Centro de Investigación Científica y de Educación Superior de Ensenada, Baja California

Maestría en Ciencias en Electrónica y Telecomunicaciones con orientación en Control e Instrumentación

---

Implementación de modelos presa-depredador en enjambres de robots móviles

Tesis

para cubrir parcialmente los requisitos necesarios para obtener el grado de Maestro en Ciencias

Presenta:

Lizbeth Carrasco Gutiérrez

Ensenada, Baja California, México

2021

Tesis defendida por

Lizbeth Carrasco Gutiérrez

y aprobada por el siguiente Comité

Dr. César Cruz Hernández
Codirector de tesis

Dr. Rigoberto Martínez Clark
Codirector de tesis

Dr. Javier Pliego Jiménez

Dr. Miguel Angel Murillo Escobar

Dra. Beatriz Cordero Esquivel

Dra. María del Carmen Maya Sánchez
Coordinador del Posgrado en Electrónica y Telecomunicaciones

Dr. Pedro Negrete Regagnon
Director de Estudios de Posgrado

Resumen de la tesis que presenta Lizbeth Carrasco Gutiérrez como requisito parcial para la obtención del grado de Maestro en Ciencias en Electrónica y Telecomunicaciones con orientación en Control e Instrumentación.

Implementación de modelos presa-depredador en enjambres de robots móviles

Resumen aprobado por:

Dr. César Cruz Hernández	Dr. Rigoberto Martínez Clark
Codirector de tesis	Codirector de tesis

La relación presa-depredador corresponde a un sistema dinámico natural con muchas variables. Por años, la comunidad científica ha volcado sus esfuerzos para identificar las reglas que la componen y rigen, y basados en los estudios de diferentes interacciones entre especies se han elaborado algunos modelos matemáticos para esta relación. Entre los más citados se encuentra el modelo Lotka-Volterra. En este trabajo de tesis, se utilizan algunas modificaciones de este modelo para construir algoritmos por medio de máquinas de estados finitos, con el fin de reproducir este comportamiento natural en un grupo de robots móviles de bajo costo. El objetivo principal es observar el efecto del desplazamiento de los individuos en el comportamiento de las poblaciones al agregar la competencia intraespecífica a las reglas de interacción y confirmar la presencia de caos. La implementación y eficacia de los algoritmos propuestos son puesto a prueba mediante los softwares de simulación V-Rep y MATLAB.

Palabras clave: Modelo Presa-Depredador, Robots móviles, Control bio-inspirado, Máquina de estados finitos, Sistemas complejos.

Abstract of the thesis presented by Lizbeth Carrasco Gutiérrez as a partial requirement to obtain the Master of Science degree in Electronics and Telecommunications with orientation in Control and Instrumentation.

Implementation of prey-predator models in swarms of mobile robots

Abstract approved by:

Dr. César Cruz Hernández
Thesis Co-Director

Dr. Rigoberto Martínez Clark
Thesis Co-Director

The prey-predator relationship corresponds to a natural dynamic system with many variables. For years, the scientific community has turned its efforts to identify the rules that compose and govern it, and based on studies of different interactions between species, some mathematical models have been developed for this relationship. Among the most cited is the Lotka-Volterra model. In this thesis work, some modifications of this model are used to build algorithms by means of finite state machines, in order to reproduce this natural behavior in a group of low-cost mobile robots. The main objective is to observe the effect of the displacement of individuals on the behavior of populations by adding instraspecific competition to the interaction rules, and confirm the presence of chaos. The implementation and efficacy of the proposed algorithms are tested by using V-REP and MATLAB simulation software.

Keywords: Prey-Predator model, Mobile robots, Bio-inspired control, Finite state machine, Complex systems.

Dedicatoria

A mi madre, por estar presente para darme el impulso que necesitaba para alentarme a concluir con esta parte de mi vida.

A mi hermano, mi padre y a "P.", por ser parte de mi vida.

A mis profesores por su tiempo y su paciencia.

Agradecimientos

Al Centro de Investigación Científica y de Educación Superior de Ensenada por la instrucción recibida y el uso de sus instalaciones en beneficio de mi formación académica.

Al Consejo Nacional de Ciencia y Tecnología (CONACyT) por brindarme el apoyo económico para realizar mis estudios de maestría. No. de becario: **831766**

Al Consejo Nacional de Ciencia y Tecnología (CONACYT) por el apoyo económico recibido a través del proyecto de Investigación en ciencia básica entre instituciones, “Sincronización de Sistemas Complejos y Algunas Aplicaciones”. Ref. 166654 y continuación (A1-S-31628).

Tabla de contenido

Página

Tabla de contenido (continuación)

Lista de figuras

Figura Página

Lista de figuras (continuación)

Lista de figuras (continuación)

Figura Página

Lista de tablas

Capítulo 1. Introducción

Cabezas Venegas (2017), describe a los organismos como “sistemas abiertos que continuamente se encuentran interactuando con todo lo que compone el ambiente que lo rodea” (p.11), y procede a exponer información interesante:

- En primer lugar el autor describe que en un escenario ecológico tienen lugar numerosas interacciones, las cuales se desarrollan entre los individuos de diferentes especies y aquellos que pertenecen a la misma. De igual forma, entre los participantes de dichas interacciones se puede incluir a los factores físico-químicos, que están presentes en el medio.
- En el texto también se explica que una comunidad biológica podría definirse como una congregación de poblaciones de distintas especies, las cuales se establecen cerca unas de otras, permitiendo una interacción frecuente (p.62).

El autor destaca a la competencia, la depredación y la simbiosis como las interacciones mas importantes presentes dentro de una comunidad. Ya que contribuyen a controlar las poblaciones y al equilibrio entre los recursos y la cantidad de consumidores. Debido a que afectan elementos como la supervivencia y la reproducción, participando en el proceso de selección natural.

Cabezas Venegas (2017) expresa, además, que si se considera que en las interacciones anteriormente mencionadas se ven involucradas dos especies, dichas interacciones pueden definirse si se considera la forma en que los participantes se ven afectados, ya sea de forma positiva (+) o negativa (-), lo anterior puede comprenderse mejor con el apoyo de la tabla 1 (pp. 62, 63).

Con relación a lo mencionado anteriormente, nuestra investigación se enfoca en la interacción denominada como depredación, la cual, como se describe en la tabla 1, involucra el beneficio de un individuo a expensas de otro que resulta perjudicado. Ejemplos de lo anterior pueden ser vistos en numerosos escenarios en la naturaleza. Por ejemplo, un ave cazando insectos para alimentarse, las leonas trabajando en equipo para dar caza a una gacela o un grupo de orcas que efectúan estrategias para llevar a cabo la caza de sus presas. Este último caso se explica con más detalle en el trabajo de Smith *et al.* (1981).

Tabla 1. *Interacciones entre organismos*. Adaptado de *Biología la vida en la Tierra (Audesirk* et al.*, 2008) (p.538).* Se indica con (+) cuando una especie se beneficia de la relación, con (-) cuando resulta perjudicada y con (0) cuando la relación resulta indiferente. (Cabezas Venegas, 2017, p.63).

Tipo de interacción		Efecto sobre organismo A	Efecto sobre organismo B
Competencia entre A y B		-	-
Depredación de A contra B		+	-
Simbiosis	**Parasitismo** de A en B	+	-
	Comensalismo de A con B	+	0
	Mutualismo entre A y B	+	+

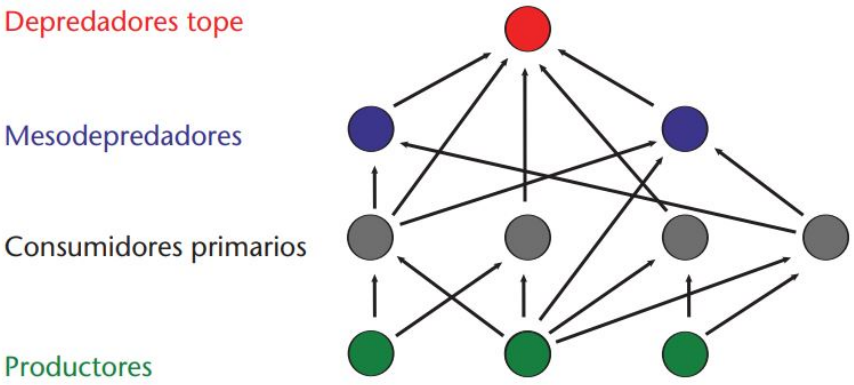


Figura 1. En esta red o trama trófica compleja, Di Bitetti (2008) indica que “los círculos representan distintas especies y las flechas indican las relaciones tróficas que mantienen entre ellas. La altura indica el nivel trófico ocupado por cada especie que interviene en esta trama.” (p.34)

Ferreras (2015) describe en su trabajo a la depredación como un proceso en el que un individuo (el depredador), se alimenta de otro (la presa), el cual inicialmente se encuentra con vida y puede ser consumido de forma parcial o completamente (p.2).

En López Bravo *et al.* (2015) se menciona que a pesar del efecto perjudicial que las acciones del grupo depredador tienen sobre el crecimiento de las poblaciones de las cuales se alimenta, dichas acciones además de fortalecer a la población depredada (eliminando a los individuos débiles o enfermos), previene eventos que pueden resultar perjudiciales tanto para los depredadores como para las presas mismas, como sería la sobre-población. Lo cual podría conducir a la extinción de la población de las presas. Es por ello que resulta importante para ambos bandos, el mantener el equilibrio presente en su peculiar interacción, pues solo de esa forma ambos grupos podrán sobrevivir (pp. 55, 56).

La observación de Di Bitetti (2008) coincide con lo descrito anteriormente, ya que menciona que el papel de los depredadores (ó carnívoros), es de suma importancia dentro de los ecosistemas naturales, por lo que su ausencia puede resultar de forma similar a lo descrito en López Bravo *et al.* (2015) a la extinción de otras especies, esto debido a un “efecto dominó”, al cual también se le puede denominar como “cascada trófica” (p.33).

El término “*cascada trófica*” es definido en el trabajo de Di Bitetti (2008) (pp.33-34) como un conjunto de palabras, las cuales hace referencia a la relación “causa efecto” resultado de las interacciones que tienen lugar entre “aquellos que consumen” y “los que son consumidos”, (*trófico= nutrición o alimento*). Dentro de dicha interacción, por supuesto, existen distintos niveles, los cuales varían de acuerdo al tipo de alimentación de los individuos. Y se resume en la figura 1, la cual es un diagrama presente en el trabajo de Di Bitetti (2008). De arriba hacia abajo, se encuentran los depredadores tope (quienes obtienen su alimento del grupo de meso depredadores y el de los herbívoros/consumidores primarios), los depredadores secundarios o meso depredadores (cuya fuente de alimento principal son los consumidores primarios), los consumidores primarios (cuyo sustento son los productores primarios) y al final los productores (grupo conformado por plantas o vegetales, este puede ser considerado como el más básico).

Di Bitetti (2008) comenta que en la naturaleza, “las cadenas lineales”, que consisten en interacciones de una o pocas especies que solamente involucran a los niveles inmediatos superior o inferior, es un evento que no ocurre con normalidad. Las especies omnívoras, quienes se caracterizan por alimentarse tanto de plantas como de animales de otros niveles (consumidores primarios o pequeños depredadores), pueden encontrarse con regularidad en las comunidades ecológicas. De forma similar, los depredadores tope obtienen sus alimentos de los niveles inferiores. Es debido a lo anterior que las cadenas tróficas son en realidad, redes complejas, similares a lo representado en la figura 1. Cuya característica principal es que las modificaciones en la población o en las acciones dentro de alguno de los niveles tienen un efecto en los demás, lo cual puede ir incluso mas allá de los niveles inmediatos. El autor menciona, como ejemplo de esto último, que una variación en la población de carnívoros puede repercutir en la vegetación misma.

Para aclarar un poco más este tipo de interacción, se hace alusión a un caso expuesto dentro del documento ya citado de Di Bitetti (2008) (pp. 34-36), en este se expresa que de 1960 a 1980, los ecólogos discutieron acerca de cual era el elemento responsable de modificar la diversidad biológica y la densidad de población de las especies presentes en las cascadas tróficas de un ecosistema. Un grupo sostenía que la diversidad y la cantidad de niveles tróficos que componen un ecosistema varia en función de la energía presente en el sistema, ya que la disponibilidad de nutrientes facilita el convertir dicha energía en biomasa. Lo que se traduce en una regulación de un nivel inferior a uno superior (de abajo hacia arriba). Los consumidores primarios consumen dicha energía y luego la transmiten a los niveles superiores al ser posteriormente devorados y así sucesivamente.

Los que se oponían a esa idea, argumentaban que las interacciones presentes entre los seres vivos eran las que afectaban la diversidad y la densidad de las poblaciones presentes en la naturaleza, (regulación de arriba hacia abajo). Basaban su argumento en la idea de que son las acciones de los depredadores las que provocan el cambio en la biodiversidad, regulan las poblaciones depredadas, por ejemplo los herbívoros, y gracias al efecto dominó esto afecta a su vez a los niveles inferiores, en este caso la población de las plantas. Analizando lo anterior, se puede apreciar que, gracias a las propiedades de la cascada trófica, el grupo depredador puede tener influencia

en aquellas poblaciones que no interactúen de forma directa con ellos. Sin embargo, Di Bitetti señala que, con el pasar del tiempo, se determinó que ambas ideas eran correctas.

Di Bitetti (2008) menciona que dicho desacuerdo pudo verse influenciado por lo que ocurre en la actualidad, con ello se refiere a la influencia que tienen las acciones del ser humano sobre los ecosistemas, provocando la extinción o la desaparición de grupos depredadores en ecosistemas naturales y artificiales (regulación de abajo hacia arriba). Seguidamente, se procede a describir el caso en el que un sistema ubicado en la costa pacifico norte de Norteamericana, en el que se podía encontrar a las nutrias (*Enhydra lutris*) en el papel del depredador, cuya principal fuente de alimento son seres invertebrados, como el erizo de mar, quienes a su vez, se alimentan de un alga conocida como *kelp* (también denominada como *cachiyuyo* por el autor), la cual es muy voluminosa y capaz de formar grupos densos, los cuales se denominan “bosques” en el fondo del escenario.

Di Bitteti explica que, durante una época, los depredadores (las nutrias), fueron perseguidas por el hombre, de tal forma que su población se vio seriamente perjudicada, decreciendo de forma drástica. Lo anterior provocó que el número de erizos prosperara, y al alimentarse mayormente de algas, estas también decrecieron en número hasta casi desaparecer. Sin embargo, algún tiempo después, se estableció una prohibición en la caza de las nutrias, lo cual permitió que el mamífero pudiera recolónizar su territorio. La serie de sucesos descrita hasta el momento, puso a prueba lo expuesto en la hipótesis de las cascadas tróficas y la regulación de arriba hacia abajo, pues gracias a la reaparición del grupo depredador, los erizos fueron atacados nuevamente, reduciendo su número. Por lo que las formaciones de algas tuvieron oportunidad de prosperar. Con esto se puede apreciar que el depredador (la nutria en este ejemplo), puede tener una enorme influencia en la dinámica de su ecosistema.

Sin embargo, la historia expuesta por Di Bitteti continua, añadiendo que, a finales de los años ochenta, la población de las nutrias volvió a sufrir bajas significativas. Luego de algunas investigaciones, cuyo propósito era encontrar el motivo de semejantes perturbaciones, se determinó con base a la evidencia, que el decremento observado en la población de las nutrias era causado por la adición al ecosistema de un nuevo depredador tope, el cual era la orca (*Orcinus orca*). A pesar de que las focas y los lobos

marinos constituyen su fuente principal de alimento, la orca, al darse cuenta de que la población de sus presas predilectas ya no es suficiente para satisfacer sus necesidades (se contempla a la pesca comercial, los cambios climáticos y otra serie de factores externos, como los culpables del decremento poblacional de dichas presas), se vio orillada a buscar fuentes de alimento alternativas, lo cual la llevó al escenario habitado por la nutria. No obstante, para fin de que una orca pueda subsistir alimentándose exclusivamente de nutrias, se calculó que requeriría un total aproximado de 1825 por año, por lo que se determinó que el decremento poblacional de las nutrias podría ser el resultado de la depredación de tres o cuatro orcas.

La adición del nuevo depredador tope y el decremento en la población de nutrias fueron factores que permitieron que los erizos retomaran el dominio del fondo marino y a su vez las algas disminuyeran en número. El escenario descrito, (a pesar de mostrar una leve influencia por parte del ser humano), contribuye a la deducción de que se trata de un ecosistema regulado de arriba hacia abajo y, además, sirve como ejemplo de lo que es una cascada trófica. Siguiendo esa línea de pensamiento, en el primer nivel se encontraría la orca (depredador tope), enseguida en el área de los mesodepredadores se encontraría la nutria, en los consumidores primarios se ubicaría al erizo de mar y al final, el área de los productores pertenecería a las algas/kelp.

1.1. Antecedentes

En Francesca y Birattari (2016) (pp. 1, 2), se describe un enjambre robótico como un grupo de robots cuyo propósito consiste en llevar a cabo una tarea o misión que está más allá de las habilidades de un solo individuo robótico. En este caso el grupo coopera y se organiza automáticamente. Esto implica que no es necesaria la presencia de un líder y la coordinación del grupo se obtiene gracias a la interacción de los miembros del grupo.

Además en el trabajo de Francesca y Birattari (2016) se añade que un enjambre no requiere de una infraestructura, pues cada individuo robótico se desenvuelve en base a los datos recolectados por sus sensores o la información que le es proporcionada

durante la comunicación con sus vecinos.

No obstante, en dicho texto también se comenta que el diseño del software de control para enjambres robóticos es un aspecto que se encuentra aún en una etapa temprana, Brambilla *et al.* (2013). Además de señalar que usualmente los diseñadores trabajan mediante el ensayo y el error. Pero desafortunadamente los resultados obtenidos hasta el momento no pueden ser aplicados como una solución general.

La aproximación usual para la creación de un software de control para los enjambres robóticos, según Francesca y Birattari (2016), es mediante el uso de métodos automáticos, ya que con ello el problema se aborda como una cuestión de optimización. Sin embargo, gran parte de los métodos automáticos sugeridos hasta ahora son del tipo robótica evolutiva (Nolfi *et al.*, 2000), esto consiste en software de control basados en redes neuronales artificiales, que son optimizadas de forma automática gracias a algoritmos evolutivos, los cuales basan su funcionamiento en procesos diseñados con base en la evolución natural. Este enfoque ha sido utilizado satisfactoriamente en el diseño de enjambres robóticos que sean capaces de cumplir con numerosas misiones y tareas. Por lo que los autores sugieren que este tipo de diseño es viable y prometedor.

En el trabajo de Francesca y Birattari (2016) (pp.2,4) se explica además que el diseño automático puede llevarse a cabo mediante dos tácticas, el método *off-line* y el *on-line*. El proceso de diseño *off-line* inicia y concluye antes de la activación del enjambre robótico. Esto puede realizarse con una simulación computacional y cuenta con algunos beneficios, entre los cuales se pueden destacar: Permite agilizar las evaluaciones ya que son más breves que las del tipo real y cuando se recurre a este tipo de métodos se pueden evitar la creación de daños en el equipo.

El método *on-line* por otra parte, es un método que se ejecuta con el enjambre robótico en un estado funcional. La ventaja que esto ofrece es que se dispone de información actualizada del ambiente donde se despliega el grupo robótico, lo cual no es posible con el otro método *off-line*. Por tanto, se puede decir que esta opción pretende ofrecer un diseño mejor adaptado a la misión o tarea que se pretende cumplir. Sin embargo, debido a que el proceso se lleva a cabo haciendo uso de los robots mientras están activos, el tiempo y los recursos computacionales son limitados, además de que se debe tener cuidado con aquellos diseños que puedan ser peligrosos para el equipo

y en consecuencia para la misión, pues el peligro de falla es un factor a tener en cuenta. Estos aspectos provocan que el área de búsqueda sea reducido a comparación del método *off-line*.

Además de lo anterior, una cuestión interesante del documento elaborado por Francesca y Birattari (2016) (pp. 5-6), es que describe algunos avances y experimentos elaborados, por ejemplo el trabajo de Wischmann *et al.* (2007), quienes realizaron un estudio de la relación entre el aprendizaje y la evolución añadidas a un escenario del tipo *depredador-presa*, pero se aclara que las pruebas fueron elaboradas únicamente en un ambiente de simulación. Por otra parte, Elfwing *et al.* (2011), analizaron la combinación del aprendizaje reforzado y la evolución integrados en un escenario de supervivencia. En este caso, a diferencia del experimento anterior, las pruebas realizadas fueron ejecutadas primeramente mediante simulaciones, donde se utilizó un grupo de cuatro y luego dos robots. Seguidamente, el software obtenido en dicha simulación se puso a prueba en un equipo físico, compuesto por dos robots Cyber Rodent. Luego, el artículo procede a exponer que, en algunos casos, la investigación no se centra del todo en las cuestiones del diseño de los métodos automáticos y se inclina mas hacia otros aspectos que, a pesar de ser interesantes (como es el caso de la factibilidad del diseño de un modelo que replique un modelo biológico o el estudio del comportamiento de un grupo animal), pueden distraer la atención de los investigadores a otras cuestiones igualmente interesantes y que podrían ayudar a la mejora del software de control de los enjambres biológicos, como son por ejemplo: ¿Cuál método automático es el mejor en determinadas circunstancias y condiciones?, ¿Cómo se desempeña en otras tareas además de para la que fue pensado?, ¿Qué aspectos pueden representar un problema?, etc. En este caso los autores exponen que preguntas como las anteriores son importantes y deben realizarse para fin de llevar a cabo la evaluación y comparación de los métodos.

Francesca y Birattari (2016) (pp. 2,6,8) expresan que los logros alcanzados hasta el momento, mas que aportaciones que favorezcan el crecimiento del conocimiento ya adquirido, pueden considerarse como contribuciones aisladas. ¿La razón?, en el artículo se expone que en la mayoría de los casos, las investigaciones que tienen como objetivo la creación de nuevos métodos e ideas de diseño automático, carecen de una comparativa con trabajos anteriores. Y esto perjudica el avance en el diseño de

software de control para enjambres robóticos. A fin de proponer una solución a dicha problemática los autores señalan algunos puntos que deberían aplicarse al momento de diseñar un software de control para enjambres de robots. En primera se sugiere que las investigaciones que tengan como finalidad la proposición o la aplicación de un método de diseño automático debe contar con un modelo de referencia para la plataforma robótica que se contemple utilizar, el método propuesto debe ser descrito con precisión en todas sus partes y parámetros. De igual forma se recomienda el establecer un proceso estándar para el estudio de los nuevos métodos e ideas y, por último, se alienta al uso de robots en los experimentos ya que los autores los consideran un elemento esencial en esta área de estudio. Basado en lo anterior, el artículo añade que, para fin de realizar una buena comparativa en los métodos de diseño es necesario tomar en cuenta todos los elementos que son utilizados durante el estudio. Con esto último también se hace mención a aspectos como el simulador, la simulación misma, el tipo de CPU, sistema operativo, etc. Esto con el fin de que sea posible la elaboración de un “modelo de referencia” que pueda ser de utilidad para trabajos y comparativas posteriores.

En el trabajo de Francesca y Birattari (2016) (p.6) también se menciona que durante el análisis de algunos artículos, notaron que en la mayoría de los casos las investigaciones tenían como objetivo la modificación de un método de diseño ya existente o la creación de uno nuevo. Y consideraban difícil encontrar más de un trabajo en el que un método de diseño automático se pusiera a prueba (sin modificar) en distintas circunstancias y cumpliendo distintas tareas. En la opinión de los autores, un modelo que requiera de modificaciones y/o ajustes a fin de poder cumplir con más de una tarea, no puede ser considerado como un método automático.

El trabajo realizado por Zhong *et al.* (2018) (pp. 160-164.) consiste en un experimento que se llevó a cabo dentro del ambiente de un simulador (V-REP) y con equipo físico. En ambos escenarios se utilizó un grupo de 10 kilobots, a los cuales se programó de tal forma que a uno se le asignó el papel del “objetivo” y el resto del grupo debía buscarlo por la zona asignada para el experimento, mientras hacen uso de sus habilidades como evadir y rodear, esto se aplica tanto a los obstáculos como a sus compañeros. Al dar con su objetivo, quien permanece inmóvil durante todo el ejercicio, el grupo lo rodea, dando por concluida la prueba. Al realizar la comparativa de

los resultados obtenidos, se apreció un mejor desempeño del grupo en el ambiente virtual que en el espacio físico. Ya que los kilobots de la simulación tuvieron un índice de éxito mayor en el encuentro de su objetivo, su tiempo fue mejor y una menor cantidad de los kilobots buscadores perdió la conexión durante el objetivo a diferencia de la contraparte con el equipo real.

Por último, se hace una mención al trabajo de Pelkonen (2018), el cual consiste en la elaboración de un modelo que sea capaz de recrear el comportamiento del sistema inmunológico humano utilizando kilobots. En este caso el equipo que se utilizó consiste en la combinación de dos herramientas computacionales, las cuales son el V-REP y el programa MATLAB. El comportamiento es descrito en dos partes “Caminata aleatoria y quimiotaxis. Los kilobots se modelan como células inmunes y el entorno es la concentración de antígenos, donde la alta concentración son los patógenos. El objetivo de los kilobots es buscar y destruir dichos patógenos en el área.”(p.3).

En este caso, se hace énfasis en la aplicación de los programas computacionales y la forma en la que fueron utilizados para la ejecución del experimento, los autores describen que el programa de MATLAB se comunica con los kilobots a fin de brindarles el valor de la concentración de su entorno, al mismo tiempo que registra la ubicación del mismo en el simulador V-REP. La programación y ejecución del código del kilobot se efectúan dentro del simulador V-REP, el cual ofrece un entorno que permite un ambiente lo más cercano posible a la realidad, dando la posibilidad de que los kilobots puedan chocar, caer, etc. El diseño del objetivo a destruir se efectúa dentro del entorno de MATLAB, lo cual implica la concentración y ubicación de los mismos. Cuando se efectúa la simulación, ambos simuladores se comunican de forma que tanto la ejecución de los kilobots y el entorno funcionen al mismo tiempo, pero MATLAB es el encargado de indicar siempre el inicio y la conclusión de la simulación.

Se ha decidido hacer mención de estas investigaciones debido a su relevancia con el proyecto de este trabajo de tesis. De Francesca y Birattari (2016), se destacan las observaciones y cuestionamientos necesarios para llevar a cabo un modelo que pueda ser de utilidad en el futuro a fin de poder alcanzar el siguiente paso hacia una mejora de los resultados obtenidos. El trabajo llevado a cabo por Zhong *et al.* (2018), permite esbozar las diferencias que pueden estar presentes en una investigación que involucra el uso del simulador V-REP al compararse con los resultados obtenidos con un equipo

real. Así como también el uso de un algoritmo cuya misión es el trabajo en equipo de sus individuos para dar búsqueda y alcance a su objetivo. Y por último, el trabajo realizado por Pelkonen (2018) sirve como referencia para el uso eficiente del simulador V-REP y el programa MATLAB, a fin de crear un escenario que pueda ser interactivo con elementos como los kilobots.

1.2. Modelo Lotka-Volterra

Según Oganician *et al.* (2017), el modelo depredador-presa básico de Lotka-Volterra se define como:

$$\frac{dP}{dt} = r_1 P - \alpha_1 PD, \qquad \frac{dD}{dt} = \alpha_2 PD - r_2 D \tag{1}$$

donde D representa a la población de depredadores, P es la población de presas y los parámetros son constantes positivas que representan (p. 7):

- r_1 : tasa de crecimiento de las presas.
- α_1: éxito en la caza del depredador, que afecta a la presa.
- r_2: tasa de crecimiento de los depredadores.
- α_2: éxito en la caza, que afecta al depredador.

En este modelo, explican los autores, el incremento que posee una componente maltusiana, $r_1 P$, la cual es proporcional al volumen de la población, y cuenta también con otro término, $\alpha_1 PD$, el cual depende de la cantidad de encuentros efectuados entre las poblaciones presa-depredador. Cuando este último término crece, afecta de forma negativa a P, reduciendo su incremento.

En el artículo también se expone que, en las ecuaciones (1), es posible observar que la ausencia de depredadores permite que la población de presas crezca de forma exponencial, mas sin embargo, si llegase a ocurrir lo contrario y las presas fueran escasas o desaparecieran, el grupo depredador se vería gravemente afectado y por

ende, su población decrecería, siguiendo de esa forma un modelo maltusiano. Por tanto, se puede deducir que a la población de depredadores le beneficia la presencia de un grupo abundante de presas, y de igual forma se ve perjudicado si su fuente de alimento es insuficiente. Por otra parte, un número pequeño de depredadores es de gran beneficio para las presas, pues permite que su población pueda incrementar. De esta forma, las interacciones entre estos dos grupos siguen un ciclo, el cual, tiende a oscilar de forma periódica, tal como se puede apreciar en la figura 2 (p.7).

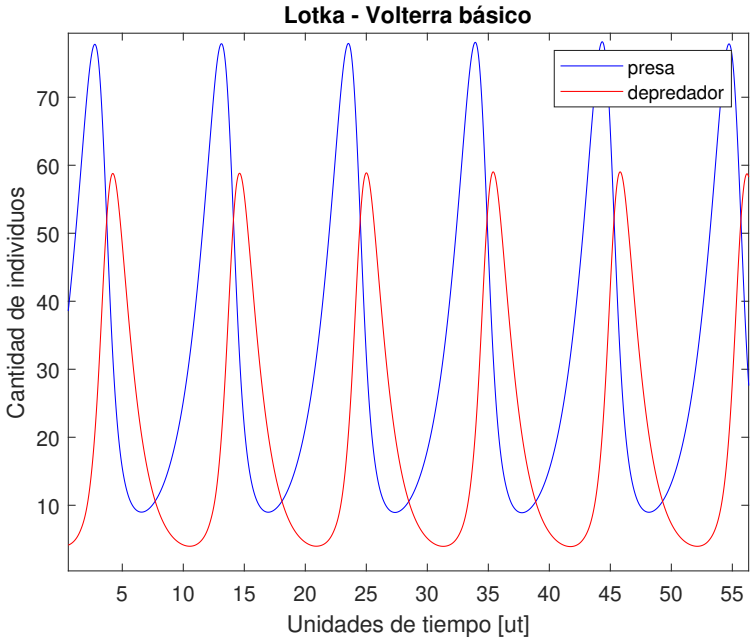


Figura 2. Comportamiento temporal de las poblaciones de presas y depredadores según el modelo Lotka-Volterra, basado en los datos de (Gómez y Vélez, 2009). El código puede encontrarse en el anexo a esta tesis, MATLAB, Main-Lotka-Volterra y Lotka-Volterra.

Oganician *et al.* (2017) expresan que el modelo de Lotka-Volterra es uno de los primeros modelos de interacción, el cual presenta un comportamiento oscilatorio, producto de la interacción entre especies del tipo presa-depredador, cuyo comportamiento también es oscilatorio. Regresando a la figura 2, en ésta se puede apreciar que la población de la especie presa (con linea azul) y la población depredadora (con linea roja), tienen un comportamiento periódico, no paralelo, pues presenta un desfase constante de la especie depredadora con respecto a la presa. Dicho desfase, explican Oganician *et al.* (2017), son los efectos de las variaciones de la población de las presas en el grupo depredador, los cuales son visibles después de cierto tiempo (p.5).

En palabras de Oganician *et al.* (2017), los modelos anteriores de dinámica de poblaciones no presentaban estas características. Pues tendían a incrementar de forma

exponencial o por el contrario, se estabilizaban en un valor de equilibrio estable. El comportamiento oscilatorio, como se expreso con anterioridad, es el resultado de la interacción entre las especies, sin tomar en consideración aspectos como los cambios en el entorno (p.6).

La comprensión de este punto resulta de vital importancia, pues el presente trabajo de tesis involucra el estudio y la implementación en enjambres de robots móviles de 3 modelos matemáticos que describen las relaciones presa-depredador: *i*) el modelo básico de Lotka-Volterra, *ii*) una modificación del modelo considerando el comportamiento de competencia entre los individuos del grupo y *iii*) un modelo con 3 participantes, con relaciones no lineales entre ellos, esto con el fin de evaluar si es posible la aparición de dinámicas caóticas a través de las interacciones.

1.3. Modelo Lotka-Volterra con competencia entre depredadores

Boccara (2010) menciona que un modelo presa-depredador mas realista debería al menos tomar en consideración alguno de los siguientes aspectos (p.33):

1. ***Competencia intraespecífica****, esto hace referencia a la competencia que surge entre los individuos que pertenecen a la misma especie.*
2. ***Respuesta funcional del depredador.*** *En este caso, nos referimos a la relación presente entre el grado de consumo del depredador y la densidad de las poblaciones de presas.*
3. ***Respuesta numérica del depredador****, se refiere a la eficiencia con la que el alimento extra se transforma en depredadores adicionales.*

Haciendo caso a lo anterior y tomando como referencia un ejemplo presentado en este documento (p.46), se modifica el modelo Lotka Volterra básico (1), al agregar el término "$c_1 D^2$", el cual evitará que la población de depredadores crezca sin límite. Boccara (2010) (p. 51) también menciona que:

> Un crecimiento exponencial de una población a menudo se denomina como *crecimiento maltusiano*, en honor a Thomas Robert Malthus, quien afirmaba que debido a que una población crece mucho más rápido que sus medios de subsistencia, “el vicio y la miseria” operarán para frenar el crecimiento de dicha población.

