ISSN 1688-2806


Universidad de la República
Facultad de Ingeniería


# Disuasión de aves mediante vehículos aéreos autónomos

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

Gabriel Rodriguez Frangias

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

Directores de Tesis

Dr. Pablo Monzón ..... Universidad de la República
Dr. Facundo Benavides ..... Universidad de la República

Tribunal

Dra. Matilde Alfaro ..... Universidad de la República
Dr. Martín Llofriu ..... Universidad de la República
Dr. Juan Bazerque ..... Universidad de la República

Director Académico

Dr. Pablo Monzón ..... Universidad de la República

Montevideo
martes 19 agosto, 2025

*Disuasión de aves mediante vehículos aéreos autónomos*, Gabriel Rodriguez Frangias.

ISSN 1688-2806

Esta tesis fue preparada en LaTeX usando la clase iietesis (v1.1).
Contiene un total de 128 páginas.
Compilada el martes 19 agosto, 2025.
http://iie.fing.edu.uy/

# Resumen

Las especies de ave consideradas plaga causan, anualmente, cuantiosas pérdidas económicas a nivel mundial. En Uruguay, la cotorra argentina (*Myiopsitta monachus*) destruye cultivos de cereales, oleaginosos y frutales, causando pérdidas estimadas en 11 millones de dólares por año. En la actualidad se utilizan diversos métodos para mantener a estas aves lejos de los cultivos, los cuales incluyen diferentes mecanismos de disuasión o alguna forma de control poblacional. Con los recientes avances en el área de la robótica y la *IoT* (Internet de las cosas), se están investigando algunas estrategias de disuasión que hacen uso de estas nuevas tecnologías. En la presente tesis de maestría, se explora la utilización de un dron autónomo con el fin de expulsar una bandada de aves de un área cultivada. Se presenta el modelado de un grupo de aves sintéticas junto con un dron simulado, y se analizan diferentes algoritmos para resolver el problema de la disuasión que incluyen heurísticas, técnicas de aprendizaje por recompensas o *RL* y la combinación de ambos enfoques. Los algoritmos desarrollados se trasladan luego a un entorno de simulación Gazebo controlado a través de ROS2, y finalmente se comparan los rendimientos de dichos algoritmos, junto con un análisis de sensibilidad de 4 variables de interés. Los experimentos realizados en los diferentes entornos de simulación muestran resultados prometedores, en donde dos de los algoritmos propuestos logran una efectividad del 100 %, expulsando la totalidad de las aves del área protegida aún bajo condiciones adversas de viento y con bandadas que exhiben muy baja cohesión.

Esta página ha sido intencionalmente dejada en blanco.

# Tabla de contenidos

Tabla de contenidos

Esta página ha sido intencionalmente dejada en blanco.

# Capítulo 1

# Introducción

Los daños causados por las aves constituyen un problema costoso y de larga data en los sectores agrícolas de todo el mundo, especialmente en cultivos de alto valor como arándanos, cerezas y uvas para vino [2,74,75]. Estados Unidos, considerado uno de los cinco principales países productores agrícolas del mundo [23], sufrió pérdidas por más de 800 millones de dólares en 2004, únicamente debido a los estorninos europeos, y cuando se consideran todas las especies de aves, las pérdidas totales ascienden a más de 4700 millones de dólares [56]. En Australia, por ejemplo, se estima un costo anual de 300 millones de dólares australianos [74]. Estas pérdidas incluyen el costo de implementación de los métodos utilizados para controlar el daño causado. Desafortunadamente, muchos de estos métodos han demostrado ser poco eficaces.

En el caso de Uruguay, las especies más dañinas son la paloma torcaza (*Columba palumbus*) y la cotorra (*Myiopsitta monachus*), en especial esta última. La cotorra es una especie de ave nativa que causa daños en cultivos de cereales, oleaginosos y frutales. En 1981, la FAO estimó unas pérdidas totales de 6 millones de dólares por daños en todos los cultivos [20]. En la actualidad, ese número asciende a los 11 millones de dólares anuales, causados por todas las especies de aves [15]. En general, es difícil calcular el daño atribuible exclusivamente a la cotorra, ya que este se da junto con otras especies de aves, además de otros organismos como mamíferos y artrópodos.

Tanto los animales salvajes como domésticos destruyen los cultivos al comerlos y pisotearlos, y además del impacto económico, representan un riesgo para la salud debido a la deposición de heces potencialmente contaminadas sobre o cerca de los cultivos [40]. En particular, las aves pueden albergar patógenos y transmitirlos a las personas a través de los alimentos cosechados. Por ejemplo, los estorninos europeos son una fuente de Salmonella enterica [13], y representan un mayor riesgo de transferencia de patógenos en comparación con otras variables como la densidad del ganado, la gestión de instalaciones o variables ambientales.

El creciente problema de las aves plaga está estrechamente relacionado con la actividad humana [10,12,50]. Más del 40 % de la tierra del planeta ha sido convertida en áreas cultivables o desarrollos urbanísticos, mientras que los hábitats naturales que aún existen se encuentran fragmentados [24]. Esto, sumado al incremento en el uso de agroquímicos, ha resultado en una pérdida de biodiversidad y en el deterioro de una de las funciones clave de los ecosistemas: el control natural de plagas [60]. Por ejemplo, ciertos hábitats naturales albergan especies de aves que consumen insectos dañinos o proveen lugares de reposo a aves depredadoras, cuya sola presencia ahuyenta a otras aves o roedores considerados plagas [43].

De todas las especies animales que pueden invadir un campo cultivado, las aves

son las más difíciles de controlar. En este caso, no basta con simplemente cercar el predio a proteger.

Por esta razón, se han diseñado e implementado diversos métodos de disuasión, a nivel mundial, que se pueden clasificar en 9 categorías [60], según la tabla 1.1, en donde se indican algunos ejemplos. La categorización es solo orientativa, ya que algunos métodos pueden pertenecer a más de una categoría.

| Categoría | Ejemplos |
|---|---|
| Visual | Láseres, espantapájaros, modelos de depredadores, piezas disecadas, globos con puntos oculares, cometas con forma de halcón, espejos, cintas reflectoras, luces estroboscópicas, banderas, aves rapaces entrenadas (cetrería). |
| Auditivo | Cañon acústico de propano, petardos, screamers (un tipo de pirotecnia), silbatos, disparos de armas de fuego, llamadas de socorro, sonidos ultrasónicos, sonidos de alta intensidad, red sónica. |
| Táctil | Púas, sustancias pegajosas. |
| Modificación de hábitat | Cultivos señuelo, eliminación de estructuras de descanso o refugio. |
| Exclusión | Redes, cercas eléctricas, cables suspendidos, dispositivos anti-perching. |
| Químicos | Repelentes como Antranilato de metilo, antraquinona, DRC-1339, Keyplex-350. |
| Reproductivos | Vacunas anticonceptivas, esterilizantes. |
| Letales | Trampas, venenos, armas de fuego, destrucción de huevos o nidos. |
| Multifacético | Gel óptico, halcones, drones. |

Tabla 1.1: Métodos de disuasión de aves comúnmente utilizados

Los métodos en la categoría visual tienen cierta efectividad por un corto período de tiempo, hasta que el ave se habitúa a ellos. Por ejemplo, globos con puntos oculares, como los mostrados en la figura 1.1(a), se utilizaron en viñedos de Nueva Zelanda, pero no resultaron económicamente viables ni efectivos [26]. En otro estudio se utilizaron láseres, como los de la figura 1.1(e), para dispersar cuervos que funcionaron en un primer momento, pero las aves volvieron a sus lugares de pernocte el mismo día en que los láseres se activaron y dichos lugares nunca fueron abandonados en forma definitiva [32].

Los métodos que utilizan sonido suelen ser más efectivos, pero al igual que los mecanismos visuales, las aves acaban por habituarse a ellos. Tienen el problema añadido de las quejas por ruido por parte de los vecinos de los productores. El método más común en esta categoría es el cañón de propano, mostrado en la figura 1.1(c), que tiene sus propios inconvenientes: debe ser reposicionado en forma periódica para ser efectivo, y además produce un sonido característico previo a la detonación que advierte a las aves, las cuales abandonan el lugar y vuelven luego de la explosión [60].

Otro mecanismo en la categoría acústica es el sonido sintético emitido mediante parlantes, que simulan llamadas de auxilio. Estas señales auditivas son más efectivas en el corto y medio plazo, siempre y cuando se incluya algún tipo de variación en el ritmo y número de señales emitidas [6] [59]. Uno de los problemas que presenta

esta solución es que puede atraer depredadores a la zona, lo cual plantea sus propios desafíos.

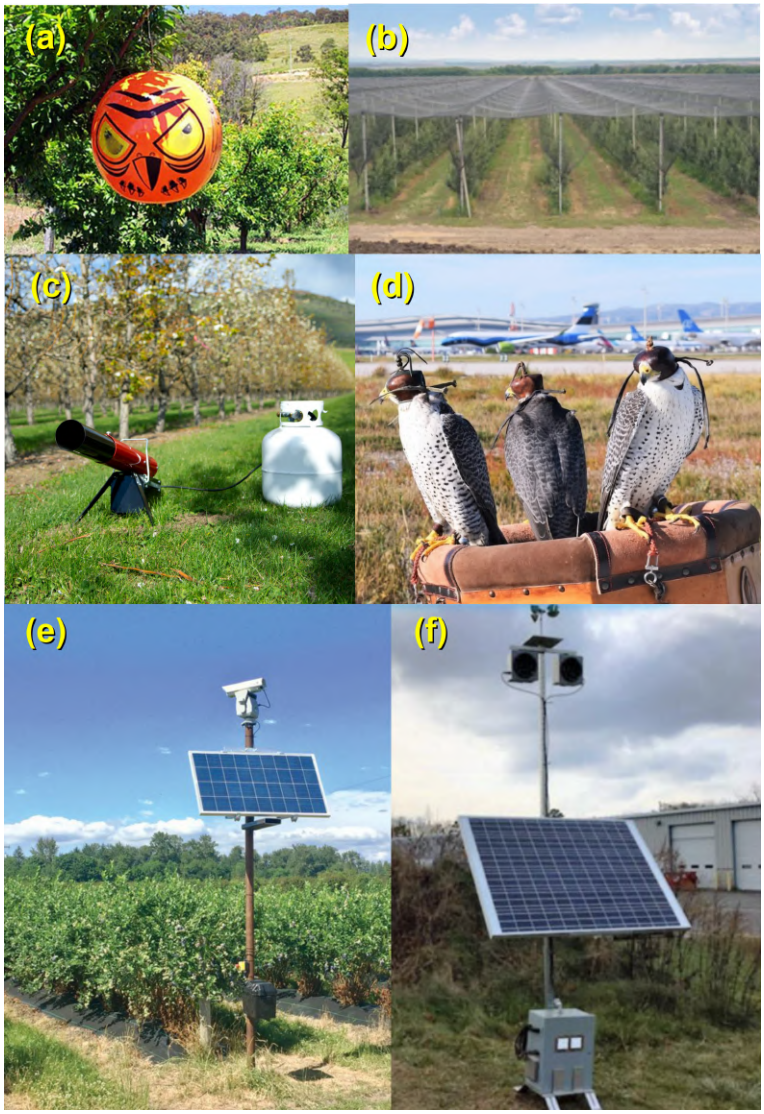



Figura 1.1: Algunos métodos de disuasión: (a) Globos con puntos oculares. (b) Redes. (c) Cañón acústico de propano. (d) Halcones en aeropuerto. (e) Láser (f) Red sónica

Otro método acústico es la denominada red sónica, mostrado en la figura 1.1(f). Este mecanismo emite un ruido blanco que se superpone con el rango de frecuencias que utilizan las aves en sus vocalizaciones, haciendo que no puedan comunicarse en forma efectiva. Se teoriza que al no poder escuchar las llamadas de alerta o auxilio, las aves perciben la zona protegida como de alto riesgo, y en consecuencia la abandonan. Por otro lado, al no poder superar la barrera de comunicación, a las aves les resulta difícil habituarse a esta señal acústica. En un estudio realizado en un campo aéreo, se constató una reducción del 82 % en la presencia de aves, y esta efectividad se mantuvo

durante varias semanas [72]. En principio, este mecanismo parece prometedor, pero son necesarios más estudios para comprobar su efectividad con otras especies no observadas en el campo aéreo, y el efecto que pueda tener sobre poblaciones de aves que no dañan los cultivos.

En cuanto a los mecanismos de exclusión, el más efectivo es cubrir el área cultivada con redes, como se ve en la figura 1.1(b). A pesar de tener buenos resultados, no es un método perfecto. Suele tener un costo muy elevado, sobre todo si el área a proteger es muy grande, y además, las redes son propensas a dañarse. También representan un peligro para otros animales que pueden quedar atrapados en ellas [60].

La categoría de modificación de hábitat abarca diferentes opciones, como por ejemplo la eliminación de estructuras de refugio, descanso o reproducción que estén cerca de los cultivos a proteger. Otra estrategia es la utilización de ciertos cultivos, como girasol o maíz, en zonas aledañas que sirvan de señuelo y que desvíen la atención de los cultivos de interés comercial [35]. Este método suele requerir la colaboración de los productores para compartir los gastos de mantener áreas comunes con cultivos señuelo.

Los métodos químicos, como repelentes, suelen ser costosos, difíciles de aplicar y su efectividad no está asegurada. Un estudio sobre el uso de antranilato de metilo muestra una reducción de más del 90 % en las pérdidas causadas por las aves [5], mientras que otro estudio muestra que esto mismo fue muy poco efectivo [2]. Los resultados varían según el tipo de cultivo y la especie de ave que se intente ahuyentar.

Por su parte, los métodos letales, como venenos o destrucción de huevos y nidos, también son costosos e insumen mucho tiempo, además de estar restringidos por la normativa más reciente. Por otro lado, están mal vistos por la opinión pública por ir en contra de los principios de conservación de la biodiversidad y la gestión ambiental [45]. El uso de venenos puede, además, afectar a otras especies, representando así un riesgo para el medioambiente.

Desde el siglo 19, el uso de halcones y búhos se ha reconocido como beneficioso para la agricultura, pero solo en las últimas décadas la cetrería se ha popularizado como método de disuasión de aves plaga en Estados Unidos [60]. Halcones entrenados, como los mostrados en la figura 1.1(d), se utilizaron inicialmente en viñedos y su presencia se tradujo en una reducción significativa del número de aves presentes en las zonas protegidas. Se reportó hasta un 95 % de reducción en las pérdidas de cultivos en comparación con los viñedos sin halcones [43]. En el Reino Unido, los halcones se utilizaron de forma efectiva en vertederos, para ahuyentar gaviotas de cabeza negra y cuervos, aunque no fueron efectivos con gaviotas de mayor tamaño [7]. En cualquier caso, este método es costoso e impráctico. Se necesitan años de entrenamiento, un cuidado especial y una licencia para poder utilizarlos. Además, para ser efectivos, su presencia debe ser permanente; de lo contrario, las aves plaga suelen regresar. Esto lo hace una solución aún más costosa. De todas formas, este método es uno de los que tiene mayor aceptación en la opinión pública, por considerarse una opción más natural.

En Uruguay, como ya se mencionó, el mayor problema lo plantea la cotorra (*Myiopsitta monachus*), mostrada en la figura 1.2(a), también conocida como cotorra argentina y que se considera una plaga desde 1947. Esta especie se reproduce en nidos comunitarios, en general a gran altura, como el que se ve en la figura 1.2(b), y el cuidado de los pichones es compartido por los miembros del grupo [11]. Esto representa una ventaja significativa, debido a que de esta forma se reduce la mortalidad de los pichones y, por tanto, el número de individuos crece en forma sostenida [15].

Históricamente, el control de la especie se basó en métodos letales mediante el uso de venenos aplicados directamente a los nidos. Un ejemplo es la técnica de la grasa, detallada en un manual de la DGSA (Dirección General de Servicios Agrícolas, MGAP) [21], que consistía en untar los nidos con una mezcla de 1 kg de grasa de


Figura 1.2: (a) Myiopsitta monachus (b) Nido comunitario

mineral de litio y 130 cc de Carbofurán, una sustancia tóxica clasificada como categoría 1, es decir, altamente peligrosa, la cual solo se consigue con receta profesional [21].

Los operarios, usando una vestimenta protectora, impregnaban con esta mezcla un cepillo articulado que se unía al extremo de una larga caña. Luego, trepando a los árboles o montados a caballo, aplicaban el veneno a cada nido. La figura 1.3 muestra un extracto de las instrucciones contenidas en el manual. La sustancia tóxica debía manejarse con extremo cuidado, por lo que se incluían instrucciones precisas sobre cómo lavar la ropa y limpiar las herramientas utilizadas luego de completar la tarea. Se advertía, además, no lavar nada en ríos o arroyos. De más está decir que era un método poco amigable con el ambiente.


Figura 1.3: Extracto del manual de la DGSA (Dirección General de Servicios Agrícolas) [21], que instruía sobre como aplicar el veneno a los nidos de cotorras.

Esta práctica se termina abandonando hacia el año 2002, en favor de técnicas de protección de cultivos, como repelentes visuales y acústicos, redes y mecanismos para

disminuir la disponibilidad de alimento o lugares de reproducción [51].

Desde el decreto 343 de 2002, y el acuerdo INIA-DGSA, se investigan técnicas de control de reproducción utilizando aceite mineral, que tiene un bajo impacto ambiental. Con este objetivo, en 2009, técnicos uruguayos visitaron el *National WildLife Research Center* (NWRC) para evaluar la posibilidad de una transferencia tecnológica sobre algunos métodos contraceptivos, habiendo observado los buenos resultados obtenidos en Estados Unidos.

El método de control poblacional mediante el uso de aceite mineral consiste en aplicar el aceite sobre los nidos cuando las cotorras están incubando (noviembre). Esto reduce la tasa de eclosión de los huevos debido a que el aceite bloquea los poros del huevo, causando la muerte del embrión por asfixia. A diferencia del veneno, este es un método no tóxico y amigable con el ambiente [20].

Finalmente, los drones y vehículos aéreos no tripulados en general, se utilizan cada vez con más frecuencia en la agricultura para realizar recuentos de cultivos, identificar áreas con posible presencia de enfermedades o insectos y evaluar el terreno antes de la siembra [60].

También se está explorando la viabilidad de utilizarlos para proteger cultivos de aves plaga [8, 9, 76, 78, 80]. A diferencia de otros métodos, las aves no pueden anticipar cuándo ni dónde aparecerá un dron, y como la mayoría no ha estado expuesta a drones previamente, es más probable que estos sean identificados como depredadores. Otra ventaja importante es que a los drones se les puede dotar de elementos visuales y auditivos que refuercen la amenaza percibida por las aves y que a la vez eviten la habituación [60].

El uso de drones no es, por supuesto, una solución definitiva ni perfecta al problema de las aves plaga, aunque sí puede contribuir en forma indirecta a la reducción de su población al restringir el acceso a las fuentes de alimento. El problema del control poblacional, sin embargo, requiere combinar el método de disuasión con otros mecanismos, como el control reproductivo, además de la implementación de políticas públicas adecuadas.

Aun así, estos vehículos tienen un gran potencial de aportar valor al sector agrícola uruguayo en general, y al problema de la protección de cultivos en particular. Por esta razón, es que el presente trabajo se enfoca en explorar el uso de un dron autónomo para abordar dicho problema. Con este objetivo, primero se estudia un posible modelo de simulación de individuos que exhiben una conducta de grupo, y luego se desarrollan algoritmos de control, basados en heurísticas y aprendizaje por recompensas, para guiar un dron que expulse una bandada de aves simuladas de un espacio aéreo determinado.

En este sentido, se propone la siguiente hoja de ruta para los próximos capítulos:

- Capítulo 2: Antecedentes de los trabajos relacionados al problema planteado.
- Capítulo 3: Se aborda el modelado del dron así como también el de una bandada de aves genérica.
- Capítulo 4: Se desarrolla una heurística para la resolución del problema.
- Capítulo 5: Se estudia la alternativa de aplicar Aprendizaje por Recompensas, o *RL* (*Reinforcement Learning*) al problema.
- Capítulo 6: Se explora la combinación de *RL* con la heurística presentada en el capítulo 4.
- Capítulo 7: Aquí se trasladan los algoritmos desarrollados en los capítulos previos a un entorno de simulación Gazebo, controlado a través de ROS2.
- Capítulo 8: Se presentan los resultados de una serie de experimentos, junto con un análisis de sensibilidad de 4 variables de interés.
- Capítulo 9: Conclusiones y trabajos a futuro.

# Capítulo 2

# Antecedentes

## 2.1. Comportamiento de las aves

La utilización de un vehículo aéreo no tripulado (UAV), o dron, para resolver el problema de la protección de cultivos, pretende emular un mecanismo natural de disuasión: el depredador. En este sentido, se han realizado varios estudios orientados a evaluar la amenaza percibida por las aves ante la presencia de este tipo de vehículos.

Para que los UAV's se conviertan en depredadores verdaderamente efectivos, es necesario incorporar elementos del comportamiento de las aves en su diseño. El objetivo es identificar los factores desencadenantes que hacen que las aves adquieran respuestas evasivas a largo plazo contra los depredadores [80].

El comportamiento de las aves en presencia de un depredador se ha estudiado extensamente en el campo de la etología [16,17,33]. La mayoría de los estudios sobre este tema buscan comprender los procesos mentales que provocan la respuesta del individuo ante la presencia de un depredador, pero rara vez abordan la problemática de la protección de cultivos. Sin embargo, estos estudios proporcionan valiosos conocimientos que ayudan a comprender cómo un UAV puede ser percibido como una amenaza.

La respuesta típica de las aves frente a un depredador es la llamada de alarma o auxilio. La primera alerta al resto del grupo sobre la presencia de una amenaza, mientras que la segunda es producida por un individuo que ha sido atrapado. Existe evidencia de que las vocalizaciones de alerta o auxilio emitidas por una especie resultan reconocidas por otras especies, debido a un fenómeno de evolución convergente que hace que estas vocalizaciones resulten similares entre sí [47, 48, 68, 71].

Sin embargo, y como ya se mencionó en el capítulo anterior, el uso de estas señales acústicas en forma aislada no resulta efectivo en el mediano plazo debido a que las aves acaban por habituarse a ellas. La evidencia empírica muestra que las vocalizaciones de alarma o auxilio son mucho más efectivas y su efecto es más duradero, si se las combina con otros estímulos que transmitan amenaza [34].

Con este objetivo, en el artículo [80] Wang y colaboradores presentan el resultado de una serie de experimentos realizados en varios viñedos australianos, en donde una de las especies objetivo era el cuervo australiano (*Corvus coronoides*). Allí se utilizó el hexacóptero mostrado en la figura 2.1, un tipo de vehículo que tiene ventajas con respecto a los drones de ala fija, debido a que pueden ejecutar maniobras aéreas en espacios más reducidos. Por otro lado, los drones de ala fija requieren operar en alturas mayores y en condiciones climáticas más restrictivas [38].

Al dron se lo equipó con un parlante que emitía diferentes vocalizaciones de alarma y auxilio, las cuales se reproducían en forma cíclica. Adicionalmente, y de acuerdo

con los resultados de las investigaciones en comportamiento de las aves, se reforzó el estímulo de amenaza añadiendo un elemento más de disuasión: se instaló, en la parte inferior del vehículo, un cuervo disecado orientado hacia abajo y con las alas abiertas, emulando haber sido atrapado por el depredador.

Figura 2.1: Dron con taxidermia utilizado en [80]

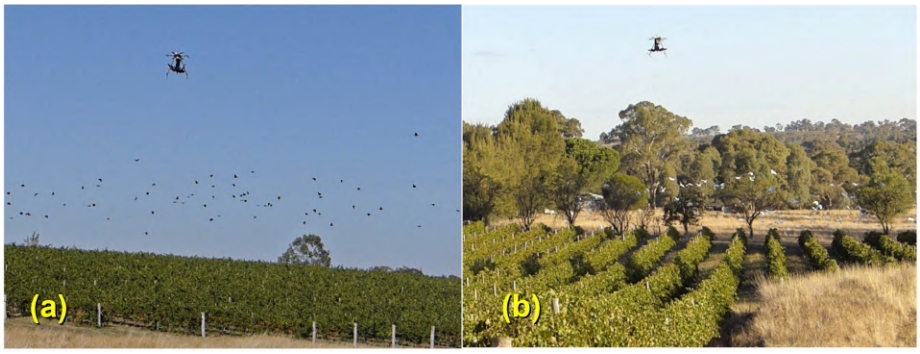



Figura 2.2: Pruebas en viñedos australianos. Dos bandadas huyen ante la presencia del dron. (a) Bandada de 100 cuervos aproximadamente. (b) Bandada de 50 cacatúas aproximadamente.

El vehículo era operado manualmente durante un tiempo máximo de 10 minutos, hasta que la batería se agotara o hasta que las aves fueran expulsadas del viñedo. El operador simplemente dirigía el vehículo hacia el centro de la bandada simulando una persecución. La figura 2.2 muestra el dron ahuyentando una bandada de cuervos (a) y un grupo de cacatúas (b).

Las principales variables observadas fueron: el radio de influencia del dron, la

duración de dicha influencia y su efectividad. El radio de influencia se determinó midiendo la distancia del dron a las aves en el momento en que se observó una reacción por parte de éstas. La duración de la influencia es el tiempo que tardaron las aves en volver a la zona, y la efectividad se refiere al número de aves presentes antes y después de haber volado el dron.

Con respecto al radio de influencia, los resultados muestran una importante variación según la especie de ave, el tamaño de la bandada y la hora del día en que se efectúa el experimento, pero en todos los casos, la distancia medida nunca fue menor a 50 metros.

En cuanto a la efectividad, el dron logra expulsar el 100 % de las aves de las zonas sobre las que vuela. Una vez alejadas de la influencia del dron, las aves se asientan a unos 450 metros de distancia, aproximadamente, y en general tardan al menos 2 horas en volver a la zona cultivada luego de que el dron ha dejado de operar. Algunos grupos regresan antes, pero se mantienen al margen de la zona protegida, presumiblemente debido a que aún perciben una amenaza.

Si bien este estudio de campo muestra resultados prometedores en cuanto a la viabilidad de los drones como elemento disuasorio, los datos no son suficientes para determinar el efecto a mediano y largo plazo de esta solución. Adicionalmente, un dron operado en forma manual es poco práctico en estos casos, además de costoso. Idealmente, el vehículo utilizado debería poder operar en forma autónoma para ser efectivo.

## 2.2. Automatización del UAV

En ese sentido, el artículo [78], de los mismos autores que el artículo previo [80], intenta abordar la cuestión de la automatización del UAV. Para lograr esto, sin embargo, se realizan algunas simplificaciones importantes.

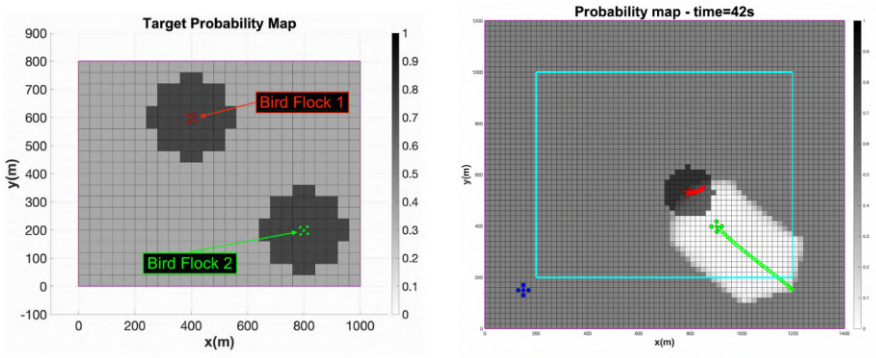
En primer lugar, la bandada se modela como un círculo de radio fijo preestablecido que circunscribe a todo el grupo de aves. El movimiento de la bandada se reduce entonces al movimiento del centro de dicho círculo. Lo que se asume aquí es que las aves toman decisiones en forma colectiva, formando así un grupo o bandada que se comporta como una entidad indivisible.

Por otro lado, el área a proteger se asume de dos dimensiones por considerarse que los cultivos se encuentran en un plano, y que las aves que vuelan más allá de cierta altura no representan una amenaza. De esta forma, las aves y el dron se modelan como objetos bidimensionales, más concretamente, como puntos en el plano. Tanto el estado del dron $X_{UAV}$ como el de las aves $X_{bird}$ incluye su posición y su velocidad lineal, esta última representada por la magnitud $V$ y el ángulo de orientación $\Psi$ respecto a un sistema de coordenadas global.

Mediante una red de sensores, tanto en tierra como a bordo del dron, se modela el área cultivada como una cuadrícula o matriz de ocupación bidimensional, en donde cada celda contiene la probabilidad de presencia de aves en el área representada por dicha celda, como se muestra en la figura 2.3(a). Las celdas más oscuras indican una mayor probabilidad de presencia de aves.

Al mismo tiempo, se mantiene una segunda matriz de idénticas dimensiones, pero cada celda de esta matriz simula el «interés» de las aves por dirigirse a ella, mediante un número $i$ entre 0 y 1. Inicialmente, todas las celdas son igualmente atractivas, con un valor nominal $i = 0.9$, pero cuando el dron pasa sobre ellas, su valor de interés $i$ cae a cero, indicando que ninguna ave se sentirá atraída hacia ella, al menos por un tiempo.

La influencia del dron decae con la distancia, por lo que el valor $i$ de las celdas aumenta gradualmente a medida que ésta se encuentra más alejada del dron, como




(a) (b)

Figura 2.3: (a) Matriz de ocupación (b) Matriz de interés. El paso del dron reduce temporalmente el valor $i$ tanto de las celdas visitadas como de las celdas adyacentes que se encuentran dentro de cierto radio de influencia.

muestra la figura 2.3(b), en donde las celdas más claras (menor valor $i$) son las que están alrededor de la trayectoria del vehículo.

Por otro lado, el valor $i$ de una celda vuelve gradualmente a su valor nominal luego de que el dron haya pasado por ella. Esto permite modelar la transitoriedad del efecto disuasorio que tiene la presencia de un depredador.

Con estos elementos, se construye un algoritmo que en cada paso selecciona una celda objetivo a la cual dirigir el dron (*waypoint*), una vez detectada la presencia de intrusos. Esta selección se basa en la optimización de una función que pondera costos y beneficios, y cuyos componentes son:

- Distancia Euclidiana entre las coordenadas del dron y el centro de la celda candidata.
- Cambio de orientación $\Psi$ que debe realizar el dron para dirigirse a la celda candidata.
- Probabilidad de presencia de aves en la celda candidata.

Optimizar esta función involucra 3 objetivos simultáneos: encontrar la celda más cercana al dron, que tenga la mayor probabilidad de contener aves y que implique una menor variación en el ángulo de giro para dirigirse hacia ella. La celda objetivo resultante se ubica, en general, sobre el círculo que circunscribe a la bandada.

Cuando el dron avanza y comienza a acercarse a la celda donde se encuentra el centro de la bandada, el interés $i$ de dicha celda comienza a reducirse. Cuando cae por debajo de cierto umbral, la bandada se desplaza a una celda adyacente con un mayor valor de $i$. Como el dron reduce el valor $i$ de las celdas frontales a él a medida que avanza, la bandada tiende a desplazarse en dirección opuesta al dron, hacia celdas que resulten más atractivas.

Esta dinámica se muestra en la figura 2.4. En el instante inicial $t = 0$, la celda en donde se encuentra la bandada tiene un valor nominal de interés $i = 0.9$. Por su parte, la celda ocupada por el dron tiene un valor $i = 0$. La escala de grises muestra la variación del valor $i$ de las celdas adyacentes al dron.

Cuando el dron avanza y se aproxima a la bandada, la celda ocupada por ésta se vuelve cada vez menos atractiva. En $t = 10$, el valor $i$ de esta celda cae hasta 0.7, el umbral mínimo necesario para que la bandada permanezca inmóvil. A partir de ese

instante, la bandada comienza a desplazarse, buscando una celda adyacente con mayor valor $i$. En $t = 20$ y $t = 30$, la bandada se dirige hacia la celda *target*, alejándose del dron en dirección opuesta.

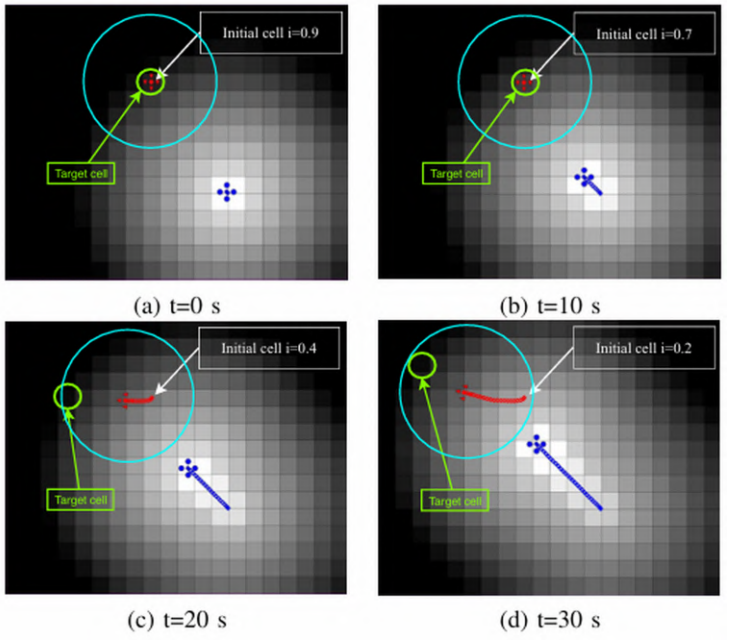



Figura 2.4: Comportamiento evasivo de la bandada ante la presencia del dron. (a) La bandada permanece inmóvil mientras ocupa una celda con interés nominal $i = 0.9$. (b) La celda ocupada por la bandada alcanza el umbral mínimo de interés, $i = 0.7$. (c) y (d) la bandada se desplaza hacia la celda marcada como *target*.

Las simulaciones de este algoritmo muestran que es un método efectivo para expulsar a las aves, al menos con el modelo simplificado de bandada utilizado, y los resultados parecen ajustarse a las observaciones experimentales registradas en el estudio de campo previo [80]. De todas formas, los autores sugieren validar los resultados con drones y aves reales.

En [76], Wang y colaboradores extienden las ideas del trabajo previo. Añaden nuevos componentes a la función de costos para incorporar información sobre la dirección de movimiento de la bandada, así como también un modelo para estimar el consumo de energía realizado por las aves, que depende de la especie, su velocidad media de desplazamiento, la distancia entre la fuente de alimentos y los nidos, etc. Se explora también la opción de utilizar múltiples drones simultáneamente y un posible mecanismo de coordinación de los mismos. Nuevamente, se realizan experimentos simulados con resultados positivos, los cuales se recomiendan contrastar con pruebas reales.

Por su parte, en [9] finalmente se realizan experimentos reales con un dron autónomo, aunque la estrategia utilizada es muy diferente a lo visto hasta ahora. En este caso, el dron (real) está programado para ejecutar dos misiones diferentes: interceptar y patrullar. La primera misión se refiere a interceptar un ave (o bandada) que se aproxime por un sector externo al campo cultivado. La idea es salir al encuentro del intruso antes de que logre penetrar el perímetro protegido. Por otro lado, la misión de patrullaje se refiere a ejecutar trayectorias predefinidas sobre el campo para ahuyentar aves que ya se encuentren en él. La figura 2.5 ilustra esta idea.

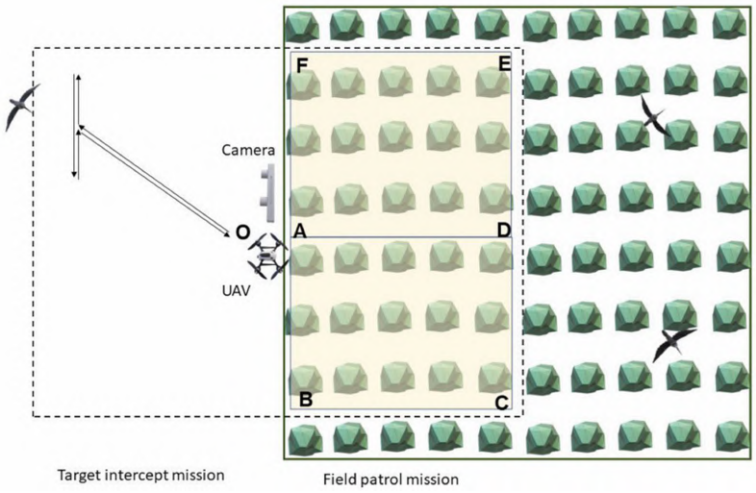



Figura 2.5: Intercepción y patrullaje [9]. La línea punteada indica el área de operación del dron. La misión de intercepción se ejecuta fuera de los limites del campo, para evitar que las aves ingresen al predio protegido. Los puntos A, B, C, D, E, F y O indican los *waypoints* que determinan la ruta seguida por el dron durante las misiones de patrullaje.

Partiendo desde su base, punto O, el dron es activado cuando los sensores detectan aves aproximarse al campo por el borde exterior. El dron es entonces desplegado y enviado a las coordenadas identificadas por los sensores. Una vez allí, ejecuta un movimiento oscilatorio de izquierda a derecha, hasta ahuyentar a los intrusos. Por otro lado, la misión de patrullaje consiste en recorrer el campo siguiendo los puntos *OABCDEF*. Esta misión se ejecuta en forma regular, siempre y cuando no se detecten intrusos que activen la misión de intercepción.

Para la detección de las aves, se utiliza una cámara estereoscópica fija, que apunta hacia el exterior del campo. Este tipo de cámaras cuenta con dos lentes que permiten capturar dos imágenes simultáneas ligeramente diferentes. Mediante software especializado, estas imágenes se procesan para obtener una nube de puntos que constituyen un mapa tridimensional del entorno. De esta forma, se pueden estimar distancias y coordenadas de los objetos fotografiados [73].

El reconocimiento de las aves en movimiento se basa en un trabajo previo de los mismos autores, en donde combinan técnicas de super-resolución con aprendizaje profundo, como redes convolucionales, que les permitieron obtener una precisión de hasta 90 % en la clasificación de las imágenes [8]. Cabe mencionar que la detección de aves no aplica al interior del campo cultivado. Allí, el dron navega a «ciegas», en el sentido de que solo ejecuta trayectorias preprogramadas que no dependen de la presencia ni la posición de posibles intrusos. Estas trayectorias se construyen con una secuencia de *waypoints* que es controlada desde tierra por la *GCWS* (*Ground Control WorkStation*).

Adicionalmente, se añadieron señales auditivas con llamadas de auxilio y de alarma, como sugiere la literatura mencionada con anterioridad. Estas señales se emitían tanto desde el dron como desde tierra, a través de parlantes fijos colocados en el perímetro del campo cultivado.

Otro elemento interesante fue la inclusión de movimientos aleatorios de corta duración (2 a 3 segundos) en las misiones de patrullaje, como ser giros de 360 grados, cambios súbitos de altitud o vuelo estacionario. Su objetivo era evitar que las aves se

habituaran a las trayectorias preprogramadas.

Los experimentos realizados se enfocaron en medir la confiabilidad de todo el sistema para poder ejecutar las misiones, y no tanto en su eficacia para ahuyentar intrusos. En ese sentido, se verificó la capacidad de procesar y reconocer aves en tiempo real a razón de 30 fps, lo cual se evaluó como satisfactorio.

