TecnoCampus
Escola Superior Politècnica

Centre adscrit a la
upf. Universitat Pompeu Fabra Barcelona

Grau en Enginyeria Electrònica Industrial i Automàtica.

MACHINE LEARNING PER EVITAR OBSTACLES AMB TURTLEBOT 3 BURGUER

Xavier Ato Adris
PONENT: Josep Lòpez Xarbau

TARDOR 2022

TecnoCampus
Mataró-Maresme

Sumari de documents

TecnoCampus
Escola Superior Politècnica

Centre adscrit a la
upf. Universitat Pompeu Fabra Barcelona

Grau en Enginyeria Electrònica Industrial i Automàtica.

MACHINE LEARNING PER EVITAR OBSTACLES AMB TURTLEBOT 3 BURGUER

Memòria

Xavier Ato Adris
PONENT: Josep Lòpez Xarbau

TARDOR 2022

TecnoCampus
Mataró-Maresme

Dedicatòria

M’agradaria dedicar aquest projecte als meus pares i a les meves germanes, els quals m’han donat el suport i la motivació en els moments que més ho he necessitat.

Agraïments

En primer lloc, agrair-li al ponent Josep Lòpez Xarbau per el suport i la motivació aportada durant el transcurs d'aquest projecte.

Seguidament, voldria agrair a la meva família per fer possible que pogués estudiar el que sempre he desitjat.

Resum

En aquesta memòria es pot contemplar un estudi, amb la finalitat de conèixer què és el Machine Learning per evitar obstacles i com el podem implementar fent servir el software ROS, mostrant quines són les llibreries i softwares externs que necessitem per aconseguir el nostre objectiu. Un cop haguem fet les proves i simulacions necessàries, compilarem el nostre programa al robot mòbil TurtleBot3 Burger per dur a terme la posada en marxa d'aquest i fer la comprovació final.

Resumen

En esta memoria se puede contemplar un estudio, con el fin de conocer qué es el Machine Learning para evitar obstáculos y como lo podemos implementar usando el software ROS, mostrando quales son las librerías y softwares externos que necesitamos para conseguir nuestro objetivo. Una vez hayamos hecho las pruebas y simulaciones necesarias, compilaremos nuestro programa al robot móvil TurtleBot3 Burger para llevar a cabo la puesta en marcha de este y hacer la comprobación final.

Abstract

In this memory we can contemplate a study, in order to know what Machine Learning to avoid obstacles is and how we can implement it using the ROS software t, showing what are the libraries and external software we need to achieve our goal. Once we have done the necessary tests and simulations, we will compile our program to the mobile robot TurtleBot3 Burger to carry out the implementation of this and make the final check.

ÍNDEX

ÍNDEX DE FIGURES

ÍNDEX DE TAULES

GLOSSARI DE TERMES

API: Interfície de programació d'aplicacions.

CPU: Unitat Central de Processament.

DQN: Deep Q Network

DDQN: Deep-Deep Q Network

GFS: Google File System.

GPL: General Public License.

GPU: Unitat de Processament Gràfic.

IoT: Internet of Things

IP: Internet Protocol

Iaas: Infraestructura com a servei.

ML: Machine Learning.

Paas: Plataforma com a servei.

ROS: Robot Operating System.

1. Objectius.

1.1. Propòsit.

El propòsit principal d'aquest projecte és realitzar un estudi del funcionament de Machine Learning d'evasió d'obstacles aplicat amb ROS. Per fer la comprovació final, es treballarà amb el robot mòbil Turtlebot 3 Burger. Aquest estudi contindrà un manual d'instruccions on es veurà reflectit la instal·lació de ROS en un sistema operatiu Ubuntu, com l'entorn de Turtlebot i els diferents sofwares i llibreries necessàries per desenvolupar el programa de Machine Learning.

1.2. Finalitat.

La finalitat del projecte és realitzar un estudi del funcionament de Machine Learning aplicat amb ROS amb l'objectiu d'aconseguir que el robot Turtlebot 3 Burger pugui moure's per qualsevol entorn sense col·lidir amb cap obstacle, per dur a terme aquest propòsit ell mateix anirà aprenent dels errors fins a arribar al seu destí. Treballarem en diferents estacions de treball, sense obstacles, amb obstacles i amb obstacles mòbils i estàtics.

1.3. Objecte.

L'objecte del present projecte és un estudi, coneixent quin és l'entorn amb el qual es pot desenvolupar aplicacions de Machine Learning aplicat a ROS. Aquest projecte tindrà un caire industrial, és a dir, la funcionalitat d'aquesta es podria implementar per diferents segments industrials.

1.4. Abast.

S'inclou un petit guió on s'explicarà com instal·lar ROS en el sistema operatiu d'Ubuntu. Per altra banda, s'explicarà quins passos s'han de seguir per poder instal·lar el Software necessari per poder aconseguir el nostre objectiu. Es mostrarà l'evolució d'aprenentatge de la nostra màquina amb la solució final d'aquesta, vist des de diferents escenaris, un sense obstacles, un altra amb obstacles estàtics un altre amb obstacles mòbils i per acabar un escenari amb obstacles estàtics i mòbils.

2. Introducció.

Actualment, la robòtica es troba cada cop més present en el nostre dia a dia, ja sigui en àmbits com podem ser en la indústria, en el domicili particular, en el treball, aplicat en les mateixes persones, etc. Per aquest motiu aquesta tecnologia s'està desenvolupant molt ràpidament per poder cobrir necessitats que els humans hem creat per poder tenir una vida més fàcil i senzilla.

Aquesta tendència és causada per un fet innat en el qual l'ésser humà tendeix a optimitzar al màxim les tasques a realitzar, des de dissenyar una roda per poder desplaçar-te amb major facilitat fins a automatitzar tota una planta industrial per dur a terme tasques que per un humà portaria tot un dia a realitzar, un robot les fa en hores. Fins al punt d'investigar tècniques com el "*Machine Learning*" el qual és un camp d'investigació dedicat a comprendre i crear mètodes que facin que una màquina "*aprengui*".

En aquest projecte, el robot amb el qual farem l'estudi i les diferents simulacions serà amb el robot mòbil Turtlebot3 Burger.

2.1 Objectius.

El projecte sorgeix de la necessitat de realitzar un estudi sobre el Machine Learning, un camp de treball el qual no està molt desenvolupat i és pràcticament nou. Per altra banda, per afegir-li una mica més de complexitat i innovació es dissenyarà una aplicació basada en el Machine Learning a partir del software ROS dedicat a l'evasió d'obstacles en temps real, on posteriorment es farà la comprovació amb el robot mòbil Turtlebot3 Burger.

El que es pretén amb aquest projecte és desenvolupar una aplicació basada en el Machine Learning utilitzant el software ROS amb l'ajuda del robot mòbil Turtlebot3 Burger. Aquesta aplicació, consistirà en el fet que el robot sigui capaç de desenvolupar-se per qualsevol entorn, sense la necessitat de tenir la un mapeig previ i poder evitar els obstacles que van apareixent en temps real.

Per poder dur a terme aquest projecte, és necessari un estudi previ sobre el funcionament del Machine Learning i com aplicar-ho al software ROS. Per tant, d'aquesta manera es pretén fer un anàlisi del software necessari i quina metodologia seguir per poder desenvolupar l'aplicació.

2.2 Revisió d'antecedents i necessitats d'informació.

2.2.1 Història del Machine Learning.

Per molt modern que sembli el concepte de 'Machine Learning' realment per poder arribar a l'origen ens remuntem a mitjans del segle XX on els matemàtics Walter Pitts i Warren McCulloch publiquen el primer model matemàtic de les xarxes neuronals humanes, on intenten fer un mapatge matemàticament del pensament i la presa de decisions en els éssers humans.

Uns anys després en l'any 1950 el matemàtic Alan Turing va dissenyar el 'Test de Turing' el qual es basa en el fet que un humà es troba davant d'una màquina sense veure-la i aquesta ha d'enganyar a l'humà perquè es pensi que es troba davant d'una persona.

Per altra banda, el 1952 Arthur Samuel va escriure el primer algoritme capaç d'aprendre. El qual es basa en un programa el qual juga una partida a les dames i en cada partida va aprenent com millorar els seus moviments, naixen així el terme de 'Intel·ligència Artificial'. Passant-se a dir algoritme "*minimax*". [1]

Durant el 1960 es desenvolupa l'algoritme 'Nearest Neighbor' que es considera com el primer algoritme de reconeixement de patrons. I no va ser fins a l'any 2003 quan es torna a reactivar l'estudi d'aquest camp, quan es publica un estudi sobre fitxers distribuïts anomenat 'Google File System' (GFS) i finalitzarà en l'any 2004 quan Google crea un paradigma de processament distribuït anomenat 'Map & Reduce'.

Google no deixa d'investigar en aquest camp fins que a l'any 2006 quan enginyers d'Apache porten a la seva culminació els paradigmes de Google creant la primera plataforma Big Data Open Source anomenada Hadoop aconseguint una gran potència de càlcul amb una gran abundància de dades disponibles. On gràcies al Big Data, el Machine Learning s'ha pogut desenvolupar de forma espectacular.[2]

Fins avui dia, on ens trobem en un punt d'expansió gegant, en què gràcies als avenços que hem aconseguit i les noves tecnologies que hem descobert podent-les aplicar al Machine Learning, es poden trobar un munt d'aplicacions en qualsevol sector, ja sigui empresarial, industrial, sanitari...

2.2.2 Machine Learning.

Des de cotxes autònoms, assistents de traducció en qualsevol idioma, surgències de compres personalitzades i molt més. Tasques complexes que fins fa uns anys era impossible de pensar, avui dia són una realitat gràcies al 'Machine Learning' (aprenentatge autònom). El Machine Learning és una disciplina de la intel·ligència artificial què, a través dels algoritmes que té, dota als ordinadors de la capacitat d'identificar patrons en dades massives i elaborar prediccions (anàlisi predictiu). Aquest anàlisi permet als computadors realitzar tasques específiques de forma autònoma, és a dir, sense necessitat de ser programades. [3] Hi han diferents mètodes d'aquest tipus d'aprenentatge, per algoritmes genètics, per xarxes neuronals, per reforç, etc.

Aquesta tecnologia és capaç d'implementar una resposta a una sèrie de dades d'entrada després d'un processament basat en predicció. La forma que té d'interpretar les dades és mitjançant matrius o vectors, cada fila de la matriu és una dada i cada columna és un factor o una característica. Existeixen varius tipus de models de predicció per poder implementar el Machine Learning. [4]

Aprenentatge supervisat

És tracta d'un model el qual és entrenat amb un conjunt de dades d'entrada conegudes prèviament per a l'obtenció de resultats. Aquest model permet adaptar i ajustar a partir de les dades de sortida, els paràmetres interns per poder ser capaç de fer prediccions amb dades que no s'han utilitzat en l'entrenament. Per tant, l'objectiu és realitzar un mapatge amb entrades i generar sortides mitjançant el Big Data i el IoT.

L'algoritme aprèn de la informació que les persones van incorporant. Per tant, aquest tipus d'aprenentatge requereix la intervenció humana per poder etiquetar, classificar i introduir les dades.

Existeixen tres tipus de models d'aprenentatge en el Machine Learning [5]:

- Regressió: Aquesta aplicació entrena a un algoritme per predir una sortida d'un rang continu de valors possibles. És a dir s'encarrega d'aproximar una funció continua segons les dades introduïdes.
  En la regressió un algoritme necessita identificar una relació funcional entre els paràmetres d'entrada i sortida. El valor de sortida no és discret com veurem en la

classificació, sinó que és una funció dels paràmetres d'entrada. L'exactitud d'un algoritme de regressió és calcula en funció de la desviació entre la sortida precisa i la sortida prevista.

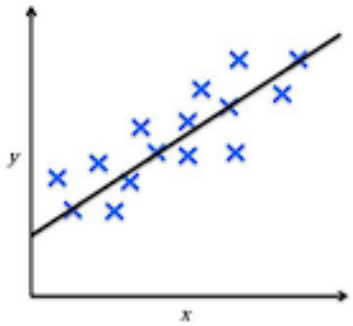


Fig 2.1. Gràfica d'aprenentatge per regressió.

Font: [3]

- Classificació: La classificació entrena un algoritme per classificar les dades en variables discretes. Durant l'entrenament els algoritmes rebran dades d'entrada amb una etiqueta de 'classificació'. L'objectiu es desenvolupar una funció aproximada i trobar relacions entre els punts que s'han entrat i els nous amb la finalitat de classificar uns en una banda i els altras en l'altra.

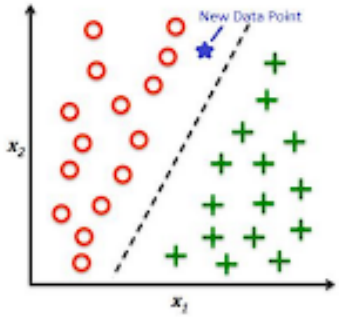


Fig 2.2. Gràfica d'aprenentatge per classificació.

Font: [3]

Aprenentatge no supervisat

En aquest mètode d'aprenentatge, hi ha dades sense etiquetar i el mateix algorisme les ha d'intentar entendre per si mateix a diferència del supervisat en el qual les dades ja van etiquetades per facilitar l'aprenentatge. L'objectiu és simplement deixar que la màquina

aprengui sense cap ajuda o indicació, ajustant els resultats i agrupacions, quan els resultats siguin els adequats, permetent que la màquina comprengui les dades i les processi com millor li sembli.[6]

Existeixen dues categories:

- Agrupament: És una tècnica que analitza les dades, les classifica segons la informació obtinguda d'aquestes sense conèixer la seva estructura. Amb l'objectiu d'aconseguir un nombre de grups que vagin classificant segons les característiques que dona la mateixa dada.

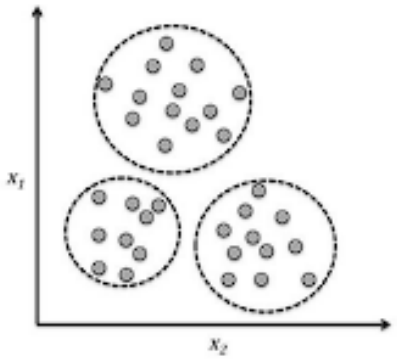


Fig 2.3. Gràfica aprenentatge per agrupament.
Font: [4]

- Reducció dimensional: El fet de poder treballar amb dades que tinguin vàries dimensions es una realitat. El seu funcionament es basa trobant similituds i relacions entre les característiques, pel que alguna informació pot ser redundant, depenent d'una altra linealment, etc... Per aquest motiu, l'objectiu principal és comprimir les dades més importants en un subespai més reduït per al seu tractament.

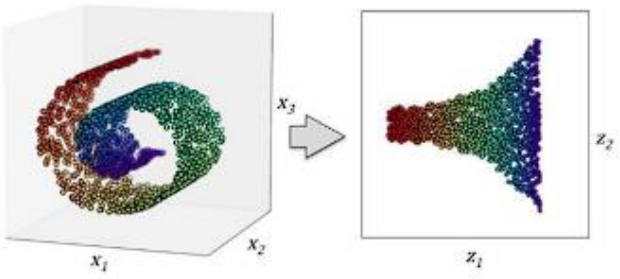


Fig 2.4. Gràfica d'aprenentatge per reducció dimensional.

Font: [4]

Aprenentatge reforçat

Consisteix en un aprenentatge profund el qual també se'l denomina "Deep Learning". Per la construcció de models basats en aquest tipus d'aprenentatge es prenen com a referència el resultat de cada interacció i s'utilitza una recompensa com a paràmetre.

Aquesta metodologia està basada en la psicologia conductista per aconseguir que una màquina trobi l'opció correcta fent que els encerts generin una recompensa i les falles penalitzacions. Obtenint que si la decisió resulta ser beneficiosa, la màquina ho aprèn de forma automàtica per torna a repetir-la en el futur. Quan falla evitarà tornar a repetir el patró fet servir. Hi ha molts mètodes d'aprenentatge profund com poden ser Q-Learning, DQN, Deep-DQN, Duel-DDQN i Actor-Critic.

- **Q-LEARNING:** És un algoritme d'aprenentatge basat en valors i es centra en l'optimització de la funció segons el entorn o el problema. Aquest model representa la qualitat amb la que un model troba la seva propera acció millorant la qualitat.
- **DQN:** En aquest algoritme combina l'algoritme de Q-Learning amb xarxes neuronals profundes per aproximar la funció Q, evitant així utilitzar una taula per representar la mateixa. Per altre banda, l'agent selecciona l'acció per el valor de Q i el mostra en l'ambient i emmagatzema la recompensa obtinguda per l'ambient, l'agent actualitza d'una forma aleatòria.

2.2.3 Implementació del Machine Learning.

Una de les principals raons per el que la implementació dels models de Machine Learning són complexes es degut a que inclòs la forma en la qual es formula el model, tendeix a ser enganyosa. Ja que en realitat, en un sistema típic, el model es una petita fracció d'un sistema global.

Per tant, en els sistemes de Machine Learning es requereix de la cooperació entre múltiples equips i especialitats. Per tant, no es pot implementar de forma aïllada, sinó que requereix d'una planificació a nivell de sistema. [7]

La planificació del sistema és la següent:

Arquitectura del sistema

Aquest és el punt de partida de qualsevol model de Machine Learning, ja que en aquest punt definirem els requisits i els objectius que volem aconseguir. No cal buscar-hi ja una solució tecnològica, primer de tot s'ha d'entendre les limitacions que tenim, quin valor està creant i per a qui va dirigit.

Per definir una bona arquitectura s'hauria de plantejar alguna de les següents preguntes:

- ¿Quin temps necessitem per poder dur a terme les prediccions, milisegons, segons, en temps real...?
- ¿Amb quina freqüència esperem actualitzar els models?
- ¿Quina és la magnitud de dades a tractar?
- ¿Quin tipus d'algoritmes esperem utilitzar?
- ¿ Es pot aplicar realment el Machine Learning a aquesta tasca?
- ¿ Quin sistema d'aprenentatge utilitzarem?

Un cop determinada la llista, la qual pot variar depenent de les especificacions que ens requereixen, ja que cadascuna d'aquestes opcions té avantatges i desavantatges a partir dels detalls del model definim el llenguatge de programació amb el qual és dura a terme el model.

Selecció del llenguatge de programació

La selecció del llenguatge de programació és un punt clau de la implementació del Machine Learning, ja que depenent dels requisits i les especificacions haurem de treballar amb un llenguatge o un altra.

Degut a la senzillesa aportada per el processament de dades i desenvolupament de models de Machine Learning, Python és una bona selecció per desenvolupar el projecte. Però, per altra banda, com hem comentat anteriorment, depenent de les especificacions i requisits pot no ser una bona selecció, ja que si la velocitat de tractament de dades és una preocupació, potser Python no és la millor selecció, però pot ajudar en altras aspectes.

Proves

Les proves del nostre sistema són fonamentals, ja que les pròpies del sistema de Machine Learning no són suficients. Pel fet que per poder verificar totalment les prediccions del nostre sistema, val la pena ampliar el conjunt d'eines de la següent forma:

- **Proves diferencials**: Es comparen les mitjanes de les prediccions del model actual amb un model anterior, amb l'objectiu de detectar si els models anteriors tenen una solució més bona que la que tenim actualment, ja que és vital per trobar errades com poden ser que s'hagi esborrat un caràcter del codi sense voler. Sense aquestes proves diferencials és molt difícil no adonar-se d'aquests errors i això pot provocar un error en cascada a la llarga.

- **Proves de referència**: Aquestes proves comparen el temps que triga a entrenar i donar les prediccions d'un model a un altra. Evitant introduir addicions al codi ineficients.

- **Proves de rendiment**: Són bàsicament proves per comprovar el rendiment al qual està treballant la CPU i quin és el consum de la memòria del computador.

Implementació

A l'hora d'implementar el nostre model de Machine Learning cal decidir si s'utilitzarà una plataforma com a server (Paas) o una infraestructura com a servei (Iaas).

Una plataforma com a servei és un sistema d'implementació molt bàsic i poc complexa el qual pot ser ideal per a prototips amb poc tràfic, però com el tràfic d'aquesta comenci a augmentar caldrà adoptar més complexitat com pot ser una infraestructura com a servei.

La forma més comuna i fàcil d'implementar Machine Learning sigui exposar el model com un servei API. Ja que les APIs poden connectar-se fàcilment a aplicacions de producció.

2.2.4 Xarxes Neuronals.

Una xarxa neuronal és un tipus de model d'aprenentatge automàtic que s'inspira en el funcionament del cervell humà. Aquestes xarxes estan formades per una gran quantitat de nodes interconnectats, anomenats "neurones", que treballen junts per a fer tasques específiques.[8]

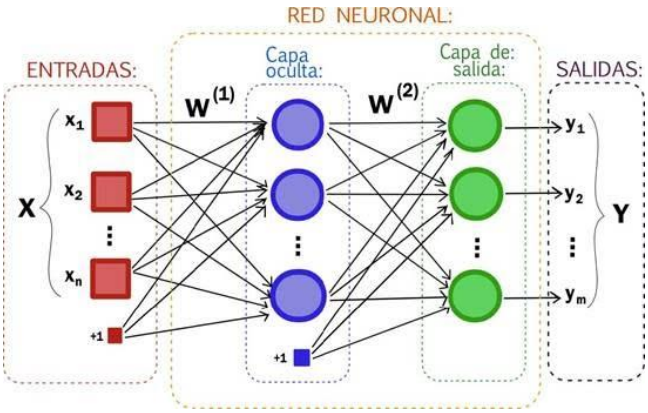


Fig 2.5. Sistema neuronal.

