ISSN 1688-2806

Universidad de la República
Facultad de Ingeniería

Algoritmos de aprendizaje automático con aplicación a enjambres de robots

Tesis presentada a la Facultad de Ingeniería de la Universidad de la República por

Leopoldo Agorio

en cumplimiento parcial de los requerimientos para la obtención del título de Magister en Ingeniería Eléctrica.

Director de Tesis
Dr. Juan Andrés Bazerque ..... Universidad de la República

Tribunal
Dr. Miguel Calvo Fullana ... Massachusetts Institute of Technology
Dr. Federico La Rocca ..... Universidad de la República
Dr. Pablo Monzón ..... Universidad de la República

Director Académico
Dr. Juan Andrés Bazerque ..... Universidad de la República

Montevideo
lunes 18 julio, 2022

Algoritmos de aprendizaje automático con aplicación a enjambres de robots, Leopoldo Agorio.

ISSN 1688-2806

Esta tesis fue preparada en LaTeX usando la clase iietesis (v1.1).
Contiene un total de 89 páginas.
Compilada el lunes 18 julio, 2022.
`http://iie.fing.edu.uy/`

All the impressive achievements of deep learning amount to just curve fitting.

Judea Pearl

Any sufficiently advanced technology is indistinguishable from magic

Arthur C. Clarke

Esta página ha sido intencionalmente dejada en blanco.

Agradecimientos

En esta monografía explico múltiples resultados vinculados con la optimización matemática y una rama de aprendizaje automático conocida como Aprendizaje por Recompensas. Tal vez tanto trabajar en esto me forjó una manera de ver el mundo que tiene su filosofía asociada. Nosotros hablamos de optimizar y siempre tenemos una función de objetivo, unas restricciones asociadas, y unas variables que podemos variar en su espacio factible. En el marco de Aprendizaje por Recompensas además, esas variables son la acción a tomar a cada instante por uno o cada uno de los agentes involucrados. La vida tiene lo suyo de optimización multi-agente, y un universo de grados de libertad mucho mayor a los pequeños problemas que planteamos en este trabajo. En la tarea titánica de reducir este gran problema a algo más parecido al caso anterior me han acompañado una cantidad de personas muy valiosas que corresponde agradecer. Como agradecerles a todas sería irrealizable, trataré de escribir algo que refleje el sentimiento, disculpándome de ya por varios de los que involuntariamente me habré olvidado.

En primer lugar agradecer a mi madre Rosario y mi hermano Rodolfo, que han tenido que escuchar ensayos, anécdotas y trancazos asintiendo con la cabeza a pesar de no comprender la tecnicidad. Me han ayudado copiosamente a ordenar la cabeza, la casa, y lo que fuera necesario para poder cumplir con la función objetivo de sacar adelante este trabajo.

Luego a Carla, mi compañera de vida durante un cuarto de su duración, durante el cual nuestras políticas individuales y colectivas han tendido a la convergencia. Ha sido mi referencia y solver de todos los problemas de decisión importantes, y la que me ha acompañado a elegir las acciones que repercutan en trayectorias deseables. Cuando el día solo ha dado recompensas negativas, pensar en volver a nuestro refugio de señal de recompensa es una guía para la acción.

A mis amigos de la 20, el iie y la fing. Por compartir -por voluntad propia o por inducción- los mismos gustos frikis, y ser la compañía más divertida con la que pasar el rato. Por ayudarme en problemas tan variados como el pago de impuestos, la preparación de una clase, el quejarse de básicamente todo, y el organizar vacaciones, despedidas, cumpleaños o eventos con las excusas más insólitas. Ver que a cada año que pasa están todos en un estado con función de valor apreciablemente más alta me provoca satisfacción y entusiasmo por las posibilidades de los ingenieros y académicos orientales.

A Pelu, Emi y Mari, que me acompañan a soñar un día en el que la única función objetivo de la humanidad no sea la maximización desalmada de la ganancia por unos pocos agentes despreciables (en número y actitud). Continuar a

proponerse ese problema sujeto a que la vida pasa, los tiempos y recompensas positivas escasean cada vez más, los inconvenientes y errores se acumulan, y los ánimos tienden a la baja, demuestra una necedad y una demencia que provocan orgullo.

A la red académica y laboral del Departamento y a nuestros colaboradores foráneos. Son los gigantes en cuyos hombros nos hemos apoyado para este trabajo, así que sin ustedes esto no sería posible. Tampoco sería posible sin el apoyo económico de la UdelaR y su beca de posgrados de la CAP, gracias.

Por último pero no menos importante, a Juan, mi tutor, por ayudarme a encontrar la trayectoria óptima para concluir este trabajo en tiempo y forma. A su vez, por la empatía y consideración que me tuvo en todo el proceso, mostrando una calidad humana que el barniz de científico esconde a primera vista. Agradezco también lo mucho que ha expandido mi espacio de estados y acciones factibles tanto dentro como fuera del Uruguay y la Academia y espero ansioso que un día volvamos a coexistir en el espacio de estados.

Agradecer anticipadamente a todos los anteriores y a los futuros acompañantes que tenga en lo que empieza desde acá. Estoy seguro que me ayudaran a encontrar una función objetivo, un espacio de estados donde moverme y unas acciones para optimizarla. Salud.

Abstract

Swarm or multi-agent robotic systems are a growing area of research. To provide wireless infrastructure on demand, it is necessary to deploy a secondary team of robots that guarantee the connectivity of the swarm. In this paper we explain an optimal positioning algorithm for this team of robots, consisting of a convex optimization stage on a probabilistic channel model and a subsequent connectivity maximization stage of a Laplacian graph.

To show the advantage of this mathematical formulation, we carried out both simulations and experiments that were carried out with a fleet of 10 Unmanned Aerial Vehicles (UAV) -assembled and configured by our research group- based on the model DJI Flame-Wheel and equipped with onboard Intel NUC mini-computers and Wi-Fi connectivity. For the experiments carried out, the UAVs established an ad-hoc network through ROS *multi-master* nodes in the Ubuntu 18 operating system.

There is also another family of autonomous algorithms of growing interest known as Reinforcement Learning (RL), in which the control to be applied arises from optimizing a reward signal. In this thesis we study a monitoring problem, formulated from the occupation restrictions of regions to be monitored by one or multiple agents, which leads to an RL problem in which the dual variables act as a reward signal.

To solve the problem in the case of a single agent monitoring several regions, we designed a parameterization through a neural network that processes the primal and dual variables in parallel. With this structural novelty, the network learns to choose navigation policies based on the degree of satisfaction of the constraints, which is observed in real time through the dual variables.

For the multi-agent case, we simulate a simplified version of the problem with a discrete state space and two agents, and by imposing that the agents have distributed policies, we achieve performance comparable to that of a centralized policy.

Esta página ha sido intencionalmente dejada en blanco.

Resumen

Los sistemas robóticos de enjambre o de múltiples agentes constituyen un área de investigación en creciente desarrollo. Para proveer infraestructura inalámbrica a demanda es necesario desplegar un equipo secundario de robots que garanticen la conectividad del enjambre. En este trabajo explicamos un algoritmo de posicionamiento óptimo para este equipo de robots, consistente en una etapa de optimización convexa sobre un modelo de canal probabilístico y una siguiente etapa de maximización de la conectividad de un grafo Laplaciano.

Para mostrar la ventaja de esta formulación matemática, llevamos a cabo tanto simulaciones como experimentos que fueron realizados con una flota de 10 Vehículos Aéreos no Tripulados (UAV por sus siglas en inglés) -ensamblados y configurados por nuestro grupo de investigación- basados en el modelo DJI Flame-Wheel y equipados con mini-computadoras Intel NUC a bordo y conectividad Wi-Fi. Para los experimentos realizados, los UAVs establecieron una red ad-hoc a través de nodos ROS *multi-master* en sistema operativo Ubuntu 18.

Existe a su vez otra familia de algoritmos autónomos de creciente interés conocida como aprendizaje por recompensas o *Reinforcement Learning* (RL), en los que el control a aplicar surge a partir de optimizar una señal de recompensa. En esta tesis estudiamos un problema de monitoreo, formulado a partir de restricciones de ocupación de regiones a monitorear por uno o múltiples agentes, que se lleva a un problema de RL en el que las variables duales actúan como señal de recompensa.

Para resolver el problema en el caso de un único agente monitoreando varias regiones, diseñamos una parametrización por medio de una red neuronal que procesa en paralelo las variables primales y las duales. Con esta novedad estructural, la red aprende a elegir políticas de navegación en función del grado de satisfacción de las restricciones, que se observa en tiempo real a través de las variables duales.

Para el caso de múltiples agentes, simulamos una versión simplificada del problema con un espacio de estados discreto y dos agentes, e imponiendo que los agentes tengan políticas distribuidas logramos un desempeño comparable al de una política centralizada.

Esta página ha sido intencionalmente dejada en blanco.

Tabla de contenidos

Capítulo 1

Introducción

Existe un creciente interés en el desarrollo de sistemas robóticos multi-agente o de enjambre [1, 2]. Los sistemas robóticos de enjambre realizan muchas tareas complejas a través de la coordinación, como cobertura colaborativa de un entorno [3] y consenso distribuido para reunión o formación [4, 5].

Existe a su vez historia y avances recientes en investigación respecto al efecto de los enlaces de comunicación y la capacidad para el enjambre de coordinar [4, 6]. Los canales inalámbricos son difíciles de predecir debido a su caracter aleatorio y fenómenos como el fading y el multicanal [7] que pueden hacer variar la calidad de la señal incluso a corta distancia [8]. Los modelos de telecomunicación existentes pueden ser divididos por la forma en la que modelan los enlaces punto a punto. La versión más sencilla se conoce como modelo de discos y considera a todos los agentes dentro de cierto rango como comunicados [9]. Otros enfoques modelan el canal a partir de la distancia entre robots y típicamente reducen el problema a resolver un grafo Laplaciano[10, 11]. Por último, los modelos probabilísticos, como el que utilizaremos, tratan de modelar la incertidumbre del canal a través de una definición en torno a media y varianza del enlace [12-15].

Varios proyectos, como el Project Loon de Google [16], Connectivity Lab de Facebook[17], o Starlink de SpaceX [18], tienen como objetivo desplegar un grupo de *routers* controlables con el fin de mantener una infraestructura inalámbrica en regiones remotas. En este trabajo, nos interesará un enfoque que busque automáticamente configuraciones para los *routers* que maximicen el desempeño de la red [19].

En este trabajo montamos un sistema de UAVs (Unmanned Aerial Vehicles) capaces de utilizar el algoritmo de ruteo probabilístico desarrollado en [19] para establecer una red de datos ad-hoc sobre 802.11n Wi-Fi. Para esto, se cuenta con un conjunto de agentes llamado equipo de tarea que realizan tareas colaborativamente, mientras otro conjunto de agentes auxiliares llamado equipo de red se despliega para mantener la confiabilidad de la red. Como aporte expandimos lo desarrollado en [19] realizando junto a los autores una batería de experimentos en mundo real poniendo a prueba el algoritmo desarrollado.

Estos experimentos implican un grado de aprendizaje autónomo de los robots en el sentido de que estos corren algoritmos de optimización internamente para

"aprender" las trayectorias que maximizan el *throughput* de red. En esta tesis quisimos ir más allá en el aprendizaje autónomo con UAVs lo que nos llevó a investigar en el área de Reinforcement Learning aplicado a enjambres de robots.

El aprendizaje por refuerzos (Reinforcement Learning - RL) estudia cómo un agente aprende a maximizar su recompensa total esperada al interactuar con un entorno a lo largo del tiempo, generalmente modelado como un proceso de decisión Markoviano (MDP) [20]. Los MDPs tienen aplicaciones muy diversas, desde mantenimiento de carreteras [21], smart grids [22], finanzas [23] y robótica [24]. En los problemas conocidos como Constrained Reinforcement Learning (CRL), además de maximizar la señal de recompensa, se debe cumplir con un conjunto explícito de restricciones. Las restricciones son necesarias para que RL sea aplicable a problemas del mundo real, en donde las restricciones pueden ser más relevantes en la especificación del problema que la propia función objetivo [25, 26].

Existen dos esquemas de solución en RL - los basados en modelo y los sin modelo, según el uso que se haga de un modelo explícito a priori del entorno. Existen soluciones mixtas [27], así como trabajos prometedores de RL sin modelo en espacios continuos [28]. Optamos por una solución sin modelo que parametriza la política a través de una red neuronal de dos capas [29]. Diseñamos esta red neuronal específicamente para nuestro problema, considerando sus variables duales y basándonos en el hecho de que una red neuronal de dos capas es un aproximador funcional universal [30].

A su vez, existe una familia de problemas en la que se cuenta con múltiples agentes que deben ser controlados (Multiple Agent RL - MARL), donde una primer opción sería utilizar una política centralizada, que tome como entrada el estado global del conjunto de agentes y decida las acciones de todos los agentes. Otra alternativa es lo que se conoce como una política distribuida, es decir, cada agente cuenta con una representación individual del estado y a partir de esta debe decidir (exclusivamente) su acción. El estudio de garantías de convergencia en MARL con restricciones es un tema reciente en publicaciones [31].

En particular en este trabajo logramos entrenar una política sin modelo, parametrizada con redes neuronales, para resolver un problema de aprendizaje por refuerzos con restricciones. El problema simulado es de monitoreo de regiones: cada región de interés debe ser cubierta una cierta porción de tiempo. Para una versión simplificada del problema, se ensayó una solución de múltiples agentes.

El trabajo se presenta estructurado de la siguiente manera:

- En el Capítulo 2 se describe el sistema físico de UAVs utilizado así como las herramientas de software y comunicación. A su vez, se recopilan los contenidos desarrollados en [19, 20, 32], necesarios para explicar tanto el algoritmo de ruteo probabilístico en UAVs, como el problema de RL abordado.

- En el Capítulo 3 se describen los experimentos llevados a cabo con el sistema de UAVs, así como las simulaciones computacionales en el sistema de *software in the loop*.

- En el Capítulo 4 se formula el problema de monitoreo con restricciones en su versión distribuida y se explica el esquema de solución propuesto, así como

los resultados obtenidos.

- Por último, en el Capítulo 5 se exponen conclusiones y se reflexiona sobre posibles trabajos a futuro.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 2

Fundamentos de la Tesis

En esta sección detallamos los fundamentos necesarios para abordar los Capítulos 3 y 4 de este documento, referidos al despliegue de un enjambre de UAVs para infraestructura Wi-Fi, y a las simulaciones en un problema de monitoreo reformulado como un problema de ACRL (*Augmented Constrained Reinforcement Learning*) respectivamente.

En particular, en la Sección 2.1 se describe el sistema físico de UAVs ensamblado y configurado, en la Sección 2.2 se detalla el algoritmo de ruteo probabilístico que corre en los UAVs, y en la Sección 2.3 se explican los fundamentos teóricos la teoría de ACRL.

2.1. Fundamentos Tecnológicos de la Tesis

En esta sección explicamos el sistema físico sobre el cual trabajé implementando el algoritmo que se desarrollará en la Sección 2.2. A su vez, expongo las herramientas tecnológicas utilizadas para montar un sistema intercomunicado de múltiples robots aéreos. El sistema en si consta de una flota de 10 UAVs (*Unmanned Aerial Vehicle*) basados en el modelo DJI FlameWheel 450 para cuadricópteros [33].

Cada UAV cuenta con un controlador de vuelo Pixhawk que resuelve lo que llamaremos bajo nivel: comandos de posición -ya sea estabilizarse manteniendo la altura (*position hold*), o moverse a un punto objetivo (*waypoint*)-. Además, cada UAV posee un procesador Intel NUC que opera en lo que llamaremos alto nivel: corre el algoritmo de ruteo que se describe en la Sección 2.2, y define qué comandos de dirección dar a cada UAV. Un UAV ensamblado y funcional se puede ver en la Figura 2.1

Tanto el ensamblaje de los UAVs, la configuración de sensores actuadores y controlador de vuelo y la configuración de los procesadores Intel NUC fueron realizadas en el primer semestre de 2020 en conjunto por Dr. Juan Bazerque, Mag. Mariana del Castillo, Dr. Mikhail Gerasimenko, y quien escribe esta monografía.

Los UAVs forman parte del proyecto *"Monitoring Crops and Livestock in Uruguay"*, dirigido por el Dr. Bazerque en colaboración con Dr. Alejandro Ribeiro, Dr.

Figura 2.1: UAV de la flota. El modelo mecánico es basado en DJI FlameWheel 450, y cuenta con un controlador de vuelo Pixhawk Autopilot comunicado a los sensores y actuadores, y un procesador Intel NUC con conectividad Wi-Fi para la comunicación con los demás UAVs.

Miguel Calvo y Mag. Daniel Mox, de la Universidad de Pennsylvania. Las componentes fueron importadas en este marco, y el ensamblaje de los UAVs se realizó en Uruguay.

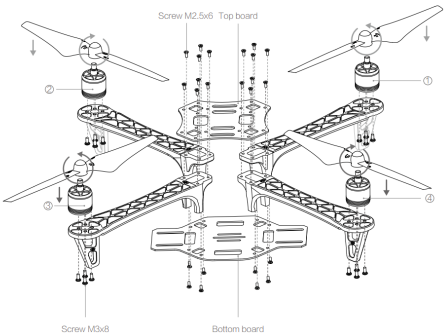
2.1.1. Hardware

Ensamblado mecánico

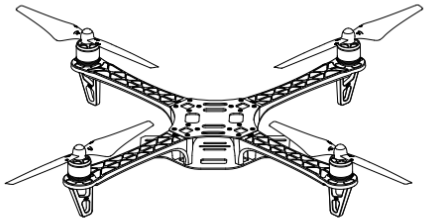
Los cuadricópteros construidos los ensamblamos según lo dispuesto en el modelo DJI FlameWheel 450. En la Figura 2.2 se observan las imágenes del propio manual de usuario, que se acompañaban de instrucciones de armado.

Por otra parte, en la figura Figura 2.3 se ve la presentación de las componentes utilizadas. Además de las componentes presentes en la Figura 2.2, se muestran adaptadores para la conexión del controlador de vuelo Pixhawk, así como la antena de GPS y las componentes de la mini PC basada en el procesador Intel NUC (memoria, disco, antenas WiFi).

Como agregado al ensamblaje, utilizamos una cúpula protectora diseñada por el grupo y realizada en impresora 3D como se muestra en la Figura 2.1.



(a) Piezas necesarias para ensamblado de brazos.



(b) Resultado esperado al ensamblar los brazos.

Figura 2.2: Figuras de referencia de ensamblado de brazos del cuadricóptero, tomadas del manual de usuario de DJI. Dos placas, la inferior y la superior, conteniendo el controlador de vuelo Pixhawk y el procesador respectivamente, se atornillan a los brazos. Cada brazo tiene un motor y un ESC (*Electronic Speed Controller*) que se conecta al Pixhawk. [33]

Sensores, actuadores y control de vuelo

Cada cuadricóptero cuenta con los siguientes actuadores y sensores:

- Cuatro motores Readytosky 2212 con ESC (Electronic Speed Controllers) SimonK 30A que forman el sistema de actuación.
- Giroscopio, acelerómetro y barómetro como parte del controlador de vuelo Pixhawk.
- Una antena de GPS, como sensor de posición global.
- Un sensor óptico LIDAR apuntando verticalmente como sensor de altitud complementario al barómetro.
- Un receptor y transmisor de radio RadioLink.

Todos estos actuadores y sensores se encuentran conectados y son comandados por el controlador de vuelo Pixhawk Autopilot, que utiliza el *firmware* open-source PX4 [35]. Este controlador se encarga de todo el control de bajo nivel del cuadrotor y permite que el cuadricóptero siga instructivas de vuelo. Utilizamos un controlador de vuelo, ya que el interés del trabajo es desarrollar algoritmos de alto nivel que asuman la lógica de posicionamiento resuelta.

El manejo por control remoto (control manual) del cuadricóptero se puede realizar a través de un radioreceptor-transmisor RadioLink. A efectos de este trabajo, el control manual lo usamos para prueba de funcionamiento y para aterrizajes de emergencia, ya que los algoritmos colaborativos se ejecutarán automáticamente utilizando la Intel NUC.

Figura 2.3: Componentes utilizadas para el ensamblado del cuadricóptero. Esta imagen, así como un instructivo de armado y configurado se pueden encontrar en [34].

Equipo informático

Para la intercomunicación entre los distintos UAVs, la comunicación con la *Laptop* que hace de estación terrestre, y la ejecución de los algoritmos que se explican en la Sección 2.2, se utilizó una mini-PC a bordo del cuadricóptero. El procesador es un Intel NUC 7 i3DNB (procesador i3, 7100U Dual Core), provisto con 16 GB de memoria RAM DDR4, y un disco de estado sólido de 500 GB.

Cuenta con una tarjeta de red AX200 y dos antenas Wi-Fi Molex 6E como transceptor de 2.4 GHz, a través de las cuales se realiza toda la comunicación [36].

