            ai.thrust(0)
    else:
        if actionLimit == 1:
            ai.setPower(20)
            ai.thrust(1)
    return

def shot(counter):
    for i in range(0, counter):
        ai.fireShot()
    return

def look4Enemy():
    Id = ai.closestShipId()  # look for the closest ship

    if Id != -1:
        Dist = ai.enemyDistanceId(Id)
        Name = ai.enemyNameId(Id)
        X = ai.screenEnemyXId(Id)
        Y = ai.screenEnemyYId(Id)
        print("Id:", Id, Name, "D", Dist, "X", X, "Y", Y)
        return [Id, Name, Dist, X, Y]
    else:
        #print("Closest target not found!")
        return [-1, '', -1, -1, -1]
```

```
def wallDetection():
    wall_feeler1 = ai.wallFeeler(500, int(ai.selfHeadingDeg()), 0, 0)
    print("The wall is", wall_feeler1, "pixels away")

    return

def angleTurn(xGoal, yGoal):
    #my position and orientation
    xNow = ai.selfX()
    yNow = ai.selfY()
    theta = ai.selfHeadingDeg() * np.pi / 180

    deltaX = xGoal - xNow
    deltaY = yGoal - yNow
    angg = mt.atan2(deltaY, deltaX)

    if angg < 0:
        angg = angg + 2 * np.pi

    return int(angg * 180 / np.pi)

def gotoXY(x_goal, y_goal, k_vel, k_power):
    #my position and orientation
    x_now = ai.selfX()
    y_now = ai.selfY()
    theta = ai.selfHeadingDeg() * np.pi / 180

    #my velocity
    vx_now = ai.selfVelX()
    vy_now = ai.selfVelY()
    v_now_mod = np.sqrt(np.power(vx_now, 2) + np.power(vy_now, 2))
    v_now = [vx_now, vy_now]

    #desired velocity
    delta_x = x_goal - x_now
    delta_y = y_goal - y_now
    dist_des= np.sqrt(np.power(delta_x, 2) + np.power(delta_y, 2))
    v_des_mod = dist_des/ k_vel
    alfa = mt.atan2(delta_y, delta_x)

    if alfa < 0:
        alfa = alfa + 2 * np.pi
    v_des = [v_des_mod * mt.cos(alfa), v_des_mod * mt.sin(alfa)]

    #controlled velocity
    v_comp = [x - y for x, y in zip(v_des, v_now)]
    v_comp_mod = np.sqrt(np.power(v_comp[0], 2) + np.power(v_comp[1], 2))
    phi = int(mt.floor(mt.atan2(v_comp[1], v_comp[0]) * 180 / np.pi))

    if phi < 0:
        phi = phi + 360

    #apply the correction
    ai.turnToDeg(phi)  #angle correction

    power = v_comp_mod * k_power

    if power > 55:
        power = 55
    else:
        power = 0.0

    ai.setPower(power)
```

```
    if v_des_mod > 1:
        ai.thrust(1)
    else:
        ai.thrust(0)

    print('x %d y %d v_now %4.1f v_des %4.1f angle %d power %3.1f' %(x_now, y_now, v_now_mod,
         v_des_mod, phi, power))

    if dist_des <= 200:
        return 1
    else:
        return 0

def mainLoop():
    #global variables
    global tickCount, mode, shieldFlag, actionLimit, currentObjective
    global counterRunAway
    global calcAngle

    try:
        #resets the state machine and important registers if we die
        if not ai.selfAlive():
            tickCount = 0
            mode = "wait"
            shieldFlag = 1  # shield always starts activated
            actionLimit = 0 # speed limitation process deactivated
            ai.setPower(0)
            ai.thrust(0) #turn off thrust

            print("Reviving now!!!")
            return

        tickCount += 1

        print("Tick count:", tickCount, "state:", mode)

        avoidSpeedLimit()

############################################################
#       STATE MACHINE
############################################################

        if mode == "wait":

            targetInfo = look4Enemy() #search for enemies
            if targetInfo[0] != -1: #enemy detected
                mode = "aim"
                print(ai.aimdir(targetInfo[0]))

            if currentObjective == 1:
                flag = gotoXY(robot.point1X, robot.point1Y, 100, 0.1)
                if flag == 1:
                    currentObjective = 2
            elif currentObjective == 2:
                flag = gotoXY(robot.point2X, robot.point2Y, 100, 0.1)
                if flag == 1:
                    currentObjective = 3

            elif currentObjective == 3:
                flag = gotoXY(robot.point3X, robot.point3Y, 100, 0.1)
                if flag == 1:
                    currentObjective = 4
            elif currentObjective == 4:
```