Font: [8]

Les xarxes neuronals van fer un gran salt quan Frank Rosenblatt va dissenyar el Perceptró [8], un tipus de classificador lineal, utilitzat per el reconeixement d'imatges a partir d'una sèrie de fotocèl·lules, potenciòmetres i motors elèctrics.

L'entrada d'un classificador lineal en qualsevol nombre de dimensions es pot expressar de la següent forma: [9]

$$f(x) = b + \sum_i wi \cdot xi \qquad (1)$$

Per tant, en la classificació la predicció està determinada per:

$$Classificació = \begin{Bmatrix} 1\ if\ f(X) > 0 \\ 0\ if\ f(X) \leq 0 \end{Bmatrix} \quad (2)$$

Per altra banda, les xarxes neuronals s'utilitzen en una àmplia varietat d'aplicacions, des del reconeixement de patrons i classificació fins al processament del llenguatge natural i la predicció.

En proporcionar una gran quantitat de dades d'entrenament, les xarxes neuronals poden "aprendre" a fer tasques complexes per si mateixes, sense necessitat de programació explícita.
Existeixen diferents tipus de xarxes neuronals, cadascuna de les quals s'adapta millor a diferents tipus de tasques i conjunts de dades. Alguns dels tipus més comuns inclouen xarxes neuronals profundes, xarxes neuronals recurrents i xarxes neuronals convolucionals.

Algunes de les equacions més comunes utilitzades en les xarxes neuronals són les següents:[10]

- Equació d'activació: aquesta equació s'utilitza per a determinar si una neurona ha de "activar-se" o no. Una vegada que s'ha activat, la neurona pot transmetre el seu senyal a altres neurones en la xarxa.

- Equació de propagació cap endavant: aquesta equació s'utilitza per a calcular la sortida d'una neurona en funció de les entrades que rep.

- Equació de retropropagació: aquesta equació s'utilitza per a actualitzar els pesos de les connexions entre neurones durant el procés d'entrenament de la xarxa.

- Equació de l'error: aquesta equació s'utilitza per a mesurar la precisió de la xarxa durant el procés d'entrenament.

- Equació del gradient descendent: aquesta equació s'utilitza per a actualitzar els pesos de les connexions entre neurones de manera que es minimitzi l'error de la xarxa.

Aquestes són només algunes de les equacions més comunes utilitzades en les xarxes neuronals. Hi ha moltes altres equacions i tècniques matemàtiques que es fan servir en el disseny i entrenament d'aquests models.

En resum, les xarxes neuronals són una eina valuosa en l'aprenentatge automàtic que s'ha utilitzat amb èxit en una àmplia varietat d'aplicacions. Encara que les xarxes neuronals són molt efectives en moltes tasques, també tenen alguns inconvenients. En particular, requereixen una gran quantitat de dades d'entrenament i poden ser difícils d'interpretar. A més, poden ser propenses a sobre ajustar, cosa que significa que poden tenir un rendiment pitjor en conjunts de dades no vistes prèviament. No obstant això, a mesura que la tecnologia avança i es desenvolupen noves tècniques d'entrenament i regularització, aquests inconvenients s'estan abordant cada vegada més.

2.2.5 Turtlebot 3 Burger.

TurtleBot 3 Burger és un robot mòbil de baix cost i de baixa grandària desenvolupada per l'empresa coreana ROBOTIS. Està dissenyat per a ser utilitzat en aplicacions d'aprenentatge i educació en robòtica. És una versió més simple i assequible del TurtleBot 3, que ve amb menys funcionalitats i components.

El TurtleBot 3 Burger està construït sobre una base de dues rodes motrius i compta amb una càmera de profunditat Intel RealSense per a la percepció del seu entorn i un controlador de moviment de dos eixos per a orientar la seva càmera. També ve amb una placa de desenvolupament ROS (Robot Operating System) anomenada OpenCR, que li permet ser programat i controlat de manera senzilla a través de ROS i un sensor de distància làser.

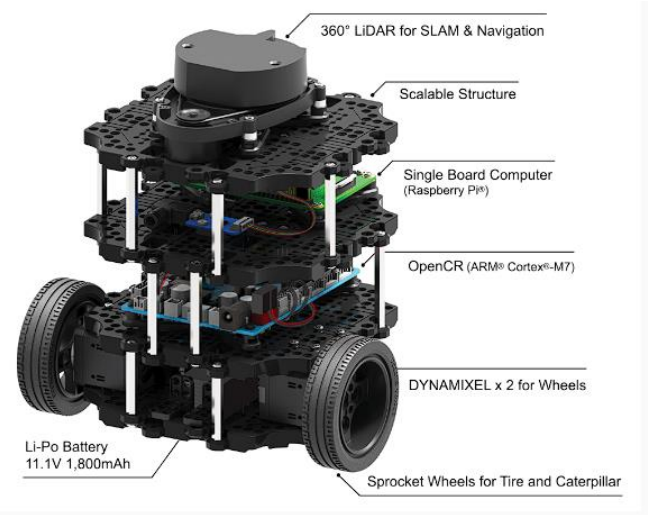


Fig 2.6. Turtlebot 3 Burger.

Font. [11]

Especificacions Turtlebot3 Burger

- Màxima velocitat de translació: 0,22 m / s

- Màx. velocitat de rotació: 162.72 ° / s

- Màx. càrrega útil: 15 kg

- Dimensions: 138 × 178 × 192 mm

- Pes (+ SBC + bateria + sensors): 995 g

- Autonomia: 2h 30m

- Temps de càrrega: 2h 30m

- Connexió per a l'ordinador: USB

- IMU: 3 eixos giroscopi, 3 eixos acceleròmetre, 3 eixos magnetòmetre

- Connectors d'alimentació:
  - 3.3V / 800 mA
  - 5V / 2A
  - 12V / 1A

- Pins d'expansió: 18 pins GPIO, 32 pins Arduino

- Diverses seqüències de sons programables

- Arduino LED x 1

- 2 botons

- Bateria de polímer de liti 11.1V 1800 mAh / 19.98 Wh 5C

- Adaptador de recàrrega:
    - Input: 100-240V, AC 50/60 Hz, 1.5A max
    - Output: 12 Vdc, 5A

Dimensions Turtlebot 3 Burger

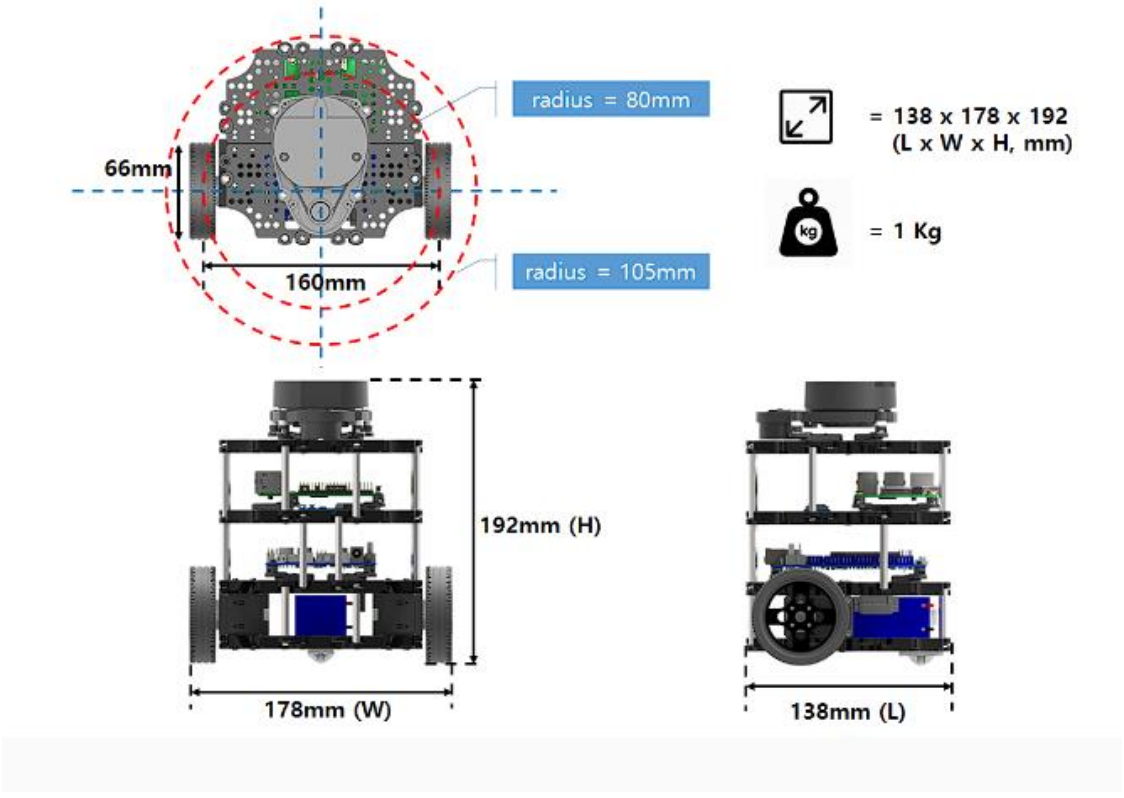


Fig 2.7. Dimensions del Turtlebot 3 Burger.

Font: [11]

2.3. Límits del projecte.

El present projecte presenta un estudi del Machine Learning utilitzant com a software ROS i treballant com a model de proves el robot mòbil Turtlebot3 Burger. Amb l'objectiu de poder dissenyar un sistema intel·ligent en el qual el robot pugui entendre el seu entorn i aprendre sobre aquest, per poder ser aplicat en un futur en un àmbit industrial.

El primer pas a realitzar serà definir l'objecte del projecte, el qual mitjançant una cerca d'informació i d'antecedents, es definiran els objectius del projecte en qüestió.

Seguidament, es determinaran quin són els objectius principals a assolir en el projecte. Quan es tinguin els objectius principals definits, es definiran una sèrie d'especificacions tècniques que s'hauran de complir per tal d'assolir els objectius prèviament marcats. Els quals són:

- Aconseguir aplicar el Machine Learning amb ROS a partir de les llibreries ja existents d'aquesta, configurant els paràmetres que nosaltres considerem per aconseguir que el robot es mogui per un entorn sense xocar-se amb res.
- Estudiar diferents escenaris ja sigui sense obstacles, amb obstacles estàtics i mòbils i analitzar el comportament del robot.

Es plantejaran una sèrie d'alternatives de solució, per definir quina és la millor alternativa a desenvolupar. Posteriorment, es realitzarà un anàlisi de viabilitat tècnica, econòmica i mediambiental de l'alternativa a desenvolupar seleccionada.

Per últim, es procedirà a realitzar la planificació del projecte i poder determinar quin serà el pressupost d'aquest.

3. Objectius de detall i especificacions tècniques.

La finalitat del present apartat és definir i exposar els diversos objectius que es pretenen assolir amb la realització d'aquest projecte. Ha sigut necessària prèviament una cerca curada del funcionament del Machine Learning i com aplicar-ho amb el software ROS, observant amb quin conjunt de llibreries i quines modificacions cal fer en els codis generats per les mateixes llibreries per poder assolir el propòsit adequadament.

Després de realitzar un anàlisi detallat, s'han obtingut uns resultats que han permès descartar aquelles idees menys favorables per a la implementació de l'aplicació del robot. A continuació, es citen els objectius definitius.

- Simular el procés d'aprenentatge del robot.

- Poder observar el seu comportament en diferents escenaris, ja sigui amb obstacles estàtics, mòbils i sense.

- Saber reconèixer els obstacles que es troben en l'entorn a partir del seu propi aprenentatge. Amb l'objectiu de podeu moure's sense topar-se amb ells.

- Aplicar-ho a un model Turtlebot3 Burger real.

Un cop determinats els objectius principals, cal determinar les especificacions tècniques corresponents.

- **Simular el procés d'aprenentatge del robot.**

    - Configurar tots els paràmetres de l'ordinador, ja sigui en termes com el sistema operatiu com és en aquest cas Ubuntu en una màquina virtual, la instal·lació de ROS i les llibreries necessàries per poder fer les simulacions.

- **Poder observar el seu comportament en diferents escenaris, ja sigui amb obstacles estàtics, mòbils i sense obstacles.**

- Determinar diferents escenaris de treball, definint si hi ha obstacles estàtics, mòbils o sense obstacles.

- **Saber reconèixer els obstacles que es troben en l'entorn a partir del seu propi aprenentatge. Amb l'objectiu de podeu moure's sense topar-se.**

  - Configurar el sensor LIDAR per poder tenir un camp de visió "real" i poder reconèixer l'entorn.

- **Aplicar-ho a un model Turtlebot3 Burger real.**

  - Configurar les connexions necessàries de l'ordinador amb el robot Turtlebot3 Burger per poder aplicar el nostre model al món real.

4. Marc conceptual.

Aquest capítol està centrat per entendre els conceptes necessaris per poder comprendre i desenvolupar el nostre sistema de Machine Learning amb ROS.

4.1 Ubuntu.

Ubuntu és un sistema operatiu basat en Linux que es distribueix com a programari lliure. Va ser creat en 2004 i és un dels sistemes operatius més populars basats en Linux. Algunes de les característiques d'Ubuntu inclouen: [12]

- Facilitat d'ús: Ubuntu ha estat dissenyat per a ser fàcil d'usar, fins i tot per a aquells que no tenen experiència amb sistemes operatius basats en Unix.

- Programari lliure: Ubuntu utilitza programari lliure i es distribueix sota la Llicència Pública General de GNU (GPL). Això significa que pots modificar i redistribuir el sistema operatiu d'acord amb les teves necessitats.

- Actualitzacions regulars: Ubuntu llança una nova versió cada sis mesos, cosa que significa que sempre estàs utilitzant l'última versió del sistema operatiu. A més, les actualitzacions de seguretat es llancen regularment per a garantir la seguretat del sistema.

- Gran quantitat de programari disponible: Ubuntu ve amb una gran quantitat de programari preinstal·lat, incloent-hi un navegador web, un client de correu electrònic, una suite d'oficina i molt més. A més, hi ha una gran quantitat de programari addicional disponible per a descarregar i instal·lar a través del Centre de Programari d'Ubuntu.

- Àmplia compatibilitat: Ubuntu és compatible amb una àmplia gamma de maquinària, cosa que significa que és possible instal·lar-ho en una gran varietat de computadores.

4.2 ROS.

ROS (Robot Operating System) és un sistema operatiu per a robots que li permet als fabricants integrar fàcilment una àmplia varietat de programari de tercers i tecnologies de sensor. ROS proporciona un marc de programació comuna i un conjunt d'eines per a ajudar els desenvolupadors a crear robots més complexos i avançats.

ROS està dissenyat per a ser modular i està compost per un conjunt de paquets de programari individual que es poden utilitzar de manera independent o combinats per a crear sistemes de robot més complexos. Cada paquet proporciona una funció específica, com la navegació, el processament d'imatges o la manipulació de braços robòtics.

Es basa en el llenguatge de programació Python i s'utilitza principalment en sistemes operatius basats en Unix, com Ubuntu. Encara que s'ha desenvolupat principalment per a robots mòbils, també s'ha utilitzat en una àmplia varietat d'altres tipus de robots, com a robots industrials, robots mèdics i robots de recerca, etc.

4.2.1 Conceptes bàsics de ROS

Alguns conceptes que s'han de saber sobre ROS són els següents:[13]

- **Node**: Un node és un programa individual que s'executa en ROS. Cada node fa una tasca específica, com rebre dades d'un sensor, processar aquestes dades o enviar comandos a un actuador. Un node té un nom únic en el sistema, ens permet connectar-se amb la resta de nodes utilitzant noms sense ambigüitat. Un node pot ser escrit usant diferents llibreries com: roscpp i rospy; roscpp és per C ++ i rospy és per Python.

  ROS ens permet a gestionar els nodes i donar-nos informació sobre ells a partir dels diferents comandos.

  - **rosnode:** Mostra informació sobre els nodes de ROS que s'estan executant. Amb el comando *rosnode list* és llista tots els nodes que estan actius.

- **rosrun:** Permet usar el nombre del paquet per executar directament un node
- **rosnode kill node:** Elimina el node.
- **rosnode ping node:** Mostra la connectivitat amb el node.
- **Rosnode cleanup:** Neteja la informació que hi ha dins d'un node.

- **Topic**: Els nodes es comuniquen entre si a través de tòpics. Un tòpic és un flux de missatges que pot ser publicat o subscrit per un node. Els nodes que publiquen missatges en un tòpic són coneguts com a publicadors, mentre que els nodes que subscriuen un tòpic són coneguts com a subscriptors.

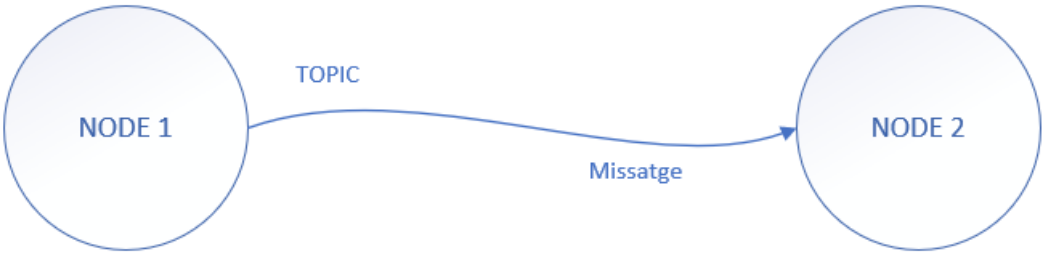


Fig 4.1. Funcionament de dos nodes connectats.
Font: Elaboració pròpia.

- **Paquet**: Un paquet és un conjunt de codi, biblioteques i arxius de configuració relacionats que s'utilitzen per a fer una tasca específica en ROS. Els paquets s'utilitzen per a organitzar el codi i fer que sigui més fàcil de reutilitzar i compartir.

- **Màster**: El Màster és un procés central que s'executa en ROS i s'encarrega de la gestió dels nodes i tòpics. El Màster manté un registre de tots els nodes i tòpics en execució i s'encarrega de la comunicació entre ells.

- **Gazebo**: Gazebo és un simulador de robots que s'utilitza sovint amb ROS. Gazebo permet als desenvolupadors provar i desenvolupar codi de robot en un entorn virtual abans de passar a la implementació en un robot físic.

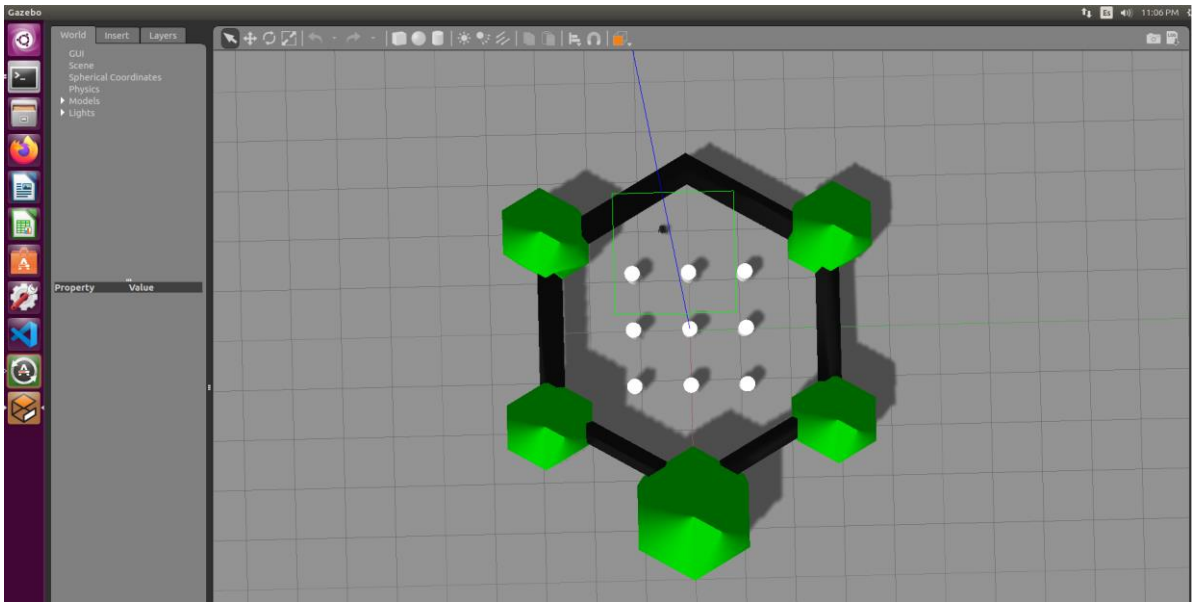


Fig 4.2. Demostració de Gazebo en Ubuntu.

Font: Elaboració propia.

4.3 TensorFlow.

Fig 4.3. Logo TensorFlow.

TensorFlow és una plataforma de codi obert per a l'aprenentatge automàtic i l'anàlisi de dades. Va ser desenvolupat originalment per Google i ha estat alliberat sota la llicència Apatxe 2.0. TensorFlow s'utilitza àmpliament en la indústria i en la recerca acadèmica per a implementar i entrenar models d'aprenentatge automàtic, així com per a realitzar anàlisi de dades i prediccions.

Algunes de les característiques de TensorFlow inclouen:

- Suport per a una àmplia gamma de plataformes: TensorFlow s'executa en moltes plataformes, incloent-hi Windows, MacOS i sistemes operatius basats en Unix, com Linux i Ubuntu. També es pot executar en dispositius mòbils i en el núvol.