Cada NUC tiene sistema operativo Ubuntu 18.04, y el framework ROS Melodic (ROS es *Robot Operating System*), un entorno de desarrollo robótico que se explica en mayor detalle en el Apéndice A. Toda la lógica de alto nivel se realiza a través de la librería *rospy*, una librería en Python que hace de interfaz con el *framework* robótico ROS [37].

2.1.2. Herramientas de software y comunicación

Redes de Wi-Fi ad-hoc

Para los experimentos del Capítulo 3 utilizamos una red de Wi-Fi ad-hoc con IP de cada UAV y de la Laptop PC fijas. La red ad-hoc funciona en forma descentralizada prescindiendo de elemento central o punto de acceso. En su lugar, cada dispositivo (UAV) participa en el ruteo encaminando paquetes hacia los otros dispositivos, con una determinación dinámica del ruteo basada en la conectividad de la red y un algoritmo de ruteo (como el descrito en la Sección 2.2).

En la red ad-hoc móvil que implementamos, cada dispositivo (al tener la movilidad asociada a un UAV) puede moverse y cambiar su posición, por lo que cambia constantemente su conexión con los otros dispositivos. Los dispositivos de la red son capaces de encaminar tráfico no relacionado con su propio uso, es decir, actuar como *router* [38].

Multi-Master ROS

El paquete *multimaster_fkie* ofrece una serie de nodos para establecer una red con múltiples ROS *masters*. En un sistema ROS tradicional hay un único ROS *master* que centraliza la información y envía copias de todos los *topics* a todos los nodos como se describe en el Apéndice A. Este esquema probó saturar la red de UAVs rápidamente debido a la cantidad de mensajes inútiles que se compartían. Por este motivo fue necesario utilizar una herramienta que permitiera sincronizar únicamente los tópicos de interés, como las posiciones de los UAVs [39] [40].

2.2. Infraestructura móvil inalámbrica a demanda

En la presente sección, explico el algoritmo desarrollado por nuestros colaboradores en el artículo [19]. Los algoritmos desarrollados en dicho artículo son los que se ejecutan en el sistema de robots aéreos que desplegamos en el marco del trabajo del grupo. Si bien los contenidos de esta sección no son de mi autoría, los incluyo como fundamento teórico para los experimentos y simulaciones que fueron parte de mi trabajo de tesis y que reporto en el Capítulo 3.

El artículo realiza un modelo y descripción matemática de un problema de optimización sobre una infraestructura móvil. El problema en cuestión es el de un conjunto de agentes móviles (en la implementación práctica serán robots aéreos), que se disponen a hacer una tarea (podría ser por ejemplo patrullaje o exploración) para la cual se necesita comunicación a través de una red de datos. El conjunto de los agentes resulta dividido en dos grupos excluyentes, uno que se dedica exclusivamente a la concreción de la tarea y otro que se dedica a mantener la red de datos funcionando.

Para lograr mantener la red de datos funcionando surgen dos preguntas que el artículo se encarga de modelar y responder:

- Dada una configuración espacial de todos los agentes, ¿cómo se deben encaminar los paquetes de la red de datos para tener una comunicación *óptima* en algún sentido?
- ¿Cómo se puede mejorar la configuración espacial de los agentes sobre los que tengo control (aquellos cuyo único cometido es mantener funcionando la red) para garantizar una comunicación *óptima*?

En esta sección se explican los conceptos desarrollados en el artículo, con la siguiente estructura:

En la Subsección 2.2.1 se define el problema de estudio y se detalla el modelo matemático con el que trabajamos. En la Subsección 2.2.2 se formula el problema de optimización que permite definir el ruteo de cada agente. En la Subsección 2.2.3 se comenta el algoritmo propuesto en el artículo para reconfigurar los agentes de red.

2.2.1. Definición del Problema

Considérese un equipo de UAVs colaborando para completar una tarea que requiere comunicación. La propuesta es: en lugar de contar con una red de comunicación inalámbrica convencional que comunique a todo el equipo, se presume que es posible comunicarse con un grupo diferente de robots: el equipo de red (*network team*), que se despliegan para formar una red ad-hoc.

Los UAVs de este equipo de red se posicionan en el ambiente y rutean paquetes de forma tal que el equipo de tarea (*task team*) puede hacer su objetivo sin preocuparse de que sus acciones tengan impacto en su habilidad para comunicarse. El foco de este trabajo es modelar el problema de posicionamiento y ruteo del equipo de red para que el equipo de tarea pueda desempeñarse correctamente.

De manera formal, suponemos que el *task team* está conformado por p UAVs (de ahora en más agentes), y el *network team* por q agentes de modo que $p+q=m$ es el número total de agentes. Llamemos $\mathcal{I}_T(1\dots p)$ e $\mathcal{I}_N(1\dots q)$ a los índices de *task agents* y *network agents* respectivamente (los agentes estarán numerados de 1 a m, e $\mathcal{I}_T(1)$ es el número del primer agente *task*).

Si bien los UAV se mueven en el espacio, su altura permanece fija para los experimentos, por lo que la posición de cada agente está modelada como un punto en el plano $x_i \in \mathbb{R}^2$, y la configuración x se define como el conjunto de las posiciones de los agentes:

$$x = [x_1, x_2, \dots x_m]^T$$

También es posible hablar de la configuración del *task team* $x_T = [x_{\mathcal{I}_T(1)}, x_{\mathcal{I}_T(2)}, \dots x_{\mathcal{I}_T(p)}]^T$ y la configuración del *network team* $x_N = [x_{\mathcal{I}_N(1)}, x_{\mathcal{I}_N(2)}, \dots x_{\mathcal{I}_N(q)}]^T$.

Modelo de canal

Para el modelo de canal se considera una función de tasa de transmisión (*transmission rate*) normalizada $R_{ij}(x_i, x_j) \in [0,1]$ para cada par de agentes localizados en las posiciones x_i y x_j. Esta función de *transmission rate* es una variable aleatoria representando la tasa de transmisión de un enlace dado. Se caracteriza a partir de su media $\bar{R}_{ij}$ y su varianza $\tilde{R}_{ij}$ para las cuales se usa el siguiente modelo [41]:

$$\bar{R}_{ij}(x_i, x_j) = \operatorname{erf}\left(\sqrt{\frac{P_{L_0}}{P_{N_0}} \|x_i - x_j\|^{-n}}\right) \tag{2.1}$$

$$\tilde{R}_{ij}(x_i, x_j) = \frac{a\,\|x_i - x_j\|}{b + \|x_i - x_j\|} \tag{2.2}$$

donde $\operatorname{erf}(x) = \frac{1}{\sqrt{\pi}} \int_{-x}^{x} e^{-t^2} dt$ es la función de error Gaussiano, y P_{L_0},P_{N_0},n,a,b son parámetros numéricos del modelo que tomaremos por dados.

Modelo de flujo de datos

Para que el *task team* cumpla su objetivo, necesitará transmitir flujos de información desde agentes fuente a agentes destino a través del *network team*. A cada flujo de información se lo indexa con $k \in \{1, 2, \dots K\}$.

En cada flujo de información existe un agente fuente, notado por $S_k \in \mathcal{I}_T$ y un agente destino $D_k \in \mathcal{I}_T$. Cada flujo necesita una tasa mínima de comunicación para ser transmitido correctamente. Notamos por m_i^k la tasa mínima que requiere inyectar el agente i para el flujo k. Para cualquier agente de red, que no contribuye paquetes nuevos y solo actúa de *relay* $m_i^k = 0$.

Los paquetes son encaminados en la red según las variables de ruteo $\alpha_{ij}^k \in [0,1]$ que representan la fracción de tiempo que el agente i utiliza transmitiendo datos asociados al flujo k hacia el agente j. Se nota por α al conjunto de las variables de ruteo. Las variables de ruteo deben satisfacer:

- $\sum_{j,k} \alpha_{ij}^k \leq 1$ y $\sum_{i,k} \alpha_{ij}^k \leq 1$: cota de transmisión y recepción respectivamente.

- Si la fuente de transmisión es el (*task*) agente i, entonces $\alpha_{ji}^k = 0 \ \forall j$ i.e, no se entrega data al agente fuente.
- Si el destino de transmisión es el (*task*) agente j, entonces $\alpha_{ji}^k = 0 \ \forall i$ i.e, el agente destino no transmite información.

Las ecuaciones de flujo quedan dadas por las siguientes ecuaciones que definen el margen de transmisión, o *throughput*:

$$b_i^k(\alpha, x) = \underbrace{\sum_{j=1}^{n} \alpha_{ij}^k R_{ij}(x_i, x_j)}_{\text{flujo saliente}} - \underbrace{\sum_{j=1}^{n} \alpha_{ji}^k R_{ij}(x_i, x_j)}_{\text{flujo entrante}} \tag{2.3}$$

Es decir, $b_i^k(\alpha, x)$ representa la transmisión original efectiva: en el caso de un agente que está oficiando de fuente para un mensaje y de *relay* para otro mensaje, el flujo entrante tiene el único propósito de ser redirigido, mientras que el flujo saliente corresponde al flujo que entró y será redirigido, más el flujo original que se desea transmitir como fuente. De esta forma el descuento "flujo saliente – flujo entrante" resulta en la cantidad de información nueva que aportó el agente i. Ya que los *rates* R_{ij} son variables aleatorias, se recurre a una representación estadística y la media y varianza de las ecuaciones de flujo resultan dadas por:

$$\bar{b}_i^k(\alpha, x) = \sum_{j=1}^{n} \alpha_{ij}^k \bar{R}_{ij}(x_i, x_j) - \sum_{j=1}^{n} \alpha_{ji}^k \bar{R}_{ij}(x_i, x_j) \tag{2.4}$$

$$\tilde{b}_i^k(\alpha, x) = \sum_{j=1}^{n} \left(\alpha_{ij}^k\right)^2 \tilde{R}_{ij}(x_i, x_j) + \sum_{j=1}^{n} \left(\alpha_{ij}^k\right)^2 \tilde{R}_{ij}(x_i, x_j) \tag{2.5}$$

En estas condiciones $\bar{b}_i^k$ denota el margen de transmisión esperado (*expected rate margin*) del agente i para el flujo k y $\tilde{b}_i^k$ la confianza en ese margen esperado[1].

Es posible pensar una primer condición de requerimiento de tasa mínima de transmisión exigiendo margen de transmisión medio de al menos m_i^k:

$$\bar{b}_i^k \geq m_i^k \tag{2.6}$$

2.2.2. Ruteo Probabilístico y problema de optimización

Ya que el margen de transmisión es una variable aleatoria, se puede formular un requisito de servicio diferente al de la Ecuación 2.6 en un sentido probabilístico, esto es:

$$\mathbb{P}\left[b_i^k(\alpha, x) \geq m_i^k\right] \geq 1 - \epsilon_k \tag{2.7}$$

Donde ϵ_k es el riesgo de que se rompa la restricción, o $1 - \epsilon_k$ es la confianza en que la restricción se respete. En estas condiciones, la pareja $\left(m_i^k, 1 - \epsilon_k\right) \stackrel{\text{def}}{=} \text{qos}_{i,k}$

[1] El agente i para el cual se calcula el margen de transmisión puede corresponder o bien a un agente fuente o bien a un agente de red, es decir $i \in S_k \cup \mathcal{I}_N$.

definen la especificación de calidad de servicio (*quality of service* - qos) para el agente i en el flujo k.

Asumiendo que el margen de transmisión es normalmente distribuido, (2.7) implica:

$$\frac{\bar{b}_i^k(\alpha, x) - m_i^k}{\sqrt{\tilde{b}_i^k(\alpha, x)}} \geq \Phi^{-1}(1 - \epsilon_k) \tag{2.8}$$

con Φ^{-1} la distribución acumulativa Gaussiana $(0, 1)$. El primer objetivo es, dada una configuración x, encontrar el mejor conjunto de variables de ruteo α que satisfagan (2.8). Esto quiere decir que para esta parte la configuración x es fija, y se optimiza exclusivamente sobre α. En particular nos interesa la solución en la cual se alcanza el más alto margen de transmisión. A estos efectos se introduce la variable de *slack* no negativa s y se formula el siguiente problema de optimización [19]:

$$\begin{aligned} \underset{\alpha, s}{\text{maximizar}} \quad & s \\ \text{sujeto a :} \quad & \frac{\bar{b}_i^k(\alpha, x) - m_i^k - s}{\sqrt{\tilde{b}_i^k(\alpha, x)}} \geq \Phi^{-1}(1 - \epsilon_k) \quad \forall i, k \\ & \alpha \in A \\ & s \geq 0 \end{aligned} \tag{2.9}$$

Donde A representa el conjunto de restricciones de ruteo, a saber: $\alpha_{ij}^k \in [0, 1]$, $\sum_{i,k} \alpha_{ij}^k \leq 1$, $\sum_{j,k} \alpha_{ij}^k \leq 1$, $\alpha_{ij}^k = 0$ para $i \in \mathcal{I}_T \setminus S_k$, $j \in \mathcal{I}_T \setminus D_k$ (los *task agents* fuente no reciben, los *task agents* destino no transmiten). Incrementar la variable de slack s es equivalente a incrementar el margen con el cual se satisface la restricción probabilística de (2.6). Esto es posible incrementando el margen esperado $\bar{b}_i^k$ (priorizando transmitir por canales con alto $\bar{R}_{ij}$) y disminuyendo la varianza $\tilde{b}_i^k$ por medio de separar rutas en múltiples agentes. En el Apéndice B se lleva el problema a un equivalente que tiene la notación necesaria para que el solver *cvxpy* pueda resolverlo.

2.2.3. Posicionamiento de los agentes de red

El problema de definir x (posicionamiento) se formulará como un problema meramente geométrico y desacopladao del problema de ruteo, siguiendo el desarrollo de [42]. El objetivo es maximizar la conectividad algebraica de un Laplaciano que depende de la posición. Para esto se fabrica un grafo cuyos nodos son los agentes, y cuyas aristas son el *transmission rate* medio dado por (2.1). Esto es, la matriz de adyacencias del grafo va dada por

$$A_{ij} = \bar{R}_{ij}(x_i, x_j) = \operatorname{erf}\left(\sqrt{\frac{P_{L_0}}{P_{N_0}} \|x_i - x_j\|^{-n}}\right) \tag{2.10}$$

por lo que el Laplaciano asociado al grafo es el que viene dado por:

$$L = \operatorname{diag}(A\mathbb{1})\mathbb{1} - A \tag{2.11}$$

donde $\mathbb{1}$ representa el vector de 1s. Es importante notar que bajo esta definición, el vector $\mathbb{1}$ es un vector propio de L con valor propio nulo. El siguiente valor propio más pequeño, se conoce como conectividad algebraica o valor propio de Fiedler y da una idea de la interconexión del grafo. Para el caso particular de un grafo formado por dos (o más) grupos disconexos, el valor propio de Fiedler es también nulo [43][44].

Ya que la Ecuación 2.10 es no convexa en x, se linealiza alrededor de la configuración de red en el paso k:

$$A_{ij} \approx \bar{R}_{ij}\Big|_{x_i(k),x_j(k)} + \nabla_{x_i}\bar{R}^T\Big|_{x_i(k),x_j(k)}(x_i - x_i(k)) + \nabla_{x_j}\bar{R}^T\Big|_{x_i(k),x_j(k)}(x_j - x_j(k))$$

Con las lineas verticales simbolizando que la función $\bar{R}$ y sus gradientes se evalúan en el punto $x_i(k), x_j(k)$. La linealización tendrá validez en un entorno Δ de $x(k)$, es decir, el siguiente punto $x(k+1)$ debe tener restringida su distancia a $x(k)$. En estas condiciones, se puede formular el siguiente problema de optimización convexa para la conectividad (en términos de Fiedler) de la red:

$$\begin{aligned}
&\underset{x\in\mathbb{R}^{2n},\gamma\in\mathbb{R},A\in\mathbb{R}^{n\times n},L\in\mathbb{R}^{n\times n}}{\text{maximizar}} \quad \gamma \\
&\text{sujeto a :} \\
&P^T L P \succcurlyeq I\gamma \\
&L = \text{diag}(A\mathbb{1}) - A \\
&A_{ij} = \bar{R}_{ij}\Big|_{x(k)} + \nabla_{x_i}\bar{R}^T\Big|_{x(k)}(x_i - x_i(k)) + \nabla_{x_j}\bar{R}^T\Big|_{x(k)}(x_j - x_j(k)) \\
&|x - x(k)| \leq \Delta
\end{aligned} \tag{2.12}$$

Donde P es una base ortogonal de $\mathbb{1}^{\perp}$, por lo que la primer restricción es que el menor valor propio supere el parámetro γ, que a su vez se está maximizando. Las siguientes restricciones son las definiciones de L y A_{ij}, y el rango de validez de la aproximación, respectivamente. A efectos prácticos, Δ se asume cubierto al estar repitiendo la optimización en cada iteración del loop principal del programa.

El problema planteado en (2.12) es un problema convexo, al presentar únicamente restricciones convexas, por lo que tiene un máximo global. Definiendo $x(k+1) = x_*$ (es decir el x que maximiza (2.12)), se obtiene la regla de reposicionamiento que utilizamos.

2.3. Aumento de estados en aprendizaje con restricciones

En la presente sección, explicamos el algoritmo desarrollado en el artículo [32]. Las definiciones previas necesarias, sobre aprendizaje por refuerzos clásico se encuentran en el Apéndice C. Con este trabajo como base, desarrollé los algoritmos de aprendizaje distribuido que detallo en el Capítulo 4.

El trabajo formula la teoría de aumento de estados (*state augmentation*) para problemas de aprendizaje por refuerzo con restricciones (*constrained reinforcement learning* o CRL de ahora en más). En un problema de CRL, se tiene una serie de umbrales que se deben alcanzar (restricciones) a través de la acumulación de ciertas recompensas (*rewards*).

En varios de estos problemas, y en particular en los problemas de monitoreo que estudiaremos, no es posible obtener una política óptima a partir de entrenar sobre una combinación lineal de las recompensas, por lo que tanto los métodos de *regularized RL*, o de Primal-Dual fallan [32].

El método explicado en [32] define un estado aumentado que incorpora los multiplicadores de Lagrange, y actualiza dinámicamente estos multiplicadores a partir de una regla inspirada en los métodos Primal-Dual.

Bajo un conjunto de hipótesis, este método permite con casi seguridad tomar acciones que se distribuyen como la política óptima [32].

2.3.1. Introducción al problema de monitoreo

Nuestro problema insignia del tipo de soluciones para los que es útil ACRL es el monitoreo de regiones. En este problema, el agente tiene como tarea monitorear varias regiones del entorno, es decir, pasar una fracción de tiempo especificada en cada región de interés. Las recompensas son valores unitarios en caso de encontrarse en una de las regiones deseadas.

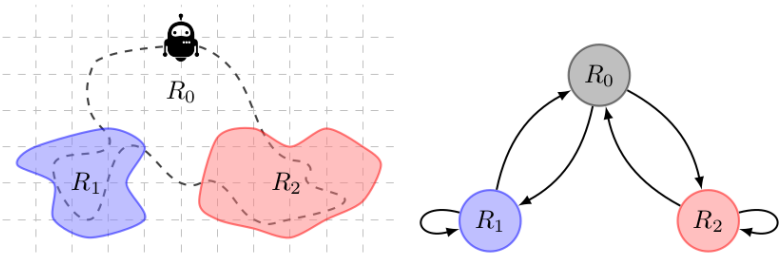


Figura 2.4: Problema de monitoreo. El agente recibe recompensas R_1 si se encuentra en la zona azul y R_2 de encontrarse en la zona roja, y recompensa nula en la región intermedia R_0. Se puede formular en un entorno continuo (izquierda), o equivalentemente como un MDP discreto (derecha). Imagen extraída de [32] con permiso de los autores.

2.3.2. Formulación matemática de CRL

Sea $t \in \mathbb{N}$ un índice temporal, y $\mathcal{S} \subset \mathbb{R}^n$, $\mathcal{A} \subset \mathbb{R}^d$ conjuntos compactos que denotan los espacios de estado y acción respectivamente. El agente queda descrito por un MDP (*Markov Decision Process*) con probabilidades de transición $p(s_{t+1} \mid s_t, a_t)$, y toma sus acciones de una política $\pi(a \mid s)$. Las acciones que toma el agente se recompensan según una serie de funciones de *reward* $r_i : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ con $i = 0, \ldots, m$, por lo que en un tiempo t, la recompensa i toma la realización $r_i(s_t, a_t)$. El interés está en los *rewards* que el agente acumula en el tiempo, por lo que para cada recompensa i, se tiene una función de valor asociada definida por:

$$V_i(\pi) \overset{\text{def}}{=} \lim_{T\to\infty} \frac{1}{T}\mathbb{E}_{s,a\sim\pi}\left[\sum_{t=0}^{T} r_i(s_t, a_t)\right] \tag{2.13}$$

El problema (primal) de CRL será el de maximizar una función de valor *objetivo* (r_0), sujeto a restricciones dadas por especificaciones para las restantes m restricciones. Definimos entonces el problema primal de aprendizaje por recompensas con restricciones (PCRL de ahora en más) como:

$$\begin{aligned} \max_{\pi} \quad & \lim_{T\to\infty} \frac{1}{T}\mathbb{E}_{s,a\sim\pi}\left[\sum_{t=0}^{T} r_0(s_t, a_t)\right] \\ \text{s.a} \quad & \lim_{T\to\infty} \frac{1}{T}\mathbb{E}_{s,a\sim\pi}\left[\sum_{t=0}^{T} r_i(s_t, a_t)\right] \geq c_i,\ i = 0, \ldots, m \end{aligned} \tag{2.14}$$

donde $c_i \in R$ es la i-ésima especificación. Notaremos por $\pi^* \subset \Pi^*$ a cualquier política que obtenga el costo óptimo $V_0(\pi^*)$ y satisfaga todas las restricciones.