Por otro lado, se midió la capacidad del dron para ejecutar y completar las trayectorias definidas por la *GCWS*. En este apartado, se alcanzó una tasa de éxito superior al 90 %. Sin embargo, se constataron problemas de precisión en el cálculo de coordenadas 3D que se encontraban a más de 5 metros de distancia. Esto fue debido, principalmente, a que el sistema no contaba con GPS. Adicionalmente, en algunos experimentos, el dron no pudo volver a su base debido a ráfagas de viento que superaban su tolerancia de operación.

Los artículos mencionados hasta ahora ( [8,9,38,76,78,80,80]) enfocan su atención en las estrategias de disuasión, ya sea mediante estímulos visuales o acústicos, así como también en el plan de vuelo del vehículo aéreo. Sin embargo, simplifican en gran medida el modelo de la bandada de aves al representarla como un círculo de radio fijo que se desplaza en un plano. Esto puede dar lugar a resultados distorsionados que se alejan de la realidad. Si bien añade complejidad, contar con un modelo de agrupación o agregación de individuos resulta fundamental a la hora de evaluar posibles soluciones al problema de la disuasión de aves.

## 2.3. Modelado de la bandada

La formación de bandadas es un tipo de comportamiento animal colectivo observado en las aves, fuertemente influenciado por la relación depredador-presa. Estudios empíricos han demostrado que las bandadas son más grandes, se mantienen durante más tiempo y muestran mayor actividad y vigilancia, cuando están en estado de alerta ante la posible presencia de depredadores [37]. Comportamientos similares se observan en otras especies, como peces, mamíferos o insectos.

Uno de los pioneros en la modelización de estos comportamientos fue Craig Reynolds, quien en 1987 publica el artículo «*Flocks, herds, and schools: a distributed behavioral model*» [58].

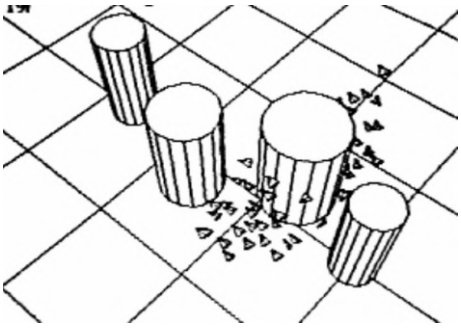


Figura 2.6: Fotograma de la simulación original de Reynolds en 1987. La bandada simulada se divide para sortear el obstáculo cilíndrico.

En él, Reynolds define tres comportamientos básicos[1] que exhibe cada uno de los individuos que forman parte de una bandada: separación, cohesión y alineamiento. La

[1]Estos comportamientos se describen en detalle en el capítulo siguiente

separación y la cohesión determinan la forma en que las aves se repelen y se atraen entre sí, respectivamente. El alineamiento, por su parte, determina cómo cada individuo tiende a seguir la dirección de movimiento promedio que exhiben sus vecinos más cercanos. Estos tres comportamientos básicos dan lugar a una conducta emergente de todo el grupo que resulta ser lo que observamos como bandada [46]. La figura 2.6 muestra un fotograma de la simulación original de 1987, en donde un grupo de aves sintéticas o boids, como los llama Reynolds, interactúa con una serie de obstáculos.

Si bien existen discrepancias entre la biología computacional y observacional, el modelo de Reynolds sigue siendo la base de los modelos desarrollados con posterioridad, los cuales buscan resolver algunas de estas discrepancias. Por ejemplo, en el artículo «*Self-organized aerial displays of thousands of starlings: a model*» [36] de Hildenbrandt y colegas, se combina el modelo original de Reynolds con un modelo aerodinámico de ala fija que describe el desplazamiento de las aves, considerando elementos como resistencia del aire, gravedad y sustentación de las alas. La figura 2.7 compara en forma esquemática ambos modelos. Aquí, los componentes del modelo de Reynolds se denominan «fuerzas sociales», las que luego se suman a las fuerzas aerodinámicas para obtener la velocidad neta del ave.

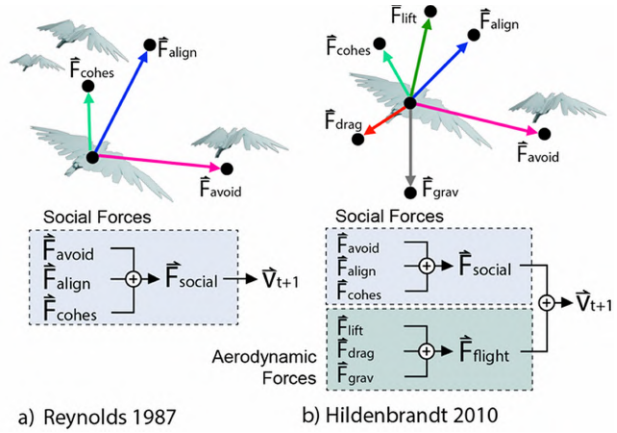



Figura 2.7: Modelos de Reynolds y Hildenbrandt. *Imagen tomada de [37].*

Por otro lado, en el artículo «*Flock2: A model for orientation-based social flocking*» [37], Hoetzlein amplía el modelo de Reynolds añadiendo el comportamiento de *bounding* o protección periférica, observado en la naturaleza, que aplica a las aves que se encuentran en la periferia de la bandada. Éstas, al quedar aisladas del resto, activan un comportamiento de protección por el cual tienden a agregarse, volando hacia el centro de masa del grupo. Esto hace que la bandada adquiera una forma más compacta y ovalada, lo cual se corresponde mejor con lo observado en aves reales.

Otra diferencia importante con respecto a los trabajos previos es que utiliza los comportamientos sociales (Reynolds + bounding) solamente para determinar una dirección de desplazamiento deseada, y no para calcular directamente la velocidad neta del ave. Esta dirección deseada, expresada como una cuaterna $Q$, es luego usada por el modelo aerodinámico, el cual se encarga de calcular la velocidad neta del ave. La figura 2.8 muestra un esquema del modelo de Hoetzlein.

Cabe mencionar que, además de estos trabajos, hay otros estudios sobre un problema no relacionado con la disuasión de aves o el modelado de bandadas, pero que involucra conceptos muy similares [69, 85]. Se trata del problema del perro pastor y el rebaño de ovejas, en donde el perro debe mantener el rebaño unido y, a la vez, dirigirlo

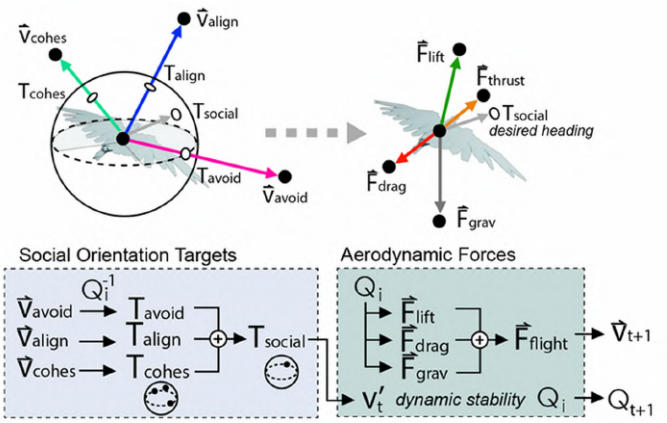



Figura 2.8: Modelo de Hoetzlein. Los componentes sociales solo se utilizan para determinar una dirección de desplazamiento deseada. Esta es luego utilizada por el modelo aerodinámico para calcular la velocidad efectiva del ave. *Imagen tomada de [37].*

hacia una posición objetivo predeterminada.

## 2.4. Rebaño de ovejas y perro pastor

El modelado del rebaño de ovejas suele basarse en alguna variante del modelo de Reynolds, mientras que el perro pastor equivale al depredador o al vehículo aéreo en el caso de la disuasión de aves. En este sentido, se destacan los siguientes dos artículos:

- «*Solving the shepherding problem: heuristics for herding autonomous, interacting agents*» [69]
- «*Learning to Herd Agents Amongst Obstacles: Training Robust Shepherding Behaviors using Deep Reinforcement Learning*» [85]

El primer trabajo resuelve el problema con una heurística sencilla, mostrada en la figura 2.9, en la que el perro pastor exhibe dos comportamientos bien definidos: conducción y agrupación.

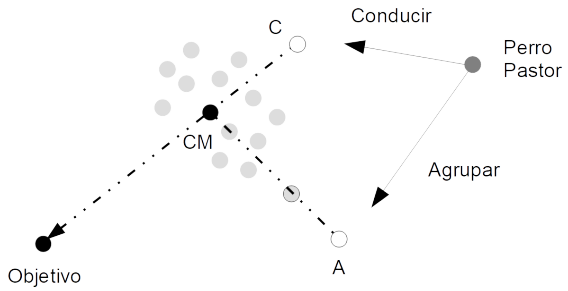



Figura 2.9: Conducción y agrupación

La conducción consiste en dirigir el perro pastor hacia el punto $C$ ubicado detrás del rebaño, en la dirección definida por el punto objetivo y el centro de masa del grupo,

punto $CM$. La idea es que cuando el perro se aproxima al punto $C$, el rebaño se aleja de él moviéndose en la dirección opuesta, es decir, hacia el objetivo.

Cuando alguno de los individuos se aleja del punto $CM$ más allá de cierto umbral predefinido, el perro pastor activa el comportamiento de agrupación, que consiste en dirigirse al punto A ubicado delante del individuo que se ha escapado, con el objetivo de llevarlo de nuevo hacia el grupo.

Estos dos comportamientos se suceden en forma alternada, dando como resultado el objetivo buscado: desplazar gradualmente el rebaño hasta el punto objetivo previamente determinado, sin dispersarlo. La figura 2.10 muestra la trayectoria generada en una simulación.

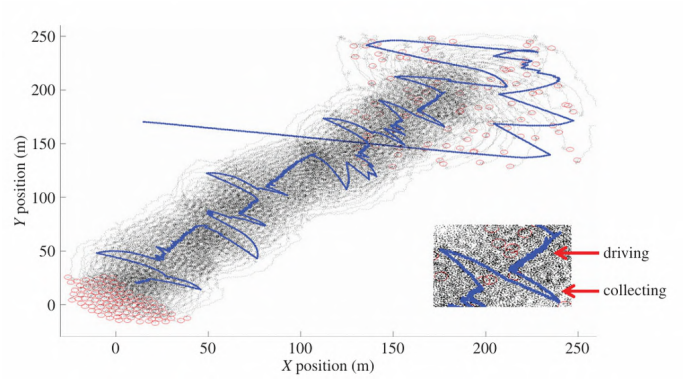



Figura 2.10: Trayectoria generada por los comportamientos de conducción (driving) y agrupación (collecting) [69]

El segundo trabajo [85] utiliza una estrategia muy diferente para resolver el mismo problema: aprendizaje por recompensas o *RL* (*Reinforcement Learning*). Aquí el problema planteado es bastante más complejo: contando con un espacio observable limitado, se pretende dirigir el rebaño a través de un área con obstáculos hasta alcanzar un punto objetivo predeterminado, como muestra la figura 2.11

La dinámica del rebaño se modela con los comportamientos de Reynolds, a los que se añaden el comportamiento de huida ante la presencia del perro pastor, así como también la evasión de obstáculos.

Por su parte, el agente (perro pastor) solo tiene visibilidad del entorno local más próximo a su alrededor, lo que define el espacio observable en cuyo centro se encuentra el agente.

El algoritmo funciona en dos etapas. Primero, mediante el uso de un *roadmap* probabilístico [41], se calcula una trayectoria deseada desde la posición actual del agente hasta el punto objetivo final. Esta trayectoria se va actualizando en cada paso a medida que el agente avanza y observa nuevos obstáculos.

A continuación, se calcula la intersección de esta trayectoria con el espacio observable, lo que determina un objetivo intermedio (*sub goal*), el cual es usado como entrada por el algoritmo de *RL*.

Valiéndose de una red neuronal convolucional, el algoritmo de *RL* procesa, en cada paso, una secuencia de imágenes del espacio observable para determinar la siguiente acción que debe realizar el agente. Cabe aclarar que el procesamiento de una secuencia de imágenes en lugar de una a la vez es necesario para brindar a la red información sobre la velocidad del agente y el rebaño.

Para la recompensa, se mide el desplazamiento que realiza el centro de la bandada y se asigna una recompensa al agente si dicho centro se acerca al objetivo intermedio, o

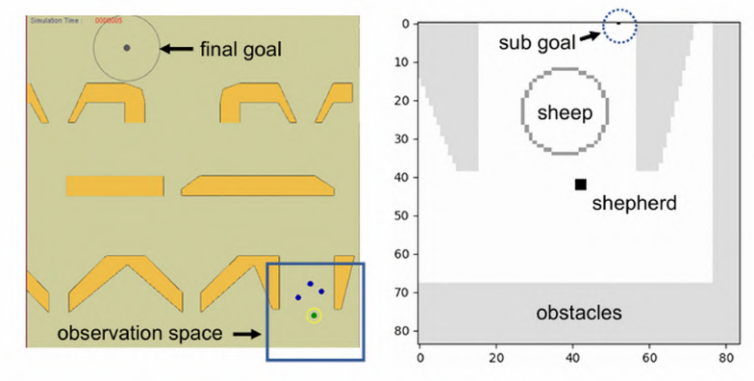



Figura 2.11: Espacio global (imagen izquierda) y espacio observable (imagen derecha) en el problema del perro pastor [85]. El perro cuenta con un espacio observable limitado por el cuadrado azul (imagen izquierda), en donde puede detectar parte de los obstáculos y parte del rebaño de ovejas. En el límite de este espacio observable, se establece un objetivo local (sub goal) hacia el cual se debe dirigir el rebaño. Este objetivo local se actualiza a medida que el perro avanza hacia el objetivo global (final goal).

una penalización en caso contrario. Adicionalmente, hay penalizaciones por colisionar con los obstáculos.

En los experimentos realizados, los autores comparan el rendimiento de este algoritmo con la heurística de conducción y agrupación, tomando como medida el tiempo requerido para alcanzar el objetivo.

El algoritmo basado en *RL* muestra ser superior a la heurística, pero solo cuando el rebaño es muy pequeño: de 4 individuos o menos. Para tamaños más grandes, el algoritmo basado en *RL* no logra resolver el problema, mientras que la heurística puede manejar grupos con cientos de individuos. Los autores sugieren, como trabajo a futuro, combinar la heurística y las técnicas de *RL*, con el objetivo de desarrollar un algoritmo híbrido y lograr así un rendimiento superior a las dos opciones por separado.

Como observación final, cabe mencionar que el relevamiento de los artículos descritos en este capítulo ha contribuido en gran medida a las ideas que se utilizaron en el presente trabajo y que se desarrollan en los siguientes capítulos. Entre esas ideas, destacamos la modelización tanto de la bandada como del área cultivada a proteger, el uso de posibles heurísticas y la opción de aplicar aprendizaje por recompensas para resolver el problema de la disuasión de aves.

Esta página ha sido intencionalmente dejada en blanco.

# Capítulo 3

# Modelado

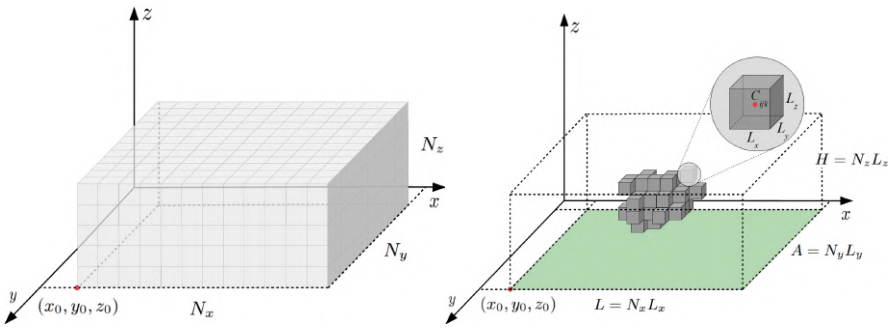
El problema a resolver involucra el modelado de tres elementos básicos:

- Espacio aéreo a proteger
- Vehículo aéreo o dron
- Bandada

Las siguientes secciones detallan cada elemento.

## 3.1. Espacio aéreo

El espacio aéreo a proteger se modela como una cuadrícula tridimensional de dimensiones $N_x$x$N_y$x$N_z$ celdas, como se muestra en la figura 3.1 (a). La base de la cuadrícula, paralela al plano $XY$, representa el área cultivada a proteger, y a partir de allí hasta una altura máxima de $N_z$ celdas. A su vez, cada celda es un prisma rectangular de dimensiones $L_x$x$L_y$x$L_z$, por lo que el espacio aéreo tiene en su base un área cultivada rectangular de largo $L = N_x L_x$ y ancho $A = N_y L_y$, y una altura $H = N_z L_z$.




(a) (b)

Figura 3.1: (a) Cuadrícula (b) Celdas ocupadas

Se asume la presencia de una red de sensores (cámaras, micrófonos, radares, etc.), tanto en tierra como a bordo del dron, que permite determinar qué celdas se encuentran

ocupadas por una o más aves en un instante dado. Dicha matriz es el producto de algoritmos de reconocimiento de aves o bandadas, y no contiene información sobre obstáculos. Esta información, que si bien es recabada por la misma red sensorial, es almacenada en otras estructuras y procesada por módulos especializados, como, por ejemplo, el algoritmo de *avoidance* presentado en el capítulo 7.

A partir de la información sensada se construye, en el instante $t$, una matriz de ocupación binaria $M_t \in [0,1]^{N_x \times N_y \times N_z}$, tal que $M_t[i,j,k] = 1$ si se detecta al menos un ave en la celda $[i,j,k]$, y 0 en caso contrario, siendo $i = 0..N_x - 1$, $j = 0..N_y - 1$ y $k = 0..N_z - 1$. La figura 3.1 (b) muestra una instancia de la matriz en donde solo se visualizan las celdas ocupadas.

Si tomamos un sistema de coordenadas cartesianas $xyz$, según la figura 3.1(a), la celda $[i,j,k]$ queda definida como el conjunto de puntos $p \in \mathbb{R}^3$ contenidos en un prisma rectangular, tal que:

$$\begin{aligned}[i,j,k] = \{p = (x,y,z) \in \mathbb{R}^3 \ / & iL_x \leq x - x_0 < (i+1)L_x, \\ & jL_y \leq y - y_0 < (j+1)L_y, \\ & kL_z \leq z - z_0 < (k+1)L_z\}\end{aligned}$$

siendo $(x_0, y_0, z_0)$ el origen de la grilla. El centro $C_{ijk} = (x_i, y_j, z_k)$ de la celda $[i,j,k]$ se construye como:

$$\begin{aligned} x_i &= (i + 0.5)L_x + x_0 \\ y_j &= (j + 0.5)L_y + y_0 \\ z_k &= (k + 0.5)L_z + z_0 \end{aligned}$$

A su vez, dado el punto $p = (x,y,z) \in \mathbb{R}^3$, el mismo pertenece a la celda $[i,j,k]$ si y solo si:

$$[i,j,k] = \left[ floor\left(\frac{x - x_0}{L_x}\right), floor\left(\frac{y - y_0}{L_y}\right), floor\left(\frac{z - z_0}{L_z}\right) \right]$$

## 3.2. Dron

Para reducir la complejidad y el costo computacional de simular el dron, especialmente al utilizar algoritmos de *Reinforcement Learning*, se decide restringir la capacidad de movimiento del vehículo a solo dos componentes: velocidad horizontal hacia adelante (positiva o cero) y velocidad vertical (positiva o negativa). Estas velocidades se expresan en un sistema de coordenadas $x'y'z'$ local al dron, como muestra la figura 3.2. El plano local $x'y'$ es siempre paralelo al plano global $xy$ y el eje local $z'$ es siempre paralelo al eje global $z$.

La velocidad frontal se expresa en el eje $x'$ con el valor $v_{x'} \geq 0$ (no puede retroceder) mientras que el desplazamiento lateral es nulo con velocidad $v_{y'} = 0$ y la velocidad vertical $v_{z'}$ es tanto positiva como negativa.

Adicionalmente, se define la orientación $\theta$ del vehículo medida con respecto al sistema de coordenadas global, siendo $\theta = 0$ la orientación que coincide con el eje $x$ global positivo.

La variación de $\theta$ se denota como la velocidad angular $w$ (*yaw*) y se descarta el modelado de la rotación alrededor de los restantes ejes (*pitch* y *roll*).

La dinámica del dron se basa en la idea planteada en [81], en donde se establecen cotas de velocidad y aceleración. En este sentido, se definen las siguientes restricciones sobre el dron:

- Velocidad frontal: $v_{x'} \in [0, V_x]$

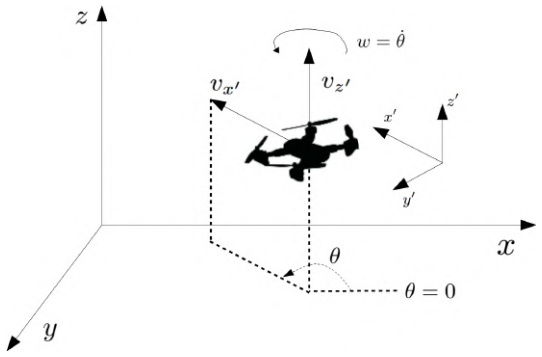



Figura 3.2: Representación del dron

- Velocidad lateral: $v_{y'} = 0$
- Velocidad vertical: $v_{z'} \in [-V_z, V_z]$
- Velocidad angular con respecto a $z'$: $w \in [-W, W]$
- Velocidad angular con respecto a $x'$: $w_x = 0$
- Velocidad angular con respecto a $y'$: $w_y = 0$
- Aceleración frontal: $\dot{v}_{x'} \in [-A_v, A_v]$
- Aceleración vertical: $\dot{v}_{z'} \in [-A_z, A_z]$
- Aceleración angular: $\dot{w} \in [-A_w, A_w]$

Es importante aclarar que este es un dron ideal en donde no se modelan fuerzas como gravedad, resistencia del aire o sustentación de los rotores. Esto significa que, cuando se envíe un comando de velocidad, el dron se moverá exactamente a la velocidad indicada, siempre y cuando no se viole ninguna de las restricciones previamente detalladas.

Puede existir, por tanto, una diferencia significativa entre la respuesta del dron modelado y la respuesta de un dron real. Esta diferencia también se observará al hacer la comparación con modelos más realistas utilizados en entornos de simulación como *Gazebo*[1]. En el capítulo 7 se aborda esta cuestión cuando se apliquen los algoritmos desarrollados a un dron modelado en *Gazebo* y controlado a través de ROS2.

### 3.2.1. Cálculo de posición

Si en el instante de tiempo $t$, la posición del dron es $x(t), y(t), z(t)$, entonces el estado completo del dron queda determinado por la tupla:

$$s_d(t) = [x, y, z, \theta, v_{x'}, v_{z'}, w]_{|t=t}$$

Para obtener la posición del dron en el siguiente instante de tiempo $t + \Delta t$, primero se calcula $x(t + \Delta t)$ e $y(t + \Delta t)$ aplicando una rotación paralela al plano $xy$ determinada por la velocidad angular $w$ y la velocidad frontal $v_{z'}$, para luego, mediante una traslación, obtener $z(t + \Delta t)$. A continuación se detallan los pasos.

En primer lugar, se calcula un radio de giro $r_t$ alrededor de un punto imaginario $C_t = (c_x, c_y)_{|t=t}$, como muestra la figura 3.3, en donde:

[1] Ver apéndice E

$$r_t = \frac{v_{x'}(t)}{w(t)}$$

$$\theta(t + \Delta t) = \theta(t) + w(t)\Delta t$$
$$c_x(t) = x(t) + r_t sen[\theta(t + \Delta t)]$$
$$c_y(t) = y(t) - r_t cos[\theta(t + \Delta t)]$$

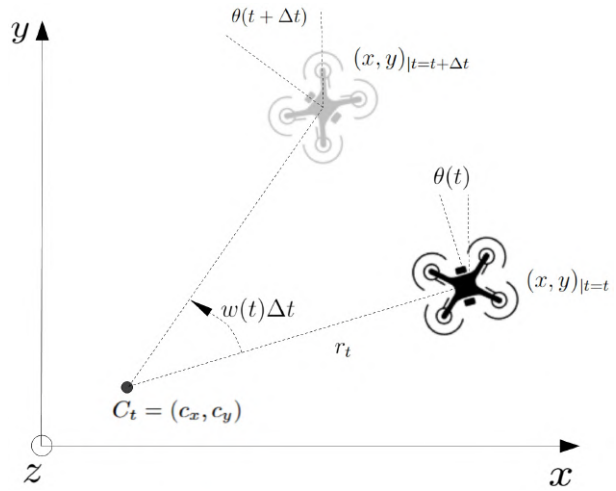



Figura 3.3: Cálculo de posición del dron en el instante $t + \Delta t$

A continuación, se aplica una rotación paralela al plano $xy$, con centro $C(t)$ y ángulo $w(t)\Delta t$:

$$x(t + \Delta t) = c_x(t) + [x(t) - c_x(t)]cos(w\Delta t) - [y(t) - c_y(t)]sen(w\Delta t)$$
$$y(t + \Delta t) = c_y(t) + [x(t) - c_x(t)]sen(w\Delta t) + [y(t) - c_y(t)]cos(w\Delta t)$$

Para evitar problemas numéricos, en la implementación se verifica si $w(t) \approx 0$, en cuyo caso se considera un desplazamiento rectilíneo con velocidad $v_{x'}(t)$.

Finalmente, la componente vertical de la posición en el instante $t + \Delta t$ se obtiene como:

$$z(t + \Delta t) = z(t) + v_{z'}(t)\Delta t$$

### 3.2.2. Cambios de velocidad

Cualquier cambio en la velocidad lineal $v_{x'}$ que se imponga en el instante $t + \Delta t$, estará sujeto a las restricciones de aceleración previamente estipuladas:

$$-A_v \leq \frac{v(t + \Delta t) - v_{x'}(t)}{\Delta t} \leq A_v$$

Con lo cual, el nuevo valor de velocidad $v(t + \Delta t)$ se truncará dentro del intervalo:

$$v_{x'}(t) - A_v\Delta t \leq v(t+\Delta t) \leq v_{x'}(t) + A_v\Delta t$$

Incorporando las cotas de velocidad mínima y máxima, el intervalo resulta ser:

$$\text{máx}(0, v_{x'}(t) - A_v\Delta t) \leq v(t+\Delta t) \leq \text{mín}(V_x, v_{x'}(t) + A_v\Delta t)$$

Aplicando restricciones análogas a las velocidades $v_{z'}$ y $w$ se obtiene que:

$$\text{máx}(0, v_{z'}(t) - A_z\Delta t) \leq v_{z'}(t+\Delta t) \leq \text{mín}(V_z, v_{z'}(t) + A_z\Delta t)$$
$$\text{máx}(0, w(t) - A_w\Delta t) \leq w(t+\Delta t) \leq \text{mín}(W, w(t) + A_w\Delta t)$$

## 3.3. Bandada

La simulación de la bandada se basa en el modelo de Reynolds [58], introducido en el capítulo 2, el cual es ampliamente utilizado por la comunidad académica. El elemento básico del modelo es el boid, o ave virtual, que exhibe una serie de comportamientos reactivos simples, en respuesta a la presencia de otros boids o de un depredador, representado en este caso por el vehículo aéreo.

La interacción de estos comportamientos da como resultado una conducta emergente que pretende asemejarse a una bandada real de aves. El modelo de boids utilizado en este trabajo se basa en una variante del modelo de Reynolds presentada en [76], la cual simplifica los cálculos del modelo original.

Los boids se definen como un punto (o partícula) en el espacio. Cada uno está sometido a una serie de fuerzas que determinan una dirección de movimiento neta. Una vez establecida esa dirección, el boid se desplaza a una velocidad predeterminada.

Formalmente cada boid se representa, en el instante $t$, como el vector $\vec{b}_i(t) \in \mathcal{R}^3$, con $i = 1, 2, .., N$. En cada instante de la simulación se calcula un vector de dirección $\vec{H}_i(t)$ que indica hacia dónde debe desplazarse el i-ésimo boid. La nueva posición del boid en el instante de tiempo $t + \Delta t$ es entonces:

$$\vec{b}_i(t+\Delta t) = \vec{b}_i(t) + v_b\Delta t\vec{H}_i(t)$$

siendo $v_b$ la velocidad de desplazamiento del boid. Se definen dos valores posibles: una velocidad de huida $v_h$ cuando el boid detecta la presencia de un depredador, y una velocidad normal $v_n$, cuando el boid se encuentra buscando alimento. Se asume $v_h \gg v_n$.

El cálculo de $\vec{H}_i(t)$ se realiza en base a 7 comportamientos básicos:

- Movimiento aleatorio
- Separación
- Cohesión
- Alineamiento
- Atracción a fuente de comida
- Protección periférica (bounding)
- Evasión del depredador

### 3.3.1. Movimiento aleatorio

Ante la ausencia de depredadores, el boid permanece estacionario y posado en un árbol alimentándose. Sin embargo, mientras realiza esta actividad, el boid puede efectuar pequeños desplazamientos de un árbol a otro.

Dado el i-ésimo boid en el instante $t$, se modela este desplazamiento como un vector aleatorio $\vec{e}_i(t) \sim \mathcal{N}(\vec{0}, \vec{\sigma}^2)$, siendo $\vec{\sigma}^2 = (\sigma_x^2, \sigma_y^2, \sigma_z^2)$ y $\vec{0}$ el vector nulo de $\mathcal{R}^3$.

### 3.3.2. Separación

La separación está determinada por la fuerza de repulsión ejercida sobre un boid por parte de sus boids vecinos, dentro de un radio de influencia predeterminado. Dado el boid $\vec{b}_i(t)$, se considera el conjunto de boids vecinos $N_{r_b}(i,t)$ que se encuentran a una distancia máxima $r_b$ del mismo:

$$N_{r_b}(i,t) = \{\vec{b}_{n_i(j)}(t),\ \|\vec{b}_i(t) - \vec{b}_{n_i(j)}(t)\| \leq r_b,\ j = 1,2,..,N_i\}$$

El mapeo $n_i(j)$ devuelve el j-ésimo vecino del i-ésimo boid, siendo $N_i$ el número total de vecinos del i-ésimo boid [2]. De este modo, el vector de separación $\vec{S}_i(t)$ se define como:

$$\vec{S}_i(t) = \sum_{j=1}^{j=N_i} \frac{\vec{b}_i(t) - \vec{b}_{n_i(j)}(t)}{\|\vec{b}_i(t) - \vec{b}_{n_i(j)}(t)\|}$$

$\vec{S}_i(t)$ es la suma normalizada de los vectores diferencia entre el i-ésimo boid y sus vecinos, dentro del radio $r_b$. Este comportamiento previene que los boids colapsen unos sobre otros. Ver figura 3.4 (a).

### 3.3.3. Cohesión

En forma similar, la cohesión está determinada por la fuerza de atracción $\vec{C}_i(t)$ de un boid hacia el centro de masa local de sus vecinos ($L\vec{C}M_i$), dentro del radio de acción $r_b$:

$$L\vec{C}M_i(t) = \frac{1}{N_i} \sum_{j=1}^{j=N_i} \vec{b}_{n_i(j)}(t)$$

$$\vec{C}_i(t) = \frac{L\vec{C}M_i(t) - \vec{b}_i(t)}{\|L\vec{C}M_i(t) - \vec{b}_i(t)\|}$$

Como su nombre lo sugiere, la cohesión mantiene localmente unidos a los boids. Al igual que en el caso de la separación, el vector $\vec{C}_i(t)$ está normalizado. Ver figura 3.4 (b).

### 3.3.4. Alineamiento

El alineamiento hace que un boid tienda a moverse en la dirección promedio en que se mueven sus vecinos. Si se toma el vector de dirección de cada vecino del i-ésimo boid en el instante previo $t - \Delta t$, es decir $\vec{H}_{n_i(j)}(t - \Delta t)$, entonces el vector de alineamiento de dicho boid queda definido como:

$$\vec{A}_i(t) = \frac{1}{N_i} \left[ \sum_{j=1}^{j=N_i} \vec{H}_{n_i(j)}(t - \Delta t) \right]$$

Ver figura 3.4 (c).

[2] Por simplicidad en la notación se omite la dependencia con $t$ y $r_b$

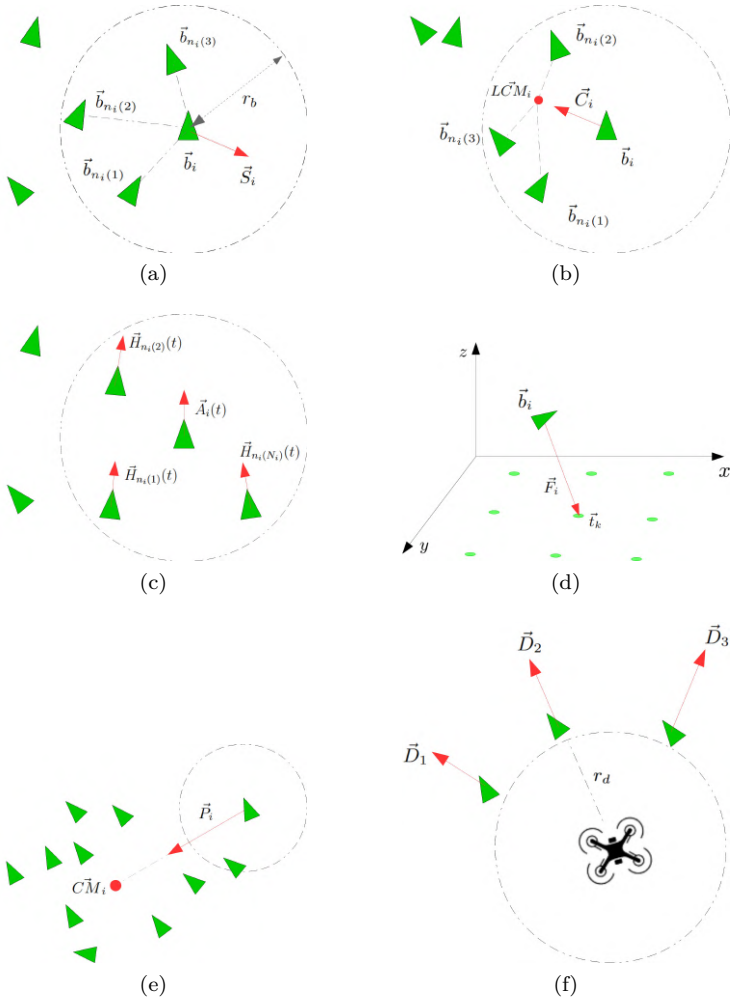



Figura 3.4: (a) Separación (b) Cohesión (c) Alineamiento (d) Atracción a la árboles (e) Bounding o protección periférica (f) Evasión

### 3.3.5. Atracción a la fuente de comida

Los tres comportamientos precedentes forman parte del modelo de Reynolds original. A dicho modelo se le añade la atracción a las fuentes de comida, las que se definen como una serie de puntos $\vec{t}_k$ fijos ubicados en la base del espacio aéreo a proteger, y que pretenden representar árboles distribuidos sobre el campo. Los boids se sienten atraídos al árbol más cercano, lo que se representa con el vector $\vec{F}_i(t)$ definido como:

$$\vec{F}_i(t) = \frac{ArgMin_{\vec{t}_k}(\|\vec{t}_k - \vec{b}_i(t)\|) - \vec{b}_i(t)}{\|ArgMin_{\vec{t}_k}(\|\vec{t}_k - \vec{b}_i(t)\|) - \vec{b}_i(t)\|}$$

El término $ArgMin_{\vec{t}_k}(\|\vec{t}_k - \vec{b}_i(t)\|)$ devuelve el árbol $\vec{t}_k$ más cercano al i-esimo boid, en el instante $t$. Ver figura 3.4(d).

### 3.3.6. Protección periférica

La protección periférica o *bounding*, es un comportamiento que se ha observado en la naturaleza [37] y que se activa cuando los individuos se encuentran en la periferia de la bandada. A medida que éstos comienzan a quedarse aislados del resto, se incrementa el riesgo de ser presa de los depredadores. En respuesta a esta situación, los individuos tienden a moverse hacia el centro de masa de todo el grupo como medida de protección.

Para modelar esta reacción, se establece un umbral $B$ que representa el número mínimo de boids vecinos que debe tener un boid dado para no activar este comportamiento, es decir, $N_i \geq B$. Una vez activado, se calcula el vector $\vec{P}_i(t)$ que apunta hacia el centro de masa $C\vec{M}(t)$ de la bandada:

$$\vec{P}_i(t) = \mathbf{1}_{(B>N_i)} \left(\frac{B - N_i}{B}\right) \left[\frac{C\vec{M}(t) - \vec{b}_i(t)}{\|C\vec{M}(t) - \vec{b}_i(t)\|}\right]$$

El término $(B - N_i)/B$ hace que la magnitud del vector se incremente a medida que $N_i$ disminuye, es decir, a medida que el boid va quedando más aislado. A su vez, el término indicatriz asegura que el vector sea nulo cuando $B \leq N_i$. Ver figura 3.4 (e).

### 3.3.7. Evasión del depredador

Finalmente se añade el comportamiento de huida del boid ante la presencia de un depredador, que en este caso esta representado por el dron. Se define el vector $\vec{D}_i(t)$ como la diferencia entre la posición del boid y el dron $\vec{p}_d(t) = [x(t), y(t), z(t)]^T$:

$$\vec{D}_i(t) = \frac{\vec{b}_i(t) - \vec{p}_d(t)}{\|\vec{b}_i(t) - \vec{p}_d(t)\|} \mathbf{1}_{\|\vec{b}_i(t) - \vec{p}_d(t)\| < r_d}$$

El comportamiento sólo se activa si el dron se encuentra a una distancia del boid menor al umbral $r_d$. Como muestra la figura 3.4 (f), los boids escapan en forma radial ante la presencia del depredador.

### 3.3.8. Desplazamiento neto

La dirección de desplazamiento resultante en el instante $t + \Delta t$ se calcula como la suma ponderada de todos los componentes anteriores, más un término de inercia representado por $\vec{H}_i(t)$:

$$\vec{H}_i(t + \Delta t) = w_h \vec{H}_i(t) + w_s \vec{S}_i(t) + w_c \vec{C}_i(t) + w_a \vec{A}_i(t) + w_f \vec{F}_i(t) + w_p \vec{P}_i(t) + w_d \vec{D}_i(t) + w_e \vec{e}_i(t)$$

$$\vec{H}_i(t+\Delta t) = w_h \vec{H}_i(t) + w_s \vec{S}_i(t) + w_c \vec{C}_i(t) + w_a \vec{A}_i(t) + w_f \vec{F}_i(t) + w_p \vec{P}_i(t) + w_d \vec{D}_i(t) + w_e \vec{e}_i(t)$$

Si se construye la matriz $R_{i,t} \in \mathcal{R}^{3\text{x}8}$ concatenando los vectores de comportamiento, tal que $R_{i,t} = [\vec{H}_i \ \vec{S}_i \ \vec{C}_i \ \vec{A}_i \ \vec{F}_i \ \vec{P}_i \ \vec{D}_i \ \vec{e}_i](t)$, y el vector de pesos $\vec{w} = [w_h, w_s, w_c, w_a, w_f, w_p, w_d, w_e]^T$, se puede entonces expresar $\vec{H}_i(t + \Delta t)$ en forma matricial como:

$$\vec{H}_i(t + \Delta t) = R_{i,t} \vec{w}$$

Se asume $w_p > w_f$, es decir, un boid dará prioridad a agregarse con el resto del grupo en busca de protección, frente a la opción de posarse en el árbol más cercano para alimentarse, especialmente si detecta la presencia de un depredador.