El modelo (1) queda entonces descrito por:

$$\begin{aligned} \frac{dP}{dt} &= r_1 P - \alpha_1 PD \\ \frac{dD}{dt} &= \alpha_2 PD - r_2 D - c_1 D^2 \end{aligned} \tag{2}$$

Como se puede apreciar, el único cambio en la ecuación es el añadido del elemento previamente mencionado, mientras el resto permanece igual al modelo Lotka-Volterra básico.

1.4. La teoría del caos

En Upadhyay *et al.* (1998) se expresa que (p. 1325):

> La existencia del caos en los sistemas ecológicos ha sido un tema de extenso debate ((May, 1987) y las referencias del mismo). Existen dos razones principales por las que esto ha tenido lugar: *i*) los modelos estudiados rara vez representan situaciones ecológicas reales y *ii*) la observación del caos en modelos de ecosistemas no han sido basados en procedimientos biológicamente sólidos. El primer motivo es debido a que la mayoría de los estudios se concentraron en modelos de una sola especie. [(May *et al.*, 1973); (May y Oster, 1976)] ya que las poblaciones biológicas no existen de forma aislada.

Además de señalar que (p. 1325-1326):

Aunque se ha observado una dinámica caótica en muchos modelos de sistemas [(Gilpin, 1979); (May y Oster, 1976); (Hassle, 1978); (Schaffer, 1985); (Schaffer y Kot, 1985); (Hastings y Powell, 1991); (Rai y Sreenivasan, 1993); (Rinaldi *et al.*, 1993)], los esfuerzos por observarla en los sistemas naturales ha sido en general un fracaso. Los teóricos no han considerado situaciones ecológicas reales mientras diseñan sus sistemas. Los ecológicos de campo con frecuencia piensan que el caos es solo una mera curiosidad matemática. Por tanto, los modelos adecuados se deben diseñar de forma que algunos de los valores de los parámetros en los modelos representen algún atributo definido de los sistemas reales que se supone que modelan las ecuaciones de evolución.

Los investigadores consideran que la causa posible por la que se ha fallado en observar el caos en las poblaciones naturales se debe ya sea por la no disponibilidad de datos de calidad adecuada (Schaffer, 1985) o la superposición (no lineal) de fluctuaciones (May, 1987) en la serie temporal determinista.

En Hastings *et al.* (1993) se indica que (p. 4-5):

La definición más simple y más intuitiva del caos es la extrema sensibilidad a las condiciones iniciales. Ya que si un sistema tiene una dinámica caótica, entonces la diferencia entre las trayectorias de dos poblaciones que tienen una ligera diferencia en las condiciones iniciales crecerá hasta que dicha diferencia sea en esencia tan grande como la variación en cualquiera de sus trayectorias. La diferencia entre las trayectorias crecerá exponencialmente (como en el crecimiento exponencial simple) a través del tiempo. En otras palabras, si existiese un error en la determinación de las condiciones iniciales, el error crecerá hasta que éste sea tan grande como la señal.

Por otra parte, en Ellner y Turchin (1995) se indica que “los sistemas caóticos son *amplificadores de ruido* que magnifican las perturbaciones” (p.343), lo cual coincide con lo expresado con anterioridad, mientras que los sistemas no caóticos son definidos por Ellne y Turchin como “*silenciadores de ruido* que amortiguan las perturbaciones” (p.343). Los autores también expresan que “los osciladores de poblaciones caóticas son mas predecibles en pequeñas escalas de tiempo que las fluctuaciones impulsadas por factores puramente exógenos tales como el clima (Hastings *et al.*, 1993)” (p.343). Este mismo artículo comenta, que con base en lo demostrado por Allen *et al.* (1993),

“el caos puede reducir, en vez de promover, la extinción de las especies en un sistema de meta-población” (p.344).

Para Torres (2005) un sistema dinámico considerado *caótico* exhibe las siguientes *características*:

1. Es un modelo matemático continuo no lineal de por lo menos 3 estados o un sistema discreto no lineal, descrito en ecuaciones diferenciales.
2. Tiene alta sensibilidad a condiciones iniciales, también conocido como “efecto mariposa”.
3. Posee un *atractor extraño*, definido como una región en su espacio de fase que no corresponde a una forma convencional, tal como punto fijo o ciclo límite (Gallego Valencia, 2010).

1.5. Caos en una cadena alimenticia de tres especies

En Hastings y Powell (1991) se comenta que “los modelos clásicos ecológicos de interacción entre poblaciones típicamente se enfocan en dos especies” (P.896).

Ha sido ampliamente reconocido que la “caricatura” limitada de los sistemas ecológicos de dos especies interactuando sólo puede dar cuenta de un pequeño número de fenómenos que son comúnmente exhibidos en la naturaleza. Esto es particularmente cierto en los estudios de la comunidad donde la esencia del comportamiento de un sistema complejo sólo puede ser entendido cuando las interacciones entre una población grande de especies es incorporada Pimm (1982); Paine (1988).

Una aproximación al estudio de una comunidad ecológica empieza con un objeto importante: su red alimenticia. Los estudios teóricos de las redes alimenticias deben abordar la cuestión de cómo acoplar el gran número de especies que interactúan. Una línea de investigación asume que “los bloques constructores” son especies interactuando en parejas (ver los influyentes libros de (May *et al.*, 1973), (Pimm, 1982)). Se asume que el comportamiento de la comunidad entera surge del acoplamiento de la fuerte interacción de dichos pares (p. 896).

Sin embargo Hastings y Powell (1991) expresan que el comportamiento importante que puede ser considerado como crítico para el funcionamiento de la comunidad solamente puede surgir a través de la interacción de tres o más especies. Además de añadir que en las comunidades marinas intermareales, los influyentes estudios de Paine (1966) demostraron la importancia del fuerte acoplamiento de niveles tróficos más altos con niveles más bajos en la red trófica. Los desarrollos matemáticos también sugieren que los modelos de las comunidades que involucran solamente a dos especies como bloques de construcción básicos pueden pasar por alto un comportamiento ecológico importante (p.896)

Los modelos continuos de dos especies, según el documento de (Hastings y Powell, 1991) tienen solo dos patrones básicos: La aproximación al equilibrio o al ciclo límite (Segel, 1984). En contraste, los modelos en tiempo discreto de incluso una especie simple puede exhibir comportamiento caótico (Por ejemplo, (May, 1974, 1976)), pero la investigación de los últimos 15 años (Por ejemplo, (Gilpin, 1979; Gukenheimer y Holmes, 1983)) demostraron las dinámicas complejas que pueden surgir en modelos de sistemas (en tiempo continuo) con tres o más especies (p. 896).

En el mismo documento se expone que la característica clave de la dinámica caótica es la dependencia sensible a las condiciones iniciales. Una investigación hecha por Gilpin (1979) mostró que *un sistema de un depredador y dos presas competidoras podía exhibir un comportamiento caótico*.

1.6. Estructura del modelo caótico

En el artículo elaborado por Rai (2004) se expresa que:

> existen dos tipos de depredadores: (*i*) los especialistas y (*ii*) los generalistas. El depredador especialista es descrito como el que muere exponencialmente rápido cuando su alimento favorito esta ausente o hay poco suministro. El segundo tipo de depredador cambia a un alimento alternativo cuando su alimento preferido esta escaso. El modelo de Rosenzweing-McArthur (RM) Rosenzweig y MacArthur (1963), es el que describe la dinámica del depredador especialista y su presa:

$$\begin{aligned} \frac{dX}{dt} &= a_1 X - b_1 X^2 - \frac{\omega Y X}{X+D}, \\ \frac{dY}{dt} &= -a_2 Y + \frac{\omega_1 Y X}{X+D_1} \end{aligned} \tag{3}$$

Aquí, Rai (p. 128) indica que:

X representa a la presa para al depredador especialista, Y con una respuesta funcional Holling tipo II, a_1 corresponde a la tasa de auto-reproducción para la presa, b_1 mide la intensidad de la competencia entre los individuos de la presa X, ω es la tasa máxima de remoción per cápita de especies de presa X debido a la depredación de su depredador Y, D es el valor de la densidad de la población X en el que la tasa de remoción per capita es la mitad de ω, el parámetro a_2 mide qué tan rápido el depredador Y muere cuando no hay presas para capturar, matar y comer, $\frac{\omega_1 X}{(X+D1)}$ es la tasa de ganancia per cápita del depredador Y, ω_1 es su valor máximo y $D1$ denota la densidad de población de la presa en la que la ganancia per cápita por unidad de tiempo en Y es la mitad de su valor máximo ($\omega 1$).

El autor (p.128) de igual forma expresa que:

Un modelo de Holling y Tanner (HT) (Pielou, 1977) describe la dinámica de un depredador generalista y su presa:

$$\begin{aligned} \frac{dZ}{dt} &= aZ(1-\frac{Z}{K}) - \frac{\omega_3 U Z}{Z+D_3}, \\ \frac{dU}{dt} &= -cU - \frac{\omega_4 U^2}{Z} \end{aligned} \tag{4}$$

Rai Vikas describe que

Z es la comida favorita del depredador generalista U. En este modelo, tanto la presa como el depredador crecen logísticamente. A es la tasa per cápita de auto reproducción del depredador Z y K es la capacidad de carga de su entorno. El parámetro ω_3 mide el máximo de la tasa per cápita de eliminación de la población de presas por su depredador U. D_3 es la constante de semi-saturación de la presa Z. El parámetro c representa la tasa per cápita de auto reproducción del depredador U y w_4 mide la gravedad de la limitación impuesta al crecimiento de la población de depredadores por la disponibilidad per cápita de sus presas.

Rai procede a crear un modelo en el que presenta una situación en la que:

> La especie presa Z sale de la parcela en búsqueda de un mejor hábitat. El sistema resultante se puede describir mediante las siguientes ecuaciones (p. 130):

$$\frac{dX}{dt} = a_1X - b_1X^2 - \frac{\omega YX}{X+D},$$

$$\frac{dY}{dt} = -a_2Y + \frac{\omega_1 YX}{X+D_1} - \frac{\omega_2 Y^2 Z}{Y^2+D_2^2}, \tag{5}$$

$$\frac{dZ}{dt} = cZ^2 - \frac{\omega_3 Z^2}{Y}$$

En este caso Rai explica que:

> El depredador generalista Z es una especie sexual. Para la mayoría de las especies sexuales en la mayoría de las densidades de población, la reproducción esta determinada principalmente por el número de hembras. Debido a la demografía sesgada por las hembras de las especies sexuales, el grupo de poblaciones es lineal en tamaño de la población, que está muy por debajo de la capacidad de carga. Sin embargo, el crecimiento de una población que se reproduce sexualmente es proporcional al cuadrado del número de individuos presentes, en situaciones en que la población se encuentra en densidades muy bajas. Esto se conoce como el efecto Allee (Allee *et al.*, 1949). En tales especies, la disminución de la población a bajas densidades ocurre debido a una variedad de causas (Lande, 1988). La más importante de ellas es la de-

> presión endogámica, que resulta de la modificación física o química del entorno de organismos por interacción social o por éxito de apareamiento dependiente de la densidad de población.
> Por ejemplo, algunos organismos acuáticos condicionan su medio, liberando sustancias que estimulan el crecimiento de especies, que tienen una estructura genética similar. Las poblaciones dispersas rara vez proporcionan suficientes oportunidades para la interacción social necesaria para la reproducción. En este caso, la última ecuación del modelo (5) se modifica a (130):

$$\frac{dZ}{dt} = cZ^2 - \frac{\omega_3 Z^2}{Y+D_3} \tag{6}$$

Rai procede, mencionando que:

> Algunos insectos depredadores superiores a menudo cambian a presas alternativas en situaciones en las que su comida favorita es escasa. Este hecho puede ser acomodado reemplazando Y^2 con Y en el último término de la segunda ecuación del modelo (5) ya que su respuesta funcional es Holling tipo II (p. 130).

De igual forma el autor explica que *las ecuaciones de (5) definen la fase lineal de este modelo. La fase no lineal puede describirse con este mismo modelo si se sustituye la ecuación* $\frac{dz}{dt}$ *con la ecuación (6). La versión de invertebrados se puede obtener reemplazando* Y^2 *con Y en el último término del modelo 5 (la segunda ecuación).*

Rai (p. 130) hace la mención:

> Se puede observar que la dinámica de la otra comunidad depredador-presa (depredador generalista-presa) se modela siguiendo un esquema dado por (Leslie, 1948; Leslie y Gower, 1960; Pielou, 1977). Esta formulación de la dinámica depredador-presa ha recibido severas criticas por parte de Yodzis (1994) y Abrams (1994).

1.7. Hipótesis

Es posible representar diversos modelos presa-depredador mediante la implementación en enjambres robóticos de bajo costo, tomando en cuenta las limitaciones de los mismos.

1.8. Objetivos

Con la realización de esta investigación, se pretende alcanzar el siguiente **objetivo general**:

Utilizar la teoría de sistemas complejos para implementar distintos modelos presa-depredador en enjambres robóticos mediante algoritmos de control distribuido.

Mediante alcanzar los **objetivos particulares**:

- Estudiar diferentes relaciones presa-depredador inspirados en la naturaleza.
- Seleccionar relaciones presa-depredador en diferentes escenarios de ocurrencia.
- Adaptar algunos modelos de relaciones presa-depredador para funcionar bajo el esquema máquinas de estado finitos (control distribuido).
- Simular y comparar algunas relaciones presa-depredador de interés.
- Implementar posiblemente algunas relaciones presa-depredador de interés en robots móviles, llamados kilobots.

Capítulo 2. Metodología

En esta investigación de trabajo de tesis, se opta por utilizar dos enfoques al momento de obtener los resultados. Estos pueden definirse como:

1. Implementación física:

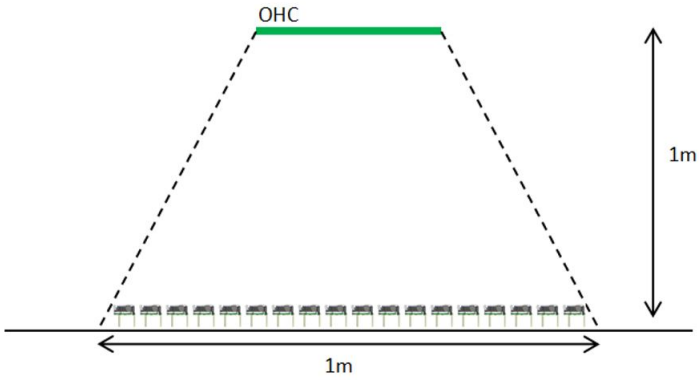


Figura 3. Área de trabajo (Tharin, 2010).

   Consiste en un enjambre de hasta 30 kilobots disponibles en el *Laboratorio de Sincronización y Sistemas Complejos* del Departamento de Electrónica y Telecomunicaciones del CICESE, el controlador, el área de trabajo y la computadora se muestran en las figuras (figuras 3 y 4).

2. Simulación virtual:

   Consiste en el empleo del simulador y los programas requeridos para trabajar con los modelos de kilobots, tales como como: la interfaz kilobot GUI, MATLAB, V-REP, FET y BASGEN (ver figura 5).

Al comienzo, se realizó un análisis con el propósito de definir las capacidades y limitaciones de los kilobots, de forma que se pudiera adaptar a los modelos matemá-

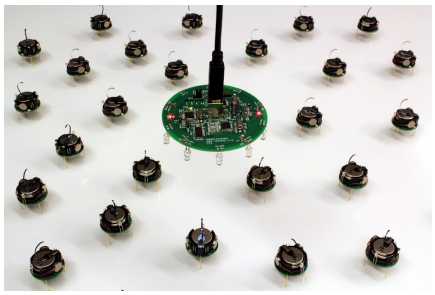


Figura 4. Apariencia del controlador y un enjambre de kilobots (Goddard *et al.*, 2015).

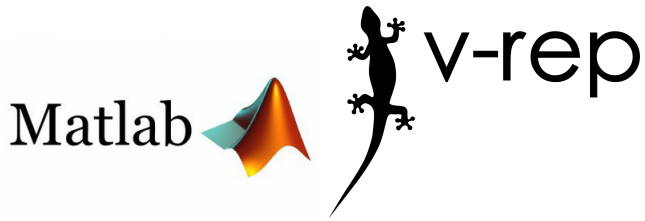


Figura 5. Simuladores utilizados para el enfoque numérico del trabajo (MATLAB, Septiembre 2017) (Robotics, Jun 25 2019).

ticos basados en las interacciones presa-depredador. Para ello, además del análisis visual se hizo uso del manual proporcionado por el fabricante (Tharin, 2010), para luego proceder a realizar las simulaciones correspondientes. No obstante, debido a las restricciones instauradas durante la pandemia, los experimentos fueron limitados casi en su totalidad a la plataforma virtual.

2.1. Simulador V-REP

El programa V-REP de *CoppeliaSim* (Rohmer *et al.*, 2013) es utilizado para realizar las pruebas y el trabajo de programación. Cabe destacar que al realizar comparaciones entre el equipo virtual y el físico, una de las diferencias principales entre ambos resultados fue el movimiento de los kilobots, esto debido a las imperfecciones en las patas de los elementos del laboratorio físico. Por ejemplo, que algunas estuvieran ligeramente torcidas o que estuvieran un poco flojas, lo cual repercutía en la dirección en la que se desplazaban, o en el tiempo de respuesta. El tiempo de respuesta del kilobot dentro del simulador transcurre lento, mientras que los kilobots reales responden de forma casi inmediata, por ende, se recomienda realizar ajustes en el código a fin de minimizar los efectos de los puntos ya mencionados.

En la figura 6 se ilustra el área de trabajo creada para realizar los ejercicios. Lo que aparece dentro del círculo rojo es un *kilobot*, dentro del círculo verde se observa el *controlador* y la flecha azul señala un pequeño muro que evita que los kilobots se dispersen fuera del límite de la señal del controlador. Si se considera lo visto en la figuras 3 y 4, es posible comprobar que los elementos son en esencia los mismos que se tienen en el laboratorio físico.

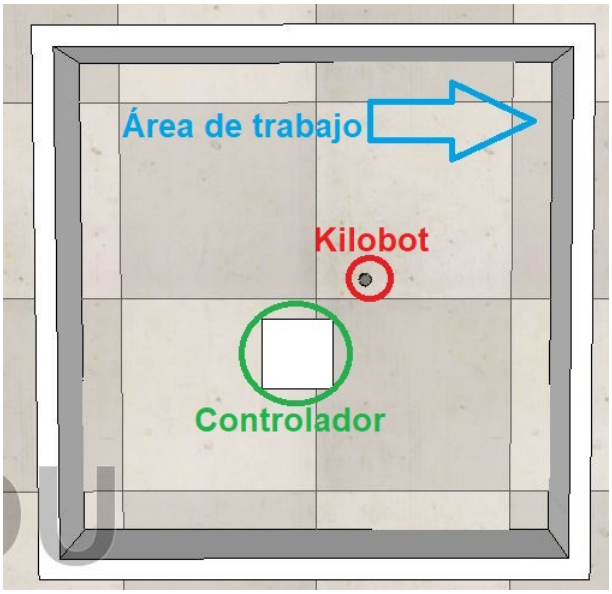


Figura 6. Ilustración del área de trabajo en V-REP.

En la figura 7 se muestra la lista donde aparecen los elementos presentes durante las simulaciones, los kilobots aparecen con la leyenda “KilobotD#*Número*”, el recuadro marcado por el círculo azul se trata del archivo que se puede modificar para crear el código fuente encargado de asignarle a los kilobots las acciones que queremos que realicen.

2.2. Adaptación del modelo L-V dentro del simulador V-REP

A continuación, se procederá a dar una explicación de los elementos que intervienen en la simulación, para luego proseguir con un pequeño ejemplo a fin de comprender mejor la funcionalidad de los modelos depredador-presa y el comportamiento de los grupos que lo componen. Se describirán los dos escenarios elaborados (una presa - un depredador y una presa - dos depredadores), así como los comportamientos correspondientes y las habilidades otorgadas.

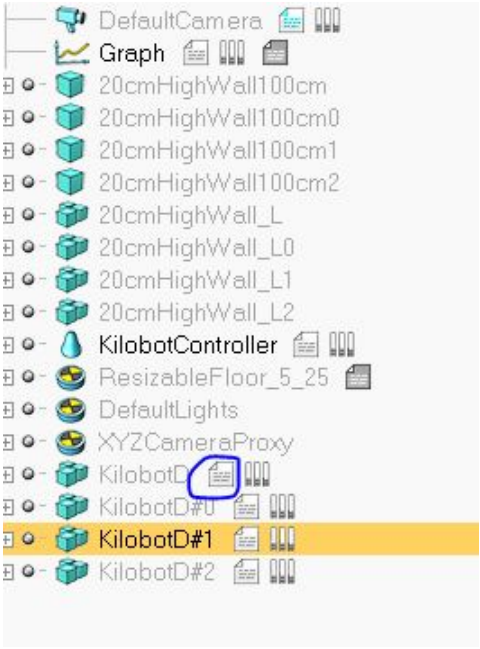


Figura 7. Lista de elementos utilizados dentro de V-REP.

2.2.1. La simulación

Al iniciar la simulación se obtiene un escenario similar al de la figura 9, en primer lugar aparece la ventana que se aprecia en el recuadro naranja, esta zona estará encargada de llevar un registro de las muertes (línea roja), el número de individuos que conforman el grupo de las presas (línea verde) y los depredadores (línea morada).

De igual forma aparece el panel del recuadro verde, el cual cumple la labor de otorgarle las ordenes de: *Run (Inicio), Pause, Charge (Cargar), BatVolt (Voltaje de la*

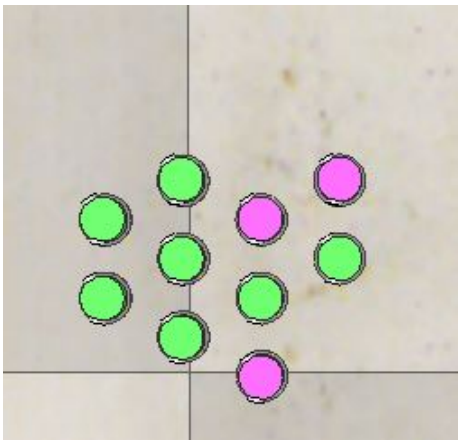


Figura 8. Ubicación inicial de los kilobots. Prueba con número de integrantes arbitrario.

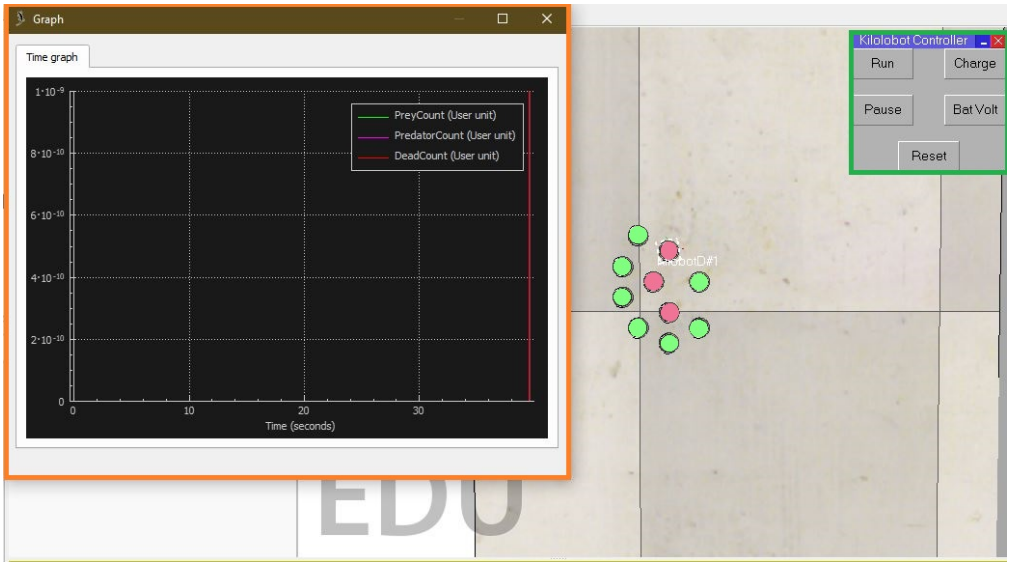


Figura 9. Herramientas de la simulación.

batería) y Reset (Reinicio) al controlador, para que éste a su vez, le envíe los comandos a los kilobots.

Por último, se observa a los kilobots, en este ejemplo se colocaron 10. Este grupo puede dividirse en dos, los depredadores (de color lila) y las presas (de color verde).

Con el transcurso del tiempo, el grupo de kilobots se desplaza por la zona y se empiezan a notar cambios, los cuales se pueden apreciar en la figura 10. En este caso, se puede observar como han muerto dos presas (kilobots en color rojo) y dos depredadores (kilobots en color blanco), estos cambios se ven reflejados en la gráfica a la izquierda, donde la línea de las presas baja hasta el número cinco, que corresponde a la cantidad de presas que continúan vivas, mientras que el número de depredadores se actualiza hasta alcanzar el uno. El número de defunciones por otra parte incrementa hasta alcanzar cuatro, que corresponde a la suma de las presas y depredadores muertos hasta ese momento.

La funcionalidad de los códigos se explicará con detalle más adelante. No obstante, cabe mencionar que es posible obtener resultados variados (figura 11) manteniendo las mismas condiciones iniciales, que en este caso consiste en la ubicación de los kilobots al principio de los ejercicios, como puede apreciarse en la figura 8.

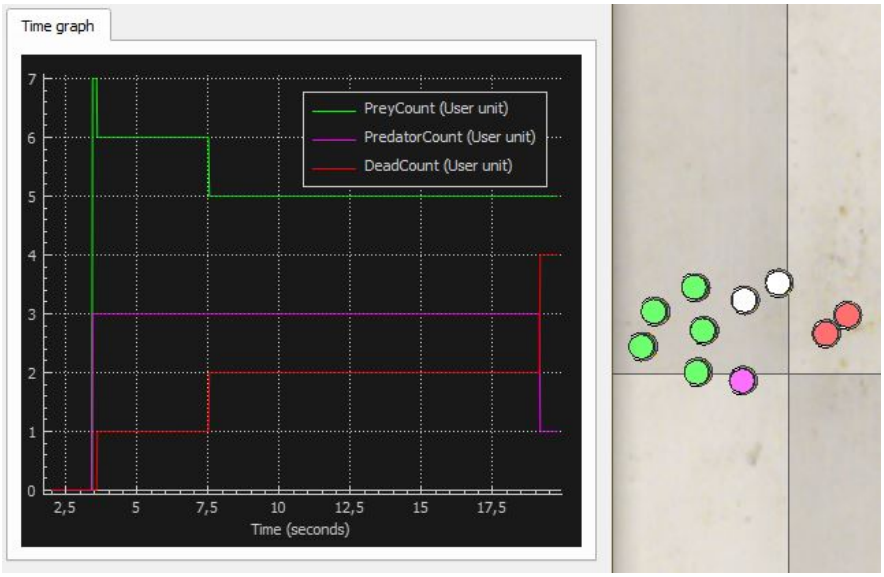
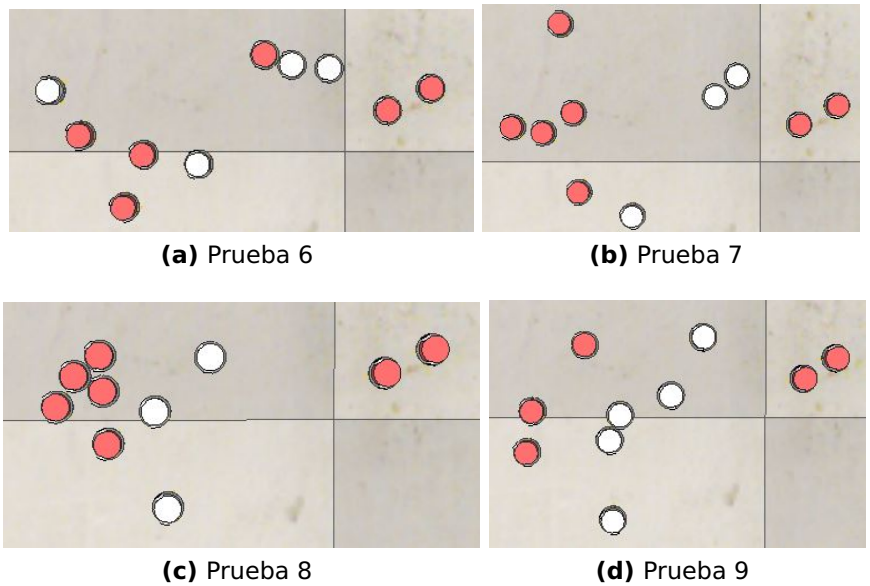


Figura 10. Registro de cambios dentro del grupo de la simulación. Código disponible en Anexo: Código, *Simulador: V-REP, Código para una presa y un depredador.*



(a) Prueba 6 **(b)** Prueba 7

(c) Prueba 8 **(d)** Prueba 9

Figura 11. Algunos resultados obtenidos en prueba con una propuesta de grupos y cantidades arbitraria, 1 presa y 1 depredador.

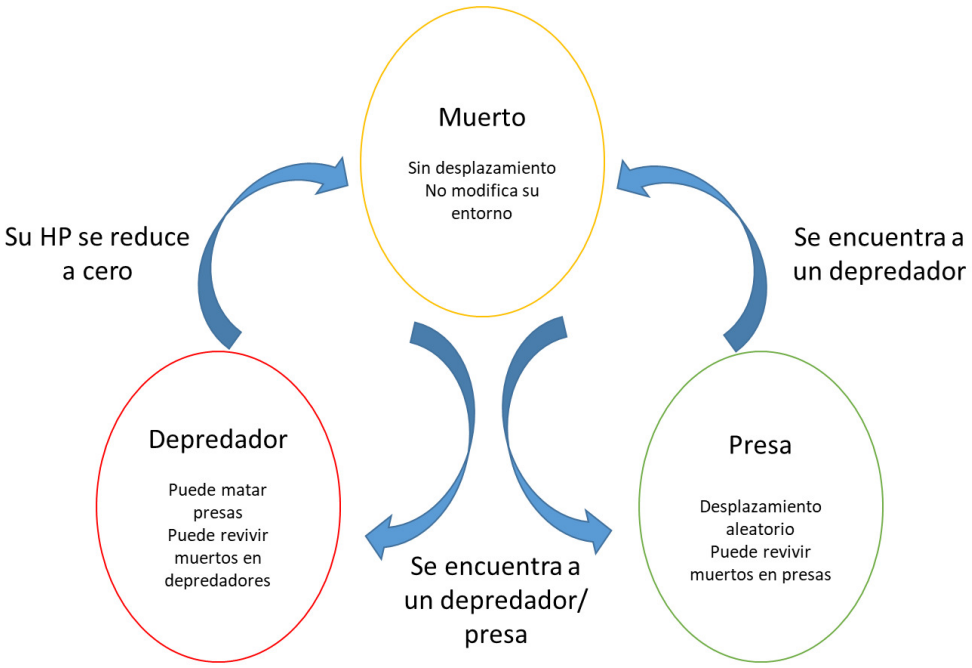


Figura 12. Esquema de la máquina de estados finitos para el algoritmo presa-depredador propuesto. El código puede ser encontrado en el anexo, *Simulador:V-REP, Código para una presa y un depredador.*

2.3. Código del escenario 1 presa y 1 depredador

En la figura 12, se muestra un diagrama de flujo, que describe las interrogantes que rigen el comportamiento de los kilobots dentro del escenario en el que interactúan una presa y un depredador.

Una presa, seguirá la lista de órdenes resumidas en la figura 12, en el óvalo verde (explicado con mayor detalle en la figura 13), en cambio si se trata de un depredador, el kilobot obedecerá lo establecido en la elipse roja (detallado en la figura 14).