En el caso del problema de monitoreo, la función objetivo será 0, y un ejemplo de especificaciones posibles podría ser $(c_1, c_2) = (0.4, 0.3)$, que indican que el agente deberá permanecer el 40 % del tiempo en la región azul y el 30 % en la región roja. Cualquier política que mantenga al agente en cada región los tiempos especificados será una política óptima.

2.3.3. Formulación dual

El problema dual o recíproco de PCRL (al que llamaremos RCRL) surge a partir de la definición del Lagrangeano, que será la nueva función objetivo:

$$\Pi(\lambda) = \operatorname*{argmax}_{\pi} \mathcal{L}(\pi, \lambda) \overset{def}{=} \operatorname*{argmax}_{\pi} V_0(\pi) + \sum_{i=1}^{m} \lambda_i \left(V_i(\pi) - c_i\right) \tag{2.15}$$

Este nuevo problema sin restricciones, se puede resolver con algoritmos estándar de RL, definiendo como *reward* dados los multiplicadores λ:

$$r_\lambda(s_t, a_t) = r_0(s_t, a_t) + \sum_{i=1}^{m} \lambda_i \left(r_i(\pi) - c_i\right) \tag{2.16}$$

Esto permite reescribir la Ecuación 2.15 como un maximizador de la función de valor $V(\pi, \lambda)$ asociada a r_λ:

$$\Pi(\lambda) = \underset{\pi}{\operatorname{argmax}}\, V(\pi, \lambda) = \underset{\pi}{\operatorname{argmax}} \lim_{T\to\infty} \frac{1}{T} \mathbb{E}_{s,a\sim\pi} \left[\sum_{t=0}^{T} r_\lambda(s_t, a_t) \right] \tag{2.17}$$

El inconveniente con esta formulación es qué coeficientes λ_i logran mantener cerca el problema RCRL con el PCRL. Para lograr esto se propone un modelo adaptivo para los λ_i.

Para esto, introducimos un índice de iteración k, un paso η y un tamaño de época o *rollout* T_0. Los multiplicadores se actualizan según la iteración:

$$\lambda_{i,k+1} = \left[\lambda_{i,k} - \frac{\eta}{T_0} \sum_{t=kT_0}^{(k+1)T_0-1} (r_i(s_t, a_t) - c_i) \right] \tag{2.18}$$

donde las acciones $a_t \sim \pi(a_t|s_t, \lambda_k)$ surgen de una política $\pi(\lambda_k) \in \Pi(\lambda_k)$, es decir, una política óptima para λ_k. En cada época k de duración T_0, el multiplicador λ_i se actualiza según si el *reward* acumulado superó el c_i especificado o no. Si el *reward* acumulado excede la especificación c_i, el multiplicador λ_i se reduce, y si está por debajo el multiplicador se incrementa.

La Ecuación 2.18 no es otra cosa que un descenso por gradiente estocástico respecto a la función dual (Ecuación 2.15). Este hecho se utiliza en el artículo [32] para probar que si se verifica:

- Factibilidad del problema primal.
- *Rewards* acotados.
- Hipótesis 3 sobre comportamiento del MDP de [32].

entonces la elección de acciones dada por $a_t \sim \pi(a_t|s_t, \lambda_k)$ y la actualización de multiplicadores dada por Ecuación 2.18 logran factibilidad para el problema primal con probabilidad 1, es decir:

$$\lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T} r_i(s_t, a_t) \geq c_i \text{ casi seguramente} \tag{2.19}$$

2.3.4. Augmented CRL

El algoritmo de aumento de estados en aprendizaje profundo con restricciones (ACRL), reinterpreta λ_k como parte del espacio de estados. Esto es a partir de notar que la Ecuación 2.18 define un proceso Markoviano al depender λ_{k+1} exclusivamente de (λ_k, s_k, a_k)

$$\lambda_{k+1} \sim p(\lambda_{k+1} \mid \lambda_k, s_k, a_k) \tag{2.20}$$

donde s_k y a_k denotan el conjunto de s_t, a_t para $t = kT_0, \ldots, (k+1)T_0 - 1$.

Esta interpretación define un MDP aumentado cuyo estado es el par (s_t, λ_k) (notar que s_t se actualiza a una frecuencia T_0 veces mayor que λ_k).

Los resultados de la Subsección 2.3.3 y esta redefinición dan lugar a los algoritmos siguientes:

Salida: Política entrenada π^*_θ

1 Tomar una muestra s, λ del espacio de estados aumentado;
2 Construir los *rewards* aumentados para λ (ec 2.16);
3 Usar algún algoritmo de RL para entrenar los parámetros:

$$\theta^* = \underset{\theta}{\text{argmax}} \lim_{T\to\infty} \frac{1}{T} \mathbb{E}_{s,a\sim\pi} \left[\sum_{t=0}^{T} r_\lambda(s_t, a_t) \right]$$

Algoritmo 1: *Loop* de entrenamiento para ACRL.

Entrada: Política π^*_θ, paso η_λ, especificaciones c_i, *rollout* T_0
Salida : Trayectorias (s_t, a_t) factibles

1 Inicializar, s_0, λ_0;
2 **for** $k = 0, 1, \ldots$ **do**
3 Tomar T_0 pasos según $a_t \sim \pi^*_\theta(a_t|s_t, \lambda_k)$;
4 Actualizar λ_k según dinámica dual (ec 2.18) ;

Algoritmo 2: *Loop* de ejecución para ACRL.

Es decir, se separa en una etapa de entrenamiento y de ejecución. En la etapa de entrenamiento, se sortean aleatoriamente estados aumentados y recorren trayectorias a partir de estas muestras sorteadas, obteniendo recompensas y actualizando los parámetros por medio de *policy gradient*. En particular, los multiplicadores duales quedan estáticos trayectoria a trayectoria del sorteo ya que la acción del agente solo hace evolucionar el estado primal.

En el *loop* de ejecución en cambio, se utiliza la política aprendida para hacer evolucionar el estado primal del agente, mientras que la dinámica dual del problema hace evolucionar a su vez el estado dual.

Capítulo 3

Despliegue de un enjambre de UAVs para infraestructura Wi-Fi

3.1. Introducción

En este capítulo explico mi contribución original en el marco del proyecto *"Monitoring Crops and Livestock in Uruguay"* haciendo uso del enjambre de UAVs descrito en el Sección 2.1 del cual ensamblé tres unidades.

Los UAVs se utilizan para correr el algoritmo de optimización en redes detallado en la Sección 2.2, haciendo uso de la formulación del problema como un SOCP según las reglas de DQCP[45] (ver Apéndice B) y la biblioteca especializada de optimización convexa *cvxpy*, de lenguaje *Python* [46]. La biblioteca de *cvxpy* es utilizada en paquetes de ROS desarrollados en *rospy* (ver Apéndice A).

El experimento descrito en este capítulo muestra la ventaja de usar agentes de red móviles comparando con una contraparte de menor desempeño con agentes de red fijos.

3.2. Descripción del escenario

En el escenario planteado de tres agentes, que se ilustra en la Figura 3.1, dos de ellos se comportan como agentes de tarea (TA por sus siglas en inglés), uno de ellos manteniendo una trayectoria dada y el otro permaneciendo inmóvil. En el experimento de campo la trayectoria es en forma de línea recta, mientras que en la simulación es una trayectoria en forma arbitraria, que logra que la distancia entre los TAs aumente y disminuya repetidas veces.

El restante es un agente de red (NA por sus siglas en inglés), que se coloca en una posición intermedia para mejorar la conectividad entre los dos TAs según dos posibilidades:

1. Permanece estático manteniendo su altitud en el lugar del que despegó. A esta opción la llamamos de NA fijo.

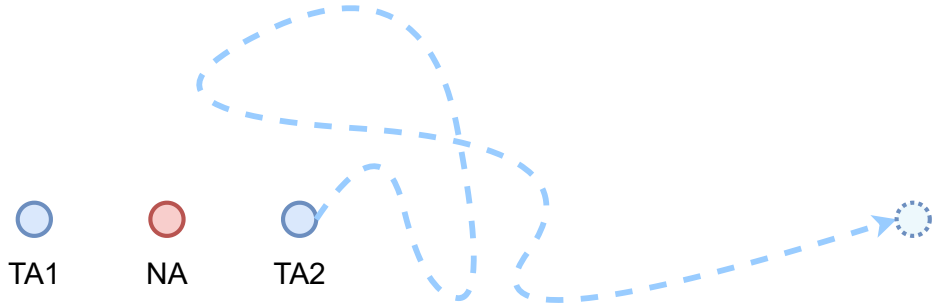


Figura 3.1: Escenario planteado con tres agentes. Los agentes representados en azul son agentes de tarea (TA - *task agents*), mientras que el coloreado en rojo es agente de red (NA - *network agents*). Uno de los agentes de tarea permanece en reposo (TA1), mientras que el restante realiza una trayectoria arbitraria variando su distancia a TA1. Del NA -que se encarga de intercomunicar a los TA- se evaluarán dos modos de operación, un modo estático, y un modo con reposicionamiento dinámico.

2. Se reconfigura según el algoritmo de la Subsección 2.2.3. A esta opción la llamamos de NA dinámico o móvil.

3.3. Simulación SITL con RVIZ

Previo a los experimentos en campo, se realizó una simulación del escenario en el software RVIZ (interfaz *software in the loop* vinculada con ROS), como se ilustra en la Figura 3.2, en la que el TA1 permanece estático y el TA2 recorre una curva en forma de "onda cuadrada" que lo aleja $40m$ del agente estático. En esta simulación se utilizó el modelo de desempeño del canal según Subsección 2.2.1 para calcular el *throughput* medio y su varianza en cada posición del escenario:

$$\bar{R}_{ij}(x_i, x_j) = \operatorname{erf}\left(\sqrt{\frac{P_{L_0}}{P_{N_0}} \|x_i - x_j\|^{-n}}\right)$$

$$\tilde{R}_{ij}(x_i, x_j) = \frac{a\,\|x_i - x_j\|}{b + \|x_i - x_j\|}$$

Los valores utilizados para el modelo se muestran en la tabla 3.1:

P_{N_0}	$-70dBm$
P_{L_0}	$-48dBm$
n	2.52
a	0.2
b	0.6

Tabla 3.1: Valores utilizados para el modelo de canal en la simulación RVIZ, coincidentes con el modelo de canal utilizado en [19]. Se eligen los valores de forma de ajustarse a curvas genéricas de intensidad de señal y tasa de error de bits [41].

Por otra parte, la especificación de la restricción, también llamado QoS (*quality of service*) se realiza especificando ϵ y m en la ecuación (2.8):

$$\frac{\bar{b}_i^k(\alpha, x) - m_i^k - s}{\sqrt{\tilde{b}_i^k(\alpha, x)}} \geq \Phi^{-1}(1 - \epsilon_k)$$

En este caso se optó por $\epsilon = 0.3$ y $m = 0.2$ que representan un requerimiento de una transmisión de al menos un 20 % de la máxima posible con una probabilidad $1 - \epsilon$ del 70 %. Reescribiendo la ecuación anterior, se llega a la expresión:

$$B_i^k = \bar{b}_i^k(\alpha, x) - \sqrt{\tilde{b}_i^k(\alpha, x)}\Phi^{-1}(1 - \epsilon_k) \geq m_i^k + s \tag{3.1}$$

Es decir: dado que s es una variable de slack, se busca que el término izquierdo de (3.1) supere por el mayor margen posible el *throughput* mínimo especificado m_i^k. B_i^k representa una calidad de desempeño que pondera negativamente la varianza de la transmisión, y es útil para comparar contra el valor especificado de m_i^k.

Los resultados de la simulación se observan en la Figura 3.3, donde se grafica tanto el *throughput* medio $\bar{b}$ para la configuración dada, como B_i^k. Para el caso del agente de red fijo, en un momento de la trayectoria los agentes de tarea se encuentran tan alejados que la comunicación se rompe, mientras que en el caso de reconfiguración dinámica se logra cumplir con los requisitos especificados durante la duración completa de la simulación.

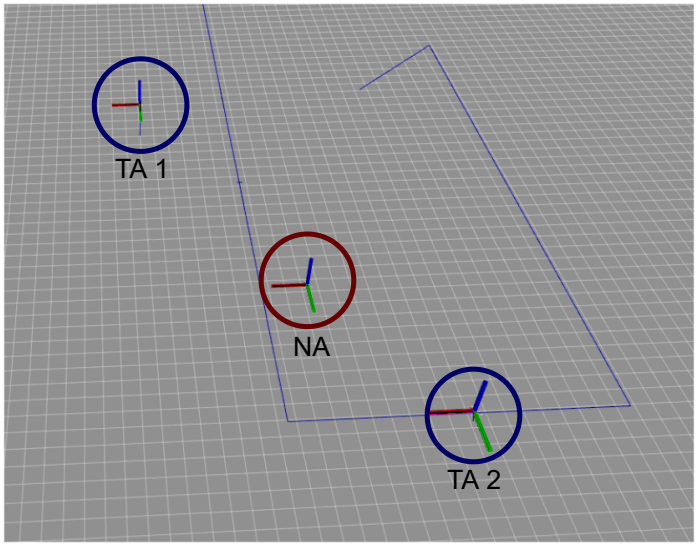
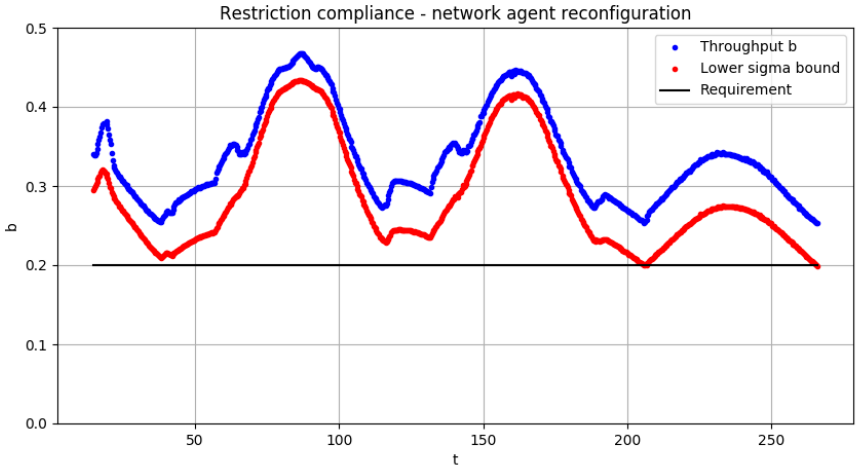


Figura 3.2: Simulación computacional SITL con un agente de tarea estático (TA1), un agente de tarea siguiendo la ruta trazada en azul (TA2), y un agente de red (NA) en una posición intermedia.

Throughput para la simulación con posicionamiento fijo del agente de red.



Throughput para la simulación con posicionamiento móvil (*moving*) del agente de red.

Figura 3.3: Resultados para la simulación realizada en RVIZ. Para el cálculo del *throughput* se utilizó el modelo de canal dado por las ecuaciones (2.1) y (2.2) y con los parámetros de la Tabla 3.1, mientras que para la especificación de restricciones se tomaron los valores $m_i^k = 0.2$ y $\epsilon = 0.3$. El *throughput* mínimo requerido de 0.2 se grafica en trazo negro, el *throughput* instante a instante dado por (2.4) se grafica en trazo azul, y el *throughput* ponderado con el descuento dependiente de $\tilde{b}$ y ϵ explicado en la ecuación (3.1), B_i^k se dibuja en trazo rojo. La especificación de restricciones se reduce a que la línea roja permanezca por encima de la línea negra, situación que solo se cumple la totalidad del tiempo cuando se utiliza el posicionamiento dinámico del agente de red.

3.4. Posicionamiento de las antenas

Un detalle de diseño para replicar nuestros experimentos es el del posicionamiento de las antenas Wi-Fi de la Intel NUC. En nuestros UAV utilizamos un par de antenas (transceptor) Wi-Fi Molex 6E de 2.4 GHz [36], como se muestran en la Figura 3.4.

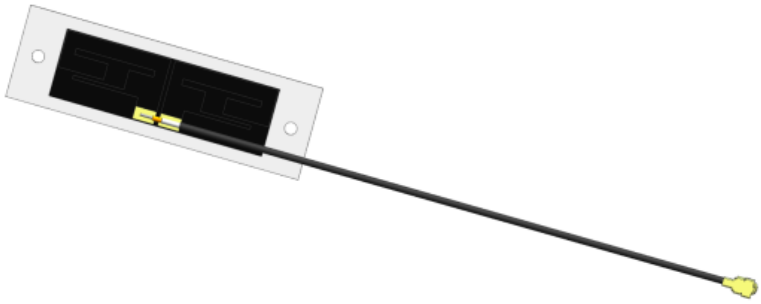


Figura 3.4: Antena de Wi-Fi utilizada. Los conectores UFL se conectan en un chip Wi-Fi conectado en la Intel NUC, y el transmisor/receptor de la antena, se pega a una pata del UAV. Pegar la antena en un brazo del UAV resulta en un desempeño subóptimo de la comunicación como se explicará en esta sección.

Al ensamblar el enjambre de UAVs con el que ahora disponemos, en una primera prueba colocamos las antenas en brazos opuestos del UAV.

Estudiando el patrón de radiación expresado en la Figura 3.5, se concluye que en caso de colocar las antenas en los brazos del UAV, el plano paralelo al piso es el plano ZY o el ZX referido a la antena, y existen direcciones en que la potencia transmitida decae hasta $15dB$, e incluso un punto ciego de potencia nula. Si en su lugar se opta por pegar la antena en una pata del UAV -alineando el eje Z de la antena con la vertical- se obtiene la mejor respuesta angular: una respuesta esencialmente homogénea para todas las direcciones en un plano paralelo al piso.

Siendo que los UAVs se mueven paralelos al piso y pueden girar su posición respecto a un eje vertical a medida que se mueven, un punto ciego en una dirección angular puede hacer caer la comunicación entre UAVs. En particular, la figura Figura 3.6 muestra datos de *throughput* obtenidos en una configuración con las antenas en los brazos, en los que se ve una serie temporal con fluctuaciones, y una falta de correlación en el gráfico de *throughput* contra distancia.

Debido al mal desempeño observado, y a lo concluido a partir de la Figura 3.5, se optó por reubicar las antenas en las patas de los UAV, como se ilustra en las Figura 3.7 y 3.8.