```
            flag = gotoXY(robot.point4X, robot.point4Y, 100, 0.1)
            if flag == 1:
                currentObjective = 1

        elif mode == "aim":

            targetInfo = look4Enemy() #search for enemies
            if targetInfo[0] == -1: #no enemies detected
                mode = "wait"
            else: #enemies detected
                calcAngle = angleTurn(targetInfo[3], targetInfo[4]) #calculate angle to turn
                if calcAngle < 0:
                    calcAngle = calcAngle + 360
                ai.turnToDeg(calcAngle)  #apply angle correction

                shield(0)
                shot(10) #shoot to target
                shield(1)

                if targetInfo[2]<=robot.distMin:
                    mode = "follow"
                else:
                    mode = "aim"

        elif mode == "follow":
            shield(0)
            shot(5)
            shield(0)
            targetInfo = look4Enemy() # search for enemies
            if targetInfo[0] == -1: #no enemies detected
                mode = "wait"
            else: #enemies detected
                if targetInfo[2]<=robot.distMin:
                    calcAngle = angleTurn(targetInfo[3], targetInfo[4]) #calculate angle to
                        turn
                    if calcAngle < 0:
                        calcAngle = calcAngle + 360
                    ai.turnToDeg(calcAngle)  #apply angle correction
                    shield(0)
                    shot(5)
                    shield(0)
                    flag = gotoXY(robot.point1X, robot.point1Y, 100, 0.1)
                    mode = "follow"
                else:
                    mode = "aim"

    except:

        # If tick crashes, print debugging information
        print(traceback.print_exc())

#os.system("start-xpilot-ng-server -map ~/Desktop/xpilot-ai/maps/simple.xp")
#ai.start(mainLoop, ["-name", "DynamicModel", "-host", "AMALIA.local", "-join"])
ai.start(mainLoop, ["-name", "DynamicModel", "-host", "esaii75.upc.es", "-join", "-
    turnResistance", "0"])
```

Referencias

[1] X.-A. 2019, «XPILOT AI,» Julio 2021. [En línea]. Available: http://xpilot-ai.org/index.php/22-home. [Último acceso: 20 Julio 2021].

[2] J.-C. Latombe, «Robot Motion Planning,» SPRINGER SCIENCE+BUSINESS MEDIA, LLC, Standford.

[3] Andaluz, Gabriela; Andaluz, Victor H.; Rosales, Andrés;, «Modelación, Identificación y Control de Robots Móviles,» Ambato, 2012.

[4] G. B. Parker y D. A. Arroyo, «THE XPILOT-AI ENVIRONMENT,» World Automation Congress, 2010.

[5] C. Harrell, B. Ghosh y R. Bowden, «Simulation Using Promodel,» New York, 2004.

[6] M. B. Ortiz-Moctezuma, «Sistemas dinámicos en tiempo continuo: Modelado y simulación,» OmniaScience, 2015.

[7] G. B. Parker, M. Parker y S. D. Johnson, «Evolving Autonomous Agent Control in the Xpilot Environment,» Edinburgh, 2005.

[8] Solaque Guzmán, Leonardo Enrique; Molina VIlla, Manuel Alejandro; Rodríguez Vásquez, Edgar Leonardo;, «SEGUIMIENTO DE TRAYECTORIAS CON UN ROBOT MÓVIL DE CONFIGURACIÓN DIFERENCIAL,» 2014.

[9] L. Solaque Guzmán, N. Muñoz Cevallos y P. Niño Suárez , «PLANIFICACIÓN DE TRAYECTORIAS PARA UN ROBOT CON RESTRICCIONES DINÁMICAS,» Ciencia e Ingeniería Neogranadina, Bogotá, 2008.

[10] Gallardo, Domingo; Colomina, Otto; Flórez, Francisco; Rizo , Ramón;, «Generación de trayectorias robustas mediante computación evolutiva,» [En línea]. Available: http://www.dccia.ua.es/ domingo/articulos/gallardo97-SCETA.pdf. [Último acceso: 23 Enero 2019].

[11] M. Parker y GB Parker, Üsing a Queue Genetic Algorithm to Evolve Xpilot Control Strategies on a Distributed System", 2006 IEEE International Conference on Evolutionary Computation , 2006, págs. 1202-1207, doi: 10.1109 / CEC.2006.1688446.

[12] G. B. Parker and M. H. Probst, Üsing evolution strategies for the real-time learning of controllers for autonomous agents in Xpilot-AI,ÏEEE Congress on Evolutionary Computation, 2010, pp. 1-7, doi: 10.1109/CEC.2010.5586222.

[13] M. Allen and P. Fritzsche, Reinforcement learning with adaptive Kanerva coding for Xpilot game AI,"2011 IEEE Congress of Evolutionary Computation (CEC), 2011, pp. 1521-1528, doi: 10.1109/CEC.2011.5949796.

[14] M. Parker and G. B. Parker, "The Evolution of Multi-Layer Neural Networks for the Control of Xpilot Agents,"2007 IEEE Symposium on Computational Intelligence and Games, 2007, pp. 232-237, doi: 10.1109/CIG.2007.368103.

[15] Reyes Vera, J. M. (2012). Máquinas abstractas de estados y sus aplicaciones. Revista Educación En Ingeniería, 7(13), 55-62. https://doi.org/10.26507/rei.v7n13.160

[16] Funes, P. A differential game and A learning game Bot.