2.3.1. Presa

El comportamiento de la presa está descrito con mayor detalle en la figura 13 mediante un diagrama de flujo. La presa, al iniciar el ejercicio, empezará a desplazarse por el terreno y se mantendrá atenta a las señales emitidas por los individuos

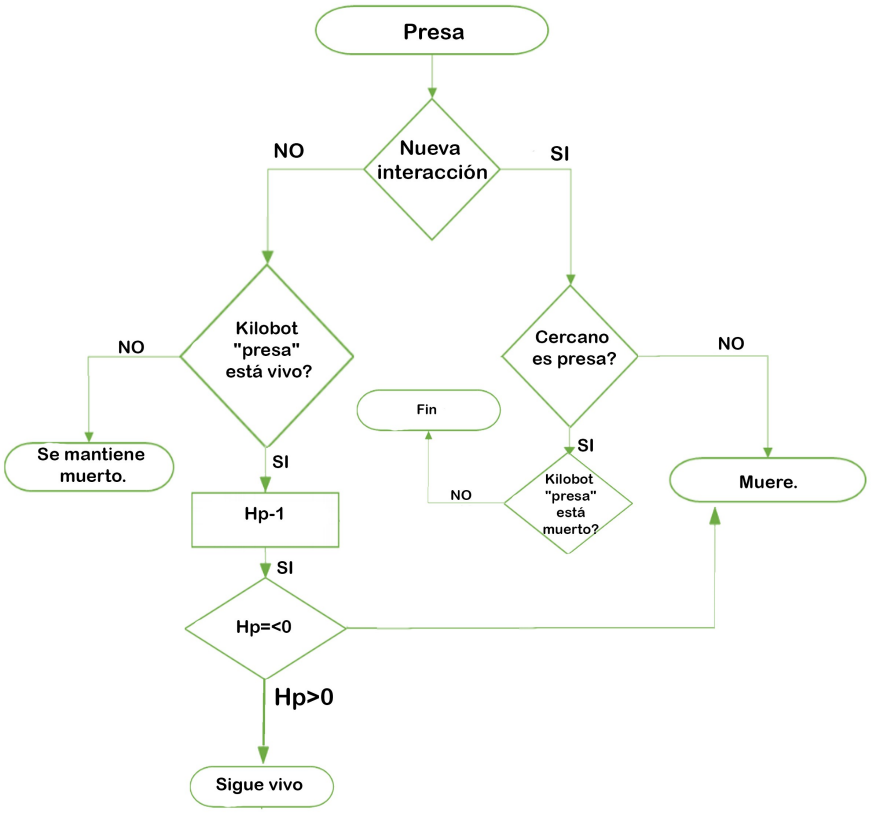


Figura 13. Diagrama de flujo del algoritmo de comportamiento para el estado presa. El código puede ser encontrado en el anexo, *Simulador:V-REP, Código para una presa y un depredador.*

que la rodean. Los cuales pueden ser otras presas vivas/muertas o depredadores vivos/muertos. Cuando el individuo se encuentre cercano a la presa en cuestión, su identidad será detectada por la misma a fin de poder responder adecuadamente establecido en el diagrama de flujo de la figura 13. Si se encuentra con una presa viva, la presa en cuestión continuará su camino, si al contrario se trata de un individuo muerto (el que se trate de una presa o un depredador no resulta de importancia este caso), la presa dará la orden de que este reviva como otra presa. No obstante, si se trata de un depredador vivo, la presa será inmediatamente devorada y adoptará el estado de muerto.

La condición principal para que un kilobot pueda revivir a otro es cumplir un cierto número de “toques” a su compañero, aunque esto en la práctica puede interpretarse como un periodo de tiempo en el que los kilobots están cerca. Pero se debe aclarar que lo que actúa dentro del código no es una medición de tiempo, si no que al contrario

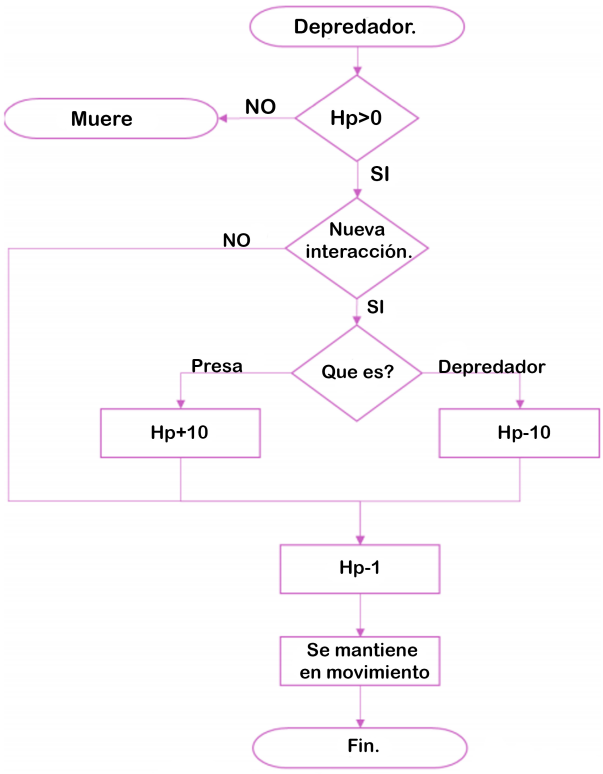


Figura 14. Diagrama de flujo del algoritmo de comportamiento para el estado depredador. El código puede ser encontrado en el anexo, *Simulador:V-REP, Código para una presa y un depredador.*

se emplea un contador de eventos, el cual ayuda a determinar esta condición para la resurrección de los kilobots muertos.

Dentro del código también se toma en cuenta el caso en que no se efectúan encuentros con otros individuos. Si esto ocurre, la presa pierde puntos de vida, esto con el fin de simular el paso del tiempo/debilidad provocada por la edad/enfermedad, aspecto que terminará por acabar con la vida de la presa cuando se alcance el límite establecido.

2.3.2. Depredador

La descripción del comportamiento de los depredadores dentro de este escenario puede ser visto con mayor detalle en la figura 14 por medio de un diagrama de flujo. El comportamiento de estos individuos, aunque similar en algunos aspectos del

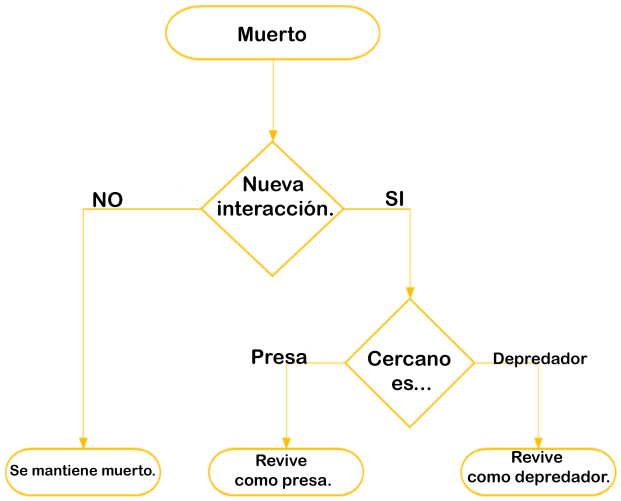


Figura 15. Diagrama de flujo del algoritmo de comportamiento para el estado de muerto. El código puede ser encontrado en el anexo, *Simulador:V-REP, Código para una presa y un depredador.*

comportamiento de las presas, se diferencia en lo siguiente:

- Pueden matar a las presas luego de alcanzar un determinado número de toques, como ya se mencionó con anterioridad.
- El encuentro con las presas beneficia al depredador permitiéndole recuperar puntos de vida (denominados como Hp dentro del código), los cuales se verán afectados de forma negativa, si el encuentro involucra a otro depredador (simulando un conflicto/pelea entre depredadores, similar a lo que ocurre en la naturaleza) o disminuyendo lentamente con el pasar del tiempo (imitando el paso del tiempo/debilidad provocada por la falta de alimento de forma similar a lo que se estableció dentro del código de las presas.
- Si el encuentro involucra a un individuo muerto, el depredador activará la resurrección del mismo, de tal forma que adopte el papel de otro depredador.

De esta forma, el depredador cuenta con una clara ventaja sobre la presa, con habilidades similares a las presentes en un escenario real. Puede consumir presas, pelear con otros depredadores y reproducirse (incrementando el número de individuos que conforman su especie). A diferencia de las presas, cuyas habilidades sólo se limitan a desplazarse y reproducirse.

2.3.3. Muertos

Por último, se hace mención de lo establecido en la figura 15, donde se explica con mayor detalle el comportamiento de los individuos presas y depredadores que cumplen con las características para activar este comportamiento, el cual establece que no pueden moverse y adoptan un tono de color distinto que facilita su clasificación, (las presas muertas cambian su tono verde por uno rojo y los depredadores abandonan su tono rosado para adoptar un color blanco). Además, permanecen listos al llamado de otros kilobots “vivos” a fin de poder revivir.

2.4. Código del escenario 1 presa y 2 depredadores

En la figura 16 se muestra el diagrama de flujo en el que podemos apreciar las interrogantes que establecen el comportamiento de los kilobots de este nuevo escenario, donde interviene un depredador que puede ser catalogado como depredador tope. Y que será descrito con mayor detalle a continuación. Esta modificación al código anterior fue elaborada con el propósito de crear un escenario en el que fuera posible el *surgimiento del caos*.

2.4.1. Presas

En este caso, el comportamiento de estos individuos se mantiene extremadamente similar al código anterior. Por tanto, se mantiene como referencia visual la figura 13 para la explicación más detallada de esta sección del código. No obstante, se considera necesario expresar que en este caso, la única variante es el añadido del depredador tope como grupo que puede atacar a las presas vivas.

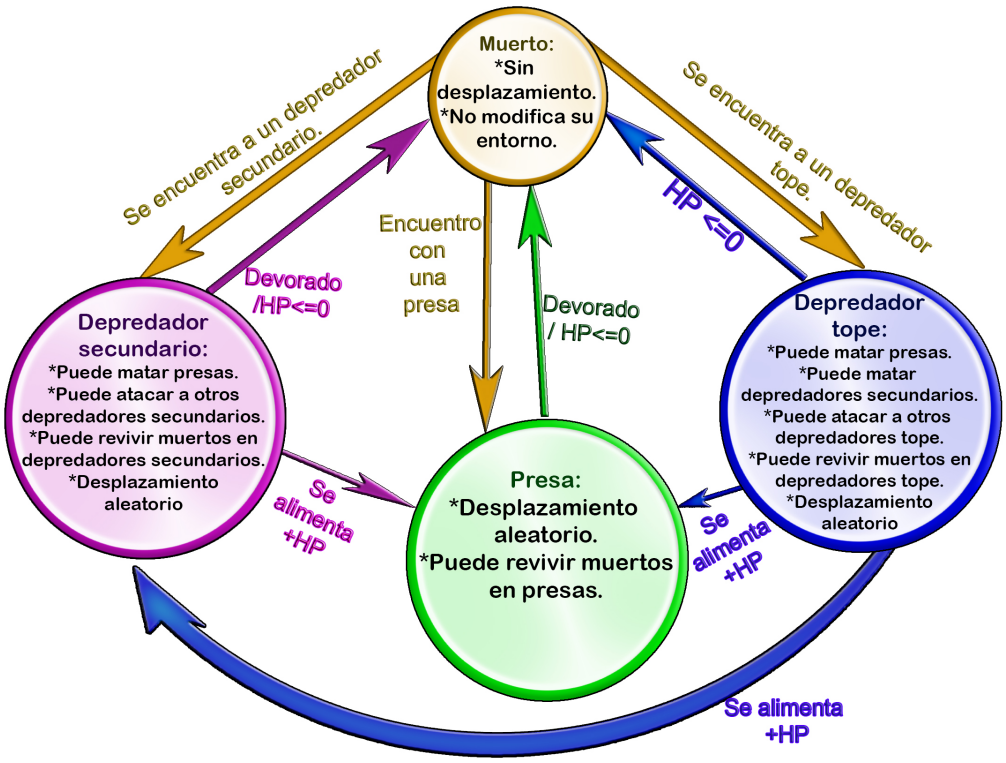


Figura 16. Esquema de la máquina de estados finitos para el algoritmo 2 depredadores y 1 presa propuesto. El código puede ser encontrado en el anexo, *Simulador: V-REP, Código para 2 depredadores y 1 presa.*

2.4.2. Depredador/Depredador secundario

El comportamiento de este individuo puede ser visto de forma más detallada en la figura 18 por un diagrama de flujo. Nuevamente, la única variante del comportamiento consiste en la inclusión de los efectos que provoca la interacción con el denominado “depredador tope”, el cual provoca la muerte inmediata del depredador secundario al entrar en contacto con éste.

2.4.3. Depredador tope

En este caso, este individuo representa el tope de nuestra cadena alimenticia. Su comportamiento es descrito con mayor detalle en la figura 19 por un diagrama de flujo. Sus habilidades se diferencian de las del depredador secundario en 2 aspectos principales, este individuo puede alimentarse tanto de las presas como de los depredadores secundarios (interacción que incrementa sus Hp) y sólo se ve afectado por el

paso del tiempo y los enfrentamientos con miembros de su mismo grupo.

2.4.4. Muertos

De forma similar al código de las presas, en este caso el comportamiento de los individuos muertos se mantiene igual al código anterior, figura 17, con la única variante de la posibilidad de revivir dentro del grupo de los depredadores tope luego de alcanzarse el número de toques requerido por el código.

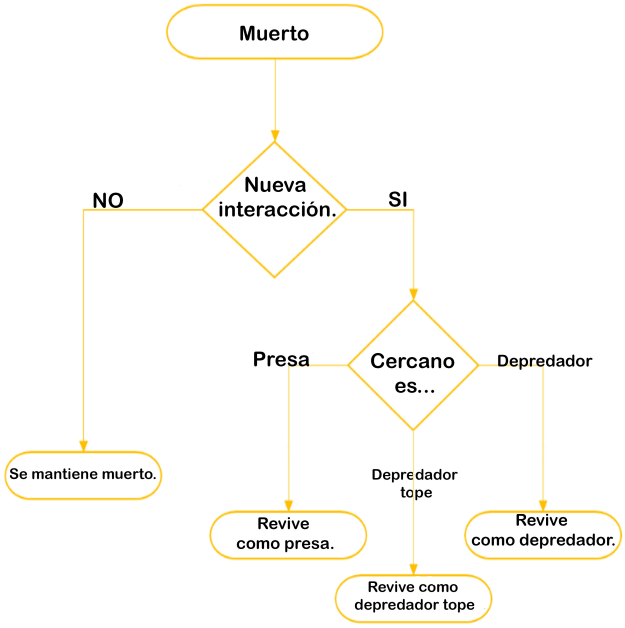


Figura 17. Diagrama de flujo del algoritmo de comportamiento para el estado “Muerto”. El código puede ser encontrado en el anexo, *Simulador: V-REP, Código para 2 depredadores y 1presa*.

2.5. Adaptación del modelo L-V dentro del ambiente de MATLAB

Se plantearon tres escenarios distintos:

- El modelo Lotka-Volterra básico *(*ver anexo: MATLAB, Main-Lotka-Volterra y Lotka-Volterra).
- Una modificación del mismo para considerar el fenónemo de competencia entre individuos de la misma especie *(ver anexo: Lotka-Volterra, Competencia intraespecifica y Competencia intraespecifica...CONT).*

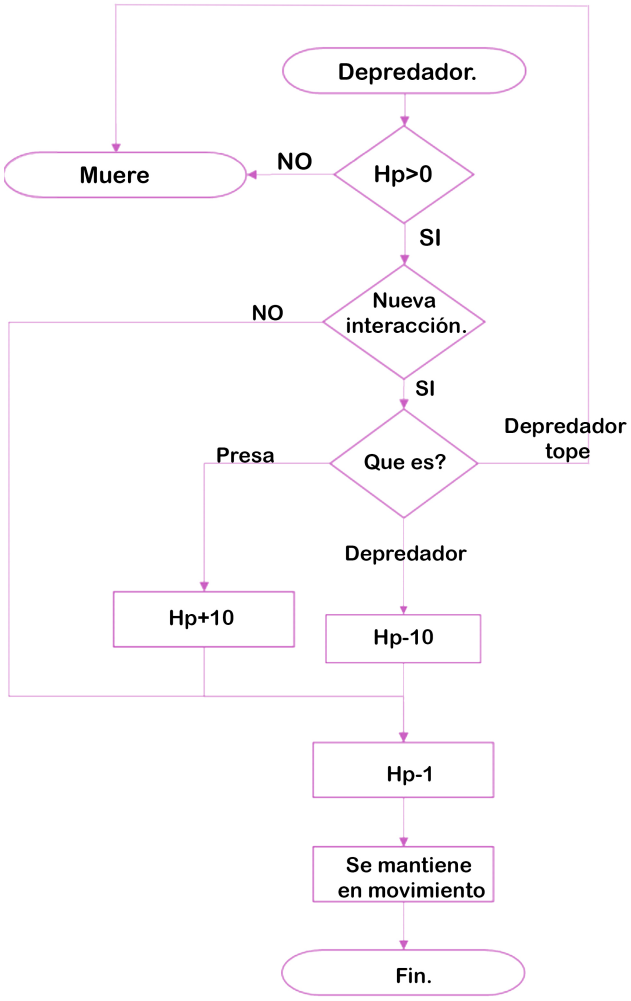


Figura 18. Diagrama de flujo del algoritmo de comportamiento para el estado Depredador. El código puede ser encontrado en el anexo, *Simulador: V-REP, Código para 2 depredadores y 1 presa*.

- Y por último, una cadena trófica con tres especies, con el fin de evaluar si esta interacción puede exhibir comportamiento caótico. *(ver anexo: Lotka Volterra, 3 especies y Lotka-Volterra, 3 especies... CONT).*

Para cada escenario, se plantearon dos simulaciones distintas:

1. Se considera el modelo matemático que describe cada una de las interacciones propuestas.
2. Se presenta una simulación donde los individuos de cada especie son representados por agentes que pueden desplazarse sobre un espacio de trabajo tipo rejilla o malla, donde en cada episodio, estos se desplazan a alguna de las 8 posiciones adyacentes a él, o pueden, por el contrario permanecer en la misma posición. Esta situación puede apreciarse en la figura 20.

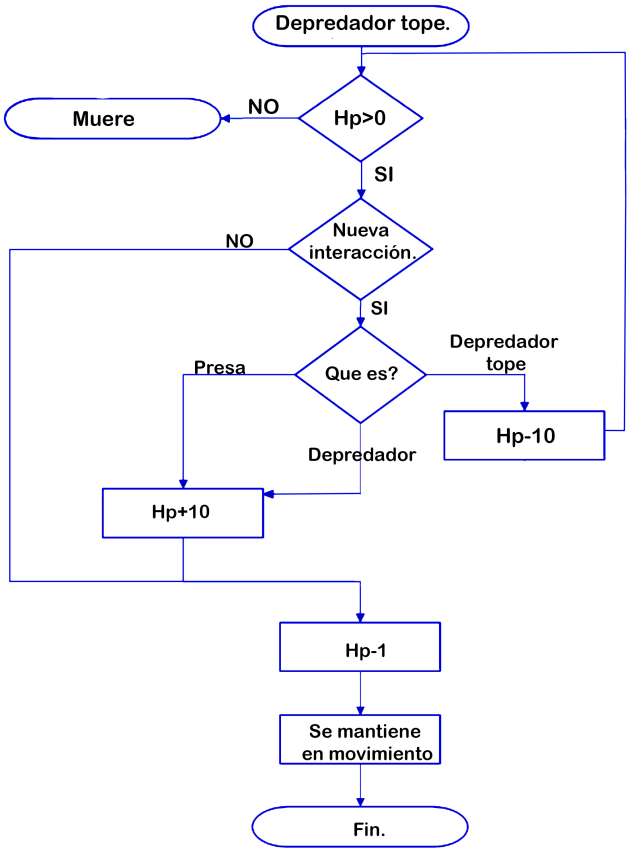


Figura 19. Diagrama de flujo del algoritmo de comportamiento para el estado “Depredador Tope”. El código puede ser encontrado en el anexo, *Simulador: V-REP, Código para 2 depredadores y 1 presa*.

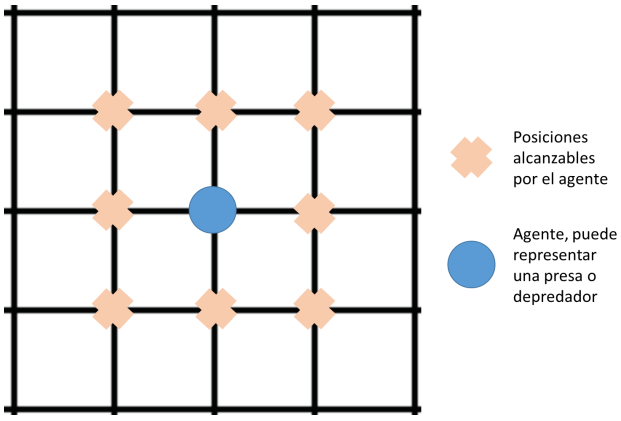


Figura 20. Ejemplo del espacio de trabajo tipo rejilla utilizado para la simulación con desplazamiento. En este modelo, los agentes (puntos azules) se pueden desplazar en los nodos de intersección de la rejilla, pudiendo alcanzar cualquiera de las 8 posiciones adyacentes (“x”).

Capítulo 3. Resultados

3.1. Simulador V-REP:

Escenario 1 depredador y 1 presa

Primeramente, con el fin de comprobar la variedad de resultados que se pueden obtener, se establecieron algunas reglas:

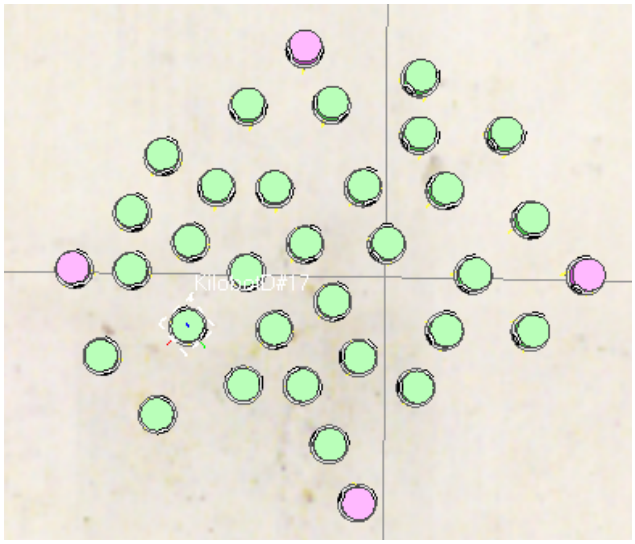


Figura 21. V-REP, Grupo de prueba: 30 presas (verdes) y 4 depredadores (rosas). Ubicación de los kilobots al principio de cada prueba efectuada.

- Se creó un espacio de trabajo de 120 cm × 120 cm (Aprox.). Dentro se colocaron arbitrariamente 34 kilobots, (30 presas y 4 depredadores, esto basado en los datos de Gómez y Vélez (2009)).
- La velocidad del simulador se incrementó 2 veces y se registró la duración de las pruebas, tomando como inicio el momento en que la gráfica del simulador detectaba la actividad de los kilobots (fig. 22), y la prueba se concluye al alcanzar la marca de los 300 s. (tiempo del simulador).
- Se pausó la simulación cada 30 s, hasta alcanzar la marca de los 90 s, con el fin de registrar los cambios en los grupos y se realizó un último registro en la marca de los 300 s. (tiempo del simulador).
- Todas las pruebas se realizaron empleando la misma ubicación inicial correspondiente a lo mostrado en la figura. 21.

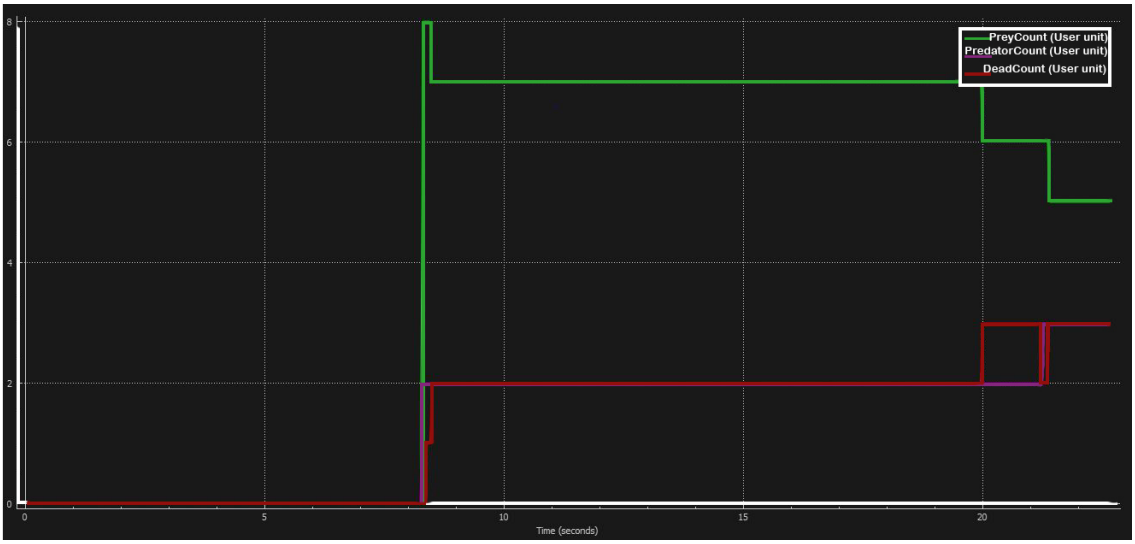


Figura 22. Gráfica de conteo de kilobots. El color verde corresponde al número de presas, el color púrpura a los depredadores y el rojo a los muertos (Presas y depredadores). En este caso se muestra un ejemplo del cambio en la gráfica al inicio de cada simulación.

En la tabla 4, se exhiben los resultados obtenidos del experimento. Durante las pruebas, el número de presas y depredadores, así como también su ubicación en el terreno y orientación al principio de cada experimento, representa una diferencia significativa en los resultados. Un ejemplo de ello puede ser visto en los resultados expuestos en la tabla 5, los cuales fueron obtenidos luego de realizar una pequeña modificación a la ubicación inicial del grupo de kilobots, mostrada en la figura 23, en este caso, la diferencia en los resultados obtenidos (disponibles en las figuras 24 y 25), es descrita a continuación: En primer lugar, el escenario modificado permitió que la población de presas fuera mayor en la marca de los 90 s, mientras el grupo de depredadores fue menor (esto podría indicar que dicha modificación le otorgó una ventaja momentánea al grupo de las presas), al concluir la prueba, las presas perecen y son algunos depredadores los que consiguen llegar vivos al final de la prueba. El número de muertos en ambos grupos, exhibe cantidades similares.

En el escenario original, por otra parte, la diferencia principal puede ser apreciada en el número de muertos, pues en este caso es el grupo depredador el que mostró un mayor crecimiento y dominio de escenario y esto provocó en consecuencia un mayor número de muertos al final de la prueba, mientras el grupo de presas concluyó sin ningún sobreviviente y un número pequeño de muertos.

Las diferencias en la prueba 4 en ambos escenarios por otro lado, mostró lo si-

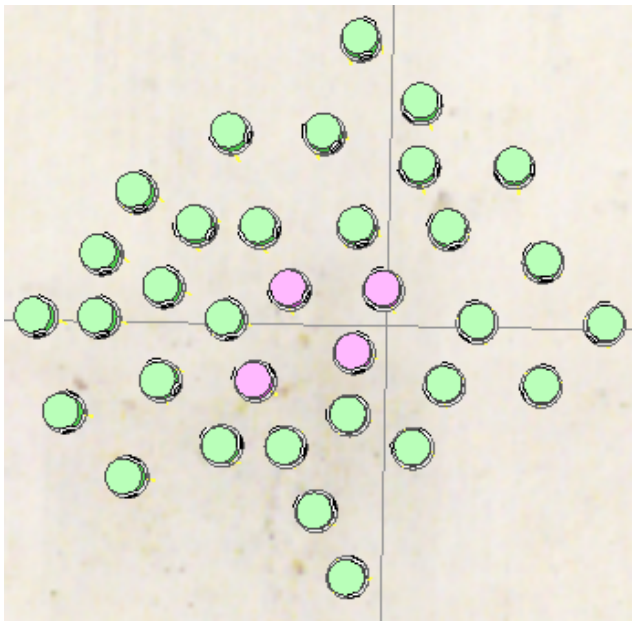


Figura 23. Nueva ubicación inicial del grupo de kilobots utilizado en el primer escenario.

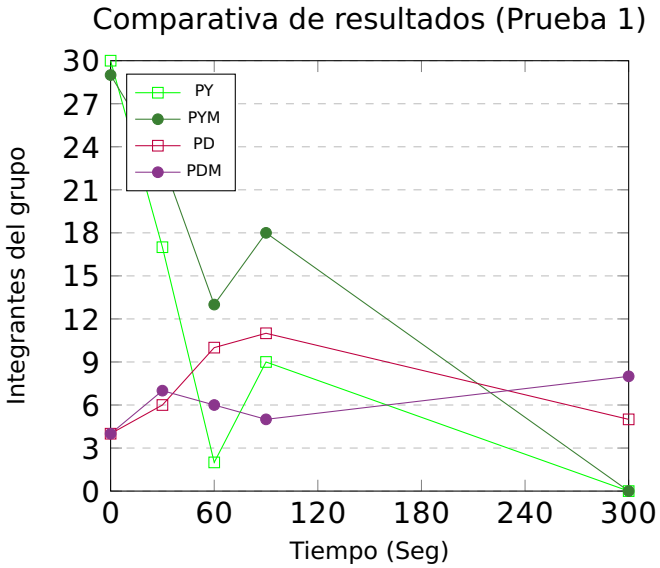


Figura 24. Comparativa de individuos vivos en cada estado de la MEF para distintos instantes de tiempo. Correspondiente al experimento 1 original y modificado [PY= Presas vivas, PYM= Presas vivas (Escenario modificado), PD= Depredadores vivos, PD = Depredadores vivos (Escenario modificado)].

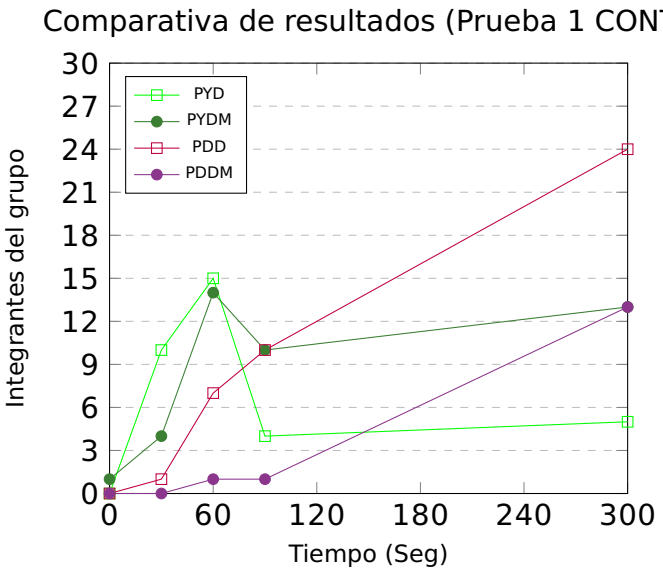


Figura 25. Comparativa de individuos muertos en cada estado de la MEF para distintos instantes de tiempo. Correspondiente al experimento 1 original y modificado [PYD= Presas muertas, PYDM= Presas muertas (Escenario modificado), PDD= Depredadores muertos, PDDM = Depredadores muertos (Escenario modificado)].

guiente: La primera diferencia notable radica en la duración de ambas pruebas, ya que en el escenario sin modificar, esta solo alcanzó la marca de los 220 s, tiempo en el que todos los kilobots perecieron mostrando una mayor población de depredadores muertos que presas muertas. Sin embargo, el escenario modificado permitió que en esta prueba, 2 presas sobrevivieran y además el grupo de presas muertas fue mayor que el de los depredadores muertos.

Lo mencionado anteriormente, en conjunto con los datos finales reportados en las tablas demostraron que la modificación realizada en la ubicación de los depredadores creó un nuevo escenario, en el que las presas tenían una mayor probabilidad de prosperar e incluso sobrevivir. Sin embargo, en el experimento original, dicha ventaja le perteneció a lo depredadores, quienes eran los que usualmente sobrevivían la prueba y cuyo grupo era el que mostraban un mayor incremento reflejado en su número de muertos.

Si se enfoca la atención en los resultados obtenidos con el escenario sin modificar, se puede apreciar que hasta el momento estos coinciden con lo dicho anteriormente en la sección 1.2, el consumo de presas aumentó conforme la población de depredadores crecía, como puede ser visto en la gráfica “resultados del experimento 1” (figura

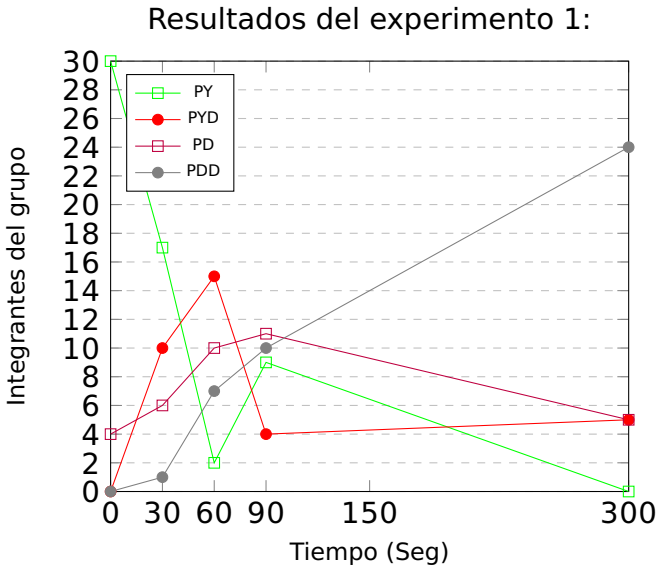


Figura 26. Cantidad de individuos en cada estado de la MEF para distintos instantes de tiempo. Correspondiente al experimento 1. (PY= Presas vivas, PYD= Presas muertas, PD= Depredadores vivos, PDD = Depredadores muertos).