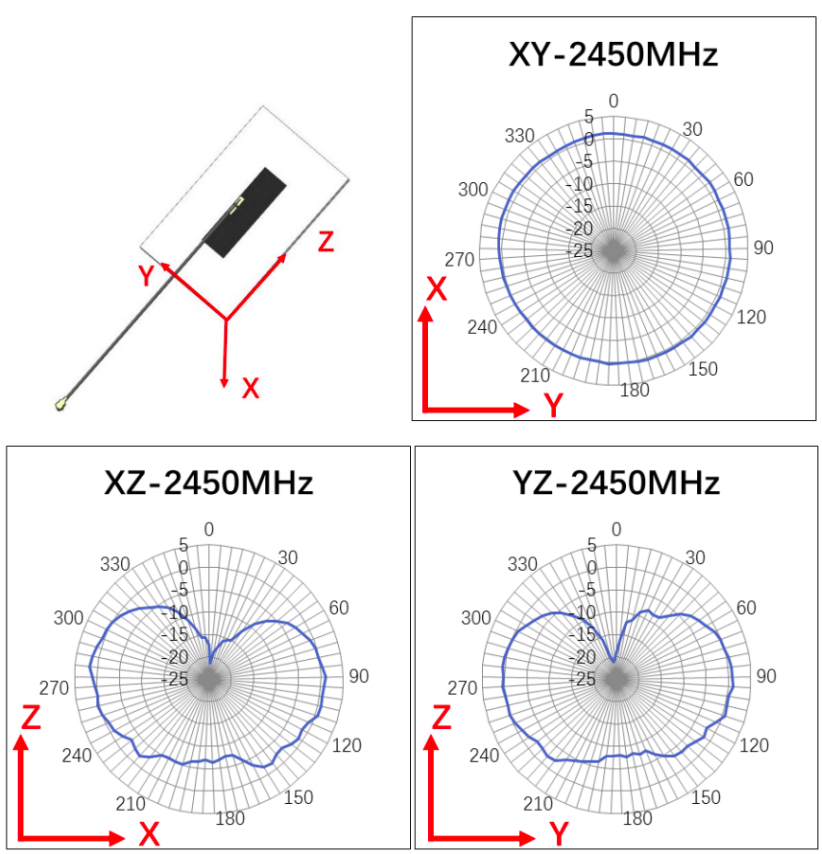
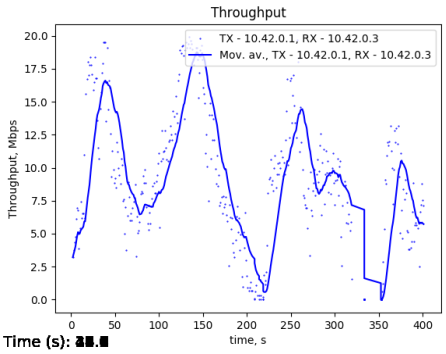
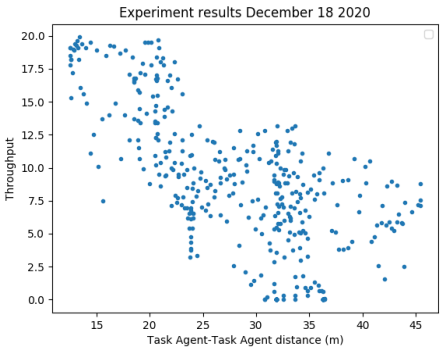


Figura 3.5: Patrón de radiación de las antenas Molex. Notar que en el caso de colocar la antena en posición vertical se obtiene la respuesta angular del gráfico XY: una respuesta esencialmente homogénea para todas las direcciones en un plano paralelo al piso. En caso de que el plano paralelo al piso sea el plano ZY o ZX referido a la antena, existen direcciones en que la potencia transmitida decae hasta $15dB$, e incluso un punto ciego. Extraído de hoja de datos de [36]



Serie temporal obtenida en experimento, con importantes fluctuaciones en *throughput*, no explicables a partir de la trayectoria definida.



Nube de puntos *throughput*-distancia. Sería esperable correlación decreciente si las antenas estuvieran colocadas correctamente.

Figura 3.6: Datos experimentales de *throughput* para un experimento con las antenas Molex en los brazos de los agentes. En el experimento los agentes se alejan sistemáticamente, lo que debería redundar en una caída relativamente monótona de la transmisión. Sin embargo, debido a la inconsistencia del patrón de radiación angular de las antenas, se observan fluctuaciones importantes en todo momento.

Figura 3.7: UAV con las antenas de Wi-Fi colocadas en los brazos. Esta configuración genera un patrón angular de radiación no homogéneo y puede traer problemas en la comunicación.

Figura 3.8: UAV con las antenas de Wi-Fi colocadas en las patas, actualmente en uso. Esta configuración aprovecha el patrón de radiación de las antenas y las permite operar en su geometría óptima.

3.5. Experimento móvil vs fijo

Una vez reubicadas las antenas en las patas de los UAVs, se procedió a llevar adelante el escenario propuesto en su versión experimental con el sistema de UAVs descrito en el Sección 2.1. En la Figura 3.9 se puede ver un diagrama de la trayectoria realizada por el agente de tarea, en la ubicación elegida para el experimento: un terreno en las afueras de Montevideo.

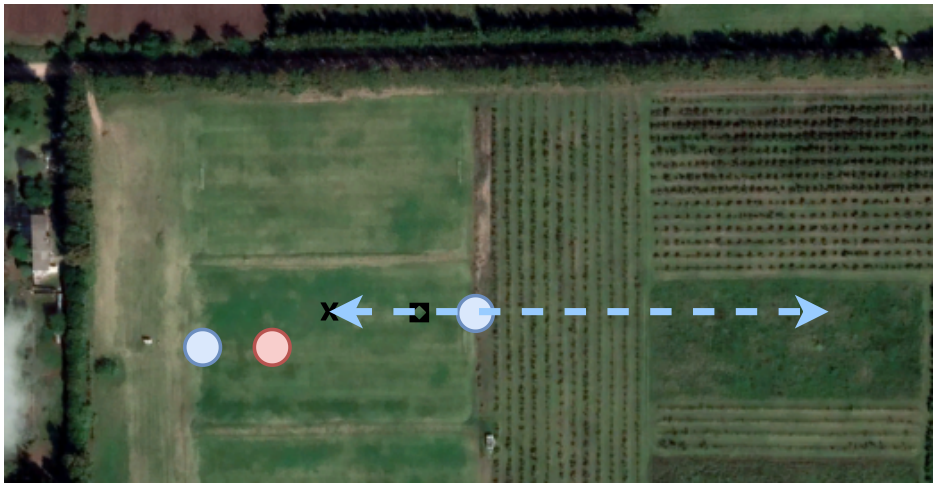


Figura 3.9: Diagrama describiendo el experimento sobre imagen satelital del terreno. Un agente de tarea (azul, derecha) se mueve en la línea negra en un ida y vuelta, despegando en la cruz y aterrizando en el cuadrado. El otro agente de tarea (azul, izquierda) permanece quieto en el punto rojo, mientras que el agente de red (rojo), se reposiciona dinámicamente o se queda estático según la opción especificada.

Los UAVs se encuentran comunicados con una red *ad-hoc* y utilizando el paquete de ROS *multi-master*, como se describió en Sección 2.1.2. Cada UAV corre el algoritmo de optimización (definido por las ecuaciones (2.9), y (2.12)) utilizando la biblioteca *cvxpy*. Para dicha optimización, se utiliza el mismo modelo de canal que en la Sección 3.3 con los parámetros especificados en la Tabla 3.1. La tasa de transmisión de mensajes en la red se mide utilizando el paquete *iperf3*, una herramienta de análisis de redes de datos que utiliza mensajes UDP para medir ancho de banda o *throughput* [47].

Mi participación en estos experimentos fue tanto a través de la configuración y mantenimiento de los UAVs y el soporte previo y durante la realización, como en el diseño experimental y el procesamiento y análisis de sus resultados. Los resultados de *throughput* para cada experimento se observan en las Figura 3.10 y Figura 3.11 respectivamente. Se observa un comportamiento similar al de la Sección 3.3: el throughput llega a bajar hasta 0 MBps durante el tramo final de la trayectoria para los casos de agente fijo, mientras que en los casos de agente dinámico, ese problema se evita. Sin embargo, a diferencia de la simulación, no se observa una mejora consistente que se puede deber en parte a la aleatoriedad del experimento y en parte a la sobrecarga que implica sobre la red el propio ruteo de los mensajes

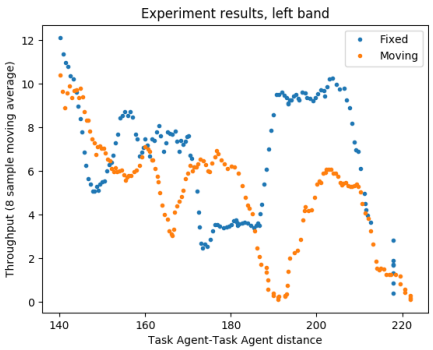


Figura 3.10: Resultados experimentales a lo largo de la banda izquierda de la cancha. En azul el desempeño en el caso de agente de red fijo, y en naranja para el agente de red móvil.

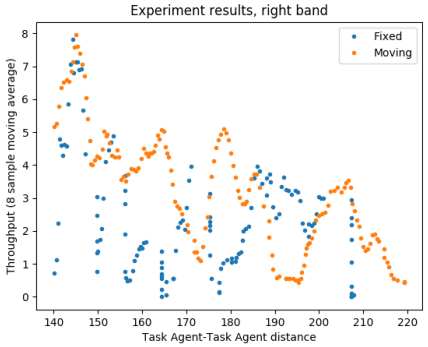


Figura 3.11: Resultados experimentales a lo largo de la banda derecha de la cancha. En azul el desempeño en el caso de agente de red fijo, y en naranja para el agente de red móvil.

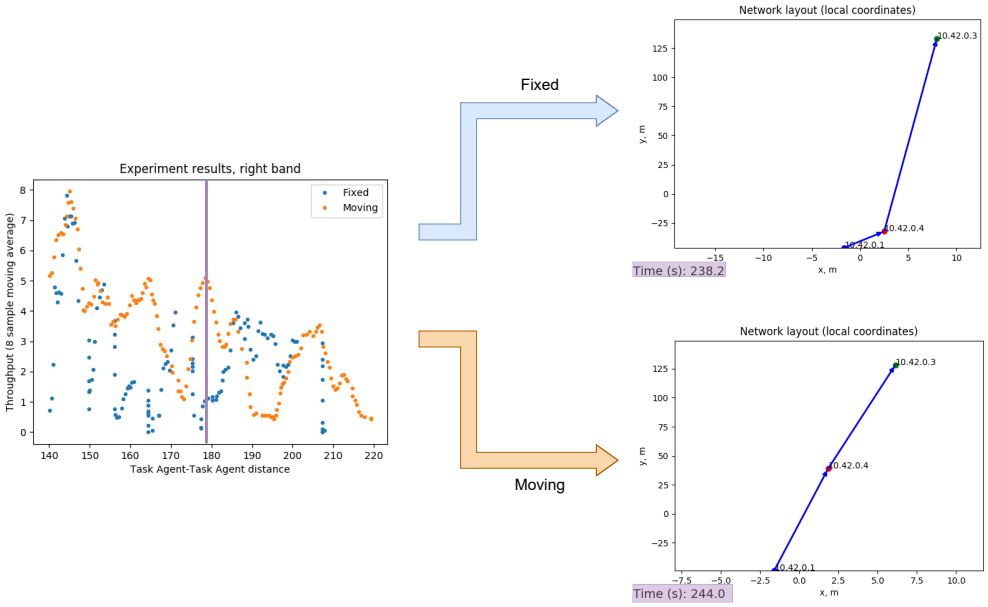


Figura 3.12: Snapshot del experimento. A la derecha las posiciones bajo los modos de operación *fijo* y *móvil* de todos los agentes en los experimentos. Notar la posición del agente de red (IP 10.42.0.4, color rojo), que en el posicionamiento dinámico es óptima.

a través del router.

Como complemento para la visualización, en la Figura 3.12 se muestra la posición de todos los agentes (a partir de los registros de vuelo) para un instante dado de la trayectoria. A su vez, en la Figura 3.13 se observa una fotografía de los UAV en vuelo.

Figura 3.13: Fotografía aérea del sistema en una configuración del modo de operación fijo, al segundo 20 de experimento. Con círculos sólidos se indican las posiciones de los agentes, con círculos sombreados su sombra, y con cruces sólidas sus plataformas de despegue. A su vez, la trayectoria realizada por el agente de tarea móvil se indica con una flecha punteada.

3.6. Experimento en círculo con 6 agentes

En esta sección se describe un experimento con 6 agentes realizado por el equipo para probar la potencialidad del sistema ensamblado y la coordinación entre múltiples agentes, corriendo nuestros algoritmos en un entorno *multi-master* de ROS.

Dos agentes de tarea recorren un círculo alrededor del centro de la cancha, mientras intentan comunicarse con un agente de tarea que permanece estático y en el suelo. Los restantes 3 agentes de red se posicionan dinámicamente según el algoritmo de ruteo dinámico ya explicado.

En la Figura 3.14 se muestra una fotografía aérea del conjunto de UAVs en vuelo y en la Figura 3.15 se muestra un fotograma del reporte de comunicación y posicionamiento del experimento a partir de los registros de vuelo de los agentes.

Figura 3.14: Fotografía aérea del experimento de 6 agentes. La posición de los UAV se indica con círculos azules.

Mi participación en este experimento fue a través de la configuración, mantenimiento y depuración del *hardware* y *software* de los UAVs durante la preparación y ejecución del experimento. A diferencia del detallado en la Sección 3.5, que diseñé y lideré, este experimento fue liderado por el Dr. Mikhail Gerasimenko.

Los resultados de *throughput* de este experimento no muestran una diferencia clara entre los dos modos de operación posibles para los NA, debido en parte al posicionamiento de las antenas explicado en la Sección 3.4, y en parte a la opción de tener agentes colocados a nivel de piso. Sin embargo, este experimento fue la motivación para utilizar el entorno *multi-master* detallado en la Sección 2.1.2, ya que en caso contrario el desempeño de la red se ve fuertemente disminuido. A su vez, este experimento fue esencial para el correcto diseño del de la Sección 3.5, particularmente haciendo notar los problemas de telecomunicaciones asociados a UAVs a nivel de piso, y a un incorrecto posicionamiento de las antenas de Wi-Fi.

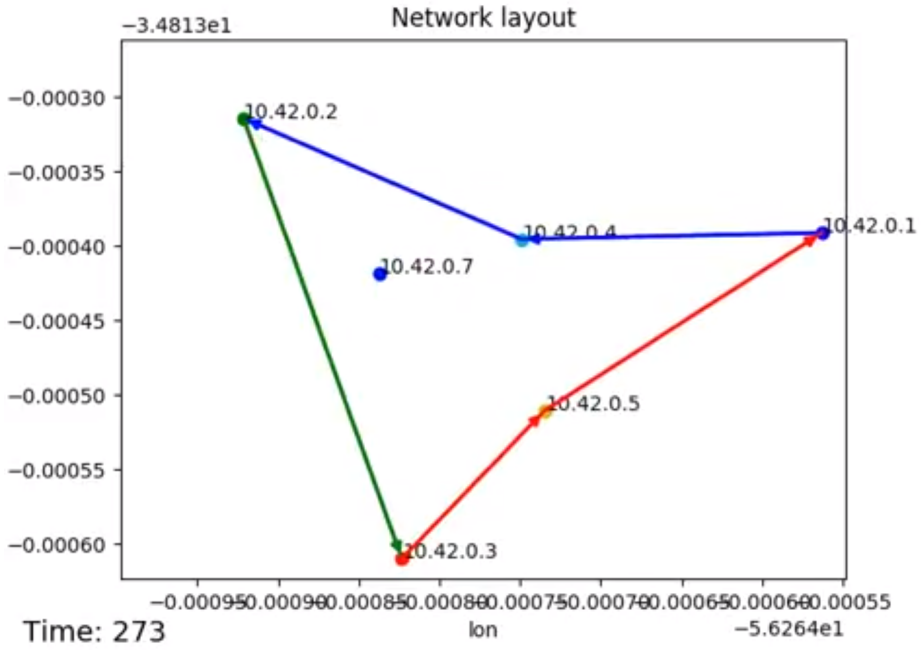


Figura 3.15: Fotograma de configuración del ruteo para un instante del experimento realizado. Los agentes de tarea utilizan a los nodos auxiliares de red para transmitir sus mensajes.

Capítulo 4

Monitoreo ACRL

4.1. Introducción

En este capítulo explicamos mi aporte realizando simulaciones de ACRL. Estas simulaciones describen un problema de monitoreo, en el que uno o varios agentes deben mantener ocupadas las regiones de interés al menos una porción de tiempo especificada. Este problema, cuya formulación más general se explica en la Sección 4.2, se lleva a la forma de ACRL basándose en los fundamentos matemáticos explicados en la Sección 2.3.

Las simulaciones realizadas en un entorno continuo y con un único agente desarrollamos en la Sección 4.3. En este caso, desarrollé el código en *python*, y parametricé la política del agente con una red neuronal utilizando la biblioteca *pytorch*. Se muestran visualizaciones tanto de la política del agente para estado aumentado fijo, como del desempeño de su política en un *loop* de ejecución. A su vez, se estudian los umbrales de desempeño del agente realizando un barrido de especificaciones de ocupación de las regiones a monitorear.

En la Sección 4.4 se simula un entorno discreto multi-agente, y se compara una política determinística y centralizada contra una política estocástica distribuida en la que cada agente actúa sin conocer el estado de su par.

Los resultados presentados en este capítulo son un punto de partida para desarrollar simulaciones multi-agente en un entorno continuo con una parametrización basada en una red neuronal.

4.2. Formulación general del problema distribuido

Consideramos un conjunto de N agentes que coexisten en un espacio $\mathcal{S}$. Este espacio $\mathcal{S}$ representa el espacio de posiciones, y podrá ser de la forma $\mathcal{S} \subset \mathbb{R}^2$ en el caso de espacio de posiciones continuo, o $\mathcal{S} \subset \mathbb{N}$ en un caso discreto. Como se definirá más adelante se trabajará con estados aumentados. Los estados aumentados contienen tanto la posición del agente como los multiplicadores de Lagrange asociados al cumplimiento de las restricciones del problema primal.

En cada instante de tiempo $t = 0, 1, \ldots$, el agente n (con $n = 1, \ldots, N$), toma una acción a_t^n según su política $\pi_n(a^n|s^n)$. Notar que la política del agente n depende únicamente de *su* estado s^n, y no del estado global $\mathbb{S} = \bigcup_{n=1}^{N} s^n$, de ahí que se esté en un contexto de política distribuida.

Los agentes colaboran para monitorear múltiples áreas de interés común $\mathcal{S}_i^{com} \subset \mathcal{S}$ con $i = 1, \ldots M_{com}$ obteniendo una recompensa unitaria si algún agente se encuentra en $\mathcal{S}_i^{com}$. A su vez, cada agente n puede deber atender hasta M_n regiones de monitoreo individuales $\mathcal{S}_j^n \subset \mathcal{S}$ con $j = 1, \ldots M_n$ obteniendo una recompensa unitaria si el agente n se encuentra en $\mathcal{S}_j^n$.

Con estas especificaciones, la recompensa asociada en tiempo t a la región S_i^{com} es:

$$r_t^i = \max_{n=1,\ldots,N} \mathbb{1}[s_t^n \in \mathcal{S}_i^{com}]$$

donde $\mathbb{1}[z]$ es la función indicatriz, tal que $\mathbb{1}[z] = 1$ cuando z se cumple y 0 en caso contrario. La notación del máximo implica que si alguna de las indicatrices resulta válida, el máximo de todas las indicatrices resultará 1.

Por otro lado, la recompensa asociada en tiempo t a la región $\mathcal{S}_j^n$ es:

$$r_t^{j,n} = \mathbb{1}[s_t^n \in \mathcal{S}_j^n]$$

Si $c_i,\ i = 1 \ldots M_{com}$ es la fracción de tiempo que cada región común $\mathcal{S}_i^{com}$ debe ser monitoreada, y $c_{j,n}$ es la fracción de tiempo que la región $\mathcal{S}_j^n$ debe ser monitoreada por el agente n, la formulación matemática del problema de monitoreo resulta el siguiente problema de optimización con restricciones:

$$\begin{aligned} P^\star = \max_{\pi_1,\ldots,\pi_N} \quad & 0 \qquad (4.1) \\ \text{sujeto a:} \quad & \lim_{T\to\infty} \frac{1}{T} \mathbb{E}_{S_t,A_t\sim\pi} \left[\sum_{t=0}^{T} r_t^i(S_t, A_t) \right] \geq c_i \ i = 1, \ldots, M_{com} \\ & \lim_{T\to\infty} \frac{1}{T} \mathbb{E}_{S_t,A_t\sim\pi} \left[\sum_{t=0}^{T} r_t^{j,1}(S_t, A_t) \right] \geq c_{j,1} \ j = 1, \ldots, M_1 \\ & \vdots \\ & \lim_{T\to\infty} \frac{1}{T} \mathbb{E}_{S_t,A_t\sim\pi} \left[\sum_{t=0}^{T} r_t^{j,N}(S_t, A_t) \right] \geq c_{j,N} \ j = 1, \ldots, M_N \end{aligned}$$

Si bien esta versión general es de interés y se encuentra en desarrollo un artículo en el que se estudia teóricamente este problema, en este capítulo presentaremos

simulaciones de dos casos particulares del problema. En la Sección 4.3 estudiamos un único agente en un espacio de posiciones continuo incluido en $\mathbb{R}^2$, mientras que en la Sección 4.4 se cuenta con dos agentes coexistiendo en un espacio discreto de tres posiciones.

4.3. Caso de un solo agente

Las simulaciones que se muestran en la presente sección son para el caso de un único agente en un entorno continuo de $\mathbb{R}^2$. El entorno consiste en $m = 3$ regiones a monitorear ubicadas en el rectángulo $\mathbb{S} = [-12.5, 12.5] \times [-12.5, 12.5]$ como se ilustra en la Figura 4.1.