# Capítulo 4

# Solución aplicando heurísitca

Una primera aproximación a la solución del problema de la disuasión de aves es el uso de una heurística sencilla. Una vez detectada la presencia de aves en el campo, se activa el dron para que ejecute una serie de maniobras con el fin de expulsarlas del área protegida.

Según el modelo de boids presentado previamente, existe una solución trivial para el caso en que la bandada se componga de 4 individuos o menos: perseguir el centro de masa del grupo hasta que este acabe por salirse completamente del campo, como muestra la figura 4.1. Esto funciona debido a que la bandada es suficientemente pequeña y, al empujarla, se mantiene unida sin que los boids se dispersen.

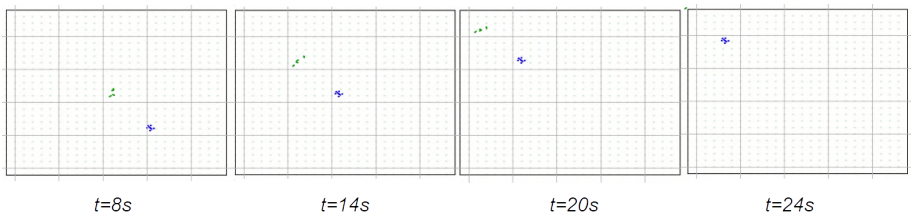



Figura 4.1: El dron (figura azul) expulsa 4 boids (puntos verdes) dirigiéndose al centro de masa de la bandada.

El resultado cambia radicalmente cuando la bandada está compuesta de 5 o más boids. Si se dirige el dron hacia el centro, la bandada se dispersa y tiende a formar múltiples bandadas más pequeñas, con lo que el objetivo inicial de expulsar a todos los boids se vuelve mucho más difícil de alcanzar. La figura 4.2 ilustra lo que sucede con una bandada de 20 boids.

La solución en este caso consiste en combinar dos o más comportamientos simples que logren mover la bandada hacia los bordes del campo, pero que al mismo tiempo la mantengan unida formando un único grupo de boids.

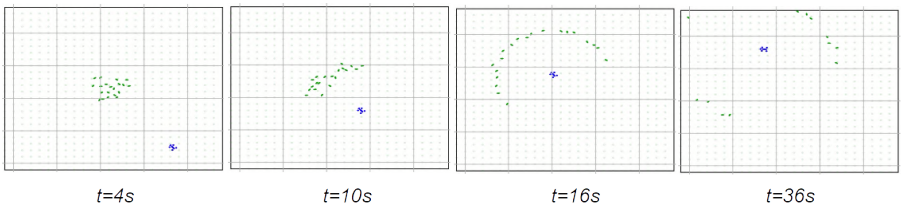



Figura 4.2: Con 20 boids, el dron no logra expulsar la bandada, que se dispersa y forma bandadas mas pequeñas

## 4.1. Persecución de flancos

La heurística denominada «Persecución de Flancos» se inspira en la heurística de conducción y agrupación [69], utilizada por el perro pastor para desplazar un rebaño de ovejas. Esta heurística se presentó en el capítulo 2 como parte de los antecedentes.

Se destacan diferencias importantes entre ambas estrategias. En el caso de los boids, no es necesario dirigir la bandada a un punto predeterminado, sino desplazarla fuera del área protegida en cualquier dirección. Otra diferencia es que los boids se modelan en tres dimensiones, mientras que el rebaño de ovejas se mueve en el plano.

La heurística «persecución de flancos» se basa en la matriz de ocupación $M_t$ presentada en el capítulo anterior, en donde $M_t[i, j, k] = 1$ sí y solo si se detecta la presencia de al menos un ave en la celda $[i, j, k]$.

La figura 4.3 muestra una instancia de la matriz de ocupación $M_t$, siendo el punto $D$ la posición actual del dron. A continuación, se proyectan las celdas ocupadas sobre el plano $xy$, así como también el punto $D$, obteniendo el punto $D'$.

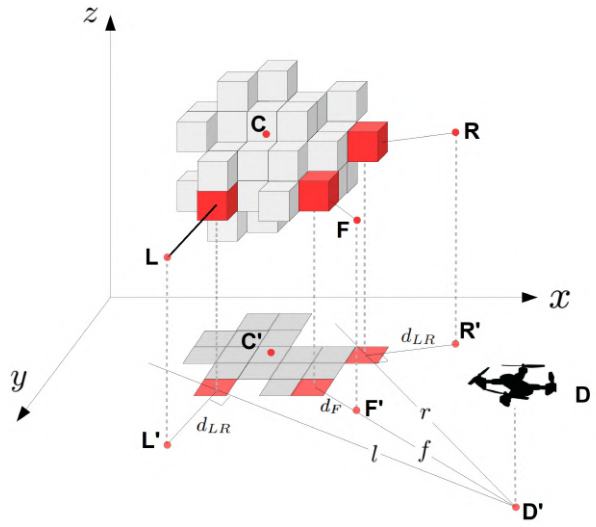



Figura 4.3: Determinación de los flancos $L$, $R$ y $F$, mediante la proyección de la matriz de ocupación sobre el plano $xy$, y el cálculo de las rectas tangentes $l, r$ y la recta frontal $f$. El punto $C$ es el centro geométrico de las las celdas ocupadas.

A partir del grupo de celdas proyectadas, se toman las tangentes a las mismas que pasan por el punto $D'$, obteniendo las rectas $l$ y $r$ (ambas contenidas en el plano $xy$).

Estas rectas pasan por el centro de las celdas tangentes, destacadas en rojo. Por otro lado, tomamos la celda proyectada más cercana al punto $D'$ y se traza la recta $f$ que pasa por el centro de dicha celda y el punto $D'$.

Luego, partiendo de las dos celdas tangentes, a la izquierda y derecha del dron, se trazan dos rectas perpendiculares a $l$ y $r$ respectivamente. Sobre estas perpendiculares, y alejándose del grupo de celdas proyectadas, se toman los puntos $L'$ y $R'$ colocados a una distancia fija $d_{LR}$. De modo similar, se toma el punto $F'$ sobre la recta $f$, siempre alejándose del grupo de celdas proyectadas, a una distancia fija $d_F$ de la celda frontal.

Luego se construyen los puntos $L$, $R$ y $F$, elevando los puntos $L'$, $R'$ y $F'$ a la altura de la celda más baja (destacada en rojo) que se proyecta sobre la celda tangente calculada previamente. A los puntos $L$, $R$ y $F$ así obtenidos, se les denomina flanco izquierdo, derecho y frontal, respectivamente.

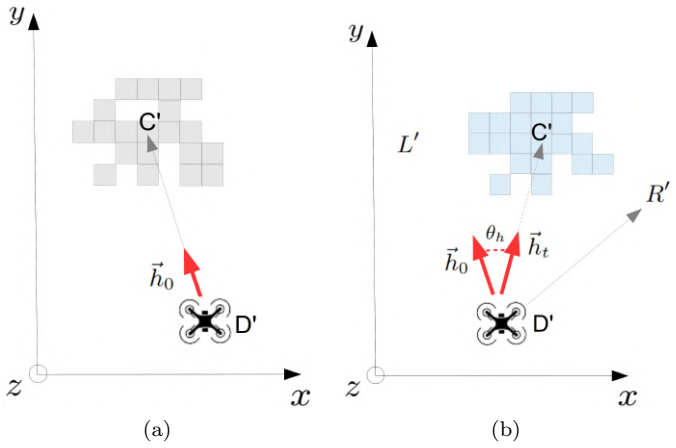



Figura 4.4: (a) Dirección guía inicial $\vec{h}_0$ (b) Ángulo $\theta_h$ entre $\vec{h}_0$ y $\vec{h}_t$. Se toma como convención el sentido horario para los valores positivos, desde $\vec{h}_0$ hacia $\vec{h}_t$.

Por otro lado, se define el punto $C$ como el centro geométrico de los centros de las celdas ocupadas; esto es:

$$C = \frac{1}{N_{occupied}} \sum_{i=0}^{N_x-1} \sum_{j=0}^{N_y-1} \sum_{k=0}^{N_z-1} (x_i, y_j, z_k) M_t(i,j,k)$$

$$N_{occupied} = \sum_{i=0}^{N_x-1} \sum_{j=0}^{N_y-1} \sum_{k=0}^{N_z-1} M_t(i,j,k)$$

En el instante inicial $t = 0$, cuando se detecta la presencia de una bandada por primera vez, se establece un vector bidimensional $\vec{h}_0 = C' - D'$, como muestra la figura 4.4, siendo $C'$ la proyección del punto $C$ sobre el plano $xy$. Este vector se utiliza como dirección guía por la cual se intenta desplazar a la bandada hacia el exterior del campo cultivado. En todo instante $t > 0$, se vuelve a calcular esta dirección en base a las coordenadas de los puntos $C'$ y $D'$ en el instante $t$, obteniendo el vector $\vec{h}_t$. En dicho instante, se observa el ángulo $\theta_h(t)$ formado por los vectores $\vec{h}_0$ y $\vec{h}_t$, es decir:

$$\theta_h(t) = cos^{-1}\left[\frac{<\vec{h}_t, \vec{h}_0>}{\|\vec{h}_t\|\|\vec{h}_0\|}\right] \tag{4.1}$$

Se toma como convención que los valores positivos de $\theta_h(t)$ están determinados por el sentido horario que va desde $\vec{h}_0$ a $\vec{h}_t$.

Todos los elementos mencionados previamente permiten definir la heurística propuesta que consiste en dirigir el dron hacia alguno de los flancos $L, R, F$ o $C$, según el siguiente criterio: Por defecto, el dron se dirige al centro geométrico de las celdas ocupadas, punto $C$, con el objetivo de desplazar la bandada hacia el exterior del espacio aéreo. Si la distancia entre los puntos $L$ y $R$ es mayor a un cierto umbral predeterminado $u_{LR}$, entonces el dron se dirige hacia el flanco izquierdo $L$ si $\theta_h(t) < 0$, o hacia el flanco derecho $R$ si $\theta_h(t) \geq 0$.

Esto significa que la elección del flanco izquierdo o derecho se basa en el error de orientación del dron con respecto al vector $\vec{h}_0$, establecido en el instante inicial. El comportamiento buscado con esta estrategia es lograr desplazar la bandada en una dirección preestablecida, para evitar que el dron persiga a las aves dentro del campo sin lograr expulsarlas.

Cabe mencionar que el uso de esta dirección guía, establecida en el instante inicial, no pretende ser la ruta de desplazamiento óptima de la bandada en términos de ahorro de energía por parte del dron. El propósito de $\vec{h}_0$ es evitar que el dron ataque el mismo flanco durante mucho tiempo, lo que puede dar como resultado que la bandada se desplace en círculos sin salir del área protegida. El diagrama de la figura 4.5 ilustra dicha situación.

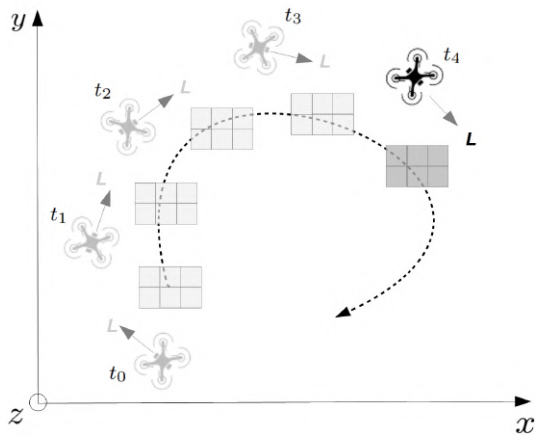



Figura 4.5: El dron elige el flanco izquierdo $L$ en los instantes $t_0, t_1, t_2, t_3$ y $t_4$. Como resultado la bandada se mueve en círculos sin ser expulsada.

Este desplazamiento circular de la bandada puede ocurrir si, en lugar de usar una dirección guía, se elige, por ejemplo, atacar el flanco que se encuentre más alejado del centro geométrico de la bandada, como se hace en la heurística del perro pastor [69]. En dicha estrategia, el desplazamiento del rebaño se hace en la dirección dada por el centro del grupo y un punto objetivo fijo. De esta forma, el comportamiento de conducción dirige al perro hacia un punto que siempre se encuentra detrás del rebaño, en relación al objetivo predeterminado, lo cual asegura un desplazamiento en la dirección correcta. En el caso de los boids, cualquier punto fuera del espacio aéreo es un objetivo potencial, por lo cual no existe una dirección de desplazamiento preestablecida que asegure la expulsión de la bandada. Por esta razón, es necesario establecer alguna dirección guía en el instante $t = 0$; de lo contrario, se estaría desplazando la bandada hacia un

objetivo móvil sin garantía de éxito.

Finalmente, para completar la heurística, se observa la distancia entre $D'$ (posición del dron en el plano $xy$) y $F'$. Si la misma es menor a un umbral predefinido $u_F$, entonces el dron se dirige hacia el flanco frontal $F$ con el fin de evitar que la bandada pase por encima o por debajo de él.

Esta última acción toma precedencia sobre las anteriores porque se considera un fallo grave que el dron quede por delante de la bandada. El algoritmo 1 muestra el seudocódigo de la heurística «Persecución de Flancos».

**Algorithm 1:** Persecución de Flancos

**Data:** $E$, $T_{max}$, $u_{LR}$, $u_F$
**Result:** $E$ actualizado

$E.reset()$
// Mientras haya boids dentro del espacio aéreo y $t < T_{max}$
**while** $E.b_{in} > 0$ *and* $E.t < T_{max}$ **do**
  $D \leftarrow E.s_D.\vec{P}_t$ ; // Posición del dron
  $L, C, R, F \leftarrow H.flanks(E)$ ; // Flancos
  $D' \leftarrow Pr_{xy}(D)$
  $F' \leftarrow Pr_{xy}(F)$
  $\vec{h}_t \leftarrow \frac{C-D}{\|C-D\|}$ ; // Dirección actual
  ; // Seleccionar el flanco objetivo:
  **if** $\|D' - F'\| < u_F$ **then**
    $T = F$ ; // Objetivo = Flanco frontal
  **else**
    **if** $\|L - R\| \leq u_{LR}$ **then**
      $T = C$ ; // Objetivo = Centro de la bandada
    **else**
      $\theta_h = cos^{-1}\left[\frac{<\vec{h}_t, \vec{h}_0>}{\|\vec{h}_t\|\|\vec{h}_0\|}\right]$ ; // Angulo entre $h_0$ y $h_t$
      **if** $\theta_h > 0$ **then**
        $T = R$ ; // Objetivo = Flanco Derecho
      **else**
        $T = L$ ; // Objetivo = Flanco Izquierdo
      **end**
    **end**
  **end**
  $v_{x'}(t), v_{z'}(t), w(t) \leftarrow H_{vzw}(D, T)$ ; // Calcular velocidades
  $E.step(v_{x'}(t), v_{z'}(t), w(t))$ ; // Ejecutar un paso de la simulación
  $E.t \leftarrow E.t + \Delta t$
**end**

La variable $E$ es una clase que encapsula el entorno de simulación de los boids y el dron. Contiene los siguientes datos y métodos:

- $E.t$: Almacena el instante de tiempo actual de la simulación.
- $E.b_{in}$: Contiene la cantidad de boids presentes actualmente dentro de los límites del espacio aéreo.
- $E.s_D$: Clase anidada que contiene todo el estado del dron:

  - $\vec{P}$: Posición en el sistema de coordenadas global.
  - $\theta$: Angulo de orientación alrededor del eje $z'$ (yaw).
  - $v_{x'}$: Velocidad frontal en el eje $x'$, local al dron.
  - $v_{z'}$: Velocidad vertical en el eje $z'$, local al dron.
  - $w$: Velocidad angular alrededor del eje $z'$, local al dron.
- $E.M_t$: Matriz de ocupación en el instante $t$.
- $E.reset()$: Función que reinicia el entorno, colocando la bandada y el dron en sus posiciones iniciales.
- $E.step(v_{x'}(t), v_{z'}(t), w(t))$: Función que ejecuta un paso de la simulación, moviendo el dron con velocidades $v_{x'}(t)$, $v_{z'}(t)$ y $w(t)$.

La clase $H$ implementa funciones auxiliares de la heurística. Por ejemplo, $H.flanks(E)$ recibe el entorno de simulación, y a partir de la matriz de ocupación y la posición del dron, calcula los flancos $L, R, F$, así como también el centro geométrico de la bandada $C$.

Por su parte, la función $H_{vzw}(D, T)$ implementa una heurística que calcula las velocidades $v_{x'}(t)$, $v_{z'}(t)$ y $w(t)$ que se deben aplicar al dron, para que este se desplace desde su posición actual, punto $D$, hasta el punto objetivo $T$.

### 4.1.1. Función $H_{vzw}$

Cabe recordar que, por razones de seguridad, al dron no se le permite moverse lateralmente ni en reversa, sino solo hacia adelante y con velocidad angular alrededor del eje $z'$. Por tanto, para dirigirse hacia un punto dado en el espacio, el dron deberá orientarse en la dirección de dicho punto y avanzar hacia él.

Para lograrlo, una posibilidad es girar en el lugar hasta quedar perfectamente alineado con el punto objetivo, y luego aplicar velocidades lineales máximas hasta alcanzar las coordenadas deseadas. Sin embargo, esta opción no resulta eficiente porque implica detenerse en pleno vuelo cada vez que cambia el objetivo a perseguir.

Por tanto, para implementar la función $H_{vzw}$, se decidió asignar las velocidades del dron en función del error de orientación $\beta(t)$ y el ángulo de elevación $\alpha(t)$ con respecto al punto objetivo $T$, según las siguientes reglas:

$$v_{x'}(t) = V_x cos[\alpha(t)] \left[1 - \frac{|\beta(t)|}{\pi}\right]$$

$$v_{z'}(t) = V_z sen[\alpha(t)] \left[1 - \frac{|\beta(t)|}{\pi}\right]$$

$$w(t) = W \left[\frac{\beta(t)}{\pi}\right]$$

Siendo $V_x$ y $V_z$ las velocidades lineales máximas, $\alpha(t)$ el ángulo $\widehat{TDT'}$ y $\beta(t)$ el ángulo $\widehat{D''D'T''}$, como muestra la figura 4.6.

La idea es que cuando el dron está de *espaldas* al objetivo, es decir, cuando el error de orientación es máximo ($\beta(t) = \pm\pi$) las velocidades lineales $v_{x'}(t)$ y $v_{z'}(t)$ se hacen cero, mientras que la velocidad angular $w(t)$ es máxima. A medida que el dron gira y el error de orientación se reduce, $v_{x'}(t)$ y $v_{z'}(t)$ se incrementan, mientras que $w(t)$ disminuye, hasta que el error es cero y el dron está perfectamente alineado con el objetivo. En este caso, el dron se desplaza en línea recta con velocidad $\vec{v}_{net}(t) = \vec{v}_{x'}(t) + \vec{v}_{z'}(t)$, tal que:

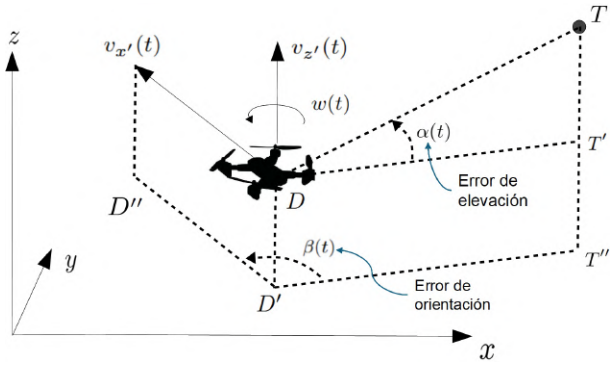



Figura 4.6: $v_{x'}(t)$, $v_{z'}(t)$ y $w(t)$ en función de $\alpha(t) = \widehat{TDT'}$ y $\beta(t) = \widehat{D''D'T''}$

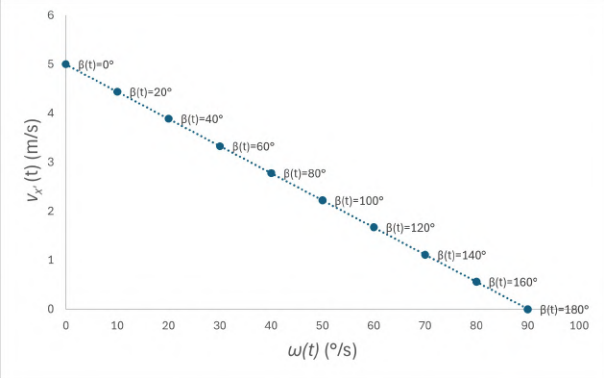



Figura 4.7: $v_{x'}(t)$ vs $w(t)$ para diferentes valores de $\beta(t)$, tomando $\alpha(t) = 0$, $V_x = 5m/s$ y $W = 90°/s$

$$\|\vec{v}_{net}(t)\| = \sqrt{V_x^2 cos^2[\alpha(t)] + V_z^2 sen^2[\alpha(t)]}$$

La figura 4.7 muestra las velocidades $v_{x'}(t)$ y $w(t)$ obtenidas por $H_{vzw}$ para diferentes valores de $\beta(t)$, tomando como ángulo de elevación $\alpha(t) = 0$, velocidad frontal máxima $V_x = 5m/s$ y velocidad angular máxima $W = 90$ °/s. Por su parte, la figura 4.8 muestra 14 trayectorias generadas por $H_{vzw}$, cuando el dron parte con un error de orientación inicial $\beta(0) = \frac{i}{10}360°$, siendo $i = 0, 1, .., 13$.

Una ejecución del algoritmo 1 con 25 boids puede verse en un vídeo de YouTube a través de este **link**. Allí se muestran tres vistas del espacio aéreo en forma simultánea: superior, trasera y lateral, que permiten apreciar mejor la trayectoria que describe el dron. Este parte de la esquina inferior derecha (según la vista superior) y se dirige hacia la bandada colocada inicialmente en el centro del campo cultivado. Se muestra, además, un vector (línea verde) que apunta hacia el flanco seleccionado por la heurística en cada momento. La línea roja describe la trayectoria realizada por el dron.

La figura 4.9 muestra una secuencia de fotogramas del video mencionado, en donde las subfiguras (a), (b), (c) y (d) muestran los instantes $t = 0s, 26s, 59s$ y $94s$, respectivamente. Se puede apreciar la oscilación que describe el dron al perseguir el flanco elegido por la heurística en cada momento. La bandada, inicialmente posada sobre los árboles, comienza a dispersarse hacia los lados a la vez que se mueve en dirección

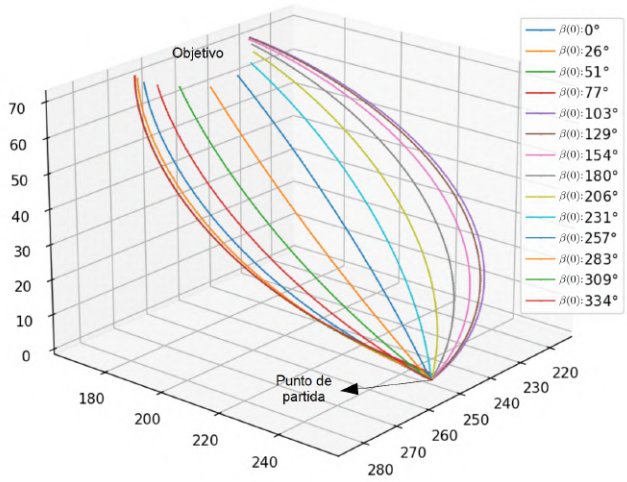



Figura 4.8: 14 trayectorias para diferentes valores de orientación inicial $\beta(0)$

opuesta al dron.

Como se mencionó en el capítulo 3, el dron modelado es ideal, solo sujeto a cotas máximas de velocidad y aceleración. Esto significa que las velocidades $v_{x'}, v_{z'}$ y $w$ calculadas por la heurística $H_{vzw}$ son las velocidades con las que efectivamente se mueve el dron, siempre que se respeten las cotas máximas. Esto no ocurrirá en la realidad, o incluso en una simulación en Gazebo, en donde existen otras restricciones no modeladas aquí, como la gravedad, resistencia del aire o la potencia de los rotores.

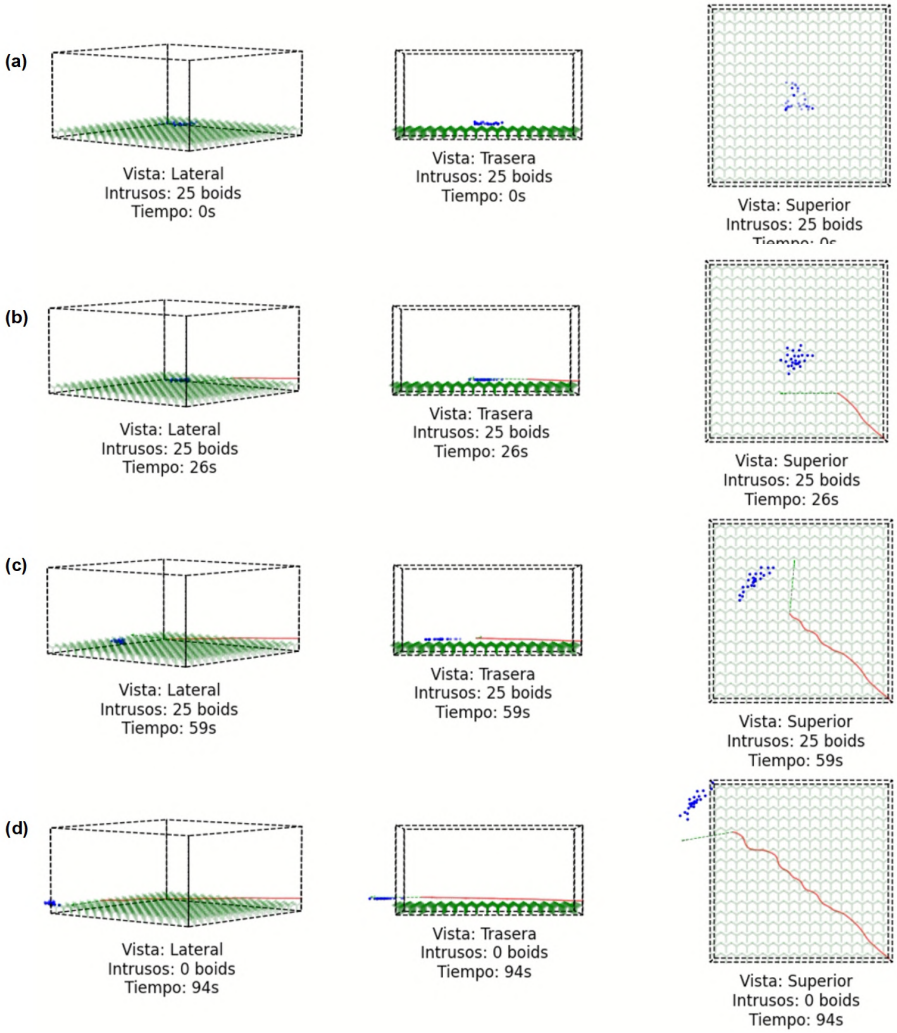



Figura 4.9: Fotogramas de una ejecución de la heurística $H$ con 25 boids, para los instantes de tiempo $t = 0s$ (a), $26s$ (b), $59s$ (c) y $94s$ (d). Cada figura muestra tres vistas: lateral, trasera y frontal, junto con el número de boids que aún están dentro del espacio aéreo.

Esta página ha sido intencionalmente dejada en blanco.

# Capítulo 5

# Solución aplicando *Reinforcement Learning*

En este capítulo se aborda el problema de la disuasión de aves aplicando el algoritmo de *Reinforcement Learning*, o *RL*[1], denominado *Deep Deterministic Policy Gradient* o *DDPG* [44], el cual se detalla en el apéndice C. Este algoritmo, a diferencia de sus predecesores, tales como *Q-Learning* y *DQN*, trabaja directamente sobre espacios de estados y acciones continuos. Esto evita, por ejemplo, tener que discretizar el estado del dron y sus acciones, que son inherentemente continuos, resolviendo así el problema de dimensionalidad.

Para poder utilizar *DDPG*, es necesario definir previamente los siguientes elementos:

- Espacio de estados $S$
- Espacio de acciones $A$
- Esquema de recompensas $R$
- Arquitectura de las redes Actor($\pi$) y Crítico($Q$)

A continuación, se detalla cada uno de estos elementos.

## 5.1. Espacio de Estados

Cada estado debe codificar toda la información relevante del sistema, que en este caso es la posición, orientación y velocidad del dron en el instante $t$, así como también la distribución de la bandada.

### 5.1.1. Dron

En el capítulo previo se introdujo la clase $E$ que controla el entorno de simulación, la cual a su vez contiene la subclase $s_D$ que representa el estado del dron:

$$s_D(t) = (x, y, z, \theta, v_{x'}, v_{z'}, w)|_{t=t}$$

siendo $x(t), y(t), z(t)$ su posición expresada en un sistema de coordenadas global $xyz$, $\theta(t)$ su orientación, y $v_{x'}(t), v_{z'}(t), w(t)$ sus velocidades, expresadas en el sistema de coordenadas $x'y'z'$ local al dron, en donde $z'||z$.

[1] Ver apéndice B

Los valores en cada dimensión del estado están expresados en un sistema de unidades predeterminado. Para poder entrenar una red neuronal [2] que funcione correctamente con un sistema de unidades arbitrario, es necesario normalizar cada valor, por ejemplo, dentro del intervalo $[-1, 1]$.

De este modo, si el espacio aéreo a proteger tiene las dimensiones $L$x$A$x$H$, se define el estado normalizado del dron como:

$$\hat{s}_D(t) = [\hat{x}, \hat{y}, \hat{z}, \hat{\theta}, \hat{v}_{x'}, \hat{v}_{z'}, \hat{w}]|_{t=t}$$
$$= \left[\frac{2(x-x_0)}{L} - 1, \frac{2(y-y_0)}{A} - 1, \frac{2(z-z_0)}{H} - 1, \frac{2\theta}{\pi} - 1, \frac{2v_{x'}}{V_x} - 1, \frac{v_{z'}}{V_z}, \frac{w}{W}\right]_{|t=t}$$

siendo $\theta(t) \in [0, 2\pi]$, $(x_0, y_0, z_0)$ el origen de la grilla y $V_x, V_z, W$ las velocidades máximas del dron, introducidas en el capítulo 3. El sistema de coordenadas normalizado tiene como origen el centro del espacio aéreo. Los límites de este espacio tomarán los valores $-1$ y 1 según corresponda, como muestra la figura 5.1. Lo mismo ocurre con las velocidades, por ejemplo, $v_{x'} \in [0, V_x]$ mientras que $\hat{v}_{x'} \in [-1, 1]$, siendo $\hat{v}_{x'} = \frac{2v_{x'}}{V_x} - 1$.

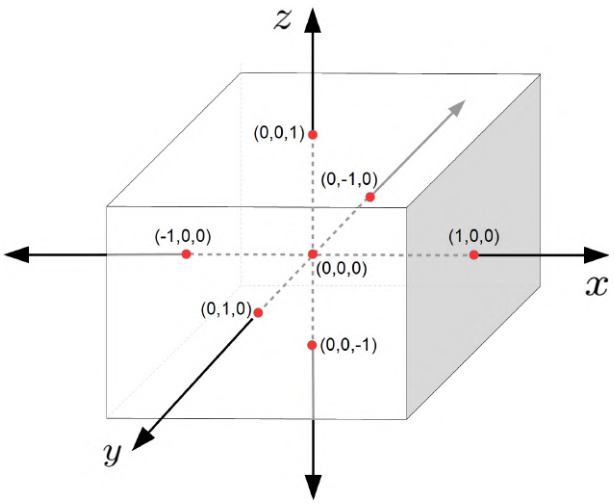



Figura 5.1: Espacio aéreo normalizado a $[-1, 1]^3$

### 5.1.2. Bandada

La codificación del estado de la bandada es un poco más compleja. Idealmente, debería estar representada por la matriz de ocupación $M_t$. Sin embargo, la inclusión de esta matriz incrementaría en forma significativa la dimensión del estado $s_t$, por lo que el rendimiento del algoritmo de entrenamiento se degradaría considerablemente. Por otro lado, la mayoría de las celdas de la matriz $M_t$ contienen el valor cero (matriz dispersa), por lo que se estaría haciendo un uso muy ineficiente de la memoria.

Por esta razón, se decidió representar la bandada con un muestreo de $N_m$ celdas ocupadas de la matriz $M_t$, siendo $N_m$ un hiperparámetro. Cada celda muestreada $\vec{c_i}$ se representa por su centro $[x_i(t), y_i(t), z_i(t)]^T$. En caso de haber menos de $N_m$ celdas ocupadas, simplemente se completa la lista con la celda origen $[0, 0, 0]^T$. Por otro lado, el muestreo de celdas no es al azar, sino que se seleccionan siguiendo el algoritmo 2.

[2] El lector no familiarizado con el concepto de red neuronal, puede encontrar una rápida introducción en [31]

**Algorithm 2:** Muestreo de celdas ocupadas

**Data:** Conjunto de celdas ocupadas $O$
**Data:** Constante $N_m$
**Data:** Posición del dron $\vec{p}$
**Result:** Conjunto muestreo $C$

```
1  if |O| < N_m then
       // La cantidad de celdas ocupadas es menor a N_m
2      C ← O
3      for i ← 1 to N_m − |O| do
4          C ← C ∪ {[0, 0, 0]^T} ;                    // Completar con celda nula
5      end
6  else
7      c⃗_0 ← ArgMin_c⃗ {‖c⃗ − p⃗‖, c⃗ ∈ O} ;            // Celda más próxima al dron
8      C ← {c⃗_0} ;                                  // C: conjunto de celdas muestreadas
9      for i ← 1 to N_m − 1 do
           // Añadir la celda más alejada de C
10         c⃗_i ← ArgMax_c⃗ {D_set(c⃗, C), c⃗ ∈ (O − C)}
11         C ← C ∪ {c⃗_i}
12     end
13 end
14 return C
```

En dicho algoritmo, el conjunto de celdas muestreadas $C$ se inicializa con la celda ocupada más próxima al dron. Luego, en forma iterativa, se toma del conjunto de celdas ocupadas $O$ aún no seleccionadas $(O - C)$, aquella que se encuentre a mayor distancia del conjunto $C$ actual. Se repite el proceso hasta completar una lista de $N_m$ celdas muestreadas.

La distancia entre una celda y el conjunto $C$, se define en forma análoga a la distancia entre un punto y una recta; esto es, la mínima distancia euclidiana entre dicho punto y cualquier otro punto perteneciente a la recta.

De este modo, se define la distancia entre la celda $\vec{c}$ y el conjunto de celdas $C$ como:

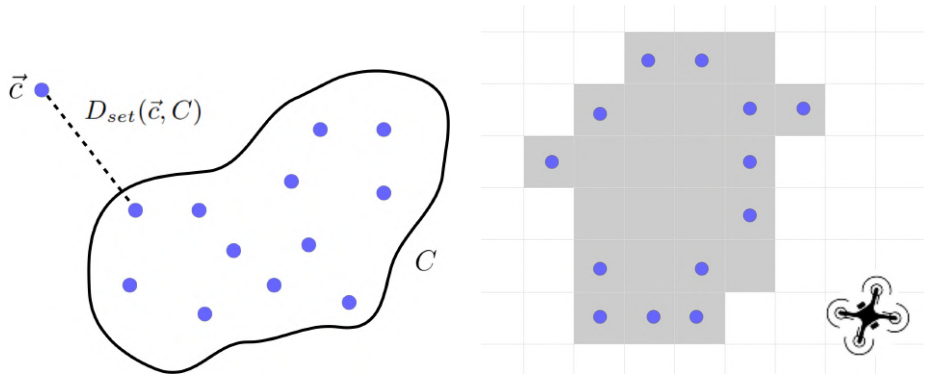
$$D_{set}(\vec{c}, C) = \text{mín}\{\|\vec{c} - \vec{x}\|,\ \vec{x} \in C\}$$

La figura 5.2(a) muestra, en forma esquemática, una instancia de $D_{set}(\vec{c}, C)$. Este algoritmo de muestreo tiende a seleccionar aquellas celdas que se encuentran sobre los límites de la bandada, lo cual brinda una representación adecuada de la distribución espacial de las celdas, como se observa en la figura 5.2(b). Adicionalmente, el conjunto de celdas seleccionadas con este método es más estable a lo largo del tiempo, a diferencia de lo que ocurre con un método de selección estocástico, en el cual una determinada celda será seleccionada de forma intermitente en cada instante de tiempo.

Una vez construido el conjunto $C$, se puede definir el estado de la bandada $s_B$ como una tupla que contiene los centros de las celdas muestreadas $c_1, c_2, ..., c_{N_m}$:

$$s_B(t) = (x_1, y_1, z_1, ..., x_{N_m}, y_{N_m}, z_{N_m})\,|_{t=t}$$

Luego, aplicando la normalización dentro del intervalo $[-1, 1]$, se obtiene $\hat{s}_B(t)$:




(a) (b)

Figura 5.2: (a) $D_{set}(\vec{c}, C)$: mínima distancia Euclidiana entre la celda $\vec{c}$ y toda celda $\vec{x} \in C$. (b) Conjunto de celdas ocupadas $O$ (en gris) y conjunto de celdas muestreadas $C$ (con punto azul).

$$\hat{s}_B(t) = 2\left[\frac{x_1(t)-x_0}{L}, \frac{y_1(t)-y_0}{A}, \frac{z_1(t)-z_0)}{H}, ...., \frac{x_{N_m}(t)-x_0}{L}, \frac{y_{N_m}(t)-y_0}{A}, \frac{z_{N_m}(t)-z_0}{H}\right] - \vec{1}$$

### 5.1.3. Estado completo

Finalmente, con las tuplas normalizadas $\hat{s}_D(t)$ y $\hat{s}_B(t)$ se construye todo el estado necesario para entrenar la red neuronal:

$$s_t = (\hat{s}_D, \hat{s}_B)|_{t=t}$$

En donde $s_t \in [-1, 1]^{7+3N_m}$

## 5.2. Espacio de Acciones

Al igual que el espacio de estados, el espacio de acciones es continuo. Cada acción $a_t$ representa las velocidades lineal y angular normalizadas que se deben aplicar al dron en el instante $t$:

$$a_t = (\hat{v}_{x'}, \hat{v}_{z'}, \hat{w})|_{t=t}$$

Estas velocidades se obtienen como salida de la última capa de la red $\pi_\psi$ del algoritmo $DDPG$. En esta capa se aplica la función tangente hiperbólica ($tanh$) como función de activación, cuyos valores se encuentran en el intervalo $(-1, 1)$; por tanto, es necesario revertir la normalización para obtener las velocidades efectivas $v_{x'}, v_{z'}$ y $w$ que se aplican al dron:

$$v_{x'}(t) = \left(\frac{\hat{v}_{x'}(t)+1}{2}\right) V_x$$

$$v_{z'}(t) = \hat{v}_z(t) V_z$$

$$w(t) = \hat{w}(t) W$$

## 5.3. Recompensas

En el problema de la disuasión de aves, el objetivo es desplazar la bandada hacia los límites del espacio aéreo en cualquier dirección hasta lograr expulsarla. Un esquema de recompensas directamente alineado con este objetivo es otorgar al agente una recompensa cuando la bandada se aleja del centro del espacio aéreo, o una penalización en caso contrario.