- Llenguatges de programació: TensorFlow suporta molts llenguatges de programació, incloent-hi Python, C++ i R. Això significa que pots utilitzar el llenguatge de programació que prefereixis per a treballar amb TensorFlow.

- Alta escalabilitat: TensorFlow és capaç d'executar còmput en paral·lel a través de molts nuclis de processador i en màquines amb diverses GPU. Això fa que sigui ideal per a l'entrenament de models d'aprenentatge automàtic de gran escala.

- Àmplia comunitat i suport: TensorFlow té una àmplia comunitat d'usuaris i desenvolupadors on s'hi troba molta documentació i recursos disponibles en línia.

4.4 Keras.

Fig 4.4. Logo Keras.

Keras és un software de codi obert per el programari d'aprenentatge profund escrit en Python. Va ser desenvolupat per a permetre als investigadors d'aprenentatge profund a implementar i avaluar fàcilment diferents models de xarxes neuronals.[14]
Algunes de les principals característiques de Keras són:

- Facilita el procés d'implementació i experimentació amb xarxes neuronals, ja que proporciona una interfície senzilla i fàcil d'usar per a definir i entrenar models.

- Compatible amb diferents plataformes de back-end, com TensorFlow, Theano i CNTK. Això significa que pot utilitzar Keras en una àmplia varietat de sistemes, incloent CPUs, GPUs i sistemes en el núvol.

- Ofereix una gran quantitat de funcions preconstruidas per a afegir capes i fer tasques comunes d'aprenentatge profund, com la classificació, la regressió i la predicció de seqüències.

- Proporciona suport per a xarxes neuronals de diverses capes i tipus, incloent-hi xarxes feedforward, xarxes recurrents i xarxes d'atenció.

4.5 Anaconda.

Fig 4.5. Logo Anaconda.

Anaconda és una distribució de Python i R que inclou més de 1.500 paquets de ciència de dades, com NumPy, Colles i Matplotlib. Va ser dissenyada per a facilitar la gestió de paquets i el desplegament d'aplicacions de ciència de dades, i és molt popular entre científics de dades i analistes de dades.[15]

Algunes de les principals característiques d'Anaconda són:

- Inclou un instal·lador que fa que sigui fàcil instal·lar Anaconda i tots els seus paquets en un sol pas.

- Proporciona una eina anomenada "Conda" que permet instal·lar, actualitzar i desinstal·lar paquets de manera fàcil i ràpida.

- Ve amb una aplicació anomenada "Anaconda Navigator" que proporciona una interfície gràfica per a gestionar els paquets i llançar aplicacions.

- Ofereix suport per a múltiples plataformes, incloent-hi Windows, macOS i Linux.

5. Generació i plantejament de possibles alternatives de solució.

Un cop descrit els diferents mètodes per implementar el model amb els seus objectius i especificacions tècniques respectives, el següent pas consisteix a proposar un seguit d'alternatives de solució, les quals ens permeti triar la solució més idònia.

L'entorn amb el qual treballarem serà ROS, ja que el Turtlebot 3 Burger està pensat per treballar-hi amb aquest sistema. Pel fet que permet programar els nodes que enllacen els diferents components del Turtlebot 3 Burger.

Primer de tot cal definir la distribució de ROS amb la qual treballarem, ja que contempla un conjunt de versions. Cada distribució permet als desenvolupadors treballar amb un codi base relativament estable fins que estiguin llestos per poder encaminar tot.

Depenent de les necessitats de cada robot és selecciona una distribució o un altre. Les més recomanades per la mateixa companyia són ROS Noetic, ROS Melodic i ROS Kinetic. Les quals destaquen per les seves altes capacitats de desenvolupament dins de l'entorn del Turtlebot 3 Burger, per tant, s'haurà de fer un estudi sobre la selecció de la distribució que millor encaixi amb les nostres necessitats.

Seguidament, caldrà determinar quin model d'aprenentatge s'utilitzarà per desenvolupar el nostre model de Machine Learning.

Hem estudiat l'aprenentatge supervisat el qual es tracta d'un model que és entrenat amb un conjunt de dades d'entrada conegudes prèviament per a l'obtenció de resultats. Per poder aplicar-ho hi ha dos mètodes, la regressió i la classificació tal com hem vist anteriorment.

Per altra banda, tenim l'aprenentatge no supervisat en el qual hi ha dades sense etiquetar i el mateix algorisme les ha d'intentar entendre per si mateix a diferència del supervisat en el qual les dades ja van etiquetades per facilitar l'aprenentatge. Per poder aplicar-ho hi ha dos mètodes, l'agrupament i la reducció dimensional tal com hem vist anteriorment.

I per últim hem vist l'aprenentatge reforçat en el qual per la construcció de models basats en aquest tipus d'aprenentatge es prenen com a referència el resultat de cada interacció i

s’utilitza una recompensa com a paràmetre. És a dir, quan la màquina encerta, se li recompensa positivament, de tal forma que recorda si ho fa bé tindrà una recompensa. I en el cas contrari, si ho fa malament se li recompensa negativament per tal que la màquina recordi que l’acció feta no és el correcte. Hi ha varies formes d’aplicar l’aprenentatge reforçat com poden ser Q-Learning, DQN, Deep-DQN, Duel-DDQN i Actor-Critic.

Per tant, un cop analitzat totes les possibles alternatives de solució, caldrà determinar quina de totes aquestes és la més òptima en la realització d’aquest projecte.

6. Selecció de l'alternativa més adequada

Un cop plantejades les alternatives de solució en l'apartat anterior, s'ha d'escollir quina de les propostes serà la més idònia. Per poder decidir quina de les solucions proposades serà la que es desenvolupi, s'ha fet una reflexió basada en la facilitat d'implementació del model plantejat, estudiant quina alternativa disposa més recursos per posar-la en marxa.

Per tant, després d'analitzar les diferents versions de ROS i els recursos que disposa, s'ha decidit dur a terme el projecte de Machine Learning amb el robot Turtlebot 3 Burger a través de ROS Kinetic. S'ha cregut que aquesta versió de ROS és la que millor encaixa amb les necessitats requerides per dur a terme el projecte, ja que està preparat per poder treballar amb el marc conceptual descrit anteriorment aplicat amb el robot Turtlebot 3 Burger, ja que disposa de llibreries per poder fer un mapatge de l'entorn on es troba el robot, permetent fer simulacions on es coneix la zona de treball d'aquest, les seves limitacions de l'entorn i com és desenvolupa veient la posició exacta del robot.

Per altra banda, alhora de determinar quin model d'aprenentatge s'utilitzarà, després d'estudiar els diferents models, s'ha arribat a la conclusió de que el que millor encaixa amb les nostres necessitats serà l'aprenentatge reforçat aplicant el mètode d'agent DQN, ja que l'objectiu no és entrar-li diferents entrades i que la mateixa màquina vagi aprenent sobre elles, classificant-les, agrupant-les, etc. El que volem és aconseguir que la màquina observi el seu entorn i vagi aprenent sobre aquest. I per aquest motiu es pensa que si la màquina va aprenent en funció de recompenses positives i negatives cada cop que fa una decisió entenent què és correcte i que incorrecte, arribarà el moment en el qual sabrà cap a on anar i cap a on no.

7. Anàlisi de viabilitat

L'estudi de viabilitat és essencial per poder garantir si és factible i rendible desenvolupar el projecte atenent a les seves característiques tecnològiques. Per arribar a entendre si el projecte arribarà a ser factible, cal primer assegurar que el projecte es realitzable tècnicament i tenir en compte l'impacte ambiental que genera, per a incorporar modificacions al projecte a fi d'assegurar el compliment de la normativa. Per tant, la viabilitat del projecte s'analitza a nivell tècnic, ambiental i econòmic.

Un cop fet aquest anàlisi s'arriba a la conclusió de si la inversió del projecte val o no la pena.

7.1. Viabilitat Tècnica.

La viabilitat tècnica és la condició que determina si és possible el funcionament i desenvolupament del projecte atenent a les seves característiques tecnològiques.

Com es va comentar en l'apartat 6, en la realització del projecte es treballarà principalment amb l'entorn de ROS Kinetic. Com ROS treballa millor en el sistema operatiu d'Ubuntu, s'haurà de dur a terme la instal·lació de Linux Ubuntu al PC amb el qual es treballi.

Seguidament, un cop instal·lat Ubuntu a partir d'una màquina virtual, s'haurà de preparar l'entorn de ROS amb la versió Kinetic, instal·lant els paquets i llibreries necessàries per poder fer la posada en marxa del robot Turtlebot 3 Burger.

I per últim, un cop es tingui a punt els apartats anteriors, es realitzarà a instal·lar els softwares necessaris per poder desenvolupar el model de Machine Learning, instal·lant Keras, TensorFlow i Anaconda.

Per acabar es farà una prova a nivell d'exemple per comprovar que tots els paquets han estat configurats correctament.

7.2. Viabilitat Mediambiental.

El present projecte no consta d'un gran impacte mediambiental, ja que en tractar-se d'un projecte de caràcter educatiu, és a dir que no anirà destinat a la planta de producció de cap indústria, serà purament un model teòric on en un futur si es volgués si es podria aplicar en aplicacions reals.

Per altra banda, podem veure un estudi exhaustiu sobre la viabilitat mediambiental d'aquest projecte en l'*Annex I* on es mostra en detall diferents aspectes que poden afectar el medi ambient.

Com a conclusió, observem que el nivell d'impacte que pot causar és pràcticament inexistent.

7.3. Viabilitat Econòmica.

L'estudi de la viabilitat econòmica en aquest projecte, no s'ha desenvolupat com a producte comercial, sinó com a projecte de recerca i desenvolupament destinat cap a la universitat Tecnocampus ubicada a Mataró (Barcelona).

Al ser un projecte sobre l'estudi del Machine Learning a partir de ROS aplicat a un robot Turltebot3 Burger, es fa l'estudi econòmic del material necessari per poder abordar el cost d'inversió principal del projecte.

Material	**Cost**
Lenovo IdeaPad Gaming 3 15ACH6 AMD Ryzen 7 5800H/16GB/512GB SSD/ RTX 3050Ti/15.6"	**999,00€**
TurtleBot 3 Burger	**592,00€**
COST TOTAL	**1.591,00€**

Taula 7.1. Cost d'Inversió inicial

Font: Elaboració Pròpia

8. Planificació.

S'ha realitzat una planificació del projecte on es podrà observar en el següent punt la distribució d'aquestes tenint en compte que l'inici és el dia 7 d'octubre de 2022 i finalitzarà el dia 15 de juny de 2023.

8.1. Tasques principals

La totalitat d'aquest projecte ha estat dividida en 15 activitats principals. Degut a la grandària d'alguna d'aquestes activitats, s'han dividit en subactivitats.

Un cop realitzar amb el programa MS-Project, es podrà veure a partir de les prerelacions descrites [veure annex I] on es marquen en vermell les activitats crítiques en el diagrama de Gantt.

El projecte consta de les següents activitats:

Nom de la Tasca	Duració	Comença	Finalitza
Inici	0 hores	vie 07/10/22	vie 07/10/22
Identificar el problema	**45 hores**	**vie 07/10/22**	**jue 27/10/22**
Objecte del Problema	5 hores	vie 07/10/22	lun 10/10/22
Revisió d'antecedents	25 hores	lun 10/10/22	jue 20/10/22
Recerca d'informació	17 hores	lun 10/10/22	mar 18/10/22
Abast del Projecte	5 hores	vie 21/10/22	lun 24/10/22
Objectiu i especificacions tècniques	10 hores	lun 24/10/22	jue 27/10/22
Selecció de l'alternativa més adequada	**31 hores**	**vie 28/10/22**	**vie 11/11/22**
Cerca d'alternatives de solució	17 hores	vie 28/10/22	vie 04/11/22
Estudi de les diferents alternatives	10 hores	vie 04/11/22	mié 09/11/22
Selecció de l'alternativa més adequada	4 hores	jue 10/11/22	vie 11/11/22
Anàlisi de Viabilitat	**20 hores**	**vie 11/11/22**	**lun 21/11/22**

Anàlisi de viabilitat tècnica	6 hores	vie 11/11/22	mar 15/11/22
Anàlisi de viabilitat mediambiental	8 hores	mar 15/11/22	jue 17/11/22
Anàlisi de viabilitat econòmica	6 hores	vie 18/11/22	lun 21/11/22
Planificació	**16 hores**	**mar 22/11/22**	**mar 29/11/22**
Identificació de tasques	4 hores	mar 22/11/22	mié 23/11/22
Establir duració de tasques	2 hores	mié 23/11/22	mié 23/11/22
Establir tasques predecessores	2 hores	jue 24/11/22	jue 24/11/22
Elaboració de la planificació en MS-Project	8 hores	jue 24/11/22	mar 29/11/22
Entrega Avantprojecte	0 hores	mar 29/11/22	mar 29/11/22
Instal·lació dels diferents Softwares	**14 hores**	**mar 29/11/22**	**lun 05/12/22**
Instal·lació d'Ubuntu al Pc	4 hores	mar 29/11/22	mié 30/11/22
Instal·lació de ROS al Pc	2 hores	mié 30/11/22	jue 01/12/22
Instal·lació paquets Turtlebot 3	2 hores	jue 01/12/22	jue 01/12/22
Instal·lació de Tensorflow al Pc	2 hores	vie 02/12/22	vie 02/12/22
Instal·lació d'anaconda al PC	2 hores	vie 02/12/22	lun 05/12/22
Instal·lació de Keras al Pc	2 hores	lun 05/12/22	lun 05/12/22
Preparació del Turtlebot 3 Burger	**8 hores**	**mar 06/12/22**	**jue 08/12/22**
Preparació dels paràmetres del robot	5 hores	mar 06/12/22	mié 07/12/22
Prepació del làser Lidar	3 hores	mié 07/12/22	jue 08/12/22
Preparació del model de Machine Learning	**30 hores**	**jue 08/12/22**	**jue 22/12/22**
Paquets de Machine Learning a ROS	1 hora	jue 08/12/22	jue 08/12/22
Disseny del sistema de recompenses	12 hores	vie 09/12/22	mié 14/12/22
Disseny del model d'aprenentatge reforçat	17 hores	jue 15/12/22	jue 22/12/22
Disseny dels diferents escenaris	**60 hores**	**jue 22/12/22**	**jue 19/01/23**
Disseny del escenari sense obstacles	11 hores	jue 22/12/22	mié 28/12/22
Disseny del escenari amb obstacles estàtics	14 hores	mié 28/12/22	mar 03/01/23

Disseny del escenari amb obstacles mòbils	16 hores	mié 04/01/23	mié 11/01/23
Disseny del escenari amb obstacles mòbils i estàtics	19 hores	mié 11/01/23	jue 19/01/23
Simulacions	**114 hores**	**jue 19/01/23**	**mar 14/03/23**
Simulació del primer escenari	17 hores	jue 19/01/23	vie 27/01/23
Simulació del segon escenari	27 hores	vie 27/01/23	jue 09/02/23
Simulació del tercer escenari	30 hores	jue 09/02/23	jue 23/02/23
Simulació del quart escenari	40 hores	jue 23/02/23	mar 14/03/23
Lliurament memòria intermedia	0 hores	mar 14/03/23	mar 14/03/23
Model Real	**72 hores**	**mar 14/03/23**	**jue 30/03/23**
Guardar el model entrenat	50 hores	mar 14/03/23	vie 17/03/23
Dissenyar el escenari real	10 hores	lun 20/03/23	vie 24/03/23
Comprovació del entrenament en un robot real	12 hores	lun 27/03/23	jue 30/03/23
Conclusions	**6 hores**	**vie 31/03/23**	**lun 03/04/23**
Observacions dels resultats obtinguts	4 hores	vie 31/03/23	lun 03/04/23
Linies futures de treball	2 hores	lun 03/04/23	lun 03/04/23
Lliuremament memòria final	0 hores	lun 03/04/23	lun 03/04/23
Hores total del projecte	**416 hores**		

Taula 8.1. Activitats del projecte

Font: Elaboració pròpia

8.2. Cost tasques

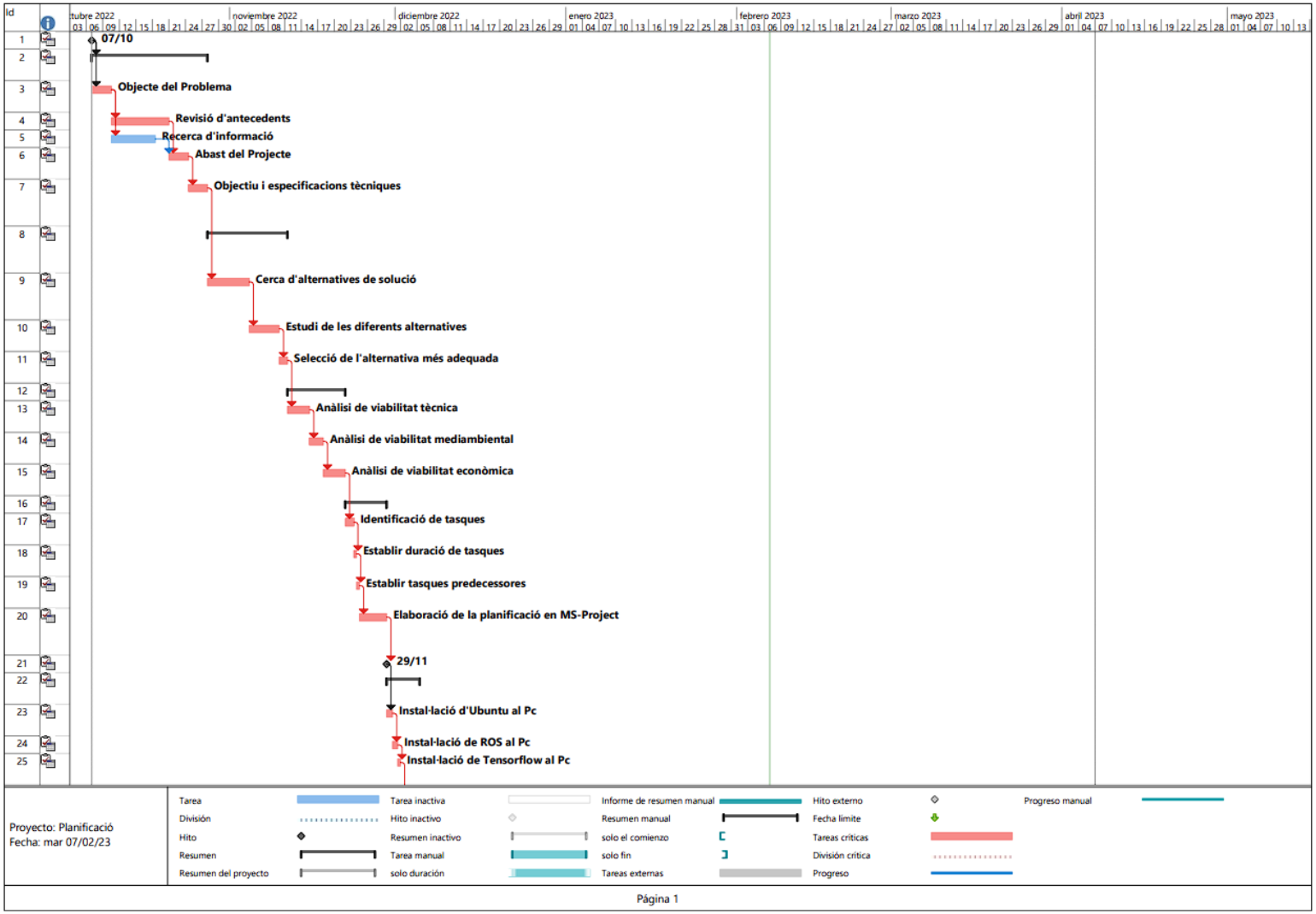
Cal destacar el cost de cada tasca, per tal de poder fer més endavant el pressupost del projecte, és per això que el cost/hora és de 40 €/hora. Un cop especificat el cost del recurs és pot imputar a cadascuna de les tasques establertes per tal de determinar el seu preu segons la durada i el recurs.