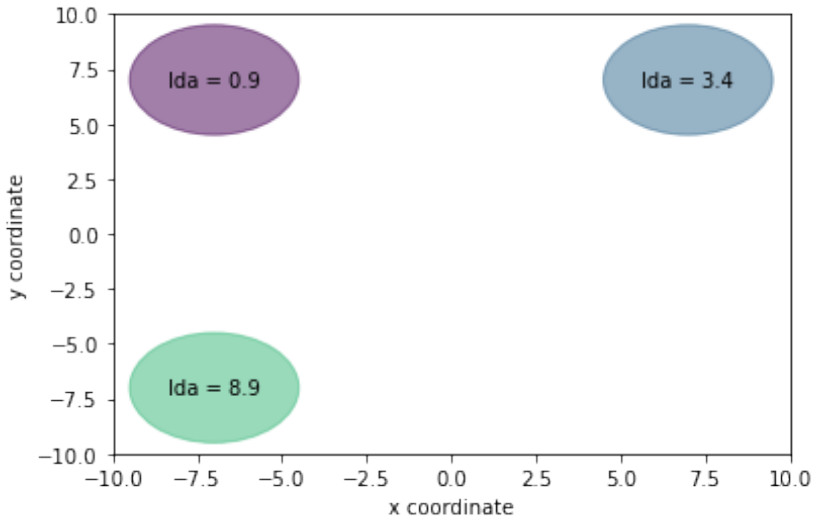


Figura 4.1: Entorno continuo utilizado. En este caso se cuenta con tres regiones a monitorear, representadas como círculos de colores de radio 2.5 y centros $[-7.0, 7.0], [7.0, 7.0], [-7.0, -7.0]$. Los multiplicadores asociados a las tres regiones son $(\lambda_0, \lambda_1, \lambda_2) = (0.9, 3.4, 8.9)$ y determinan la recompensa que recibiría el agente en este instante en caso de ocuparlas.

El estado (primal) del agente es un punto de $s_t \in \mathbb{R}^2$ que da su posición, y su acción $a_t \in \mathbb{R}^2$ permite que el estado evolucione según la ley:

$$s_{t+1} = s_t + \Delta \frac{a_t}{||a_t||} \tag{4.2}$$

Es decir, puede desplazarse en cada instante a lo sumo una distancia Δ (usaremos típicamente $0.2 \leq \Delta \leq 1.0$) en cualquier dirección. En este caso, no hace falta distinguir entre regiones individuales y colectivas, y el problema se reduce a:

$$\begin{aligned} P^\star = \max_{\pi} \quad & 0 \\ \text{sujeto a:} \quad & \lim_{T\to\infty} \frac{1}{T} \mathbb{E}_{S_t, A_t \sim \pi} \left[\sum_{t=0}^{T} r_t^i(S_t, A_t) \right] \geq c_i \;\; i = 1, 2, 3 \end{aligned} \tag{4.3}$$

En estas condiciones, y siguiendo el desarrollo de la Subsección 2.3.3 es posible formular el problema de ACRL dado por las recompensas aumentadas de la ecuación (2.16)

$$r_\lambda(s_t, a_t) = r_0(s_t, a_t) + \sum_{i=1}^{m} \lambda_i \left(r_i(s_t, a_t) - c_i\right)$$

y la actualización de multiplicadores de la ecuación de dinámica dual (2.18):

$$\lambda_{i,k+1} = \left[\lambda_{i,k} - \frac{\eta}{T_0} \sum_{t=kT_0}^{(k+1)T_0-1} \left(r_i(s_t, a_t) - c_i\right)\right] \tag{4.4}$$

donde los multiplicadores se actualizan cada T_0 pasos del problema en tiempo discreto. En esta sección utilizamos $T_0 = 30$. Este parámetro debe configurarse contemplando un compromiso entre seguir rápidamente los cambios de λ, y dejar que la política pueda actuar según lo aprendido para λ_k.

Dado que la función r_0 es nula, y que el factor con c_i solo acarrea un término constante que no tiene repercusiones para la actualización de gradientes, redefinimos:

$$r_\lambda(s_t, a_t) = \sum_{i=1}^{m} \lambda_i r_i(s_t, a_t) = \sum_{i=1}^{m} \lambda_i \mathbb{1}[s_t \in S_i] \tag{4.5}$$

Es decir, en caso de que el agente se encuentre en la región i, obtendrá recompensa λ_i, y en caso de no encontrarse en ninguna región obtendrá recompensa nula.

Notar que al trabajar con estados aumentados, la política depende también del multiplicador: $\pi(a_t|s_t, \lambda_k)$

El objetivo es lograr que el agente aprenda una política $\pi(a_t|s_t, \lambda_k)$, que logre mantener monitoreadas las regiones los tiempos requeridos c_i. La idea general es que el agente debe aprender a tomar acciones que lo lleven hacia la región con el multiplicador más elevado. Ya que -de acuerdo a (4.4)- al permanecer en una región, el multiplicador asociado a la misma decrece, y al ausentarse de las otras sus multiplicadores crecen, eventualmente el multiplicador asociado a otra región superará al de la actual y el agente deberá moverse para atenderla y así maximizar la recompensa.

Se decide separar el problema en un *loop* de entrenamiento del agente, en el que para un mismo episodio de entrenamiento se mantendrán congelados los multiplicadores y un *loop* de ejecución, en el que los multiplicadores se actualizarán dinámicamente en ejecución pero la política del agente permanecerá incambiada.

Otro detalle a notar es que según (4.4) los multiplicadores no tienen cota superior, sin embargo, para entrenar la política $\pi(a_t|s_t, \lambda_k)$ será conveniente entrenarla en un rango acotado de $\lambda \in (0, \lambda_{\max})$, por lo que es preciso definir cómo actuar en caso de que algún multiplicador supere $\lambda_{\max}$. Para esto, optamos por una simple normalización a $\lambda_{\max}$:

$$\lambda_i = \lambda_{\max} \frac{\lambda_i}{\max_i \lambda_i} \forall i \tag{4.6}$$

Esta normalización es válida ya que al cumplirse $r_0 = 0$, maximizar el Lagrangiano, o el Lagrangiano escalado por un factor resulta en el mismo problema.

4.3.1. Loop de entrenamiento

Llamamos *loop* de entrenamiento al algoritmo diseñado a efectos de obtener la política (paramétrica y diferenciable según los parámetros θ), que maximice la recompensa esperada para el problema aumentado (4.5). Como adelantamos anteriormente, para esta etapa no es necesario respetar la dinámica dual que impone (4.4), ya que el objetivo es lograr una política que funcione para cualquier valor λ y luego evaluar su desempeño dinámico en ejecución.

Adoptamos como base el esquema de *policy gradient* definido por la Ecuación C.12 de la Sección C.4 y que induce el algoritmo 6. Esto es, se llevan a cabo episodios de largo T, se calcula el retorno a cada instante, y se realiza el paso de gradiente dado por

$$\theta_{k+1} = \theta_k + \alpha G_k \nabla \ln \pi_\theta(a_k \mid s_k, \lambda_k) \tag{4.7}$$

Política Gaussiana

A su vez, como es usual para problemas en entorno continuo, se optará por una política Gaussiana. Es decir, nuestra parametrización tomará como entrada el estado (aumentado) actual, y devolverá como resultado un vector de $\mathbb{R}^2$, que se utilizará como media de una normal de varianza constante σ.

$$a \sim \mathcal{N}(\mu_\theta(s_t, \lambda_t), \sigma) = \mu_\theta(s_t, \lambda_t) + \nu, \text{ con } \nu \sim \mathcal{N}(0, \sigma) \tag{4.8}$$

La varianza σ es esencial para que el agente explore correctamente, como se desprende de calcular explícitamente el término

$$\nabla \ln \pi_\theta(a \mid s, \lambda) = \nabla \mu_\theta(s, \lambda) \frac{(a - \mu_\theta(s, \lambda))}{2\sigma^2} = \nabla \mu_\theta(s, \lambda) \frac{\nu}{2\sigma^2} \tag{4.9}$$

Es decir, mientras mayor sea el ruido de exploración mayor será el gradiente a propagar. Se debe fijar σ contemplando un compromiso entre exploración-explotación que permita explorar $\mathcal{S}$, pero no termine en una política tan ruidosa que dificulte mantenerse en las regiones objetivo.

Batchs con idéntica inicialización

Para agilizar el proceso de entrenamiento, se optó por la adaptación del algoritmo que se observa en la Figura 4.2, donde se observan $num\_batch = 4$ episodios consecutivos de entrenamiento, pertenecientes al mismo *batch*.

En vez de cada episodio volver a sortear el estado aumentado (s, λ), se reutiliza por un número de episodios el estado aumentado sorteado inicialmente.

Esto reduce la aleatoriedad y mejora el entrenamiento, debido a que la ecuación (4.7) es una estimación de la esperanza de G a partir de una única muestra. Tomar

batch permite estimar G con mayor cantidad de muestras y reducir la varianza del estimador de gradiente, lo que permite dirigirse más directamente al óptimo.

Otra forma de ver la mejora en usar *batchs* es por inspección de (4.7) sustituyendo (4.9). Cada término $\nabla\mu_\theta$ resulta ponderado por un factor de la forma $G_k(a-\mu_\theta)$. Tomar varias muestras resulta en una mejor ponderación de la dirección que otorga mayor retorno.

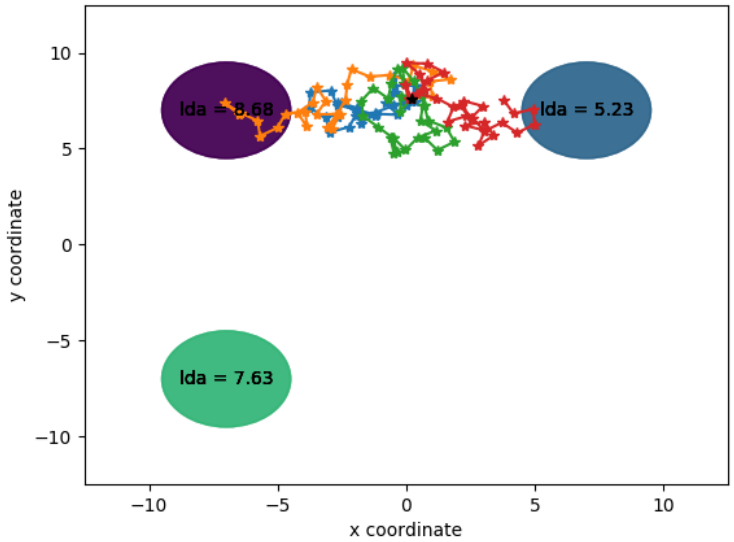


Figura 4.2: Cuatro episodios consecutivos de entrenamiento en colores rojo,verde,azul y naranja. El punto inicial para los episodios se dibuja en color negro. En cada episodio, el agente actúa según la política $\pi(a|s,\lambda)$ durante $T = 30$ pasos, y obtiene una recompensa igual a λ_i por cada instante que permanezca en la región i. Las regiones, dibujadas en violeta, azul y verde, verifican en esta oportunidad $\lambda_0 = 8.68, \lambda_1 = 5.23, \lambda_2 = 7.63$. Una vez concluido cada episodio, la política es actualizada según el paso de gradiente, y una vez concluido el bloque de 4 episodios, se vuelven a sortear tanto la posición inicial como los valores de los multiplicadores asociados a las regiones.

Distribución probabilística para muestreo de λ

Si bien la solución más natural para el sorteo del estado es tomar variables aleatorias uniformes en el rango a evaluar, para el caso de λ esto trae problemas de implementación asociados al desempeño en ejecución.

Esto se debe a que en ejecución, las distribuciones no se ven de manera uniforme, sino que es usual tener un multiplicador notoriamente por encima de los otros (lo que según (4.6) implica un multiplicador en $\lambda_{\max}$ y los otros cerca de 0).

Por esto, se optó por el siguiente método para sortear aleatoriamente los λ, que cuenta con un hiperparámetro p_{uni}:

- Con probabilidad p_{uni}, $\lambda_i \sim \mathcal{U}(0, \lambda_{\max})\ \forall i$.

- Con probabilidad $1 - p_{uni}$, se elige aleatoriamente un índice $h = \mathcal{U}(0, m)$. Al multiplicador de índice h se le asigna $\lambda_h \sim \mathcal{U}(\frac{3}{4}\lambda_{\max}, \lambda_{\max})$, mientras que al resto de los multiplicadores se les asigna $\lambda_i \sim \mathcal{U}(0, \lambda_{\max}/4)\ \forall i \neq h$.

Esto fuerza al entrenamiento a ponderar más fuertemente distribuciones de λ en donde hay un multiplicador marcadamente por encima del resto. Para los experimentos mostrados utilizamos $p_{uni} = 0.5$.

Resumen de algoritmo de entrenamiento

El *loop* de entrenamiento se resume en el algoritmo 3. Cada $batch\_size = 4$ episodios se genera una nueva posición inicial aleatoria y un nuevo conjunto de multiplicadores aleatorios con la distribución de probabilidad para λ dada por el p_{uni} seleccionado. En cada episodio, se actúa según la política actual durante $T = 30$ pasos, y luego se actualiza la política en función de las recompensas obtenidas y la ecuación de θ_{k+1}, que es la ecuación de REINFORCE [20].

Entrada: Política paramétrica diferenciable $\pi_\theta(a \mid s, \lambda)$,
hiperparámetros α, T, num_episodes, batch_size
Salida : Parámetros entrenados θ^*

Inicializar, $\theta \in \mathbb{R}^d$, por ejemplo $\theta = 0$;
for *num_episodes/batch_size* **do**
  Generar una posición aleatoria $s_0 \sim \mathcal{U}(\mathcal{S})$;
  Generar un vector de multiplicadores aleatorio $\lambda_0 \sim \text{dist}\lambda$;
  for *batch_size* **do**
    Inicializar $s = s_0$, $\lambda = \lambda_0$;
    Generar un episodio $S_0, A_0, R_1, \ldots S_{T-1}, A_{T-1}, R_T \sim \pi_\theta$;
    Calcular $G_t = \sum_{i=t+1}^{T} R_i$;
    Actualizar $\theta_{k+1} = \theta_k + \alpha \sum_{t=0}^{T} G_t \nabla_\theta \ln \pi_\theta(a_t \mid s_t, \lambda_k)$;

Algoritmo 3: Algoritmo de entrenamiento para el caso continuo de un solo agente, constando de *num_episodes* episodios de entrenamiento. En cada iteración del primer bucle, se genera una nueva posición inicial aleatoria y un nuevo conjunto de multiplicadores aleatorios con la distribución de probabilidad distλ que surge de la elección de $p_{uni} = 0.5$. Estos valores se utilizan como inicialización una cantidad de veces definida por el hiperparámetro batch_size = 4. En cada episodio, se actúa según la política durante $T = 30$ pasos, y luego se actualiza la política en función de las recompensas obtenidas y la ecuación de θ_{k+1}, que es la ecuación de REINFORCE [20].

4.3.2. Parametrización con Red Neuronal

Como parametrización de $\mu_\theta(s, \lambda)$ se optó por una red neuronal con una etapa de expansión dimensional de s y λ y dos capas *fully connected*, que combinan las *features* que resultan de la capa de expansión. Este proceso se ilustra en el esquema de la Figura 4.3.

El estado aumentado (s, λ) se descompone en sus componentes λ y su componente posición $s = (x, y)$. El vector de λ es pasado por una capa *fully-connected* para generar un vector de *features* de dimensión n_λ. En paralelo, el vector de posición atraviesa una capa FC para generar n_s *features*.

Separar el vector en esta etapa de pre-procesamiento fue clave para que el sistema funcionara con más de dos regiones. La razón de separar s y λ es permitir que la red aprenda a multiplexar entre varias políticas de navegación según los valores de λ, es decir, el valor de λ -visto a través de las n_λ *features* a aprender- debe definir a cuál de las regiones apuntar. A su vez, procesar separadamente s y λ es una decisión de diseño que recoge la naturaleza distinta de estos vectores por el propio problema formulado.

Estas *features* se concatenan en un vector de $n_\lambda + n_s$ *features* y son alimentadas a una red neuronal de dos capas completamente conectadas. Una red neuronal de dos capas completamente conectadas es interpretable como un clasificador lineal a trozos del vector de entrada. De esta manera, la red divide el espacio de *features* de posición y multiplicadores en hiperplanos y decide qué desplazamiento tomar según en qué región de ese espacio se encuentra el estado.

La función de activación a la salida de cada capa es una tanh(), que presenta la propiedad de tener su rango de salida acotado al intervalo $[-1, 1]$. Se eligió esta función de activación para este entorno con salida continua ya que la alternativa de ReLU tendía a generar salidas de gran magnitud que perjudicaban el desempeño. A su vez, a la salida de la capa de dimensión $n_h \times n_\lambda + n_s$ se introduce un *dropout* de $p = 0.6$, que ayuda al entrenamiento de la red forzando a que la red neuronal aprenda a funcionar prescindiendo de un conjunto aleatorio de neuronas [48].

Los valores necesarios para replicar la red utilizada son los de la tabla 4.1:

n_λ	128
n_s	64
n_h	512
learning rate	1.0e−5
número de episodios	$80k$

Tabla 4.1: Valores utilizados para la parametrización propuesta y su entrenamiento.

Por construcción (gracias a la activación tanh()), a la salida de la red neuronal se tiene un vector de dos elementos $\mu = (\mu_x, \mu_y) \in ([-1, 1], [-1, 1])$. Para determinar la acción, a este valor se le suma un ruido gaussiano de varianza $\sigma = 1.0$ (como define la ecuación (4.8)) y luego se actualiza el estado según la ecuación (4.2) con $\Delta = 0.8$.

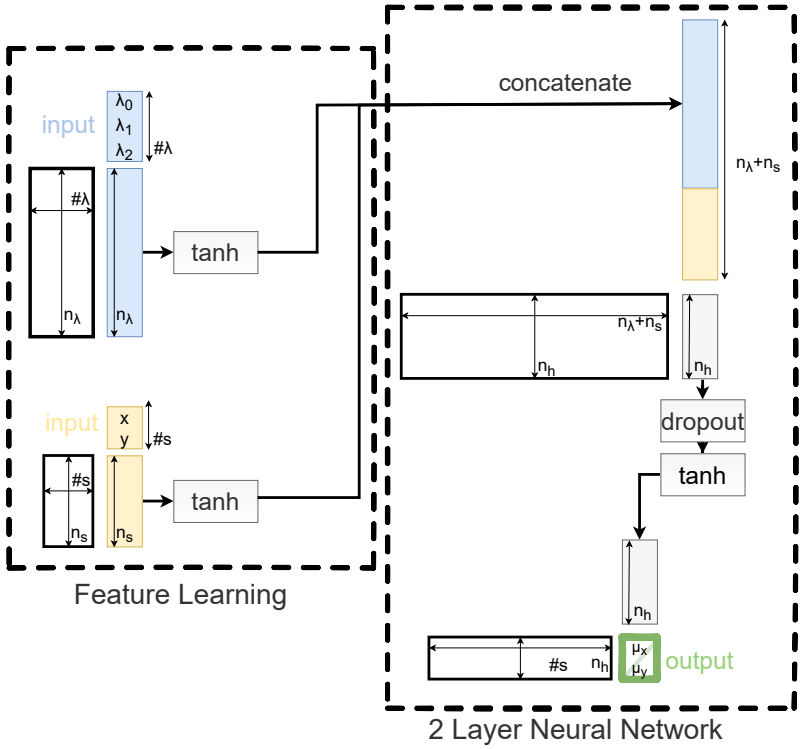


Figura 4.3: Representación de la política paramétrica. El estado se descompone en sus componentes λ y su componente posición (x, y) El vector de λ es pasado por una capa *fully-connected* (FC) para generar un vector de *features* de dimensión n_λ. En paralelo, el vector de posición atraviesa una capa FC para generar n_s *features*. Estas *features* se concatenan en un vector de $n_\lambda + n_s$ *features* y son alimentadas a una red neuronal de dos capas completamente conectadas con *dropout* $p = 0.6$. La función de activación a la salida de cada capa es una $\tanh()$, lo que es necesario para que los valores no se disparen.

Entrenamiento con backprop en pytorch

Es importante notar que la red neuronal utilizada en la simulación fue definida en la biblioteca *pytorch*, especializada en entrenamiento de redes neuronales.

Debido a su algoritmo interno de *back-propagation* (cálculo automatizado de gradientes), no es necesario programar explícitamente la ecuación (4.9), ya que la biblioteca se encarga de calcular gradientes automáticamente a partir de la arquitectura especificada.

Sin embargo, sí resulta necesario reescribir la ecuación (4.7), para llevarla a la forma estándar de descenso de gradiente sobre una función de pérdida que se utiliza en *pytorch*. Es decir, el algoritmo de *pytorch* espera recibir una función de pérdida escalar $L_\theta(\text{episodio})$ para realizar descenso por gradiente sobre los parámetros θ de la forma:

$$\theta_{k+1} = \theta_k - \alpha \nabla_\theta L_\theta(\text{episodio}) \tag{4.10}$$

Según el algoritmo de REINFORCE (Sección C.4), en cada episodio tenemos:

For $t = 0, 1, \ldots T-1$:

$$G_t = \sum_{i=t+1}^{T} R_i$$
$$\theta_{t+1} = \theta_t + \alpha G \nabla \ln \pi_\theta(a_t \mid s_t, \lambda_k)$$

por lo que al cabo de un episodio,

$$\theta_{k+1} = \theta_k + \alpha \sum_{t=0}^{T} G_t \nabla_\theta \ln \pi_\theta(a_t \mid s_t, \lambda_k) \tag{4.11}$$

De comparación directa entre las ecuaciones (4.10) y (4.11) surge que la función de pérdida se debe definir como:

$$L_\theta(\text{episode}) = -\sum_{t=0}^{T} G_t \ln \pi_\theta(a_t \mid s_t, \lambda_k) \tag{4.12}$$

por lo que para calcular la función de pérdida que precisa la biblioteca *pytorch* se debe llevar cuenta de las recompensas de un episodio, de forma de calcular el retorno G, y de $\ln \pi_\theta(a_t \mid s_t, \lambda_k)$ de cada paso, que para el caso de políticas gaussianas basta con un llamado a la función *distribution.log_prob(action)*.