26), en los primeros 30 s (*línea rosa*), la población de presas (*PY*) pereció al punto de solo quedar 17 vivos, el grupo depredador (*PD*) aprovechó esos recursos para aumentar ligeramente su población. En la marca de los 60 s, se presenta un cambio significativo, las presas sucumben por completo al ataque de los depredadores, quienes incrementan su número, y debido a ello se presentan lo que podría identificarse como los primeros indicios de competencia intraespecífica, lo cual se refleja en el número de muertes del grupo depredador (*PDD*). Cuando se cumplen 90 s, el alimento recabado con anterioridad (representado por el grupo de muertos tanto de presas como depredadores), permite que el grupo de depredadores continúe creciendo hasta alcanzar el máximo posible, sin embargo, dicho incremento en población provoca que la competencia entre los depredadores empeore, este comportamiento en conjunto con la falta de recursos da pie a que se desarrolle lo que se puede apreciar en la marca de los 300 s. En este punto, los depredadores sucumben al hambre y la población se extingue.

En la gráfica "*Resultados de experimento 4*" (figura 27), se observa un escenario diferente, aquí el ataque a las presas empieza a partir de la marca de los 30 s, provocando el incremento en el grupo depredador, en los 60 s, las presas han perecido por completo, no obstante la competencia por los recursos provoca que las defunciones en

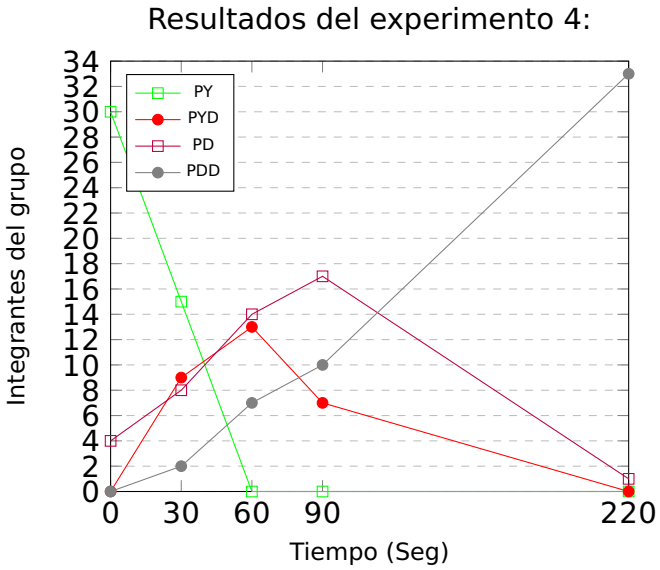


Figura 27. Cantidad de individuos en cada estado de la MEF para distintos instantes de tiempo. Correspondiente al experimento 4. (PY= Presas vivas, PYD= Presas muertas, PD= Depredadores vivos, PDD = Depredadores muertos).

el lado depredador incrementen de forma notable. El incremento en la distancia entre los depredadores es un factor clave en la rápida extinción de dichos individuos, como puede observarse en la figura 30b. Ya que con esto, los depredadores se ven imposibilitados de obtener una forma de continuar prosperando, reviviendo a sus camaradas fallecidos, lo cual se puede interpretar como un acto de reproducción o canibalismo en un escenario real, representado por la figura 30c, en la que la distancia entre los kilobots permite que la prueba pueda alcanzar la marca de los 300 s.

Por tanto, se puede decir que el aprovechamiento de recursos es un elemento clave en la supervivencia de los individuos, tal como sucede en la naturaleza. Y esto se ve reflejado en la duración de los experimentos, en esta ocasión se propuso un tiempo límite para todas las pruebas, pero en casos anteriores, al permitir que las pruebas concluyeran cuando todos los participantes fallecieran, el tiempo de las mismas se prolongó de forma indefinida cuando 2 o más individuos se desplazaban juntos, lo cual puede representar a los casos en los que las poblaciones recurren a medidas desesperadas con el fin de mantenerse con vida, recurriendo por ejemplo, al canibalismo, un comportamiento común cuando el alimento no es suficiente. Los resultados también pueden compararse a los escenarios en los que ocurre una desestabilización en los ecosistemas, que permiten que algunas poblaciones crezcan al grado de acabar

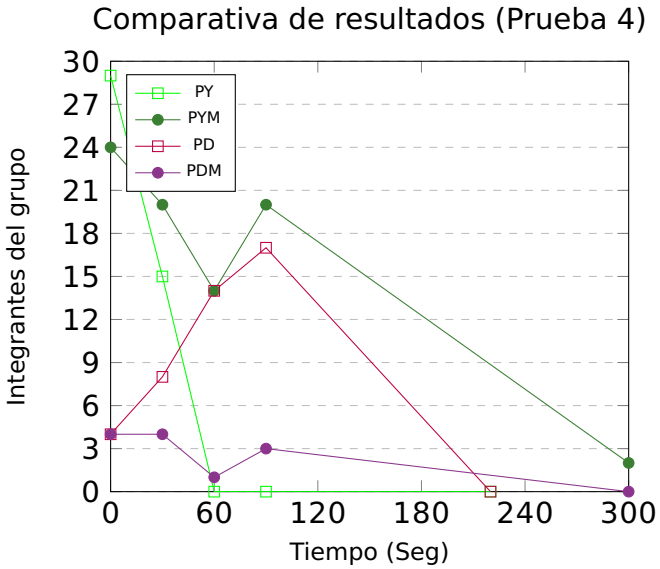


Figura 28. Comparativa de individuos vivos en cada estado de la MEF para distintos instantes de tiempo. Correspondiente al experimento 4 original y modificado [PY= Presas vivas, PYM= Presas vivas (Escenario modificado), PD= Depredadores vivos, PD = Depredadores vivos (Escenario modificado)].

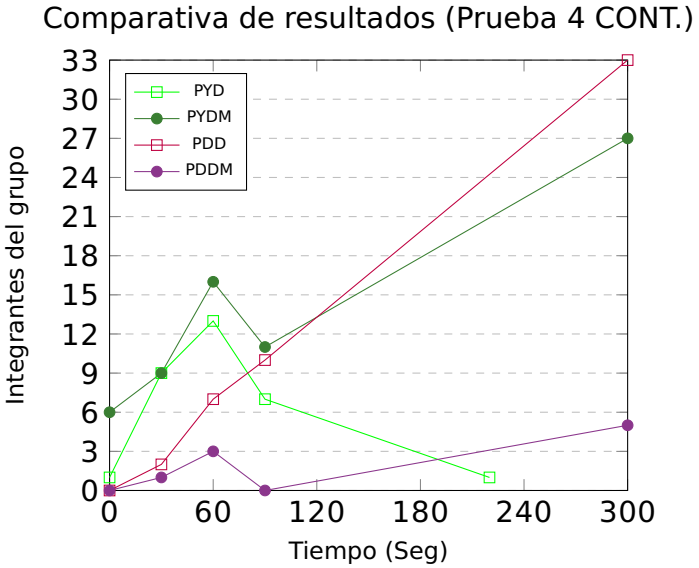
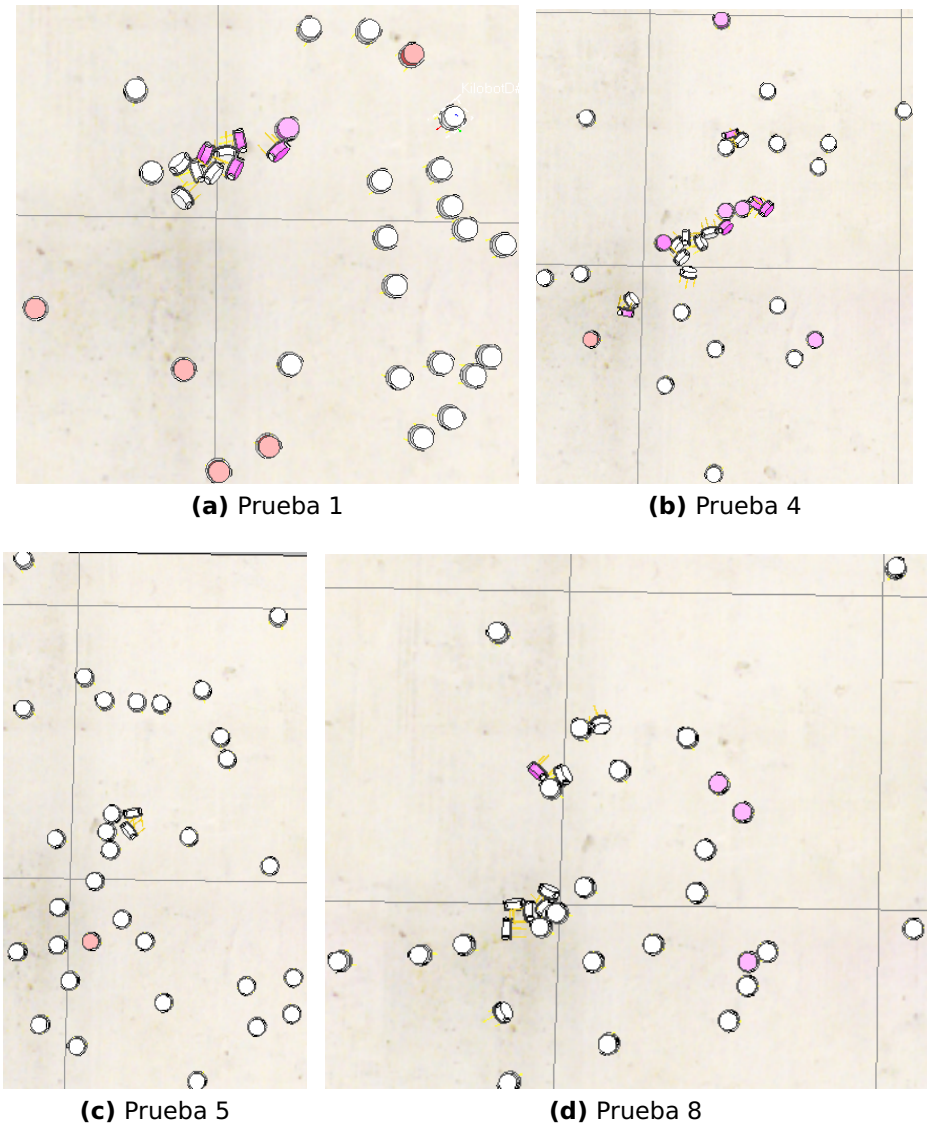


Figura 29. Comparativa de individuos muertos en cada estado de la MEF para distintos instantes de tiempo. Correspondiente al experimento 4 original y modificado [PYD= Presas muertos, PYDM= Presas muertas (Escenario modificado), PDD= Depredadores muertos, PDDM = Depredadores muertos (Escenario modificado)].



(a) Prueba 1 **(b)** Prueba 4

(c) Prueba 5 **(d)** Prueba 8

Figura 30. Algunos resultados obtenidos en prueba con grupos basados en Gómez y Vélez (2009), 1 presa y 1 depredador.

con los recursos naturales, lo cual a su vez provoca su propia extinción debido a la creciente falta de recursos y alimento. Esto puede ser el resultado de factores externos (enfermedades, propios depredadores, cambios en el clima, etc.) que llegan a perjudicar a eslabones clave en algunas cadenas tróficas.

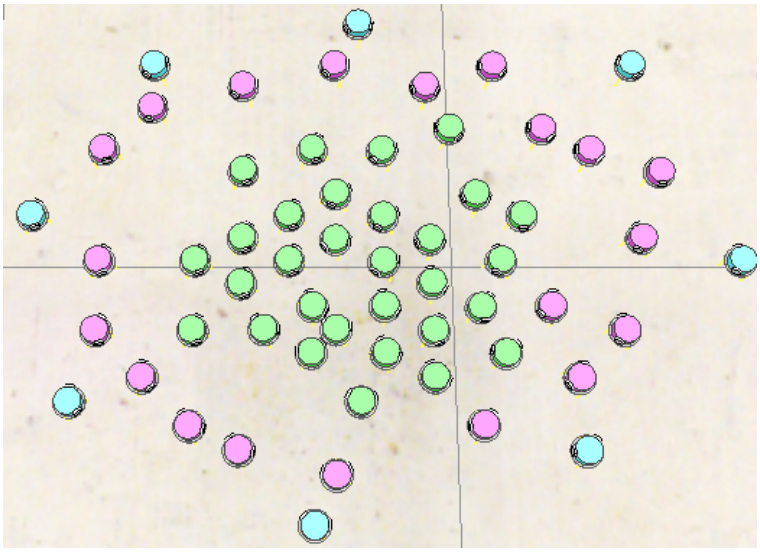


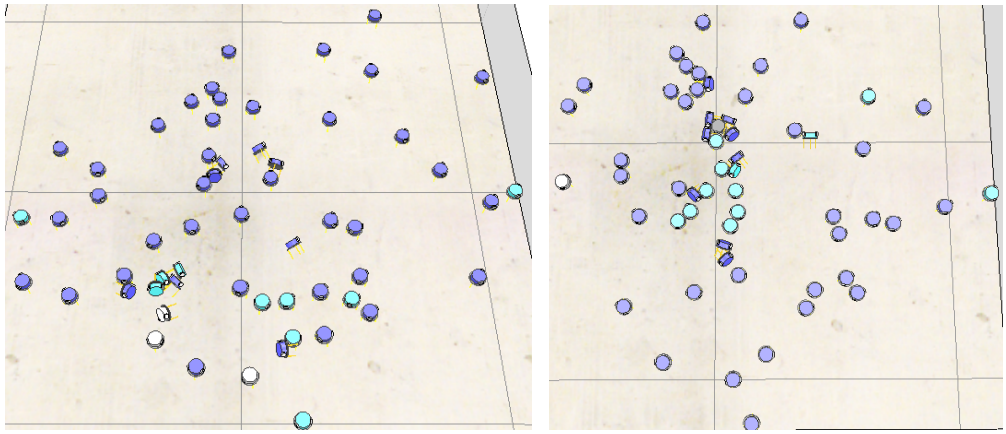
Figura 31. V-REP, Grupo de prueba: 30 presas (verdes) y 20 depredadores (rosas) y 8 depredadores tope (azul). Ubicación inicial de los kilobots al principio de cada prueba efectuada.

3.1.1. Escenario, 1 presa y 2 depredadores

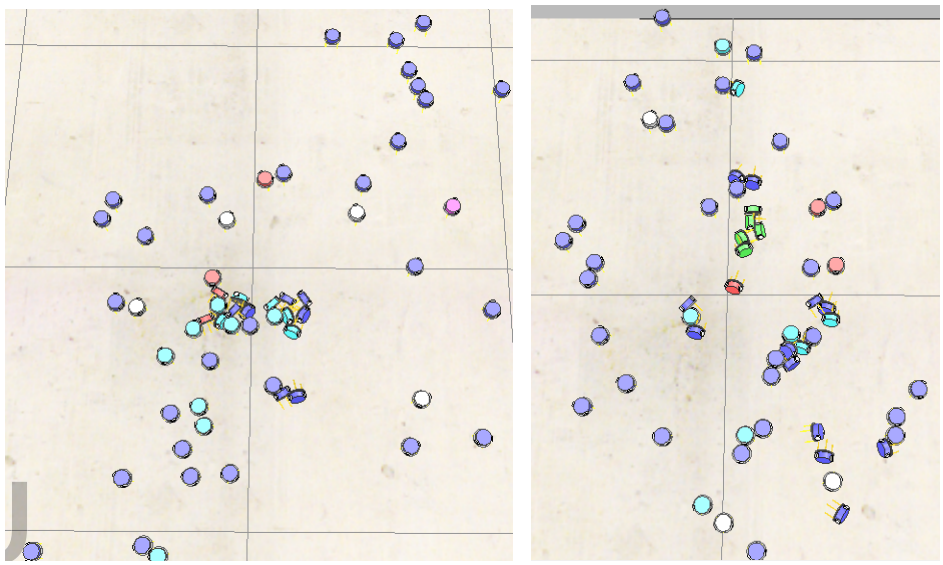
Al igual que en el escenario anterior, con el fin de comprobar la variedad de resultados que se pueden obtener, se establecieron las siguientes reglas:

- Se creó un espacio de trabajo de 120 x 120 cm (aprox.). Dentro se colocaron arbitrariamente 58 kilobots, el grupo se compone de: 30 presas, 20 depredadores y 8 depredadores tope (propuestas arbitrarias).
- La velocidad del simulador se incrementó 2 veces y se registró la duración de las pruebas, tomando como inicio el momento en que la gráfica del simulador detectaba la actividad de los kilobots (figura 22), y la prueba se concluye luego de alcanzar la marca de los 300 s (tiempo del simulador).
- Se pausó la simulación cada 30 s, hasta alcanzar la marca de los 300 s (tiempo del simulador), con el fin de registrar los cambios en los grupos.
- Todas las pruebas se realizaron utilizando la misma ubicación inicial correspondiente a lo mostrado en la figura 31.

En las tablas 6 y 7, se exhiben los resultados obtenidos durante este experimento, este nuevo escenario puede compararse a los casos en los que un agente externo ingresa



(a) Prueba 1 **(b)** Prueba 2



(c) Prueba 3 **(d)** Prueba 8

Figura 32. Algunos resultados obtenidos en prueba con grupos basados en la información de Rai (2004) 2 depredadores y 1 presa. Las figuras corresponden a lo visto durante la última muestra (300 s) en las pruebas 1, 2, 3 y 6, respectivamente.

a un ecosistema donde no existe un individuo que represente una amenaza para el recién llegado, provocando un desequilibrio que usualmente termina en la extinción de alguna de las especies nativas de la zona.

En el escenario propuesto, el depredador tope solo puede ser atacado por otro miembro de su misma especie, sin embargo tiene la posibilidad de obtener beneficios de los dos grupos presentes en el área (depredadores secundarios y presas), de forma similar a lo mostrado en la figura 1 en la sección 1.1, misma en la que se puede apreciar las limitantes de nuestros depredadores, quienes solo pueden depredar a las

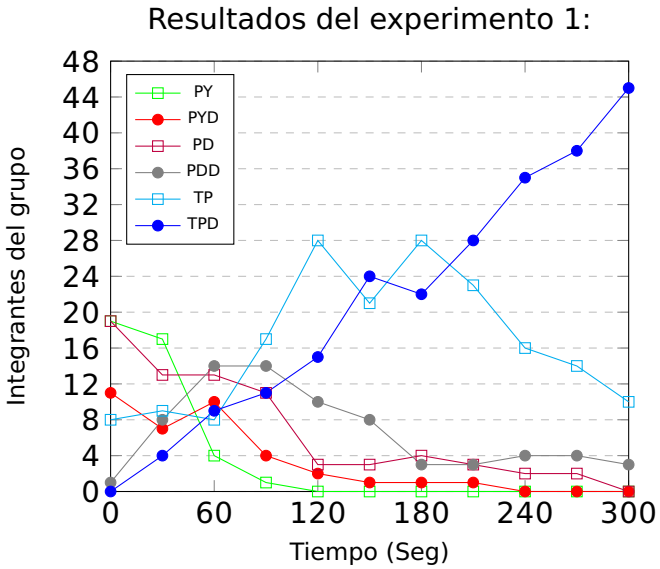


Figura 33. Cantidad de individuos en cada estado de la MEF para distintos instantes de tiempo. Correspondiente al experimento 1. (PY= Presas vivas, PYD= Presas muertas, PD= Depredadores vivos, PDD = Depredadores muertos, TP= Depredadores Tope, TPD=Depredadores Tope Muertos).

presas (consumidores primarios) y competir con los miembros de su misma especie.

En las figuras 33 y 34 se muestran de forma gráfica los cambios en la población de los grupos durante las pruebas 1 y 6 respectivamente. En la primera, el grupo depredador tope mostró un claro incremento, mientras los depredadores secundarios y las presas decrecieron en su número hasta extinguirse al final de la prueba. Sin embargo, el escenario obtenido al final de la prueba 6, mostró una pequeña resistencia en el grupo de las presas, quienes se mantuvieron juntas hasta la marca final, aspecto que les permitió sobrevivir. Este aspecto es lo que, como se mencionó en la sección anterior, permite que las pruebas alcancen una duración indefinida, pues impide que lo individuos perezcan.

Análisis de los resultados de los modelos matemáticos con MATLAB

Como se mencionó con anterioridad, la siguiente fase de la investigación consistió en la creación de 3 escenarios (descritos ya brevemente en la sección 2.5) que fueron analizados mediante 2 simulaciones distintas con el simulador *MATLAB* y que serán explicados a detalle a continuación:

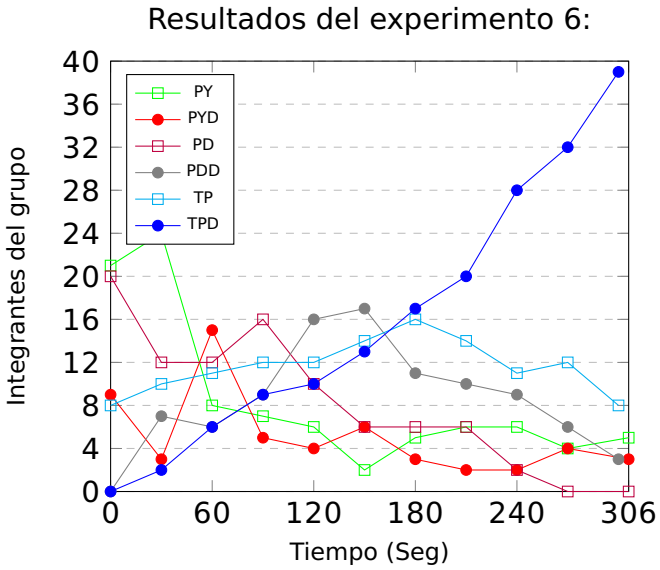


Figura 34. Cantidad de individuos en cada estado de la MEF para distintos instantes de tiempo. Correspondiente al experimento 6. (PY= Presas vivas, PYD= Presas muertas, PD= Depredadores vivos, PDD = Depredadores muertos, TP= Depredadores Tope, TPD=Depredadores Tope Muertos).

Modelo Lotka-Volterra básico

Para esta simulación se consideraron los datos de Gómez y Vélez (2009), una cantidad inicial de 30 presas y de 4 depredadores con los siguientes valores de los parámetros:

- $r_1 = 0.55$
- $\alpha_1 = 0.027$
- $\alpha_2 = 0.026$
- $r_2 = 0.83$

Los resultados pueden observarse en las figuras 2, 35 y 36.

En la figura 2 se apreció el carácter cíclico de las poblaciones, lo cual coincide con lo reportado previamente en Oganician *et al.* (2017). En este caso, cuando la población de presas (línea azul) aumenta, la cantidad de depredadores se eleva de forma similar y disminuye a la par de la cantidad de individuos de su contraparte. En el instante en que el número de presas es bajo, la población depredadora disminuye a un nivel que permite a la población de presas recuperarse. En la figura 35 se puede apreciar

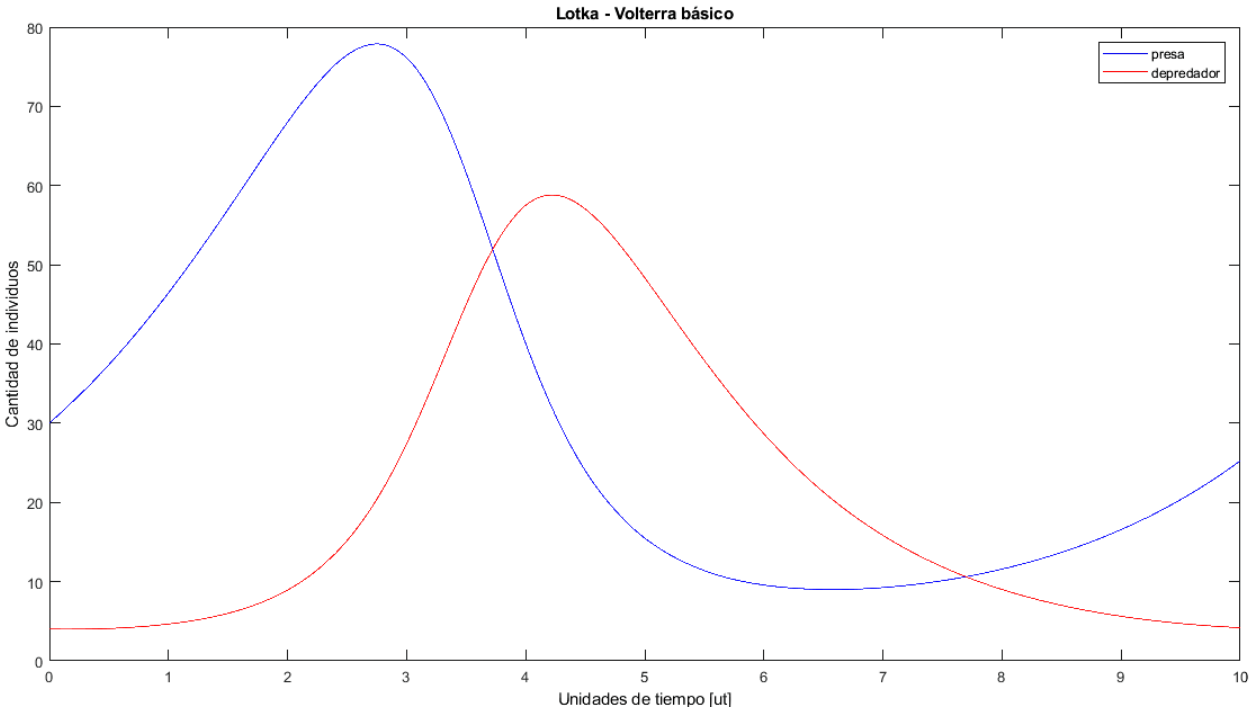


Figura 35. Acercamiento a un par de ciclos de la relación presa - depredador. Ver Anexo:Código, MATLAB, *Main-Lotka-Volterra* y *Lotka-Volterra*.

con más detalle esta relación. Por otra parte, en la figura 36 se observa el retrato de fase del sistema, el cual muestra un ciclo límite, correspondiente al comportamiento periódico.

En la simulación con agentes móviles se propuso continuar con lo establecido en Gómez y Vélez (2009), estableciendo 30 presas y 4 depredadores, para las poblaciones en t = 0, las posiciones iniciales se seleccionan de forma arbitraria, tal como se observa en la figura 37 y la tasa de nacimientos de presa y tasa de muerte de depredadores se mantienen (r_1 = 0.55 y r_2 = 0.83 respectivamente). Cabe destacar que en esta sección, se modificó el número de muestras de 10000 a 1000 con el fin de facilitar la ejecución de las mismas, ya que con la variable anterior el ordenador no respondía de forma adecuada.

En la figura 38 se puede apreciar cómo las poblaciones consiguen alcanzar un punto de aparente estabilización (con acercamiento en la figura 39) ya que a pesar de que existen instantes de tiempo en que la población depredadora desaparece, esta vuelve a resurgir en periodos espaciados a lo largo del resto de la prueba (ver figura

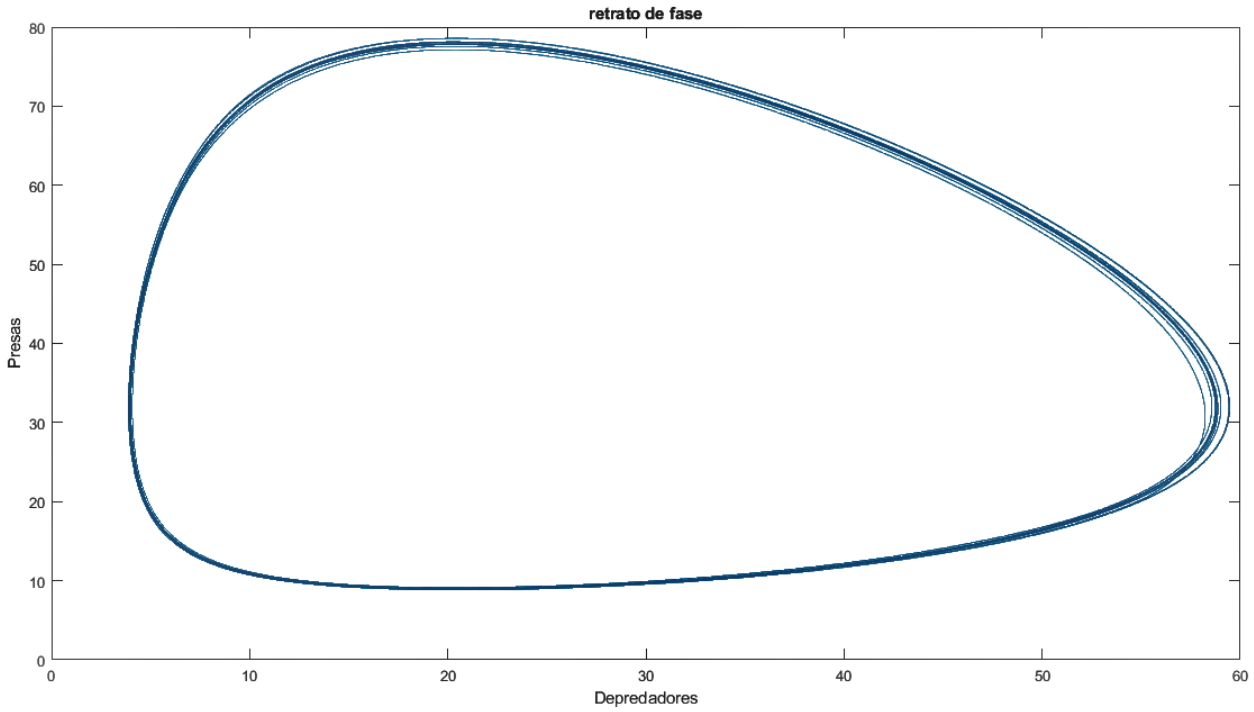


Figura 36. Retrato de fase para la simulación propuesta en el escenario Lotka-Volterra básico. Ver Anexo: Código, MATLAB, *Main-Lotka-Volterra* y *Lotka-Volterra*.

40), se podría teorizar que dichas reapariciones impide que la población de presas no alcance la sobre-población. En el caso del retrato de fase (a pesar de la modificación ya mencionada en el número de muestras) se obtuvo un gráfico en el que se puede contemplar como la mayoría de las trayectorias se concentran en una zona (ver figura 41).

3.2. Resultados obtenidos en el modelo con competencia entre depredadores

Haciendo uso de la ecuación (2), se procedió a sustituir los elementos de la misma con los datos proporcionados por el trabajo de Gómez y Vélez (2009). Para el factor añadido, "c_1", el cual se tomará como factor de competencia intraespecífica, se propuso un valor arbitrario pequeño de $c_1 = 0.009$.

Los resultados obtenidos se muestran y describen a continuación:

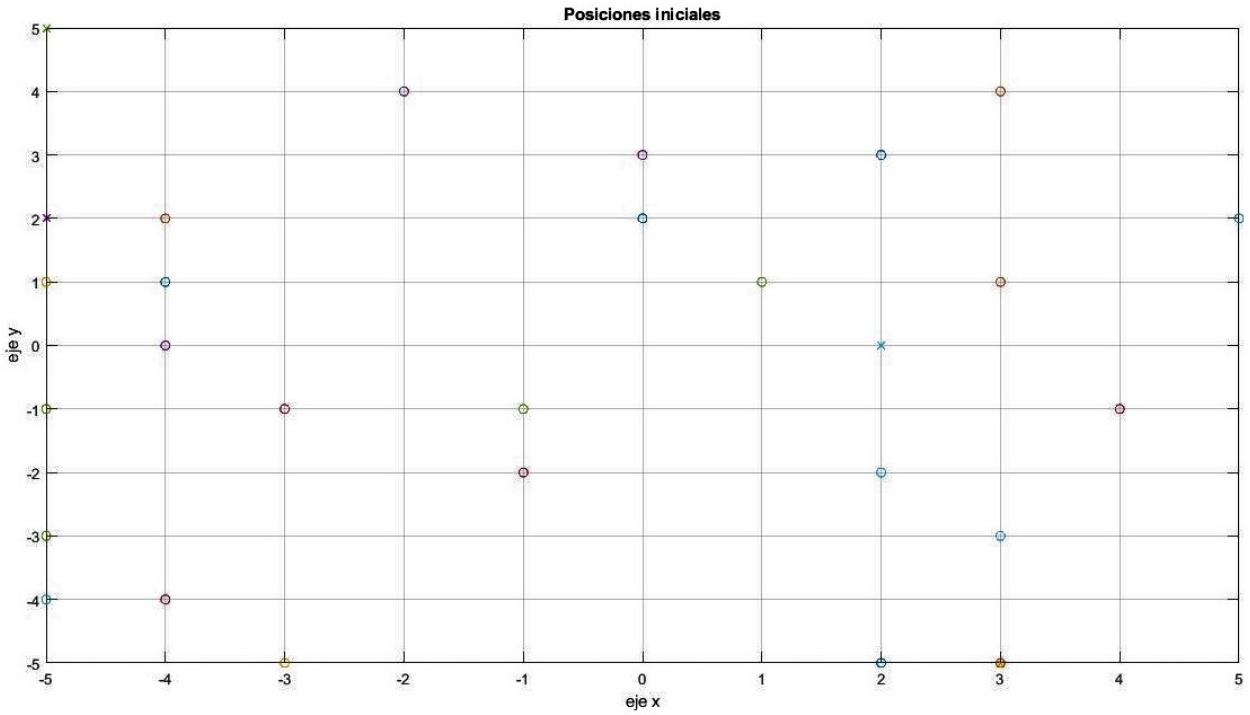


Figura 37. Posiciones iniciales de las presas (O) y los depredadores (x). Código disponible en Anexo: Código, MATLAB, *Simulación, presa-depredador con desplazamiento.*

En la figura 42, se puede apreciar como las poblaciones de presas y depredadores se estabilizan con el tiempo hasta alcanzar las cantidades de 20.33 presas (línea roja) y 38.95 depredadores (línea azul). Dicho punto fijo también puede apreciarse en las gráficas de retrato de fase en las figuras 43 y 44.