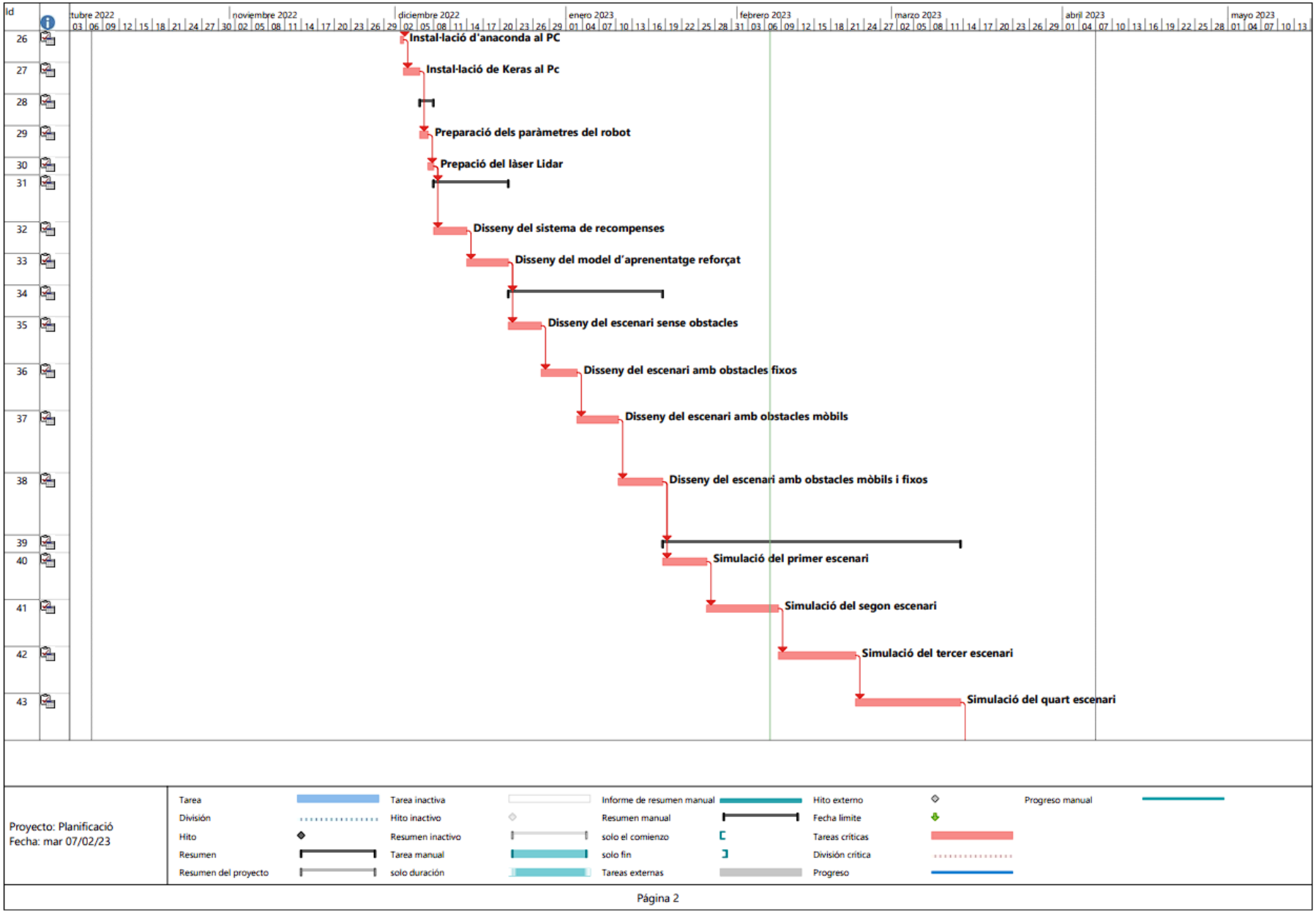
Nom de la Tasca	Cost activitat
1. Identificar el Problema	2.480,00 €
2. Selecció de l'alternativa més adequada	1.240,00 €
3. Anàlisi de Viabilitat	800,00 €
4. Planificació	640,00 €
5. Instal·lació dels diferents softwares	560,00 €
6. Preparació del Turtlebot 3 Burger	320,00 €
7. Preparació del model de Machine Learning	1.200,00 €
8. Disseny dels diferents escenaris	2.400,00 €
9. Simulacions	4.560,00 €
10. Model real	1.480,00 €
11. Conclusions	240,00 €
Cost total del projecte	**15.920,00 €**

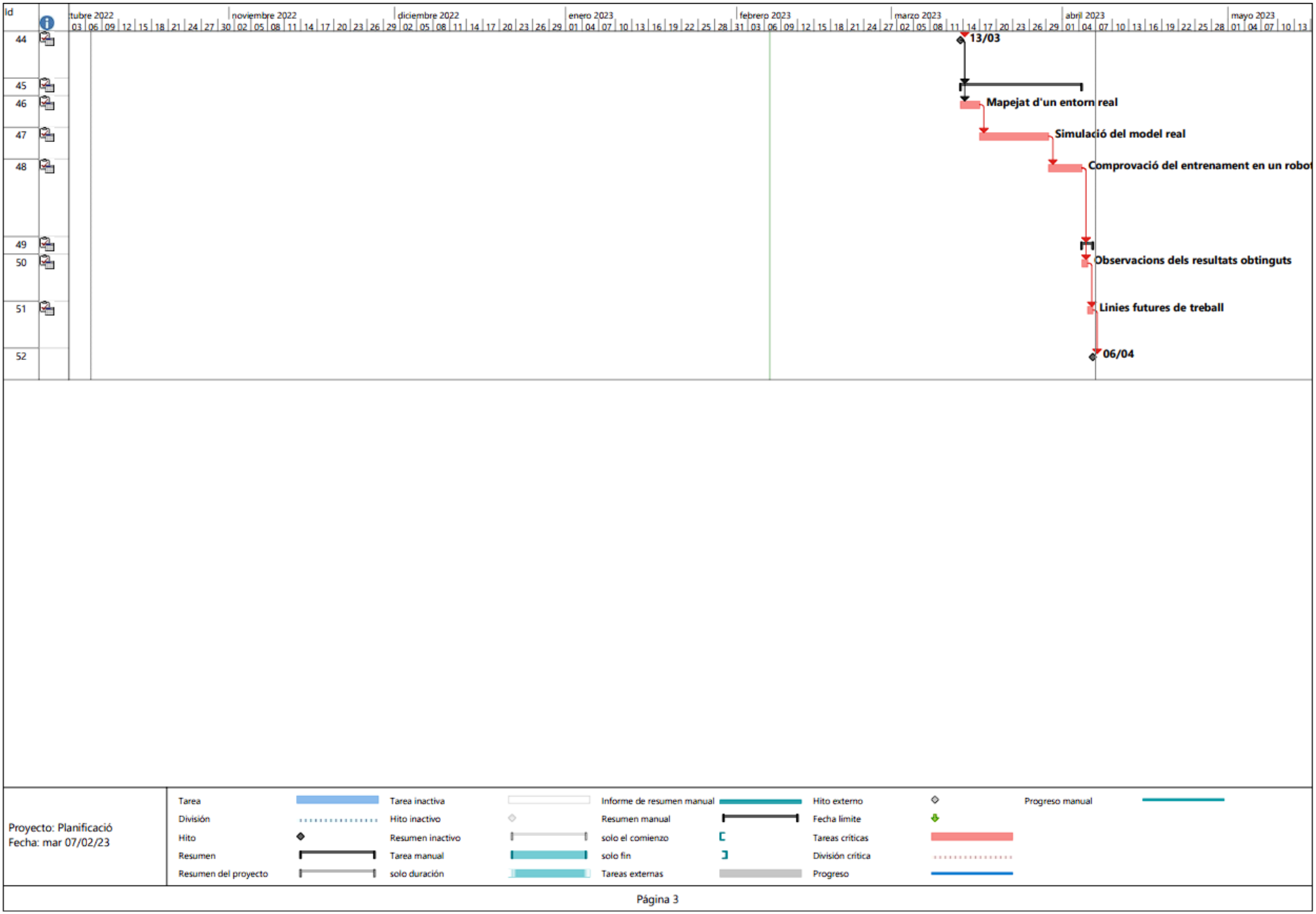
Taula 8.2. Cost activitats

Font: Elaboració pròpia.

8.2. Diagrama de Gantt




Id
tubre 2022
noviembre 2022
diciembre 2022
enero 2023
febrero 2023
marzo 2023
abril 2023
mayo 2023
26 Instal·lació d'anaconda al PC
27 Instal·lació de Keras al Pc
28
29 Preparació dels paràmetres del robot
30 Prepació del làser Lidar
31
32 Disseny del sistema de recompenses
33 Disseny del model d'aprenentatge reforçat
34
35 Disseny del escenari sense obstacles
36 Disseny del escenari amb obstacles fixos
37 Disseny del escenari amb obstacles mòbils
38 Disseny del escenari amb obstacles mòbils i fixos
39
40 Simulació del primer escenari
41 Simulació del segon escenari
42 Simulació del tercer escenari
43 Simulació del quart escenari
Proyecto: Planificació
Fecha: mar 07/02/23
Tarea
División
Hito
Resumen
Resumen del proyecto
Tarea inactiva
Hito inactivo
Resumen inactivo
Tarea manual
solo duración
Informe de resumen manual
Resumen manual
solo el comienzo
solo fin
Tareas externas
Hito externo
Fecha límite
Tareas críticas
División crítica
Progreso
Progreso manual


Id
tubre 2022
noviembre 2022
diciembre 2022
enero 2023
febrero 2023
marzo 2023
abril 2023
mayo 2023
44
45
46
47
48
49
50
51
52
13/03
Mapejat d'un entorn real
Simulació del model real
Comprovació del entrenament en un robot
Observacions dels resultats obtinguts
Linies futures de treball
06/04
Proyecto: Planificació
Fecha: mar 07/02/23
Tarea
División
Hito
Resumen
Resumen del proyecto
Tarea inactiva
Hito inactivo
Resumen inactivo
Tarea manual
solo duración
Informe de resumen manual
Resumen manual
solo el comienzo
solo fin
Tareas externas
Hito externo
Fecha límite
Tareas críticas
División crítica
Progreso
Progreso manual
Página 3

9. Instal·lació dels diferents softwares

Per poder fer la realització del present projecte s'ha necessitat instal·lar l'entorn de treball necessari per poder treballar amb el robot TurtleBot3 i aplicar-li un model de Machine Learning. Per tant, s'han seguit una sèrie de passos per poder instal·lar l'entorn de treball corresponent.

9.1 Instal·lació d'Ubuntu al Pc

La instal·lació d'Ubuntu és fàcil i ràpid, el seu instal·lador és bastant intuïtiu i només cal seguir les instruccions que indica, a més a més es un sistema operatiu bastant fiable, accessible i gratuït.

Si per defecte ja es té instal·lat Windows i no volem canviar de sistema operatiu, una opció és accedir-hi a Ubuntu a partir d'una màquina virtual, tal i com mostraré en els següents passos.

Primer de tot, com a màquina virtual utilitzarem Oracle VM VirtualMachine, un potent producte de virtualització x86 compatible amb AMD64/Intel64. Per altre banda és pot executar en hosts com Windows, Linux, macOS i Solaris. Per tant accedim a la seva pàgina web [16] i descarreguem l'arxiu.

Fig 9.1 Pàgina Web de Oracle VM

Font: Elaboració pròpia

Un cop instal·lat, seguim els passos que ens indica el instal·lador i ja el tindrem configurat.

Seguidament, descarreguem la imatge de la versió 16.04 d'Ubuntu des de la pàgina web oficial [17].

Per poder instal·lar Ubuntu 16.04 és necessari disposar com a mínim de 5GB. Seguidament obrim el Oracle VM VirtualMachine i anem a la pestanya superior on posa "Nueva", escrivim el nom d'Ubuntu perquè la màquina interpreti que estem treballant amb un arxiu de tipus Linux i afegim una memòria de 4 Gb aproximadament tal i com es pot observar a continuació.

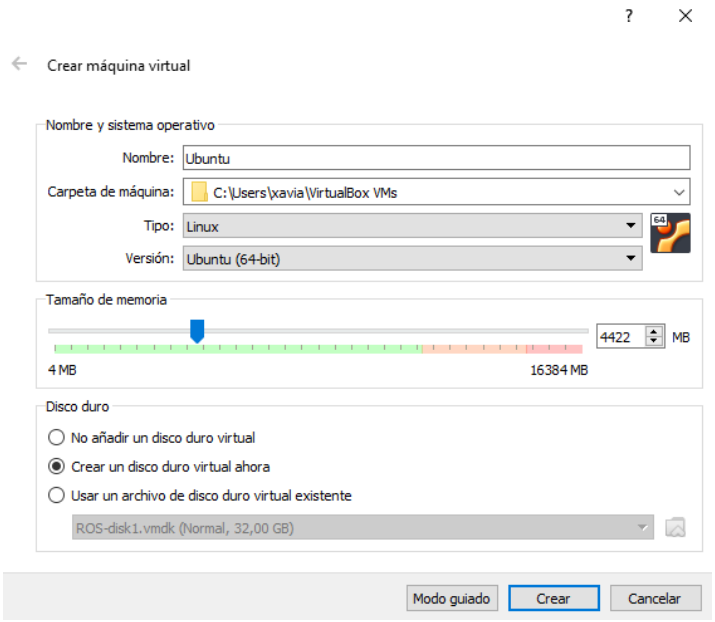


Fig 9.2 Configuració d'Ubuntu a la màquina virtual I.

Font: Elaboració pròpia.

Li donem a "*crear*" i se'ns apareixerà una finestra per configurar el tamany del disc amb el que treballarem, el deixem tal i com és mostra a continuació.

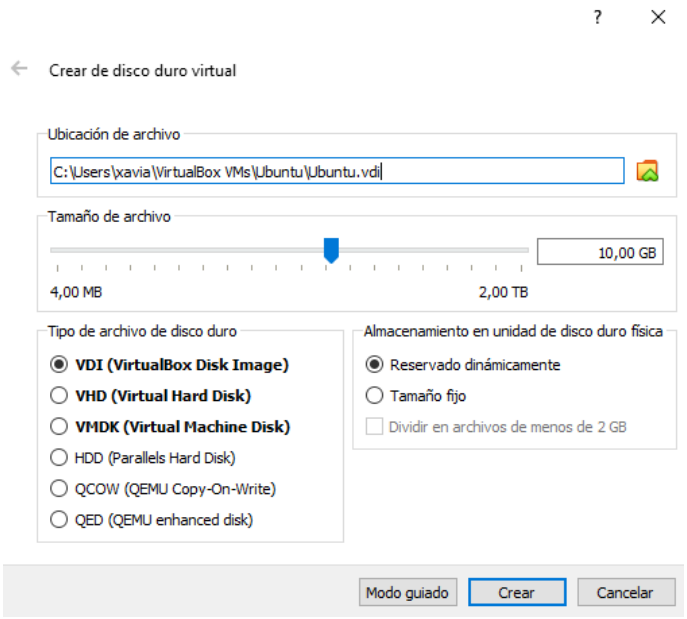


Fig 9.3 Configuració d'Ubuntu a la màquina virtual II.

Font: Elaboració pròpia.

Creem el nostre disc i fem la primera arrencada, on se'ns mostrarà una finestra per seleccionar el disc amb la versió d'Ubuntu que hem descarregat anteriorment. Li donem a "añadir" i seleccionem el disc descarregat.

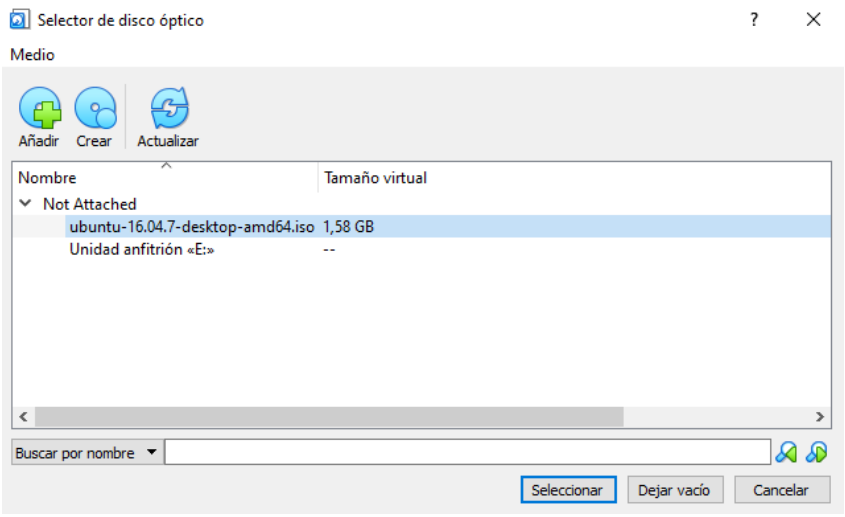


Fig 9.4 Configuració d'Ubuntu a la màquina virtual III.

Font: Elaboració pròpia.

A partir d'aquest pas la instal·lació és trivial, només s'han de seguir els passos fins a arribar a tenir Ubuntu instal·lat a l'ordinador.

Fig 9.5 Escriptori d'Ubuntu.

Font: Elaboració pròpia.

9.2 Instal·lació de ROS al Pc

La instal·lació de ROS és bastant fàcil i intuïtiva, ja que la pròpia pàgina web mostra els passos a seguir que s'han de fer depenent del paquet seleccionat. En aquest cas, com hem comentat anteriorment, farem la instal·lació per el paquet Kinetic de ROS.

Instruccions per instal·lar ROS

Primer de tot, obrirem el terminal d'Ubuntu i introduirem els següents comandos per instal·lar ROS al Pc.

```
$ sudo apt-get update
$ sudo apt-get upgrade
$            wget            https://raw.githubusercontent.com/ROBOTIS-
GIT/robotis_tools/master/install_ros_kinetic.sh
$ chmod 755 ./install_ros_kinetic.sh
$ bash ./install_ros_kinetic.sh
```

Amb aquestes instruccions instal·lem directament ROS al sistema operatiu d'Ubuntu.

Instal·lem els paquets dependents de ROS

Obrim un altre terminal d'Ubuntu i introduïm les següents instruccions per instal·lar els paquets dependents.

```
$ sudo apt-get install ros-kinetic-joy ros-kinetic-teleop-twist-joy \
  ros-kinetic-teleop-twist-keyboard ros-kinetic-laser-proc \
  ros-kinetic-rgbd-launch ros-kinetic-depthimage-to-laserscan \
  ros-kinetic-rosserial-arduino ros-kinetic-rosserial-python \
  ros-kinetic-rosserial-server ros-kinetic-rosserial-client \
  ros-kinetic-rosserial-msgs ros-kinetic-amcl ros-kinetic-map-server \
  ros-kinetic-move-base ros-kinetic-urdf ros-kinetic-xacro \
  ros-kinetic-compressed-image-transport ros-kinetic-rqt* \
  ros-kinetic-gmapping ros-kinetic-navigation ros-kinetic-interactive-
markers
```

I per altre banda, també cal instal·lar msgpack ja que és un paquet necessari per treballar amb els diferents softwares.

```
$ pip install msgpack argparse
```

Amb aquestes instruccions ja tenim ROS instal·lat al sistema operatiu d'Ubuntu amb els paquets dependents necessaris.

9.3 Instal·lar paquets TurtleBot3

Un cop instal·lat Ubuntu i ROS, s'ha d'instal·lar els diferents paquets del Turtlebot 3 i definir que estem treballant amb el model Burger.

Primer de tot instal·lem Turtlebot 3, obrim el terminal d'Ubuntu i escrivim les següents instruccions.

```
$ sudo apt-get install ros-kinetic-dynamixel-sdk
$ sudo apt-get install ros-kinetic-turtlebot3-msgs
$ sudo apt-get install ros-kinetic-turtlebot3
```

Per alter banda, es defineix que s'està treballant amb el model Burger del Turtlebot 3.

```
$ echo "export TURTLEBOT3_MODEL=burger" >> ~/.bashrc
```

Si la instrucció es completa sense errors, la preparació del TurtleBot3 Burger és considera finalitzada.

9.4 Instal·lació dels paquets necessaris

Per poder interconnectar el nostre model del Turtlebot3 Burger amb el sistema de Machine Learning cal tenir instal·lat una sèrie de paquets. Com seran en el nostre cas TensorFlow, Keras i Anaconda. Cada paquet el que fa és crear un sistema de connexió per poder interconnectar les API's de Keras en TensorFlow gràcies al software d'Anaconda.

9.4.1 Instal·lació de TensorFlow al Pc

El procés d'instal·lació de TensorFlow en el nostre sistema operatiu d'Ubuntu és gaire senzill gràcies a ROS, on la pròpia pàgina web ens mostra un exemple de com instal·lar-ho amb una única instrucció.

Per altre banda, aquest tutorial usa Python 2.7 (només CPU), en algun cas que es treballes amb un altre versió de python s'hauria d'anar a la pàgina web de TensorFlow.

La instrucció que s'ha de posar en el terminal d'Ubuntu és el següent:

```
$ pip install --ignore-installed --upgrade
https://storage.googleapis.com/tensorflow/linux/cpu/tensorflow-1.8.0-
cp27-none-linux_x86_64.whl
```

Un cop executada la instrucció, si no surt cap missatge d'error, TensorFlow s'haurà instal·lat correctament.

9.4.2 Instal·lació de Keras al Pc

Necessitarem instal·lar Keras per poder afegir les APIs de xarxes neuronals avançades escrites en Python per poder aplicar-les en TensorFlow.

Per fer això, ROS ens proporciona les instruccions necessàries per realitzar una correcta instal·lació d'aquest en el sistema operatiu d'Ubuntu.

La instrucció per instal·lar Keras és la següent:

```
$ pip install keras==2.1.5
```

És molt comú que surti un missatge d'error en relació amb la incompatibilitat amb *"tensorboard"*, un mètode funcional per solucionar aquest problema consisteix en forçar a instal·lar tensorboard amb la següent expressió.

```
$ pip install tensorboard
```

Ara sí, un cop realitzat la instal·lació i no surt cap error podem donar per instal·lat correctament Keras en Ubuntu, per tant ja podem utilitzar les APIs de xarxes neuronals en Tensorflow per dissenyar un model de Machine Learning. Per últim, caldrà instal·lar el mètode de poder unificar Keras amb TensorFlow.

9.4.3 Instal·lació d'Anaconda al Pc

Com s'ha comentat anteriorment, treballem amb la versió de Python 2.7, per tant, descarregarem la versió d'Anaconda 5.2.0 la qual es compatible amb la versió de Python desitjada.

Primer de tot, per descarregar Anaconda haurem de descarregar l'arxiu SH (Shell Script) des de la pàgina web de ROS. Un cop descarregat l'arxiu, obrim el terminal d'Ubuntu i escrivim la següent instrucció per accedir al directori on s'ha descarregat i executar-lo.

```
$ bash Anaconda2-5.2.0-Linux-x86_64.sh
```

S'accepten tots els termes de llicencia a partir del terminal i ja s'hauria instal·lat Anaconda.