4.3.3. Visualización de política entrenada

Siguiendo el algoritmo 3, y la parametrización descrita en la Subsección 4.3.2 y con la implementación de *back-propagation* que surge de (4.12), se entrenó un agente durante 80.000 episodios obteniendo la curva de aprendizaje (recompensa en función de número de episodio) de la Figura 4.4. Los valores utilizados fueron los de la Tabla 4.1, un *dropout* de $p = 0.6$, y un sorteo para λ con $p_{uni} = 0.5$.

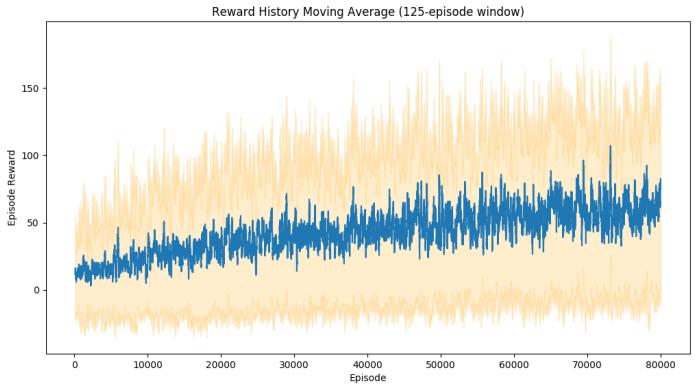


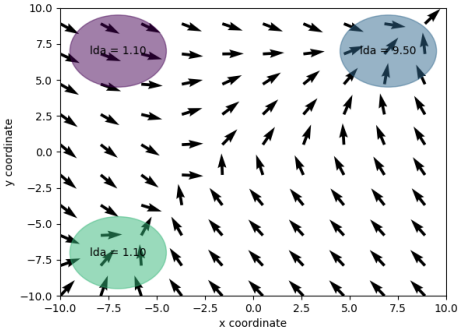
Figura 4.4: Desempeño según episodio del *loop* de entrenamiento, mostrando una media móvil de 125 episodios. Se observa un comportamiento creciente de la recompensa esperada. Notar que la recompensa de un episodio aleatorio depende fuertemente de los valores sorteados para s, λ.

Lo obtenido en la Figura 4.4 indica que la parametrización está logrando efectivamente aprender a obtener mayores recompensas, pero no proporciona información sobre el tipo de políticas que se están tomando ni sobre la posibilidad de resolver el problema con restricciones.

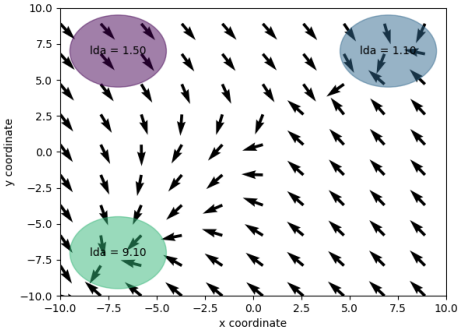
Para tener una idea de cómo el agente se comporta para un valor de λ fijado (por ejemplo, entre actualizaciones sucesivas de λ), realizamos el gráfico vectorial que se observa en la Figura 4.5. En este gráfico se representa en cada punto de una grilla de muestreo cuál es la acción media que el agente toma de acuerdo a $\pi_\theta(s, \lambda)$.

Ya que los gráficos vectoriales de la política no dan información sobre el desempeño en ejecución del agente, es necesario definir los parámetros de ejecución (es decir, η y c_i en (4.4)) para mostrar un *loop* de ejecución.

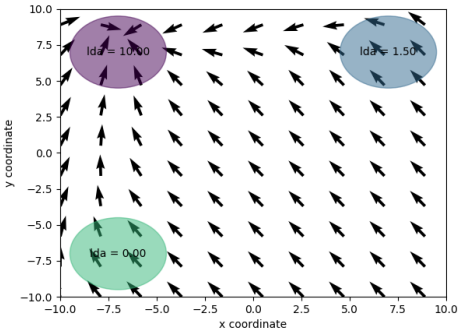
En la figura Figura 4.6 se observa una trayectoria, con el color representando la evolución temporal, así como la evolución de los λs normalizados, y el porcentaje de cubrimiento de cada región. En esta ejecución se observa que ante unos requerimentos de cubrimiento de regiones de $c = (0.4, 0.1, 0.05)$, el agente logra decidir alternar entre las distintas regiones y llegar a los cubrimientos especificados correctamente.



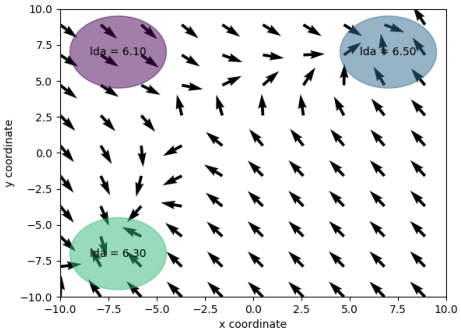
Política obtenida para $\lambda = (1.1, 9.5, 1.1)$.



Política obtenida para $\lambda = (1.5, 1.1, 9.1)$.



Política obtenida para $\lambda = (10.0, 1.5, 0.0)$.



Política obtenida para $\lambda = (6.1, 6.5, 6.3)$.

Figura 4.5: Gráfico vectorial de la política para varios valores específicos de λ. En cada punto de la grilla, se representa la acción media que el agente tomaría de acuerdo a $\pi_\theta(a|s, \lambda)$. Se observa que el agente es capaz de orientarse hacia la región con mayor recompensa en los tres primeros casos, donde una recompensa es notoriamente superior al resto. En el último caso, las tres regiones otorgan recompensas similares, y el agente puede actuar en dirección a una región que no sea la más valiosa.

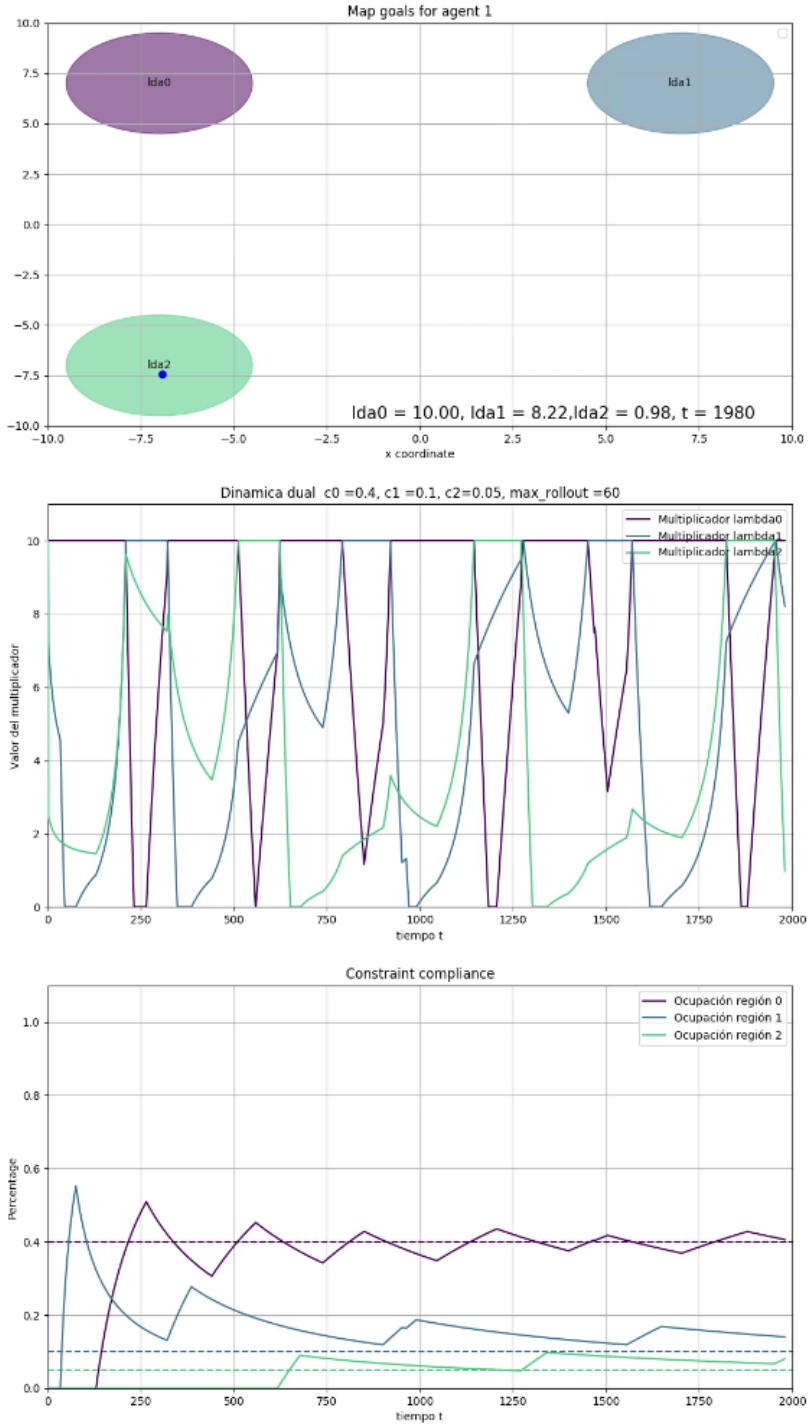


Figura 4.6: Evolución de multiplicadores y valores de cumplimiento medio en un *loop* de ejecución de 2000 pasos, con valor de $\eta = 1$ y $c = (0.4, 0.1, 0.05)$. Se grafica también el valor de los multiplicadores normalizado, así como el cumplimiento acumulado de cada restricción. El agente solo cambia su política cada $T = 60$ pasos, ignorando los valores de λ entre sucesivas actualizaciones

4.3.4. Umbral de factibilidad para política entrenada

Las figuras presentadas en la Subsección 4.3.3 permiten ver el desempeño del agente en un *loop* de ejecución para valores concretos de restricciones. Para evaluar el desempeño del agente en general, analizamos el umbral de factibilidad para el agente entrenado, es decir, cuáles son los valores límites de c_0, c_1, c_2 en los cuales el agente es capaz de alcanzar los cumplimientos especificados.

Para esto, optamos por realizar un mapa de calor como se muestra en la Figura 4.7. En cada punto de la grilla se grafica con color el *slack* obtenido de la restricción más comprometida, definido como slack $= o_i - c_i$ donde $o_i = \frac{1}{T_{exe}} \sum r_i$ es la ocupación promedio de la región i en los $T_{exe} = 8000$ pasos de ejecución. Se elige un tiempo de 8000 pasos al ser suficientemente largo como para que el agente pueda haber cubierto todas las regiones que son necesarias.

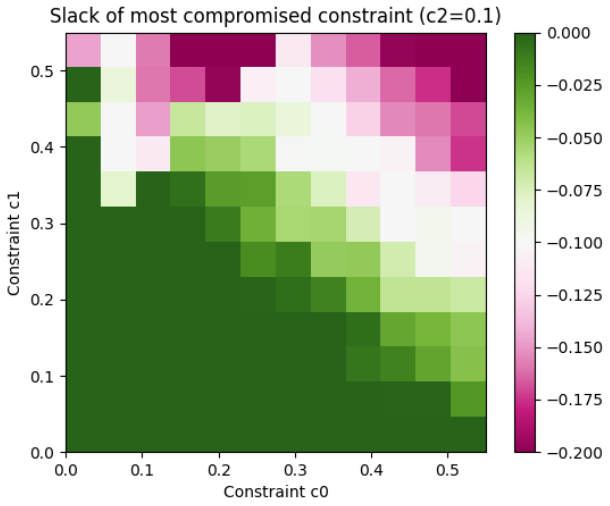


Figura 4.7: Mapa de calor del cumplimiento de restricciones para todos los valores posibles de c. En cada punto se grafica con color el *slack* obtenido de la restricción más comprometida.

De la figura se desprende que el agente es capaz de resolver correctamente los escenarios planteados para una suma de restricciones menor a ~ 0.8 (que resulta ser la recta que cubre la diagonal de la figura).

4.4. Caso de 2 agentes en entorno discreto

En esta sección se muestra un algoritmo distribuido en un MDP finito de solo tres estados posibles, con solo dos acciones posibles por estado, que es una versión simplificada del entorno de múltiples regiones de la Sección 4.3 para el caso de múltiples agentes.

En este caso el entorno queda definido por tres estados, discretos (G,C,B) (*Goal* u objetivo-Centro-Batería), y por transiciones probabilísticas como se muestran en la Figura 4.8. El agente es capaz de tomar las acciones discretas "izquierda" y "derecha". Las acciones hacia la derecha hacen transicionar al agente en a lo sumo dos pasos hacia la batería, mientras que las acciones hacia la izquierda tienen, en caso de encontrarse en el estado C, una probabilidad $1-\alpha$ de mover al agente hacia el *goal*, y una probabilidad α de mantenerlo en el estado C. Similarmente, si el agente se encuentra en el estado G y actúa hacia la izquierda, tiene una probabilidad $1-\alpha$ de mantenerse en esa posición y una probabilidad α de transicionar hacia C. Esta definición del MDP trata de emular, en la región G, tanto el hecho de que no necesariamente se llega en un único paso a la región objetivo, como el hecho de que incluso con la política de mantenerse en G, existe posibilidad de dejar el estado como sucede con la política Gaussiana (4.8).

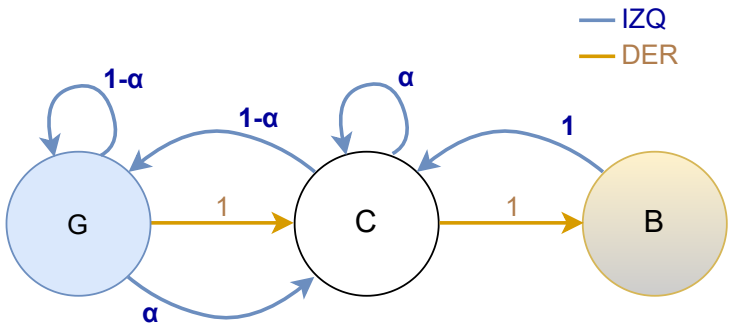


Figura 4.8: Entorno definido por tres estados, discretos GCB (*Goal* u objetivo-Centro-Batería), y por transiciones probabilísticas . El agente es capaz de tomar las acciones discretas "izquierda" y "derecha". Las acciones hacia la derecha hacen transicionar al agente en a lo sumo dos pasos hacia la batería, mientras que las acciones hacia la izquierda tienen, en caso de encontrarse en el estado Centro, una probabilidad $1-\alpha$ de mover al agente hacia el *goal*, y una probabilidad α de mantenerlo en el estado Centro. Similarmente, si el agente se encuentra en el estado *Goal* y actúa hacia la izquierda, tiene una probabilidad $1-\alpha$ de mantenerse en esa posición y una probabilidad α de transicionar hacia el Centro.

A su vez, en este caso multi-agente, es necesario especificar qué regiones son comunes a ambos agentes y qué regiones son solo de interés de un agente. G es una región común, por lo que $r_t^G = \text{máx}_{n=1,2}\, \mathbb{1}[s_t^n = G]$. C es un estado de transición y no otorga recompensa, y B es la región de monitoreo individual para cada agente, que puede tener en principio especificaciones de ocupación diferentes. La especificación del problema queda definida con tres variables de especificación

de ocupación diferentes (c_G, c_{B1}, c_{B2}), y el problema resulta:

$$
\begin{aligned}
P^\star = \max_{\pi} \quad & 0 \qquad (4.13)\\
\text{sujeto a:} \quad & \lim_{T\to\infty} \frac{1}{T}\mathbb{E}_{S_t,A_t\sim\pi}\left[\sum_{t=0}^{T} r_t^G(S_t,A_t) = \sum_{t=0}^{T} \max_{n=1,2} \mathbb{1}[s_t^n = G]\right] \geq c_G\\
& \lim_{T\to\infty} \frac{1}{T}\mathbb{E}_{S_t,A_t\sim\pi}\left[\sum_{t=0}^{T} r_t^{B1}(S_t,A_t) = \sum_{t=0}^{T} \mathbb{1}[s_t^1 = B]\right] \geq c_{B1}\\
& \lim_{T\to\infty} \frac{1}{T}\mathbb{E}_{S_t,A_t\sim\pi}\left[\sum_{t=0}^{T} r_t^{B2}(S_t,A_t) = \sum_{t=0}^{T} \mathbb{1}[s_t^2 = B]\right] \geq c_{B2}
\end{aligned}
$$

Es decir, la especificación es que G esté ocupado por algún agente una fracción c_G del tiempo, que B esté ocupado por el agente 1 una fracción c_{B1} del tiempo, y que B esté ocupado por el agente 2 una fracción c_{B2} del tiempo. En el problema dual, se tendrán los multiplicadores $\lambda = (\lambda_G, \lambda_{B1}, \lambda_{B2})$, que se actualizarán según el incumplimiento de la restricción como en (4.4).

4.4.1. Política centralizada

Se utilizó una política centralizada para tener de referencia, en la que el estado del sistema es $\{G, C, B\} \times \{G, C, B\}$ y el espacio de acciones es $\{L, R\} \times \{L, R\}$. Un posible estado del sistema puede ser GC (correspondiente al agente 1 en G y el agente 2 en C), y una subsecuente acción RR puede llevar al sistema al estado CB.

En este caso no separamos el *loop* de entrenamiento del de ejecución, debido a que la baja dimensionalidad del problema permite resolver la política óptima cada vez que se requiera para el valor de λ. Para esto utilizamos *Value Iteration*[20] para obtener la mejor política determinística. El proceso se resume en el algoritmo 4.

Entrada: Requisitos de ocupación (c_G, c_{B1}, c_{B2})
Salida : Episodio de ejecución.
Inicializar, $s = CC$, $\lambda = (\lambda_G, \lambda_{B1}, \lambda_{B2})$;
for *num_eps* **do**
  Resolver MDP por *Value Iteration* y obtener $\pi^*(a_1, a_2|s_1, s_2, \lambda)$;
  for $t = 1 \dots T$ **do**
    Generar un episodio $S_0, A_0, R_1, \dots S_{T-1}, A_{T-1}, R_T \sim \pi^*$;
  Actualizar λ según (4.4);

Algoritmo 4: *Loop* de ejecución bajo una política centralizada. Luego de cada actualización de las variables duales, el MDP de 9 estados y 4 acciones posibles, se resuelve por medio de *Value Iteration*.

4.4.2. Política distribuida

Llamaremos política distribuida a una política $\pi(a_1, a_2|s_1, s_2, \lambda)$ basada en dos políticas autónomas $\pi_1(a_1|s_1, \lambda)$, $\pi_2(a_2|s_2, \lambda)$. Estas son autónomas en el sentido de que cada agente desconoce el estado del otro. Cada agente puede tomar acciones para cambiar únicamente su estado, y conoce el vector completo de multiplicadores, cumpliéndose en todo momento

$$\pi(a_1, a_2|s_1, s_2) = \pi_1(a_1|s_1)\pi_2(a_2|s_2) \tag{4.14}$$

El esquema de operación propuesto en esta sección sigue una lógica de supervisión mutua de los agentes que se itera en sucesión. Uno de los agentes -s.p.g el 2- toma su primer acción "no supervisado" y recibe recompensas según el MDP aumentado dado por λ para un único agente (la recompensa de λ_G se obtendrá solo si el agente 2 se encuentra en G). Posteriormente, el otro agente toma su acción en un MDP redefinido y sujeto a las acciones del primer agente.

Esta redefinición del MDP se hace teniendo en cuenta las probabilidades de ocupación de cada región que surgen de la política del agente 2, y las recompensas para el problema multi-agente.