Para lograr esto, primero se define el valor $D_{flock}(t)$ como la distancia de la bandada al centro del espacio aéreo, haciendo uso de la función $D_{set}$ introducida previamente:

$$D_{flock}(t) = D_{set}(\vec{c_f}, O_t)$$

$$\vec{c_f} = \left[x_0 + \frac{L}{2}, y_0 + \frac{A}{2}, z_0 + \frac{H}{2}\right]^T$$

$$O_t = \{[x_i, y_j, z_k]^T|_{t=t} \ / \ M_t[i,j,k] = 1\}$$

siendo $\vec{c_f}$ el centro geométrico de un espacio aéreo con dimensiones $L$x$A$x$H$ y origen $(x_0, y_0, z_0)$, y $O_t$ el conjunto de celdas ocupadas en el instante $t$ representadas por sus centros.

La expulsión de la bandada fuera del espacio aéreo protegido puede hacerse hacia los laterales del campo cultivado o hacia arriba del mismo, más allá de la altura máxima definida por la grilla de ocupación. Si bien ambas opciones son válidas, desplazar la bandada hacia arriba implica dejarla dentro del perímetro del campo cultivado, a una altura que en principio es arbitraria. Si el campo es extenso, es probable que las aves desciendan sobre el mismo un tiempo después, en lugar de alejarse. Esto, a su vez, volvería a activar el dron que tal vez se encuentre aún en vuelo de regreso a su base. Para evitar esta situación y aprovechar mejor el despliegue del dron, se decide expulsar la bandada hacia los bordes laterales del campo y no simplemente hacia arriba. Por esta razón, se considera la distancia de las celdas proyectadas sobre el plano $xy$ al centro del campo cultivado, como muestra la figura 5.3,

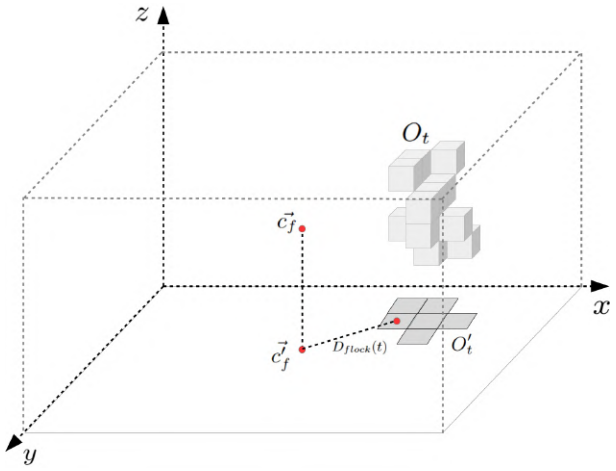



Figura 5.3: Recompensa $r_t$

De esta forma se redefine $D_{flock}(t)$ como:

$$D_{flock}(t) = D_{set}(\vec{c_f}, O'_t)$$

siendo:

$$\vec{c_f'} = Pr_{xy}(\vec{c_f})$$

$$O_t' = \{Pr_{xy}(c) \ / \ c \in O_t\}$$

Adicionalmente, se define la distancia $d_t$ entre el dron y el centro de las celdas ocupadas:

$$d_t = \|[x(t), y(t), z(t)]^T - \vec{C}_{flock}(t)\|$$

$$\vec{C}_{flock}(t) = \frac{1}{|O_t|} \sum_{\vec{c_i} \in O_t} \vec{c_i}$$

siendo $\vec{C}_{flock}(t)$ el centro geométrico de los centros de las celdas ocupadas.

Por otro lado, también se define el valor $b_t$ como la cantidad de boids dentro del espacio aéreo en el instante $t$.

Finalmente, con estos tres elementos, $D_{flock}(t), d_t$ y $b_t$, se define la recompensa $r_t$ que recibe el agente luego de ejecutar la acción $a_t$ aplicando *potential based-reward shaping*[3] :

$$r_t = w_0[D_{flock}(t) - D_{flock}(t-1)] + w_1[b_{t-1} - b_t] + w_2[d_{t-t} - d_t]\mathbf{1}_{dt>u_d} + \epsilon$$

siendo $\epsilon < 0$ y $w_i > 0$ $i = 0, 1, 2$. Las constantes $w_i$ asignan pesos relativos a cada componente de la recompensa.

El término $[D_{flock}(t) - D_{flock}(t-1)]$, ponderado con $w_0$, premia al agente (con un valor positivo) cada vez que la bandada se desplaza en la dirección correcta, alejándose del centro del campo, y lo penaliza (con un valor negativo) en caso contrario.

Por su parte, el término $[b_{t-1} - b_t]$, ponderado con $w_1$, recompensa al agente cada vez que se reduce la cantidad de boids dentro del espacio aéreo, reforzando la importancia de expulsar la bandada.

El término $[d_{t-t} - d_t]$, ponderado con $w_2$, incentiva al agente a acercarse a las aves y lo penaliza cuando se aleja. Sin embargo, este término solo está presente cuando el dron se encuentra más allá de cierta distancia de la bandada, determinada por el umbral $u_d$. Este incentivo, por tanto, se desactiva cuando el dron está a una distancia menor a $u_d$ del centro de la bandada, evitando así que interfiera con los otros componentes de la recompensa.

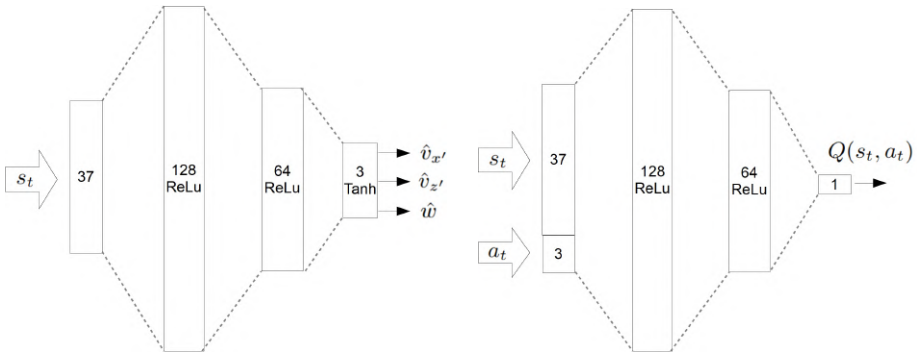
Finalmente, si el dron permanece inmóvil, cada término de la recompensa se vuelve nulo. Por esta razón, se añade la constante negativa $\epsilon$ que penaliza al dron por no moverse.

### 5.3.1. Arquitectura de las redes

Tanto la red del actor $\pi_\psi$, como la del crítico $Q_\theta$, se componen de dos capas ocultas totalmente conectadas de 128 y 64 neuronas, respectivamente, con funciones de activación *ReLU* [4], como muestra la figura 5.4. En la capa de salida de la red $\pi_\psi$ se aplica la función *tanh*. La elección se basó en una versión reducida de la arquitectura propuesta en [85], tomando en cuenta solo las capas totalmente conectadas y descartando las capas de convolución previas que procesan las imágenes de los sensores utilizados en dicho estudio.

[3] Ver descripción de este concepto en el apéndice D

[4] El lector no familiarizado con el concepto de función de activación, puede encontrar una rápida introducción en [31]




(a) (b)

Figura 5.4: (a) Red $\pi_\psi$ (b) Red $Q_\theta$

Eligiendo el número de celdas muestreadas $N_m = 10$, el estado $s_t$ tiene dimensión $7 + 3N_m = 37$. Este es el número de neuronas en la capa de entrada de la red $\pi_\psi$, y su salida se corresponde con las velocidades normalizadas del dron $\hat{v}_{x'}$, $\hat{v}_{z'}$ y $\hat{w}$.

En cuanto a la red $Q_\theta$, la entrada de la misma tiene dimensión $37 + 3 = 40$, siendo la concatenación del estado $s_t$ con la acción $a_t$ ejecutada en dicho estado. La salida es escalar y representa la estimación del valor $Q(s_t, a_t)$.

## 5.4. Resultados

Para el entrenamiento de las redes se utilizó el algoritmo detallado en el apéndice F. Luego de realizar aproximadamente un centenar de entrenamientos, variando diferentes parámetros (*learning rate*, número de boids, arquitectura de las redes, amplitud del ruido de exploración, pesos $w_i$, etc.), se constató un desempeño muy pobre de la política obtenida por el algoritmo $DDPG$. En la mayoría de los casos, la red neuronal entrenada no logra expulsar a todas las aves, y cuando lo logra, tarda mucho tiempo.

No se pudo determinar la causa de este mal resultado, pero se especula que puede estar ocurriendo algún tipo de *overfitting* en las primeras etapas del entrenamiento o que el algoritmo alcance mínimos locales de los cuales no puede salir. También es posible que se produzca el fenómeno de *reward hacking*, en donde el agente converge a una política que explota la recompensa en forma repetida sin lograr resolver el problema.

Si bien el desempeño observado no era el esperado, está en línea con los resultados hallados en [85]. En ese trabajo, ya mencionado en el capítulo 2 de antecedentes, se aborda el problema del perro pastor y el rebaño de ovejas utilizando el modelo de Reynolds. Los autores aplican el algoritmo de *RL* conocido como *Proximal Policy Optimization* [66] o $PPO$ (que es posterior a $DDPG$), el cual utiliza políticas estocásticas para guiar el rebaño hacia una posición objetivo a través de una serie de obstáculos.

El esquema de recompensas usado es muy similar al utilizado en el presente trabajo, en donde se premia al agente cuando el rebaño se desplaza hacia el objetivo, y se lo penaliza en caso contrario. Los resultados presentados sugieren que el algoritmo sólo funciona con rebaños de tamaño reducido, de hasta 4 individuos, siempre y cuando no se encuentren demasiado dispersos. Esta limitación también se ha observado en el caso de los boids.

Los autores del artículo comparan el rendimiento del algoritmo $PPO$ con la heurísti-

ca de conducción y agrupación, también mencionada en el capítulo 2. La heurística no solo es más eficiente, sino que también puede manejar grupos de cientos de individuos. En sus conclusiones, los autores sugieren como trabajo futuro combinar la heurística con un algoritmo de *RL*.

Siguiendo esta sugerencia es que se aborda, en los siguientes capítulos, la combinación del algoritmo *DDPG* con la heurística de persecución de flancos, presentada en la sección 4.1, para averiguar si es posible potenciar su rendimiento.

# Capítulo 6

# Combinando Heurística con *Reinforcement Learning*

En principio, se pueden plantear diversas formas de combinar las técnicas de $RL$ con la heurística de *persecución de flancos*. El presente capítulo presenta dos estrategias identificadas como $RL1$ y $RL2$, y que más adelante se comparan con la heurística, identificada como $H$.

## 6.1. Algoritmo RL1

Esta es la opción más simple, que consiste en utilizar la heurística $H$ para identificar, en cada instante, el flanco a perseguir para luego utilizar una red neuronal, previamente entrenada, que desplace el dron hasta dicho flanco. El algoritmo 3 muestra el seudocódigo.

**Algorithm 3:** Algoritmo $RL1$

**Data:** Entorno $E$, $T_{max}$
**Result:** $E$ actualizado

$E.reset()$
// Mientras haya boids dentro del espacio aéreo y $t < T_{max}$
**while** $E.b_{in} > 0$ *and* $E.t < T_{max}$ **do**
    $\hat{F} = Normalize(H.target(E))$ ; // Flanco objetivo
    $\hat{s}_D = Normalize(E.s_D)$ ; // Estado del dron
    $\hat{s}_t = (\hat{s}_D, \hat{F})$ ; // Estado completo
    $v_{x'}, v_{z'}, w \leftarrow Denormalize(rl1.Forward(\hat{s}_t))$ ; // Forward de red $rl1$
    $E.step(v_{x'}, v_{z'}, w)$ ; // Ejecutar un paso de la simulación
    $E.t \leftarrow E.t + \Delta t$
**end**

En cada paso se selecciona el flanco $\hat{F}$ a perseguir según el algoritmo 1 presentado en el capítulo 4, mediante la función $H.target(E)$ que recibe el entorno $E$. Luego se construye el estado $\hat{s}_t$ concatenando todo el estado del dron (posición, orientación y velocidades) con las coordenadas del flanco $\hat{F}$.

El nuevo estado $\hat{s}_t$ así obtenido, que está normalizado en el intervalo $[-1, 1]$, se utiliza como entrada de la red neuronal $rl1$. Esta red devuelve las velocidades normalizadas del dron que, una vez revertida la normalización, son utilizadas para ejecutar un paso de la simulación mediante $E.step()$.

La arquitectura de la red es similar a la utilizada en el capítulo anterior, es decir, dos capas ocultas de 128 y 64 neuronas, respectivamente. Sin embargo, esta red solo recibe el estado del dron $s_D$ y las coordenadas del flanco a perseguir, por lo que no utiliza ninguna información sobre la bandada:

$$s_t = [S_D, \vec{F}]^T|_{t=t}$$

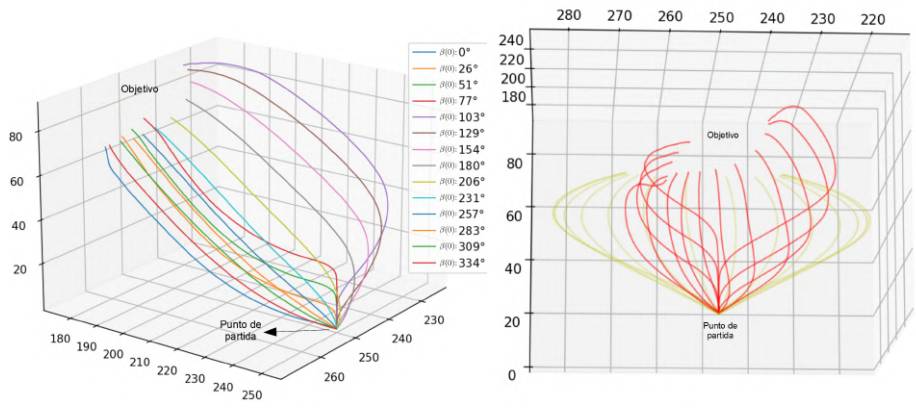
De esta forma, el espacio de estados normalizado es $S = [-1, 1]^{10}$ y el de acciones $A = [-1, 1]^3$. El algoritmo de entrenamiento es nuevamente $DDPG$. En cada episodio de entrenamiento se inicializa en forma aleatoria la posición del dron dentro del espacio aéreo, así como también la posición del objetivo.

Para la recompensa se utiliza la variación de la distancia Euclidiana entre la posición actual del dron $\vec{P}_t$ y el flanco objetivo $\vec{F}$:

$$r_t = \|\vec{F} - \vec{P}_{t-1}\| - \|\vec{F} - \vec{P}_t\|$$

De este modo si el agente se acerca a las coordenadas del objetivo $\vec{F}$, recibe un valor positivo, y en caso contrario un valor negativo. La red entrenada de esta forma, simplemente mueve el dron hacia las coordenadas indicadas por el flanco seleccionado. La elección de este flanco sigue estando a cargo de la heurística; sin embargo, se descarta la función $H_{vzw}$ que asignaba las velocidades del dron en forma proporcional al error de orientación y elevación entre el dron y el objetivo. Este cálculo lo hace ahora la red $rl1$.

La figura 6.1 (a) muestra las trayectorias generadas por esta red, mientras que la parte (b) compara estas trayectorias con las generadas por la función $H_{vzw}$, que ya fueron presentadas en la figura 4.8.




(a) (b)

Figura 6.1: (a) Trayectorias generas por la red neuronal $rl1$, para diferentes valores de $\beta(0)$. (b) Trayectorias generadas por la heurística (amarillo) vs trayectorias generadas por la red neuronal (rojo).

Las trayectorias generadas por la red neuronal no son simétricas e incluso algunas recorren una distancia mayor que su correspondiente trayectoria generada con la

heurística $H_{vzw}$. Sin embargo, el dron tarda menos tiempo en acercarse al objetivo, como se evidencia en la tabla 6.1. Para cada error de orientación inicial $\beta(0)$, se muestra el tiempo en segundos que tarda el dron en llegar a cierta distancia predeterminada del punto objetivo.

La combinación de velocidades lineal y angular elegidas por la red neuronal en cada instante resulta ser más eficiente que las velocidades elegidas por la función $H_{vzw}$, en iguales condiciones.

| $\beta(0)$ | $H_{vzw}$ | $rl1$ |
|---|---|---|
| 0.0° | 26.6 s | 15.8 s |
| 25.7° | 29.4 s | 16.2 s |
| 51.4° | 32.2 s | 17.2 s |
| 77.1° | 34.0 s | 18.6 s |
| 102.9° | 33.4 s | 20.0 s |
| 128.6° | 31.6 s | 17.8 s |
| 154.3° | 29.4 s | 15.6 s |
| 180.0° | 26.6 s | 14.0 s |
| 205.7° | 23.6 s | 13.0 s |
| 231.4° | 20.6 s | 12.8 s |
| 257.1° | 18.0 s | 12.8 s |
| 282.9° | 17.8 s | 13.0 s |
| 308.6° | 20.6 s | 13.6 s |
| 334.3° | 23.6 s | 14.6 s |

Tabla 6.1: Duración en segundos de las trayectorias mostradas en la figura 6.1(b), para la heurística $H_{vzw}$ y la red neuronal $rl1$.

Una ejecución del algoritmo 3 con 25 boids se puede ver en *Youtube* a través de este **link**. La figura 4.9 muestra una secuencia de fotogramas del video mencionado, en donde las subfiguras (a), (b), (c) y (d) muestran los instantes $t = 0s, 25s, 60s$ y $82s$, respectivamente. La bandada se encuentra inicialmente en el centro del campo, mientras que el dron parte de la esquina inferior derecha según la vista superior.

Se puede observar que el dron ejecuta una trayectoria con curvas más pronunciadas que en el caso de la heurística $H$.

## 6.2. Algoritmo RL2

Durante el entrenamiento de la red neuronal del algoritmo $RL1$, no se utiliza ninguna información sobre la elección de flancos (heurística $H$) o la bandada. El objetivo de esa red es desplazar el dron hacia un punto en el espacio previamente determinado. Tanto la heurística como la distribución de la bandada solo intervienen durante la explotación de la red, luego del entrenamiento.

La idea del algoritmo $RL2$ es incorporar estos dos elementos en el entrenamiento, con el fin de obtener una red que controle el dron completamente, sin la asistencia de la heurística e incorporando el estado de la bandada en todo momento.

De este modo, el algoritmo $RL2$ se presenta en el seudocódigo 4. El espacio de estados normalizado $S$ es idéntico al detallado en la sección 5.1.3:

$$S = [-1, 1]^{37}$$
$$\hat{s}_t = (\hat{s}_D, \hat{s}_B)|_{t=t}$$

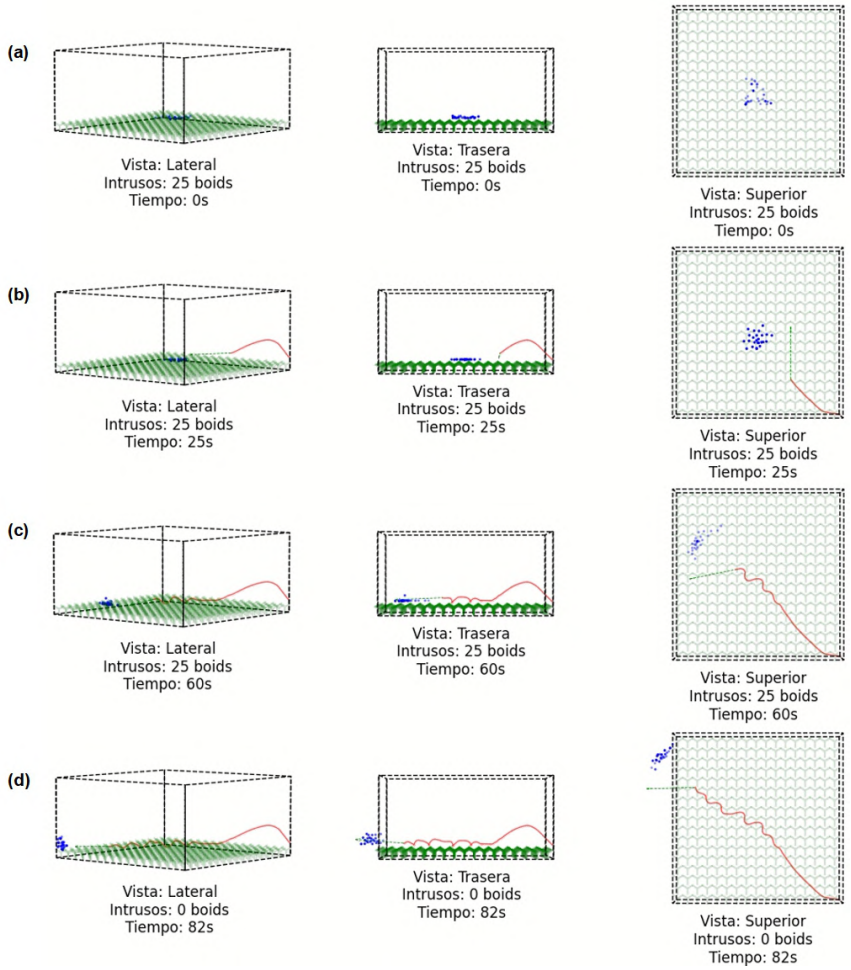



Figura 6.2: Fotogramas de una ejecución del algoritmo $RL1$ con 25 boids, para los instantes de tiempo $t = 0s$ (a), $25s$ (b), $60s$ (c) y $82s$ (d). Cada figura muestra tres vistas: lateral, trasera y frontal, junto con el número de boids que aún están dentro del espacio aéreo.

Siendo $s_D$ el estado del dron y $s_B$ el estado de la bandada, luego de hacer un muestreo de 10 celdas de la matriz $M_t$.

El espacio de acciones también es idéntico al utilizado previamente, $A = [-1, 1]^3$, siendo $a_t = (\hat{v}_{x'}, \hat{v}_{z'}, \hat{w})|_{t=t}$

Para el entrenamiento de la red, se modifica la recompensa utilizada por el algoritmo presentado en el capítulo 4, añadiendo un valor positivo si las acciones del agente se asemejan a las acciones de la heurística, y un valor negativo en caso contrario. Es una idea muy parecida a la presentada en [14], donde se utiliza una heurística $h$ para guiar el entrenamiento de un algoritmo de $RL$ genérico. El objetivo de esta estrategia es introducir información experta en el entrenamiento, añadiendo un sesgo a la recompensa para que el agente priorice las acciones que ejecutaría la heurística en cada estado, pero a la vez permitirle explorar variantes de dicha heurística que podrían resultar más eficientes.

El algoritmo de entrenamiento es nuevamente $DDPG$, pero a diferencia de la red

**Algorithm 4:** Algoritmo $RL2$

**Data:** Entorno $E$, $T_{max}$
**Result:** $E$ actualizado

$E.reset()$;
// Mientras haya boids dentro del espacio aéreo y $t < T_{max}$
**while** $E.b_{in} > 0$ *and* $E.t < T_{max}$ **do**
  $\hat{s}_D = Normalize(E.s_D)$ ; // Estado del dron
  $\hat{s}_B = Normalize(E.s_B)$ ; // Estado de la bandada
  $\hat{s}_t = (\hat{s}_D, \hat{s}_B)$
  $v_{x'}, v_{z'}, w \leftarrow Denormalize(rl2.Forward(\hat{s}_t))$ ; // Forward de red $rl2$
  $E.step(v_{x'}, v_{z'}, w)$ ; // Ejecutar un paso de la simulación
  $E.t \leftarrow E.t + \Delta t$
**end**

$rl1$, en el caso de $rl2$ se utiliza la heurística en cada episodio para seleccionar un flanco y luego se recompensa al agente si se acerca a él y se le penaliza en caso contrario.

De esta forma, se define la recompensa $r_t$ para el algoritmo $RL2$ como:

$$r_t = \epsilon + w_0(\|\vec{P}_{t-1} - \vec{F}_{t-1}\| - \|\vec{P}_t - \vec{F}_{t-1}\|) + w_1(b_{t-1} - b_t) \tag{6.1}$$

siendo $\epsilon < 0$ y $w_0, w_1 > 0$. $\vec{P}_t$ es la posición actual del dron, $\vec{F}_{t-1}$ el flanco seleccionado por la heurística en el instante $t-1$ y $b_t$ el número actual de boids dentro del espacio aéreo.

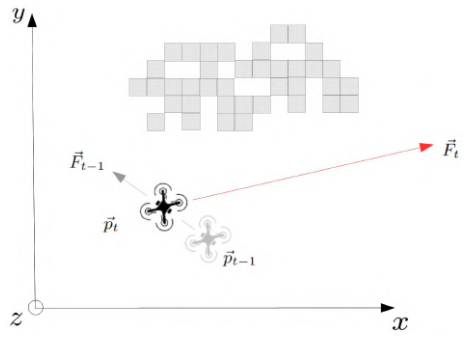



Figura 6.3: El dron se acerca al objetivo establecido en el instante $t-1$, no al nuevo objetivo establecido en el instante actual $t$.

El componente ponderado con $w_0$ premia o penaliza al agente dependiendo de si éste se acerca o se aleja, respectivamente, del flanco seleccionado en el instante previo. Cabe mencionar que ya no es necesario incentivar al agente para que se acerque a la bandada (componente $d_t$ en el algoritmo del capítulo 5) porque ahora se lo incentiva a dirigirse hacia los flancos.

Por otro lado, es importante notar que la comparación de distancias en el instante $t$ debe hacerse con respecto al flanco seleccionado en el instante $t-1$. Esto se debe a que el dron persigue un objetivo móvil y, por tanto, es necesario evaluar si el dron se acerca o se aleja del objetivo establecido en el paso anterior, y no del nuevo objetivo establecido en el paso actual.

Esto último se observa más claramente cuando el flanco cambia de izquierda a derecha o viceversa, como muestra la figura 6.3. La distancia entre $\vec{P}_t$ y $\vec{F}_t$ es significativamente mayor que la distancia entre $\vec{P}_{t-1}$ y $\vec{F}_{t-1}$, por lo que si se incorpora a la recompensa la diferencia entre ambas, se estaría asignando un valor negativo a la acción elegida por el agente, cuando en realidad debería recibir una recompensa positiva por haberse acercado al objetivo establecido en el instante previo.

En cuanto a la estrategia de entrenamiento, se utilizó el algoritmo detallado en el apéndice F. Una ejecución del algoritmo $RL2$ con 25 boids puede verse en *YouTube* a través de este **link**. Se incluye el indicador de flancos a modo de comparación con la heurística $H$.

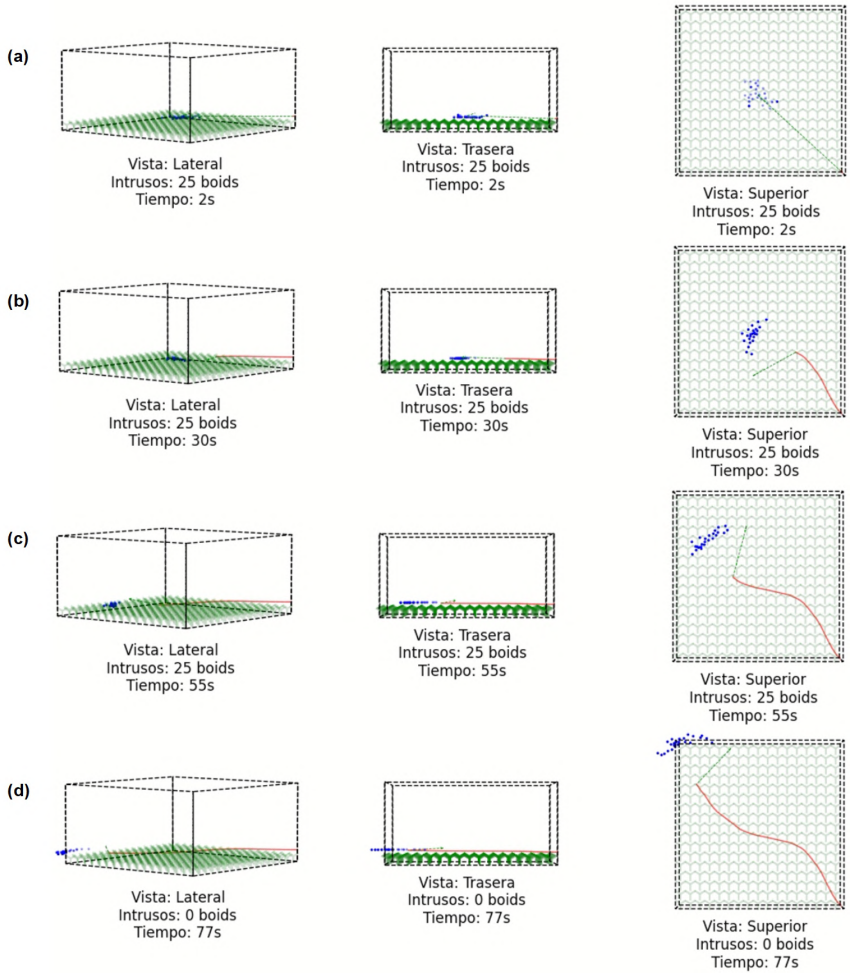



Figura 6.4: Fotogramas de una ejecución del algoritmo $RL2$ con 25 boids, para los instantes de tiempo $t = 2s$ (a), $30s$ (b), $55s$ (c) y $77s$ (d). Cada figura muestra tres vistas: lateral, trasera y frontal, junto con el número de boids que aún están dentro del espacio aéreo.

La figura 6.4 contiene una secuencia de fotogramas del mencionado video, en donde las subfiguras (a), (b), (c) y (d) muestran los instantes $t = 2s, 30s, 55s$ y $77s$, respectivamente. A diferencia del algoritmo $RL1$, el dron vuela rasante sobre los árboles, describiendo una trayectoria más directa hacia la bandada y mucho menos sinuosa.

Es importante hacer notar, además, que al incorporar la heurística al entrenamiento, el algoritmo $DDPG$ converge a una política que efectivamente resuelve el problema, aunque bajo ciertas condiciones que se verán más adelante. Esto representa una mejora con respecto al esquema de recompensas utilizado por el algoritmo del capítulo 5, el cual no incluía información experta y no lograba expulsar a las aves del espacio aéreo.

Esta página ha sido intencionalmente dejada en blanco.

# Capítulo 7

# Solución en ROS2+Gazebo

Para verificar los algoritmos detallados en los capítulos anteriores en un entorno de simulación más realista, se decidió utilizar el entorno de simulación *Gazebo Garden* [29] en combinación con la plataforma de control robótico ROS2 *Humble* [61] y el visualizador de marcadores Rviz2 [62].

*Gazebo* permite simular fenómenos físicos, incluyendo gravedad, resistencia del aire, viento, etc. con los cuales un modelo de dron puede interactuar. Como escenario de simulación, se diseñó un campo con árboles frutales muy básico, utilizando Blender 4.0 [25], como muestra la figura 7.1(a), en donde se puede apreciar, en la parte central de la imagen, una bandada de aves que sobrevuela los cultivos a cierta distancia de la edificación que se ve en primer plano.

Para simular la bandada, se coloca un modelo de ave en la posición asignada a cada boid, como se muestra en la figura 7.1(b). Cabe destacar que la bandada no está sujeta a las leyes físicas simuladas en *Gazebo*, sino que su movimiento es controlado exclusivamente por el modelo de boids presentado en el capítulo 3.

Para la simulación del vehículo aéreo se eligió el modelo de dron *3DR Iris* [1] de la compañía 3DRobotics, que cuenta con 4 rotores y un peso aproximado de 1.3 kg. La simulación del mismo la lleva a cabo un paquete de ROS2 desarrollado por *Ardupilot* [4]. La parte (c) de la figura 7.1 muestra el dron estacionado en su base.

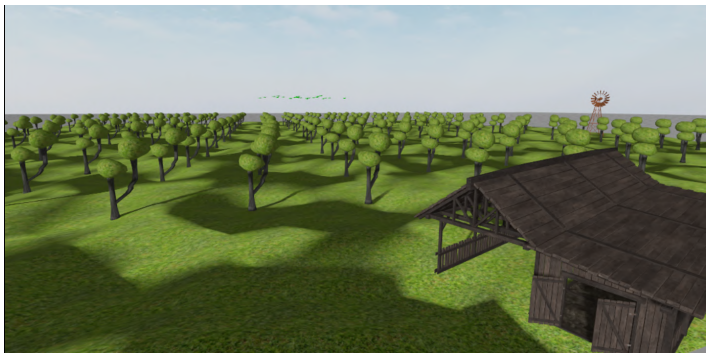
## 7.1. Control del dron

El paquete ROS2 de Ardupilot provee una serie de tópicos (con prefijo /ap) que permiten controlar el dron, así como también recibir información sobre su estado: posición, velocidad y orientación. La tabla 7.1 muestra los tres tópicos de Ardupilot usados con este fin.

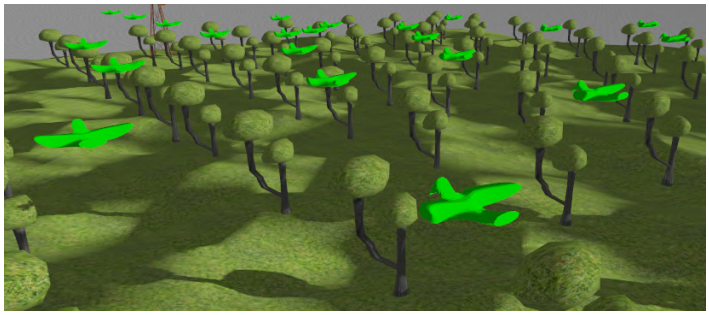
A modo de ejemplo, para especificar las velocidades lineales y angulares que se desea aplicar al dron, un nodo ROS2 debe publicar en el tópico **/ap/cmd_vel** un mensaje con la estructura indicada en el algoritmo 5.

La variable *msg* se inicializa con el constructor de mensaje TwistStamped(), que incluye una marca de tiempo. El campo *stamp* contiene la hora en que se generó el mensaje, lo que indica a ROS2 un orden de procesamiento, en caso de recibir múltiples mensajes simultáneamente.

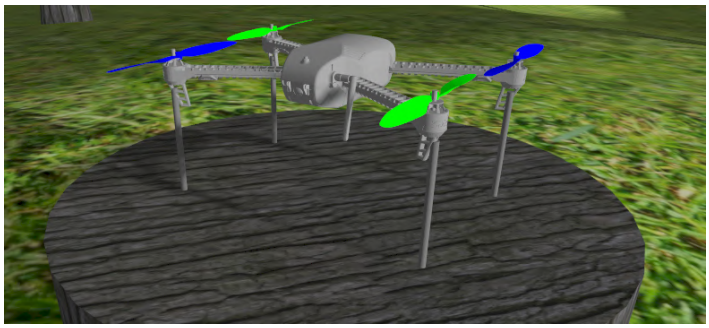
El campo *frame_id* indica el marco de referencia en el que están expresadas las velocidades, en este caso «world», que ofrece un sistema de coordenadas global fijo. ROS2 y Ardupilot se encargan de transformar estas velocidades al sistema de referencia local del dron.



(a)



(b)



(c)

Figura 7.1: (a) Campo de árboles frutales en Gazebo. (b) Detalle de la bandada de boids sobrevolando los árboles. (c) Dron 3DR Iris.

En las estructuras *twist.linear* y *twist.angular* se especifican las velocidades correspondientes a cada uno de los ejes $x, y, z$, para luego publicar el mensaje en el tópico **/ap/cmd_vel**.

Para obtener la posición y orientación actual del dron, un nodo ROS2 debe suscribirse al tópico **/ap/pose/filtered**. Los datos contenidos en este tópico son confeccionados por Ardupilot a partir de sensores como GPS e IMU. Las lecturas así obtenidas se combinan aplicando un filtro de Kalman (de allí el calificativo *filtered* del tópico) para producir una estimación más precisa del estado del dron.

| Tópico | Tipo | Uso |
|---|---|---|
| /ap/cmd_vel | TwistStamped | Recibe las velocidades lineales y angulares que comandan el dron. |
| /ap/pose/filtered | PoseStamped | Devuelve la posición actual del dron junto con su orientación en forma de cuaternio |
| /ap/twist/filtered | TwistStamped | Devuelve la velocidad lineal y angular actual del dron en los tres ejes. |

Tabla 7.1: Tópicos de Ardupilot usados para controlar y recibir información del dron

**Algorithm 5:** Publicar velocidad

**Data:** Velocidades lineales $v_x, v_y, v_z$
**Data:** Velocidades angulares $w_x, w_y, w_z$
1 msg ← TwistStamped()
2 msg.header.stamp ← Now()
3 msg.header.frame_id ← «world»
4 msg.twist.linear.x ← $v_x$
5 msg.twist.linear.y ← $v_y$
6 msg.twist.linear.z ← $v_z$
7 msg.twist.angular.x ← $w_x$
8 msg.twist.angular.y ← $w_y$
9 msg.twist.angular.z ← $w_z$
10 **/ap/cmd_vel**.publish(msg)

El algoritmo 6 muestra cómo extraer la información de este tópico.

**Algorithm 6:** Obtener posición

**Data:** Mensaje msg, de tipo PoseStamped
**Result:** Posición $(x, y, z)$. Orientación $(pitch, roll, yaw)$
1 $x \leftarrow$ msg.pose.position.x
2 $y \leftarrow$ msg.pose.position.y
3 $z \leftarrow$ msg.pose.position.z
4 $(q_x, q_y, q_z, q_w) \leftarrow$ msg.pose.orientation
5 $pitch = sin^{-1}[2(q_x q_z - q_w q_y)]$
6 $roll = tan^{-1}\left[2\left(\frac{q_x q_y + q_w q_z}{q_w q_w + q_x q_x - q_y q_y - q_z q_z}\right)\right]$
7 $yaw = tan^{-1}\left[2\left(\frac{q_y q_z + q_w * q_x}{q_w * q_w - q_x * q_x - q_y * q_y + q_z * q_z}\right)\right]$

La posición del dron se extrae directamente de la estructura *msg.pose*. Sin embargo, los ángulos de orientación, *pitch*, *roll* y *yaw*, deben calcularse a partir del cuaternio *msg.pose.orientation*, según las fórmulas indicadas [83]. En particular, es de interés el valor de *roll*, que es el ángulo de giro alrededor del eje vertical y que es utilizado por los algoritmos $H$, $RL1$ y $RL2$.

Finalmente, las velocidades instantáneas del dron se obtienen directamente del

**Algorithm 7:** Obtener velocidad

**Data:** Mensaje msg, de tipo TwistStamped
**Result:** Velocidad lineal $(v_x, v_y, v_z)$. Velocidad angular $(w_x, w_y, w_z)$

```
1 v_x ← msg.twist.linear.x
2 v_y ← msg.twist.linear.y
3 v_z ← msg.twist.linear.z
4 w_x ← msg.twist.angular.x
5 w_y ← msg.twist.angular.y
6 w_z ← msg.twist.angular.z
```

tópico **/ap/twist/filtered** sin ser necesaria una transformación, según se muestra en el algoritmo 7.