Sin embargo, en el caso de la prueba que tomaba en cuenta el movimiento de los individuos del grupo fue necesario un ajuste en los factores de tasa de muerte del grupo depredador y en la competencia intraespecífica, ya que los valores anteriores fueron demasiado altos para este escenario, como se puede ver en las figuras 45 y 46.

Por tanto, se modificaron esos factores a $b_1 = 0.011$ taza de muerte del depredador y $c_1 = 0.000298$ (factor de competencia intraespecífica), respectivamente, el número de iteraciones se mantiene igual que en el modelo anterior. Los resultados obtenidos tras la modificación ya mencionada se muestran en las figuras 47, 48 y 49. En este caso, tanto en el registro de cambio en la población como en el retrato de fase, las gráficas nos muestran que, de forma similar a lo que obtuvimos en la prueba anterior, estas tienden a estabilizarse en cero.

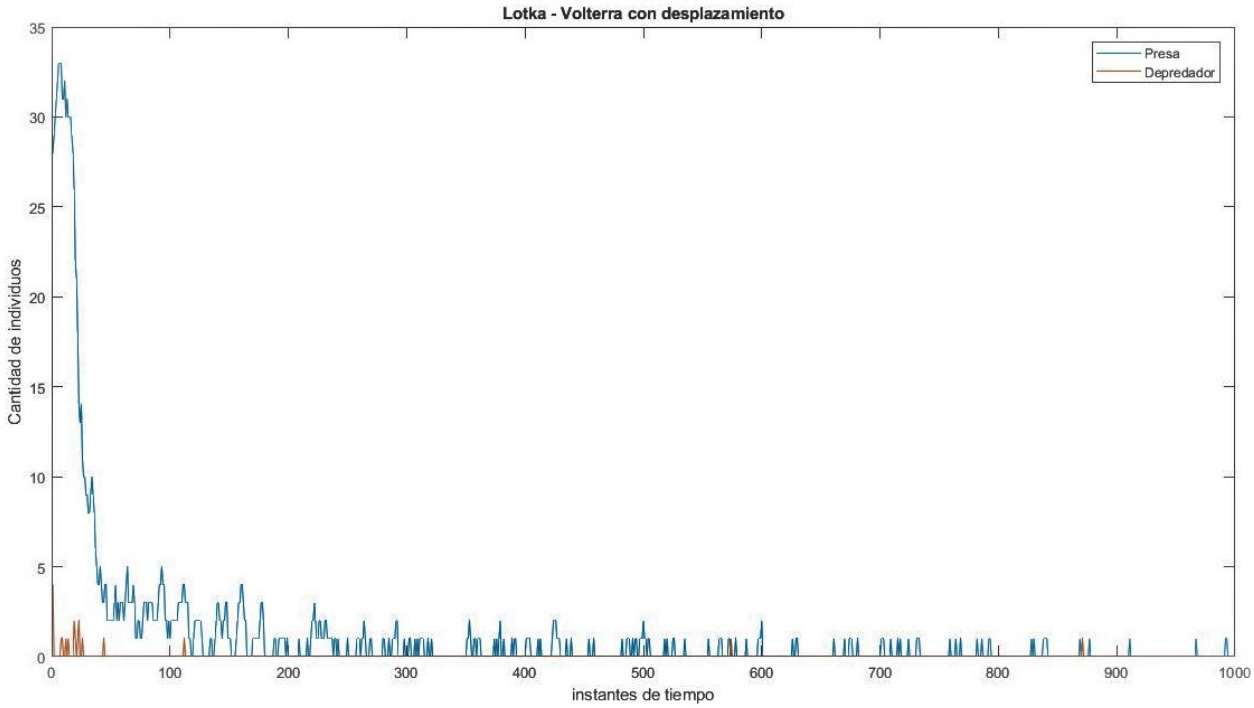


Figura 38. Cantidad de individuos en cada población para cada instante de tiempo de la simulación del modelo Lotka-Volterra básico con agentes que se desplazan. Código disponible en Anexo: Código, MATLAB, *Simulación, presa-depredador con desplazamiento*.

Resultados obtenidos del modelo con 3 especies, 2 depredadores y 1 presa

A continuación se presentan los resultados obtenidos al simular el modelo, considerando las siguientes poblaciones iniciales (propuestas arbitrariamente): $x(0) = 30$, $y(0) = 20$ y $z(0) = 8$. Los coeficientes del sistemas se establecieron de la siguiente manera: $a_1 = 2$, $b_1 = 0.05$, $\omega = 1$, $D = 10$, $a_2 = 1$, $\omega_1 = 2$, $D_1 = 10$, $\omega_2 = 1.45$, $D_2 = 10$, $c = 0.0257$, $\omega_3 = 1$ y $D_3 = 20$ basados en el texto de Rai (2004).

Para nuestro experimento, se seleccionaron las ecuaciones propuestas por la información anterior. que corresponden al modelo (5) y la ecuación (6):

$$\frac{dX}{dt} = a_1 X - b_1 X^2 - \frac{\omega YX}{X+D},$$

$$\frac{dY}{dt} = -a_2 Y + \frac{\omega_1 YX}{X+D_1} - \frac{\omega_2 Y^2 Z}{Y^2+D_2^2}, \tag{7}$$

$$\frac{dZ}{dt} = cZ^2 - \frac{\omega_3 Z^2}{Y+D_3}$$

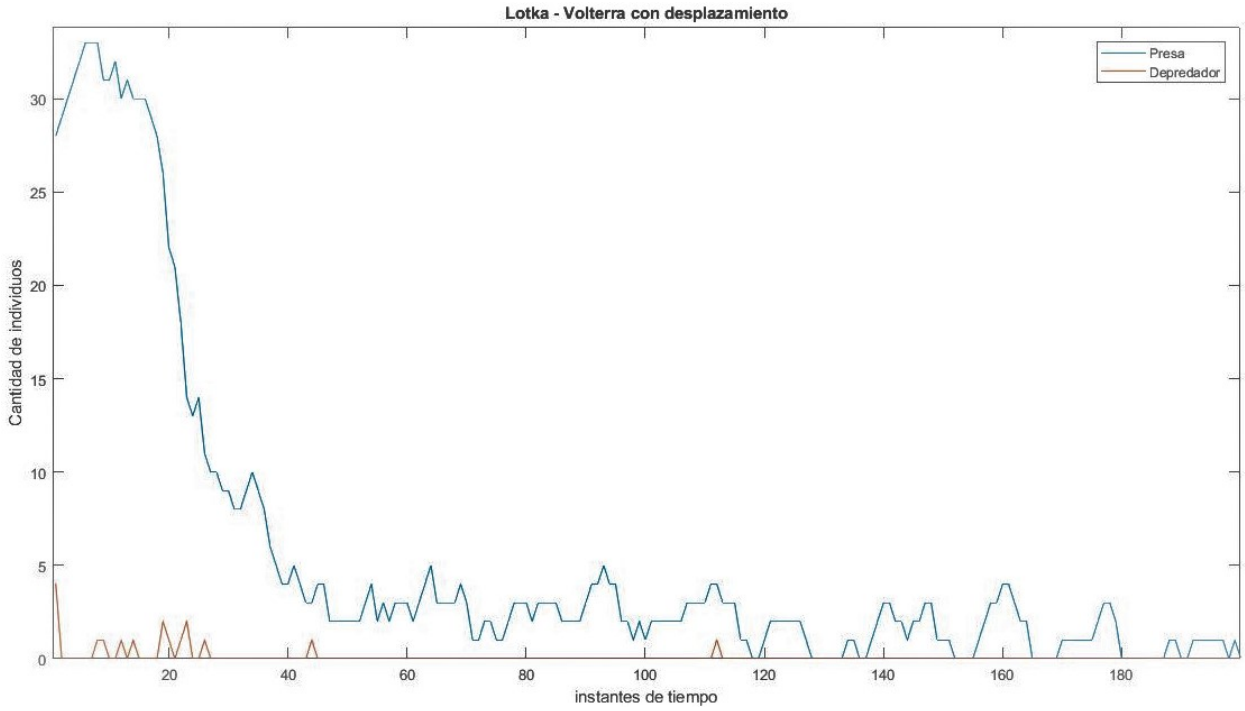


Figura 39. Cantidad de individuos en cada población para cada instante de tiempo de la simulación del modelo Lotka-Volterra básico con agentes que se desplazan (Acercamiento). Código disponible en Anexo: Código, MATLAB, *Simulación, presa-depredador con desplazamiento*.

Como ya se mencionó, corresponde a un escenario de una cadena trófica más completa, donde se consideran las interacciones de 3 especies (depredador tope (o depredador generalista), depredador especialista y una presa). El objetivo principal al estudiar esta cadena es identificar la aparición de *caos* en el sistema (7). Dentro de las condiciones necesarias para que se presenten dinámicas caóticas, se requiere que sea no lineal y de por lo menos 3 estados en su versión continua. Él modelo elegido se asemeja a lo descrito en la figura 1.

En la figura 50 se puede apreciar una respuesta diferente a lo visto en los escenarios anteriores. En este caso no se aprecia una respuesta periódica en el comportamiento de nuestras poblaciones de prueba, esto en conjunto con lo visto en la figura 51, en la que se puede ver un *atractor extraño*, nos indica la posibilidad de la presencia de caos. Por tanto, se procede a confirmar este hecho mediante el método descrito a continuación.

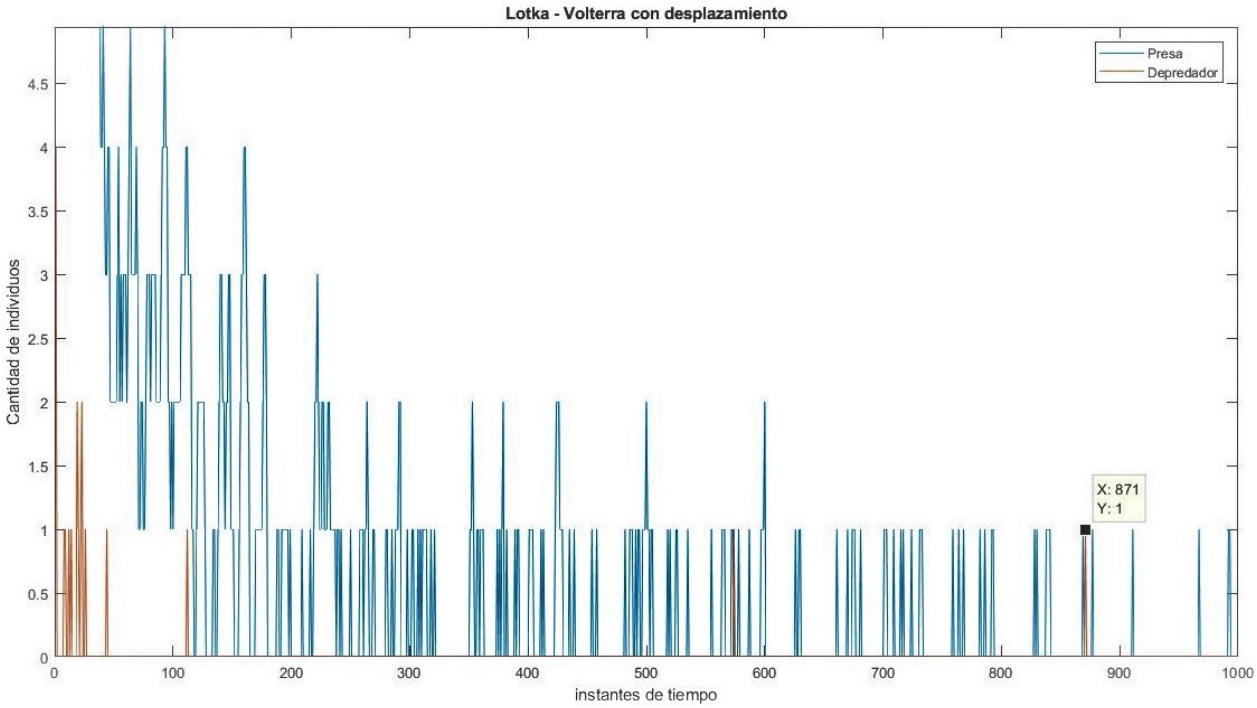


Figura 40. Cantidad de individuos en cada población para cada instante de tiempo de la simulación del modelo Lotka-Volterra básico con agentes que se desplazan (Acercamiento 2). Código disponible en Anexo: Código, MATLAB, *Simulación, presa-depredador con desplazamiento.*

3.3. Análisis de la existencia de caos (El código)

En diversos artículos se ha establecido que el análisis y la detección del caos es un aspecto complejo. En nuestro caso, se optó por hacer uso de los exponentes de Lyapunov. Para lo cual, se recurrió a lo mencionado en el artículo elaborado por Wolf (2013) y que será explicado de forma resumida a continuación:

“El programa FET (y su preprocesador BASGEN), permite estimar el exponente de Lyapunov en cualquier serie de tiempo (p.4).”

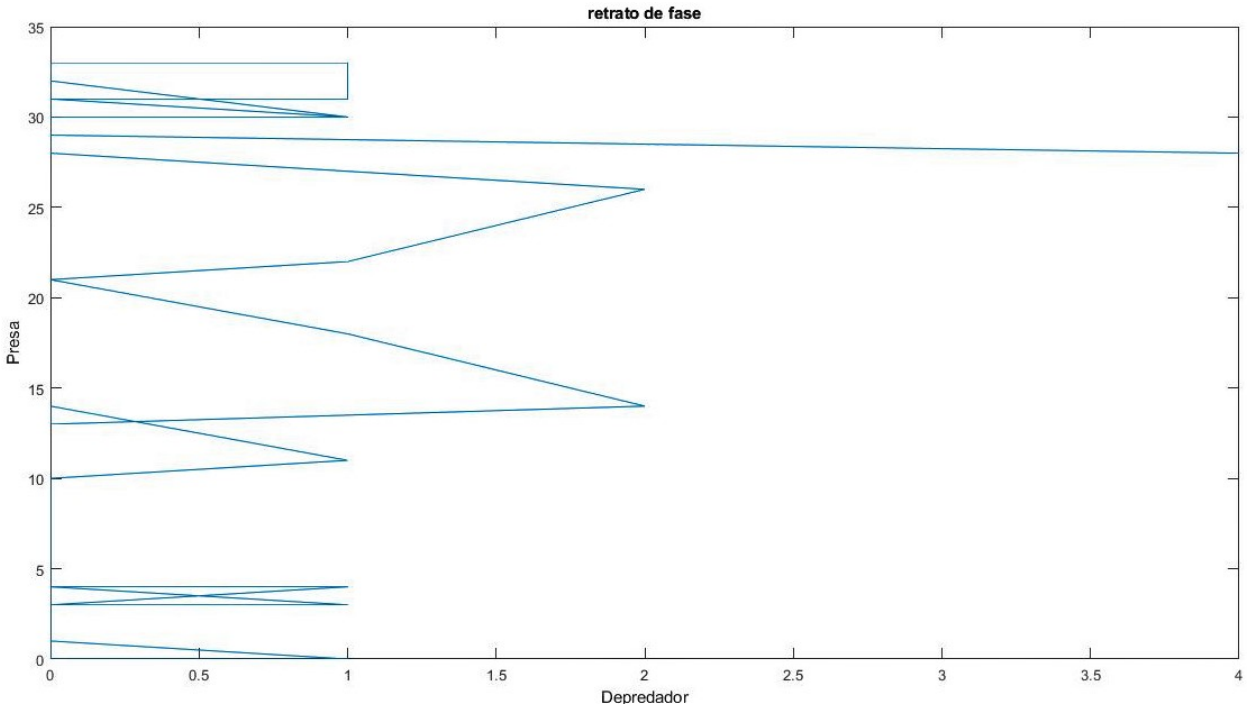


Figura 41. Retrato de fase de la simulación, Lotka-Volterra, propuesta con agentes que se desplazan. Código disponible en Anexo: Código, MATLAB, *Simulación, presa-depredador con desplazamiento.*

FET Y BASGEN hacen uso del método de la reconstrucción del retraso en el tiempo (también conocido como reconstrucción retrasada y reconstrucción en el espacio fase.

La reconstrucción con retraso constituye una “órbita” n-dimensional fuera de la serie de tiempo, una vez que el usuario selecciona 2 parámetros: la dimensión de incrustación y el retraso en el tiempo “tau” (p. 5).

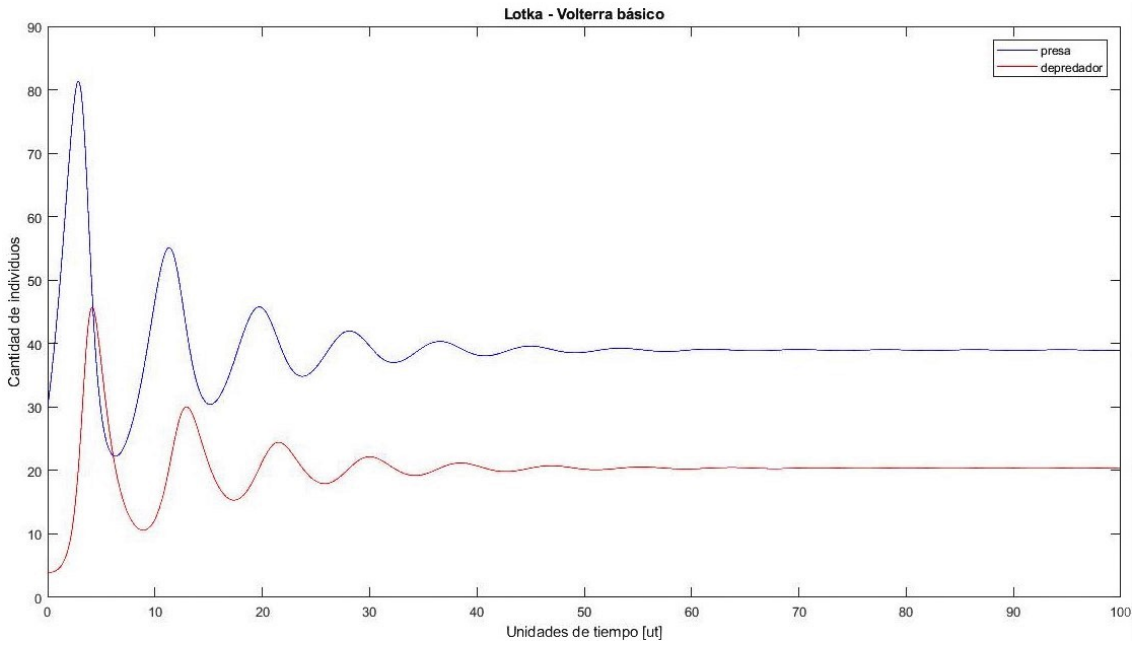


Figura 42. Comportamiento a través del tiempo de las poblaciones de presa y depredador en el escenario de competencia intraespecífica. Ver Anexo:Código, MATLAB, Lotka-Volterra, Competencia intraespecífica y Competencia intraespecífica...CONT.

Una consecuencia importante del procedimiento del retraso es que el tamaño (extensión lineal) de la órbita reconstruida en cada una de las n-dimensiones es igual a la diferencia entre la mayor y la menor de las cantidades en la serie de tiempo original. El resultado de la reconstrucción gráfica puede visualizarse como si se tomara un largo trozo de espagueti (la serie de tiempo) y se retorciera en un plato para definir un camino a través de una región acotada del espacio, retorciendo el trozo, segmentos de la región de la serie de tiempo que estaban muy lejos (temporalmente) pueden ser traídos a una aproximación cercana (espacialmente).

Algunas de las propiedades importantes de la reconstrucción con retraso son (p. 5):

1. Las series de tiempo periódicas se vuelven órbitas cercanas de fase espacial.
2. Una serie de tiempo infinita de ruido blanco puede convertirse en una gráfica de dispersión que podría llenar por completo un espacio n-dimensional.
3. Las series de tiempo caóticas se reconstruyen en órbitas fractales cuyos segmentos orbitales exhiben dependencia sensible.

En el artículo de Wolf (2013), se expresa que “al realizarse la reconstrucción, el usuario elige el valor de la n-dimensión por el método de “adivinación” o “suposición”

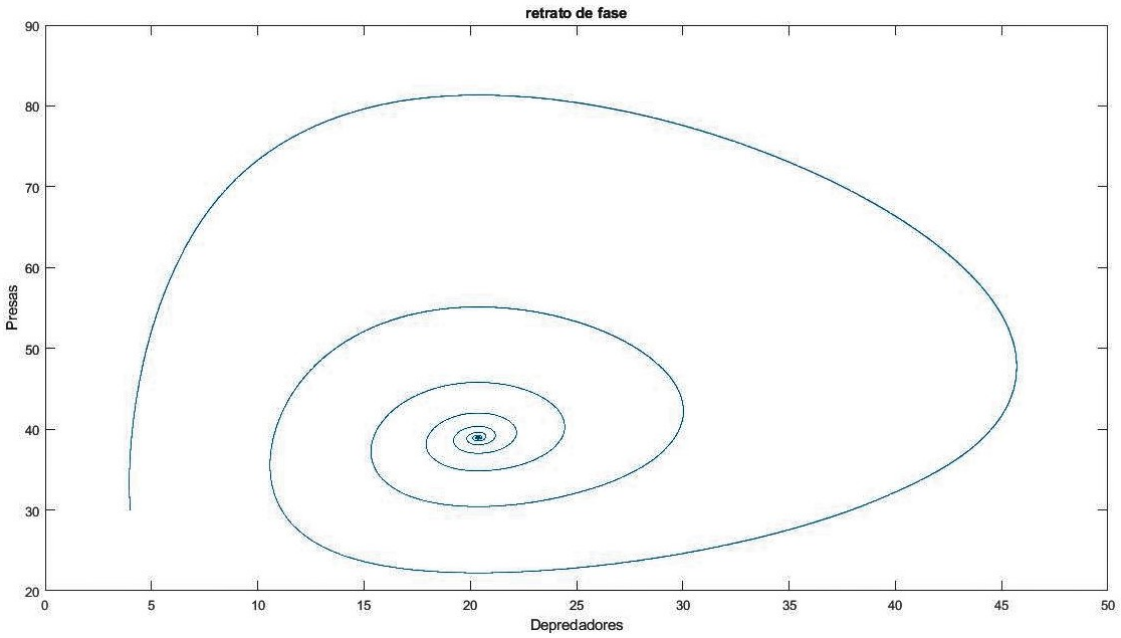


Figura 43. Retrato de fase, Lotka-Volterra competencia intraespecifica. Se observa la estabilización en un punto. Ver Anexo: Código, Lotka-Volterra, Competencia intraespecífica.

para asegurar que la órbita sea topológicamente razonable en n-dimensiones.” Y hace referencia a que “una elección óptima de tau “engordaría” la órbita de la reconstrucción retrasada de tal forma que tenga una apariencia lo más simple posible. Entre más simple sea la estructura, mayor sera la precisión del FET al cuantificar la divergencia orbital”. De igual forma se expresa que “la dependencia de *tau* de las estimaciones del exponente de Lyapunov es casi siempre muy débil, a menudo se elige *tau* como el número de puntos de datos que corresponden a 1/3 del periodo medio de la órbita reconstruida” (p.5).

¿Como funcionan?

Wolf (2013) expresa que “FET estima el exponente de Lyapunov dominante (el mayor positivo) en la serie de tiempo. Esto se consigue obteniendo el promedio de la tasa exponencial de divergencia de segmentos cortos de la órbita reconstruida con retraso” (p.6).

Su funcionamiento, según Wolf (2013) es el siguiente (p. 6):

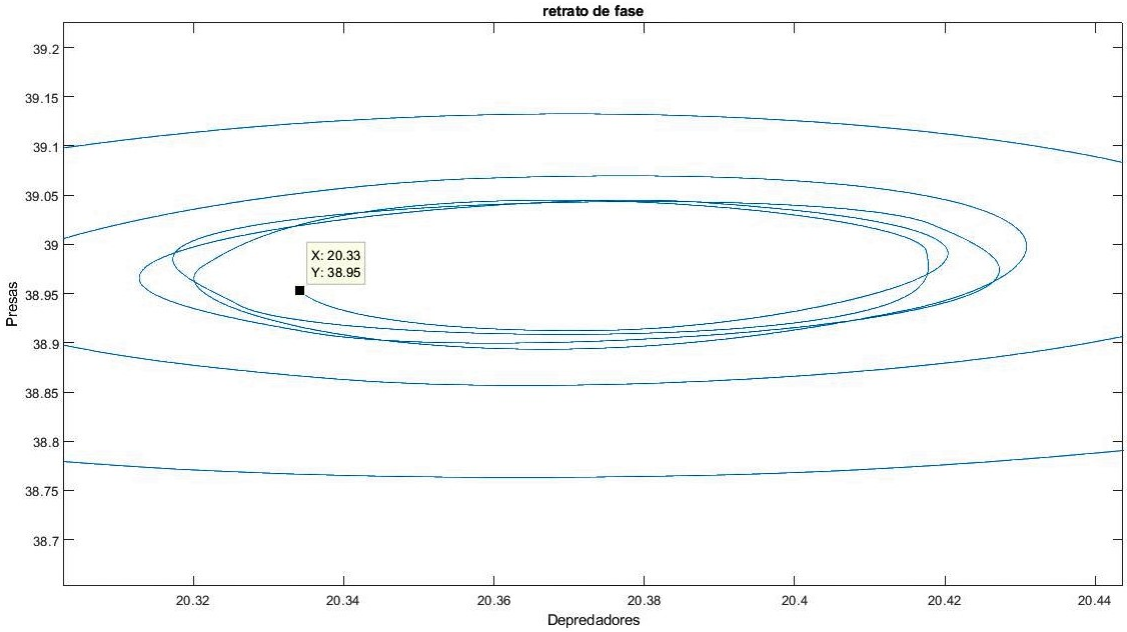


Figura 44. Retrato de fase, Lotka-Volterra competencia intraespecifica. Se observa la estabilización en un punto (acercamiento). Ver Anexo:Codigo, Lotka-Volterra, Competencia intraespecífica.

1. FET crea una órbita de espacio de fase multidimensional a partir de una serie de tiempo unidimensional mediante la reconstrucción de retardo.

2. Haciendo uso de la base de datos creada por BASGEN, FET localiza un par de puntos que se encuentren cercanos uno del otro en la órbita de espacio de fase reconstruida.

3. FET sigue cada uno de los puntos conforme viajen una distancia corta a lo largo de la órbita de espacio de fase. Se puede comparar la separación inicial (distancia ordinaria euclideana) de los puntos a su separación al final del intervalo. El algoritmo (base 2) del radio de la separación final a la inicial de estos puntos es una estimación local de la órbita divergente.

4. Si los 2 puntos continúan bastante juntos al final de este intervalo, se conservan ambos, se desarrollan un poco más a lo largo de la órbita y se calcula el siguiente valor local de divergencia orbital. Si los puntos se han alejado mucho más, se mantiene uno de los puntos y se utiliza la base de datos para encontrar un reemplazo adecuado para el otro punto.

5. Al promediar las tasas locales de divergencia orbital y dividir por el tiempo total de viaje a lo largo de la órbita, se obtiene la tasa promedio de divergencia a largo plazo de las órbitas cercanas. La palabra cercana es importante, ya que las contribuciones provienen de segmentos orbitales que están razonablemente juntos en todo momento.

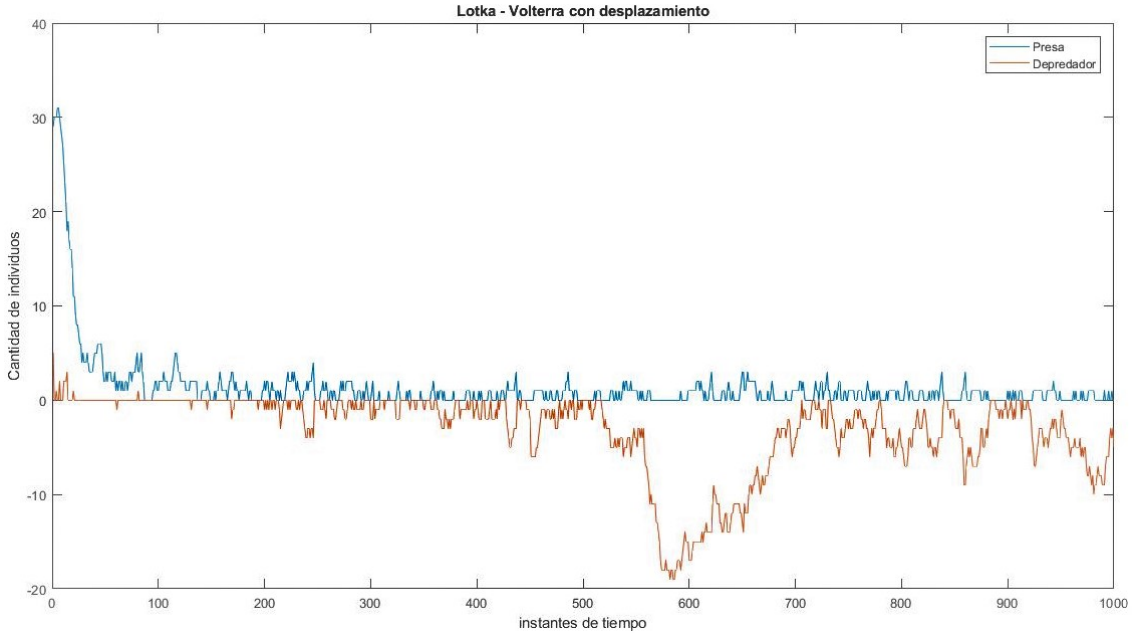


Figura 45. Cambio en poblaciones, competencia intraespecífica, datos originales.

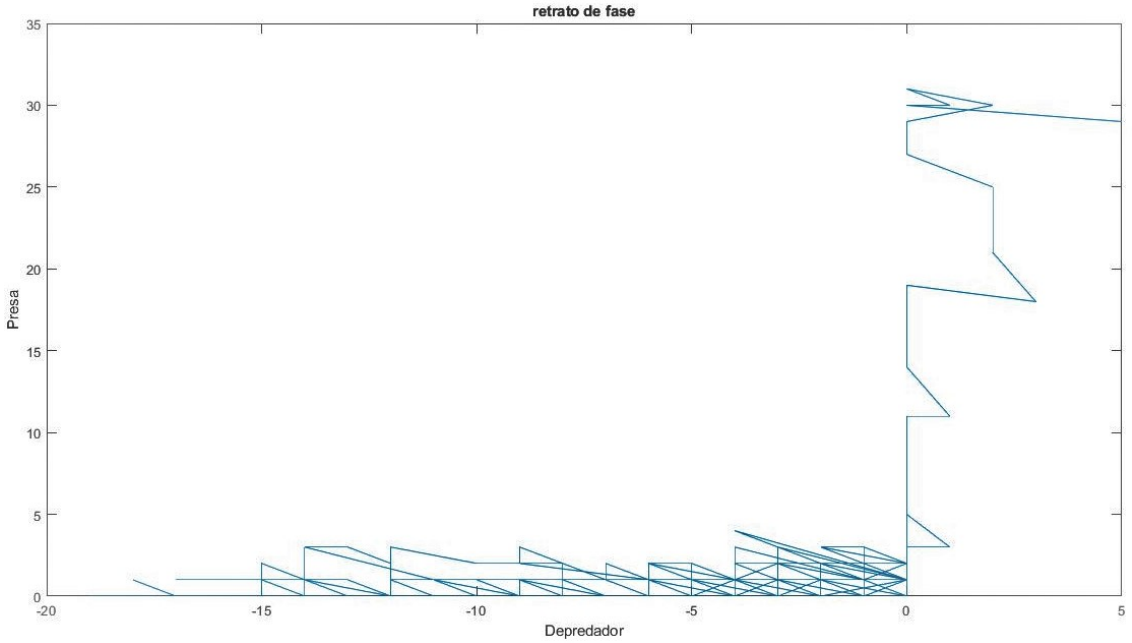


Figura 46. Retrato de fase, competencia intraespecífica. Datos originales.