Si las probabilidades estacionarias del agente 2 son $p^\infty = (p_G^\infty, p_C^\infty, p_B^\infty)$, por definición de esperanza, se tiene para el agente 1:

$$r(s_1{=}X) = p_B^\infty r(s_1{=}X, s_2{=}B) + p_C^\infty r(s_1{=}X, s_2{=}C) + p_G^\infty r(s_1{=}X, s_2{=}G) \tag{4.15}$$

Lo que, combinado con (4.5) resulta en:

$$\begin{aligned} r(s_1{=}B) &= p_B^\infty(\lambda_{B1} + \lambda_{B2}) + p_C^\infty \lambda_{B1} + p_G^\infty(\lambda_G + \lambda_{B1}) \\ r(s_1{=}C) &= p_B^\infty \lambda_{B2} + p_C^\infty \cdot 0 + p_G^\infty \lambda_G \\ r(s_1{=}G) &= p_B^\infty(\lambda_G + \lambda_{B1}) + p_C^\infty \lambda_G + p_G^\infty \lambda_G \end{aligned} \tag{4.16}$$

Si bien una primer idea podría ser emular la política determinística utilizada en la Subsección 4.4.1, esto lleva a un problema tipo *deadlock* en el que rápidamente para ningún agente es conveniente cambiar su política sujeto a la actuación del otro agente, cayéndose en un óptimo local. Por ejemplo, en el caso de que $\lambda_G > \lambda_{B1} > \lambda_{B2}$, si el agente 1 adopta la política de ir a G, el agente 2 adoptará la política de mantenerse en B y no es posible escapar del óptimo local encontrado.

Por esto se opta por una política estocástica *Softmax* [**gao2017properties**] de 6 pesos -que determinan la probabilidad de elegir cada una de las 2 acciones posibles en cada uno de los 3 estados posibles- para cada agente. Esto permitirá que hayan caminos suaves entre el óptimo local y el óptimo global. El agente 2 realiza un episodio de T pasos y actualiza sus pesos θ_2 de acuerdo a REINFORCE (4.7), y luego el agente 1 realiza un episodio sujeto a la política del agente 2 (como en (4.16)) y actualiza sus pesos. Posteriormente revierten sus roles y mantienen el *loop* hasta converger la política bajo algún criterio de parada.

Nuevamente, el tamaño del problema permite "descartar" el entrenamiento y generar una política óptima para cada nuevo conjunto de λ que surjan. De todo lo anterior, el algoritmo distribuido resulta:

Entrada: Requisitos de ocupación (c_G, c_{B1}, c_{B2})
Salida : Episodio de ejecución.
Inicializar, s_1, s_2, $\lambda = (\lambda_G, \lambda_{B1}, \lambda_{B2})$, θ_1, θ_2;
for *num_eps* **do**
  while *no convergido* **do**
    Redefino recompensas para el agente 1 dado θ_2 según (4.16);
    Episodio de REINFORCE actualizando θ_1;
    Redefino recompensas para el agente 2 dado θ_1 según (4.16);
    Episodio de REINFORCE actualizando θ_2;
  for $t = 1 \dots T$ **do**
    Generar un episodio $S_0, A_0, R_1, \dots S_{T-1}, A_{T-1}, R_T \sim \pi_1, \pi_2$;
  Actualizar λ según (4.4);

Algoritmo 5: *Loop* de ejecución bajo una política distribuida. Luego de cada actualización de las variables duales, los agentes alternan episodios de REINFORCE y redefiniciones de su MDP sujeto a la política del otro.

4.4.3. Resultados y aportes para futuro trabajo

En la Figura 4.9 se muestra un episodio de ejecución llevado adelante por la política distribuida para los agentes. Es de notar que los valores elegidos para $c_0 = 0.95$ y $\alpha = 0.1$ exigen coordinación, ya que es necesario que ambos agentes permanezcan en G simultáneamente una parte del tiempo, ya que de elegir que un agente se encargue unilateralmente de G, a lo sumo podrá estar un 90 % del tiempo, no alcanzando la restricción de $c_0 = 0.95$.

En Figura 4.10 se muestra la comparación entre las regiones de factibilidad para la política centralizada y distribuida. Se observa que la política distribuida logra resolver correctamente el problema incluso con valores que exigen una fuerte coordinación entre los agentes.

Estos resultados dan intuición sobre las posibilidades de una política distribuida para aprender a coordinar acciones entre los agentes. La versión del algoritmo 5 que reemplaza la redifinición del MDP por su versión de una muestra, es simplemente tomar un episodio en el que ambos agentes accionan y asignar recompensas a cada uno según las recompensas aumentadas que surgen de (4.13). Es decir, cada agente solo ve al otro a través de las recompensas, ya que estas dependen de la acción de los dos agentes.

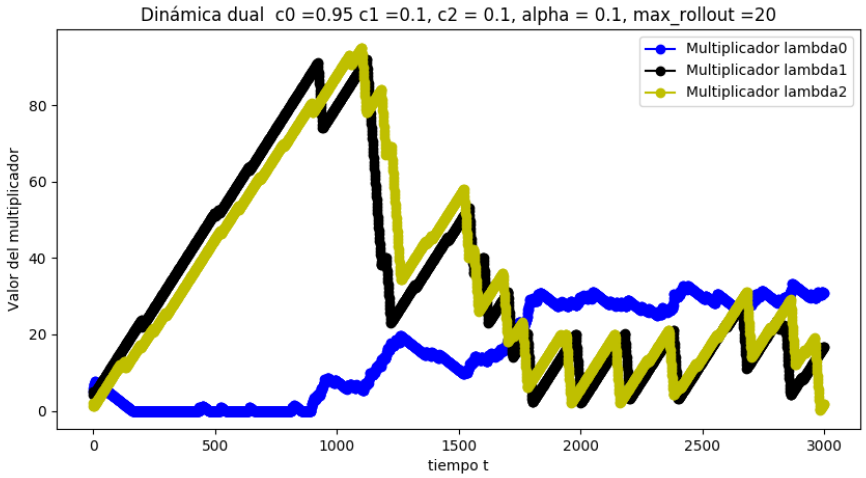
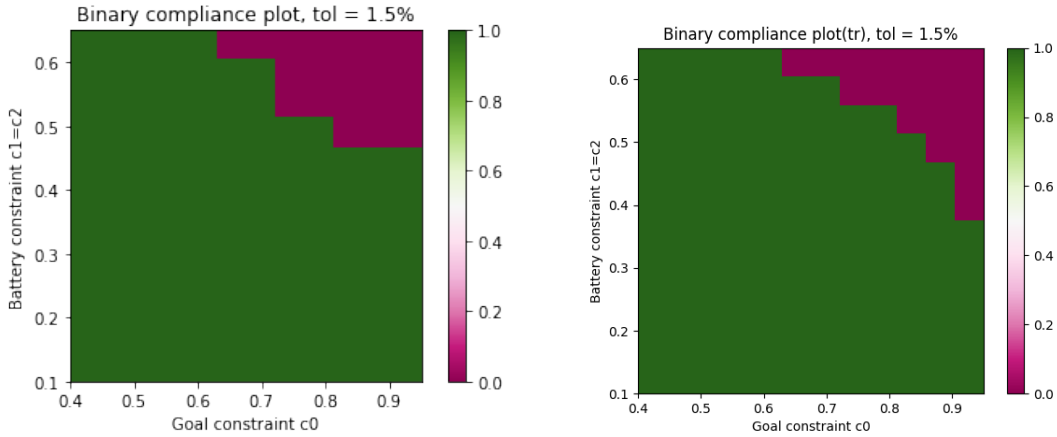


Figura 4.9: Episodio ejecutado para una política distribuida. En azul se muestra el multiplicador λ_G, mientras que en amarillo y negro se grafican $(\lambda_{B1}, \lambda_{B2})$. Para los valores elegidos la única forma de mantener la restricción del *goal* (c_G), es que los agentes opten la mayor parte del tiempo por coincidir en G. Valores finales de cumplimiento medio: $o_G = 0.952, o_{B1} = 0.096, o_{B2} = 0.100$



Mapa de cumplimiento de especificaciones con política centralizada.

Mapa de cumplimiento de especificaciones con política distribuida

Figura 4.10: Mapa de calor del cumplimiento de restricciones para todos los valores posibles de c. Cada gráfico binario muestra para qué valores fue posible cumplir con las restricciones con una tolerancia de al menos un $1.5\,\%$.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 5

Conclusión y trabajo a futuro

En colaboración con la Universidad de Pennsylvania, logramos ensamblar una flota de UAVs de 10 unidades equipada con mini-computadoras Intel NUC a bordo y conectividad Wi-Fi. Este sistema, que logró correr algoritmos sincronizados entre 6 unidades de la flota, es un punto de partida promisorio para futuros trabajos de investigación del Instituto de Ingeniería Eléctrica de Facultad de Ingeniería.

Los experimentos realizados respecto al algoritmo de ruteo probabilístico ensayado en la flota de UAVs, muestran la ventaja de utilizar un equipo de UAVs enrutadores móviles para garantizar conectividad respecto a la alternativa de mantenerlos fijos. Se encuentra en elaboración una publicación científica basada en los resultados de este proceso experimental.

La flota de UAVs es altamente versátil y se encuentra en condiciones de correr algoritmos de optimización distribuida tales como los desarrollados en el Capítulo 4. Estos logran mostrar las potencialidades del aprendizaje por refuerzos en resolver problemas específicados matemáticamente a partir de restricciones, y son un ejemplo de uso de redes neuronales adaptadas específicamente a un problema con variables duales.

Más aún, el algoritmo distribuido formulado en dicho capítulo muestra la posibilidad de agentes de coordinar a pesar de contar con políticas individuales y una representación parcial del estado del sistema. Resta a futuro escalar este algoritmo a una parametrización con redes neuronales y un espacio de estados continuo para resolver un problema más general que el simulado.

Los resultados obtenidos en el caso de un solo agente, y la perspectiva de escalarlos para un sistema multi-agente, configuran la base de una futura publicación científica que avanza las fronteras del estado del arte en el área de CRL multi-agente.

Esta página ha sido intencionalmente dejada en blanco.

Apéndice A

ROS - Robot Operating System

ROS es un *framework* modular para desarrollo de *software* a utilizar en robots[49]. Provee herramientas similares a las de un sistema operativo tales como abstracción a nivel de hardware, control de dispositivos de bajo nivel, implementación de funcionalidades de uso común e intercambio de mensajes entre distintos procesos.

Está basado en una arquitectura de grafo en donde nodos reciben, envían y administran mensajes conteniendo datos de sensores, variables de control, estados y entradas de actuadores.

ROS se puede pensar en tres niveles de abstracción[50]. En el nivel más alto -de comunidad- se encuentran las distribuciones, repositorios, wikis y APIs (*Application Programming Interface*) o librerías de cliente, siendo *rospy* la más notoria - una interfaz para poder programar módulos de ROS en *Python*.

En un segundo nivel se encuentra el concepto de paquete, que funciona a nivel de sistema de archivos. Son la unidad principal de organización y contienen conjuntos de nodos, librerías y archivos.

En el tercer y más bajo nivel (el verdaderamente específico de ROS) se encuentran los nodos, tópicos, servicios y mensajes que conforman el grafo computacional.

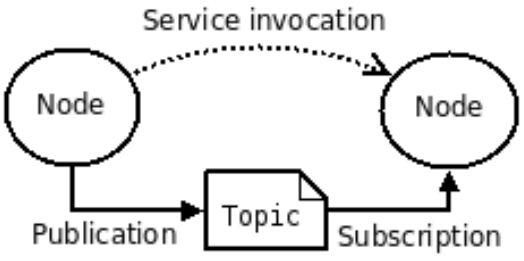


Figura A.1: Esquema nodo-tópico definiendo un grafo computacional. En este esquema los nodos se representan con círculos, y los *topics* se representan como rectángulos que constan de publicadores y suscriptores.

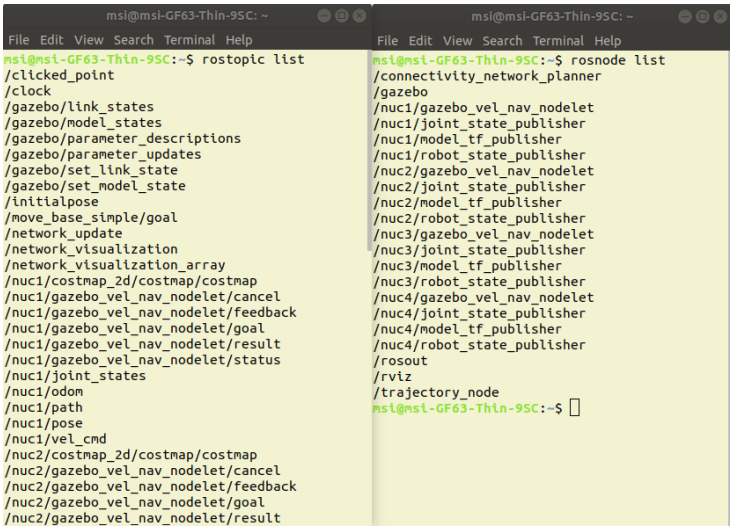


Figura A.2: Ejemplo de lista de *topics* y nodos para simulación *in the loop* (SIL) de sistema de vuelo. El experimento cuenta con 4 agentes denominados nuc1..4, y está simulado en el entorno de SIL Gazebo.

Nodo

Los nodos son los procesos que realizan cálculo y algoritmos. Toman información de tópicos (mensajes con estructura publicación-suscripción que serán definidos más adelante) y pueden (o no) publicar información en alguno de ellos. ROS está diseñado para ser modular a un nivel granular, en el que un sistema de control robótico suele contar de múltiples nodos. Por ejemplo, en el listado de nodos que se ve en la Figura A.2, correspondiente a una simulación *in the loop*[1] de vuelo de 4 UAVs, se cuenta con nodos que controlan la navegación y la dinámica de velocidad (como /nuc1/gazebo_vel_nav_nodelet), nodos que transforman coordenadas entre sistemas de coordenadas (/nuc1/model_tf_publisher), y nodos que se encargan de calcular información a publicar (el nodo /nuc1/robot_state_publisher, por ejemplo, calcula la información a publicar en el tópico /nuc1/pose).

Los nodos se programan con ayuda de alguna de las librerías de cliente (APIs) asociadas: *roscpp* basada en C++, o *rospy* basada en *Python*.

Master

El ROS *master* es un nodo especial que provee capacidad de registro y búsqueda a todo el grafo computacional. Sin él, los nodos no se pueden encontrar, intercambiar mensajes o utilizar servicios. Se invoca a través del comando *roscore*. Como se desarrolla en la Sección 2.1.2, es posible contar con un esquema distribuido, *multi-master*, para que el nodo de sincronización no corra en un único equipo, y además para permitir reducir el flujo de información intercambiada.

[1]Los programas tipo SIL (*software in the loop*) son programas creados con el objetivo de simular el comportamiento de un sistema físico en tiempo real.

Tópicos

Los mensajes entre nodos se intercambian a través de un sistema tipo publicación-suscripción. Los nodos envían información publicando a un tópico, y la reciben suscribiéndose al tópico apropiado. En un mismo tópico pueden haber múltiples suscriptores y publicadores concurrentes, y un nodo puede publicar y suscribir en varios tópicos.

En la simulación de la Figura A.2 se cuenta con *topics* que llevan información del sistema de UAVs simulado y de las herramientas de SIL en sí. El topic nucx/-pose por ejemplo, contiene la información de posicionamiento del agente número x.

Una alternativa a los nodos y *topics* son los servicios y parámetros, pero no entraremos en detalle ya que el esquema base de suscriptor-publicador se mantiene incambiado.

Launch Files

Roslaunch es una herramienta que permite ejecutar sincronizadamente un conjunto de nodos y topics especificados en un archivo *.launch*. Permite generar grupos de nodos en conjunto, así como los *topics* y servicios con los que interactúa cada grupo.

Rospy API

La librería de cliente para ROS que más usamos para el desarrollo de experimentos es *rospy*, que permite programar módulos de ROS en *Python*. En *rospy* se desarrollaron todos los scripts que lanzan los experimentos del Capítulo 5. *Rospy* permite tanto lanzar archivos *.launch* (es decir conjuntos de nodos, tópicos y demás) como interactuar con tópicos, nodos, servicios y parámetros de una manera sencilla. Es una herramienta central ya que varios de los paquetes centrales de ROS están desarrollados en *Python*, a través de *rospy*, como los paquetes *rostopic* (que permite mostrar los tópicos que están activos en un *roscore*) y *rosservice* (que es similar a *rostopic* pero con servicios).[37]

Esta página ha sido intencionalmente dejada en blanco.

Apéndice B

Equivalente SOCP

Es necesario reformular el problema de la Sección 2.3 de forma que sea resoluble por un optimizador convexo, que utiliza las reglas de programación disciplinada quasi-convexa (DQCP)[45]. Para esto, basta escribir la primer restricción de la Ecuación 2.9 como una restricción cónica. Un SOCP está definido por las ecuaciones:

$$\min_{z} \qquad f^T z \tag{B.1}$$

$$\text{sujeto a :} \qquad \|M_l z + n_l\| \leq c_l^T z + d_l, \quad l = 1, \dots L \tag{B.2}$$

Es decir: una función objetivo lineal en la variable de optimización z (B.1), y tantas restricciones como sea necesario de cota sobre la norma de combinaciones lineales de la variable (B.2). Nótese que restricciones lineales de desigualdad son un caso particular de (B.2) con $M_l = 0$.

En (2.9) las variables de optimización son s y los kn^2 elementos de la forma α_{ij}^k, que se pueden pensar en un único vector α de elementos con único índice p : $\alpha_p,\ p = 1 \dots kn^2$. Buscando la forma de (B.2) en (2.4) y (2.5), se observa:

$$\bar{b}_i^k = \sum_{j=1}^{n} \alpha_{ij}^k \bar{R}_{ij}(x_i, x_j) - \sum_{j=1}^{n} \alpha_{ji}^k \bar{R}_{ij}(x_i, x_j) \text{ (es decir, } \bar{b} \text{ es una combinacion lineal de } \alpha)$$

$$\sqrt{\tilde{b}_i^k} = \sqrt{\sum_{j=1}^{n} \left(\alpha_{ij}^k\right)^2 \tilde{R}_{ij}(x_i, x_j) + \sum_{j=1}^{n} \left(\alpha_{ij}^k\right)^2 \tilde{R}_{ij}(x_i, x_j)}$$

Como a efectos del problema x es fijo, $\bar{b}_i^k$ resulta una combinación lineal de elementos de α, y $\tilde{b}_i^k$ una combinación lineal de elementos de α al cuadrado. Por lo tanto, pensando α como vector se puede escribir:

$$\bar{b}_i^k = r^T \alpha$$

$$\sqrt{\tilde{b}_i^k} = \sqrt{\sum_{p=1}^{Kn^2} \alpha_p^2 c_p^2} = \|C\alpha\|$$

Donde se definió $C = \text{diag}([c_1, c_2, \ldots, c_{kn^2}])$. Con estas definiciones (2.9) se reescribe como:

$$\|C\alpha\| \leq r^T\alpha - m - s$$

que resulta una restricción cónica en la variable (α, s) por comparación directa con (B.2). Por otro lado, las restricciones sobre α son restricciones de desigualdad, que son un caso particular de (B.2) como se explicó antes.

Apéndice C

Aprendizaje por Recompensas

En este capítulo explico la formulación general de *Reinforcement Learning*, y algunas de las técnicas que utilicé en la Sección 2.3. En particular, me concentro en la técnica de *policy gradient*, y las parametrizaciones en base a redes neuronales.

Entenderemos por *Reinforcement Learning* (aprendizaje por refuerzos, aprendizaje por recompensas, o RL de ahora en mas), la familia de métodos asociados a decidir acciones a tomar para maximizar una señal de recompensa (o *reward*). En la familia de problemas más completos, las acciones del agente afectan no solo la recompensa inmediata sino el estado futuro y en consecuencia toda futura recompensa[20].

Para llegar a los métodos y parametrizaciones deseados es preciso definir anteriormente:

- Agente y entorno, así como estado, acción y recompensa. Proceso de decisión Markoviano, y la definición probabilística del problema de RL.
- Retornos y funciones de valor. Definición de política de actuación.
- Políticas paramétricas, y *policy gradient*.

C.1. Interacción agente-entorno

Para la formulación matemática en términos de Procesos de Decisión Markovianos (MDPs), es preciso separar el agente, que aprende y toma las decisiones, del entorno, que compone todo lo que es externo al agente.

El agente y el entorno interactúan de manera secuencial cada instante de tiempo discreto $t = 0, 1, 2, ...$ En cada instante, el agente recibe una representación del estado del entorno $S_t \in \mathcal{S}$, y en base a esa información toma una decisión por la acción $A_t \in \mathcal{A}$. En el siguiente instante, recibe una recompensa $R_{t+1} \in \mathbb{R}$ y un estado actualizado S_{t+1}. El proceso se dibuja esquemáticamente en la Figura C.1. En términos de control clásico, es análogo pensar en el agente, entorno y acción como controlador, planta y señal de control respectivamente.