## 7.2. Nodos

La simulación del dron y la bandada en Gazebo se realiza mediante la coordinación de un grupo de nodos ROS2, desarrollados en Pyhton y C++. La tabla 7.2 lista los nodos más relevantes.

| Nodo | Lenguaje | Autor | Descripción |
|---|---|---|---|
| ardupilot_dds | C++ | ardupilot.org | Se encarga de controlar el dron a través de comandos recibidos en el tópico ap/cmd_vel. |
| ros_gz_bridge | C++ | gazebosim.org | Provee una interfaz entre ROS2 y *Gazebo Transport*, la capa de comunicación de Gazebo. |
| rviz | C++ | ros.org | Inicia la utilidad de visualización de marcadores Rviz2 |
| flock3d | Python | a medida | Se encarga de la simulación de la bandada de boids. |
| flock_move | C++ | a medida | Mueve cada modelo de boid en Gazebo a la posición indicada por flock3d |
| nav_rl | Python | a medida | Contiene toda la lógica de control del dron necesaria para ejecutar episodios en secuencia, utilizando los algoritmos $H$, $RL1$ o $RL2$. |
| avoidance_srv | Python | a medida | Ejecuta el algoritmo de evasión de obstáculos. |

Tabla 7.2: Principales nodos ROS2 utilizados para simular en Gazebo los algoritmos $H$, $RL1$ y $RL2$.

La figura 7.2 muestra los nodos (rectángulos) y tópicos (elipses), junto con la dirección en que los nodos publican y se suscriben a dichos tópicos. El nodo ardupilot_dss, que controla el dron, publica los tópicos de posición y velocidad descritos en la sección

anterior.

Por otro lado, ardupilot_dss se suscribe al tópico *ap/cmd_vel*, en donde recibe los comandos de velocidad lineal y angular que debe aplicar al dron, provenientes del nodo nav_rl.

El nodo flock3d, que se encarga de la simulación de la bandada, utiliza el mismo sistema de coordenadas del entorno de entrenamiento, el cual difiere del sistema de coordenadas utilizado en Gazebo. Se realiza, por tanto, una conversión bidireccional entre ambos sistemas: por un lado, se traduce la posición de cada boid en el entorno de entrenamiento a su correspondiente posición en el entorno Gazebo, y por otro, se traduce la posición del dron en Gazebo a su correspondiente posición en el entorno de entrenamiento.

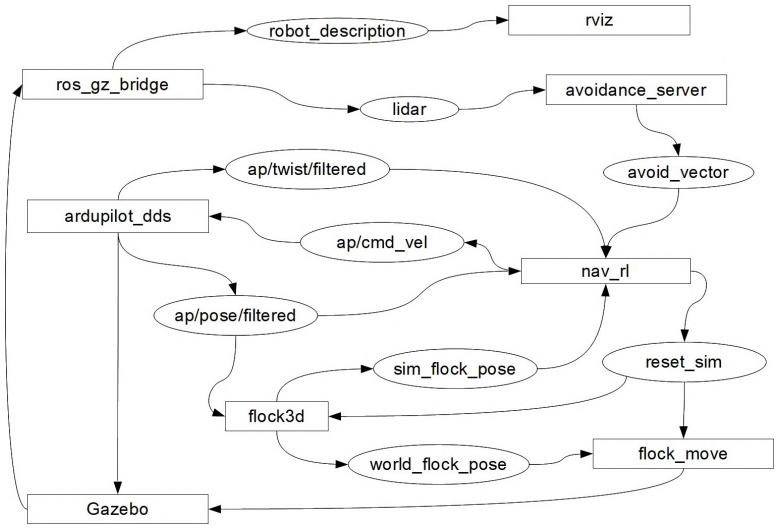



Figura 7.2: Nodos (rectángulos) y tópicos (elipses) en ROS2

Esto significa que coexisten dos coordenadas diferentes por cada boid, una proveniente del entorno *simulado* (de entrenamiento) y la otra del entorno Gazebo, que juega el rol de entorno *real*. Estas coordenadas se publican en los tópicos *sim_flock_pose* y *world_flock_pose* respectivamente. Estos tópicos son de tipo *PoseArray*, lo que significa que se codifican las coordenadas de todos los boids en un array, y luego este array es publicado en un único mensaje.

El nodo nav_rl recibe las coordenadas de la bandada en el tópico *sim_flock_pose* y las utiliza para construir la entrada de las redes neuronales o para el cálculo de flancos de la heurística $H$. Una vez que el episodio se completa, el nodo publica un mensaje vacío en el tópico *reset_sim*, para indicar que se reinicie la posición de la bandada y así poder comenzar con el siguiente episodio.

El nodo flock_move, desarrollado en *C++*, se encarga de generar en Gazebo cada uno de los boids de la bandada, cada vez que recibe un nuevo conjunto de coordenadas en el tópico *world_flock_pose*. Por cada boid, el nodo invoca el servicio de Gazebo */world/ «world name»/create* [27], que permite hacer *spawn* de un modelo predefinido, que en este caso es el modelo de un boid, y luego invoca el servicio */world/«world name»/set_pose* que coloca al modelo en las coordenadas especificadas. Como se mencionó previamente, cada modelo de boid no está afectado por las leyes físicas, pero sí tiene definido el efecto de colisión, basado en una geometría esférica que cubre las dimensiones del modelo.

El nodo ros_gz_bride, como su nombre lo indica, sirve como puente entre ROS2 y Gazebo y permite hacer un mapeo bidireccional entre tópicos nativos de Gazebo y ROS2. Por un lado, publica el tópico *robot_description* que es utilizado por Rviz2 para visualizar la actividad del dron. Por otro lado, también publica el tópico *lidar* que contiene las lecturas de un sensor LiDar colocado en la parte frontal del dron. Este sensor, que gira 360° en forma horizontal y 90° en forma vertical, permite detectar posibles obstáculos y es utilizado por el nodo avoidance_server, detallado en las siguientes secciones.

Un ejemplo de episodio en Gazebo, usando los algoritmos $H$ y $RL2$ y con una bandada de 100 boids, puede verse en *Youtube* a través de los siguientes links: **H** y **RL2**. En dichos videos, el dron despega de su base ubicada en una esquina del campo cultivado. Una vez alcanza cierta altura predeterminada, se inicia la estrategia de disuasión. La figura 7.3 muestra una secuencia de fotogramas del algoritmo $RL2$. La mitad izquierda de la imagen contiene la simulación en Gazebo, en donde la cámara sigue al dron, colocándose unos metros por detrás de este. Por otro lado, la mitad derecha de la imagen muestra en forma simultánea la visualización de marcadores en Rviz2. Allí se puede apreciar, además de los límites del campo, cómo evoluciona la trayectoria que el dron ejecuta junto con un grupo de cubos que representan las celdas ocupadas de la matriz $M_t$.

### 7.2.1. Nodo nav_rl

El nodo nav_rl contiene la lógica principal de ejecución de episodios, que permite realizar experimentos en forma automatizada. El algoritmo 8 muestra el seudocódigo correspondiente.

En primer lugar, se establece qué algoritmo se va a ejecutar: $H, RL1$ o $RL2$. A continuación, el nodo se suscribe a los tópicos *sim_flock_pose*, *ap/pose/filtered* y *ap/twist/filtered* (líneas 1,2 y 3). Con la información recibida, se construye el estado del dron $s_D$ y el estado de la bandada $s_B$. Luego, se normalizan estos estados dentro del intervalo $[-1, 1]$, obteniendo $\hat{s}_D$ y $\hat{s}_B$ (líneas 4 a 7).

Si el episodio ha terminado (no hay boids presentes en el espacio aéreo o se alcanzó el tiempo máximo $T_{max}$), entonces se dirige el dron hacia su posición inicial para comenzar un nuevo episodio (líneas 8 a 11). Esto se logra simplemente ejecutando, en forma repetida, la heurística $H_{vzw}$ hasta alcanzar la posición de partida. Una vez allí, se publica un mensaje vacío en el tópico *reset_sim* para indicar a los demás nodos que se debe inicializar la posición de la bandada.

Si el episodio aún no finalizó, entonces es necesario calcular las velocidades a aplicar al dron para ejecutar un nuevo paso de la simulación. A partir de aquí se discute según el algoritmo que se esté ejecutando (línea 13 en adelante).

Si se trata de la heurística ($A = H$), entonces se obtiene el flanco $\vec{T}$ a perseguir con $H.target(E)$, para luego calcular las velocidades $v_{x'}, v_{z'}$ y $w$ mediante la heurística $H_{vzw}$. Para esto se toma la posición actual del dron $\vec{P}$ del estado $\hat{s}_D$ (líneas 14 y 15).

En el caso de tratarse del algoritmo $RL1$ (línea 17), se obtiene nuevamente el flanco a perseguir que luego se normaliza, obteniendo $\hat{T}$. Con el estado del dron $\hat{s}_D$ y el flanco $\hat{T}$ se construye el estado $\hat{s}_t$. Con este estado, a su vez, se hace un *forward* de la red *rl1* para obtener finalmente las velocidades deseadas del dron (líneas 18 a 21).

Por último, si se trata del algoritmo $RL2$, simplemente se hace un *forward* de la red *rl2* con el estado completo, es decir, $\hat{s}_t = (\hat{s}_D, \hat{s}_B)|_{t=t}$ (líneas 23 a 25).

Las velocidades $v_{x'}$ y $v_{z'}$ obtenidas en cada caso están expresadas en el sistema de coordenadas $x'y'z'$ local al dron, por lo que es necesario convertirlas al sistema global de coordenadas $xyz$. De esta forma, se obtienen las velocidades deseadas del dron en los tres ejes: $v_x, v_y, v_z$ (línea 29).

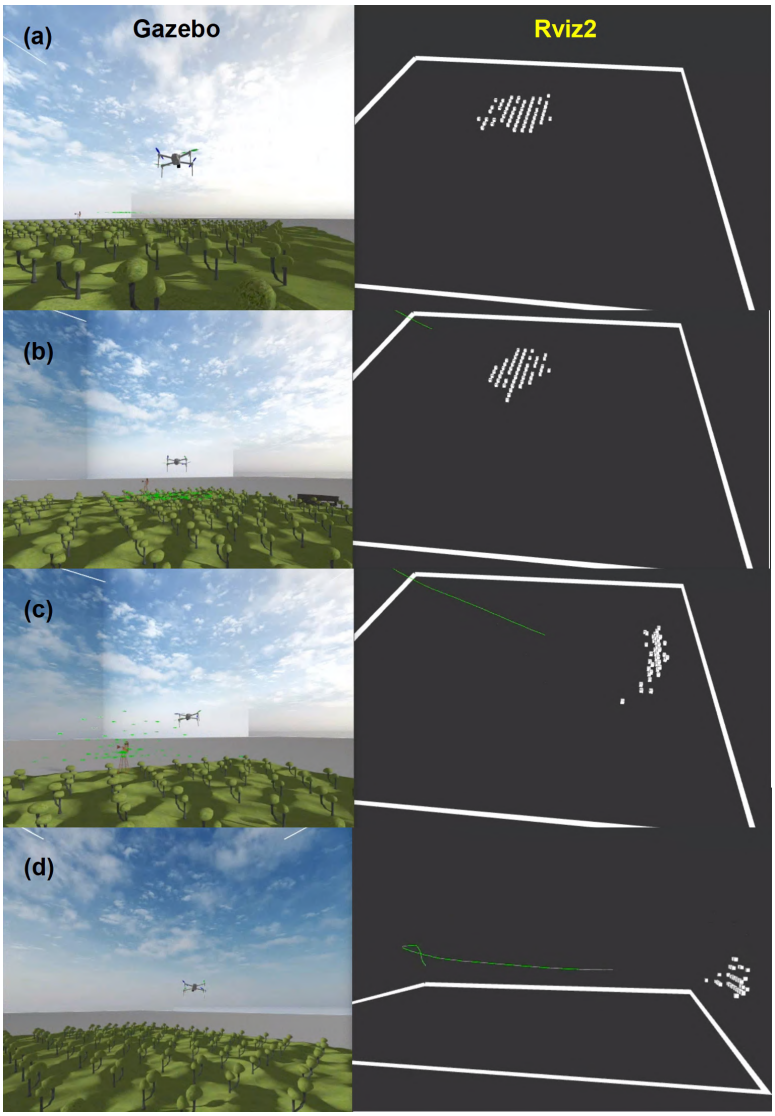



Figura 7.3: Fotogramas de un episodio del algoritmo $RL2$ en Gazebo con 100 aves. (a) El dron despega de su base y (b) se eleva en línea recta hasta una altura predeterminada para iniciar la estrategia de disuasión. (c) Los boids comienzan a reaccionar ante la presencia del dron. (d) El episodio finaliza, con todas las celdas ocupadas estando fuera de los limites del campo.

Por otro lado, las velocidades angulares deseadas en los tres ejes son $w_x = 0, w_y = 0, w_z = w$. Cabe aclarar que el sistema de coordenadas local $x'y'z'$ se refiere al dron ideal definido en el entorno de entrenamiento, en donde $z'||z$ y no existe rotación con respecto a los ejes $x'$ e $y'$ (*pitch* y *roll*).

Finalmente, las velocidades globales deseadas alimentan la entrada de un controlador proporcional de constante $K_p$, que devuelve las velocidades efectivas a aplicar al dron: $v'_x, v'_y, v'_z, w'_x, w'_y, w'_z$ (línea 30). La figura 7.4 muestra el diagrama del controlador, en donde:

**Algorithm 8:** Nodo nav_rl

**Data:** Entorno de entrenamiento $E$
**Data:** Algoritmo A=$H, RL1$ o $RL2$

subscribe_to($sim\_flock\_pose$) ; // Posición de los boids
subscribe_to($ap/pose/filtered$) ; // Posición del dron
subscribe_to($ap/twist/filtered$) ; // Velocidades del dron
$E.s_D \leftarrow$ buid_state(«Dron») ; // Construir estado del dron
$E.s_B \leftarrow$ buid_state(«Flock») ; // Construir estado de la bandada
$\hat{s}_D \leftarrow Normalize(E.s_D)$
$\hat{s}_B \leftarrow Normalize(E.s_B)$
**if** $E.b_{in} = 0$ *or* $E.t = T_{max}$ **then**
    Volar dron a su posición de inicio
    reset_sim.publish($empty$) ; // Publicar en tópico reset_sim
    $E.reset()$ ; // Inicializar episodio
**else**
    **if** $A = H$ **then**
        // Heurisica $H$: Seleccionar flanco y calcular velocidades
        $\vec{T} \leftarrow H.target(E)$
        $v_{x'}, v_{z'}, w \leftarrow H_{vzw}(\hat{s}_D.\vec{P}, T)$
    **else**
        **if** $A = RL1$ **then**
            // Algoritmo $RL1$: Seleccionar flanco y usar red $rl1$
            $\vec{T} \leftarrow H.target(E)$
            $\hat{T} \leftarrow Normalize(\vec{T})$
            $\hat{s}_t = (\hat{s}_D, \hat{T})$
            $v_{x'}, v_{z'}, w \leftarrow Denormalize(rl1.forward(\hat{s}_t))$
        **else**
            // Algoritmo $RL2$: Calcular velocidades usando red $rl2$
            **if** $A = RL2$ **then**
                $\hat{s}_t = (\hat{s}_D, \hat{s}_B)$
                $v_{x'}, v_{z'}, w \leftarrow Denormalize(rl2.forward(\hat{s}_t))$
            **end**
        **end**
    **end**
    $(v_x, v_y, v_z) \leftarrow$ toGlobalCoordinates$(v_{x'}, 0, v_{z'})$
    $(v'_x, v'_y, v'_z, w'_x, w'_y, w'_z) \leftarrow PID_{K_p,0,0}(v_x, v_y, v_z, 0, 0, w)$ ; // PID
    ap.cmd_vel.publish$(v'_x, v'_y, v'_z, w'_x, w'_y, w'_z)$ ; // Publicar en cmd_vel
**end**

$$\vec{V}_{set\_point} = [v_x, v_y, v_z, 0, 0, w]^T$$
$$\vec{V}_{published} = [v'_x, v'_y, v'_z, w'_x, w'_y, w'_z]^T$$
$$\vec{V}_{measured} = [v''_x, v''_y, v''_z, w''_x, w''_y, w''_z]^T$$
$$\vec{V}_{published} = \vec{V}_{set\_point} + K_p(\vec{V}_{set\_point} - \vec{V}_{measured})$$

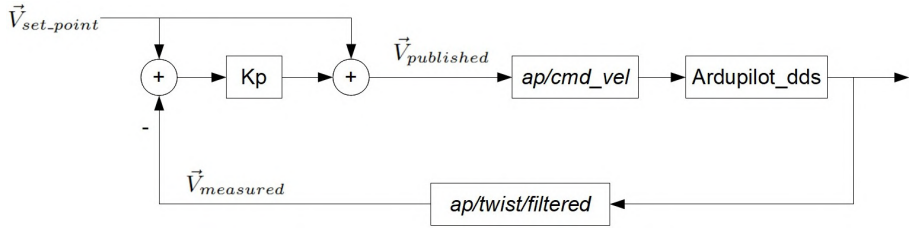



Figura 7.4: Controlador proporcional

Siendo $\vec{V}_{set\_point}$ el vector de velocidades deseadas, $\vec{V}_{measured}$ el vector de velocidades instantáneas actuales provenientes del tópico *ap/twist/filtered*, y $\vec{V}_{published}$ las velocidades publicadas en el tópico *ap/cmd_vel* con destino al nodo ardupilot_dds.

En caso de que el error entre $\vec{V}_{set\_point}$ y $\vec{V}_{measured}$ sea cero, se continúa publicando la velocidad $\vec{V}_{set\_point}$ porque, de lo contrario, el dron se detendría al no recibir comandos durante más de tres segundos.

### 7.2.2. Nodo avoidance_server

El nodo avoidance_server se encarga de generar un vector de evasión de obstáculos $\vec{a}$, el cual se publica en el tópico *avoid_vector*. Este vector es luego añadido a las velocidades lineales deseadas $v_x, v_y$ y $v_z$, para obtener un vector de velocidades netas $\vec{V}_{net}$:

$$\vec{V}_{net} = [v_x, v_y, v_z]^T + \vec{a}$$

$\vec{V}_{net}$ es la nueva velocidad que debe adquirir el dron para evitar una colisión, en lugar de $[v_x, v_y, v_z]^T$, que fue establecida por algoritmos «ciegos», que no toman en cuenta ninguna información de los obstáculos.

De este modo, el vector $\vec{a}$ define un campo potencial repulsivo alrededor de los obstáculos que modifica el vector de velocidad deseada del dron, como muestra la figura 7.5.

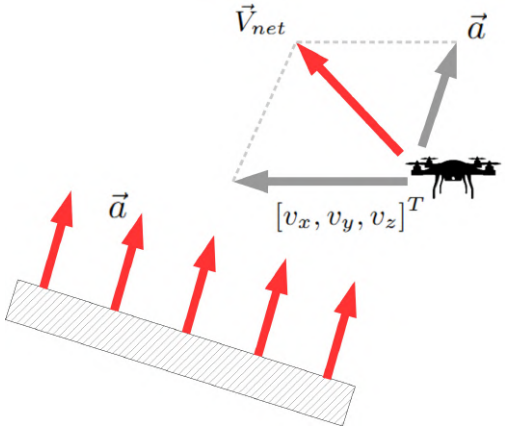



Figura 7.5: Campo potencial repulsivo alrededor de los obstáculos

Para determinar el vector $\vec{a}$, se hace uso de un sensor LiDaR montado al frente del

dron, como se muestra en la figura 7.6. El sensor realiza un escaneo horizontal de 360° y uno vertical de 90°, con lo cual se cubre una semiesfera por delante del vehículo.

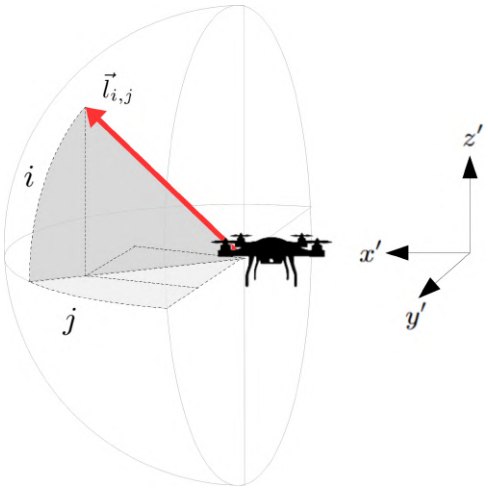



Figura 7.6: Semiesfera de detección de objetos utilizando un LiDar.

Tomando lecturas separadas por un grado de arco, el sensor devuelve una matriz de distancias $L \in R^{360x90}$, en donde $L[i,j]$ es la distancia medida cuando el sensor se encontraba en la configuración de ángulos $i, j$. Esta dirección queda definida por el vector $\vec{l}_{ij}$, tal que:

$$\vec{l}_{ij} = [sin(j)|cos(i)|, cos(j)cos(i), sen(i)]^T$$

Tomando un umbral de distancia máxima $u_l$ para detectar obstáculos, se define el vector $\vec{a}$ como la suma ponderada de los vectores opuestos a $\vec{l}_{ij}$:

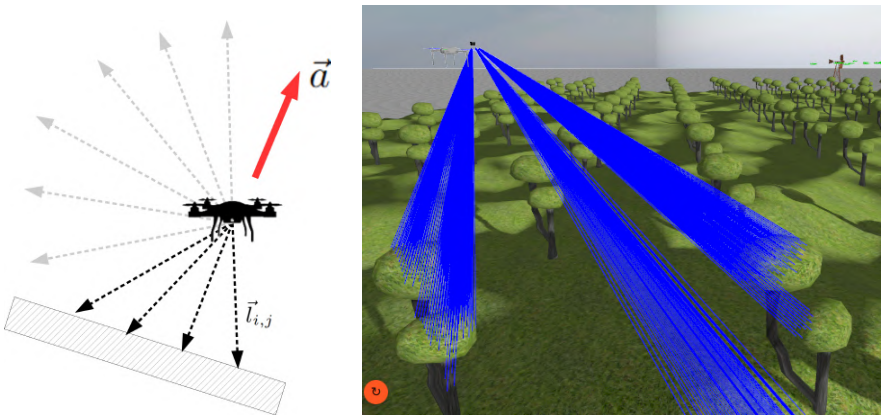
$$\vec{a} = -\sum_{L[i,j] \leq u_l} \left( \frac{u_l}{L[i,j]} \right)^4 \vec{l}_{ij}$$

El factor de ponderación $(u_l/L[i,j])^4$ hace que los vectores $\vec{l}_{ij}$ asociados a lecturas más pequeñas de distancia contribuyan en mayor medida al vector resultante $\vec{a}$. De esta forma, se da prioridad a los obstáculos que estén más cerca del dron. El exponencial 4 permite que la norma del vector $\vec{a}$ se incremente rápidamente a medida que el dron se acerca a una superficie.

Cabe aclarar que si el sensor no mide ninguna distancia menor al umbral $u_l$, el vector $\vec{a}$ es nulo, por lo que la velocidad $[v_x, v_y, v_z]^T$ no se ve afectada. La figura 7.7(a) muestra en color negro los vectores $l_{i,j}$ que intervienen en el cálculo de $\vec{a}$, y en color gris los vectores que se descartan por no impactar en ninguna superficie. Por su parte, la figura 7.7(b) muestra la visualización del LiDaR en Gazebo, donde se aprecia como el sensor detecta la copa de los árboles más próximos.

Por razones de seguridad y para no depender únicamente del algoritmo de *avoidance*, se añade una restricción a la velocidad vertical $v_z(t)$ para que el dron vuele siempre por encima del nivel de los árboles y por debajo de una altura máxima predefinida.

De esta forma, si se define $H_{min}$ y $H_{max}$ como los límites de altura permitida, $z(t)$ la altura actual del dron y $\Delta H$ cierto umbral de seguridad, entonces la velocidad vertical deseada $v_z(t)$ se recorta de la siguiente forma:




(a) (b)

Figura 7.7: (a) Cálculo del vector $\vec{a}$. (b) Visualización del sensor LiDaR en Gazebo.

$$v_z(t) = \begin{cases} v_z(t)\left[\frac{z(t)-H_{min}}{\Delta H}\right] & \text{Si } v_z(t) < 0 \text{ y } z(t) \leq H_{min} + \Delta H \\ 0 & \text{Si } z(t) \geq H_{max} \\ v_z(t) & \text{en otro caso} \end{cases} \tag{7.1}$$

Cuando la altura del dron cae por debajo de $H_{min} + \Delta H$, y se intenta imponer una velocidad $v_z(t) < 0$, entonces dicha velocidad se reduce linealmente por el factor $(z(t) - H_{min})/\Delta H$, hasta llegar a cero una vez alcanzada la altura $H_{min}$. Por debajo de esta altura, la velocidad $v_z(t)$ es siempre positiva.

La figura 7.8 ilustra esta restricción.

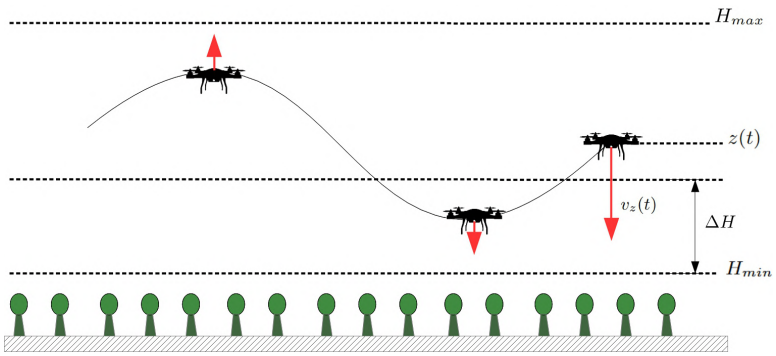



Figura 7.8: Recorte de la velocidad vertical $v_z(t)$, segun la altura $z(t)$ del dron. Por encima de $H_{max}$, se asigna velocidad vertical cero. Por debajo de $H_{min} + \Delta H$ y si $v_z(t) < 0$, entonces $v_z(t)$ se reduce linealmente hasta llegar a cero cuando el dron alcanza la altura $H_{min}$.

Esta página ha sido intencionalmente dejada en blanco.

# Capítulo 8

# Resultados

Para evaluar el desempeño de la heurística y de las redes una vez entrenadas, se realizaron una serie de experimentos en dos etapas, primero en el entorno de entrenamiento y luego en Gazebo/ROS2. Se tomó como medida de rendimiento el tiempo que tarda cada algoritmo en expulsar la bandada. Luego se confeccionaron box plots para visualizar la distribución de estos tiempos de ejecución y, mediante tests estadísticos, se determinó qué algoritmo funciona mejor en cada uno de los entornos.

Finalmente, se realizó un análisis de sensibilidad, en Gazebo/ROS2, sobre cuatro variables de interés que se consideraron relevantes: número de aves que componen la bandada, peso relativo del comportamiento de *bounding*, distancia de interacción entre los boids y el desempeño del dron bajo condiciones de viento.

Todos los experimentos se llevaron a cabo utilizando una laptop con las siguientes especificaciones:

- Modelo: Inspiron 15 3525, Dell Inc.
- Procesador: AMD Ryzen 7 5825U, 2000 Mhz, 8 Cores, 16 Logical Processors
- RAM: 16GB
- Almacenamiento: 512GB Solid State Driver
- SO: Ubuntu 22.04 LTS, Jammy Jellyfish

## 8.1. Parámetros

El apéndice A lista los valores de todos los parámetros utilizados en los experimentos. Para simplificar los algoritmos y sin pérdida de generalidad, el espacio aéreo se dividió en celdas cúbicas ($L_x = L_y = L_z$) mientras que el campo cultivado se estableció como un área cuadrada ($N_x = N_y$).

Por cuestiones de seguridad, la velocidad lineal máxima del dron se limitó a 4 m/s, permitiendo así que el algoritmo de *avoidance* funcione en forma efectiva. A diferencia del entorno de entrenamiento, los valores de velocidad máxima (lineal y angular) solo se utilizan para normalizar la entrada de las redes neuronales y no para simular el dron; esta tarea es responsabilidad de Gazebo. Por la misma razón, aquí no es necesario especificar los valores de aceleración máxima $A_v$ y $A_z$.

En cuanto a los parámetros de la bandada, se han tomado los mismos valores de pesos relativos indicados en [69], excepto el *bounding* ($w_p$) y la atracción a los árboles ($w_f$), que no están incluidos en dicho artículo.

El modelo de vehículo aéreo utilizado se basa en el dron real 3DR Iris, como el mostrado en la figura 8.1, de la compañía *3DRobotics*. Se trata de un dron con un peso total, incluyendo la batería, de 1282 gr y con unas dimensiones de 55 cm de ancho (motor a motor) y 10 cm de alto. Tiene una capacidad de carga de 400 gr, cuenta con GPS y está equipado con una batería de litio que le brinda una autonomía de vuelo de 10 a 15 minutos. Es operado por radiofrecuencia con un rango de operación de hasta 1 km [3].

Figura 8.1: Dron 3DR Iris de 3DRobotics

## 8.2. Experimentos

### 8.2.1. Diseño

La primera etapa de experimentos consistió en ejecutar 100 episodios de cada algoritmo ($H$, $RL1$ y $RL2$) en el entorno de entrenamiento, y 30 episodios por algoritmo en el entorno Gazebo/ROS2. La diferencia en el número de episodios se debe a que el entorno Gazebo/ROS2 es mucho más exigente en términos de recursos computacionales.

En cada episodio, el dron parte de la esquina inferior izquierda del campo, como muestra la figura 8.2, mientras que la bandada se coloca en el centro del mismo, un poco por encima del nivel de los árboles. A cada boid se le asigna una posición inicial aleatoria, dentro de un perímetro tridimensional preestablecido alrededor del centro de la bandada. El episodio termina cuando todos los boids son expulsados, es decir, cuando todas las celdas de la matriz de ocupación contienen el valor 0. Si esto no ocurre, el episodio termina cuando se alcanza un tiempo máximo de ejecución predeterminado.

Al comienzo de la simulación, el dron está desactivado y estacionado en su base. A continuación, se ejecuta la secuencia de despegue desde la consola *mavproxy.py* [57], que arma los motores y eleva el dron hasta una altura de 12 metros, mediante los siguientes comandos:

```
$mavproxy.py --master=udp:127.0.0.1:14550
MAV > mode guided
GUIDED > arm throttle
GUIDED > takeoff 12
```

Una vez alcanzada la altura de trabajo, el nodo *nav_rl* comienza a ejecutar los episodios en forma secuencial, a razón de 30 episodios por algoritmo. Al terminar cada episodio, el nodo se encarga de dirigir el dron hasta el punto de partida. Una vez allí, y sin aterrizar, se inicia el siguiente episodio.

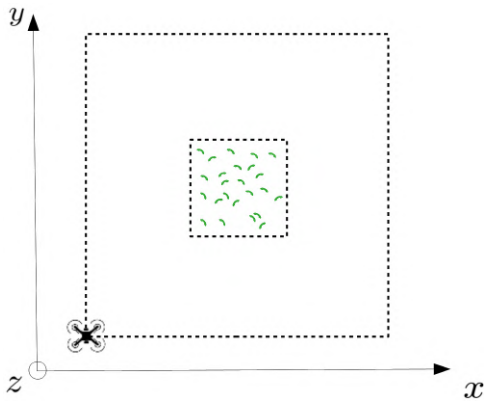



Figura 8.2: Configuración inicial de cada episodio. Los boids se colocan inicialmente dentro del cuadrado central.

### 8.2.2. Pruebas en el entorno de entrenamiento

La tabla 8.1 muestra las estadísticas de los tiempos de ejecución de los experimentos realizados en el entorno de entrenamiento para cada algoritmo, mientras que la figura 8.3 muestra los box plot correspondientes.

| | H | RL1 | RL2 |
|---|---|---|---|
| Min | 91.8 | 78.6 | 65.6 |
| Max | 100.4 | 91.6 | 79.8 |
| Media | 96.1 | 83.9 | 71.0 |
| Mediana | 95.9 | 83.5 | 70.6 |
| Std | 1.42 | 2.55 | 2.57 |
| Efectividad | 100 % | 100 % | 100 % |

Tabla 8.1: Estadísticas de los tiempos de ejecución, en segundos, de los 100 episodios de cada algoritmo

Un test ANOVA de los 3 grupos de datos muestra que las diferencias entre los tiempos medios de cada algoritmo son estadísticamente significativas. Un t-test realizado a cada par de grupos de datos muestra que las diferencias entre cada par de algoritmos también son estadísticamente significativas. Hay una clara mejora en los tiempos de $RL1$ con respecto a la heurística $H$, a la vez que también se observa una mejora sustancial al pasar de $RL1$ a $RL2$. De todas formas, es necesario contrastar estos resultados con los experimentos en Gazebo/ROS2, para obtener un panorama más realista del desempeño de la solución.

### 8.2.3. Pruebas en Gazebo/ROS2

La tabla 8.2 muestra las estadísticas de los tiempos de ejecución de los experimentos realizados en el entorno Gazebo/ROS2 para cada algoritmo, mientras que la figura 8.4 muestra los box plot correspondientes.

Lo primero que se observa es que los tiempos de ejecución son bastante menores a los registrados en el entorno previo. Esto simplemente se debe a que el espacio aéreo en Gazebo tiene la mitad de las dimensiones del espacio aéreo utilizado por el entorno

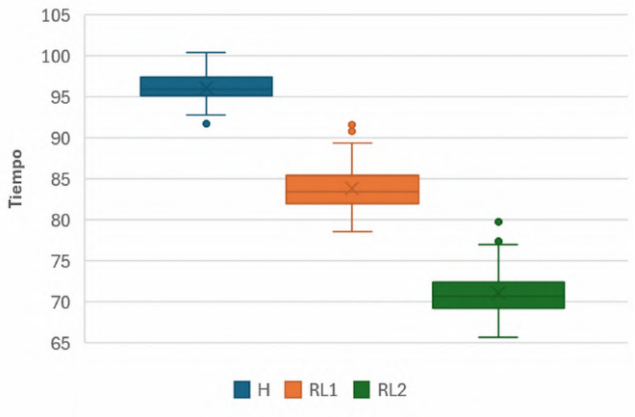



Figura 8.3: Box plot de los 100 experimentos en el entorno de entrenamiento, para los algoritmos $H, RL1$ y $RL2$.

| | H | RL1 | RL2 |
|---|---|---|---|
| Min | 44.66 | 36.34 | 37.4 |
| Max | 67.42 | 53.08 | 62.4 |
| Media | 52.55 | 44.49 | 43.02 |
| Mediana | 53.15 | 45.28 | 41.61 |
| Std | 5.57 | 4.20 | 4.89 |
| Efectividad | 100 % | 100 % | 100 % |

Tabla 8.2: Estadísticas de los tiempos de ejecución de los 100 episodios de cada algoritmo en el entorno de entrenamiento.

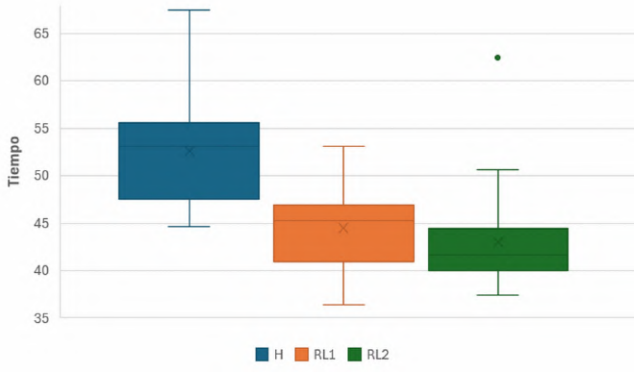



Figura 8.4: Box plot de los 30 experimentos en el entorno de Gazebo/ROS2, para los algoritmos $H, RL1$ y $RL2$.

de entrenamiento.

En cuanto al desempeño relativo de los algoritmos, se sigue observando una diferencia estadísticamente significativa entre la heurística $H$ y el algoritmo $RL1$. Por otro lado, si bien hay una leve mejora en los tiempos medios al pasar de $RL1$ a $RL2$, la diferencia ya no es estadísticamente significativa, de acuerdo al t-test aplicado a estos

dos grupos de datos.

Es necesario recordar que ambos entornos son muy diferentes. En el entorno de entrenamiento no se modela ninguna ley física, como la gravedad o la resistencia del aire. El dron, modelado como una partícula, puede moverse en forma precisa ante cambios bruscos de velocidad y solo está limitado por cotas máximas de aceleración y velocidad.

Por su parte, el dron en Gazebo necesita mucho más tiempo para completar un giro, además de que debe contrarrestar la gravedad en todo momento, por lo que su respuesta es muy diferente a la observada en el entorno de entrenamiento.

La red neuronal entrenada en dicho entorno aprende a explotar las características de un dron ideal que no existe en la realidad. Esto se observa en los tres algoritmos, pero el problema es más pronunciado en $RL2$, porque no cuenta con la asistencia de la heurística en la etapa de explotación.

### 8.2.4. Sesgo

Durante los experimentos se constató que el rendimiento del algoritmo $RL2$ se ve fuertemente afectado por la posición de partida del dron. Los resultados precedentes se obtuvieron colocando el dron en el punto de partida más favorable, esto es, la esquina inferior izquierda según la figura 8.3. A medida que la posición inicial del dron se aleja de esta esquina, sobre el perímetro del campo, el rendimiento del algoritmo se degrada considerablemente, y en algunos casos no logra expulsar la bandada en forma completa en el tiempo máximo asignado a cada episodio.

De acuerdo a lo observado, el sesgo de la red neuronal $rl2$ hace que el agente tienda a desplazar la bandada hacia un mismo sector del campo, por lo que la posición inicial más favorable del dron es el sector opuesto. La figura 8.5 muestra, en forma esquemática, lo que sucede cuando el dron parte de alguna de las 4 esquinas del campo.

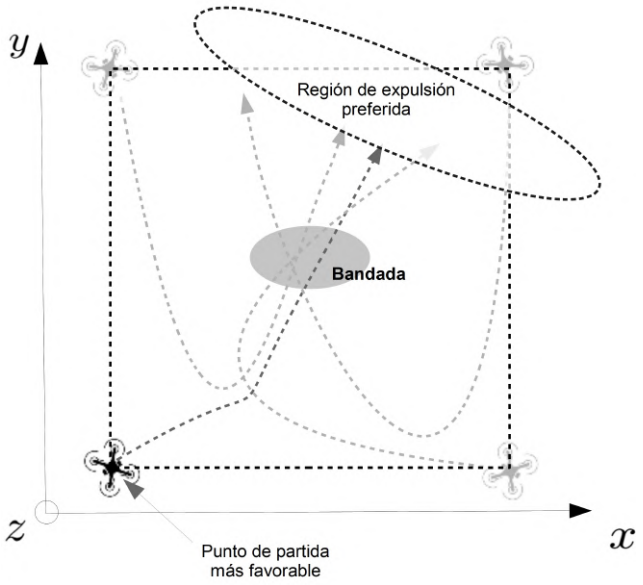



Figura 8.5: Sesgo del algoritmo $RL2$

Se especula que la razón pudo haber sido el uso de un sistema de coordenadas

global fijo durante el entrenamiento, en lugar de un sistema de coordenadas local al dron, en donde este se mantenga fijo en el origen, mientras que las aves se mueven a su alrededor.