Un cop instal·lada, s'haurà descriure la següent instrucció.

```
$ source ~/.bashrc
$ python -V
```

Si la instal·lació s'ha executat correctament, hauria de sortir en el terminal " Python 2.7xx :: Anaconda, Inc. "

Per altre banda, per poder utilitzar ROS i Anaconda en conjunt, s'ha d'afegir un altre paquet dependència de ROS.

```
$ pip install -U rosinstall empy defusedxml netifaces
```

10. Preparació del Turtlebot 3 Burger

Abans de començar a dissenyar els escenaris i afegir-hi el model de Machine Learning al robot, cal definir-li uns certs paràmetres com és en l'activació del làser perquè sigui capaç de reconèixer l'entorn, i per altre banda, s'ha de modificar els paràmetres de moviment del robot per tenir una fluïdesa en el seu recorregut.

10.1 Preparació dels paràmetres del robot

El Turtlebot3 Burger per defecte té una velocitat lineal de 0.15 m/s. La velocitat angular a la qual gira està determinada per les accions. Les accions és un paràmetre en el qual es relaciona una variable a la velocitat angular (rad/s) a la que ha de girar el robot tal i com és pot observar a la següent taula.

Acció	Velocitat Angular (rad/s)
0	-1.5
1	-.075
2	0
3	0.75
4	1.5

Taula 10.1: Relació entre l'acció i la velocitat angular.

Font[18]

En les següents imatges es pot observar des de un punt de vista més gràfic la interpretació de la Taula 10.1.

Podem observar quan l'acció és 0 (és a dir gira cap a la Esquerra) la velocitat angular és -1.5 rad/s a la Fig 10.1.

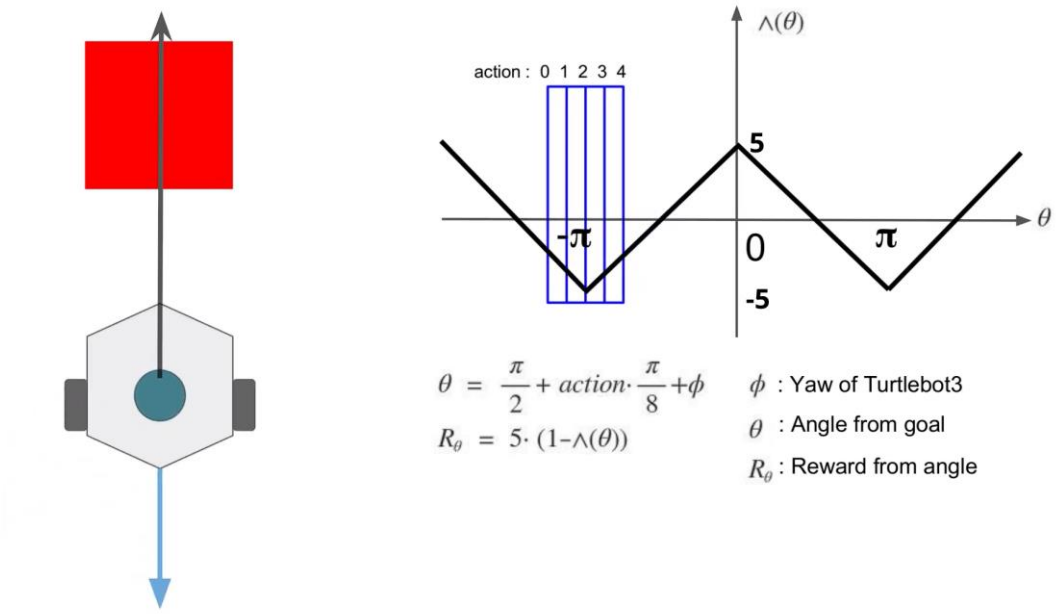


Figura 10.1:Representació quan l'acció és 0 la velocitat angular és -1.5 rad/s.

Font: [18]

Per altre banda, podem observar quan l'acció és 2 (no gira) la velocitat angular és 0 rad/s.

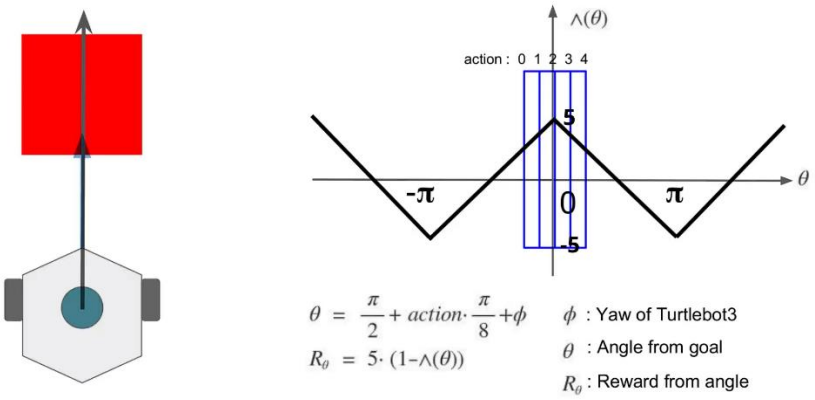


Figura 10.2:Representació quan l'acció és 2 la velocitat angular és 0 rad/s.

Font: [18]

I per últim s'observa que quan l'acció és 4 la velocitat (gira a la seva dreta) angular és 1.5 rad/s.

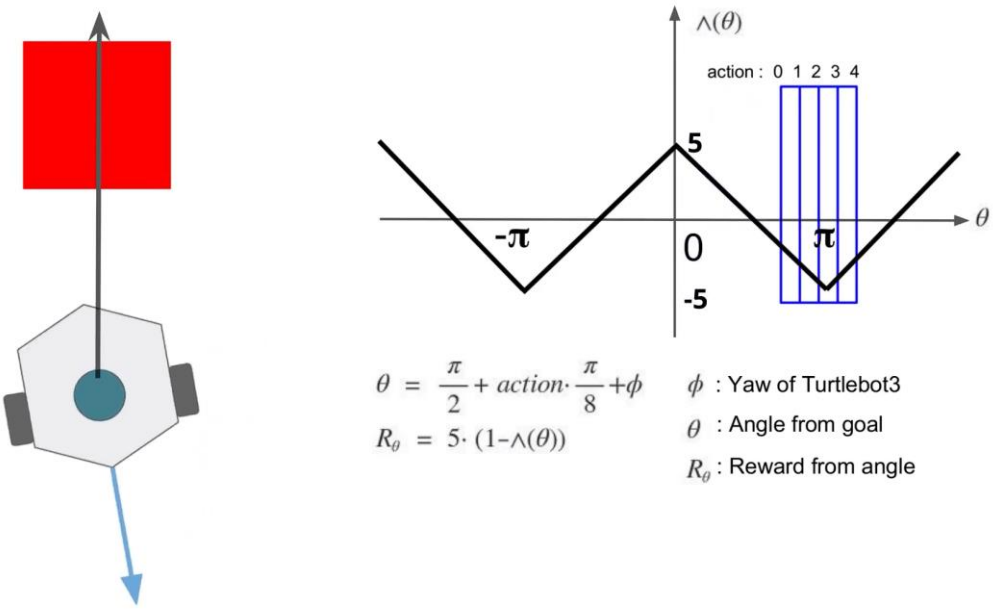


Figura 10.3: Representació quan l'acció és 4 la velocitat angular és 1.5 rad/s.

Font: [18]

Si anem a Visual Studio i obrim el següent directori:

Ubuntu/catkin_ws/src/turtlebot3/turtlebot3_navigation/param/dwa_local_planner_params_burger.yaml

En aquest directori és pot modificar cada paràmetre del robot ja sigui en termes de moviment lineal, velocitat angular, la distancia fins a la recompensa, etc.

10.2 Prepació del Làser Lidar

El Turtlebot 3 utilitza el làser Lidar per observar el seu entorn i determinar la seva situació actual. El seu valor predeterminat de *"samples"* de LDS del TurtleBot3 és de 360º la qual cosa no ens interessa gaire ja que no volem que el robot pugui veure a tot arreu, sinó que el robot es fixi en una zona determinada, tal i com podem observar en la següent figura:

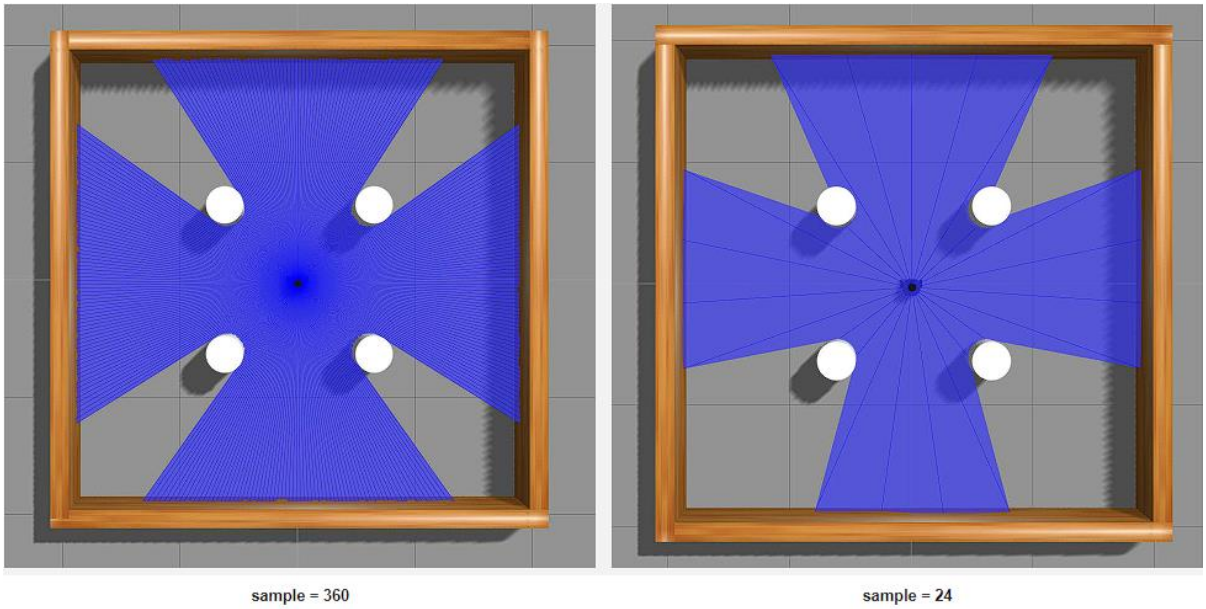


Fig 10.4: Comparació de Samples del LDS del Turtlebot 3.

Font: [18]

És molt important determinar el valor del LDS a 24 ja que com veurem més endavant, tindrà un gran efecte a determinar el State per estudiar la distància que hi ha del robot al punt de recompensa.

Per modificar el valor de mostres del LDS s'ha d'anar al següent directori:

ubuntu/catkin_ws/src/turtlebot3/turtlebot3_description/urdf/turtlebot3_burger.gazebo.xacro

En les primeres línies es mostra la instrucció per activar el làser Lidar del TurtleBot3 Burger. Cal canviar el valor de default de false a true.

```
<?xml version="1.0"?>
<robot name="turtlebot3_burger_sim" xmlns:xacro="http://ros.org/wiki/xacro">
  <xacro:arg name="laser_visual" default="true"/>  #Canviar el valor de default=false a true
  <xacro:arg name="imu_visual"   default="false"/>
```

Per altre banda, volem canviar el valor de mostres del làser. Reduint-ho de 360 a 24, amb l'objectiu d'aconseguir una observació de l'entorn més realista.

```
<scan>
  <horizontal>
    <samples>360</samples>   # Canviar el valor de samples a 24
    <resolution>1</resolution>
    <min_angle>0.0</min_angle>
    <max_angle>6.28319</max_angle>
  </horizontal>
</scan>
```

11. Preparació del model de Machine Learning

El mètode aplicat per implementar el Machine Learning en el Turtlebot3 Burger és basa en el aprenentatge reforçat, un algoritme en el que el computador aprèn a partir de compensacions de les accions realitzades en el escenari. Aquestes compensacions poden ser tant positives com negatives, depenent de l'objectiu què es volguí aprendre. En el nostre cas és desenvolupar-se en un entorn sense xocar-se amb cap obstacle que se li presenti.

Per altre banda, hi ha diferents mètodes per aplicar l'aprenentatge reforçat, en aquest projecte s'ha triat com a mètode el Deep Q-learning (DQN)

11.1 Paquets de Machine Learning a ROS

Abans de començar la instal·lació dels paquets de Machine Learning, ens hem d'assegurar que tenim instal·lat els paquets següents: **turtlebot3, turtlebot3_msgs i turtlebot3_simulations.**

Es necessari tenir el paquet de simuladors del Turtlebot3 perquè necessitarem treballar amb l'eina Gazebo per fer córrer el model de Machine Learning. En tot cas si no s'ha instal·lat el paquet de simuladors del Turtlebot3, les instruccions per instal·lar-ho són les següents.

```
$ cd ~/catkin_ws/src/
$ git clone -b kinetic-devel https://github.com/ROBOTIS-
GIT/turtlebot3_simulations.git
$ cd ~/catkin_ws && catkin_make
```

Un cop tenim tots els paquets preparats, es comença amb la instal·lació de paquets de Machine Learning. Turtlebot3 compta amb un paquet especialitzat en Machine Learning on dins d'aquest ja ve un model base amb el seu codi predefinit, on només cal ajustar-ho a la especificació que un necessita.

Les instruccions a seguir per instal·lar-ho són les següents:

```
$ cd ~/catkin_ws/src/
$ git clone -b kinetic-devel https://github.com/ROBOTIS-
GIT/turtlebot3_simulations.git
$ cd ~/catkin_ws && catkin_make
```

Per rectificar els problemes que poden haver-hi, cal desinstal·lar i instal·lar de nou la llibreria “Numpy” un paquet d’àlgebra lineal de Python utilitzat sobretot en models d’aprenentatge autònom, entre ells els Machine Learning. S’ha de desinstal·lar un parell de cops per poder verificar que s’ha desinstal·lat completament.

```
$ pip uninstall numpy
$ pip show numpy
$ pip uninstall numpy
$ pip show numpy
```

Per últim, comprovem si existeix algun arxiu “Numpy” escrivint en el terminal “pip show numpy”. Si no surt cap arxiu en el terminal, voldrà dir que la desinstal·lació s’ha efectuat correctament, per tant tornem a instal·lar la llibreria “Numpy”

```
$ pip install numpy pyqtgraph
```

11.2 Disseny del sistema de recompenses

El disseny de les recompenses és molt important per el aprenentatge del robot. Una recompensa pot ser positiva com negativa, en funció de les accions realitzades per aquest. Quan el Turtlebot3 arriba a l’objectiu sense xocar amb cap obstacle, aconsegueix una recompensa positiva, per altre banda, obté una recompensa negativa si es xoca amb l’entorn. Com a tal, l’objectiu principal del robot és aconseguir recompenses positives, per això, haurà d’aprendre que no xocar-se amb obstacles i arribar al objectiu marcat és una condició necessària per aconseguir recompenses positives.

La funció matemàtica que determina aquest sistema de recompenses és la següent:

Reward Formula

$$
\begin{aligned}
&\text{if } -\frac{1}{2}\pi < \theta < \frac{1}{2}\pi, \\
&\quad R_\theta \geq 0 \\
&\text{else} \\
&\quad R_\theta < 0 \\
&\\
&\text{if } D_c < D_g, \\
&\quad R_d > 2 \\
&\text{else} \\
&\quad 1 < R_d \leq 2 \\
&\\
&R = R_\theta \cdot R_d
\end{aligned}
$$

θ : Angle from TurtleBot3 to Goal
D_c : Current distance from Goal
D_g : Absolute distance from Goal
R_θ : Reward from θ
R_d : Reward from distance
R : Reward

On a partir de les figures 10.1 10.2 i 10.3 és pot observar com aplicar-les en funció de les accions del Turtlebot 3.

Per aplicar en el nostre disseny el sistema de recompenses s'ha de modificar els escenaris que hem instal·lat a partir de la llibreria de **turtlebot3_simulations.** Per tant en els quatre escenaris que treballarem modifiquem el valor del *"setReward"* que podrem trobar en el següent directori:

"ubuntu/catkin_ws/src/turtlebot3_machine_learning/turtlebot3_dqn/src/turtlebot3_dqn/environment_stage_#.py" On # Significa escenari 1, 2, 3 o 4.

```
92      def setReward(self, state, done, action):
93          yaw_reward = []
94          current_distance = state[-1]
95          heading = state[-2]
96
97          for i in range(5):
98              angle = -pi / 4 + heading + (pi / 8 * i) + pi / 2
99              tr = 1 - 4 * math.fabs(0.5 - math.modf(0.25 + 0.5 * angle % (2 * math.pi) / math.pi)[0])
100             yaw_reward.append(tr)
101
102         distance_rate = 2 ** (current_distance / self.goal_distance)
103         reward = ((round(yaw_reward[action] * 5, 2)) * distance_rate)
104
105         if done:
106             rospy.loginfo("Collision!!")
107             reward = -200
108             self.pub_cmd_vel.publish(Twist())
109
110         if self.get_goalbox:
111             rospy.loginfo("Goal!!")
112             reward = 200
113             self.pub_cmd_vel.publish(Twist())
114             self.goal_x, self.goal_y = self.respawn_goal.getPosition(True, delete=True)
115             self.goal_distance = self.getGoalDistace()
116             self.get_goalbox = False
117
118         return reward
```

Fig 11.1: Codi del sistema de recompenses.

Com podem observar, s'ha afegit la fórmula descrita anteriorment, depenent de l'acció realitzada i la distància entre el robot i l'objectiu determina una recompensa, si el robot ha xocat, en el controlador salta una senyal de col·lisió i la recompensa és de -200 punts, per altra banda, si ha arribat a l'objectiu sense cap problema surt un missatge de que ho ha aconseguit i se li afegeixen 200 punts.

Aconseguint així que el robot aprengui a què no xocar-se amb l'entorn és una acció correcta i l'haurà d'aplicar en l'entorn en el qual es troba.

11.3 Disseny del model d'aprenentatge reforçat

Com s'ha descrit anteriorment, aquest projecte es realitza utilitzant com a model d'aprenentatge reforçat el sistema DQN. On aquest mètode d'aprenentatge reforçat selecciona una xarxa neuronal profunda aproximant la funció de cada acció (valor Q) evitant així utilitzar una taula per representar la mateixa. En realitat aquest model utilitza dues xarxes neuronals, la primera, la xarxa neuronal principal, representada pels paràmetres de θ descrites anteriorment. S'utilitza per estimar els valors de l'estat i l'acció actuals. La segona, la xarxa neuronal objectiu, parametritzada per θ' te la mateixa funció que la primera, però en aquest cas aproxima els valors del següent estat i acció.

Tot l'aprenentatge passa en la xarxa principal, ja que la segona es "congela", és a dir, que els seus paràmetres no varien en unes iteracions (10000) aproximadament i després la xarxa principal copia a l'objectiu, transmitent així l'aprenentatge d'una a l'altre, aconseguint estimacions més precises.

Per millorar el model s'han utilitzat una sèrie de híperparàmetres que determinen diferents aspectes essencials en l'aprenentatge del robot.

Híper-paràmetre	Valor	Descripció
Episode_step	6000	El pas de temps d'un episodi
Target_variable	2000	Tassa d'actualització de la xarxa destí
discount_factor	0.99	Representa la quantitat de valor perdut en els esdeveniments futurs segons la distància
learning_rate	0.00025	Velocitat d'aprenentatge. Si es molt gran no funciona bé i si es molt petit triga molt en aprendre.
epsilon	1.0	La probabilitat de triar una acció aleatòria
epsilon_decay	0.99	Temps de reducció de la epsilon. Quan acaba un episodi, la epsilon es redueix.
epsilon_min	0.05	El mínim de epsilon.

batch_size	64	Mida d'un grup de mostres d'entrenament
train_start	64	Comença a entrenar si la mida de la memòria de reproducció és superior a 64.
memory	1000000	La mida de la memòria de reproducció.

Taula 11.1:Taula dels diferents Híper-paràmetres.

Font: Elaboració pròpia

Podem modificar aquests híper-paràmetres accedint-hi des de el següent directori:

"ubuntu/catkin_ws/src/turtlebot33_machine_learning/turtlebot3_dqn/nodes/turtlebot3_dqn_stage_#". S'ha de fer per els quatre diferents escenaris.

```
EPISODES = 3000

class ReinforceAgent():
    def __init__(self, state_size, action_size):
        self.pub_result = rospy.Publisher('result', Float32MultiArray, queue_size=5)
        self.dirPath = os.path.dirname(os.path.realpath(__file__))
        self.dirPath = self.dirPath.replace('turtlebot3_dqn/nodes', 'turtlebot3_dqn/save_model/stage_1_')
        self.result = Float32MultiArray()

        self.load_model = False
        self.load_episode = 0
        self.state_size = state_size
        self.action_size = action_size
        self.episode_step = 6000
        self.target_update = 2000
        self.discount_factor = 0.99
        self.learning_rate = 0.00025
        self.epsilon = 1.0
        self.epsilon_decay = 0.99
        self.epsilon_min = 0.05
        self.batch_size = 64
        self.train_start = 64
        self.memory = deque(maxlen=1000000)

        self.model = self.buildModel()
        self.target_model = self.buildModel()

        self.updateTargetModel()

        if self.load_model:
            self.model.set_weights(load_model(self.dirPath+str(self.load_episode)+".h5").get_weights())

            with open(self.dirPath+str(self.load_episode)+'.json') as outfile:
                param = json.load(outfile)
                self.epsilon = param.get('epsilon')
```

Fig 11.2: Codi de la implementació d'híper-paràmetres.