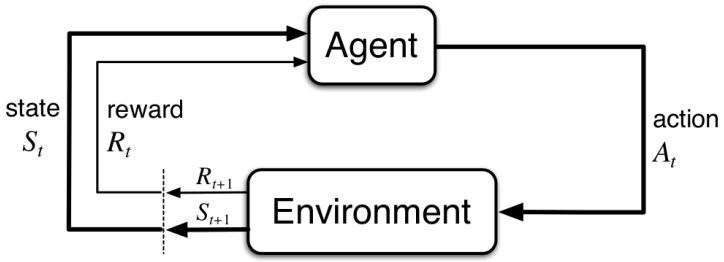


Figura C.1: Interacción agente-entorno: el agente recibe del entorno un estado S_t y una recompensa R_t y decide tomar la acción A_t. Como consecuencia, el entorno evoluciona al estado S_{t+1} y otorga la recompensa R_{t+1} [20].

La dinámica estocástica del proceso queda definida según:

$$p(s', r \mid s, a) \stackrel{def}{=} \Pr\left(S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\right) \tag{C.1}$$

Es decir, el sistema queda definido en tanto quede definida la probabilidad de que dado cualquier estado s y acción a en tiempo $t-1$, la recompensa en tiempo t sea r y el estado s'. Por simplicidad, la Ecuación C.1 representa la dinámica de un sistema discreto (tanto S_t como A_t son variables aleatorias discretas), pero es fácilmente extendible a un sistema continuo, como los que se usan en el Capítulo 4.

La Ecuación C.1 solo define completamente la dinámica en virtud de que el proceso sea Markoviano. Esto es, la probabilidad de cada posible valor para S_t, R_t depende exclusivamente del valor tomado por el estado y acción inmediatamente anterior S_{t-1}, A_{t-1} y dados estos, es descartable toda historia anterior. Esta restricción, que tomamos como hipótesis, es más sobre la definición del estado que sobre el proceso, ya que el estado debe incluir toda la información que sea necesaria para influir el futuro.

Categoría dentro de aprendizaje automático

El aprendizaje por recompensa no califica ni dentro de la familia de algoritmos de aprendizaje automático conocidos como aprendizaje supervisado, en los que el objetivo es aprender a partir de ejemplos etiquetados por un experto, ni dentro de los conocidos como aprendizaje no supervisado, que se suelen asociar a problemas de reconocimiento de estructura (como el *clustering* o el *Principal Component Analysis*).

Según Richard Sutton[20], RL se considera un paradigma independiente de los dos anteriores y como tal tiene dificultades propias, muy notablemente el compromiso exploración-explotación que no está presente en los paradigmas de aprendizaje supervisado o no supervisado: un agente en un contexto de RL tiene una dicotomía permanente entre obtener experiencias estado-acción nuevas con la posibilidad de mejorar las recompensas, o aprovechar las experiencias acumuladas que más recompensa le garantizaron en tiempo pasado.

C.2. Retorno, política y funciones de valor

Todos los algoritmos de RL precisan estimar qué tan bueno es para el agente estar en un determinado estado. El concepto de "bueno" se define en término de la recompensa futura esperada, o el retorno esperado.

Retorno

El concepto de retorno está asociado a la posibilidad de acumulación de recompensa a futuro y queda definido en su versión más general por la ecuación:

$$G_t \stackrel{def}{=} R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{C.2}$$

Donde a $\gamma \leq 1$ se lo conoce como descuento, y tiene el objetivo de priorizar para el cálculo recompensas más cercanas. El caso $\gamma = 1$, que utilizaremos, es un caso particular sin descuento en el que toda recompensa se computa igualmente para el retorno. Otro caso particular de interés es el de episodio de duración finita T que se puede obtener como caso particular de la Ecuación C.2 definiendo $R_{t>T} = 0$.

Política

Formalmente una política es un mapa del espacio de estados a probabilidades de elegir cada acción del conjunto de acciones.

$$\pi(a \mid s) \stackrel{def}{=} \Pr(A_t = a \mid S_t = s) \tag{C.3}$$

Si el agente utiliza la política π en tiempo t, entonces $\pi(a \mid s)$ es la probabilidad de $A_t = a$ si $S_t = s$. Los algoritmos de RL son los que definirán cómo esa política se cambia según la experiencia del agente.

Funciones de valor

La función de valor de un estado s bajo la política π se denota $v_\pi(s)$ y es el retorno esperado de partir en el estado s y evolucionar según la política π y la propia dinámica del proceso. Para MDPs, $v_\pi(s)$, la función de valor de estado para la política π, queda definida por la ecuación:

$$v_\pi(s) \stackrel{def}{=} \mathbb{E}_\pi \left[G_t \mid S_t = s \right] \tag{C.4}$$

donde la esperanza se toma sobre una variable aleatoria dado que el agente sigue la política π. Por otro lado, se define la función de valor estado-acción $q_\pi(s, a)$ como el retorno esperado de empezar en s, tomar la acción a (que no necesariamente es posible bajo π), y luego seguir la política π:

$$q_\pi(s, a) \stackrel{def}{=} \mathbb{E}_\pi \left[G_t \mid S_t = s, A_t = a \right] \tag{C.5}$$

C.3. Políticas Paramétricas

Nos interesan los métodos que se basan en una política paramétrica. Usamos la notación $\theta \in \mathbb{R}^d$ para el vector de parámetros de la política parámetrica $\pi(a \mid s, \theta)$ o $\pi_\theta(a \mid s)$.

Los métodos con políticas parametricas se basan en el gradiente de alguna medida de desempeño escalar $J(\theta)$. Apropiadamente, ya que el objetivo es maximizar esta medida, corresponde actualizar θ según un ascenso por gradientes estocástico en J:

$$\theta_{k+1} = \theta_k + \alpha\widehat{\nabla J(\theta_k)} \tag{C.6}$$

Donde $\widehat{\nabla J(\theta_k)}$ es un estimado estocástico del gradiente. Los métodos que siguen este esquema general se llaman métodos de *policy gradient*.

Parametrización continua

Como ejemplo de parametrización de política, tomamos las redes neuronales de dos capas que se usarán en el Sección 2.3. En un espacio continuo es usual parametrizar la media[1] de una política Gaussiana, esto es:

$$\pi_\theta(a \mid s) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(a - \mu(s,\theta))^2}{2\sigma^2}\right) \tag{C.7}$$

El diseño reside en la parametrización de la media, que puede ser por ejemplo una función lineal de un *feature vector*:

$$\mu(s, \theta) = \theta^T x(s) \tag{C.8}$$

o una red neuronal que tome el vector de estados como entrada y μ como salida.

Red Neuronal de dos capas para parametrización de política

Otra alternativa es parametrizar a partir de una red neuronal completamente conectada. Si se tiene un vector de estado s de dimensión D, y se requiere un vector de salida de dimensión C, es posible construir una red neuronal con una capa oculta de dimensión H como se ilustra en la Figura C.2, esto es:

$$\begin{aligned} z &= sW_1 + b_1 \\ a &= \text{ReLU}(z) \\ \mu &= aW_2 + b_2 \end{aligned} \tag{C.9}$$

donde $W_1 \in \mathbb{R}^{D\times H}$ y $W_2 \in \mathbb{R}^{H\times C}$ son las matrices de pesos, $b_1 \in \mathbb{R}^H$ y $b_2 \in \mathbb{R}^C$ son los bias, y $s \in \mathbb{R}^D$, $\mu \in \mathbb{R}^C$ son la entrada y salida (estado y acción media) respectivamente. La segunda ecuación se puede sustituir por otra no-linealidad en lugar de ReLU, tal como tanh.

[1] Es posible parametrizar tanto la media como la varianza, pero optamos por dejar la varianza como un hiperparámetro fijo por simplicidad.

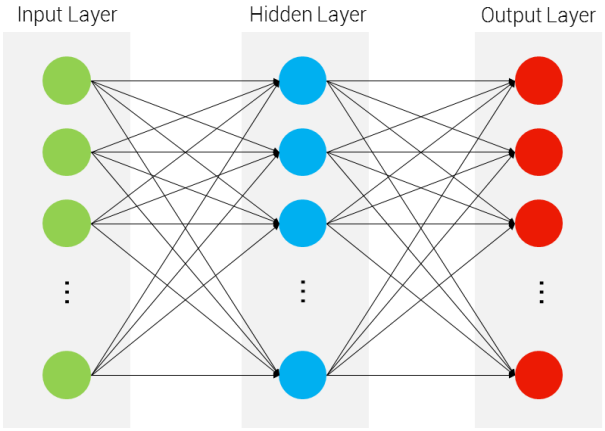


Figura C.2: Diagrama de bloques para una red neuronal de dos capas completamente conectada.

C.4. Algoritmos de entrenamiento

Los algoritmos de *policy gradient* se basan en el *Policy Gradient Theorem*[20]. El mismo, establece que tomando $J(\theta) = v_{\pi_\theta}(s_0)$, se cumple

$$\nabla J(\theta) \propto \sum_s \mu(s) \sum_a q_{\pi_\theta}(s,a) \nabla \pi_\theta(a \mid s) \tag{C.10}$$

Donde $\mu(s)$ es la distribución estacionaria para s bajo la política π_θ. El teorema se puede reescribir[51] convenientemente:

$$\begin{aligned}
\nabla J(\theta) &\propto \sum_s \mu(s) \sum_a q_{\pi_\theta}(s,a) \nabla \pi_\theta(a \mid s) \\
&= \sum_s \mu(s) \sum_a \pi_\theta(a \mid s) q_{\pi_\theta}(s,a) \frac{\nabla \pi_\theta(a \mid s)}{\pi_\theta(a \mid s)} \\
&= \mathbb{E}_{\pi_\theta} \left[q_{\pi_\theta}(s,a) \frac{\nabla \pi_\theta(a \mid s)}{\pi_\theta(a \mid s)} \right] && \mathbb{E}_{\pi_\theta} \equiv \mathbb{E}_{s \sim \mu(s), a \sim \pi_\theta} \\
&= \mathbb{E}_{\pi_\theta} \left[q_{\pi_\theta}(s,a) \nabla \ln \pi_\theta(a \mid s) \right] && \nabla \log x = \frac{\nabla x}{x} \\
&= \mathbb{E}_{\pi_\theta} \left[G_t \nabla \ln \pi_\theta(a \mid s) \right] && q_{\pi_\theta}(s,a) = \mathbb{E}_{\pi_\theta} \left[G_t \mid S_t, A_t \right]
\end{aligned} \tag{C.11}$$

La forma del *Policy Gradient Theorem* de la Ecuación C.11 induce la familia de algoritmos estocásticos conocidos como *policy gradient*, que surgen de estimar $\nabla J(\theta)$ como una muestra de la forma $\hat{G}_t \nabla \ln \pi_\theta(a \mid s)$. Por ejemplo, la regla de actualización del método REINFORCE resulta:

$$\theta_{k+1} = \theta_k + \alpha G_k \nabla \ln \pi_\theta(a \mid s) \tag{C.12}$$

Que resulta en el algoritmo:

Entrada: Política paramétrica diferenciable $\pi_\theta(a \mid s)$, paso α, largo episódico T
Salida : Parámetros entrenados θ

Inicializar, $\theta \in \mathbb{R}^d$, por ejemplo $\theta = 0$;
for *criterio de parada* **do**
  Generar un episodio: $S_0, A_0, R_1, \ldots S_{T-1}, A_{T-1}, R_T$ siguiendo π_θ;
  for $t = 0, 1, \ldots T-1$ **do**
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$;
    $\theta \leftarrow \theta + \alpha \gamma^t G \nabla \ln \pi_\theta(a \mid s)$;

Algoritmo 6: REINFORCE: algoritmo de referencia para *policy gradient.*[20]

Referencias

[1] Manuele Brambilla, Eliseo Ferrante, Mauro Birattari y Marco Dorigo. "Swarm robotics: a review from the swarm engineering perspective". En: *Swarm Intelligence* 7.1 (2013), págs. 1-41.

[2] Marco Dorigo, Guy Theraulaz y Vito Trianni. "Reflections on the future of swarm robotics". En: *Science Robotics* 5.49 (2020), eabe4385.

[3] Jorge Cortes, Sonia Martinez, Timur Karatas y Francesco Bullo. "Coverage control for mobile sensing networks". En: *IEEE Transactions on robotics and Automation* 20.2 (2004), págs. 243-255.

[4] Ali Jadbabaie, Jie Lin y A Stephen Morse. "Coordination of groups of mobile autonomous agents using nearest neighbor rules". En: *IEEE Transactions on automatic control* 48.6 (2003), págs. 988-1001.

[5] Reza Olfati-Saber, J Alex Fax y Richard M Murray. "Consensus and cooperation in networked multi-agent systems". En: *Proceedings of the IEEE* 95.1 (2007), págs. 215-233.

[6] Stephanie Gil, Swarun Kumar, Dina Katabi y Daniela Rus. "Adaptive communication in multi-robot systems using directionality of signal strength". En: *The International Journal of Robotics Research* 34.7 (2015), págs. 946-968.

[7] Annamalai Annamalai y Chintha Tellambura. "Error rates for Nakagami-m fading multichannel reception of binary and M-ary signals". En: *IEEE Transactions on Communications* 49.1 (2001), págs. 58-68.

[8] Andrea Goldsmith. *Wireless communications*. Cambridge university press, 2005.

[9] Sonia Martinez, Jorge Cortes y Francesco Bullo. "Motion coordination with distributed information". En: *IEEE control systems magazine* 27.4 (2007), págs. 75-88.

[10] Yoonsoo Kim y Mehran Mesbahi. "On maximizing the second smallest eigenvalue of a state-dependent graph Laplacian". En: *Proceedings of the 2005, American Control Conference, 2005.* IEEE. 2005, págs. 99-103.

[11] Ethan Stump, Ali Jadbabaie y Vijay Kumar. "Connectivity management in mobile robot teams". En: *2008 IEEE international conference on robotics and automation*. IEEE. 2008, págs. 1525-1530.

[12] Jonathan Fink, Alejandro Ribeiro y Vijay Kumar. "Robust control of mobility and communications in autonomous robot teams". En: *IEEE Access* 1 (2013), págs. 290-309.

[13] Yasamin Mostofi. "Communication-aware motion planning in fading environments". En: *2008 IEEE International Conference on Robotics and Automation*. IEEE. 2008, págs. 3169-3174.

[14] Yasamin Mostofi, Alejandro Gonzalez-Ruiz, Alireza Gaffarkhah y Ding Li. "Characterization and modeling of wireless channels for networked robotic and control systems-a comprehensive overview". En: *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2009, págs. 4849-4854.

[15] Yuan Yan y Yasamin Mostofi. "Robotic router formation in realistic communication environments". En: *IEEE Transactions on Robotics* 28.4 (2012), págs. 810-827.

[16] Alphabet Inc. *Project Loon*. URL: `https://x.company/projects/loon/` (visitado 30-09-2010).

[17] Meta. *Connectivity Lab*. URL: `https://about.fb.com/news/2014/03/announcing-the-connectivity-lab-at-facebook/` (visitado 30-09-2010).

[18] SpaceX. *Starlink*. URL: `https://www.starlink.com/` (visitado 30-09-2010).

[19] Daniel Mox, Miguel Calvo-Fullana, Mikhail Gerasimenko, Jonathan Fink, Vijay Kumar y Alejandro Ribeiro. "Mobile wireless network infrastructure on demand". En: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, págs. 7726-7732.

[20] Richard S Sutton y Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[21] Kamal Golabi, Ram B Kulkarni y George B Way. "A statewide pavement management system". En: *Interfaces* 12.6 (1982), págs. 5-21.

[22] Iordanis Koutsopoulos, Vassiliki Hatzi y Leandros Tassiulas. "Optimal energy storage control policies for the smart power grid". En: *2011 IEEE International Conference on Smart Grid Communications (SmartGridComm)*. IEEE. 2011, págs. 475-480.

[23] Pavlo Krokhmal, Stanislav Uryasev y Grigory Zrazhevsky. "Risk management for hedge fund portfolios: a comparative analysis of linear rebalancing strategies". En: *The Journal of Alternative Investments* 5.1 (2002), págs. 10-29.

[24] Yinlam Chow, Aviv Tamar, Shie Mannor y Marco Pavone. "Risk-sensitive and robust decision-making: a cvar optimization approach". En: *Advances in neural information processing systems* 28 (2015).

[25] Jay H Heizer y Barry Render. *Principles of operations management*. Pearson Educación, 2003.

[26] Stephen Boyd, Stephen P Boyd y Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[27] Vladimir Feinberg, Alvin Wan, Ion Stoica, Michael I Jordan, Joseph E Gonzalez y Sergey Levine. "Model-based value expansion for efficient model-free reinforcement learning". En: *Proceedings of the 35th International Conference on Machine Learning (ICML 2018)*. 2018.

[28] Thomas Degris, Patrick M Pilarski y Richard S Sutton. "Model-free reinforcement learning with continuous action in practice". En: *2012 American Control Conference (ACC)*. IEEE. 2012, págs. 2177-2182.

[29] Timothy Sullivan. *Implementing Policy Gradients on CartPole with PyTorch*. 2018. URL: https://github.com/tims457/RL_Agent_Notebooks/blob/master/Policy%20Gradient%20with%20Cartpole%20and%20PyTorch.ipynb.

[30] George Cybenko. "Approximation by superpositions of a sigmoidal function". En: *Mathematics of control, signals and systems* 2.4 (1989), págs. 303-314.

[31] Songtao Lu, Kaiqing Zhang, Tianyi Chen, Tamer Basar y Lior Horesh. "Decentralized policy gradient descent ascent for safe multi-agent reinforcement learning". En: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 10. 2021, págs. 8767-8775.

[32] Miguel Calvo-Fullana, Santiago Paternain, Luiz FO Chamon y Alejandro Ribeiro. "State Augmented Constrained Reinforcement Learning: Overcoming the Limitations of Learning with Rewards". En: *arXiv preprint arXiv:2102.11941* (2021).

[33] DJI. *FlameWheel 450 User Manual*. English. DJI. Mayo de 2015. 9 **pagetotals**. URL: http://dl.djicdn.com/downloads/flamewheel/en/F450_User_Manual_v2.2_en.pdf (visitado 30-09-2010). Prox.

[34] Juan Bazerque. *How to build a drone*. 2020. URL: https://iie.fing.edu.uy/personal/jbazerque/en/drone/ (visitado 30-09-2010).

[35] *3DR Pixhawk 1 Flight Controller*. URL: https://docs.px4.io/master/en/flight_controller/pixhawk.html (visitado 30-09-2010).

[36] Molex. *Molex 2.4GHz/5GHz Wi-Fi Antenna (100mm)*. URL: https://www.mouser.com/datasheet/2/276/1/2042811200_ANTENNAS-1380485.pdf (visitado 30-09-2010).

[37] *Rospy Package Summary*. URL: wiki.ros.org/rospy (visitado 30-09-2010).

[38] *Wireless ad hoc network*. URL: https://en.wikipedia.org/wiki/Wireless_ad_hoc_network (visitado 30-09-2010).

[39] *Multimaster Package Summary*. URL: http://wiki.ros.org/multimaster_fkie (visitado 30-09-2010).

[40] *Multimaster ROS systems*. URL: http://www.iri.upc.edu/files/scidoc/1607-Multi-master-ROS-systems.pdf (visitado 30-09-2010).

[41] Rajeev Shorey, A Ananda, Mun Choon Chan y Wei Tsang Ooi. *Mobile, wireless, and sensor networks: technology, applications, and future directions*. John Wiley & Sons, 2006.

[42] Daniel Mox, Mikhail Gerasimenko, Mariana del Castillo, Leopoldo Agorio, Miguel Calvo-Fullana, Juan Bazerque, Alejandro Ribeiro y Vijay Kumar. "Mobile Wireless Network Infrastructure on Demand 2". En: *On preparation* ().

[43] *Algebraic Connectivity*. URL: https://mathworld.wolfram.com/AlgebraicConnectivity.html (visitado 30-09-2010).

[44] Fan RK Chung y Fan Chung Graham. *Spectral graph theory*. 92. American Mathematical Soc., 1997.

[45] A. Agrawal y S. Boyd. *Disciplined Quasiconvex Programming*. 2020. URL: https://www.cvxpy.org/tutorial/dqcp/index.html.

[46] Akshay Agrawal y Stephen Boyd. "Disciplined quasiconvex programming". En: *Optimization Letters* 14.7 (2020), págs. 1643-1657.

[47] ESnet Berkeley. *iperf3: A TCP, UDP, and SCTP network bandwidth measurement tool*. 2014. URL: https://github.com/esnet/iperf.

[48] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever y Ruslan Salakhutdinov. "Dropout: a simple way to prevent neural networks from overfitting". En: *The journal of machine learning research* 15.1 (2014), págs. 1929-1958.

[49] *What is ROS?* URL: http://wiki.ros.org/ROS/Introduction (visitado 30-09-2010).

[50] *ROS Filesystem Level*. URL: http://wiki.ros.org/ROS/Concepts (visitado 30-09-2010).

[51] *Policy Gradient Algorithms*. URL: https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html#policy-gradient-theorem (visitado 30-09-2010).

Índice de tablas

Esta página ha sido intencionalmente dejada en blanco.

Índice de figuras

Esta es la última página.
Compilado el lunes 18 julio, 2022.
http://iie.fing.edu.uy/