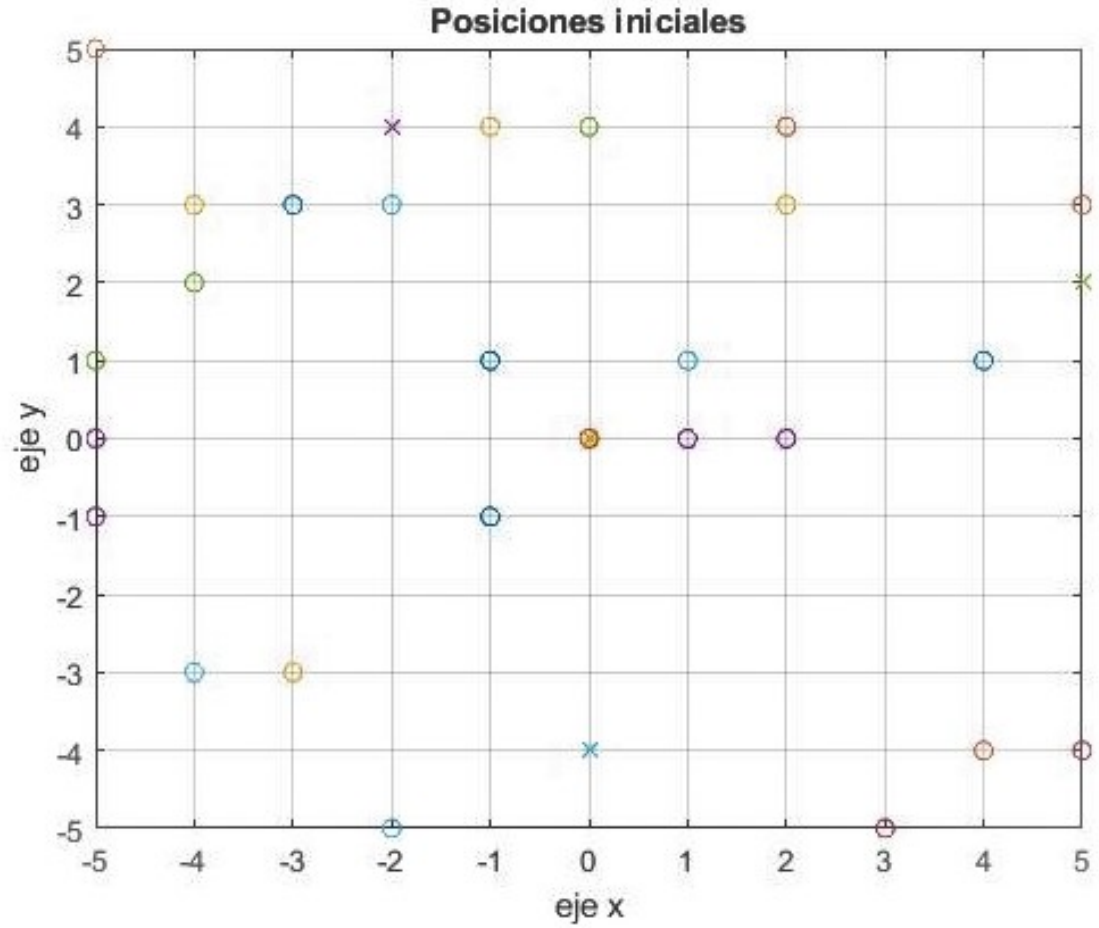


Figura 47. Posición inicial (“o”= presas, “x” depredadores), competencia intraespecífica. Datos modificados Código disponible en Anexo, Lotka-Volterra, Competencia intraespecífica, con movimiento.

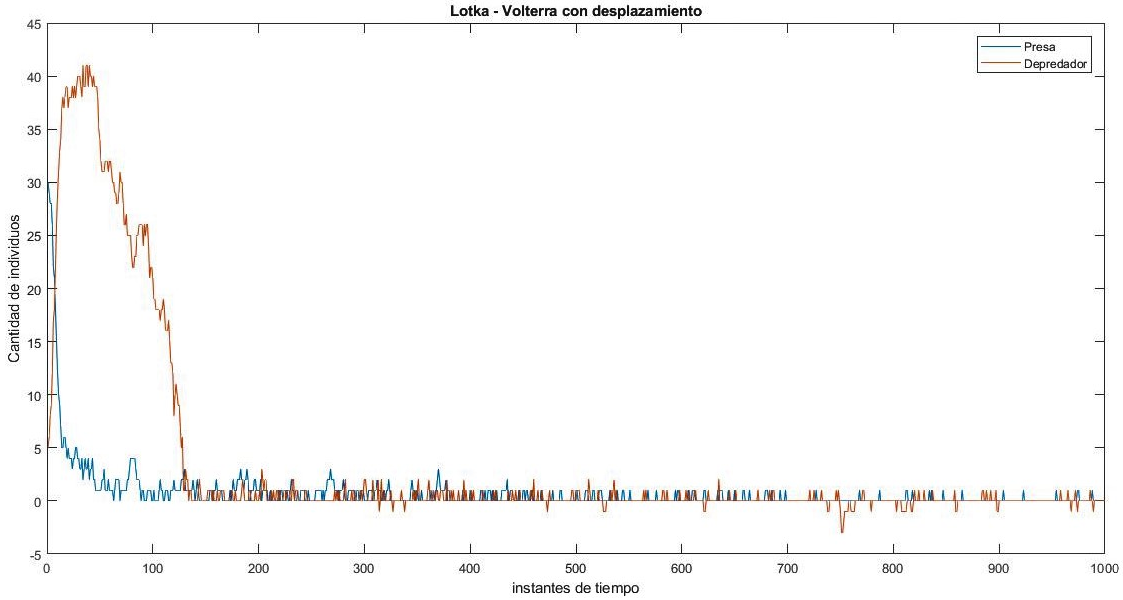


Figura 48. Registro de cambios en población, competencia intraespecífica. Datos modificados. Código disponible en Anexo, Lotka-Volterra, Competencia intraespecífica, con movimiento.

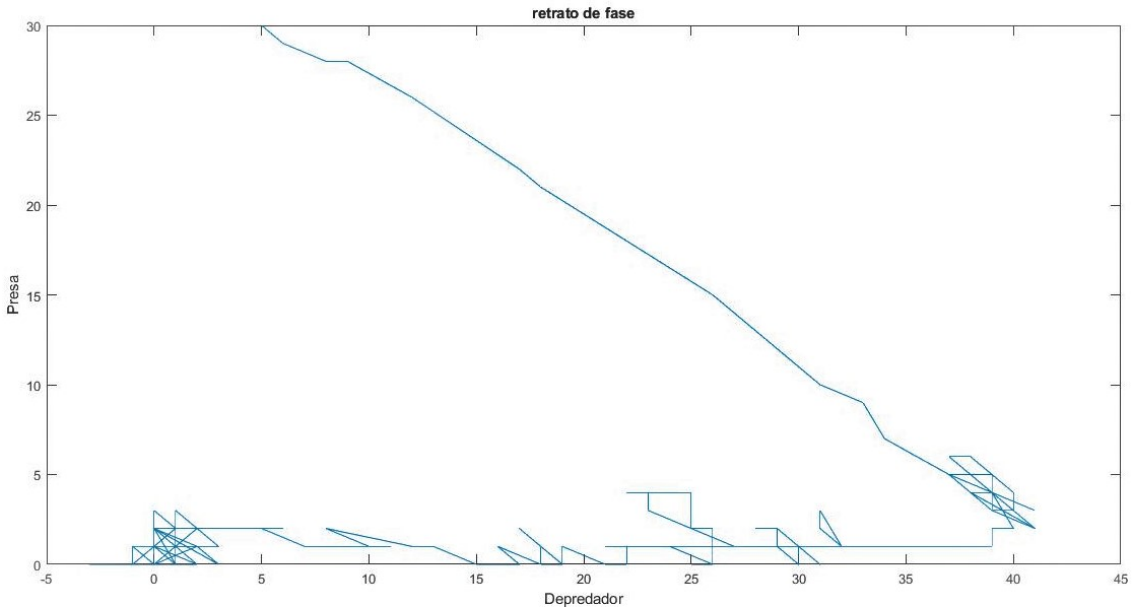


Figura 49. Retrato de fase, competencia intraespecífica. Datos modificados. Código disponible en Anexo, Lotka-Volterra, Competencia intraespecífica, con movimiento.

El documento luego expresa que:

> FET presenta su estimación corriente del exponente dominante de Lyapunov numéricamente en la parte superior de la pantalla y luego la escribe en el archivo "fet.out", los segmentos de divergencia orbital se muestran gráficamente durante el cálculo como una verificación de su solidez. Este documento contiene volcados de gráficos que muestran la apariencia de segmentos “buenos” y “malos” (p. 6).

> La función de BASGEN es crear una base de datos de la serie de tiempo proporcionada por el usuario de tal forma que FET pueda utilizar las series de tiempo de forma más eficiente. Con la serie de tiempo original y la base de datos, FET puede rápidamente localizar todos los puntos que están cerca a cualquier punto especificado en un espacio fase reconstruido (p. 7).

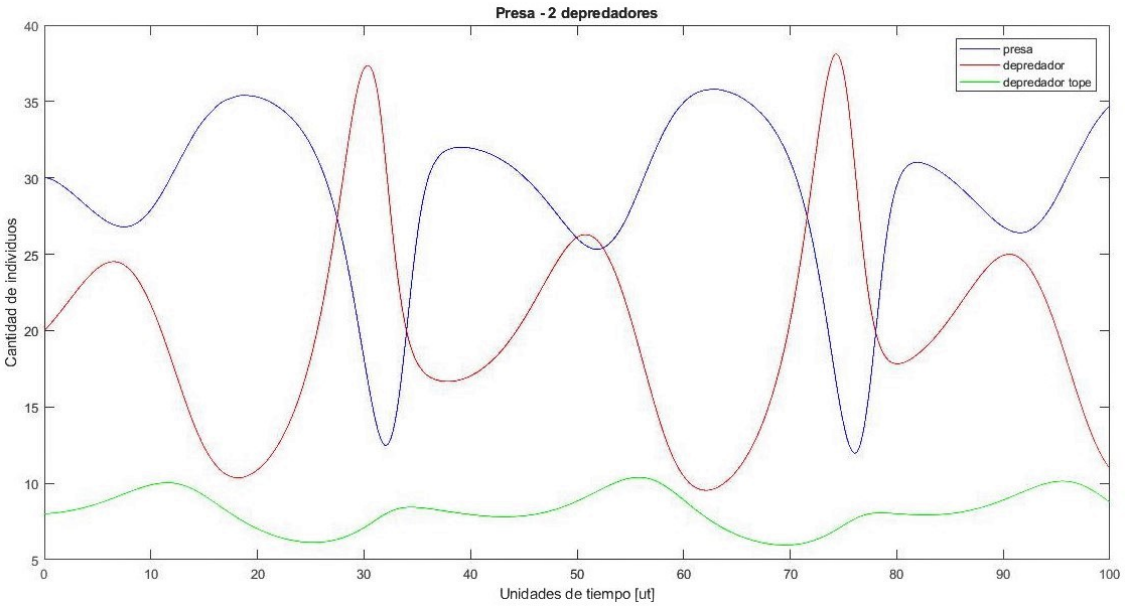


Figura 50. Comportamiento de los individuos a través del tiempo para el modelo simulado de tres especies. Ver Anexo: Código, MATLAB, Lotka-Volterra, 3 especies.

BASGEN coloca los puntos de la órbita reconstruida en una cuadrícula n-dimensional con cuadros de ires por lado. De los cuadros ocupados ires n-dimensionales, la basta mayoría probablemente se encuentre vacía, por tanto BASGEN utiliza un arreglo virtual para definir la cuadrícula- solo contenga cuadros con uno o más puntos consumido de memoria. ires es elegido por el usuario para que el promedio de cuadros ocupados contenga un razonable número de puntos, por ejemplo 50. Cuando FET necesita los puntos cercanos a un punto específico en el espacio fase, se determina en qué cuadro se encuentra dicho punto, y se busca en la base de datos. La base de datos regresa una lista de todos los puntos contenidos en dicho cuadro, dado como las posiciones de la primera coordenada en la serie de tiempo original. FET ahora trabajará con los “50” puntos de ese cuadro, en vez de los miles de puntos en la serie de tiempo (p. 7).

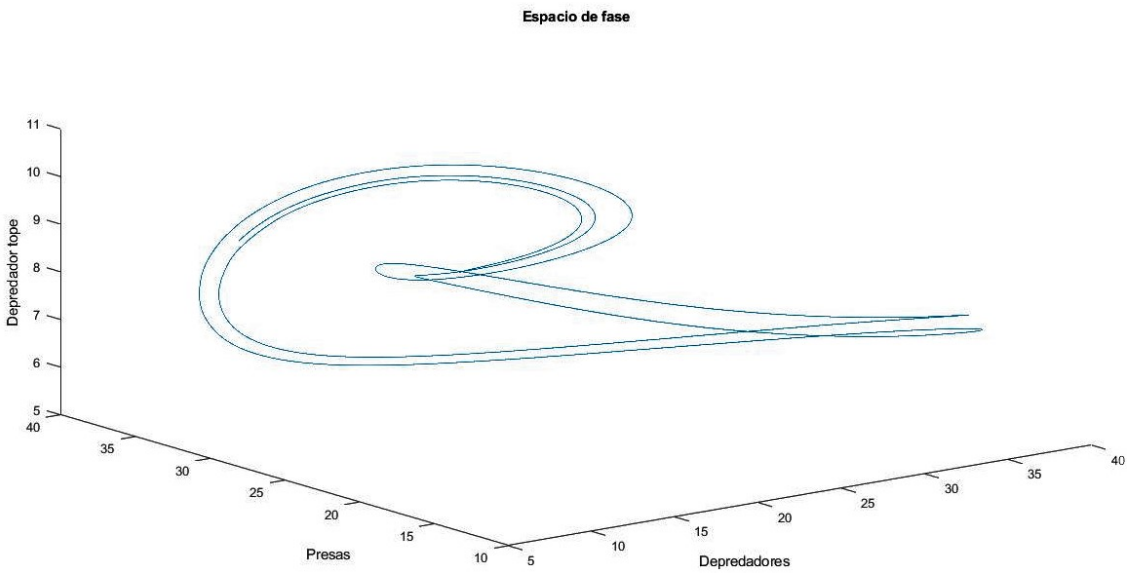


Figura 51. Retrato de fase para el modelo simulado de tres especies. Ver Anexo: Código, MATLAB, Lotka-Volterra, 3 especies.

Interpretación gráfica y texto de salida:

La salida del archivo FET es exactamente la misma que la salida de texto en la pantalla durante la ejecución del programa. La primera columna es la ubicación actual dentro de la órbita reconstruida, que aumentaba en pasos de evolución. La segunda y tercera columna son la distancia entre los 2 puntos al principio y al final del segmento actual. La cuarta columna es la estimación corriente del exponente de Lyapunov. La quinta columna contiene el cambio en la orientación angular (en grado) en un reemplazo (p. 16).

3.4. Análisis de la existencia de caos (La prueba)

La comprobación de la existencia de caos se llevo a cabo de la siguiente forma. En primer lugar procedemos a guardar la serie de un estado en la forma de archivos *.txt*, los datos se separan en *Presas, Depredadores y Depredadores tope*, ver figura 52, los cuales luego se procede a ingresar en el código de la sección *testbench.m* (ver, Anexo MATLAB, Análisis de caos, testbench), al correr el programa se obtienen las correspondientes gráficas de reconstrucción del diagrama de fase ver figuras 53, 54 y 55. El exponente de Lyapunov puede ser encontrado en un archivo denominado *fetout.txt*,

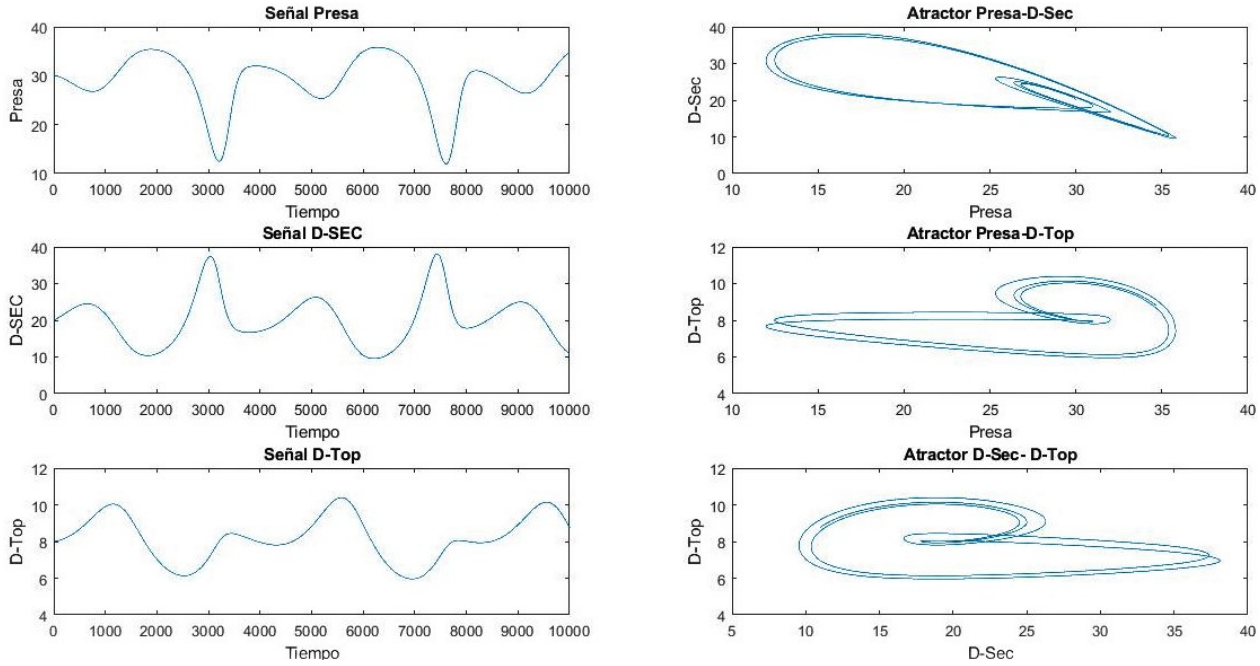


Figura 52. Gráficas comparativas para el modelo simulado de tres especies. Ver Anexo: Código, MATLAB, Lotka-Volterra, 3 especies.

en la forma de una matriz de 5 columnas, la cual ya fue descrita con anterioridad. El número de muestras corresponde a 10,000. No obstante en la tabla 2 se muestran los últimos valores de cada uno de los archivos ”.txt´´.

Tabla 2. Resultados, prueba para comprobar la existencia de caos. Se exponen los valores presentes al final del documento ”.txt´´ generado luego de cada prueba.

Categoría	C-1	C-2	C-3	C-4	C-5
PRESA	9993	0.0068	0.0079	0.2089	
DEPREDADOR	9993	0.0034	0.0103	0.2664	13
DEPREDADOR TOPE	9993	0.0094	0.0106	0.0549	

Para concluir de forma satisfactoria con esta prueba, es necesario repetirla con un escenario en el que el modelo no presente caos. Esto puede ser posible analizando lo dicho en el documento de Rai (2004), en éste se expone que “el modelo matemático sólo presenta dinámica caótica cuando α_2 tiene el valor de 1. Para otros valores el caos no puede ser encontrado” (p.133).

La característica principal de los resultados es la falta de signos negativos. Lo cual puede interpretarse como una señal significativa de la presencia de caos. Por tanto, a fin de corroborar que nuestra suposición y los resultados son correctos, se procedió a realizar la segunda prueba, tomando como referencia lo dicho en Rai (2004). A continuación se mostraran los resultados obtenidos al modificar el parámetro anterior a uno que permita la ausencia de caos.

Se procedió a modificar el modelo matemático (presentado anteriormente en la sección 3.2) para asignarle el valor de $\alpha_2 = 3$ (valor elegido de forma arbitraria), las gráficas obtenidas con ese nuevo elemento se presentan en las figuras 56, 57 y 58. De igual forma se generó un nuevo conjunto de archivos “.txt”, en la tabla 3, se reportan algunos de los valores correspondientes.

Tabla 3. Resultados, prueba escenario sin caos. Se exponen los valores del documento ”.txt´´ generado luego de cada prueba.

Categoría	C-1	C-2	C-3	C-4	C-5
PRESA	816	0.0000	0.0000	-0.0081	
DEPREDADOR	9993	0.0000	0.0000	-0.0082	
DEPREDADOR TOPE	9993	0.0002	0.0002	-0.0001	

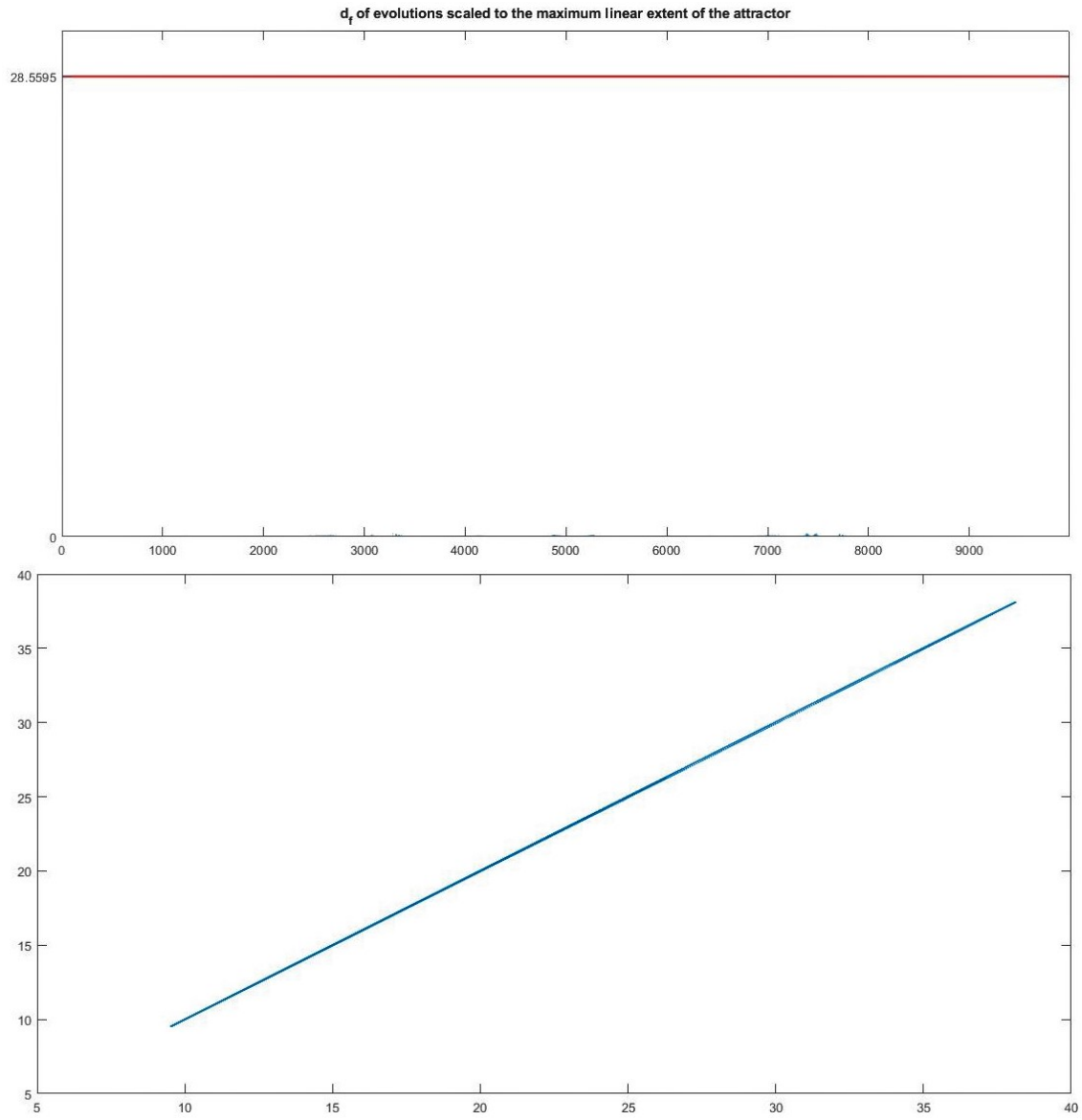


Figura 53. Análisis de caos, depredador. Gráficos obtenidos luego de ejecutar el código de MATLAB. Ver Anexo: Código, análisis de caos.

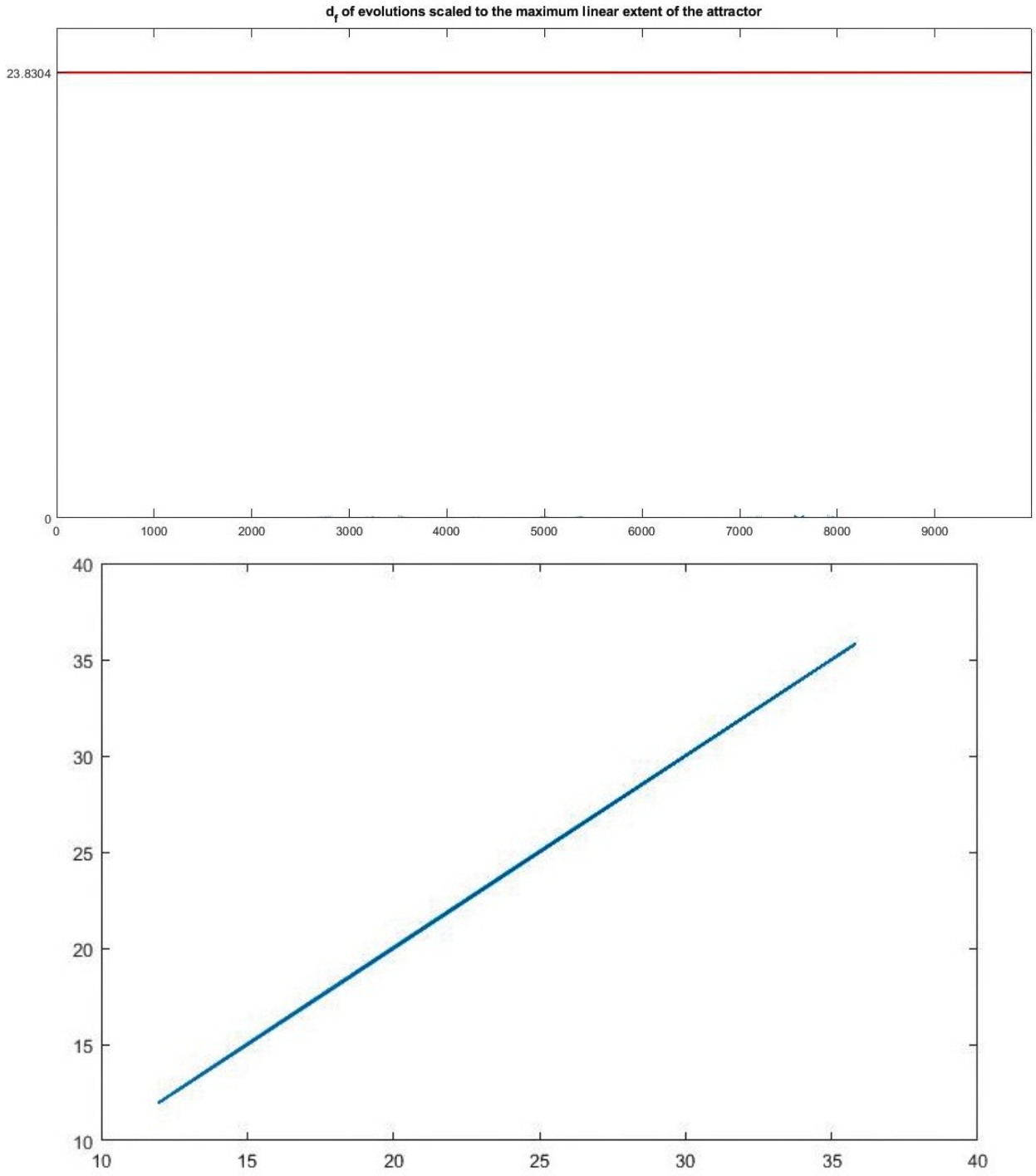


Figura 54. Análisis de caos, presa. Gráficos obtenidos luego de ejecutar el código de MATLAB. Ver Anexo: Código, MATLAB, Análisis de caos.

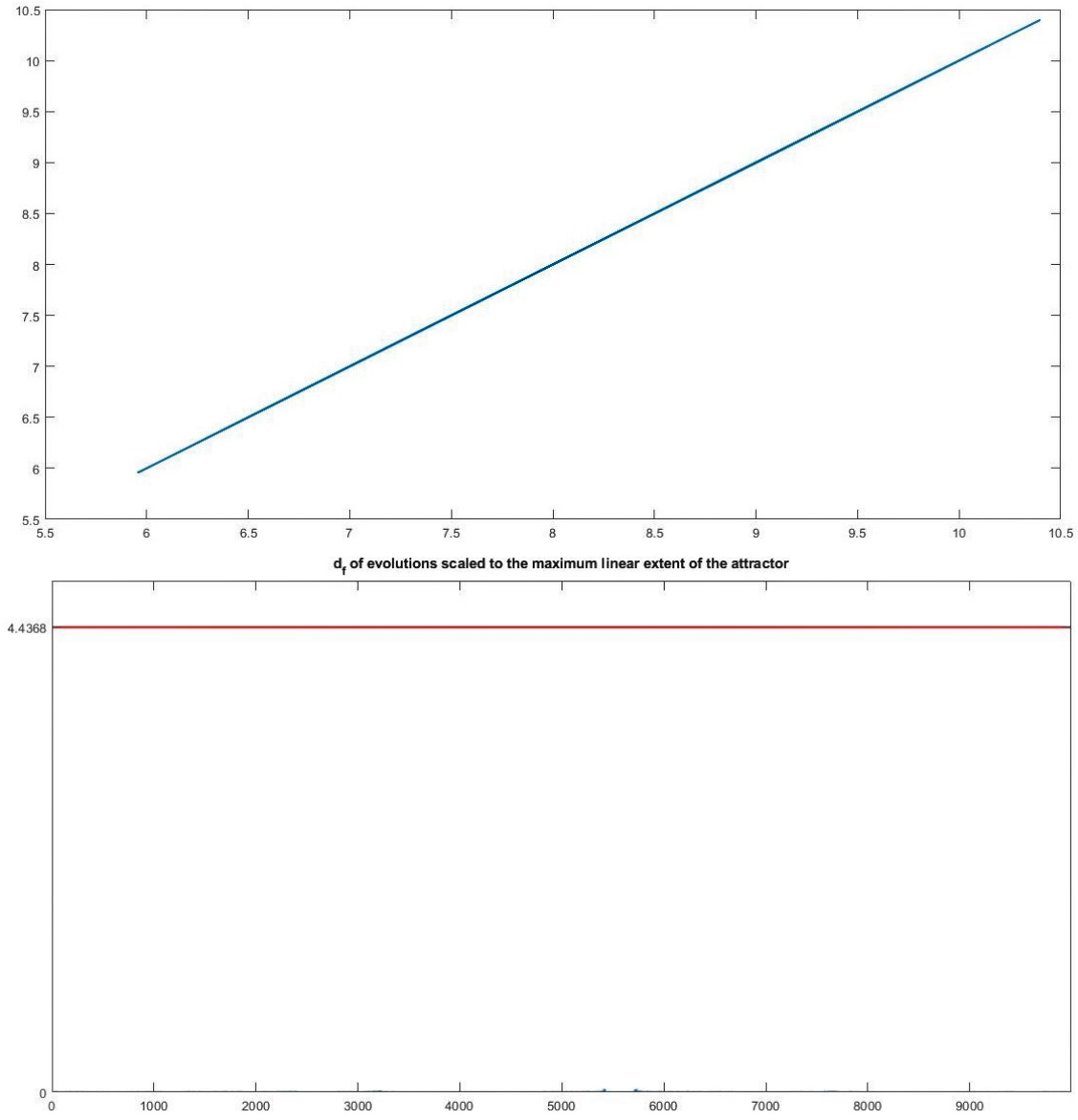


Figura 55. Análisis de caos, depredador tope. Gráficos obtenidos luego de ejecutar el código de MATLAB. Ver Anexo: Código, MATLAB, Análisis de caos.