12. Disseny dels diferents escenaris

En aquest apartat es centrarà bàsicament en mostrar els diferents dissenys d'escenaris i amb quins directoris es poden accedir a ells. Principalment, farem les simulacions per quatre escenaris diferents per veure fins a on és capaç d'arribar el Turtlebot3 Burger i quants episodis necessita per aprendre en els diferents escenaris a no xocar-se amb l'entorn i moure's lliurement per aquest.

Per dur a terme els models de simulació, ens hem basat en els models definits per la llibreria **turtlebot3_simulations.** On podem observar un escenari sense obstacles, un altre amb obstacles estàtics, un altre amb obstacles mòbils i per acabar un escenari amb obstacles mòbils i estàtics.

12.1 Disseny del escenari sense obstacles.

El primer escenari que tractarem serà un quadrat de 4x4 sense obstacles tal i com podem observar en la figura 12.1.

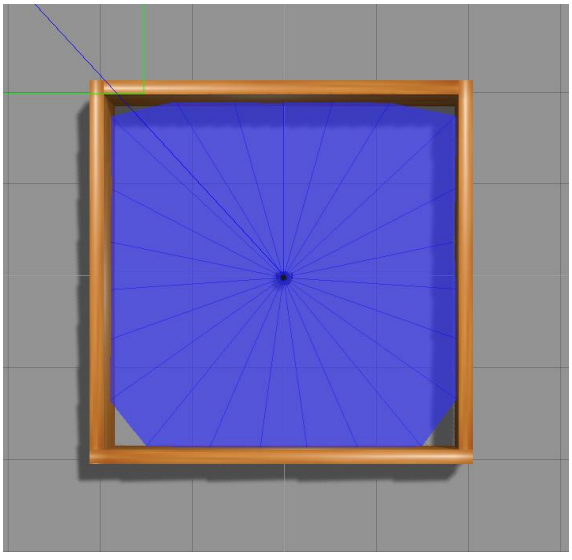


Fig 12.1. Primer escenari sense obstacles.

Font: Elaboració pròpia.

L'objectiu principal d'aquest escenari consisteix en observar com es comporta el robot en un entorn buit, és a dir com aprèn de ràpid en aquest escenari, per després comparar-ho amb els altres escenaris els quals tindran diferents obstacles.

Per poder accedir-hi a aquest escenari, s'ha d'obrir la consola de comandos d'Ubuntu i primerament hem d'inicialitzar ROS amb la següent instrucció:

$ roscore

Seguidament, obrim un altre terminal i escrivim la següent instrucció

$ roslaunch turtlebot3_gazebo turtlebot3_stage_1.launch

12.2 Disseny del escenari amb obstacles estàtics.

El segon escenari que es tracta serà un quadrat 4x4 igual que l'anterior, però amb la diferència que en aquest model hi ha quatre obstacles cilíndrics estàtics, tal com es pot observar a la Fig 12.2.

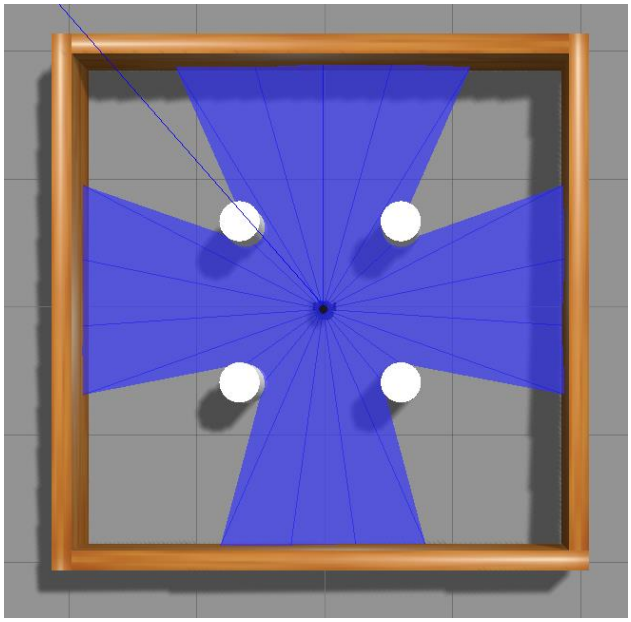


Fig 12.2. Segon escenari amb obstacles estàtics.

Font: Elaboració pròpia.

L'objectiu principal d'aquest escenari consisteix en observar com és comporta el robot en un entorn amb quatre obstacles estàtics amb la finalitat d'estudiar la diferència que hi

ha hagut amb el primer escenari per aprendre a arribar al seu destí sense col·lisionar amb cap obstacle estàtic.

Per accedir-hi a aquest escenari, s'ha d'obrir la consola de comandos d'Ubuntu i primerament hem de inicialitzar ROS amb la següent instrucció:

$ roscore

Seguidament, obrim un altre terminal i escrivim la següent instrucció

$ roslaunch turtlebot3_gazebo turtlebot3_stage_2.launch

12.3 Disseny del escenari amb obstacles mòbils.

El tercer escenari que es tracta serà un quadrat 4x4 amb la diferència de la resta que en aquest model hi ha quatre obstacles cilíndrics mòbils, tal i com es pot observar a la Fig 12.3.

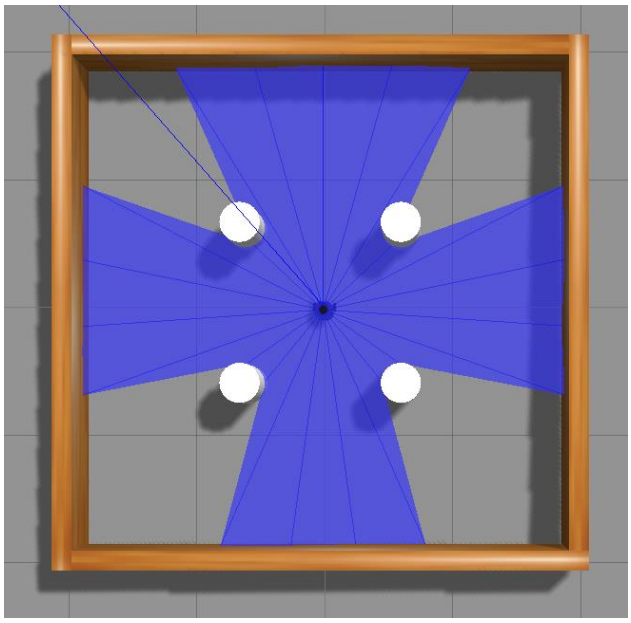


Fig 12.3. Tercer escenari amb obstacles mòbils.

Font: Elaboració pròpia.

L'objectiu principal d'aquest escenari consisteix a estudiar el comportament del robot enfront en circumstàncies d'objectes mòbils, aquest obstacles podrien simular situacions

com vianants, automòbils, etc. Serà curiós avaluar la diferència de simulacions que ha realitzat aquesta situació en diferència a la resta.

Per accedir-hi a aquest escenari, s'ha d'obrir la consola de comandos d'Ubuntu i primerament hem d'inicialitzar ROS amb la següent instrucció:

$ roscore

Seguidament, obrim un altre terminal i escrivim la següent instrucció

$ roslaunch turtlebot3_gazebo turtlebot3_stage_3.launch

12.4 Disseny del escenari amb obstacles mòbils i estàtics

L'últim escenari consisteix en un mapa 5x5 tal i com podem observar a la Fig 12.4 amb murs estàtics i obstacles cilíndrics que es van movent per tot el mapa.

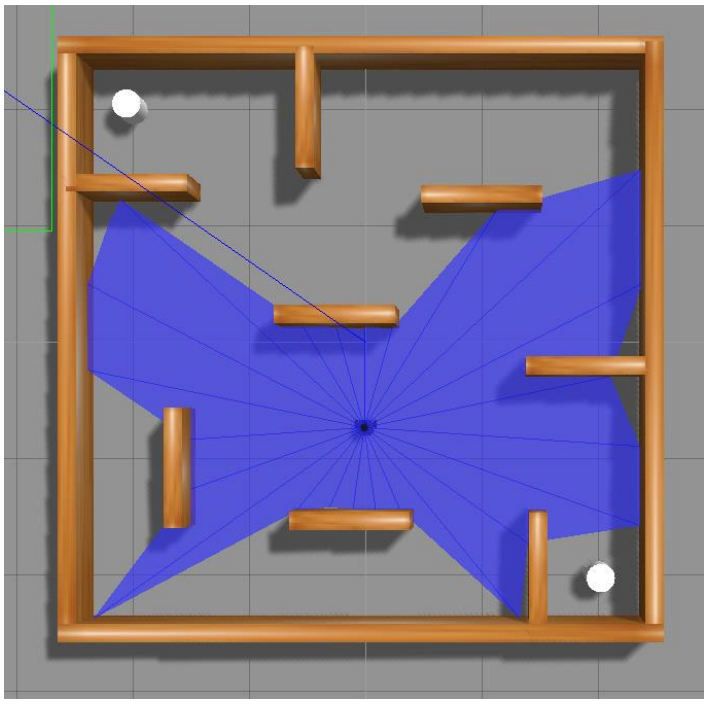


Fig 12.4. Quart escenari amb obstacles mòbils i estàtics.

Font: Elaboració pròpia.

L'objectiu d'aquest escenari és el més complexa de tots, ja que s'estudia com interactua el robot en un entorn que gairebé intervenen molts aspectes que a la vida real poden passar-hi, com poden ser llocs tancats amb persones, cotxes, vianants, una paret, etc.

És a dir, situacions quotidianes de la vida real, per això és el més interesant de tots els escenaris.

S'estima que el temps d'aprenentatge per aquest model serà gairebé elevat, ja que hi han moltes variables que el robot ha d'interpretar, a diferència del primer escenari on només hi ha les quatre parets que delimiten l'espai.

Per accedir-hi a aquest escenari, s'ha d'obrir la consola de comandos d'Ubuntu i primerament hem de inicialitzar ROS amb la següent instrucció:

$ roscore

Seguidament, obrim un altre terminal i escrivim la següent instrucció

$ roslaunch turtlebot3_gazebo turtlebot3_stage_4.launch

13. Simulacions

En aquest apartat, es centrarà en unificar tot el que hem descrit anteriorment i es començarà a fer les simulacions del model pels diferents escenaris. S'entrenarà el model tants cops com facin falta, depenent de la rapidesa d'aprenentatge. Per alguns escenaris, hi caldrà més temps que per uns altres ja que hi ha de simples i d'altres més complexes. Com més temps entreni el model, millor resposta anirà adquirint.

Per visualitzar i agrupar els resultats, carregarem un gràfic on es visualitza el total de punts que ha aconseguit el robot i per altra banda el valor màxim de valor Q (el qual hem descrit en l'apartat 11.3).

Per altra banda, veurem en temps real les accions que està prenent el robot, tal com s'ha vist en l'apartat 10. S'escriu en el terminal la següent instrucció per obrir els diferents gràfics.

$ roslaunch turtlebot3_dqn result_graph.launch

El que visualitzarem serà el següent:

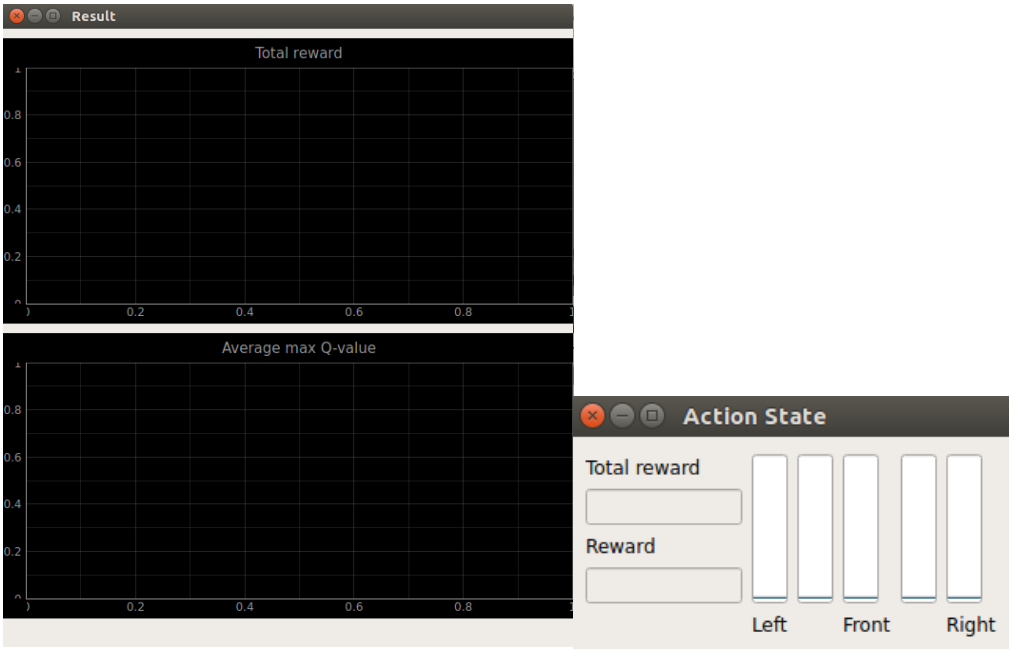


Fig 13.1. Gràfics de les recompenses, el valor de Q i les accions.

Font: Elaboració Pròpia.

En la implementació de DQN, s'utilitza una tècnica anomenada "Epsilon-Greedy", en la qual se selecciona l'acció amb la major Q-value amb una probabilitat de 1-epsilon i una acció aleatòria amb una probabilitat d'èpsilon. A mesura que el model s'entrena, es redueix gradualment el valor d'èpsilon perquè el model depengui menys de l'exploració i més de l'explotació del seu coneixement actual i aconsegueixi l'objectiu principal que és moure's per un entorn sense col·lidir amb cap obstacle, ja que el que fa el robot en les simulacions és arribar a uns punts on es segur estar ja que no hi ha obstacles, i per tant s'ha de moure sense xocar amb res i com a tal ha d'aprendre a desenvolupar-se per aquell entorn sense xocar-se amb cap obstacle.

13.1 Simulació del primer escenari

Primer de tot, per començar amb el entrenament del robot executant el primer escenari, hem de carregar aquest en Gazebo. Per fer-ho utilitzem la instrucció descrita a l'apartat 12.1 i el minimitzem.

$ roslaunch turtlebot3_gazebo turtlebot3_stage_1.launch

Un cop carregat el escenari, obrim un altre terminal per carregar els gràfics del aprenentatge reforçat DQN tal i com acabem de veure anteriorment i el minimitzem.

$ roslaunch turtlebot3_dqn result_graph.launch

I per finalitzar, carreguem el codi on es troba el sistema de recompenses que hem dissenyat, juntament amb el model DQN descrit, per fer-ho, s'ha d'obrir un altre terminal on escrivim el següent.

$ roslaunch turtlebot3_dqn turtlebot3_dqn_stage_1.launch

Aquest terminal no el minimitzem, ja que com es pot veure en l'Annex 3 aquest ens mostra quan el robot col·lideix i quan arriba al seu objectiu, els punts i el valor de la èpsilon per cada episodi, això és molt important perquè podem anar veient el seu aprenentatge de forma directa.

Un cop carregat tot, el robot comença amb el seu aprenentatge. Durant els primers deu episodis, el robot col·lideix constantment ja que està interpretant que al xocar se li està

restant punts i ha de trobar la forma de poder sumar-ho tal i com podem observar en la següent figura.

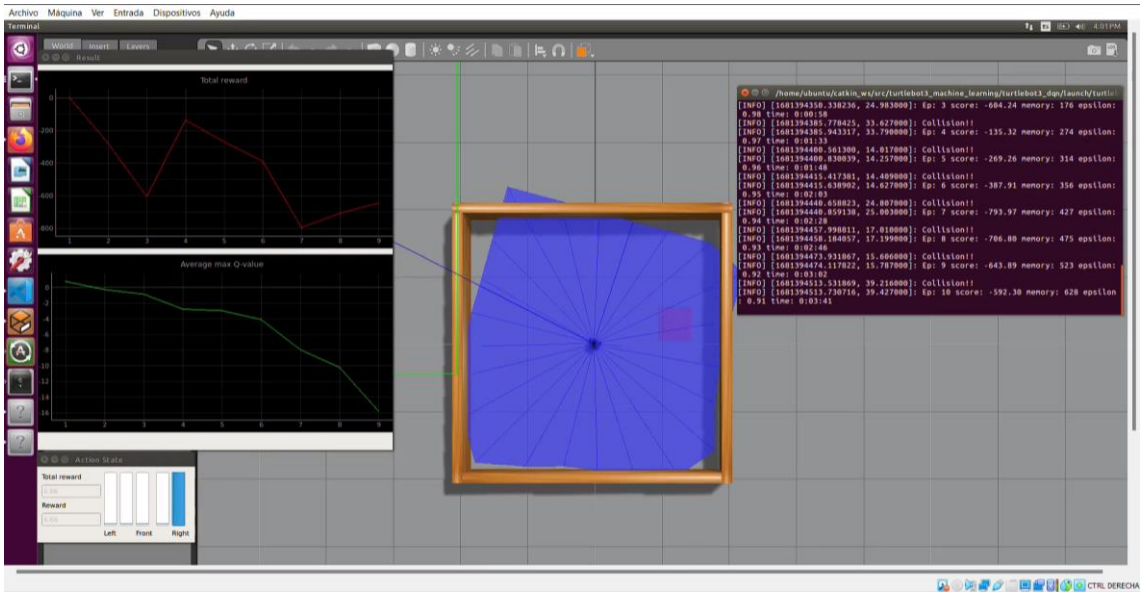


Fig 13.2 Simulació primer escenari episodi 0-10.

Font: Elaboració pròpia.

Com podem observar, el punt vermell dins del quadrat es el destí al que el robot ha d'arribar sense col·lidir amb cap obstacle, aquest va canviant cada cop que arriba al destí, si el robot xoca amb alguna cosa, aquest marcador no varia de lloc. Per altra banda, s'observa la gràfica representativa de les recompenses que va adquirint el robot i per altre el valor de Q màxim que es determina per cada episodi. Seguidament, a la dreta observem els resultats de cada episodi.

Al principi el robot no entén que el que ha de fer es anar al punt marcat sense col·lidir amb res, per tant com podem observar ell no para de xocar-se amb les parets del quadrat durant aquests episodis fins al punt que arribarà al destí i començarà a guanyar punts, entenent que és el que ha de fer i acabarà fent-ho sense xocar-se amb res.

Al cap de cinquanta episodis podem observar que ha entès el seu objectiu i arriba al seu punt destí directament sense xocar-se amb les parets tal com podem observar en la següent figura.

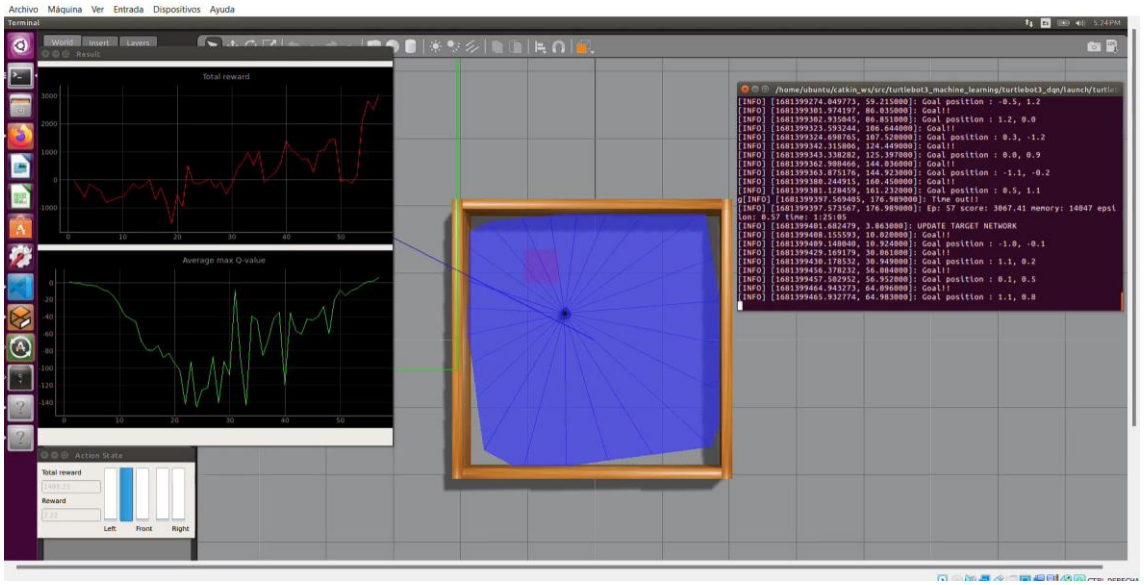


Fig 13.3 Simulació primer escenari episodi 50-60.

Font: Elaboració pròpia.

Aconseguint així l'objectiu principal d'aquest projecte el qual era aplicar el Machine Learning aplicant el mètode DQN. Ja que com podem observar el robot en un principi no sabia el que estava fent ni el que havia de fer, xocant-se i no arribant al seu destí correctament, però amb el pas dels episodis, ha anat entenent que no s'ha de xocar amb les parets, ja que ocasiona una puntuació negativa cap a ell i a l'haver arribat al destí ha obtingut punts, llavors és centra només a arribar a aquells destins marcats.