Este cambio en el sistema de coordenadas debería eliminar cualquier preferencia en la dirección de movimiento del dron a la hora de desplazar la bandada.

## 8.3. Análisis de Sensibilidad

Para el análisis de sensibilidad se identificaron cuatro variables de interés que afectan el rendimiento de los algoritmos:

- $N_b$: Número de aves de la bandada.
- $w_p$: Peso relativo del comportamiento de *bounding*.
- $r_b$: Umbral de interacción entre boids.
- $\vec{v}_{wind}$: Dirección del viento.

Si bien hay muchos más parámetros que afectan tanto la conducta emergente de la bandada como la respuesta del dron, durante los experimentos se observó que las 4 variables listadas tienen un impacto mucho más significativo en el desempeño general de la solución.

Todos los experimentos de esta etapa se realizaron en Gazebo/ROS2 y, al igual que antes, se ejecutaron 30 episodios para cada combinación de parámetros, registrándose el tiempo requerido para expulsar completamente a la bandada. Cada una de las variables de interés tomó una serie de valores predeterminados, mientras que los demás parámetros se mantuvieron constantes e iguales a los establecidos en el apéndice A. En particular, cuando las variables de interés se mantienen constantes, sus valores son:

| Variable | Valor |
|---|---|
| $N_b$ | 25 |
| $w_p$ | 0.4 |
| $r_b$ | 4 m |
| $\vec{v}_{wind}$ | $\vec{0}$ |

Tabla 8.3: Valor de control de las variables de interés.

### 8.3.1. $N_b$: Número de aves

El número de aves $N_b$ que componen la bandada toma los valores: 25, 35, 50, 70, 100 y 200. La tabla 8.4 muestra las estadísticas de los tiempos de ejecución de 30 episodios lanzados para cada tamaño de bandada y para cada algoritmo ($H, RL1$ y $RL2$). Por su parte, la figura 8.6 muestra los diagramas box plot correspondientes a cada uno de los tres algoritmos, junto con una comparación de los tiempos medios a medida que aumenta el tamaño de la bandada. Todos los valores están expresados en segundos.

El rendimiento de los 3 algoritmos no parece verse afectado en forma significativa al aumentar la cantidad de aves en la bandada. Todos los tiempos medios se mueven dentro del rango que va de 35 a 60 segundos.

| Algoritmo H | | | | | | |
|---|---|---|---|---|---|---|
| $N_b$ | Min | Max | Media | Mediana | Std | Efectividad |
| 25 | 43.42 | 48.51 | 45.23 | 44.9 | 1.34 | 100 % |
| 35 | 50.36 | 66.68 | 56.34 | 55.58 | 3.51 | 100 % |
| 50 | 52.11 | 66.66 | 58.99 | 58.87 | 3.29 | 100 % |
| 70 | 50.39 | 60.77 | 54.07 | 53.41 | 2.83 | 100 % |
| 100 | 49.64 | 64.79 | 53.05 | 52.11 | 3.30 | 100 % |
| 200 | 50.85 | 74.65 | 56.47 | 53.77 | 5.70 | 100 % |
| Algoritmo RL1 | | | | | | |
| $N_b$ | Min | Max | Media | Mediana | Std | Efectividad |
| 25 | 34.22 | 39.93 | 37.24 | 37.26 | 1.36 | 100 % |
| 35 | 42.54 | 59.21 | 49.66 | 48.84 | 4.10 | 100 % |
| 50 | 44.21 | 59.52 | 53.72 | 54.34 | 3.67 | 100 % |
| 70 | 42.57 | 59.66 | 49.59 | 47.95 | 5.08 | 100 % |
| 100 | 42.62 | 52.79 | 46.65 | 46.70 | 2.56 | 100 % |
| 200 | 43.64 | 64.23 | 49.71 | 48.02 | 5.11 | 100 % |
| Algoritmo RL2 | | | | | | |
| $N_b$ | Min | Max | Media | Mediana | Std | Efectividad |
| 25 | 33.31 | 39.84 | 35.68 | 35.33 | 1.75 | 100 % |
| 35 | 35.13 | 58.84 | 40.05 | 39.15 | 4.53 | 100 % |
| 50 | 37.77 | 90.91 | 45.89 | 40.40 | 12.64 | 100 % |
| 70 | 36.01 | 57.97 | 41.55 | 40.06 | 4.92 | 100 % |
| 100 | 36.38 | 97.14 | 45.01 | 39.69 | 13.22 | 100 % |
| 200 | 37.22 | 124.45 | 53.81 | 44.30 | 21.69 | **80 %** |

Tabla 8.4: Estadísticas de los 30 episodios ejecutados de cada algoritmo, para diferentes tamaños de bandada. Todos los valores están expresados en segundos. Las filas resaltadas señalan efectividad $< 100\,\%$.

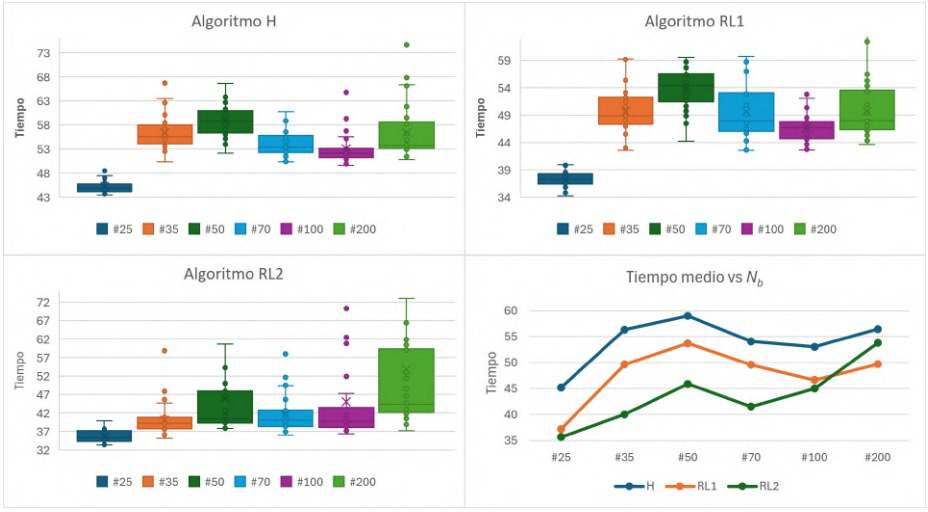



Figura 8.6: Box plot de los tiempos de ejecución de los episodios para cada tamaño de bandada. En la esquina inferior derecha se comparan los tiempos medios de cada algoritmo, a medida que aumenta el número de aves.

Por otro lado, se mantiene la relación entre los algoritmos, observándose una mejora de $RL1$ con respecto a la heurística $H$, y una mejora de $RL2$ con respecto a $RL1$. Sin embargo, $RL2$ resulta ser más sensible al tamaño de la bandada y su rendimiento se deteriora rápidamente más allá de las 70 aves. A partir de las 100 aves, $RL2$ comienza a fallar en algunos episodios, mientras que los otros dos algoritmos logran expulsar la bandada en todos los casos. También, la desviación estándar registrada por $RL2$ aumenta considerablemente con bandadas de 100 o más aves.

Esto probablemente se deba a que el algoritmo $RL2$ se ha entrenado con bandadas de solo 25 aves, y no pudo generalizar una solución con tamaños mayores. Aún así, su desempeño sigue siendo superior al de los otros dos algoritmos cuando la bandada tiene menos de 70 individuos.

Por otro lado, se observa un fenómeno inesperado: los tiempos de los 3 algoritmos se incrementan cuando crece el número de aves, pero luego decrecen transitoriamente cuando el tamaño de la bandada pasa de 50 a 70, para luego volver a deteriorarse más allá de las 100 aves. La causa de que esto ocurra no está del todo clara. Una simple comparación visual de episodios con diferentes tamaños de bandada sugiere que hay una densidad de aves crítica a partir de la cual todo el grupo se vuelve más compacto, ocupando así un área total menor al área que ocuparía una bandada con menos aves. Esto ocurre en algún punto entre las 50 y 70 aves. Cuando la bandada ocupa un área más reducida, la oscilación del dron entre los diferentes flancos es menos pronunciada, lo cual tiene como consecuencia que se tarde menos tiempo en desplazar a todo el grupo fuera del espacio aéreo.

Es posible que esto pueda estar relacionado con el umbral de distancia de interacción entre las aves, $r_b$. Al haber una mayor cantidad de individuos por unidad de área (mayor densidad), estos se encuentran más cerca unos de otros, incrementando así las probabilidades de hallarse a una distancia menor a $r_b$. Esto, a su vez, hace que los comportamientos de cohesión y alineamiento se activen con mayor frecuencia, provocando que todo el grupo se compacte y ocupe un área total más reducida. Sin embargo, esto ocurre hasta cierto límite. Con una bandada de más de 100 individuos, el área ocupada comienza a incrementarse nuevamente, lo que provoca que los tiempos de ejecución vuelvan a deteriorarse. En cualquier caso, para poder comprobar lo que está ocurriendo realmente, es necesario realizar más experimentos para extraer estadísticas acerca de la activación de los comportamientos con diferentes tamaños de bandada.

### 8.3.2. $w_p$: Bounding

El peso relativo del *bounding* o protección periférica, $w_p$, toma los valores: 0.25, 0.3, 0.4 y 0.5. Cabe recordar que se impone la restricción $w_p > w_f = 0.2$, lo que significa que las aves prefieren agregarse por protección frente a la opción de detenerse en un árbol para alimentarse ($w_f$), especialmente cuando detectan la presencia de un depredador.

La tabla 8.5 muestra las estadísticas de los tiempos de ejecución de 30 episodios lanzados para cada valor $w_p$ y para cada algoritmo ($H, RL1$ y $RL2$). Por su parte, la figura 8.7 muestra los diagramas box plot correspondientes a cada uno de los tres algoritmos, junto con una comparación de los tiempos medios a medida que aumenta el valor de $w_p$.

| Algoritmo H | | | | | | |
|---|---|---|---|---|---|---|
| $w_p$ | Min | Max | Media | Mediana | Std | Efectividad |
| 0.25 | 51.03 | 100.00 | 70.58 | 70.03 | 14.67 | 100 % |
| 0.3 | 44.58 | 82.72 | 62.35 | 63.22 | 11.83 | 100 % |

| | | | | | | |
|---|---|---|---|---|---|---|
| 0.4 | 44.66 | 67.42 | 52.55 | 53.15 | 5.56 | 100 % |
| 0.5 | 43.14 | 56.14 | 49.37 | 49.53 | 4.15 | 100 % |
| Algoritmo RL1 | | | | | | |
| $w_p$ | Min | Max | Media | Mediana | Std | Efectividad |
| 0.25 | 48.09 | 81.58 | 62.35 | 62.25 | 8.10 | 100 % |
| 0.3 | 43.32 | 66.11 | 53.54 | 54.24 | 4.87 | 100 % |
| 0.4 | 36.34 | 53.08 | 44.49 | 45.28 | 4.20 | 100 % |
| 0.5 | 35.20 | 48.74 | 41.62 | 41.66 | 3.03 | 100 % |
| Algoritmo RL2 | | | | | | |
| $w_p$ | Min | Max | Media | Mediana | Std | Efectividad |
| 0.25 | 100.00 | 100.00 | 100.00 | 100.00 | 0.00 | **0 %** |
| 0.3 | 44.16 | 100.00 | 71.03 | 71.00 | 15.32 | **90 %** |
| 0.4 | 37.40 | 62.40 | 43.02 | 41.61 | 4.89 | 100 % |
| 0.5 | 36.44 | 48.77 | 41.00 | 40.40 | 3.06 | 100 % |

Tabla 8.5: Estadísticas de los 30 episodios ejecutados de cada algoritmo, para diferentes valores de $w_p$. Todos los valores están expresados en segundos. Las filas resaltadas señalan efectividad $<$ 100 %.

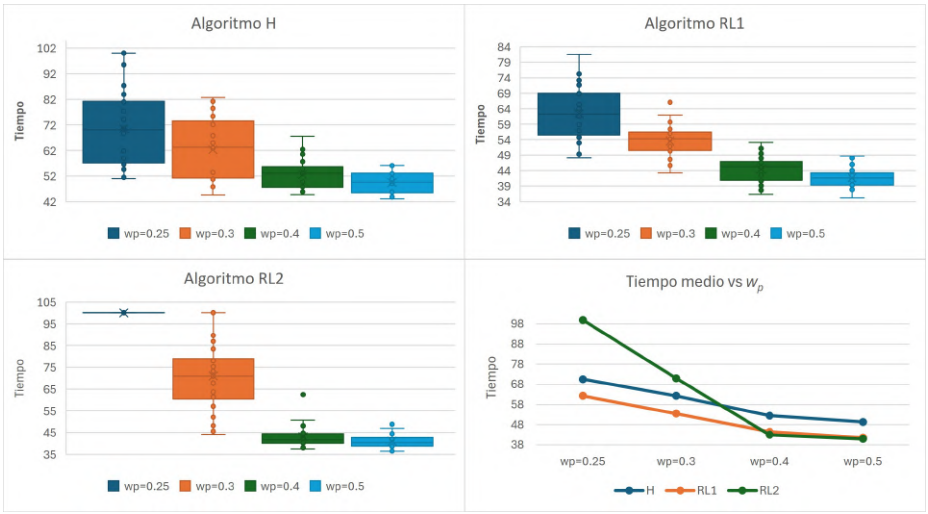



Figura 8.7: Box plot de los tiempos de ejecución de los episodios para cada valor de *bounding*, $w_p$. En la esquina inferior derecha se comparan los tiempos medios de cada algoritmo, a medida que aumenta el valor de $w_p$.

Los gráficos box plot muestran un marcado deterioro del rendimiento de los tres algoritmos a medida que el peso relativo del *bounding* disminuye y se acerca al valor $w_f$, que en este caso es 0.2. El algoritmo más sensible a este parámetro es $RL2$, que falla el 100 % de las veces cuando $w_p = 0.25$, mientras que los otros dos algoritmos logran expulsar la bandada por completo, aunque tardan más que con valores de *bounding* mayores.

Para el entrenamiento de la red del algoritmo $RL2$, se utilizó el valor $w_p = 0.4$ el cual se mantuvo fijo durante todo el proceso. Por esta razón, cabe hacer la misma reflexión que en el caso del número de aves: la red neuronal no está generalizando bien la solución para aves con $w_p < 0.4$. Cuando $w_p \geq 0.4$, la bandada tiende a agregarse y

mantenerse unida a lo largo del tiempo, aún ante el avance del dron. Esto hace que la persecución de los flancos izquierdo y derecho no sea tan necesaria, resultando en una trayectoria más rectilínea hacia el centro de la bandada. Sin embargo, cuando $w_p < 0.4$, esta estrategia provoca que la bandada se disperse y las aves acaban por formar un círculo alrededor del dron, el cual pasa a convertirse en el centro geométrico de toda la bandada. En esta situación, la red neuronal no es capaz de continuar desplazando las aves hacia el exterior y el dron se queda volando en círculos. Este problema no ocurre con la heurística $H$ y el algoritmo $RL1$, porque en esos casos siempre se persiguen los flancos, evitando así que el dron quede rodeado por las aves.

### 8.3.3. $\vec{v}_{wind}$: Viento

En general, se sugiere evitar volar drones en condiciones atmosféricas adversas. En particular, para principiantes se recomienda volar con vientos menores a 15 mph, equivalente a unos 24 km/h, mientras que operadores expertos pueden operar fácilmente con vientos de hasta 20 mph [42].

Sin embargo, en las pruebas hechas en Gazebo, el modelo del dron 3DR Iris no pudo sostener su posición frente a vientos superiores a los 15 km/h, por lo que las pruebas de viento se hicieron utilizando un valor algo menor, alrededor de 14.5 km/h $\approx$ 4m/s.

Para la simulación del viento, se activó el plugin de Gazebo *WindEffects* [1], el cual permite simular el efecto viento con una velocidad lineal específica. Esta velocidad se indica como una terna $x, y, z$ en el archivo SDF que define el mundo simulado, bajo la etiqueta **world**:

```
<world name="mundo">
    <wind>
        <linear_velocity>x y z</linear_velocity>
    </wind>
</world>
```

En los experimentos, se utilizaron 4 direcciones de viento diferentes, correspondientes a los cuatro puntos cardinales, según la tabla 8.6 y la figura 8.8.

| Dirección | $\vec{v}_{wind}$ |
|---|---|
| Norte | (0, -4, 0) |
| Sur | (0, 4, 0) |
| Este | (-4, 0, 0) |
| Oeste | (4, 0, 0) |
| $\vec{0}$ | (0, 0, 0) |

Tabla 8.6: Velocidades de viento utilizadas. Valores expresados en m/s

[1] Basado en la clase «*gz::sim::systems::WindEffects*»

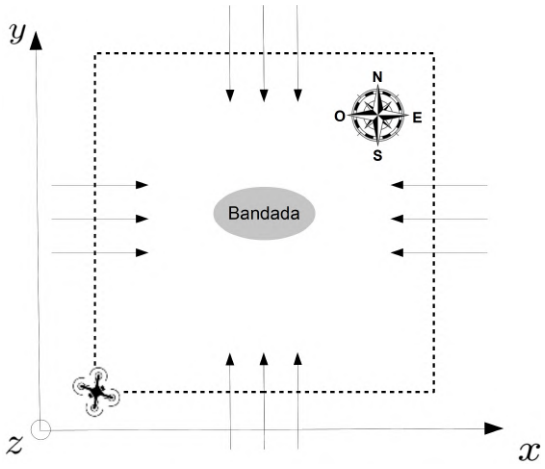



Figura 8.8: Direcciones de viento provenientes de los 4 puntos cardinales.

Se ejecutaron 30 episodios con viento norte, sur, este, oeste y sin viento. La tabla 8.7 muestra las estadísticas de los tiempos de ejecución para cada configuración de viento y algoritmo.

| Algoritmo H | | | | | | |
|---|---|---|---|---|---|---|
| $\vec{v}_{wind}$ | Min | Max | Media | Mediana | Std | Efectividad |
| Este | 74.52 | 95.35 | 82.10 | 82.02 | 4.59 | 100 % |
| Oeste | 51.73 | 69.58 | 58.88 | 58.44 | 3.91 | 100 % |
| Sur | 51.04 | 67.78 | 59.41 | 59.24 | 3.54 | 100 % |
| Norte | 73.40 | 92.87 | 81.84 | 80.80 | 4.93 | 100 % |
| $\vec{0}$ | 43.42 | 48.51 | 45.23 | 44.9 | 1.34 | 100 % |
| Algoritmo RL1 | | | | | | |
| $\vec{v}_{wind}$ | Min | Max | Media | Mediana | Std | Efectividad |
| Este | 66.43 | 106.91 | 76.72 | 74.01 | 9.88 | 100 % |
| Oeste | 42.89 | 59.76 | 50.08 | 49.27 | 3.88 | 100 % |
| Sur | 44.48 | 60.02 | 51.86 | 51.70 | 4.20 | 100 % |
| Norte | 63.81 | 86.87 | 75.03 | 73.95 | 6.51 | 100 % |
| $\vec{0}$ | 34.22 | 39.93 | 37.24 | 37.26 | 1.36 | 100 % |
| Algoritmo RL2 | | | | | | |
| $\vec{v}_{wind}$ | Min | Max | Media | Mediana | Std | Efectividad |
| Este | 58.27 | 150.03 | 84.04 | 71.52 | 29.22 | **97 %** |
| Oeste | 34.78 | 50.53 | 41.07 | 40.14 | 3.98 | 100 % |
| Sur | 36.61 | 98.41 | 47.28 | 45.11 | 11.31 | 100 % |
| Norte | 77.13 | 108.89 | 87.04 | 85.55 | 7.67 | **83 %** |
| $\vec{0}$ | 33.31 | 39.84 | 35.68 | 35.33 | 1.75 | 100 % |

Tabla 8.7: Estadísticas de los 30 episodios ejecutados de cada algoritmo, para las diferentes direcciones de viento. Las filas resaltadas señalan efectividad $< 100\,\%$.

De acuerdo a los box plot de la figura 8.9, los tres algoritmos reducen su rendimiento ante la presencia de viento, como era de esperarse. Sin embargo, todos logran completar la tarea de expulsar a la bandada, excepto el algoritmo $RL2$, que falló en 3 episodios.

Se observa, además, que todos los algoritmos funcionan mejor con viento a favor,

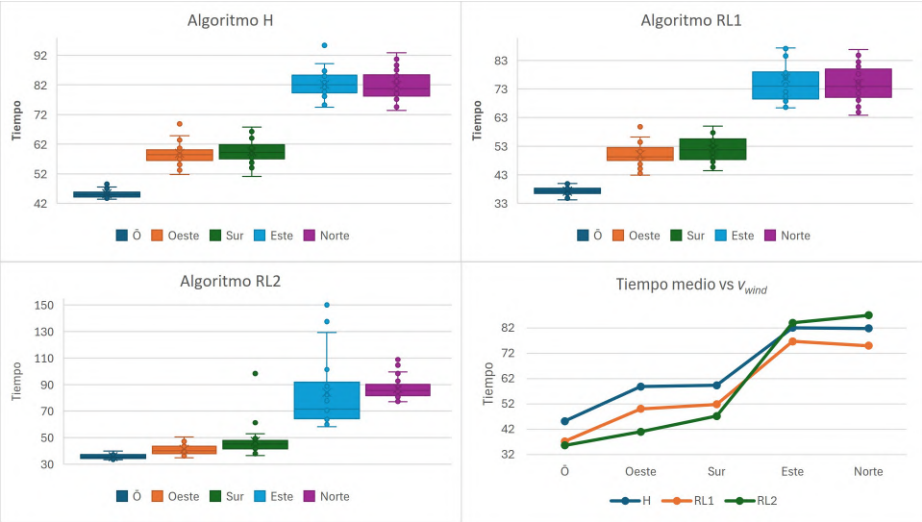



Figura 8.9: Box plot de los tiempos de ejecución de los episodios para cada dirección del viento. En la esquina inferior derecha se comparan los tiempos medios de cada algoritmo para cada dirección de viento.

es decir, viento sur y oeste, según el diagrama de la figura 8.8, y empeoran significativamente con viento en contra.

En el gráfico comparativo de tiempos medios, se puede apreciar que $RL1$ sigue teniendo un mejor rendimiento que $H$ en las 5 configuraciones y, a su vez, $RL2$ supera a $RL1$ cuando cuenta con viento a favor. Sin embargo, $RL2$ es el algoritmo más afectado cuando hay viento en contra, resultando ser el de peor rendimiento con viento este y norte.

Similar a lo que sucedía en el caso del *bounding*, $RL2$ es el más sensible a las condiciones de viento. Nuevamente, la red neuronal no logra generalizar bien en estas condiciones, principalmente porque no estuvo expuesta a ellas.

### 8.3.4. $r_b$: Distancia de interacción entre aves

La distancia de interacción entre aves, $r_b$, toma los valores: 1.5, 2.5, 4.0, 5.0 y 6.0 metros. La tabla 8.8 muestra las estadísticas de los tiempos de ejecución de 30 episodios lanzados para cada valor de $r_b$ y para cada algoritmo ($H, RL1$ y $RL2$). Por su parte, la figura 8.10 muestra los diagramas box plot correspondientes a cada uno de los tres algoritmos, junto con una comparación de los tiempos medios a medida que aumenta el valor de $r_b$. Todos los tiempos están expresados en segundos.

| Algoritmo H | | | | | | |
|---|---|---|---|---|---|---|
| $r_b$ | Min | Max | Media | Mediana | Std | Efectividad |
| 1.5 | 38.53 | 40.78 | 39.75 | 39.68 | 0.58 | 100 % |
| 2.5 | 43.42 | 48.51 | 45.23 | 44.91 | 1.34 | 100 % |
| 4.0 | 50.00 | 69.82 | 57.23 | 56.75 | 3.97 | 100 % |
| 5.0 | 59.74 | 150.01 | 80.63 | 72.32 | 21.90 | **97 %** |
| 6.0 | 71.28 | 150.01 | 114.63 | 112.46 | 27.48 | **73 %** |
| Algoritmo RL1 | | | | | | |

| $r_b$ | Min | Max | Media | Mediana | Std | Efectividad |
|---|---|---|---|---|---|---|
| 1.5 | 29.38 | 35.42 | 32.61 | 32.61 | 1.57 | 100 % |
| 2.5 | 34.22 | 39.93 | 37.24 | 37.27 | 1.36 | 100 % |
| 4.0 | 43.00 | 54.84 | 48.90 | 48.91 | 2.83 | 100 % |
| 5.0 | 54.50 | 83.02 | 63.40 | 59.70 | 8.60 | 100 % |
| 6.0 | 61.28 | 150.00 | 82.30 | 74.91 | 24.37 | **93 %** |
| Algoritmo RL2 | | | | | | |
| $r_b$ | Min | Max | Media | Mediana | Std | Efectividad |
| 1.5 | 32.86 | 35.95 | 34.28 | 34.16 | 0.75 | 100 % |
| 2.5 | 33.31 | 39.84 | 35.68 | 35.33 | 1.75 | 100 % |
| 4.0 | 35.13 | 102.77 | 47.28 | 44.85 | 12.68 | 100 % |
| 5.0 | 53.80 | 150.01 | 79.95 | 68.02 | 28.16 | **87 %** |
| 6.0 | 59.69 | 150.03 | 127.99 | 132.77 | 23.29 | **57 %** |

Tabla 8.8: Estadísticas de los 30 episodios ejecutados de cada algoritmo, para las diferentes valores de $r_b$. Las filas resaltadas señalan efectividad $< 100\,\%$.

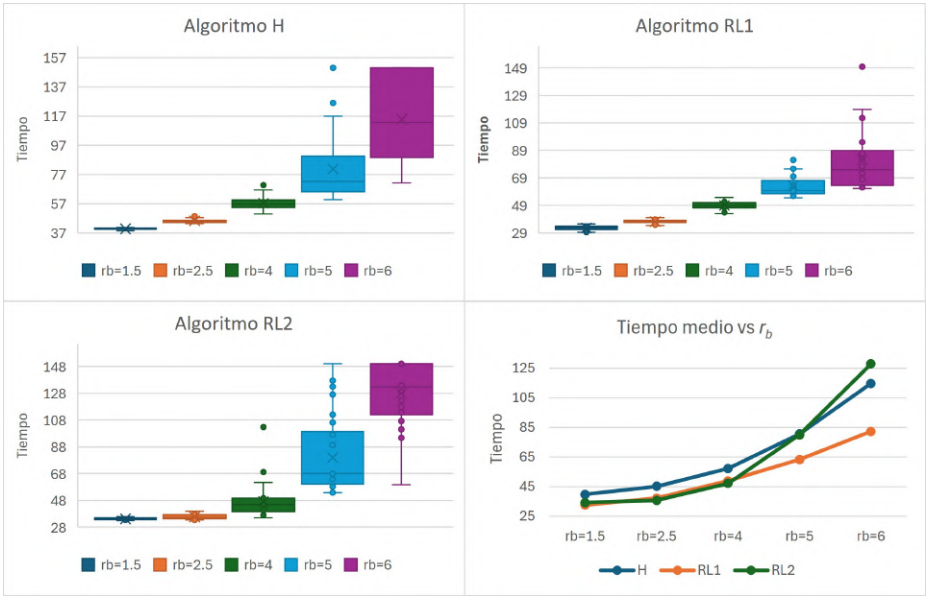



Figura 8.10: Box plot de los tiempos de ejecución de los episodios para cada valor de $r_b$. En la esquina inferior derecha se comparan los tiempos medios de cada algoritmo, a medida que aumenta la distancia entre las aves.

El rendimiento de los tres algoritmos se ve fuertemente impactado por este parámetro. Los tiempos de ejecución se deterioran rápidamente al aumentar el valor de $r_b$ desde 1.5 hasta 6 m.

El parámetro $r_b$ está estrechamente relacionado con el comportamiento de *bounding*. El valor de este parámetro determina la capacidad de cada ave de detectar vecinos. Al reducirse, el ave queda aislada con más frecuencia y, por tanto, también aumenta la frecuencia con la cual se activa el comportamiento de *bounding*. Esto no solo ocurre en la periferia de la bandada, sino también en el interior de la misma si

el valor $r_b$ es muy pequeño. Como consecuencia, la bandada se vuelve cada vez más compacta y es más fácil para el dron dirigirla hacia afuera.

Según el gráfico comparativo de tiempos medios vs $r_b$ de la figura 8.10, los algoritmos $RL1$ y $RL2$ tienen rendimientos muy similares entre sí, pero superan a la heurística $H$ cuando $r_b \leq 4$. Por encima de ese valor, $RL1$ sigue siendo superior a $H$, pero los tiempos de $RL2$ se disparan y acaba por ser el peor algoritmo cuando $r_b = 6$.

Nuevamente, la red que utiliza $RL2$ no está preparada para lidiar con grupos de aves que tienden a dispersarse, por haber sido entrenada con una bandada que se mantiene relativamente compacta durante todo el episodio, aún frente al avance del dron.

# Capítulo 9

# Conclusiones

En la presente tesis de maestría se exploró el uso de un vehículo aéreo autónomo con el fin de disuadir y expulsar aves plaga de áreas con cultivos de interés comercial. Este es un problema que ocasiona importantes pérdidas económicas, no solo en Uruguay, sino también a nivel mundial.

Proteger los cultivos del ataque de las aves no es un problema reciente, lo que ha dado lugar al uso de una variedad de métodos de control de plagas que ha ido evolucionando a lo largo del tiempo [60]. El avance de las tecnologías de la información y la robótica en los últimos años ha generado un interés creciente de los investigadores en la aplicación de estas soluciones al sector agrícola en general, y a la protección de cultivos en particular. En este último apartado, el estudio de la factibilidad del uso de drones autónomos para la disuasión de aves resulta de particular interés.

Por esta razón, se inició el trabajo realizando un relevamiento de los estudios previos relacionados con esta temática. En dichos estudios, se evaluó, por ejemplo, la respuesta de las aves ante la presencia de un dron mediante la realización de pruebas de campo con drones reales controlados manualmente por un operador [79, 80]. Estos experimentos mostraron resultados prometedores, aunque no se pudo evaluar el efecto a mediano y largo plazo de la solución. En otro trabajo se abordó la automatización del vehículo para evaluar la efectividad del sistema a la hora de reconocer aves invasoras, así como también su capacidad para ejecutar misiones de patrullaje sobre el área protegida [9]. Aquí también se obtuvieron resultados prometedores, aunque algo limitados, debido a ciertas carencias tecnológicas del hardware utilizado. En otra serie de estudios se hizo foco en el modelado del problema, en donde se simuló tanto el dron como una bandada de aves genérica [76, 81]. Si bien los resultados de estas simulaciones fueron más que aceptables, los mismos deben tomarse con cautela debido a que la simplicidad del modelo de bandada utilizado puede dar lugar a resultados poco realistas.

Otros trabajos analizados presentan modelos más refinados de bandada, todos ellos basados en el concepto de boid elaborado originalmente por Greig Reynolds en 1987, el cual utiliza la idea de comportamientos individuales que dan lugar a una conducta grupal emergente [36, 37, 58]. Sin embargo, estos estudios no abordan el problema de la disuasión, sino que se enfocan en desarrollar modelos de bandada más precisos que se ajusten mejor a lo observado en la naturaleza.

Finalmente, se analizaron 2 trabajos que abordan un problema muy similar al de la disuasión de aves: el desplazamiento de un rebaño de ovejas hacia un punto objetivo, dirigido por un perro pastor [69, 85]. En dichos trabajos se presentan dos estrategias muy diferentes. Por un lado, una heurística simple de conducción y agrupación, y por

otro, la utilización de técnicas de aprendizaje por recompensas, o RL.

De todos los estudios analizados se extrajeron varias ideas para desarrollar los modelos y algoritmos utilizados en esta tesis. El siguiente paso fue la modelización del espacio aéreo como una grilla de ocupación tridimensional, cuyas celdas indican la presencia de aves. Luego, se procedió a la especificación del modelo de bandada a utilizar, basado en el concepto de boids de Reynolds, junto con un modelo simplificado de dron que permitió realizar simulaciones con un costo computacional relativamente bajo.

A continuación se desarrolló la heurística «Persecución de Flancos», codificada como $H$, que a partir de la grilla de ocupación, determina una serie de puntos o flancos alrededor de la bandada. Mediante la persecución alternada de estos flancos por parte del dron, se logra desplazar gradualmente a toda la bandada hasta lograr expulsarla por completo del espacio aéreo.

Luego se exploró la utilización de técnicas de aprendizaje por recompensas, mediante el uso del algoritmo *DDPG* (*Deep Deterministic Policy Gradient*), para entrenar una red neuronal que controle el dron, en lugar de usar la heurística $H$. Este intento resultó infructuoso debido a que la red entrenada no era capaz de resolver el problema, y cuando lo hacía, tardaba mucho más tiempo que la heurística. Si bien no se pudo determinar la causa exacta de este fallo, se especula que la naturaleza dispersa de las recompensas utilizadas hace que el algoritmo *DDPG* no logre converger en un tiempo razonable. También es posible que caiga en mínimos locales de los cuales no es capaz de salir, debido a la complejidad de la función de pérdida de este problema.

A partir de este resultado, se investigó la opción de combinar RL con la heurística $H$, desarrollándose así dos algoritmos: $RL1$ y $RL2$. $RL1$ utiliza la heurística $H$ para determinar el flanco a perseguir en cada momento y luego dirige el dron hacia dicho flanco, utilizando una red neuronal preentrenada para mover el vehículo hacia un punto objetivo dado.

$RL2$, por su parte, también hace uso de una red neuronal para controlar el dron, pero a diferencia de $RL1$, no se determina ningún flanco a perseguir. En cambio, la heurística $H$ se utiliza durante el entrenamiento, añadiendo un sesgo a la recompensa para que el algoritmo priorice las decisiones tomadas por la heurística, pero que a la vez sea capaz de explorar alternativas potencialmente superiores. La tabla 9.1 resume el uso que hace cada algoritmo de la heurística $H$ para la determinación de flancos, y en qué etapa se da esta utilización, ya sea entrenamiento o explotación.

| Algoritmo | Entrenamiento | Explotación |
|---|---|---|
| $RL1$ | No | Si |
| $RL2$ | Si | No |

Tabla 9.1: Etapas (entrenamiento/explotación) de cada algoritmo en las que se utiliza la heurística $H$ para determinar el flanco a atacar.

Como resultado, al combinar la heurística $H$ con el algoritmo *DDPG* se observaron rendimientos superiores en comparación con la utilización de estas dos técnicas por separado. La métrica utilizada para medir el rendimiento fue el tiempo que tarda el dron en expulsar a la bandada del espacio aéreo. Con esta métrica, resultó que el algoritmo $RL1$ supera ampliamente a la heurística $H$, mientras que $RL2$ supera a su vez a $RL1$, siendo estas diferencias estadísticamente significativas. Es importante hacer notar que estos resultados se observan en el entorno de entrenamiento, en donde se modela un dron ideal, solamente sujeto a cotas máximas de velocidad y aceleración.

Sin embargo, estos rendimientos dependen de la posición de partida del dron, siendo una de las esquinas del campo cultivado la posición óptima. A medida que la posición inicial del dron se aleja de esta ubicación óptima, el rendimiento del algoritmo

$RL2$ se degrada rápidamente, y en menor medida también el rendimiento de $RL1$. Este sesgo probablemente se deba a que las posiciones del dron y la bandada que recibe la red neuronal están expresadas en un sistema de coordenadas global, en lugar de un sistema local al dron, en donde este se encuentre fijo en el origen.

Para validar los algoritmos desarrollados en condiciones más realistas, los mismos fueron adaptados para ser aplicados a un entorno de simulación Gazebo controlado a través de ROS2. El modelo de dron utilizado en dicho entorno se basa en el dron real 3DR Iris, de la compañía 3DRobotic, el cual es controlado por un nodo ROS2 de Ardupilot. Se utilizó, además, un controlador proporcional para reducir el error entre la velocidad instantánea del dron medida por los sensores y la velocidad determinada por las redes neuronales o la heurística $H$, según el caso.

Los experimentos realizados en este entorno corroboraron la mejora en rendimiento que muestra $RL1$ con respecto a la heurística $H$. Sin embargo, $RL2$ no logra superar a $RL1$ como ocurría en el entorno de entrenamiento, sino que su rendimiento es muy similar al de $RL1$. Esto se debe a que el entorno Gazebo es muy diferente al entorno de entrenamiento, en donde, por ejemplo, no se modelan fuerzas físicas como la gravedad, resistencia del aire o la sustentación de los rotores. Estas simplificaciones hacen que la red neuronal entrenada en este entorno aprenda a explotar características ideales que no existen en Gazebo ni tampoco en la realidad.

Como parte de la experimentación, se llevó a cabo un análisis de sensibilidad de 4 variables de interés: número de boids que componen la bandada, peso relativo del comportamiento de bounding, distancia de interacción entre boids y el efecto del viento proveniente de los cuatro puntos cardinales.

El número de boids que componen la bandada no afectó significativamente el rendimiento de los algoritmos, observándose un leve deterioro progresivo a medida que la bandada crece en tamaño. Por otro lado, a partir de los 50 boids, la bandada se vuelve más compacta, lo que, a su vez, resulta en una mejora general del rendimiento de todos los algoritmos. A partir de los 70 boids, el rendimiento vuelve a deteriorarse nuevamente, siendo $RL2$ el más afectado. De todos modos, es probable que este fenómeno sea producto del modelo de bandada utilizado y no tenga su contraparte en la naturaleza.

El peso relativo del bounding, o protección periférica, y la distancia de interacción entre boids resultaron tener un impacto importante en el desempeño de los 3 algoritmos. Cuando el peso del bounding decrece o cuando la distancia de interacción entre boids se incrementa, la bandada se vuelve menos compacta, haciendo que los algoritmos tarden más tiempo en expulsarla. Tanto la heurística $H$ como $RL1$ comienzan a oscilar con más frecuencia entre un flanco y otro a medida que la bandada se dispersa. Por su parte, $RL2$ es nuevamente el más afectado. Debido a que casi no oscila entre los flancos, acaba dispersando la bandada, la cual forma un círculo a su alrededor y a partir de allí el dron ya no es capaz de continuar desplazándola.

Finalmente, el efecto del viento depende de la dirección desde donde éste provenga. Cuando el dron tiene viento a favor, su desempeño mejora, logrando desplazar a la bandada en menos tiempo en comparación con la situación de tener viento en contra. Nuevamente, $RL2$ resulta ser el algoritmo más afectado cuando la dirección del viento no es favorable. Cabe mencionar que el viento solo afecta al dron y no a los boids, los cuales son solo marcadores de posición.

Luego de haber analizado el desempeño general de los algoritmos, y cómo éstos se ven afectados al variar ciertos parámetros del entorno, se puede concluir que la heurística $H$ y $RL1$ resultan ser algoritmos bastante robustos a la hora de cumplir el objetivo de expulsar la bandada en su totalidad. Si bien $RL1$ utiliza la heurística para seleccionar los flancos a perseguir, su red neuronal resulta ser más eficiente que la heurística al momento de dirigir el dron hacia dichos flancos.