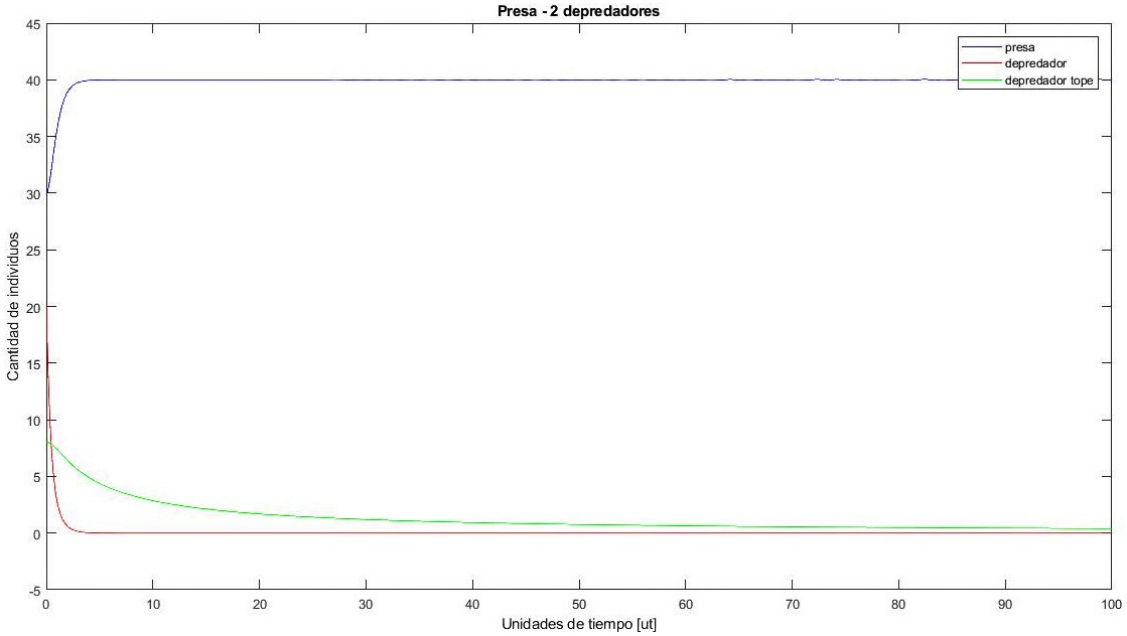


Figura 56. Escenario sin caos, $\alpha_2 = 3$, Cambios en las poblaciones.

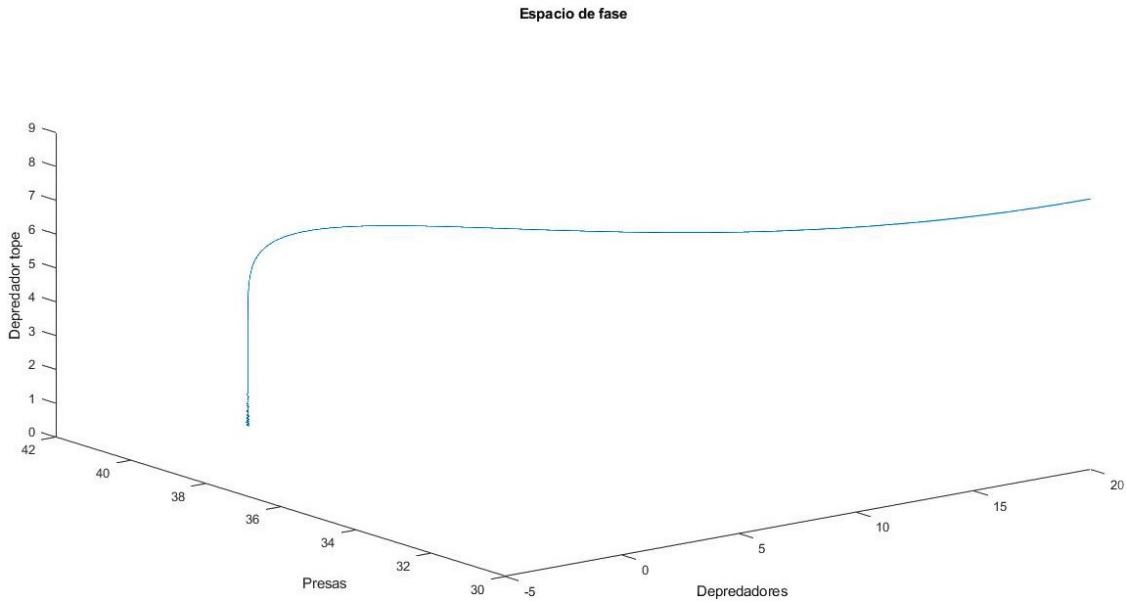


Figura 57. Escenario sin caos, $\alpha_2 = 3$, espacio de fase.

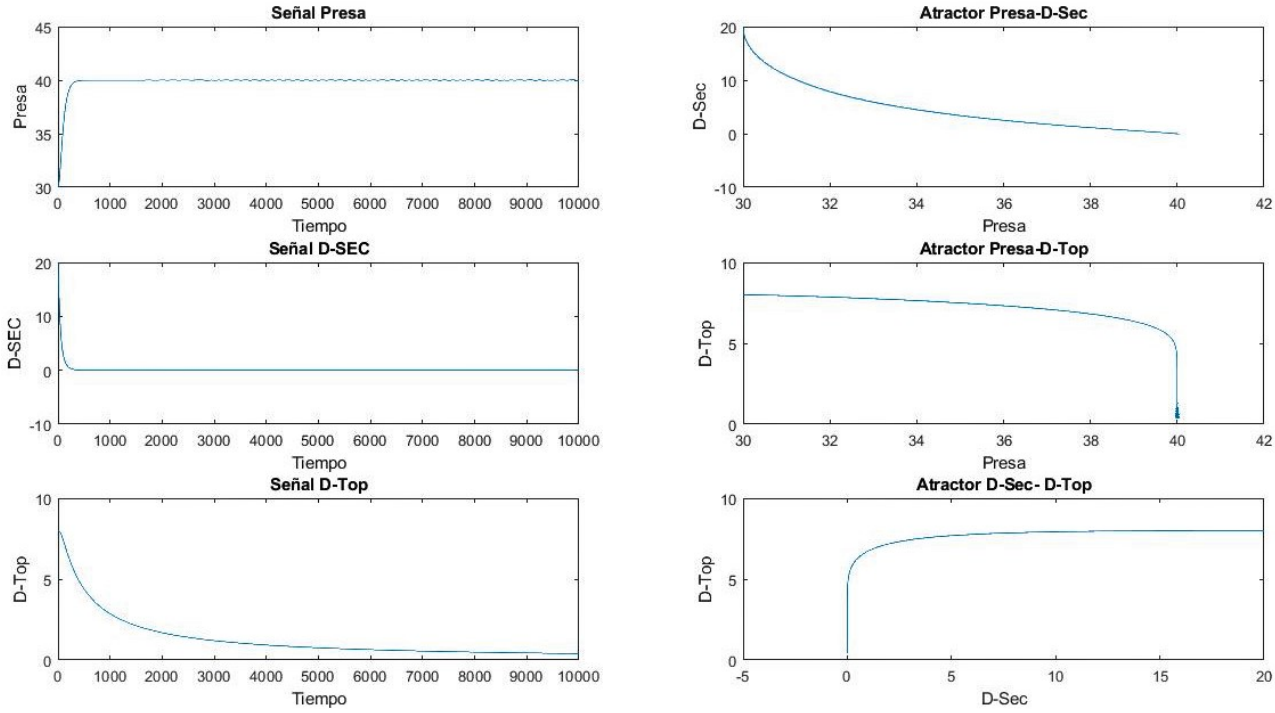


Figura 58. Escenario sin caos, $\alpha_2 = 3$, gráficas comparativas.

Capítulo 4. Discusión

4.1. La obtención y ausencia de caos

Los resultados de la comprobación de la presencia y ausencia de caos, nos permiten suponer que lo obtenido durante nuestras pruebas puede tratarse en efecto de caos. Ya que los resultados obtenidos del código de Wolf (2013), nos ofrecen mas evidencia que nos permite sustentar nuestra anterior suposición.

Sin embargo, además de lo anterior la clara diferencia entre lo que puede ofrecer un escenario caótico y no caótico es una gran referencia durante el estudio de estos casos, por tanto es recomendable el contar con información que pueda servir de base durante estos análisis. En este caso Rai (2004), describe que los modelos presentados en su trabajo presentan un comportamiento al borde del caos, por lo que se requieren de condiciones analizadas de forma que se pueda evitar resultados falsos. Es por ello que en este caso, se optó por utilizar los valores sugeridos en los experimentos del ya mencionado artículo, a fin de tener una referencia lo mas confiable posible, así como también comprender mejor el funcionamiento de estos escenarios.

Durante el experimento, nuestro escenario no caótico presentó gráficas en las que la señal tiende a estabilizare, además de presentar exponentes de Lyapunov negativos en sus archivos “.txt”.

En caso contrario, el escenario caótico, como se mencionó con anterioridad, presentó atractores extraños en sus diagramas de espacio de fase y sus gráficas de población no mostraban un comportamiento que diera a entender que pudiera estabilizarse, además de valores positivos en sus archivos de exponentes de Lyapunov.

4.2. ¿Qué se podría agregar, corregir o mejorar en los grupos de prueba?

En este caso, debido al largo confinamiento por el COVID-19, se tuvieron que realizar modificaciones al experimento a fin de respetar las normas sanitarias, por lo que se optó por recurrir a las herramientas virtuales. Sin embargo, se pueden mencionar las siguientes posibles mejoras para el grupo de prueba físico y virtual:

- Similar a lo realizado por Zhong *et al.* (2018), se podría modificar el código elaborado en V-REP de forma que se añadiera un comportamiento de búsqueda, y "rodeo". De forma que los depredadores y depredadores tope puedan realizar de forma mas eficiente su labor de búsqueda de alimento.
- Continuando con la propuesta anterior, también seria recomendable realizar modificaciones al comportamiento de las presas, en este caso se propone añadir un comportamiento de evasión e incremento de velocidad, de forma que puedan oponer resistencia al ataque de sus depredadores. Esto de igual forma podría resultar beneficioso para los grupos de depredadores.
- Otra habilidad interesante para el grupo de las presas sería el adaptar su código, de forma que puedan emitir una alarma a sus compañeros cuando alguno de los miembros detecte la presencia cercana de algún depredador. Esto también podría formar parte de la lista de habilidades de los depredadores, en esta situación puede ser (en el caso de los depredadores especialistas/secundarios) alertar de la presencia de los depredadores tope o en caso contrario de la presencia de presas cercanas, de forma que se reagrupen a fin de mejorar su estrategia de cacería.
- Un aspecto que también se puede mejorar es que los individuos de todos los grupos eviten alejarse mucho de sus camaradas ya que tanto en las pruebas físicas como virtuales este aspecto es una constante que puede reducir las posibilidades de supervivencia de los kilobots durante las pruebas.

4.3. Observaciones y posibles mejoras del área de prueba

El principal obstáculo que se presentó durante la realización de esta serie de experimentos, fue el aspecto de como llevar a cabo la recolección de datos y la comparativa entre los modelos matemáticos y los obtenidos durante las simulaciones en el laboratorio virtual/físico.

En este caso no se pudo llevar a cabo una comparativa satisfactoria de los resultados obtenidos durante las simulaciones. Este aspecto podría mejorarse si se crea un

área de trabajo similar a lo visto en (Pelkonen, 2018). La adición de un entorno que facilite el trabajo y comunicación de los programas V-REP y MATLAB podría permitir la implementación de escenarios de prueba más complejos y una recopilación de datos mucho más completa. Un ejemplo puede ser la elaboración de un escenario en el que el grupo de presas puedan recurrir al empleo de refugios y áreas de alimento (que serían determinadas zonas del área de prueba establecidas con anterioridad), con lo que podrían aumentar su calidad de vida durante la simulación.

Otro aspecto a mejorarse en la simulación es el código que permite el registro del cambio en las poblaciones de los kilobots, pues en algunas ocasiones presenta incoherencias respecto a las cantidades presentes durante la prueba. Se intentó arreglar este aspecto durante la realización de esta prueba, sin embargo el tiempo no fue suficiente para dar con el error que provocaba este error del código.

En cuanto a la zona de prueba con el equipo físico, se recomendaría la obtención de elementos que permitan un registro similar a lo que pretendería alcanzar en la simulación. Esto podría ser posible con elementos como el Kilogrid, ver figura 59, descrito en la página *http://www.k-team.com/kilogrid* como una herramienta que permite que los kilobots interactúen en un escenario mas complejo, ya que facilita el envió y recepción de información entre los kilobots y el experimentador.

4.4. Otros obstáculos a tener en consideración en el futuro

Un detalle importante que surgió y que puede ser de importancia en el futuro, es el hecho de que el grupo de prueba tiene un límite. Ya que si se diera el caso, en que una población surgiera vencedora (caso de extinción de una de las especies), la otra indudablemente moriría pues la falta de recursos le impedirían continuar subsistiendo. Y gracias a que se trata de un ambiente controlado (y aislado), los factores de migración e inmigración no son tomados en cuenta durante las simulaciones. Si en el futuro las habilidades del grupo de las presas y los depredadores mejoran y se consigue obtener un ambiente de coexistencia, este sería un posible próximo paso a mejorar en futuros proyectos.

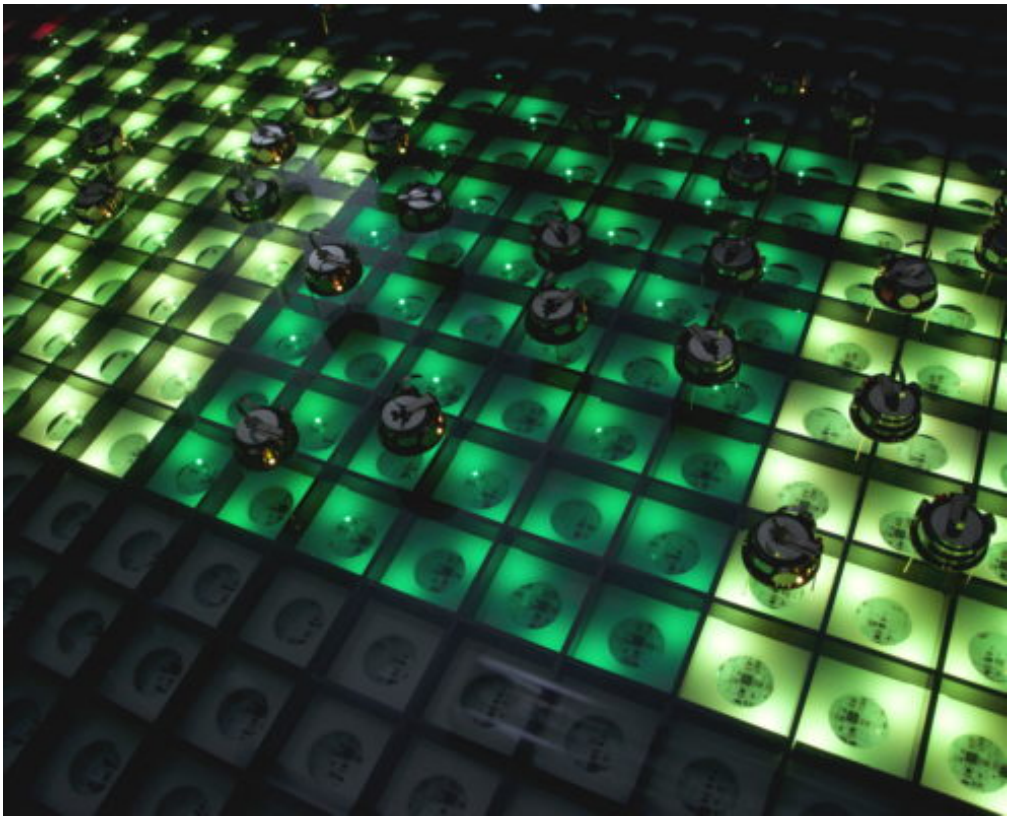


Figura 59. Imagen ejemplo del kilogrid K-TEAM (26-09-2018)

Capítulo 5. Conclusiones

5.1. Conclusiones generales

El objetivo de esta investigación era la elaboración de modelos presa-depredador con enjambres robóticos. Para ello, primeramente se analizaron distintos modelos matemáticos que fueron reportados en la literatura con el fin de replicar el comportamiento que se observa en la naturaleza. Como fue el modelo Lotka-Volterra básico, Lotka-Volterra con competencia intraespecífca y un modelo que permitía el surgimiento de caos.

Sin embargo, debido a factores ajenos a nuestra voluntad, la investigación se vio limitada al uso de los recursos virtuales a fin de poder concluir con el experimento. Por lo cual, se recurrió a programas como V-REP y MATLAB. En una primera etapa se elaboraron códigos que permitieran que los kilobots adoptaran un comportamiento similar al esperado en un escenario con una presa y un depredador. Una vez alcanzado dicho objetivo, se realizaron algunas pruebas para comprobar su funcionamiento y el alcance del mismo.

El siguiente paso fue la mejora del código anterior, de forma que se pudiera crear un escenario que permitiera el surgimiento de comportamiento caótico. Para esto, era necesario que en dicho escenario participaran 3 grupos diferentes, los cuales se establecieron como: *depredador tope*, *depredador secundario* y *presa*. De forma similar al escenario anterior, una vez elaborado el código, se llevaron a cabo algunas pruebas para comprobar su funcionamiento.

En ambos casos se llevó a cabo un conteo de los individuos muertos y vivos, lo cual se registró en unas tablas a fin de poder realizar el análisis de los resultados obtenidos. Estos en conjunto con el comportamiento observado en los kilobots a lo largo de las pruebas, nos permitió presenciar un comportamiento similar al que se puede apreciar en la naturaleza, como es el consumo de las presas por parte de los depredadores, el beneficio de permanecer cerca de otros miembros de la misma especie (comportamiento que le permitió a algunos individuos prolongar su esperanza de vida durante el periodo de las pruebas), las ventajas que ofrecen las habilidades otorgadas a los integrantes de las pruebas (que representaría las ventajas y desventajas presentes en los grupos de presas y depredadores), etc.

En la segunda etapa de la investigación se llevaron a cabo algunas pruebas en el programa de MATLAB, donde se creo un código que pudiera representar algo similar a lo que se esperaba obtener en el simulador de V-REP (los agentes que se desplazan por el espacio tipo rejilla), a fin de poder contar con algo que se pudiera comparar con los resultados obtenidos de los modelos matemáticos que se utilizaron como referencia. Los resultados de las pruebas nos indicaron que se obtuvo un comportamiento similar a lo ya establecido por investigaciones anteriores. Sin embargo, sería recomendable mejorar el escenario de forma que se pudiera realizar una mejor comparativa entre la simulación/ejercicio elaborado con los kilobots y los modelos matemáticos.

La última etapa de nuestro experimento consistió en la comprobación de la presencia de comportamiento caótico. Para ello se utilizó el código elaborado por Wolf (2013) el cual utiliza el método de determinación de exponentes de Lyapunov de una serie de tiempo. Los resultados obtenidos luego del análisis, nos indicaron que las condiciones establecidas permitían que el caos estuviera presente. Esto concuerda con lo establecido previamente en el documento que se utilizó como referencia, con ello nos referimos al trabajo elaborado por Rai (2004).

Sin embargo, tanto en el área física como virtual es necesario una mejora de las herramientas, a fin de poder obtener una mejor captura de datos que permita un análisis y comparativa más exactos. Lo elaborado hasta el momento ha permitido que se tenga una mejor perspectiva de las posibles fallas que hayan tenido lugar durante la investigación, así como también un mejor entendimiento de lo que se requiere durante la ejecución de las pruebas. Por lo que se espera que en el futuro dichos errores y omisiones no se vuelvan a repetir.

Lo elaborado hasta el momento deja la posibilidad a mejoras que, como se mencionó con anterioridad, podrían abrir la puerta a la construcción de escenarios más complejos, esto con la adición de los elementos descritos ya previamente, como son un mejor escenario de simulación y el uso de herramientas físicas que permitan una mejor comunicación y recolección de información. Además de lo anterior, se recomienda el uso de lo aconsejado en el trabajo de Francesca y Birattari (2016), ya que la correcta redacción y descripción de los avances alcanzados, facilitará la comunicación entre los investigadores lo que a su vez nos permitirá alcanzar mayores avances en los trabajos futuros.

5.2. Trabajo futuro

En futuros trabajos se recomienda tomar en consideración las observaciones hechas previamente, sin embargo un punto de partida recomendable puede ser el mejorar la zona de pruebas, de esa forma el mejorar el código de los grupos participantes podría resultar más sencillo. De igual forma se recomienda tomar en consideración las observaciones hechas previamente respecto a la gráfica de registro de la población del simulador V-REP, ya que con ello se podrá obtener un registro de los cambios en la población de los grupos participantes más confiable.

Literatura citada

Abrams, P. A. (1994). The fallacies ofratio-dependent"predation. *Ecology*, **75**(6): 1842–1850.

Allee, W. C., Park, O., Emerson, A. E., Park, T., y Schmidt, K. P. (1949). *Principles of animal ecology*. WB Saunders Company, Philadelphia and London. Recuperado el 08 de agosto 2021 de: https://www.biodiversitylibrary.org/bibliography/7325.

Allen, J., Schaffer, W. M., y Rosko, D. (1993). Chaos reduces species extinction by amplifying local population noise. *Nature*, **364**(6434): 229–232.

Audesirk, T., Audesirk, G., y Byers, B. E. (2008). Biología: la vida en la tierra. En: *Biología: la vida en la tierra*. pp. 928–928.

Boccara, N. (2010). *Modeling complex systems*. Springer Science & Business Media.

Brambilla, M., Ferrante, E., Birattari, M., y Dorigo, M. (2013). Swarm robotics: A review from the swarm engineering perspective. *Swarm Intelligence*, **7**: 1–41.

Cabezas Venegas, N. A. (2017). Población. factores que intervienen en la dinámica de población. variaciones de una población (competencia, predación, simbiosis). Recuperado el 29 de abril 2021 de: http://repositorio.une.edu.pe/handle/UNE/2963.

Chen, J. (2019). *The predator-prey evolutionary robots system: from simulation to real world*. Tesis de doctorado, Department of Computer Science, Faculty of Informatics, Vrije Universiteit Amsterdam. Recuperado el 08 de agosto 2021 de: https://staff.fnwi.uva.nl/a.visser/education/masterProjects/master_thesis_jiunhan_chen.pdf.

Di Bitetti, M. S. (2008). Depredadores tope y cascadas tróficas en ambientes terrestres. *Ciencia hoy*, pp. 32–41.

Dorigo, M., Birattari, M., y Brambilla, M. (2014). Swarm robotics. *Scholarpedia*, **9**(1): 1463. revision #138643.

Elfwing, S., Uchibe, E., Doya, K., y Christensen, H. I. (2011). Darwinian embodied evolution of the learning ability for survival. *Adaptive Behavior*, **19**(2): 101–120.

Ellner, S. y Turchin, P. (1995). Chaos in a noisy world: new methods and evidence from time-series analysis. *The American Naturalist*, **145**(3): 343–375.

Ferreras, P. (2015). Relaciones ecológicas de los depredadores. efectos del control de sus poblaciones. Recuperado el 29 de abril 2021 de: https://digital.csic.es/bitstream/10261/145759/1/ecolgicadepredador.pdf.

Francesca, G. y Birattari, M. (2016). Automatic design of robot swarms: Achievements and challenges. *Frontiers in Robotics and AI*, **3**: 29.

Gallego Valencia, J. P. (2010). *Aplicación de la teoría de caos para el análisis y pronóstico de series de tiempo financieras en Colombia*. Maestría en ingeniería administrativa, Universidad Nacional de Colombia.

Gilpin, M. E. (1979). Spiral chaos in a predator-prey model. *The American Naturalist*, **113**(2): 306–308.

Goddard, T., Do, T., Riel-Mehan, M., Johnson, G., Ling, A., Helkme, K., Broeker, M., y Lim, W. (2015). Kilobots, t-cells and cancer. Recuperado el 13 de agosto 2020 de: https://www.cgl.ucsf.edu/chimera/data/kilobots-jan2015/cancerbots.html.

Gómez, J. y Vélez, C. (2009). Modelo presa-depredador y su contextualización en el ámbito nacional e internacional. Recuperado el 29 de abril 2021 de: www.eafit.edu.co/programas-academicos/pregrados/ingenieria-matematica/practicas-investigativas/Documents/modelo-presa-depredador.pdf.

Gukenheimer, J. y Holmes, P. (1983). Nonlinear oscillations. *Dynamical Systems, and Bifurcation of Vector Fields, Springer-Verlag, NY*.

Hassle, M. (1978). The dynamics of arthropod predator-prey systems. En: *Monographs in Population Biology 13*. Princeton University Press New Jersey, pp. 1–237.

Hastings, A. y Powell, T. (1991). Chaos in a three-species food chain. *Ecology*, **72**(3): 896–903.

Hastings, A., Hom, C. L., Ellner, S., Turchin, P., y Godfray, H. C. J. (1993). Chaos in ecology: is mother nature a strange attractor? *Annual review of ecology and systematics*, **24**(1): 1–33.

K-TEAM (10-11-2011). Kilobot. https://www.k-team.com/mobile-robotics-products/kilobot.

K-TEAM (26-09-2018). Kilogrid. http://www.k-team.com/kilogrid.

Lande, R. (1988). Genetics and demography in biological conservation. *Science*, **241**(4872): 1455–1460.

Lee, S.-H. (2006). Predator's attack-induced phase-like transition in prey flock. *Physics Letters A*, **357**(4-5): 270–274.

Leslie, P. y Gower, J. (1960). The properties of a stochastic model for the predator-prey type of interaction between two species. *Biometrika*, **47**(3/4): 219–234.

Leslie, P. H. (1948). Some further notes on the use of matrices in population mathematics. *Biometrika*, **35**(3/4): 213–245.

López Bravo, M., Granda Velepucha, S., y González Carrasco, V. (2015). Principios de la ecología general. http://repositorio.utmachala.edu.ec/handle/48000/6851, Accedido: 18.05.2021.

MATLAB (Septiembre 2017). www.mathworks.com/products/matlab.html. R 2017b (9.3.0.713579), 64-bit (win64).

May, R. (1987). Chaos in natural populations. *Proc. Roy. Soc., London Series A*, **27**: 419–428.

May, R. M. (1974). Biological populations with nonoverlapping generations: stable points, stable cycles, and chaos. *Science*, **186**(4164): 645–647.

May, R. M. (1976). Simple mathematical models with very complicated dynamics. *Nature*, **261**(5560): 459–467.

May, R. M. *et al.* (1973). *Stability and complexity in model ecosystems*. NJ, Princeton University Press.

May, R. M. y Oster, G. F. (1976). Bifurcations and dynamic complexity in simple ecological models. *The American Naturalist*, **110**(974): 573–599.

May-Cen, I. d. J. (2016). Modelos de dinámica poblacional en ecología. *Revista del Centro de Graduados e Investigación Instituto Tecnológico de Mérida*, **32**(60): 50–5.

Nishimura, S. I. (2002). A predator's selection of an individual prey from a group. *Biosystems*, **65**(1): 25–35.

Nolfi, S. (2012). Co-evolving predator and prey robots. *Adaptive Behavior*, **20**(1): 10–15.

Nolfi, S., Floreano, D., y Floreano, D. D. (2000). *Evolutionary robotics: The biology, intelligence, and technology of self-organizing machines*. MIT press.

Oganician, A., Antonio, J., *et al.* (2017). Modelo depredador-presa de volterra-lotka. pp. 1–28. Universidad de La Laguna, Sección de Biología, Facultad de ciencias.

Paine, R. T. (1966). Food web complexity and species diversity. *The American Naturalist*, **100**(910): 65–75.

Paine, R. T. (1988). Road maps of interactions or grist for theoretical development? *Ecology*, **69**(6): 1648–1654.

Paleologos, M. F., Iermanó, M. J., Blandi, M. L., y Sarandón, S. J. (2017). Las relaciones ecológicas: un aspecto central en el rediseño de agroecosistemas sustentables, a partir de la agroecología. *Redes (St. Cruz Sul, Online)*, **22**(2): 92–115.

Pelkonen, J. (2018). *Exploration of Spatiotemporal Systems with Swarm of Kilobots in V-REP Simulation.*. Master's thesis, Aalto University. School of Electrical Engineering.

Pielou, E. C. (1977). *Mathematical ecology*. Número 574.50151. Wiley, New York.

Pimm, S. (1982). Food webs chapman and hall. *London. 219p*.

Rai, V. (2004). Chaos in natural populations: edge or wedge? *Ecological Complexity*, **1**(2): 127–138.

Rai, V. y Sreenivasan, R. (1993). Period-doubling bifurcations leading to chaos in a model food chain. *Ecological modelling*, **69**(1-2): 63–77.

Rinaldi, S., Muratori, S., y Kuznetsov, Y. (1993). Multiple attractors, catastrophes and chaos in seasonally perturbed predator-prey communities. *Bulletin of mathematical Biology*, **55**(1): 15–35.

Robotics, C. (Jun 25 2019). V-rep pro edu. www.coppeliarobotics.com. Version 3.6.2.(rev. 0) 64bit (serialization version 22) (Qt Version 5.12.1, MSVC2017), Copyright (c) 2006-2019 Coppelia Robotics GmbH.

Rohmer, E., Singh, S. P. N., y Freese, M. (2013). Coppeliasim (formerly v-rep): a versatile and scalable robot simulation framework. En: *Proc. of The International Conference on Intelligent Robots and Systems (IROS)*. www.coppeliarobotics.com.

Rosenzweig, M. L. y MacArthur, R. H. (1963). Graphical representation and stability conditions of predator-prey interactions. *The American Naturalist*, **97**(895): 209–223.

Schaffer, W. M. (1985). Order and chaos in ecological systems. *Ecology*, **66**(1): 93–106.

Schaffer, W. M. y Kot, M. (1985). Do strange attractors govern ecological systems? *BioScience*, **35**(6): 342–350.

Segel, L. A. (1984). *Modeling dynamic phenomena in molecular and cellular biology*. Cambridge University Press.

Smith, T. G., Siniff, D. B., Reichle, R., y Stone, S. (1981). Coordinated behavior of killer whales, orcinus orca, hunting a crabeater seal, lobodon carcinophagus. *Canadian Journal of Zoology*, **59**(6): 1185–1189.

Tharin, J. (2010). *Kilobot User Manual*. K-Team S.A., Z.I des Plans-Praz, CH-1337 Vallorbe, Switzerland. **http://ftp.k-team.com/kilobot/user_manual/Kilobot_UserManual.pdf**

Torres, N. (2005). Caos en sistemas biológicos. *Matematicalia: Revista Digital de Divulgación Matemática de la Real Sociedad Matemática Española*, **1**(3): 2.

Upadhyay, R. K., Iyengar, S., y Rai, V. (1998). Chaos: an ecological reality? *International Journal of Bifurcation and Chaos*, **8**(06): 1325–1333.

Wischmann, S., Stamm, K., y Wörgötter, F. (2007). Embodied evolution and learning: The neglected timing of maturation. En: *European Conference on Artificial Life*. Springer, pp. 284–293.

Wolf, A. (2013). Lyapunews – instructions for using the algorithm presented in physica 16d.

Wolf, A., Swift, J. B., Swinney, H. L., y Vastano, J. A. (1985). Determining lyapunov exponents from a time series. *Physica D: Nonlinear Phenomena*, **16**(3): 285–317.

Yodzis, P. (1994). Predator-prey theory and management of multispecies fisheries. *Ecological applications*, **4**(1): 51–58.

Zhong, V. J., Umamaheshwarappa, R. R., Dornberger, R., y Hanne, T. (2018). Comparison of a real kilobot robot implementation with its computer simulation focussing on target-searching algorithms. En: *2018 International Conference on Intelligent Autonomous Systems (ICoIAS)*. IEEE, pp. 160–164.