El valor de l'èpsilon que s'ha extret al final de l'últim episodi (60) és de 0.5582661385478638 la qual cosa és bastant bona ja que determina que el model no dependria d'una exploració addicional per arribar a l'objectiu, ell ja directament va cap a ell sense necessitat d'explorar el seu entorn i ,per tant, el Q-Value serà positiu.

Aquest és l'escenari més senzill on no hi ha obstacles pel mig que dificultin l'aprenentatge d'aquest, per tant, ara veurem com reacciona el robot enfront situacions més adverses.

13.2 Simulació del segon escenari

De la mateixa forma que hem fet anteriorment, primer de tot obrim el escenari corresponent tal i com s'ha descrit en el apartat 12.2 i el minimitzem.

$ roslaunch turtlebot3_gazebo turtlebot3_stage_2.launch

Seguidament, un cop carregat el escenari, obrim un altre terminal per carregar els gràfics del aprenentatge reforçat DQN tal i com acabem de veure anteriorment i el minimitzem.

$ roslaunch turtlebot3_dqn result_graph.launch

I per finalitzar, carreguem el codi on es troba el sistema de recompenses que hem dissenyat, juntament amb el model DQN descrit, per fer-ho, s'ha d'obrir un altre terminal on escrivim el següent.

$ roslaunch turtlebot3_dqn turtlebot3_dqn_stage_2.launch

Aquest terminal no el minimitzem pels motius descrits anteriorment i comencem a entrenar el model. Durant els primers episodis, el robot actua col·lidint constantment amb el seu entorn, ja que a diferència de l'anterior model, aquest model compta de més obstacles i per tant té més inconvenients per aprendre. Com podem observar en els episodis 20-30 encara actua com si fossin dels primers, ja que li està costant més arribar al punt destí tal com es pot observar a la fig 13.4.

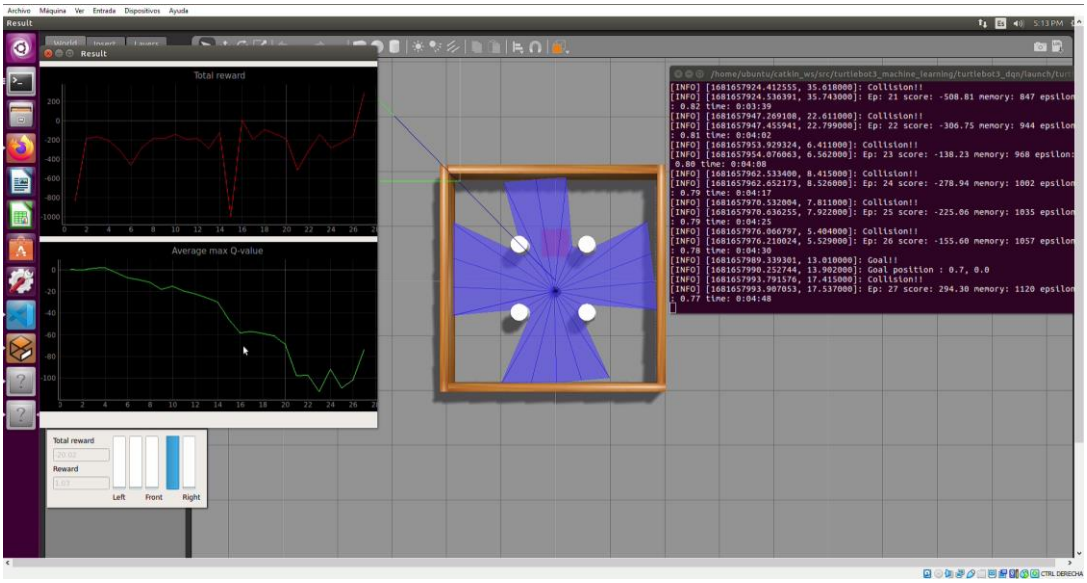


Fig 13.4 Simulació segon escenari episodi 20-30.

Font: Elaboració pròpia.

Com es pot observar aquest model costa una mica més d'aprendre, ja que se li estan posant obstacles estàtics pel camí la qual cosa dificulta l'arribada al punt destí del robot i incrementa la probabilitat de què xoqui amb algun obstacle i com a tal causa que el Q-Value sigui al principi relativament baix, ja que la epsilon es bastant gran i per tant necessita de més temps per explorar l'entorn i aprendre. Però a mesura que el robot va encertant l'objectiu, progressivament es va notant una millora en les respostes d'aquest.

Es pot visualitzar en el gràfic de recompenses que hi ha certs pics, aquests pics s'ocasionen quan el robot es troba en una situació que no sap que fer per lo tant fa un pic d'increment i decreixent dels punts per restaurar-se. També es troba quan la puntuació és molt negativa els punts s'estabilitzen perquè no costi tant que aprengui i tingui punts positius.

A mesura que van passant episodis, el robot mica en mica va aprenent a reconèixer amb major facilitat els obstacles i a saber evitar-los, tal com veiem en la següent figura.

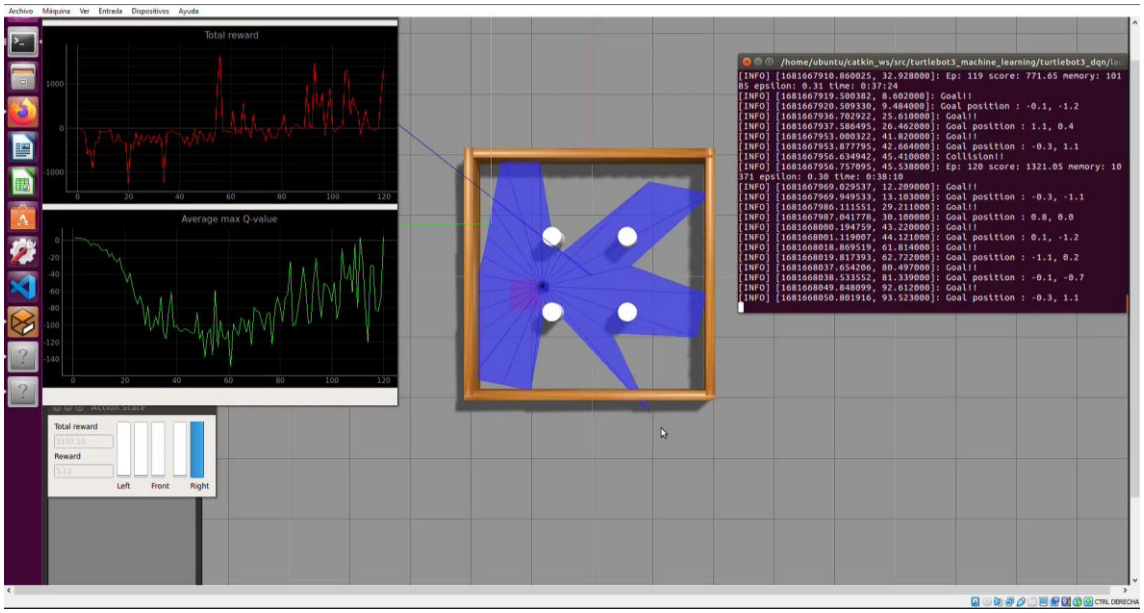


Fig 13.5 Simulació segon escenari episodi 20-30.

Font: Elaboració pròpia.

Com podem observar el robot ha après a evitar els obstacles estàtics que hi ha en el escenari i a arribar directament al objectiu amb la major rapidesa possible. La seva èpsilon és de 0.20434346174623952 la qual cosa és bastant baixa ja que el Q-Value es bastant bó i per tant, el robot no te gaire dependència en explorar l'entorn per arribar al destí. Per tant podem comprovar que no es xocaria gairebé amb l'entorn si fos un model real.

13.3 Simulació del tercer escenari

A continuació, es procedeix a realitzar l'entrenament del model per al tercer escenari, un escenari idèntic a l'anterior amb la diferència que els obstacles no són estàtics sinó mòbils, la qual cosa augmenta significativament la complexitat d'aprenentatge a la que s'enfronta al robot perquè no només té un inconvenient que són els obstacles sinó que es van movent i els ha de visualitzar i els ha d'envoltar per tal de no col·lidir.

Com hem fet anteriorment, primer de tot carreguem en Gazebo el escenari amb la següent instrucció.

$ roslaunch turtlebot3_gazebo turtlebot3_stage_3.launch

Seguidament, un cop carregat el escenari, obrim un altre terminal per carregar els gràfics del aprenentatge reforçat DQN i el minimitzem.

$ roslaunch turtlebot3_dqn result_graph.launch

Per finalitzar, carreguem el codi on es troba el sistema de recompenses que hem dissenyat, juntament amb el model DQN descrit, per fer-ho, s'ha d'obrir un altre terminal on escrivim el següent.

$ roslaunch turtlebot3_dqn turtlebot3_dqn_stage_3.launch

Aquest terminal no el minimitzem pels motius descrits anteriorment i comencem a entrenar el model.
En aquest escenari, el robot trigarà bastant en poder començar a millorar els seus resultats, ja que en aquest cas com hem dit anteriorment, es troba enfront d'un escenari amb obstacles que es van movent, la qual cosa generarà al principi que es xoqui amb bastant freqüència i la seva puntuació sigui negativa. Com podem observar a la següent figura durant els episodis 50-60 encara el robot no ha aconseguit arribar a cap marcador, en tots aquests episodis ha xocat, i no és fins al episodi 150 aproximadament, quan aconsegueix arribar al primer marcador.

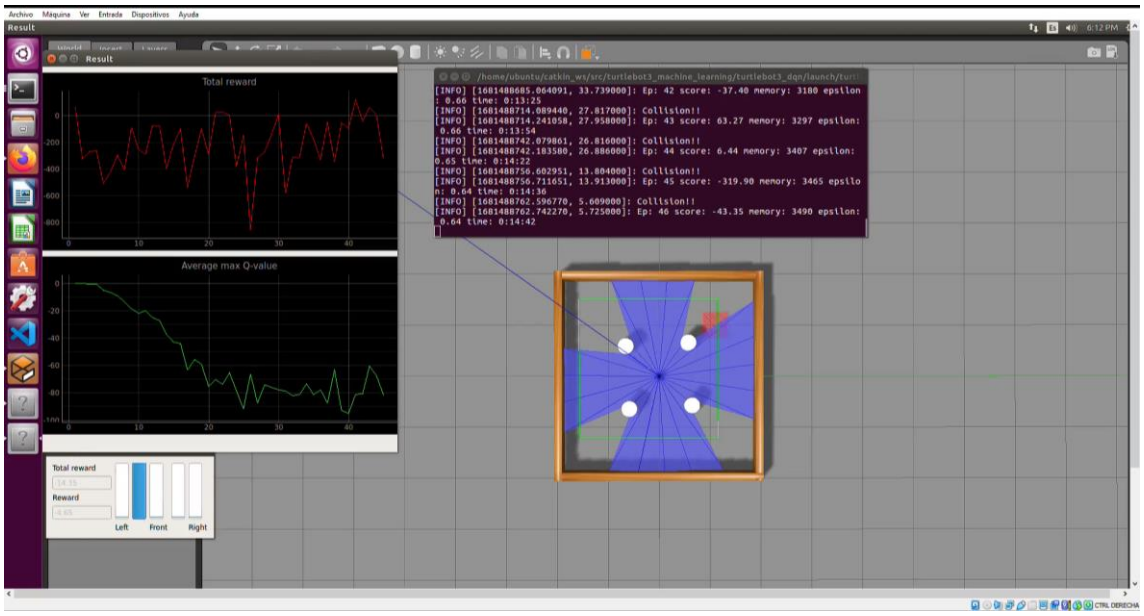


Fig 13.6 Simulació tercer escenari episodi 50-60.

Font: Elaboració pròpia.

És normal que trigui tant a arribar al punt, ja que ell està aprenent a base de adonar-se que col·lidir està malament i ha d'aprendre a sortir d'aquell patró que segueixen els obstacles per poder visualitzar i arribar al punt. Per tant, cada episodi en el qual s'equivoca, realment no es un error sinó que està aprenent a envoltar els obstacles aprenent els patrons que segueixen.

Per altra banda, podem visualitzar els pics descrits anteriorment, en aquest cas el més segur és que són a causa de l'alta puntuació negativa que te en aquell moment.

El robot de mica en mica a mesura que ja ha après a evitar els obstacles cada cop va arribant al destí amb major facilitat, si observem els episodis 660-670 observem que el robot gairebé ha aconseguit evitar els obstacles mòbils i a arribar als punts marcats amb una facilitat bastat notable, tal i com podem veure en la següent figura.

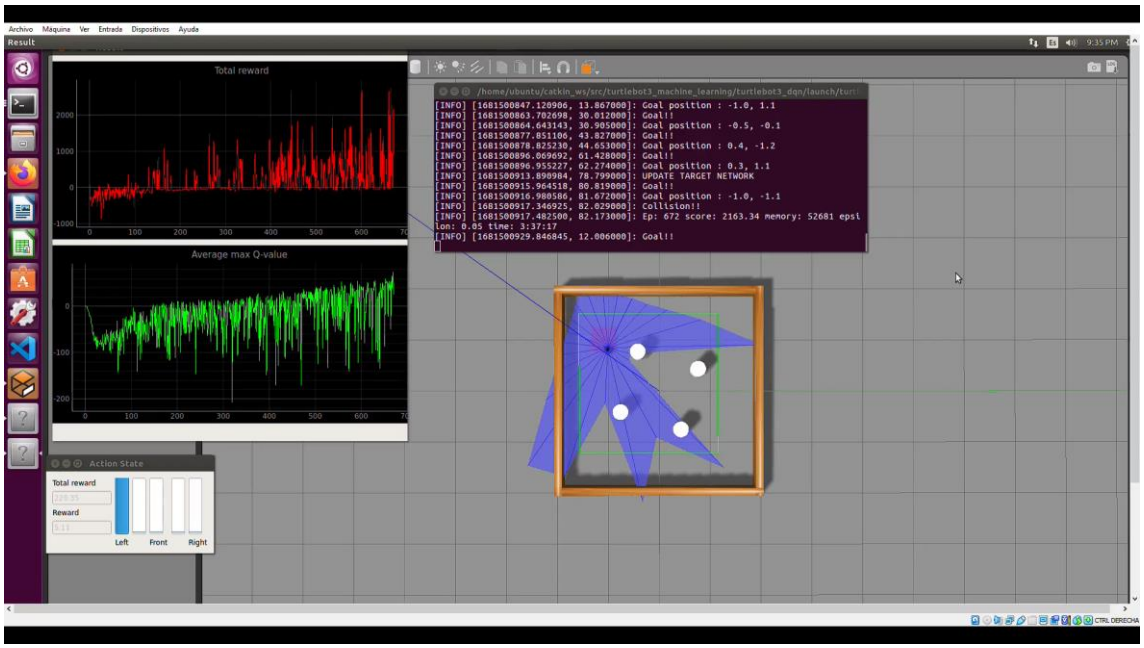


Fig 13.7 Simulació tercer escenari episodi 660-670.

Font: Elaboració pròpia.

Com s'observa a la figura 13.6 el robot ha après mitjançant el mètode reforçat de Machine Learning aplicant el mètode DQN a evitar obstacles mòbils i a arribar a un destí sense que es xoqui amb aquests. És curiós comentar que aquest robot no sabia que era un obstacle mòbil i ha hagut d'aprendre a esquivar els obstacles aprenent el patró que segueixen i així poder evitar xocar en el moment que el robot vol passar per mig.

El valor de l'èpsilon que s'ha extret al final de l'últim episodi (690) és de 0.04953625663766238 la qual cosa és bastant bona ja que determina que el model no dependria d'una exploració addicional per arribar a l'objectiu, ell ja directament va cap a ell sense necessitat d'explorar el seu entorn.

13.4 Simulació del quart escenari

Per finalitzar els entrenaments, acabem amb l'escenari en el qual s'implementa els tres escenaris anteriors. Això vol dir que el grau de dificultat al qual s'enfronta el robot és encara major a la resta de models. S'espera que es necessitin molts episodis per completar l'entrenament d'aquest, ja que si en l'anterior vam necessitar més de 600, en aquest s'estima que sigui encara major.

Aquest escenari és dels més curiós de tots, ja que es el que més s'assembla a una situació real on el robot s'enfronta a diferents obstacles i ha d'evitar col·lidir amb aquests. El robot s'entrenarà anant a uns punts determinats sense xocar-se amb l'entorn. Com a tal el robot va aprenent que arribant a punts sense xocar-se es positiu arran de les recompenses que se li va donant quan arriba al punt sense xocar. Per després poder guardar el model i aplicar-ho a un entorn real on el robot es podrà moure per un entorn sense col·lidir amb res.

Com hem fet anteriorment, primer de tot carreguem en Gazebo el escenari amb la següent instrucció.

$ roslaunch turtlebot3_gazebo turtlebot3_stage_4.launch

Seguidament, un cop carregat l'escenari, obrim un altre terminal per carregar els gràfics de l'aprenentatge reforçat DQN i el minimitzem.

$ roslaunch turtlebot3_dqn result_graph.launch

Per finalitzar, carreguem el codi on es troba el sistema de recompenses que hem dissenyat, juntament amb el model DQN descrit, per fer-ho, s'ha d'obrir un altre terminal on escrivim el següent.

$ roslaunch turtlebot3_dqn turtlebot3_dqn_stage_4.launch

Aquest terminal no el minimitzem pels motius descrits anteriorment i comencem a entrenar el model.

En aquest escenari, el robot trigarà encara molt més a començar a aprendre cap a on ha d'anar i cap a on no, ja que aquest escenari com s'ha comentat anteriorment compte amb una complexitat encara major, ja que s'està treballant amb un escenari el qual es conjunt de tots els anteriors, la qual cosa dificulta la rapidesa en la qual aprèn el robot. És a dir, hi ha una major probabilitat de que el robot es xoqui que de que arribi a l'objectiu. Per tant, de cada 200 episodis, igual només arriba 1 o 2 cops al objectiu. També pot ser que ni arribi.

Durant els primers episodis com podem veure a la següent figura, el robot actua com s'ha predit, amb certa dificultat per poder desplaçar-se per l'espai sense col·lidir amb cap obstacle.

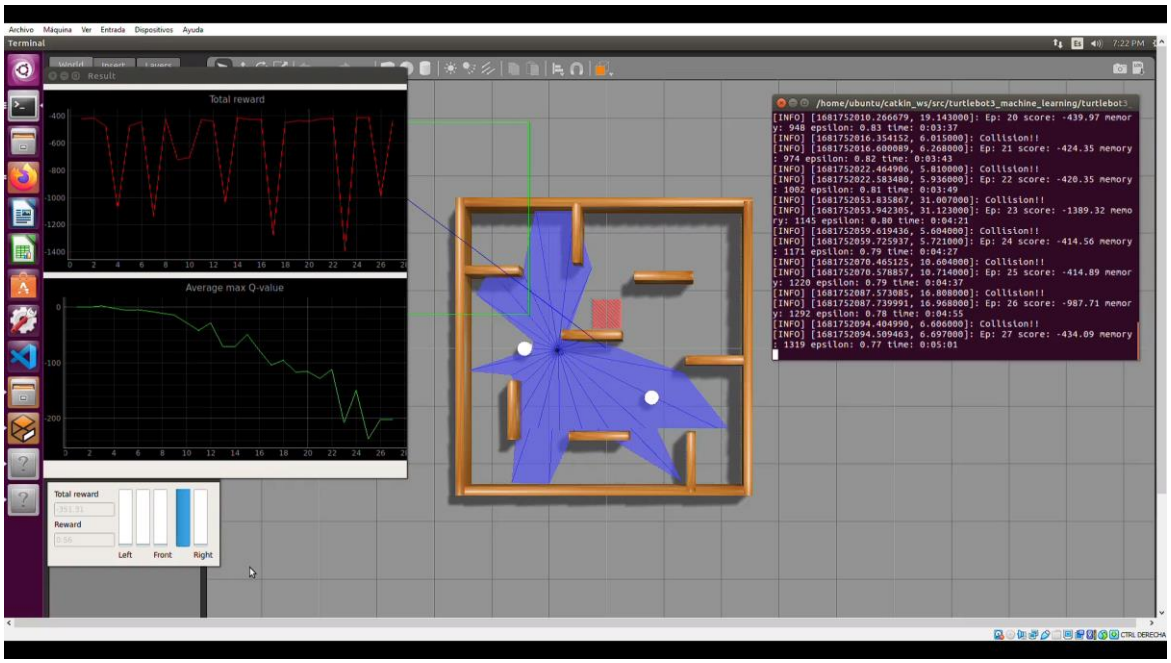


Fig 13.8 Simulació quart escenari episodi 0-30

Font: Elaboració pròpia.

Com es pot observar, en els primers trenta episodis, el robot col·lideix molt amb els obstacles, com s'ha comentat, és normal, a conseqüència de què aquest entrenament requereix molt de temps i paciència, ja que n'hi han moltes més accions a tenir en compte.

Durant el cicle d'entrenament, fins a l'episodi 100-150 que el robot aconsegueix encertar el primer punt sense col·lidir amb cap obstacle. Després de molt de temps de processament i d'espera, el model assoleix finalitzar el seu entrenament en l'episodi 860-880 tal com es pot veure en la següent figura, en el terminal es visualitza els últims episodis i tots són encertats.