Por su parte, $RL2$ muestra un mejor desempeño que $RL1$ sólo cuando la bandada se mantiene relativamente compacta, por ejemplo, con un alto valor de $w_p(bounding)$, o con un bajo valor de $r_b$ (distancia entre los boids). Si cualquiera de estas condiciones se relaja, el rendimiento del algoritmo se degrada rápidamente, mientras que $RL1$ y la heurística $H$ continúan funcionando razonablemente bien.

Como comentario final, cabe destacar que el enfoque utilizado en el presente trabajo para abordar el problema de las aves plaga, parece ser prometedor. Ya sea que se trate de la heurística «Persecución de Flancos» o su combinación con técnicas de RL, los algoritmos propuestos resultan ser efectivos cuando se enfrentan a un modelo de bandada que es bastante más realista que los modelos utilizados en trabajos previos.

## 9.1. Trabajos a futuro

En cuanto a posibles mejoras y trabajos a futuro, se identifican varios puntos que quedaron fuera del alcance de la presente tesis, y que sería interesante poder explorar.

En primer lugar, mejorar la simulación de la bandada incorporando un modelo aerodinámico como el propuesto en [37], el cual tome en consideración fuerzas como la gravedad, sustentación de las alas y resistencia del aire. Esto, claro está, añade complejidad al modelo al introducir más parámetros que deben ser ajustados. Idealmente, se deberían ajustar partiendo de observaciones reales de bandadas que respondan a la presencia de un dron teleoperado.

En segundo lugar, se considera importante reemplazar el entorno de entrenamiento utilizado en este trabajo, en el cual se modela un dron ideal y muy simplificado que se aleja bastante del modelo en Gazebo y aún más de la realidad. Una posibilidad es utilizar el entorno de simulación de drones *gym-pybullet-drones* [55], el cual extiende la plataforma de simulación *pybullet* [18] para incorporar drones y algoritmos de RL. Otra opción interesante es *Omnidrones* [84] que se basa en el simulador *Omniverse Isaac Sim* [54] de *Nvidia*. Estos simuladores no solo incorporan modelos de drones más realistas, sino que también están integrados con librerías de RL, que incluyen implementaciones de los algoritmos más populares como DDPG, TD3, PPO o SAC. A su vez, ofrecen la opción de entrenar múltiples agentes en paralelo, lo que acelera en gran medida los tiempos de entrenamiento.

Otra mejora sugerida es la de simular la bandada directamente en una de estas plataformas de entrenamiento[1], en lugar de utilizar un código auxiliar totalmente desacoplado del entorno de simulación principal. De esta forma, los boids se verán afectados por las mismas fuerzas aerodinámicas que actúan sobre el dron.

Con respecto a la estrategia de entrenamiento, se sugiere utilizar bandadas con diferentes características en cada episodio, por ejemplo, variando el número de aves, la distancia de interacción entre las mismas ($r_b$) o el peso relativo del *bounding* ($w_p$), con el fin de lograr una política que generalice mejor la solución.

Como se mencionó en el capítulo 8, las redes neuronales entrenadas mostraron un sesgo importante con respecto a la posición de partida del dron. Es muy probable que esto se deba al uso de un sistema de coordenadas global fijo, por lo que se sugiere utilizar un sistema de coordenadas local al dron, en donde éste se encuentre inmóvil en el origen. Esto debería eliminar cualquier preferencia de las redes a la hora de elegir una dirección de movimiento.

En cuanto a las técnicas de RL utilizadas, se sugiere trabajar con algoritmos más avanzados y recientes, como ser TD3, PPO o SAC. Por otro lado, también sería interesante entrenar por separado diferentes comportamientos del dron, teniendo, por

[1] Tal vez desarrollando un plugin específico

ejemplo, una red neuronal para la elección de flancos y otra para la persecución de los mismos.

En cuanto a la combinación de técnicas de RL y heurísticas, se pueden explorar métodos alternativos al utilizado en el capítulo 6 como, por ejemplo, añadir a la recompensa proveniente del entorno $r_{env}$ una función de similitud:

$$r_t = r_{env} + \lambda.similitud(a_{rl}, a_h)$$

siendo $\lambda$ el peso relativo que tiene la heurística en la recompensa, mientras que la función *similitud* mide la diferencia entre la acción del agente $a_{rl}$ y la acción de la heurística $a_h$. Esta función devuelve un valor alto cuando ambas acciones se asemejan entre sí, y un valor más bajo en caso contrario. Otra opción posible es el promediado de acciones en cada paso del entrenamiento:

$$a_t = (1 - \gamma_t)a_{rl} + \gamma_t a_h$$

siendo $\gamma_t \in [0, 1]$ un valor que decae con el tiempo. Inicialmente, cuando $\gamma_t \approx 1$, el aprendiz ejecuta acciones muy parecidas a las generadas por la heurística y gradualmente va tomando el control a medida que acumula experiencia.

Como sugerencia final, todo este trabajo debería validarse con drones reales, primero en un entorno de laboratorio controlado utilizando aves simuladas, para luego pasar a realizar experimentos en condiciones reales dentro de un predio cultivado.

Esta página ha sido intencionalmente dejada en blanco.

# Apéndice A

# Hiperparámetros y GitLab

## A.1. Hiperparámetros

Las siguientes tablas detallan los valores de los hiperparámetros utilizados en los experimentos del capítulo 8.

| Algoritmo DDPG | | |
|---|---|---|
| Símbolo | Descripción | Valor |
| $l_{ra}$ | *Learning rate* del actor | 0.001 |
| $l_{rc}$ | *Learning rate* del crítico | 0.001 |
| $\gamma$ | Discount rate | 1.0 |
| $\tau$ | Tasa de actualización de redes target | 0.002 |
| - | Tamaño del buffer ER | 20000 |
| - | Tamaño del *mini-batch* | 50 |
| - | Número máximo de episodios de entrenamiento | 3500 |
| $T_{max}$ | Tiempo máximo por episodio | 220s |
| $\sigma_{max}$ | Std máxima del proceso Ornstein-Ulhenbeck | 1.0 |
| $\sigma_{min}$ | Std mínima del proceso Ornstein-Ulhenbeck | 0.15 |
| Umbrales | | |
| Símbolo | Descripción | Valor |
| $V_x$ | Máxima velocidad frontal del dron | 4m/s |
| $V_z$ | Máxima velocidad vertical del dron | 4m/s |
| $W$ | Máxima velocidad angular del dron | $\frac{\pi}{2}$r/s |
| $N_x$ | Número de celdas del espacio aéreo sobre el eje $x$ | 50 |
| $N_y$ | Número de celdas del espacio aéreo sobre el eje $y$ | 50 |
| $N_z$ | Número de celdas del espacio aéreo sobre el eje $z$ | 16 |
| $L_x$ | Largo de cada celda | 2.5m |
| $L_y$ | Ancho de cada celda | 2.5m |
| $L_z$ | Altura de cada celda | 2.5m |
| $L$ | Largo del campo ($N_x$x$L_x$) | 125m |
| $A$ | Ancho del campo ($N_y$x$L_y$) | 125m |
| $H$ | Altura del espacio aéreo ($N_z$x$L_z$) | 40m |
| - | Dimensiones de la matriz $M_t$ | 50x50x16 |
| $N_m$ | Número de celdas muestreadas de $M_t$ | 10 |
| $u_{RL}$ | Distancia entre los flancos $R$ y $L$ | 20m |

| | | |
|---|---|---|
| $u_F$ | Distancia entre el dron y el flanco $F$ | 10m |
| $d_{LR}$ | Distancia a las rectas tangentes r y f para tomar los puntos $R'$ y $L'$ sobre el plano $xy$ | 15m |
| Bandada | | |
| Símbolo | Descripción | Valor |
| $N_b$ | Número de boids en la bandada | 25 |
| $w_c$ | Peso relativo de la cohesión | 1.05 |
| $w_s$ | Peso relativo de la separación | 2.0 |
| $w_a$ | Peso relativo del alineamiento | 0.3 |
| $w_f$ | Peso relativo de la atracción a los árboles | 0.2 |
| $w_p$ | Peso relativo de la protección periférica (*Bounding*) | 0.2 |
| $w_d$ | Peso relativo de la depredador | 1.0 |
| $w_e$ | Peso relativo del movimiento aleatorio | 0.2 |
| $w_h$ | Peso relativo de la inercia | 0.2 |
| $r_b$ | Umbral de influencia entre boids | 4m |
| $r_d$ | Umbral de distancia entre el dron y los boids para activar el comportamiento de evasión | 32m |
| $v_n$ | Velocidad normal de los boids | 3m/s |
| $v_h$ | Velocidad de huida de los boids | 30m/s |

Tabla A.1: Valores de parámetros utilizados en los experimentos.

## A.2. GitLab

La implementación del algoritmo *DDPG* junto con el entorno de entrenamiento y la simulación de los boids, puede obtenerse clonando el proyecto **RL Flock** del *GitLab* de Facultad de Ingeniería:

```
1 git clone git@gitlab.fing.edu.uy:gabriel.rodriguez.frangias/rl-
      flock.git
```

Por su parte, el código asociado a los nodos ROS2 y la simulación Gazebo detallada en el capitulo 7, se obtiene clonando el proyecto **Ardupilot Flock**:

```
1 git clone git@gitlab.fing.edu.uy:gabriel.rodriguez.frangias/
      ardupilot-flock.git
```

El archivo README contiene todas las instrucciones de instalación y uso.

# Apéndice B

# Reinforcement Learning

## B.1. Agente-Entorno

En los últimos años, el paradigma de aprendizaje por recompensas o *Reinforcement Learning (RL)* [70] ha visto importantes avances, de los cuales se ha beneficiado el área de control.

Los algoritmos basados en este paradigma suelen utilizar métodos de aproximación de funciones, como ser redes neuronales (entre otros), en donde se establece una correspondencia entre el conjunto de estados posibles del sistema de interés, y las acciones que el controlador puede efectuar a partir de dichos estados. A esta correspondencia se la conoce como política y es lo que define el comportamiento del controlador a lo largo del tiempo. El objetivo de un algoritmo de RL es encontrar la política óptima que maximice cierta función de utilidad predeterminada.

La aproximación a la política ideal se logra en forma progresiva a través de la interacción de un agente (o controlador) con el entorno o dominio del problema a resolver. Luego de cada acción tomada por el agente, el entorno devuelve el nuevo estado en el que se encuentra luego de ejecutar dicha acción, además de un valor numérico que representa la recompensa o penalización obtenida por haber alcanzado ese nuevo estado. La figura B.1 muestra un esquema de la interacción agente-entorno.

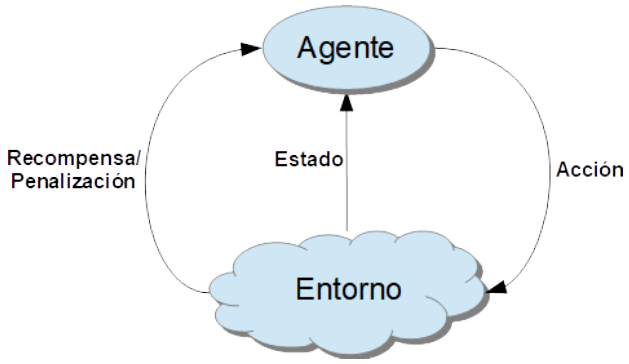



Figura B.1: Interacción del agente con el entorno

Luego, la evaluación de una política dada se basa en la acumulación de dichas recompensas o penalizaciones que el agente obtiene a lo largo de una secuencia de

interacciones con el entorno. A esta secuencia de interacciones se la denomina episodio. El objetivo es maximizar la recompensa o minimizar la penalización total obtenida en cada episodio.

## B.2. Proceso de decisión de Markov

En términos formales, RL se modela como un proceso de decisión de Markov $MDP = (S, A, P_a, R_a)$, en donde:

- $S$ es el espacio de estados del sistema, que puede ser discreto o continuo
- $A$ es el espacio de acciones que puede ejecutar el agente. Puede ser discreto o continuo.
- $P_a(s_{t_k}, s_{t_{k+1}})$ es la probabilidad de transición entre los estados $s_{t_k}$ y $s_{t_{k+1}}$, luego de ejecutar la acción $a$ desde el estado $s_{t_k}$, en el instante $t_k$[1].
- $R(s_{t_k}, s_{t_{k+1}})$ es la recompensa esperada luego de producirse la transición del estado $s_{t_k}$ a $s_{t_{k+1}}$, esto es, $R : S\text{x}S \to R$

El objetivo de RL es aprender una política óptima $\pi^*$ que maximice la recompensa total esperada al ejecutar la secuencia de acciones dictaminada por dicha política. La política aprendida puede ser estocástica, esto es:

$$\pi : \mathcal{S}\text{x}A \to [0, 1]$$
$$\pi(s, a) = Pr(A_t = a | S_t = s)$$

O puede ser determinista:

$$\pi : \mathcal{S} \to \mathcal{A}$$
$$\pi(s) = a$$

En este trabajo se asume una política determinista. Con estos elementos, se define un experimento o episodio de $\tau$ pasos como la secuencia de tuplas:

$$\{(s_{t_k}, a_{t_k}, r_{t_k}, s_{t_{k+1}})\}$$

siendo $k = 0, 1, ..., \tau - 1$, $s_{t_k} \in \mathcal{S}$ es el estado del sistema en el instante $t_k$, y $r_{t_k}$ la recompensa (o penalización) devuelta por el entorno al ejecutar la acción $a_{t_k}$. Si el episodio fue generado utilizando la política $\pi$, entonces $a_{t_k} = \pi(s_{t_k})$ y $r_{t_k} = R(s_{t_k}, s_{t_{k+1}})$, para todo $k \geq 0$.

Hallar la política óptima $\pi^*$ implica maximizar la recompensa esperada a lo largo del tiempo:

$$\pi^* = \arg\max_{\pi} \left\{ E_{s_{t+1} \sim P_{a_t = \pi(s_t)}(s_t, s_{t+1})} \left[ \sum_{t=0}^{\infty} \gamma^t R(s_t, s_{t+1}) \right] \right\}$$

siendo $\gamma \in (0, 1]$ el *discount factor*. Un valor menor a uno indica que la importancia de las experiencias futuras es cada vez menor a medida que estas experiencias ocurren más adelante en el tiempo. En general, se suele utilizar $\gamma = 1$ o un valor muy cercano a 1.

En la práctica, la política $\pi^*$ es estimada mediante la realización de experimentos sucesivos, es decir, el agente interactúa con el entorno en forma repetida, lo que le permite acumular información para poder estimar el valor esperado de la recompensa total. Esto es muy útil cuando no se cuenta con un modelo del entorno.

[1] Se asume una discretiación temporal dada tal que $t_{k+1} - t_k = \Delta t$ constante $\forall k \geq 0$.

## B.3. Función Q y ecuación de Bellman

El algoritmo de RL hace uso de la función $Q_\pi : SxA \rightarrow R$, tal que $Q_\pi(s, a)$ es el valor esperado de la recompensa obtenida si se parte del estado $s$, se toma la acción $a$, y se continúa aplicando la política $\pi$ a partir de ese momento en adelante. De esta forma, partiendo del instante $t = t_k$, resulta:

$$Q_\pi(s_{t_k}, a_{t_k}) = E_{s \sim P_a}\left[\sum_{i=k}^{\infty} \gamma^{i-k} R(s_{t_i}, s_{t_{i+1}})\right]$$

Por simplicidad en la notación, la abreviación $s \sim P_a$ reemplaza la expresión: $s_{t_i} \sim P_{a_{t_i}=\pi(s_{t_i})}(s_{t_i}, s_{t_{i+1}})$. Luego, aplicando linealidad del valor esperado y separando el primer término de la sumatoria, se obtiene:

$$Q_\pi(s_{t_k}, a_{t_k}) = E_{s \sim P_a}[R(s_{t_k}, s_{t_{k+1}})] + E_{s \sim P_a}\left[\sum_{i=k+1}^{\infty} \gamma^{i-k} R(s_{t_i}, s_{t_{i+1}})\right]$$

$$Q_\pi(s_{t_k}, a_{t_k}) = E_{s \sim P_a}[R(s_{t_k}, s_{t_{k+1}})] + Q_\pi(s_{t_{k+1}}, \pi(s_{t_k}))$$

Esta recursión permite estimar en forma iterativa el valor de la función $Q$ si se parte de un valor inicial. Los algoritmos de RL se basan en esta idea, que se conoce como la ecuación de Bellman [22]:

$$Q_\pi(s_{t_k}, a_{t_k}) \leftarrow \alpha Q_\pi(s_{t_k}, a_{t_k}) + (1 - \alpha)[r_{t_k} + \gamma Q_\pi(s_{t_{k+1}}, \pi(s_{t_k}))]$$

siendo $\alpha$ es el learning rate utilizado para la aproximación de $Q$, y $r_{t_k}$ la recompensa obtenida luego de ejecutar la acción $a_{t_k}$ desde el estado $s_{t_k}$, habiendo alcanzado luego el estado $s_{t_{k+1}}$. Este paso iterativo es básicamente un promedio (dado por $\alpha$) entre la estimación actual del valor de $Q_\pi$ y la nueva estimación que se obtiene una vez ejecutada la acción $a_{t_k}$ y observada la recompensa $r_{t_k}$, utilizando en cada paso la política $\pi$.

Un desarrollo mucho más detallado del paradigma de RL se puede obtener en «*Reinforcement learning: an introduction*» [70] de Richard Sutton y Andrew Barto.

Esta página ha sido intencionalmente dejada en blanco.

# Apéndice C

# Algoritmos de RL

En este apartado se hace una muy breve reseña de tres algoritmos de *RL*: *Q-Learning*, *DQN* y *DDPG*, siendo este último el algoritmo utilizado en la presente tésis.

## C.1. Q-Learning

En 1989, Chistopher Watkins introduce en su tesis doctoral el algoritmo conocido como *Q-Learning* [82], que utiliza el concepto de función $Q$ para entrenar un agente. El algoritmo no utiliza un modelo del entorno en donde opera el agente, por lo que dicho algoritmo se clasifica como *model free*.

Tanto el espacio de estados $S$ como el de acciones $A$ son discretos. Los valores $Q(s, a)$ se almacenan en una tabla de dimensiones $|S|$x$|A|$ y la regla de actualización de la función $Q$ es:

$$Q(s,a) \leftarrow Q(s,a) + \alpha * [r + \gamma * \underset{a' \in A}{\text{máx}}(Q(s',a')) - Q(s,a)]$$

siendo $s'$ el estado resultante de aplicar la acción $a$ desde el estado $s$, y $r$ la recompensa obtenida. En este caso, el uso de la función *max* es posible debido a que el espacio de acciones es discreto. La política $\pi$, aprendida por el algoritmo, se basa directamente en los valores de la tabla $Q$. Dado un estado $s$, se define:

$$\pi(s) = arg \underset{a' \in A}{\text{máx}} Q(s,a')$$

El algoritmo 9 muestra el pseudocódigo de Q-Learning, en donde se ejecuta un número predeterminado de episodios, cada uno con un número máximo de pasos. En cada uno de estos pasos, se selecciona una acción mediante el mecanismo $\epsilon_{greedy}$. Esto consiste en tomar una acción al azar con probabilidad $\epsilon$, y con probabilidad $1-\epsilon$ tomar la acción *greedy* que maximiza el valor $Q$.

El valor $\epsilon$ puede reducirse a medida que avanza el algoritmo, en lugar de ser fijo. En ese caso, el proceso comienza con un $\epsilon$ relativamente alto para permitir una mayor exploración. A medida que el algoritmo avanza, $\epsilon$ se reduce para favorecer la explotación de los valores $Q$ ya aprendidos.

Q-Learning solo es aplicable a espacios de estados y acciones discretos, aunque siempre es posible discretizar espacios continuos. Sin embargo, esta es su principal desventaja: la dimensionalidad. Dependiendo del problema, el tamaño de la tabla $Q$ se vuelve rápidamente inmanejable.

**Algorithm 9:** Q-Learning

**Data:** Entorno $E$. Constantes $\gamma$, $\alpha$ y $\epsilon$
Inicializar aleatoriamente los valores $Q(s, a)$
// Para cada episodio:
**for** *episode* $\leftarrow 1$ *to* $N$ **do**
    $s_0 \leftarrow E.reset()$ ; // Estado inicial
    $t \leftarrow 0$
    // Para cada paso en el episodio:
    **while** $t < T$ *o* $s_t$ *no es terminal* **do**
        // $\epsilon$-greedy
        $n \sim \mathcal{N}(0, 1)$
        **if** $n < \epsilon$ **then**
            Elegir $a_t$ al azar
        **else**
            $a_t = arg\,\text{máx}_{a' \in A}\, Q(s_t, a')$
        **end**
        $(r_t, s_{t+1}) \leftarrow E.step(a_t)$ ; // Ejecutar acción
        // Actualizar tabla $Q$:
        $Q(s, a) \leftarrow Q(s, a) + \alpha * [r_t + \gamma * \text{máx}_{a' \in A}(Q(s_{t+1}, a')) - Q(s, a)]$
        $s \leftarrow s'$
    **end**
**end**

## C.2. DQN

En 2013, *DeepMind Tecnologies* introduce el algoritmo conocido como *Deep Reinforcement Learning* o DQN [52]. En ese trabajo se aborda el problema de la dimensionalidad sustituyendo la tabla $Q$ por una red neuronal que aproxima la función $Q$ mediante el mecanismo de descenso por gradiente. Las acciones, sin embargo, continúan siendo discretas.

Durante el entrenamiento, se hace uso de un buffer denominado *Experience Replay* o ER, de tamaño $N_e$. Las acciones que ejecuta el agente no son inmediatamente utilizadas en la actualización de los pesos de la red neuronal, sino que se almacenan en el ER junto con la acción ejecutada, la recompensa obtenida y el nuevo estado alcanzado. Luego, en forma periódica, se realiza una actualización en lote de la red (descenso por gradiente) a partir de un conjunto de entradas del ER seleccionadas al azar. Esta selección aleatoria de acciones evita la correlación que, en caso contrario, tendrían las acciones consecutivas ejecutadas por el agente.

Por otro lado, se mantiene una segunda versión de la red neural $Q$, llamada red target o $Q'$. Inicialmente, los pesos de ambas redes son idénticos, pero luego se actualizan en momentos diferentes. La red $Q'$ se mantiene sin cambios durante un número de pasos predefinido, luego de los cuales se iguala nuevamente a la red $Q$.

La idea es que la red $Q'$, la cual es una versión un poco más antigua de $Q$, se utilice en la recursión dada por la ecuación de Bellman. De no hacerlo así, se estaría actualizando la red $Q$ a partir de sí misma y en el mismo instante de tiempo, lo cual genera problemas de estabilidad en el entrenamiento.

El algoritmo 10 muestra el seudocódigo de DQN. Al comienzo se inicializan las redes $Q$, $Q'$ y el buffer de experiencias $ER$. Los dos bucles principales son idénticos

a los de Q-Learning. En cada paso del episodio se obtiene la acción $a_t$ mediante el mecanismo $\epsilon_{greedy}$. Luego se ejecuta la acción y se registra la transición $(s_t, a_t, r_t, s_{t+1})$ en el buffer $ER$ mediante la función $ER.push$.

**Algorithm 10:** DQN

**Data:** Entorno $E$. Constantes $\gamma$, $\alpha$, $N_e$, $C$ y $\epsilon$
Inicializar buffer $ER$ con capacidad $N_e$
Inicializar la red $Q$ con valores aleatorios
Inicializar la red $Q'$ tal que $Q' = Q$
**for** *episode* $\leftarrow 1$ *to* $N$ **do**
    $s_0 \leftarrow E.reset()$ ; // Estado inicial
    $t \leftarrow 0$,
    **while** $t < T$ *o* $s_t$ *no es terminal* **do**
        // $\epsilon$-greedy
        $n \sim \mathcal{N}(0, 1)$
        **if** $n < \epsilon$ **then**
            Elegir $a_t$ al azar
        **else**
            $a_t = arg\,\text{máx}_{a' \in A}\, Q(s_t||a')$ ; // ||: concatenación de estados
        **end**
        $(r_t, s_{t+1}) \leftarrow E.step(a_t)$ ; // Ejecutar acción
        $ER.push(s_t, a_t, r_t, s_{t+1})$ ; // Añadir experiencia al buffer
        Sortear un lote de transiciones $\{(s_i, a_i, r_i, s_{i+1})\}$ del buffer $ER$
        // Calcular nueva estimación de $Q(s_i, a_i)$:

$$y_i = r_i + \gamma \max_{a' \in A} Q'(s_{i+1}||a')$$

        // Calcular función de perdida $J$:

$$J = [y_i - Q(s_i||a_i)]^2$$

        Aplicar un paso de descenso por gradiente de la red $Q$ según $J$
        Cada $C$ pasos hacer $Q' = Q$
        $t \leftarrow t + 1$
    **end**
**end**

Cuando el buffer $ER$ cuenta con un cierto número de entradas, se sortea un lote de experiencias de tamaño fijo. Con este lote se calcula la función de pérdida $J$ que luego es utilizada para realizar un paso de descenso por gradiente en la red $Q$. Notar que la nueva estimación de $Q$, $y_i$, se basa en la red $Q'$, mientras que la red $Q$ se utiliza para calcular el error de estimación por mínimos cuadrados ($J$). Finalmente, cada cierto número de pasos $C$, se igualan los pesos de las redes haciendo $Q' = Q$.

Este algoritmo es un avance con respecto a Q-Learning debido a que elimina la tabla $Q$ reemplazándola por una red neuronal. Sin embargo, aún requiere un espacio de acciones discreto.

## C.3. DDPG: Deep Deterministic Policy Gradient

En 2015, *DeepMind* introduce el algoritmo *Deep Deterministic Policy Gradient* o DDPG [44], que es el algoritmo utilizado en el presente trabajo.

DDPG aborda problemas en donde el espacio de estados y acciones es continuo. Para esto, utiliza redes neuronales tanto para evaluar la función $Q$ como la política $\pi$.

### C.3.1. Redes Neuronales

DDPG se clasifica como un algoritmo del tipo actor-crítico, en donde el actor es el agente que interactúa con el entorno, y el crítico es la función $Q$ que evalúa qué tan buenas son las acciones ejecutadas por el actor. Se utilizan en total 4 redes neuronales:

- Actor: Red $\pi_\psi$ con parámetros $\psi$
- Crítico: Red $Q_\theta$ con parámetros $\theta$
- Actor Target: Red $\pi_{\psi'}$ con parámetros $\psi'$
- Crítico Target: Red $Q_{\theta'}$ con parámetros $\theta'$

Las redes target cumplen el mismo rol que en el algoritmo $DQN$: Son versiones un poco más antiguas de las redes principales, y se utilizan en la evaluación de la ecuación de Bellman con el fin de estabilizar el proceso de entrenamiento.

### C.3.2. Exploración

Al ser la red $\pi_\psi$ determinista, es necesario introducir algún mecanismo de exploración para que el entrenamiento sea efectivo. Sin embargo, ya no es posible usar $\epsilon_{greedy}$ porque el espacio de acciones es continuo y, por tanto, $argmax$ deja de tener sentido [1].

Para lograr una exploración efectiva, a cada acción calculada por la red $\pi_\psi$ en el i-ésimo paso, se le añade un ruido aleatorio $n_i$, obteniendo así la acción $a_i$ que es la que efectivamente ejecuta el agente:

$$a_i = \pi_\psi(s_i) + n_i$$

El ruido $n_i$ es muestreado a partir de un proceso *Ornstein-Uhlenbeck* [77], en donde las muestras obtenidas en forma consecutiva están correlacionadas entre sí. Cada muestra $n_i$ está definida por la siguiente ecuación en diferencias:

$$n_i = n_{i-1} + \theta(\mu - n_{i-1}) + \sigma w$$

Siendo $0 < \theta \ll 1$, $w \sim \mathcal{N}(0,1)$ y $(\mu, \sigma)$ la media y amplitud del ruido aplicado en cada paso. Tomando $\mu = 0$ y reordenando términos, resulta:

$$n_i = (1-\theta)n_{i-1} + \sigma w$$

### C.3.3. Experience Replay

Al igual que el algoritmo DQN, DDPG hace uso del buffer *Experience Replay* o $ER$, de tamaño máximo $N_e$. Una entrada en este buffer, en el i-ésimo paso, es una tupla con la siguiente estructura:

$$(s_i, a_i, r_i, s_{i+1}, t_i)$$

[1] De todos modos, existen formas de hacer esto, como el algoritmo DDPG-argmax [49], pero incrementan la complejidad de la implementación

siendo $a_i$ la acción ejecutada por el agente en el estado $s_i$, $r_i$ la recompensa obtenida, $s_{i+1}$ el estado alcanzado luego de ejecutar la acción, y $t_i$ es un indicador booleano tal que $t_i = 1$ si el estado $s_{i+1}$ alcanzado es terminal, o $t_i = 0$ en caso contrario.

Durante el entrenamiento, al igual que en el caso de DQN, se sortea un conjunto de tuplas del buffer $ER$, llamado mini-batch, de tamaño $M$ el cual es utilizado para actualizar las redes neuronales. Si bien el tamaño máximo del buffer $ER$ es relativamente grande, por ejemplo $N_e = 20.000$, el mini-batch es mucho más pequeño, por ejemplo $M = 50$ o 100 tuplas.

Esto no representa una ineficiencia del algoritmo en el sentido de que puedan quedar experiencias sin utilizar durante el entrenamiento. El sorteo del mini-batch y la consecuente actualización de las redes se realiza en cada paso del algoritmo. Esto significa que cada experiencia que entra al buffer $ER$ permanece allí durante $N_e$ pasos, en cada uno de los cuales se realiza un sorteo de $M$ experiencias. Por tanto, cada experiencia será seleccionada, en promedio, $M$ veces durante su paso por el *Experience Replay*.

## C.3.4. Funciones de pérdida

### Actor

Para la actualización de las redes se utiliza el procedimiento de descenso por gradiente. Para el caso del actor con red $\pi_\psi$, el objetivo es maximizar la recompensa total:

$$J_\psi = \mathbb{E}_{s_t \sim P_a}[Q_\theta(s_t, \pi_\psi(s_t))]$$

La dirección de descenso por gradiente con respecto a los parámetros $\psi$ de la política $\pi_\psi$ es entonces:

$$-\nabla_\psi J_\psi = -\nabla_\psi \mathbb{E}_{s_t \sim P_a}[Q_\theta(s, a)|_{s=s_t, a=\pi_\psi(s_t)}]$$

$$-\nabla_\psi J_\psi = -\mathbb{E}_{s_t \sim P_a}[\nabla_a Q_\theta(s, a)|_{s=s_t, a=\pi_\psi(s_t)} \nabla_\psi \pi_\psi(s)|_{s=s_t}]$$

Para la estimación de $J_\psi$, se hace uso del mini-batch $B$ de tamaño $M$, tal que:

$$B = \{(s_i, a_i, r_i, s_{i+1}, t_i) \in ER,\ i = 0, ..M-1\}$$

$$J_\psi \approx -\frac{1}{M} \sum_{i=0}^{M-1} Q_\theta(s_i, a_i)$$

$$a_i = \pi_\psi(s_i) + n_i$$

En la estimación de $J_\psi$ se incorpora la exploración mediante el uso de la acción modificada con el ruido $n_i$, la cual fue almacenada en el buffer.

### Crítico

Para el caso del crítico, el objetivo es minimizar el error entre la estimación actual de $Q$ y la nueva estimación, $y_t$, luego de ejecutar una acción y observar la recompensa obtenida. Se define entonces la siguiente función con parámetros $\theta$:

$$J_\theta = \mathbb{E}_{s_t \sim P_a}\left[(Q_\theta(s_t, \pi_\psi(s_t)) - y_t)^2\right]$$

$$y_t = R(s_t, \pi_\psi(s_t)) + \gamma Q_\theta(s_{t+1}, \pi_\psi(s_{t+1}))$$

Para obtener una estimación de $J_\theta$ se vuelve a utilizar el mini-batch $B$, pero el cálculo de $y_t$ se basa en las redes target:

$$J_\theta \approx \frac{1}{M} \sum_{i=0}^{M-1} [Q_\theta(s_i, a_i) - y_i]^2$$

$$y_i = r_i + \gamma(1 - t_i) Q_{\theta'}(s_{i+1}, \pi_{\psi'}(s_{i+1}))$$

La expresión precedente es básicamente el error cuadrático medio entre la estimación actual de $Q_\theta$ y la nueva estimación luego de observar las experiencias contenidas en el mini-batch.

Como se mencionó anteriormente, las redes target $Q_{\theta'}$ y $\pi_{\psi'}$ que intervienen en el cálculo de $y_i$ reducen la inestabilidad durante el entrenamiento que de lo contrario surgiría al utilizar las mismas redes que se están actualizando.

### C.3.5. Actualización de redes target

Los parámetros de las redes target $\pi_{\psi'}$ y $Q_{\theta'}$ se actualizan en cada paso del entrenamiento mediante la siguiente regla conocida como *soft update*:

$$\psi' \leftarrow \tau\psi + (1 - \tau)\psi'$$

$$\theta' \leftarrow \tau\theta + (1 - \tau)\theta'$$

Siendo $0 < \tau \ll 1$ un hiperparámetro. Este paso se diferencia del algoritmo DQN, que simplemente igualaba las redes cada cierta cantidad de pasos. Aquí la actualización de las redes target ocurre en todos los pasos del algoritmo y las mismas nunca llegan a igualarse a las redes principales.

### C.3.6. Algoritmo

El algoritmo 11 presenta el seudocódigo de *Deep Determinstc Policy Gradient*. La estructura de este algoritmo es muy similar a la de los dos algoritmos anteriores, en especial DQN. Al comienzo, se inicializan las 4 redes neuronales y el buffer $ER$. El paso con $\epsilon_{greedy}$ se reemplaza con el proceso estocástico *Ornstein-Uhlenbeck*.

Al igual que DQN, se sortea un lote o mini-batch de experiencias del buffer $ER$, que luego se utilizan para actualizar las redes $\pi_\psi$ y $Q_\theta$ mediante descenso por gradiente.

Finalmente, se realiza un soft update de las redes target $\pi_{\psi'}$ y $Q_{\theta'}$.

**Algorithm 11:** DDPG

**Data:** Entorno $E$. Constantes $M$, $N$, $T$, $\theta$, $\sigma$

Inicializar aleatoriamente los pesos $\psi$ y $\theta$ de las redes $\pi_\psi$ y $Q_\theta$

Inicializar las redes target con $\psi' \leftarrow \psi$ y $\theta' \leftarrow \theta$

Inicializar buffer $ER$

**for** *episode* $\leftarrow 1$ *to* $N$ **do**

  Inicializar proceso aleatorio $n(\theta, \sigma)$

  $s_0 \leftarrow E.reset()$ ; // Estado inicial

  $t \leftarrow 0$, $t_0 \leftarrow 0$

  **while** $t < T$ *or* $t_t = 0$ **do**

    $a_t = \pi_\psi(s_t) + n_t$ ; // Proceso *Ornstein-Uhlenbeck*

    $(s_t, a_t, r_t, s_{t+1}, t_t) \leftarrow E.step(a_t)$ ; // Ejecutar acción

    $ER.push(s_t, a_t, r_t, s_{t+1}, t_t)$ ; // Añadir experiencia al buffer

    Muestrear un mini-batch aleatorio $B = \{(s_i, a_i, r_i, s_{i+1}, t_i)\}$

    // Estimación de nuevo valor de Q:

    $y_i \leftarrow r_i + \gamma(1 - t_i)Q_{\theta'}(s_i, \pi_{\psi'}(s_{i+1}))$

    // Actualizar red $Q_\theta$ minimizando:

    $\frac{1}{M}\sum_{i=0}^{M-1}\left[Q_\theta(s_i, a_i) - y_i\right]^2$

    // Actualizar red $\pi_\psi$ minimizando:

    $-\frac{1}{M}\sum_{i=0}^{M-1} Q_\theta(s_i, a_i)$

    // Actualizar redes target:

$$\psi' \leftarrow \tau\psi + (1 - \tau)\psi'$$

$$\theta' \leftarrow \tau\theta + (1 - \tau)\theta'$$

    $t \leftarrow t + 1$

  **end**

**end**

Esta página ha sido intencionalmente dejada en blanco.

# Apéndice D

# Potential-Based Reward Shaping

La técnica de *reward shaping* basada en funciones de potencial se utiliza para proporcionar orientación adicional al agente, acelerando el proceso de aprendizaje al otorgar recompensas intermedias hasta alcanzar el objetivo final [39].

Si se modela el problema de $RL$ como un proceso de decisión de Markov $M = (S, A, P_a, R_a)$, el mecanismo de *Reward Shaping* entrena el agente en un segundo proceso de Markov $M' = (S, A, P_a, R'_a)$ en donde:

$$R'_a = R_a + F$$

$$F : S\text{x}S \rightarrow \mathcal{R}$$

Es decir, se modifica la recompensa original añadiendo el término $F$. Los demás elementos del proceso $M'$ permanecen inalterados con respecto al proceso original $M$.

En principio, las políticas óptimas de $M$ y $M'$ no son necesariamente iguales. Sin embargo, en [53] se muestra que el conjunto de políticas óptimas permanece invariante si se elige $F$ de la siguiente manera:

$$F(s_t, s_{t+1}) = \gamma\phi(s_{t+1}) - \phi(s_t)$$

siendo $0 < \gamma \leq 1$ el *discount factor* utilizado en la ecuación de Bellman. Tomando $\phi(s) \geq 0$ para todo $s \in S$, se evita que el agente explote en forma repetida la recompensa obtenida durante una transición de estados ventajosa $s \rightarrow s'$, volviendo al estado anterior $s$ una y otra vez.

Efectivamente, si $F(s, s') > 0$ se cumple que $F(s, s') + F(s', s) \leq 0$, anulando así la ventaja de la transición $s \rightarrow s'$:

$$\begin{aligned} F(s, s') + F(s', s) &= \gamma\phi(s') - \phi(s) + \gamma\phi(s) - \phi(s') \\ &= \gamma(\phi(s') + \phi(s)) - (\phi(s') + \phi(s)) \\ &= (\gamma - 1)(\phi(s') + \phi(s)) \leq 0 \end{aligned} \tag{D.1}$$

La desigualdad D.1 es cierta porque se asume $0 < \gamma \leq 1$ y $\phi(s) \geq 0$ para todo $s \in S$.

## D.1. Función $\phi$

Para la recompensa utilizada en el algoritmo del capítulo 5, se elige la función $\phi$ como la suma ponderada de los siguientes elementos:

- $D_{set}(c'_f, O'_t)$: Distancia de la bandada al centro del espacio aéreo, en su versión bidimensional.
- $b_t$: Cantidad de aves aún presentes dentro del campo.
- $d_t$: Distancia entre dron y el centro de masa de la bandada.

$$\phi(s_t) = w_0 D_{set}(c'_f, O'_t) - w_1 b_t - w_2 d_t$$

Las constantes $w_i > 0$ asignan pesos relativos a cada componente de la recompensa.

Luego, eligiendo la recompensa $R_a$ del proceso de Markov original como una constante $\epsilon < 0$, y el *discount factor* $\gamma = 1$, la recompensa $r_t$ resulta ser:

$$r_t = \epsilon + w_0[D_{set}(c'_f, O'_t) - D_{set}(c'_f, O'_{t-1})] + w_1[b_{t-1} - b_t] + w_2[d_{t-t} - d_t]\mathbf{1}_{dt>u_d}$$

La recompensa $r_t$ así definida, premia al agente cada vez que la bandada se desplaza en la dirección correcta, alejándose del centro del campo, y lo penaliza en caso contrario. También se lo premia cada vez que un ave sale del espacio aéreo, reforzando la importancia de expulsar a la bandada.

El término ponderado con $w_2$, incentiva al agente a acercarse a la bandada sólo si la distancia entre ambos es mayor al umbral $u_d$.

# Apéndice E

# ROS2 y Gazebo

## E.1. ROS2

ROS 2 (Robot Operating System 2) [61] es un conjunto de bibliotecas y herramientas de código abierto diseñadas para facilitar el desarrollo de software para robots. Es la evolución del sistema original ROS (Robot Operating System), presentando mejoras en cuanto a seguridad, tiempo real y escalabilidad.

Es compatible con Linux, Windows, macOS y ofrece una arquitectura distribuida basada en Data Distribution Service (DDS) [19] como middleware de comunicación. Para desarrollar código en ROS2, es necesario definir los siguientes elementos básicos:

- Workspace
- Package
- Nodos
- Mensajes

### E.1.1. Workspace

ROS 2 se basa en la combinación de espacios de trabajo o workspaces. Un workspace es el lugar del sistema en donde se aloja el código desarrollado, junto con las variables de ambiente necesarias para ejecutar dicho código. El workspace principal de ROS 2 se denomina «underlay», mientras que los workspaces locales creados posteriormente se denominan «overlays». Al desarrollar con ROS 2, normalmente se tienen varios workspaces activos simultáneamente.

Para crear un workspace, simplemente se crea un directorio destino y, bajo este, el subdirectorio *src* donde residirá el código fuente:

```
$ mkdir -p ~/ros2_ws/src
$ cd ~/ros2_ws/src
$ source /opt/ros/humble/setup.bash
```

El script setup.bash configura las variables de ambiente del workspace principal de ROS2.

### E.1.2. Package

Un package es una unidad organizacional dentro de ROS2 que permite instalar y compartir el código desarrollado. Al crear el package, se define el lenguaje de programación a utilizar, que puede ser Python o C++.

Por ejemplo, para crear un package basado en código Python, se ejecuta el siguiente comando desde el directorio *src*:

```
$ cd ~/ros2_ws/src
$ ros2 pkg create --build-type ament_python --license Apache-2.0
    my_package
```

El comando precedente creará la siguiente estructura de directorios y archivos:

```
~ros2_ws/
       src/
         my_package/
           package.xml
           resource/my_package
           setup.cfg
           setup.py
           my_package/
```

### E.1.3. Nodos

Un nodo en ROS2 es un código ejecutable que reside dentro de un package. Un nodo puede encargarse de codificar un comportamiento simple del robot, procesar y publicar datos de uno o varios sensores, ofrecer servicios a otros nodos, etc.

Por ejemplo, para crear un nodo que imprima el texto «Hola mundo», colocar el archivo Python hola_mundo.py bajo el directorio src/my_package/my_package, conteniendo el siguiente código:

```
import rclpy
from rclpy.node import Node
class My_Node(Node):
    def __init__(self):
        super().__init__("my_node")
        print('Hola mundo...')

def main(args=None):
    rclpy.init(args=args)
    my_node = My_Node()
    rclpy.spin(my_node)
    my_node.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

Listing E.1: Nodo hola_mundo.py

El nodo se define extendiendo la clase Node. El procedimiento main utiliza el API rclpy.spin para iniciar y mantener en ejecución el nodo.

Luego es necesario añadir un punto de entrada en el archivo *setup.py* para poder ejecutar el nodo. En dicho archivo, añadir la siguiente línea dentro de la lista *console_scripts*:

```
entry_points={
        'console_scripts': [
                'hola_mundo = my_package.hola_mundo:main',
        ],
},
```

Para poder ejecutar el nodo, es necesario compilar el package en el cual reside (aún en el caso de tratarse de un nodo Python). El proceso de compilación genera

el código ejecutable, junto con varios archivos de configuración, bajo los directorios *install* y *build*, los cuales se crean en caso de no existir. Así, para compilar el package *my_package* se utiliza el comando *colcon build*:

```
$ colcon build --packages-select my_package
```

Finalmente, para ejecutar el nodo, se utiliza el comando *run* de ROS2:

```
$ cd ~/ros2_ws
$ ros2 run my_package hola_mundo
$ Hola mundo...
```

### E.1.4. Mensajes

Los mensajes son utilizados por los nodos para intercambiar información a través de canales denominados tópicos. Un nodo puede publicar datos en varios tópicos y, a la vez, suscribirse a otros tópicos para recibir datos. Esto significa que la información publicada en un tópico no tiene un destinatario predeterminado, sino que es necesario que el nodo destino se suscriba al tópico deseado. Por otro lado, varios nodos pueden publicar en un mismo tópico en forma simultánea. La figura E.1 ilustra esta relación.

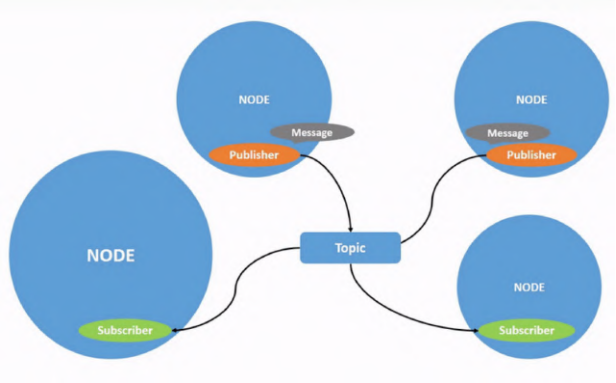



Figura E.1: Múltiples nodos pueden publicar en un mismo tópico. A su vez, múltiples nodos pueden suscribirse a dicho tópico [64].

Por ejemplo, el siguiente código implementa un nodo que publica el mensaje «Hola Mundo ...», dos veces por segundo, en el tópico de nombre *topic*:

```
import rclpy
from rclpy.node import Node
from std_msgs.msg import String

class MinimalPublisher(Node):
    def __init__(self):
      super().__init__('minimal_publisher')
      self.publisher_ = self.create_publisher(String, 'topic', 10)
      timer_period = 0.5  # seconds
      self.timer = self.create_timer(timer_period, self.
    timer_callback)
      self.i = 0

    def timer_callback(self):
      msg = String()
      msg.data = 'Hola Mundo: %d' % self.i
```

```
        self.publisher_.publish(msg)
        self.get_logger().info('Publishing: "%s"' % msg.data)
        self.i += 1

def main(args=None):
    rclpy.init(args=args)

    minimal_publisher = MinimalPublisher()
    rclpy.spin(minimal_publisher)
    # Destroy the node explicitly
    # (optional - otherwise it will be done automatically
    # when the garbage collector destroys the node object)
    minimal_publisher.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

Listing E.2: Código del nodo minimal_publisher que publica el mensaje «Hola Mundo» dos veces por segundo [65].

En primer lugar, se declara la publicación del tópico etiquetado como *topic*. La función *timer_callback*, invocada cada 0.5 segundos por el *timer*, publica el mensaje «Hola Mundo: $i$» en el tópico, siendo $i$ un secuencial que se incrementa con cada invocación.

Por su parte, el siguiente código implementa un nodo que se suscribe al tópico *topic* para consumir los datos enviados por el publicador *minimal_publisher*:

```
import rclpy
from rclpy.node import Node

from std_msgs.msg import String


class MinimalSubscriber(Node):

    def __init__(self):
        super().__init__('minimal_subscriber')
        self.subscription = self.create_subscription(
            String,
            'topic',
            self.listener_callback,
            10)
        self.subscription  # prevent unused variable warning

    def listener_callback(self, msg):
        self.get_logger().info('I heard: "%s"' % msg.data)


def main(args=None):
    rclpy.init(args=args)

    minimal_subscriber = MinimalSubscriber()

    rclpy.spin(minimal_subscriber)

    # Destroy the node explicitly
    # (optional - otherwise it will be done automatically
    # when the garbage collector destroys the node object)
    minimal_subscriber.destroy_node()
    rclpy.shutdown()
```

```
34
35
36 if __name__ == '__main__':
37     main()
```

Listing E.3: Código del nodo *minimal_subscriber* que se suscribe al topico *topic* [65].

El constructor de la clase *MinimalSubscriber* crea la suscripción al tópico *topic*, la cual se asocia a la función *listener_callback*. Esta función es invocada cada vez que hay un mensaje disponible en el tópico. En este caso, la función de callback simplemente imprime en consola el mensaje recibido.

Para obtener una descripción más detallada de los nodos publicador y el suscriptor, consultar [65], o los tutoriales de ROS2 en [63].

## E.2. Gazebo

Gazebo es un simulador 3D de código abierto para robótica, que permite simular robots en entornos complejos, tanto interiores como exteriores.

Ofrece un motor de física con el que se puede simular la dinámica de un robot, junto con elementos del entorno como, por ejemplo, viento. Tiene la capacidad de renderizar dichos entornos, incluyendo iluminación, sombras y texturas.

Es posible, también, simular diferentes tipos de sensores, como cámaras, láseres, IMUs, etc., los cuales pueden ser añadidos al robot.

Por otro lado, se integra con ROS2 a través del package *ros_gz_bridge*, el cual permite el intercambio de mensajes entre ROS 2 y Gazebo Transport, aunque solo para ciertos tipos de datos.

La figura E.2 muestra la interfaz gráfica de Gazebo, donde se presenta un mundo muy simple con 5 volúmenes de ejemplo.

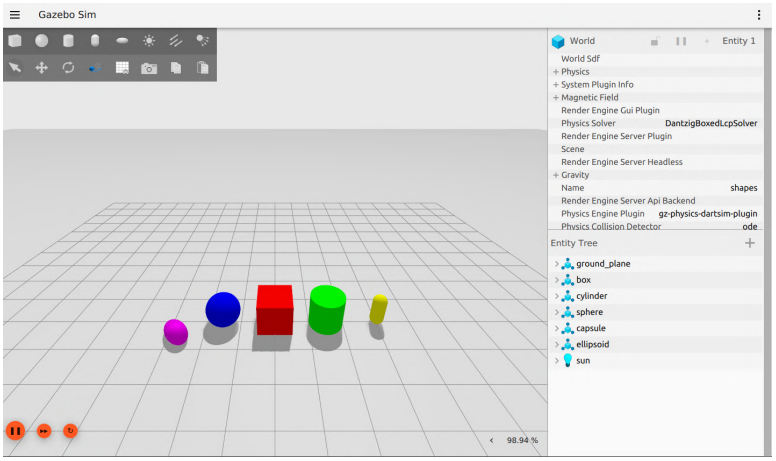



Figura E.2: Interfaz gráfica de Gazebo mostrando 5 volúmenes diferentes [30].

### E.2.1. Mundos simulados

La definición de mundos virtuales en Gazabo se basa en un archivo XML con formato SDF (Simulation Description Format) [67]. Este formato permite describir objetos

y diversos elementos que forman parte del entorno, así como también características de visualización y control. Su estructura básica es la siguiente:

```
<?xml version="1.0" ?>
<sdf version="1.8">
    <world name="world_demo">
    ...
    ...
    </world>
</sdf>
```

Listing E.4: Archivo world_demo.sdf

Bajo la etiqueta *world*, es posible incluir otros archivos conteniendo diferentes modelos de robot. Una vez creado el archivo SDF, se inicia la simulación del mismo con el comando **gz**:

```
$ gz sim world_demo.sdf
```

### E.2.2. Modelos

Los modelos se definen bajo la etiqueta *model* dentro de *world*:

```
<?xml version="1.0" ?>
<sdf version="1.8">
    <world name="world_demo">
    ...
       <model name='vehicle_blue'>
          <pose>0 0 0 0 0 0</pose>
          ...
       </model>
    </world>
</sdf>
```

En el código precedente se define el modelo *vehicle_blue*. La etiqueta *pose* se utiliza para definir la posición y orientación del modelo, mediante la especificación de 6 valores:

```
    <pose >x y z r p y</pose>
```

siendo $x, y, z$ la posición del modelo en el espacio y $r, p, y$ su orientación en ángulos de *roll*, *pitch* y *yaw* respectivamente.

### E.2.3. Links

En Gazebo, los *links* representan objetos rígidos con ciertas propiedades como masa, inercia, apariencia visual y geometría de colisión. Para su definición se utiliza la etiqueta *link* dentro del modelo, por ejemplo:

```
...
  <model name='vehicle_blue'>
  <pose>0 0 0 0 0 0</pose>
    <link name='chassis'>
        <pose relative_to='__model__'>0.5 0 0.4 0 0 0</pose>
        <inertial>
            <mass>1.14395</mass>
            <inertia>
                <ixx>0.095329</ixx>
                <ixy>0</ixy>
                <ixz>0</ixz>
```

```
                <iyy>0.381317</iyy>
                <iyz>0</iyz>
                <izz>0.476646</izz>
            </inertia>
        </inertial>
        <visual name='visual'>
            <geometry>
                <box>
                    <size>2.0 1.0 0.5</size>
                </box>
            </geometry>
            <material>
                <ambient>0.0 0.0 1.0 1</ambient>
                <diffuse>0.0 0.0 1.0 1</diffuse>
                <specular>0.0 0.0 1.0 1</specular>
            </material>
        </visual>
        <collision name='collision'>
            <geometry>
                <box>
                    <size>2.0 1.0 0.5</size>
                </box>
            </geometry>
        </collision>
    </link>
  </model>
  ...
```

Listing E.5: Especificación del chasis de un robot de 3 ruedas

El código precedente define un *link* llamado *chassis*, al cual se le asigna una posición relativa al modelo, una matriz inercial, una geometría (prisma rectangular), propiedades del material (cómo interactúa con la luz del entorno) y la geometría de colisión que en este caso coincide con la geometría del objeto. La figura E.3 muestra el resultado.

Este chasis representa un robot con dos ruedas laterales y una rueda esférica frontal. La rueda derecha se define como un *link* con forma cilíndrica, según la siguiente especificación:

```
<link>
    <pose relative_to="chassis">-0.5 -0.6 0 -1.5707 0 </pose>
    <inertial>
        <mass>1</mass>
        <inertia>
            <ixx>0.043333</ixx>
            <ixy>0</ixy>
            <ixz>0</ixz>
            <iyy>0.043333</iyy>
            <iyz>0</iyz>
            <izz>0.08</izz>
        </inertia>
    </inertial>
    <visual name='visual'>
        <geometry>
            <cylinder>
                <radius>0.4</radius>
                <length>0.2</length>
            </cylinder>
        </geometry>
        <material>
```

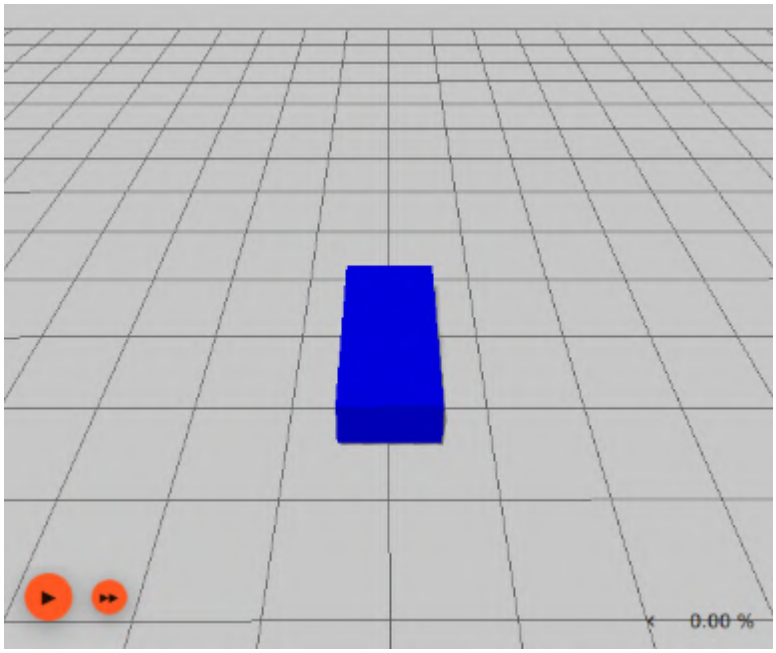



Figura E.3: Render del link chassis en Gazebo.

```
            <ambient>1.0 0.0 0.0 1</ambient>
            <diffuse>1.0 0.0 0.0 1</diffuse>
            <specular>1.0 0.0 0.0 1</specular>
        </material>
    </visual>
    <collision name='collision'>
        <geometry>
            <cylinder>
                <radius>0.4</radius>
                <length>0.2</length>
            </cylinder>
        </geometry>
    </collision>
</link>
```

Listing E.6: Especificación de la rueda derecha del robot

El resultado se muestra en la figura E.4. La rueda izquierda se define en forma análoga.

### E.2.4. Joints

Los *joints* conectan *links* y determinan cómo estos interactúan. Por ejemplo, el *link* de la rueda derecha debe rotar con respecto al *link* del chasis. Esto se especifica utilizando la etiqueta *joint* y el atributo *revolute*:

```
<joint name='right_wheel_joint' type='revolute'>
    <pose relative_to='right_wheel'/>
    <parent>chassis</parent>
    <child>right_wheel</child>
    <axis>
```

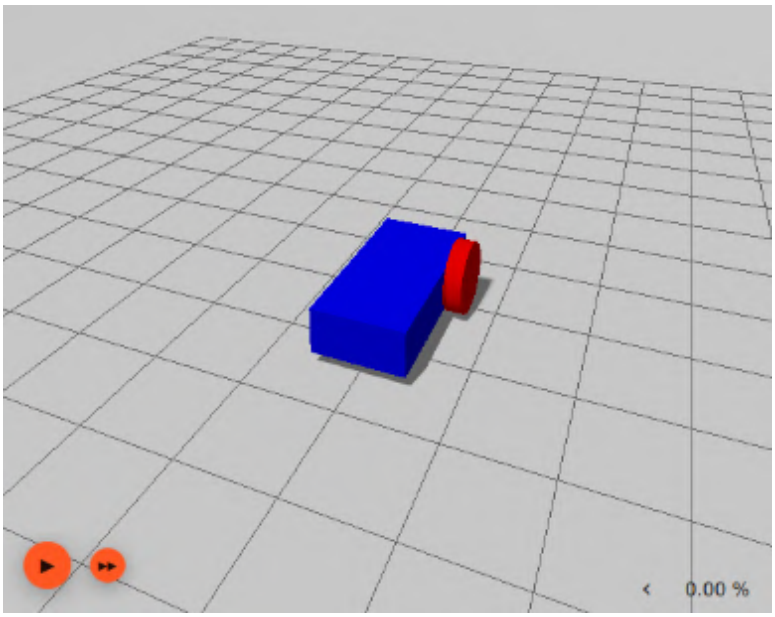



Figura E.4: Render del chasis del robot junto con la rueda derecha.

```
        <xyz expressed_in='__model__'>0 1 0</xyz>
        <limit>
            <lower>-1.79769e+308</lower><!--negative infinity-->
            <upper>1.79769e+308</upper><!--positive infinity-->
        </limit>
    </axis>
</joint>
```

Listing E.7: Joint entre el chasis y la rueda derecha

En el código precedente, se define el eje de rotación de la rueda como el eje $y$ del modelo. Los límites de rotación superior e inferior se definen como $\pm\infty$.

Para obtener una descripción más detallada sobre la construcción de modelos en Gazebo, consultar los tutoriales en [28].

Esta página ha sido intencionalmente dejada en blanco.

# Apéndice F

# Algoritmo de entrenamiento

El entrenamiento de la red neuronal del algoritmo $RL2$ y del algoritmo descrito en el capítulo 5, se detalla en el seudocódigo 12. Luego de inicializar el agente $DDPG$ y el *buffer* de experiencias ER con 20000 entradas, se procede a ejecutar un máximo de 1000 episodios.

**Algorithm 12:** Entrenamiento de RL2

**Data:** Entorno $E$

$state\_dim \leftarrow 37$ ; // Dimensión del espacio de estados

$action\_dim \leftarrow 3$ ; // Dimensión del espacio de acciones

$agent \leftarrow DDPGagent(state\_dim, action\_dim)$ ; // Agente DDPG

$ER \leftarrow ExperienceReplay(20000)$ ; // *Buffer* de experiencias

$episode \leftarrow 0$

// Entrenar durante un máximo de 1000 episodios

**while** $episode < 1000$ **do**

  $s_0 \leftarrow E.reset()$ ; // Inicializar episodio

  $n \leftarrow OUNoise()$ ; // Inicializar proceso Ornstein-Ulhenbeck

  $done \leftarrow false$

  // 200 segundos por episodio

  **while** $E.t < 200$ *and not* $done$ **do**

    $a_t \leftarrow agent.get\_action(s_t)$

    $a_t \leftarrow n.add\_noise(a_t)$

    $s_{t+1}, r_t, done \leftarrow E.step(a_t)$

    // Añadir experiencia al buffer

    $ER.add(s_t, a_t, r_t, s_{t+1}, done)$

    // Obtener lote de 50 experiencias aleatorias del buffer

    $batch \leftarrow ER.sample\_batch(50)$

    // Actualizar las redes con el lote seleccionado

    $agent.update\_nets()$

  **end**

  $episode \leftarrow episode + 1$

**end**

**return** $agent.get\_nets()$

En cada episodio, se inicializa el entorno mediante la función $E.reset()$, la cual devuelve el estado inicial del sistema, $s_0$. En este estado inicial la bandada esta ubicada en el centro del campo, mientras que la posición del dron se asigna en forma aleatoria dentro del espacio aéreo. Por otro lado, se inicializa el proceso estocástico *Ornstein-Ulhenbeck* (variable $n$) que se usará para añadir ruido a la acción determinada por el agente. A continuación, el episodio se ejecuta durante un tiempo máximo de 200 segundos o hasta que la variable booleana *done* contenga el valor $true$, lo cual indica que el agente a logrado expulsar a todos los boids del espacio aéreo.

En cada paso del episodio, el agente devuelve la acción $a_t$ a partir del estado actual $s_t$. A esta acción, se la añade un ruido aleatorio utilizando la variable $n$, la cual mantiene el estado del proceso *Ornstein-Ulhenbeck*. Una vez obtenida la acción modificada, se ejecuta un paso de la simulación a través de la función $E.step(a_t)$. Luego se registra la experiencia en el *buffer* ER, la cual se compone del estado $s_t$, la acción ejecutada $a_t$, la recompensa obtenida $r_t$, el siguiente estado alcanzado $s_{t+1}$ y el indicador booleano de fin de episodio, *done*. Finalmente, se sortea un lote de 50 experiencias del *buffer* ER para actualizar las redes según las reglas indicadas por el algoritmo $DDPG$ (detalladas en el apéndice C). Una finalizado el proceso de entrenamiento, se devuelven las redes entrenadas por el agente.

# Referencias


[1] 3DRobotics. Iris operation manual. `https://www.drones.nl/media/files/drones/1456528063-3dr-iris-operation-user-manual-en-v6.pdf`, 2014. [Accessed 20-04-2025].

[2] A Anderson, CA Lindell, Karen M Moxcey, WF Siemer, George M Linz, PD Curtis, JE Carroll, CL Burrows, Jason R Boulanger, KMM Steensma, et al. Bird damage to select fruit crops: The cost of damage and the benefits of control in five states. *Crop Protection*, 52:103–109, 2013.

[3] arducopter.co.uk. 3dr iris quadcopter. `https://www.arducopter.co.uk/iris-quadcopter-uav.html`. [Accessed 08-05-2025].

[4] Ardupilot. Github - ardupilot/ardupilot: Arduplane, arducopter, ardurover, ardusub. `https://github.com/ArduPilot/ardupilot?tab=coc-ov-file`. [Accessed 20-04-2025].

[5] Leonard R Askham. Efficacy of methyl anthranilate as a bird repellent on cherries, blueberries and grapes. In *Proceedings of the Vertebrate Pest Conference*, volume 15, 1992.

[6] T Aubin. Synthetic bird calls and their application to scaring methods. *Ibis*, 132(2):290–299, 1990.

[7] Andrew T Baxter and John R Allan. Use of raptors to reduce scavenging bird numbers at landfill sites. *Wildlife Society Bulletin*, 34(4):1162–1168, 2006.

[8] Santosh Bhusal, Uddhav Bhattarai, and Manoj Karkee. Improving pest bird detection in a vineyard environment using super-resolution and deep learning. *IFAC-PapersOnLine*, 52(30):18–23, 2019.

[9] Santosh Bhusal, Manoj Karkee, Uddhav Bhattarai, Yaqoob Majeed, and Qin Zhang. Automated execution of a pest bird deterrence system using a programmable unmanned aerial vehicle (uav). *Computers and Electronics in Agriculture*, 198:106972, 2022.

[10] Enrique H Bucher and Rosana M Aramburú. Land-use changes and monk parakeet expansion in the pampas grasslands of argentina. *Journal of Biogeography*, 41(6):1160–1170, 2014.

[11] Enrique H Bucher, Liliana F Martin, Mónica B Martella, and Joaquín L Navarro. Social behaviour and population dynamics of the monk parakeet. In *Proc. Int. Ornithol. Congr*, volume 20, pages 681–689, 1990.

[12] Sonia B Canavelli, Rosana Aramburú, and María Elena Zaccagnini. Considerations for reducing conflicts around damage of agricultural crops by monk parakeet (myiopsitta monachus). *El Hornero*, 27(01), 2012.

[13] James C Carlson, Alan B Franklin, Doreene R Hyatt, Susan E Pettit, and George M Linz. The role of starlings in the spread of salmonella within concentrated animal feeding operations. *Journal of Applied Ecology*, 48(2):479–486, 2011.

[14] Ching-An Cheng, Andrey Kolobov, and Adith Swaminathan. Heuristic-guided reinforcement learning, 2021.

[15] Sobre Ciencia. Las pérdidas que ocasionan las aves plaga a los cultivos en uruguay. https://www.youtube.com/watch?v=ZbNUW2QBm3M&t=2s. [Accessed 30-04-2025].

[16] Michael R Conover. How birds interpret distress calls: implications for applied uses of distress call playbacks. In *Proceedings of the Vertebrate Pest Conference*, volume 16, 1994.

[17] Michael R Conover and James J Perito. Response of starlings to distress calls and predator models holding conspecific prey. *Zeitschrift für Tierpsychologie*, 57(2):163–172, 1981.

[18] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org, 2016–2021.

[19] dds foundation.org. Dds portal data distribution services. https://www.dds-foundation.org/. [Accessed 05-07-2025].

[20] Ministerio de Ganaria. Cotorra en uruguay. https://www.gub.uy/ministerio-ganaderia-agricultura-pesca/politicas-y-gestion/cotorra. [Accessed 30-04-2025].

[21] DGSA:Dirección General de Servicios Agrícolas (MGAP). Campaña de lucha contra la cotorra. http://www.chasque.net/dgsa/Operaciones/Cotorra/cartillaCOTORRA.htm#:~:text=Se%20ha%20usado%20con%20%C3%A9xito,implementar%20y%20sobre%20todo%20segura.&text=Para%20minimizar%20el%20da%C3%B1o%20producido,con%20tener%20un%20t%C3%A9cnica%20eficiente. [Accessed 03-05-2025].

[22] A.K. Dixit. *Optimization in Economic Theory*. Oxford University Press, 1990.

[23] Julie L Elser, Catherine A Lindell, Karen MM Steensma, Paul D Curtis, Deanna K Leigh, William F Siemer, JR Boulanger, and Stephanie A Shwiff. Measuring bird damage to three fruit crops: A comparison of grower and field estimates. *Crop Protection*, 123:1–4, 2019.

[24] Jonathan A Foley, Ruth DeFries, Gregory P Asner, Carol Barford, Gordon Bonan, Stephen R Carpenter, F Stuart Chapin, Michael T Coe, Gretchen C Daily, Holly K Gibbs, et al. Global consequences of land use. *science*, 309(5734):570–574, 2005.

[25] Blender Foundation. 4.0 — blender.org. https://www.blender.org/download/releases/4-0/. [Accessed 20-04-2025].

[26] Y Fukuda, CM Frampton, and GJ Hickling. Evaluation of two visual birdscarers, the peaceful pyramid® and an eye-spot balloon, in two vineyards. *New Zealand Journal of Zoology*, 35(3):217–224, 2008.

[27] gazebosim.org. Gazebo sim: User commands class reference. https://gazebosim.org/api/sim/7/classgz_1_1sim_1_1systems_1_1UserCommands.html. [Accessed 22-04-2025].

[28] gazebosim.org. Gazebo tutorials. https://gazebosim.org/docs/latest/tutorials/. [Accessed 15-06-2025].

[29] gazebosim.org. Getting Started with Gazebo garden. https://gazebosim.org/docs/garden/getstarted/. [Accessed 20-04-2025].

[30] gazebosim.org. Understanding the gui gazebo. https://gazebosim.org/docs/latest/gui/. [Accessed 14-06-2025].

[31] geeksforgeeks.org. Artificial neural networks and its applications. https://www.geeksforgeeks.org/artificial-intelligence/artificial-neural-networks-and-its-applications/. [Accessed 06-08-2025].

[32] W. P. Gorenzel, B. F. Blackwell, G. D. Simmons, T. P. Salmon, and R. A. Dolbeer and. Evaluation of lasers to disperse american crows, corvus brachyrhynchos, from urban night roosts. *International Journal of Pest Management*, 48(4):327–331, 2002.

[33] Andrea S Griffin. Social learning about predators: a review and prospectus. *Learning & Behavior*, 32(1):131–140, 2004.

[34] Andrea S Griffin, Hayley M Boyce, and Geoff R MacFarlane. Social learning about places: observers may need to detect both social alarm and its cause to learn. *Animal Behaviour*, 79(2):459–465, 2010.

[35] Heath M Hagy, George M Linz, and William J Bleier. Optimizing the use of decoy plots for blackbird control in commercial sunflower. *Crop Protection*, 27(11):1442–1447, 2008.

[36] Hanno Hildenbrandt, Claudio Carere, and Charlotte Hemelrijk. Self-organized aerial displays of thousands of starlings: a model. *Behavioral Ecology*, 21:1349–1359, 01 2010.

[37] Rama Carl Hoetzlein. Flock2: A model for orientation-based social flocking. *Journal of Theoretical Biology*, 593:111880, 2024.

[38] Imran Mohd Hornain and Nik Fadzly N Rosely. Evaluating drones as bird deterrents in industrial environments: multirotor vs fixed-wing efficacy. *Bulletin of Electrical Engineering and Informatics*, 13(6):3960–3967, 2024.

[39] Sinan Ibrahim, Mostafa Mostafa, Ali Jnadi, Hadi Salloum, and Pavel Osinenko. Comprehensive overview of reward engineering and shaping in advancing reinforcement learning applications. *IEEE Access*, 12:175473–175500, 2024.

[40] Michele T Jay-Russell. What is the risk from wild animals in food-borne pathogen contamination of plants? *CABI Reviews*, pages 1–16, 2014.

[41] L.E. Kavraki, P. Svestka, J.-C. Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[42] Brad Knudsen. Flying a drone in the wind, cold, and other challenging environments. https://scanifly.com/blog/flying-a-drone-in-the-wind-cold-and-other-challenging-environments/#:~:text=If%20you're%20just%20beginning,as%20high%20as%2020%20mph. [Accessed 08-05-2025].

[43] Sara M Kross, Jason M Tylianakis, and Ximena J Nelson. Effects of introducing threatened falcons into vineyards on abundance of passeriformes and bird damage to grapes. *Conservation biology*, 26(1):142–149, 2012.

[44] TP Lillicrap. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[45] George M Linz, Enrique H Bucher, Sonia B Canavelli, Ethel Rodriguez, and Michael L Avery. Limitations of population suppression for protecting crops from bird depredation: A review. *Crop Protection*, 76:46–52, 2015.

[46] Tyson Macaulay. Chapter 12 - threats and impacts to the iot, section emergent behavior threats. In Tyson Macaulay, editor, *RIoT Control*, pages 221–278. Morgan Kaufmann, Boston, 2017.

[47] Robert D. Magrath, Tonya M. Haff, Pamela M. Fallow, and Andrew N. Radford. Alarming features: birds use specific acoustic properties to identify heterospecific alarm calls. *Proceedings of the Royal Society B: Biological Sciences*, 279(1746):4176–4184, 2012.

[48] Robert D. Magrath, Tonya M. Haff, Pamela M. Fallow, and Andrew N. Radford. Birds learn socially to recognize heterospecific alarm calls by acoustic association. *Current Biology*, 28(17):2632–2637.e3, 2018.

[49] Guillaume Matheron, Nicolas Perrin, and Olivier Sigaud. The problem with ddpg: understanding failures in deterministic environments with sparse rewards. *arXiv preprint arXiv:1911.11679*, 2019.

[50] Barry J McMahon, Beatriz Arroyo, Nils Bunnefeld, Martina Carrete, Francis Daunt, and Juliette C Young. Birds and people: from conflict to coexistence. *Ibis*, 166(1):23–37, 2024.

[51] MGAP. Guía de buenas prácticas para el manejo de cotorras en cultivos. `https://www.gub.uy/ministerio-ganaderia-agricultura-pesca/sites/ministerio-ganaderia-agricultura-pesca/files/2021-02/Gu%C3%ADa%20de%20buenas%20practicas%20para%20el%20manejo%20de%20cotorras%20fin_0.pdf`. [Accessed 03-05-2025].

[52] Volodymyr Mnih. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[53] A. Ng, Daishi Harada, and Stuart J. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *International Conference on Machine Learning*, 1999.

[54] Nvidia. Isaac sim. `https://developer.nvidia.com/isaac/sim`. [Accessed 04-06-2025].

[55] Jacopo Panerati, Hehui Zheng, SiQi Zhou, James Xu, Amanda Prorok, and Angela P. Schoellig. Learning to fly—a gym environment with pybullet physics for reinforcement learning of multi-agent quadcopter control. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021.

[56] David Pimentel, Rodolfo Zuniga, and Doug Morrison. Update on the environmental and economic costs associated with alien-invasive species in the united states. *Ecological economics*, 52(3):273–288, 2005.

[57] pypi.org. Mavproxy 1.8.71. `https://pypi.org/project/MAVProxy/`. [Accessed 05-07-2025].

[58] Craig W. Reynolds. Flocks, herds, and schools: a distributed behavioral model. *Seminal graphics: pioneering efforts that shaped the field*, 1987.

[59] Raoul FH Ribot, Mathew L Berg, Katherine L Buchanan, and Andrew TD Bennett. Fruitful use of bioacoustic alarm stimuli as a deterrent for crimson rosellas (platycercus elegans). *Emu-Austral Ornithology*, 111(4):360–367, 2011.

[60] Paula Rivadeneira, Sara Kross, Nora Navarro-Gonzalez, and Michele Jay-Russell. A review of bird deterrents used in agriculture. In *Proceedings of the vertebrate Pest conference*, volume 28, 2018.

[61] ros.org. Ros 2 documentation: Humble. `https://docs.ros.org/en/humble/index.html`. [Accessed 20-04-2025].

[62] ros.org. Rviz2 rviz 1.9.0. `https://docs.ros.org/en/rolling/p/rviz2/`. [Accessed 28-06-2025].

[63] ros.org. Tutorials ros 2 documentation: Humble. `https://docs.ros.org/en/humble/Tutorials.html`. [Accessed 14-06-2025].

[64] ros.org. Understanding topics: Ros 2. https://docs.ros.org/en/humble/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Topics/Understanding-ROS2-Topics.html. [Accessed 14-06-2025].

[65] ros.org. Writing a simple publisher and subscriber (python) ros 2 humble. https://docs.ros.org/en/humble/Tutorials/Beginner-Client-Libraries/Writing-A-Simple-Py-Publisher-And-Subscriber.html. [Accessed 14-06-2025].

[66] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[67] sdformat.org. Sdformat specification. http://sdformat.org/spec. [Accessed 14-06-2025].

[68] Michael D. Sorenson, Mark E. Hauber, Diane Colombelli-Négrel, and Sonia Kleindorfer. Referential signaling in a communally breeding bird. *Proceedings of the National Academy of Sciences*, 119(38):e2222008120, 2022.

[69] D Strömbom, RP Mann, AM Wilson, S Hailes, AJ Morton, DJT Sumpter, and AJ King. Solving the shepherding problem: heuristics for herding autonomous, interacting agents. jr soc. interf. 11 (100)(2014).

[70] Richard S Sutton and Andrew Barto. *Reinforcement learning : an introduction*. The Mit Press, 2018.

[71] Toshitaka N. Suzuki. Semantic communication in birds: evidence from field research. *Ecological Research*, 31(3):307–319, 2016.

[72] John P Swaddle, Dana L Moseley, Mark K Hinders, and Elizabeth P. Smith. A sonic net excludes birds from an airfield: implications for reducing bird strike and crop losses. *Ecological Applications*, 26(2):339–345, 2016.

[73] Vision Team. What is a stereo vision camera? https://www.e-consystems.com/blog/camera/technology/what-is-a-stereo-vision-camera-2/. [Accessed 05-05-2025].

[74] John Bomford Tracey, Brown Mary, Quentin Hart, Glen Saunders, and Ron Sinclair. Managing bird damage to fruit and other horticultural crops. *Australian Goverment*, 2007.

[75] John Paul Tracey and Glen Saunders. *Bird damage to the wine grape industry*. Vertebrate Pest Research Unit, NSW Agriculture Orange, 2003.

[76] Joshua Trethowan, Zihao Wang, and K. C. Wong. The viability of a grid of autonomous ground-tethered uav platforms in agricultural pest bird control. *Machines*, 11(3), 2023.

[77] G. E. Uhlenbeck and L. S. Ornstein. On the theory of the brownian motion. *Phys. Rev.*, 36:823–841, Sep 1930.

[78] Z Wang and KC Wong. Autonomous bird deterrent system for vineyards using multiple bio-inspired unmanned aerial vehicle. In *Proc. 10th Int. Micro Air Vehicle Conf. Competition (IMAV)*, volume 2018, pages 256–281, 2018.

[79] Zihao Wang, Darren Fahey, Andrew Lucas, Andrea S Griffin, Gregory Chamitoff, and KC Wong. Bird damage management in vineyards: Comparing efficacy of a bird psychology-incorporated unmanned aerial vehicle system with netting and visual scaring. *Crop Protection*, 137:105260, 2020.

[80] Zihao Wang, Andrea S. Griffin, Andrew Lucas, and K.C. Wong. Psychological warfare in vineyard: Using drones and bird psychology to control bird damage to wine grapes. *Crop Protection*, 120:163–170, 2019.

[81] Zihao Wang and KC Wong. Autonomous pest bird deterring for agricultural crops using teams of unmanned aerial vehicles. In *2019 12th Asian Control Conference (ASCC)*, pages 108–113. IEEE, 2019.

[82] C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Oxford, 1989.

[83] Wikipedia. Conversion between quaternions and Euler angles - Wikipedia — en.wikipedia.org. `https://en.wikipedia.org/wiki/Conversion_between_quaternions_and_Euler_angles`. [Accessed 20-04-2025].

[84] Botian Xu, Feng Gao, Chao Yu, Ruize Zhang, Yi Wu, and Yu Wang. Omnidrones: An efficient and flexible platform for reinforcement learning in drone control, 2023.

[85] Jixuan Zhi and Jyh-Ming Lien. Learning to herd agents amongst obstacles: Training robust shepherding behaviors using deep reinforcement learning, 2020.

Esta es la última página.
Compilado el martes 19 agosto, 2025.
http://iie.fing.edu.uy/