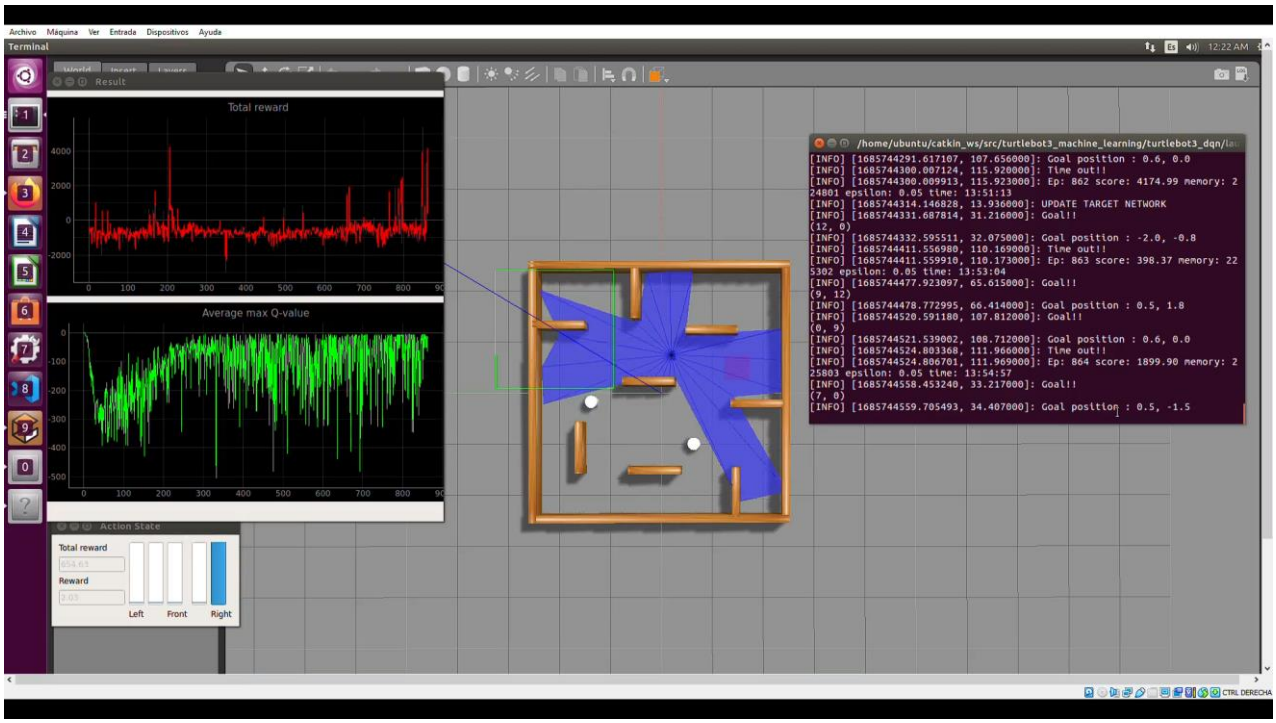


Fig 13.9 Simulació quart escenari episodi 860-870

Font: Elaboració pròpia.

Amb aquesta última simulació, podem donar per finalitzat l'entrenament del nostre robot per evitar obstacles, l'hem visualitzat en diferents entorns per avaluar la rapidesa d'aprenentatge que en te el robot depenent dels diferents obstacles en l'entorn. Un cop finalitzat podem confirmar que el robot ha après a evitar obstacles en diferents escenaris, ja que en un principi col·lidia amb tot l'entorn i a mesura que passaven els episodis, aquest començava a entendre que era un error xocar-se amb l'entorn fins a comprendre a moure's per aquest sense col·lidir.

Un cop finalitzat es confirma que a major dificultat d'escenari, major es el temps de processament que comporta entrenar el robot i major temps d'espera hem de dedicar-hi. Per altra banda, un cop finalitzat els entrenaments, se'ns genera dos arxius de tipus .h5 i .json on s'emmagatzema l'arquitectura de l'entrenament per poder implementar l'entrenament en un robot real i comprovar el funcionament en un entorn real.

14. Implementació Model Real

En aquest apartat es centra en la implementació de l'entrenament descrit anteriorment del quart escenari en el robot TurtleBot 3 Burguer per actuar en una situació real. Per acabar de comprovar si realment es pot aplicar en un robot real, es dissenyarà un escenari físic amb les seves limitacions d'espai amb certs obstacles, per estudiar el comportament del robot en aquest.

14.1 Implementar el model en el TurtleBot3 Burguer

Per començar, s'ha de comprendre com s'implementarà l'entrenament del robot en un model real. Com es pot observar en el node "turtlebot3_dqn_stage_X" independentment de l'escenari es pot apreciar, hi ha una línia de codi que ens indica que per cada episodi, s'emmagatzema un arxiu .h5 i un .json aquests arxius contenen informació important sobre el model entrenat i s'utilitzen per carregar el model més endavant.

L'arxiu .h5 emmagatzema el pes i l'arquitectura del model entrenat, és a dir, conté tota la informació necessària per poder utilitzar el model directament, incloent-hi els paràmetres de les capes de la xarxa neuronal.

Per altra banda l'arxiu .json serveix per emmagatzemar la configuració de l'arquitectura del model. Proporciona una descripció detallada de les capes del model, la seva configuració i connexions. S'utilitza per carregar l'estructura del model i es combina amb el pes de l'arxiu .h5 per reconstruir el model completament.

Un cop entès que són aquests arxius els podrem localitzar en el següent directori:

"*/home/ubuntu/catkin_ws/src/turtlebot3_machine_learning/turtlebot3_dqn/save_model*
"

Per implementar el model entrenat en el robot real, s'ha de dissenyar un programa en Python on es carreguin els arxius .h5 i .json.

Primer de tot, es genera un nou arxiu .py dins del següent directori.

"*/home/ubuntu/catkin_ws/src/turtlebot3_machine_learning/turtlebot3_dqn/nodes"*

(Si no es vol generar un nou lloc de treball, clar)

Abans de començar a programar, s'obre les propietats de l'arxiu i modifiquem els permisos d'aquests per poder donar permís d'execució del programa, ja que sense això no podrem fer res. S'haurà de veure igual que a la següent figura.

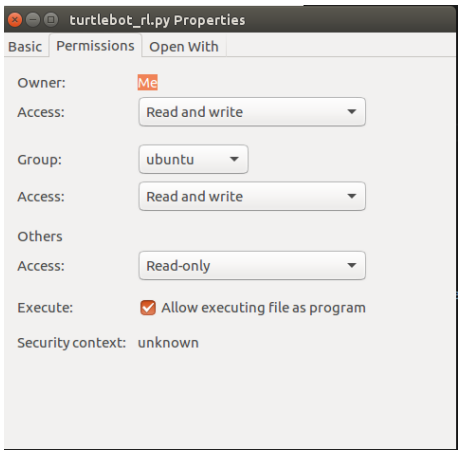


Fig 14.1. Permis d'execució de programa

Font: Elaboració pròpia.

Un cop això, obrim l'arxiu es defineixen certes variables i es carreguen els arxius .h5 i .json.

Seguidament, se li aplica l'estratègia epsilon-greedy per obtenir una conducció autònoma equilibrada en l'exploració i l'explotació de l'agent, el qual pren les decisions basant-se en una política estratega, ja que la idea és que l'agent prengui una acció aleatòria amb una probabilitat epsilon i a partir d'aquesta probabilitat es pren la millor acció fent 1-epsilon, permetent a l'agent explorar noves accions seleccionant-les aleatòriament, en comptes de seleccionar l'acció més bona segons el coneixement del model entrenat.

Un cop entès això, es defineix una variable per aconseguir les lectures del sensor LIDAR per realitzar posteriorment les prediccions del model entrenat i convertir les sortides del model en velocitats lineals i poder moure el robot depenent de les accions preses per aquest a partir de l'estratègia epsilon-greedy. S'haurà també de definir una variable per limitar el rang de les lectures del sensor LIDAR.

Per realitzar una configuració idònia de la implementació del model al robot real, s'ha hagut de buscar un equilibri molt fi entre el valor de la epsilon, per definir si volem més

exploració (accions aleatòries amb noves solucions) o explotació (accions determinades pel model entrenat) per generar una fluïdesa en la conducció autònoma del robot, ja que un equilibri dolent pot causar bucles d'incertesa del robot, a causa de que no sabria triar quina és la opció més òptima. El codi el podreu observar en l'Annex IV.

Guardem el programa, i per compilar en el lloc de treball es segueixen les següents instruccions en el terminal.

$ catkin_make

$source devel/setup.bash

Un cop compilat en el lloc de treball, com es treballa amb una versió de Python 2.7, la qual és antiga i ara la versió més utilitzada és la Python 3.X cap endavant. No podem fer un *"rosrun"* com a tal, s'haurà d'executar anant directament des del directori de Python 2.7 amb la següent instrucció.

$/home/ubuntu/anaconda2/bin/python2.7/home/ubuntu/catkin_ws/src/turtlebot3_machine_learning/turtlebot3_dqn/nodes/turtlebot3_rl.py

On "*turtlebot3_rl"* es el nom que se li ha donat al programa dissenyat.

14.2 Disseny del escenari real

S'ha hagut de dissenyar i implementar un petit escenari per poder observar el comportament del robot en una situació real. El robot es capaç de desenvolupar-se en qualsevol entorn i poder evitar qualsevol obstacle que es trobi en el seu entorn. Però si aquest escenari és semblant al model en el que s'ha entrenat, la seva precisió es encara major.

Per aquest motiu s'ha optat a utilitzar com a material taulons de fusta, ja que els materials refractaris com són en aquest cas els metàl·lics, ocasionen una imprecisió en la recollida de dades, ja que no torna el senyal igual que s'ha enviat. Seguidament, s'ha dissenyat un escenari amb les següent dimensions: 280x240x30cm en total per tal de fer diverses proves en diversos escenaris.

14.3 Comprovació del entrenament en un escenari real

El Turtlebot3 Burguer té per defecte el procés per evitar obstacles, però aquest no el fa en temps real, sinó que ha de preprocessar l'escenari abans per poder determinar els obstacles i preparar un mètode d'evasió. En el nostre cas, el que s'ha aconseguit és entrenar el model en diferents escenaris perquè el robot aprengui a desenvolupar-se per un entorn que no coneix evitant xocar-se amb l'entorn, ja sigui evitant tant obstacles estàtics com mòbils.

Un cop finalitzat aquest entrenament el robot es pot desenvolupar per qualsevol escenari sense tenir la capacitat de tenir-ho estudiat abans, ja que quan detecti algun obstacle amb el LIDAR, automàticament aplicarà el que ha après en l'entrenament previ utilitzant el mètode "Epsilon-Greedy" en una fase més d'explotació i no tant d'exploració.

Per tant, el robot presentava diversos problemes, ja que continuava col·lidint amb obstacles i no actuava com realment es volia. Perfilant el codi estructurat anteriorment per la implementació en el robot, s'ajusten correctament els valors d'èpsilon necessaris i definint els límits del LIDAR per actuar en un angle de 28º com s'ha determinat en l'entrenament. Seguidament, es millora el processament del làser aplicant-hi accions a reaccionar depenent de la distància de l'objecte i es torna a provar el model.

Es pot comprovar que el robot actua bastant ràpid davant diferents objectes, evitant-los i prenent decisions ràpides per evadir obstacles. S'ha provat de tancar-lo en escenaris petits per comprovar el seu funcionament i com s'esperava el robot evita col·lidir amb cap obstacle, aconseguint reaccionar en temps real als objectes presents.

Per corroborar aquesta informació, es va modificar l'escenari cada cert temps, afegint obstacles nous, o traient-los. I com a resultat, el robot els evitava sense cap mena de dubte, verificant que realment s'ha obtingut evitar obstacles en temps real independentment de que l'escenari sigui canviant.

Un aspecte important a comentar, és que el robot en certs aspectes arriba a xocar-se amb certs obstacles, a conseqüència de què aquests no arriben a l'altura del LIDAR, per tant, no són processats, com poden ser cantonades o objectes metàl·lics que refracten el làser diferent, donant "falsa" informació.

15. Conclusions

15.1 Resum de les observacions del projecte

En aquest treball de fi de grau, es va abordar el desafiament de millorar la capacitat de navegació autònoma d'un robot utilitzant l'aprenentatge reforçat. L'objectiu principal va ser implementar i avaluar un algorisme de Reinforced Learning en l'entorn ROS, fent servir el robot Turtlebot3 Burger com a plataforma de proves.

Primer de tot, es va realitzar un estudi profund sobre els diferents tipus d'aprenentatge que hi ha, on es va destacar el Reinforced Learning, ja que es el mètode que més s'adequa a la solució del problema. Seguidament, es va fer un estudi sobre el mètode de Reinforced Learning anomenat DQN on destaca la possibilitat d'aplicar el mètode Epsilon-Greedy en el model a millorar. Per altra banda, s'analitzen les especificacions del Turtlebot3 Burger i les diferents característiques a tenir en compte per la implementació d'aquest en ROS.

En aquest cas no s'ha treballat amb el mètode del QFD, ja que les especificacions tècniques anaven molt lligades als objectius marcats. Aquestes especificacions tècniques van ser determinades en la fase de l'avantprojecte, gràcies a l'estudi previ sobre Machine Learning i el robot Turtlebot3 Burguer, es va obtenir una visió més clara sobre el camí a seguir per determinar els objectius i les especificacions tècniques necessàries per aconseguir el nostre propòsit.

Amb la solució assolida i determinades les necessitats del projecte, s'ha realitzat la viabilitat tècnica, la viabilitat mediambiental i per últim la viabilitat econòmica.

Un cop entès el procés de disseny del model de Reinforced Learning per evadir obstacles, es van definir els diferents softwares a instal·lar com són en aquest projecte, una màquina virtual per instal·lar Ubuntu, seguidament els passos a seguir per instal·lar ROS i les seves dependències. I per la realització es va plantejar quatre escenaris, un sense obstacles, un amb obstacles estàtics, l'altre amb obstacles mòbils i per finalitzar un escenari amb obstacles estàtics i mòbils.

Tal com s'ha pogut observar en els diferents escenaris per cada entrenament, s'ha observat com a mesura que augmenta la complexitat d'aquest, el temps d'aprenentatge augmenta considerablement tenint el primer model sense obstacles après amb cent

episodis, i per altra banda el model híbrid d'obstacles fixos i mòbils a aproximadament mil episodis. S'ha destacat considerablement el consum de processador que requereix realitzar aquests entrenaments.

Per altra banda en el procés d'implementació, és on més s'ha tingut problemes, ja que primerament, s'havia de configurar la IP del robot al router per tenir com un pont de connexions entre l'ordinador i el robot. El primer problema va ser en que el robot ja tenia un router assignat, per tant, quan es volia fer una connexió amb un altre router no es podia connectar. Fins que es va entrar dins de la Raspberry del robot per visualitzar el nom del router al que es trobava connectat i així localitzar-lo. Un cop solucionat aquest problema, comença la fase d'implementació del model simulat al model real. On s'havien generat dos arxius, un .h5 i l'altre .json, els quals emmagatzemen tota l'arquitectura del model entrenat.

Per poder implementar-ho, ens trobem que aquests arxius no poden ser carregats com a tal, per el que s'ha de fer un programa en Python, dissenyar el node i llançar-ho com a node. El problema principal es donava en què s'havia de modelitzar tot el robot fent el preprocessament del LIDAR per la captació dels objectes, definint el rang del làser perquè sigui igual que en procés d'entrenament.

A continuació, un altre inconvenient que es va tenir va ser en qué havíem de determinar un sistema perqué el robot es mogués per un escenari, però amb l'entrenament d'aquest. Per fer-ho es va intentar diferents mètodes, però tots acabaven sent un fracàs. Fins que es va implementar el mètode Epsilon-Greedy el qual es basa en una fase d'explotació en la qual el robot explora sense tenir en compte l'aprenentatge i per altre l'exploració que és a dir, el robot explora però aplicant tot el que ha après. Per tant, s'ha de trobar un valor òptim entre l'explotació i la exploració aplicant un valor d'èpsilon idoni.

Per finalitzar, es va comprovar en un escenari real, amb diferents objectes com a obstacles per visualitzar el comportament d'aquest, es verifica el comportament del robot, verificant que realment, ha aprés a evadir obstacles sense necessitat de tenir mapejat l'entorn, ja que aquest respon en temps real. Per altra banda, cal destacar que en entorns amb pocs graus de llibertat el robot tendeix a col·lidir. També s'hi troba en aquesta situació amb obstacles que no superen l'alçada del LIDAR ja que no es detecten i no en són evitats. Però la funció principal es pot donar per complida.

15.2 Línies futures de treball

Com a línies futures, hi ha un ventall bastant extens dins del món del Machine Learning i la robòtica, ja que aquests sectors cada cop estan creixent d'una forma exponencial.

Si es vol continuar treballant amb el present projecte, com a línies futures, una de les primeres accions a millorar serà en la precisió del robot en la vida real, pel fet que l'actual de tant en tant, depenent de les circumstàncies i l'escenari on es trobi, pot arribar a col·lidir amb algun obstacle i quedar-se atrapat en aquest. Per aquest motiu un punt important que es podria arribar a considerar, seria la capacitat de poder corregir i trobar una solució ràpida i efectiva quan es troba en aquestes circumstàncies.

Aquesta proposta pot portar una solució a un problema que s'ha trobat en tota la fase d'implementació del Machine Learning al Turtlebot 3 Burguer i s'obriria un ventall d'aplicacions noves a les que afegir el prototip.

Per altra banda, un punt també a tenir en compte de cara al futur, podria ser l'anàlisis i detecció d'objectes. Ja que d'aquesta manera es pot identificar els obstacles que s'hi troben sota el LIDAR, aquests al no estar en el rang del làser, no són detectats i causen que el robot xoqui i es quedi atrapat. També pot ser interessant veure en temps real com el robot reconeix i evadeix obstacles com poden ser persones, vehicles, etc.

Per aplicar-hi aquest reconeixement es pot fer amb visió artificial i també amb aquesta nova implementació es podria afegir un mètode d'identificació de semàfors per saber quan creuar la vorera i quan no entre altres noves incorporacions.

Totes aquestes millores estan pensades perquè el robot tingui més fluïdesa en la seva trajectòria, solucionant problemes amb els quals ens trobem avui dia i per altra banda, afegint noves característiques per fer el model més innovador i atractiu.

Bibliografia

[1] Tokio School. (2022, April 27). *La historia del machine learning desde sus comienzos*. Tokio School. https://www.tokioschool.com/noticias/historia-machine-learning/

[2] Nalda, V. (2020, September 29). *Machine Learning: Los orígenes y la evolución*. Future Space S.A. https://www.futurespace.es/machine-learning-los-origenes-y-la-evolucion/

[3] Corporativa, I. (n.d.). *Descubre los principales beneficios del Machine Learning*. Iberdrola. Retrieved February 8, 2023, from https://www.iberdrola.com/innovacion/machine-learning-aprendizaje-automatico

[4] Ikusi. (2022, May 19). *¿Cómo implementar modelos de predicción y Machine Learning dentro de una empresa?* Ikusi. https://www.ikusi.com/mx/blog/como-implementar-modelos-de-prediccion-y-machine-learning-dentro-de-una-empresa/

[5] *¿Qué es el aprendizaje supervisado?* (n.d.). TIBCO Software. Retrieved February 8, 2023, from https://www.tibco.com/es/reference-center/what-is-supervised-learning

[6] *¿Qué es el aprendizaje no supervisado?* (n.d.). TIBCO Software. Retrieved February 8, 2023, from https://www.tibco.com/es/reference-center/what-is-unsupervised-learning

[7] Gonzalez, L. (2020, May 19). *Guía para implementar los modelos de Machine Learning* - Aprende IA. https://aprendeia.com/guia-para-implementar-los-modelos-de-machine-learning/

[8] *UNIDAD 4 REDES NEURONALES - INTELIGENCIA ARTIFICIAL*. (n.d.). Google.com. Retrieved February 8, 2023, from https://sites.google.com/site/mayinteligenciartificial/unidad-4-redes-neuronales

[9] *Redes neuronales*. (n.d.). Github.Io. Retrieved February 8, 2023, from https://ml4a.github.io/ml4a/es/neural_networks/

[10] Banoula, M. (2021, May 26). *What is Perceptron? A Beginners Guide [Updated]*. Simplilearn.com; Simplilearn. https://www.simplilearn.com/tutorials/deep-learning-tutorial/perceptron

[11] *TurtleBot3 burger*. (n.d.). ROS Components. Retrieved February 8, 2023, from https://www.roscomponents.com/en/mobile-robots/214-turtlebot3-burger.html#/courses-no/turtlebot_3_burger_model-burger_intl_

[12] *Enterprise open source and Linux*. (n.d.). Ubuntu. Retrieved February 8, 2023, from https://ubuntu.com/

[13] Robotics, E. (n.d.). *Entendiendo los nodos de ROS · Erle Robotics Gitbook*. Gitbooks.io. Retrieved February 8, 2023, from https://erlerobotics.gitbooks.io/erlerobot/content/es/ros/tutorials/understanding_ros_nodes.html

[14] *Keras: the Python deep learning API*. (n.d.). Keras.Io. Retrieved February 8, 2023, from https://keras.io/

[15] Quiroga, H. (2017). *Anaconda*. Createspace Independent Publishing Platform. From https://www.anaconda.com/

[16] *Canonical. (n.d.). Ubuntu 16.04.7 LTS (xenial Xerus). Ubuntu.com. Retrieved April 19, 2023, from https://releases.ubuntu.com/16.04.7/*

[17] *Download Ubuntu desktop. (n.d.). Ubuntu. Retrieved April 19, 2023, from https://ubuntu.com/download/desktop*

[18] *ROBOTIS e-manual. (n.d.). ROBOTIS E-Manual. Retrieved April 19, 2023, from https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/*