Anexo

Simulador: V-REP

Código para 1 presa y 1 depredador

```
----------------------------------------------------------------------
prey = 1
predator = 2
state = 0 -->  vivo=0 muerto=2
behavior = prey
reviveprey = 0 --> Contador mensajes para revivir como presa
revivepredator = 0 -->idem como depredador
revivezone = 50 -->condición para revivir
r_close = 50 -->Umbral detección robot cercano
hp_max =1000
hp = hp_max --> Puntos de vida del depredador
rebirth=1 --Flag para contar el primer ciclo después de morir (depredador)
on_off=0
reset=0
lock= LOL
limite=0
------------------------------------------------------------
--Grafica DATA
firstrun=1
x=0
y=0
z=0
sim.setIntegerSignal("preyCount",0)
sim.setIntegerSignal("predatorCount",0)
sim.setIntegerSignal("deadCount",0)
------------------------------------------------------------
-- Functions similar to C API
------------------------------------------------------------
function user_prgm()
--//////////////////////////////////////////////////////////
--//user program code goes below.
--//this code needs to exit in a resonable amount of time
--//so the special message controller can also run
--//////////////////////////////////////////////////////////
--message_rx[1] -> Es presa o depredador
--message_rx[2] -> TBD
--message_rx[3] -> vivo o muerto
----------------------------------------------------------------------
---Grafica data:
if (firstrun==1) then
    if (behavior==prey) then
```

```
         x = sim.getIntegerSignal("preyCount") --obtiene el valor de la variable global
        sim.setIntegerSignal("preyCount",x+1) --sobreescribe la variable global
    else
         y = sim.getIntegerSignal("predatorCount")
        sim.setIntegerSignal("predatorCount",y+1)
    end
    firstrun=0
end
------------------------------------------------------------------
get_message()
if (behavior == prey)then --01 *ES UNA PRESA*
   message_out(prey,on_off,state)
  if(message_rx[2]==0)then--02.5 El cercano esta inmovilizado?
     if(message_rx[6]==1) then --02 Nuevo mensaje
          if(message_rx[4]<r_close) then --03 Robot cerca
             if(message_rx[1]==prey) then --04 Cercano es presa
                if(state==2) then --05 Robot:"Que hacer si estoy muerto?"
                   --El individuo revive como una presa:
                   reviveprey=reviveprey+1
                      if(reviveprey>=revivezone)then--06 Condiciones para revivir como presa:
                         behavior=prey
                         state=0
                         hp=hp_max
                         rebirth=1
                         ------------------------------------------
                         x = sim.getIntegerSignal("preyCount")
                         sim.setIntegerSignal("preyCount",x+1)
                         z = sim.getIntegerSignal("deadCount")
                         sim.setIntegerSignal("deadCount",z-1)
                         ------------------------------------------
                      end--06 Condiciones para revivir como presa.
                else --05 Robot: "Que hacer si estoy vivo?"
                --**Codigo para caso de que el individuo este vivo**
                end--05 Que debe hacer el robot si esta vivo o muerto
             ----------------------------------
             else --04 cercano es depredador
             ----------------------------------
                if(state==2) then --07 Robot: "Que hago si estoy muerto?"
                   ------------------------------------------------
                   --REVIVIR COMO Depredador
                   revivepredator=revivepredator+1
                      if(revivepredator>=revivezone)then--08 Condiciones para revivir como depredador:
                         behavior=predator
                         hp=hp_max
                         state=0
                         rebirth=1
                         ------------------------------------------------------------------
                         y = sim.getIntegerSignal("predatorCount")
```

```
                    sim.setIntegerSignal("predatorCount",y+1)
                    z = sim.getIntegerSignal("deadCount")
                    sim.setIntegerSignal("deadCount",z-1)
                    ------------------------------------------------------------------
                 end--08 Condiciones para revivir como depredador
            else --07 Robot: "Que hago si estoy vivo?"
               state=2
               reviveprey=0
               revivepredator=0
               ------------------------------------------------------------------------
               z = sim.getIntegerSignal("deadCount")
               sim.setIntegerSignal("deadCount",z+1)
               x = sim.getIntegerSignal("preyCount")
               sim.setIntegerSignal("preyCount",x-1)
               ------------------------------------------------------------------------
            end--07 Robot: Que hago si estoy vivo/muerto?
         end--04 Cercano es prey/predator
      else --03 robot Lejos
         ------QUE HACER SI ESTA LEJOS?
      end--03 robot lejos/cerca
  else--02 NO HAY MENSAJE

  end--02 Mensaje
  end--02.5 Cercano inmovilizado
 ------------------------------------------------------------
 if(hp>0)then --16 Si sus puntos de vida son arriba de cero:
   hp=hp-1
else --16 Si sus puntos de vida son menores a cero:
   state=2
     if(rebirth==1)then--17 Condiciones para puntos de vida bajos:
       reviveprey=0
       revivepredator=0
       rebirth=0
       ----------------------------------------
       z = sim.getIntegerSignal("deadCount")
       sim.setIntegerSignal("deadCount",z+1)
       y = sim.getIntegerSignal("preyCount")
       sim.setIntegerSignal("preyCount",y-1)
       ----------------------------------------
     end--17 Condiciones para puntos de vida bajos
end--16 Si sus puntos de vida son menores a cero
-------------------------------------------------
             -->COMPORTAMIENTO PRESA VIVA O MUERTA:
 if (state==2) then --09 PRESA MUERTA
     set_motor(0,0)
     set_color(3,0,0)
     on_off=2
 else --09 PRESA VIVA
```

```
        set_motor(math.random(0,200),200-math.random(0,200)) --caminata aleatoria
        set_color(0,3,0)
        on_off=0
    end--09 PRESA VIVA/MUERTA
 ----------------------------------------------------------------
else--01*ES UN DEPREDADOR!!*
----------------------------------------------------------------
   message_out(predator,on_off,state)
   if (message_rx[6]==1) then --10 Nuevo mensaje
    if(message_rx[2]==0)then--10.5 El cercano esta inmovilizado?
      if(message_rx[4]<r_close) then --11 Robot cerca
         if(message_rx[1]==prey) then --12 Cercano es una presa
            if(state==2) then --13 Robot:"Que hacer si estoy muerto?"
               reviveprey=reviveprey+1
                 if(reviveprey>=revivezone)then--14 Condiciones para revivir como presa:
                    behavior=prey
                    state=0
                    hp=hp_max
                    rebirth=1
                    ------------------------------------------------------------
                    x = sim.getIntegerSignal("preyCount")
                    sim.setIntegerSignal("preyCount",x+1)
                    z = sim.getIntegerSignal("deadCount")
                    sim.setIntegerSignal("deadCount",z-1)
                    ------------------------------------------------------------
                 end--14 Condiciones para revivir como presa
            else --13 Robot: "Que hacer si estoy vivo?"
               --SE ALIMENTA DE PRESA
               hp=hp+10
            end--13 Robot:Que hacer si estoy vivo/muerto
         else --12 Cercano es un depredador
            if(state==2) then --15 Robot: "Que hacer si estoy muerto?"
               revivepredator=revivepredator+1
                  if(revivepredator>=revivezone)then
                     behavior=predator
                     state=0
                     hp=hp_max
                     rebirth=1
                     ----------------------------------------
                     y = sim.getIntegerSignal("predatorCount")
                     sim.setIntegerSignal("predatorCount",y+1)
                     z = sim.getIntegerSignal("deadCount")
                     sim.setIntegerSignal("deadCount",z-1)
                     ----------------------------------------
                  end
            else--15 Robot: "Que hacer si estoy vivo?"
               --Hay conflicto:
                 hp=hp-10
```

```
            end--15 Robot: "Que hacer si estoy vivo/muerto?"
         end--12 Cercano es presa/depredador
      else --11 Robot lejos
           --CODIGO: Que hacer si hay robot lejos?
      end--11 Robot lejos/cerca
    end--10.5 Cercano inmovilizado
   end--10 Nuevo msj
   ------------------------------------------------------------
   if(hp>0)then --16 Si sus puntos de vida son arriba de cero:
      hp=hp-1
   else --16 Si sus puntos de vida son menores a cero:
      state=2
        if(rebirth==1)then--17 Condiciones para puntos de vida bajos:
          reviveprey=0
          revivepredator=0
          rebirth=0
          ----------------------------------------
          z = sim.getIntegerSignal("deadCount")
          sim.setIntegerSignal("deadCount",z+1)
          y = sim.getIntegerSignal("predatorCount")
          sim.setIntegerSignal("predatorCount",y-1)
          ----------------------------------------
        end--17 Condiciones para puntos de vida bajos
   end--16 Si sus puntos de vida son menores a cero
   ------------------------------------------------------------
   --   Comportamiento predador vivo/muerto
   if (state==2) then --18 Depredador muerto
      set_motor(0,0)
      set_color(3,3,3) --blanco?
      on_off=2
   else --18 Depredador vivo
      set_motor(math.random(0,200),200-math.random(0,200)) --caminata aleatoria
      set_color(3,0,3)
      on_off=0
   end--18 Depredador vivo/muerto
end--01 DEPREDADOR/PRESA
--////////////////////////////////////////////////////////////////////////////
--//END OF USER CODE
--////////////////////////////////////////////////////////////////////////////
end
```

Código para 2 depredadores y 1 presa

```
prey=1
predator=2
TOPredator=3
behavior=prey
------------------------------------------------------------
```

```
state=0--> vivo=0 muerto=2
--------------------------
reviveprey=0
revivepredator=0
reviveTOPredator=0
revivezone=50
r_close=50
hp_max=1000
hp=hp_max
rebirth=1
on_off=0
reset=0
lock=LOL
limite=0
--------------------------
--------------------------
--Grafica DATA
firstrun=1
x=0
y=0
z=0
sim.setIntegerSignal("preyCount",0)
sim.setIntegerSignal("predatorCount",0)
sim.setIntegerSignal("deadCount",0)
------------------------------------------------------------
-- Functions similar to C API
------------------------------------------------------------
function user_prgm()
--//////////////////////////////////////////////////////////
--//user program code goes below.
--//this code needs to exit in a resonable amount of time
--//so the special message controller can also run
--//////////////////////////////////////////////////////////
--message_rx[1] -> Es presa o depredador
--message_rx[2] -> TBD
--message_rx[3] -> vivo o muerto
-------------------------------------------------------------------
---Grafica data:
if (firstrun==1) then
    if (behavior==prey) then
         ------------------------------------------------------------------------------
         x = sim.getIntegerSignal("preyCount") --obtiene el valor de la variable global
             sim.setIntegerSignal("preyCount",x+1) --sobreescribe la variable global
         ------------------------------------------------------------------------------
    else
         ------------------------------------------------------------------------------
         y = sim.getIntegerSignal("predatorCount")
             sim.setIntegerSignal("predatorCount",y+1)
         ------------------------------------------------------------------------------
    end
    firstrun=0
end
-------------------------------------------------------------------
-------------------------------------------------------------------
get_message()
if (behavior == prey)then --000 *ES UNA PRESA*
message_out(prey,on_off,state)
-----------------------------------------------
   if(message_rx[2]==0)then--000 CERCANO INMOVIL
   ------------------------------------------
     if(message_rx[6]==1)then--000 NUEVO MENSAJE
     ---------------------------------------
       if(message_rx[4]<r_close)then--000 ROBOT CERCA
       -------------------------------------------
```

```
if(message_rx[1]==prey)then--000 CERCANO ES PRESA
-------------------------------------------------------------
  if(state==2)then--000 QUE HACER CUANDO PRESA ESTA MUERTA?
  -----------------------------------------------------------
    --La presa revive como presa.
      reviveprey=reviveprey+1
      if(reviveprey>=revivezone)then--000 CONDICION REVIVIR PRESA
        behavior=prey
        state=0
        hp=hp_max
        rebirth=1
      -----------------------------------------------------------
      x=sim.getIntegerSignal("preyCount")
        sim.setIntegerSignal("preyCount",x+1)
      y=sim.getIntegerSignal("deadCount")
        sim.setIntegerSignal("deadCount",z-1)
      -----------------------------------------------------------
      end--000 CONDICION PARA REVIVIR COMO PRESA
  -----------------------------------------------------------
  else--000 QUE HACER CUANDO PRESA ESTA VIVA?
  -----------------------------------------------------------
    --NO OCURRE NADA.
  -----------------------------------------------------------
  end--000 ?QUE HACER CUANDO PRESA VIVA/MUERTA?
-------------------------------------------------------------
elseif(message_rx[1]==predator)then--000 CERCANO ES DEPREDADOR
-------------------------------------------------------------
  if(state==2)then--000 QUE HACE PRESA SI ESTA MUERTA?
  -----------------------------------------------------------
    --REVIVIR COMO DEPREDADOR
      revivepredator=revivepredator+1
      if(revivepredator>=revivezone)then--000 CONDICION REVIVIR DEPREDADOR.
         behavior=predator
         hp=hp_max
         state=0
         rebirth=1
         ---------------------------------------------------------------------
           y=sim.getIntegerSignal("predatorCount")
             sim.setIntegerSignal("predatorCount",y+1)
           z=sim.getIntegerSignal("deadCount")
             sim.setIntegerSignal("deadCount",z-1)
         ---------------------------------------------------------------------
      end--000 CONDICION REVIVIR DEPREDADOR
  -----------------------------------------------------------
  else--000 QUE HACE PRESA SI ESTA VIVA?
  -----------------------------------------------------------
    state=2
    reviveprey=0
    revivepredator=0
    reviveTOPredator=0
    -----------------------------------------------------------
      y=sim.getIntegerSignal("deadCount")
        sim.setIntegerSignal("deadCount",z+1)
      x=sim.getIntegerSignal("preyCount")
        sim.setIntegerSignal("preyCount",x-1)
    -----------------------------------------------------------
  -----------------------------------------------------------
  end--000 QUE HACE PRESA SI ESTA MUERTA/VIVA?
-------------------------------------------------------------
else--000 CERCANO ES TOPredator.
-------------------------------------------------------------
      if(state==2)then--000 QUE HACE PRESA SI ESTA MUERTA?
      ---------------------------------------------------
      --REVIVIR COMO DEPREDADOR.
```

```
                reviveTOPredator=reviveTOPredator+1
                if(reviveTOPredator>=revivezone)then--000 CONDICION TOP
                -------------------------------------------------------
                  behavior=TOPredator
                  hp=hp_max
                  state=0
                  rebirth=1
                  -----------------------------------------------------
                    y=sim.getIntegerSignal("predatorCount")
                      sim.setIntegerSignal("predatorCount",y+1)
                    z=sim.getIntegerSignal("deadCount")
                      sim.setIntegerSignal("deadCount",z-1)
                  -----------------------------------------------------
                -------------------------------------------------------
                end-- CONDICION TOP
              ---------------------------------------------------
              else--000 QUE HACE PRESA SI ESTA VIVA?
              ---------------------------------------------------
                state=2
                reviveprey=0
                revivepredator=0
                reviveTOPredator=0
                -------------------------------------------------
                  z=sim.getIntegerSignal("deadCount")
                    sim.setIntegerSignal("deadCount",z+1)
                  x=sim.getIntegerSignal("preyCount")
                    sim.setIntegerSignal("preyCount",x-1)
                -------------------------------------------------
              ---------------------------------------------------
              end--000 QUE HACE PRESA SI ESTA VIVA/MUERTA?
          -----------------------------------------------------------
          end--000 ?EL CERCANO ES...?
        ---------------------------------------------
        else--000 ROBOT LEJOS.
        ---------------------------------------------
          -- QUE HACER SI ESTA LEJOS
        ---------------------------------------------
        end--000 ROBOT LEJOS/CERCA
      -----------------------------------------
      end--000 NUEVO MENSAJE
    -----------------------------------------
    end--000 CERCANO INMOVIL
-------------------------------------------------------------------------
if(hp>0)then--000 Puntos de vida mayores a cero:
    hp=hp-1
else--000 Puntos de vida menores a cero:
    state=2
    if(rebirth==1)then--000 Condiciones para puntos de vida menores a cero.
       reviveprey=0
       revivepredator=0
       reviveTOPredator=0
       rebirth=0
    -------------------------------------------------------------------
      z=sim.getIntegerSignal("deadCount")
        sim.setIntegerSignal("deadCount",z+1)
      y=sim.getIntegerSignal("preyCount")
        sim.setIntegerSignal("preyCount",y-1)
    -------------------------------------------------------------------
    end--000 Condiciones para puntos de vida menores a cero.
end--000 Puntos de vida mayores/menores a cero:
-------------------------------------------------------------------
    -- COMPORTAMIENTO PRESA VIVA/MUERTA:
       if(state==2)then--000 PRESA MUERTA.
       ----------------------------------
```

```
        set_motor(0,0)
        set_color(3,0,0)
        on_off=2
      ----------------------------------
      else--000 PRESA VIVA.
      ----------------------------------
        set_motor(math.random(0,200),200-math.random(0,200))
        set_color(0,3,0)
        on_off=0
      ----------------------------------
      end--000 PRESA VIVA/MUERTA
-----------------------------------------------
elseif(behavior==predator)then--010 *ES UN DEPREDADOR*
message_out(predator,on_off,state)
-----------------------------------------------
if(message_rx[6]==1)then--010 NUEVO MENSAJE
-----------------------------------------------
  if(message_rx[2]==0)then--010 CERCANO INMOVIL?
  ---------------------------------------------
    if(message_rx[4]<r_close)then--010 Robot cerca?
    -------------------------------------------
      if(message_rx[1]==prey)then--010 Cercano es una presa.
      ------------------------------------------------------------------
        if(state==2)then--010 Que hace predador si esta muerto.
        ----------------------------------------------------------------
          reviveprey=reviveprey+1
          if(reviveprey>=revivezone)then--010 Condiciones para revivir como presa.
          ------------------------------------------------------------------------
            behavior=prey
            state=0
            hp=hp_max
            rebirth=1
            ----------------------------------------------------------------------
              x=sim.getIntegerSignal("preyCount")
                sim.setIntegerSignal("preyCount",x+1)
              y=sim.getIntegerSignal("deadCount")
                sim.setIntegerSignal("deadCount",z-1)
            ----------------------------------------------------------------------
          ------------------------------------------------------------------------
          end--010 Condiciones para revivir como presa.
        ----------------------------------------------------------------
        else--010 Que hace predador si esta vivo.
        ----------------------------------------------------------------
          --Se alimenta de presa.
            hp=hp+10
        ----------------------------------------------------------------
        end--010 Que hace si depredador esta vivo/muerto.
      ------------------------------------------------------------------
      elseif(message_rx[1]==predator)then--010 Cercano es un depredador.
      ------------------------------------------------------------------
        if(state==2)then--010 Que hacer si depredador esta muerto?
        ----------------------------------------------------------------
          revivepredator=revivepredator+1
          if(revivepredator>=revivezone)then--010 REVIVE
          ----------------------------------------------------------------
            behavior=predator
            state=0
            hp=hp_max
            rebirth=1
            ------------------------------------------------------------
              y=sim.getIntegerSignal("predatorCount")
                sim.setIntegerSignal("predatorCount",y+1)
              z=sim.getIntegerSignal("deadCount")
                sim.setIntegerSignal("deadCount",z-1)
```

```
            ------------------------------------------------------------
          ------------------------------------------------------------
            end--REVIVE
          ------------------------------------------------------------
          else--010 Que hacer si depredador esta vivo?
          ------------------------------------------------------------
          -- HAY CONFLICTO:
             hp=hp-10
          ------------------------------------------------------------
          end--010 Que hacer si depredador esta muerto/vivo?
        ------------------------------------------------------------------
        else--010 Cercano es un TOP
        ------------------------------------------------------------------
          if(state==2)then--Que hacer si depredador esta muerto?
          ------------------------------------------------------------
            reviveTOPredator=reviveTOPredator+1
            if(reviveTOPredator>=revivezone)then--010 REVIVE
               behavior=TOPredator
               state=0
               hp=hp_max
               rebirth=1
               ------------------------------------------------------
                 y=sim.getIntegerSignal("predatoCount")
                   sim.setIntegerSignal("predatorCount",y+1)
                 z=sim.getIntegerSignal("deadCount")
                   sim.setIntegerSignal("deadCount",z-1)
               ------------------------------------------------------
            end--REVIVE
          ------------------------------------------------------------
          else--Que hacer si dep esta vivo?
          ------------------------------------------------------------
            state=2
            reviveprey=0
            revivepredator=0
            reviveTOPredator=0
            ------------------------------------------------------
              z=sim.getIntegerSignal("deadCount")
                sim.setIntegerSignal("deadCount",z+1)
              y=sim.getIntegerSignal("predatorCount")
                sim.setIntegerSignal("predatorCount",y+1)
            ------------------------------------------------------
          ------------------------------------------------------------
          end--Que hacer si dep esta muerto/vivo?
        ------------------------------------------------------------------
        end--010 Cercano a predator es...?
      ------------------------------------------------
      else--010 Robot lejos....?
      ------------------------------------------------
        --CODIGO PARA CASO ROBOT LEJOS.
      ------------------------------------------------
      end--010 Robot cerca/lejos?
    ------------------------------------------------
    end--010 CERCANO INMOVIL?
  ------------------------------------------------
  end--010 NUEVO MENSAJE
  ------------------------------------------------
       if(hp>0)then-- 010 PUNTOS MAYORES A CERO.
       ----------------------------------------
         hp=hp-1
       ----------------------------------------
       else--010 PUNTOS MENORES A CERO.
       ----------------------------------------
         state=2
         if(rebirth==1)then--010 Condiciones puntos bajos.
```

```
       ------------------------------------------------
         reviveprey=0
         revivepredator=0
         reviveTOPredator=0
         rebirth=0
         ------------------------------------------------
         z=sim.getIntegerSignal("deadCount")
           sim.setIntegerSignal("deadCount",z+1)
         y=sim.getIntegerSignal("predatorCount")
           sim.setIntegerSignal("predatorCount",y-1)
         ------------------------------------------------
       ------------------------------------------------
       end--010 Condiciones puntos bajos.
     ----------------------------------------
     end--010 PUNTOS MAYORES/MENORES A CERO.
-------------------------------------------------------
   -- COMPORTAMIENTO PREDADOR VIVO/MUERTO
      if(state==2)then--010 depredador muerto.
      ------------------------------------------------
        set_motor(0,0)
        set_color(3,3,3)
        on_off=2
      ------------------------------------------------
      else--010 depredador vivo.
      ------------------------------------------------
        set_motor(math.random(0,200),200-math.random(0,200))
        set_color(3,0,3)
        on_off=0
      ------------------------------------------------
      end--010 depredador vivo/muerto
-----------------------------------------------
else--(behavior==TOPredator)--100 *ES UN TOP*
message_out(TOPredator,on_off,state)
-----------------------------------------------
if(message_rx[6]==1)then--100 NUEVO MENSAJE.
-----------------------------------------------
  if(message_rx[2]==0)then--100 CERCANO INMOVIL
  ---------------------------------------------
    if(message_rx[4]<r_close)then--100 ROBOT CERCA.
    -------------------------------------------
      if(message_rx[1]==prey)then--100 CERCANO PRESA.
      ---------------------------------------------------------
        if(state==2)then--100 TOP MUERTO.
        ---------------------------------------------------
          reviveprey=reviveprey+1
          if(reviveprey>=revivezone)then--100 Revivir TOP--->PRESA
             behavior=prey
             state=0
             hp=hp_max
             rebirth=1
             ------------------------------------------------
               x=sim.getIntegerSignal("preyCount")
                 sim.setIntegerSignal("preyCount",x+1)
               z=sim.getIntegerSignal("deadCount")
                 sim.setIntegerSignal("deadCount",z-1)
             ------------------------------------------------
          end--100 Revivir TOP
        ---------------------------------------------------
        else--100 TOP VIVO.
        ---------------------------------------------------
          --SE ALIMENTA DE PRESA.
          hp=hp+10
        ---------------------------------------------------
        end--100 TOP MUERTO/VIVO
```

```
      ------------------------------------------------------
      elseif(message_rx[1]==predator)then--100 CERCANO PREDADOR.
      ------------------------------------------------------
        if(state==2)then--100 Si top esta muerto...
        ------------------------------------------------------
          revivepredator=revivepredator+1
          if(revivepredator>=revivezone)then--100 reviveTOP
          ------------------------------------------------------
            behavior=predator
            state=0
            hp=hp_max
            rebirth=1
          ------------------------------------------------------
            y=sim.getIntegerSignal("predatorCount")
              sim.setIntegerSignal("predatorCount",y+1)
            z=sim.getIntegerSignal("deadCount")
              sim.setIntegerSignal("deadCount",z-1)
          ------------------------------------------------------
          ------------------------------------------------------
          end--100 reviveTOP
        ------------------------------------------------------
        else--100 Si top esta vivo...
        ------------------------------------------------------
          --Se alimenta de depredador.
            hp=hp+10
        ------------------------------------------------------
        end--100 TOP esta muerto/vivo
      ------------------------------------------------------
      else--100 CERCANO TOP.
      ------------------------------------------------------
        if(state==2)then--100 Si esta muerto...
        ------------------------------------------------------
          reviveTOPredator=reviveTOPredator+1
          if(reviveTOPredator>=revivezone)then--100 reviveTOP
          ------------------------------------------------------
            behavior=TOPredator
            state=0
            hp=hp_max
            rebirth=1
          ------------------------------------------------------
            y=sim.getIntegerSignal("predatoCount")
              sim.setIntegerSignal("predatorCount",y+1)
            z=sim.getIntegerSignal("deadCount")
              sim.setIntegerSignal("deadCount",z-1)
          ------------------------------------------------------
          ------------------------------------------------------
          end--100 reviveTOP
        ------------------------------------------------------
        else--100 Si esta vivo...
        ------------------------------------------------------
          --hay conflicto
            hp=hp-10
        ------------------------------------------------------
        end--100 Si esta muerto/vivo...?
      ------------------------------------------------------
      end--100 CERCANO ES...?
    ------------------------------------------------
    end--100 ROBOT CERCA.
  ------------------------------------------------
  end--100 CERCANO INMOVIL
------------------------------------------------
end--100 NUEVO MENSAJE
------------------------------------------------
if(hp>0)then--100 Puntos de vida mayores a cero.
```

```
-----------------------------------------------
  hp=hp-1
-----------------------------------------------
else--100 Puntos de vida menores a cero.
-----------------------------------------------
  state=2
  if(rebirth==1)then--100 Condiciones puntos de vida bajos.
  ------------------------------------------------------------
    reviveprey=0
    revivepredator=0
    reviveTOPredator=0
    rebirth=0
    ------------------------------------------------------------
      z=sim.getIntegerSignal("deadCount")
        sim.setIntegerSignal("deadCount",z+1)
      y=sim.getIntegerSignal("predatorCount")
        sim.setIntegerSignal("predatoCount",y-1)
    ------------------------------------------------------------
  ------------------------------------------------------------
  end--100 Condiciones puntos de vida bajos.
-----------------------------------------------
end--100 Puntos de vida mayores/menores a cero.
-----------------------------------------------
--Comportamiento TOP vivo/muerto.
  if(state==2)then--100 TOP muerto.
  ----------------------------------------------
    set_motor(0,0)
    set_color(0,0,3)
    on_off=2
  ----------------------------------------------
  else--100 TOP vivo.
  ----------------------------------------------
    set_motor(math.random(0,200),200-math.random(0,200))
    set_color(0,3,3)
    on_off=0
  ----------------------------------------------
  end--100 TOP vivo/muerto.
-----------------------------------------------
end--000 *EL KILOBOT ES...*
        ---------------------------------------
        end
```

MATLAB

Main-Lotka-Volterra

```
 %Main LotkaVolterra
clear all
close all
global a1 a2 b1 b2 k
a1 = 0.55;   %r1
a2 = 0.027;   %a1
b1 = 0.026;  %a2
b2 = 0.83;   %r2º
k = 10000;

x0 = [30;4];
t0 = linspace(0,100,1e4);

[t,x]=ode45(@LotkaVolterra,t0,x0);
```

```
figure()
plot(t,x(:,1),'b',t,x(:,2),'r')
legend('presa','depredador')
title('Lotka - Volterra básico')
xlabel('Unidades de tiempo [ut]')
ylabel('Cantidad de individuos')

figure()
plot(x(:,2),x(:,1))
title('retrato de fase')
xlabel('Depredadores')
ylabel('Presas')
```

LotkaVolterra

```
%lotka volterra

function f=LotkaVolterra(t,x)
global a1 a2 b1 b2 k

f = [a1*x(1) - a2*x(1)*x(2);
     b1*x(1)*x(2) - b2*x(2)];
```

Simulación, presa-depredador con desplazamiento

30 presas y 4 depredadores

```
%30 Prey- 4 Predator
clear all
close all
global ws
ws = 5; %World Size
iterations = 1000;
prey = 30; %Prey population
predator = 4; %Predator population
a1=0.55; %prey birth rate
b1 = 0.83;%0.83;%Predator death rate

eat = 0;
preyIndex = prey;
predatorIndex = predator;
t_prey = prey*ones(1,iterations); %prey count over time
t_predator = predator*ones(1,iterations); %predator count over time

for i=1:prey
    preyPos(:,1,i) = randi([-ws,ws],2,1); %prey XY initial position
    figure(4)
    plot(preyPos(1,1,i),preyPos(2,1,i),'o')
    hold on
end

for i=1:predator
    predatorPos(:,1,i) = randi([-ws,ws],2,1); %predator XY initial position
    plot(predatorPos(1,1,i),predatorPos(2,1,i),'x')

end

hold off
```

```
grid on
title('Posiciones iniciales')
xlabel('eje x')
ylabel('eje y')

for n=1:iterations %main loop
    for i=1:preyIndex %for each prey
        if preyPos(:,1,i)~=[nan;nan]
%             figure(1)
%             plot(preyPos(1,1,i),preyPos(2,1,i),'o')
%              axis([-ws ws -ws ws])
%             hold on
            preyPos(:,1,i)=move(preyPos(1,1,i),preyPos(2,1,i)); %new position

%             if rand(1,1)<b1 %probability of prey i death
%                 prey = prey-1;
%                 preyPos(:,1,i)= [nan;nan];
%             end
        end
    end
    %Prey birth probability
    if rand(1,1)<=a1
       prey = prey+1; %new prey
       preyIndex=preyIndex+1;
       preyPos(:,1,preyIndex) = randi([-ws,ws],2,1);
    end

    for i=1:predatorIndex %for each predator
        if predatorPos(:,1,i)~=[nan;nan] %if not dead
%             plot(predatorPos(1,1,i),predatorPos(2,1,i),'x')
%              axis([-ws ws -ws ws])
%             hold on
            predatorPos(:,1,i)=move(predatorPos(1,1,i),predatorPos(2,1,i)); %new position


            for k=1:preyIndex %prey encountered?
                d=sqrt((predatorPos(1,1,i)-preyPos(1,1,k))^2+(predatorPos(2,1,i)-preyPos(2,1,k))^2); %distance to prey
                if d==0
                    preyPos(:,1,k)= [nan;nan]; %prey is dead
                    prey = prey-1;
                    predator = predator+1; %new predator
                    predatorIndex=predatorIndex+1;
                    predatorPos(:,1,predatorIndex) = randi([-ws,ws],2,1);
                    eat = 1;
                    break;
                end
            end

              if predator > 0 && eat == 0
                if rand(1,1)<=b1 %probability of prey i death
                    predator = predator-1;
                    predatorPos(:,1,i)= [nan;nan];
                end
              end

              eat = 0;
        end
    end

%     pause(0.1);
%     hold off
    t_prey(n)=prey;
    t_predator(n)=predator;
end
```

```
figure(2)
plot(t_prey)
hold on
plot(t_predator)
legend('Presa','Depredador')
title('Lotka - Volterra con desplazamiento')
ylabel('Cantidad de individuos')
xlabel('instantes de tiempo')

figure (3)
plot(t_predator,t_prey)
title('retrato de fase')
xlabel('Depredador')
ylabel('Presa')

function f=move(x,y) %new position
global ws
pos = randi([0,8],1);
    switch pos
        case 1
            f=[x+1;y];
        case 2
            f=[x+1;y+1];
        case 3
            f=[x;y+1];
        case 4
            f=[x-1;y+1];
        case 5
            f=[x-1;y];
        case 6
            f=[x-1;y-1];
        case 7
            f=[x;y-1];
        case 8
            f=[x+1;y-1];
        otherwise
            f=[x;y];
    end

    if f(1)>ws || f(1)<-ws || f(2)>ws || f(2)<-ws %out of bound movement
        f=[x;y];
    end
end
```

Lotka-Volterra, Competencia intra-específica

```
%Main LotkaVolterra
clear all
close all
global a1 a2 b1 b2 b3
a1 = 0.55;
a2 = 0.027;
b1 = 0.026;
b2 = 0.83;
b3 = 0.009;

x0 = 100*[0.30;0.04];
t0 = linspace(0,100,1e4);

[t,x]=ode45(@LotkaVolterraComp,t0,x0);
```

```
figure()
plot(t,x(:,1),'b',t,x(:,2),'r')
legend('presa','depredador')
title('Lotka - Volterra básico')
xlabel('Unidades de tiempo [ut]')
ylabel('Cantidad de individuos')

figure()
plot(x(:,2),x(:,1))
title('retrato de fase')
xlabel('Depredadores')
ylabel('Presas')
```

Competencia intraespecífica... CONT

```
 %lotka volterra

function f=LotkaVolterraComp(t,x)
global a1 a2 b1 b2 b3

f = [a1*x(1) - a2*x(1)*x(2);
     b1*x(1)*x(2) - b2*x(2)- b3*x(2)^2];
```

Lotka-Volterra, Competencia intra-específica, con movimiento

```
%Prey-Predator Intraspecific Competition
clear all
close all
global ws
ws = 5; %World Size
iterations = 1000;
prey = 30; %Prey population
predator = 4; %Predator population
a1=0.55; %prey birth rate
b1 = 0.011;%Predator death rate
c1 = 0.000298;%0.0011; %Predator intraespecific competition rate

eat = 0;
preyIndex = prey;
predatorIndex = predator;
t_prey = prey*ones(1,iterations); %prey count over time
t_predator = predator*ones(1,iterations); %predator count over time

for i=1:prey
    preyPos(:,1,i) = randi([-ws,ws],2,1); %prey XY initial position
    figure(4)
    plot(preyPos(1,1,i),preyPos(2,1,i),'o')
    hold on
end

for i=1:predator
    predatorPos(:,1,i) = randi([-ws,ws],2,1); %predator XY initial position
    plot(predatorPos(1,1,i),predatorPos(2,1,i),'x')

end

hold off
grid on
title('Posiciones iniciales')
```

```
xlabel('eje x')
ylabel('eje y')

for n=1:iterations %main loop
    for i=1:preyIndex %for each prey
        if preyPos(:,1,i)~=[nan;nan]
%             figure(1)
%             plot(preyPos(1,1,i),preyPos(2,1,i),'o')
%              axis([-ws ws -ws ws])
%             hold on
            preyPos(:,1,i)=move(preyPos(1,1,i),preyPos(2,1,i)); %new position

%             if rand(1,1)<b1 %probability of prey i death
%                 prey = prey-1;
%                 preyPos(:,1,i)= [nan;nan];
%             end
        end
    end

    %Prey birth probability
    if rand(1,1)<=a1
       prey = prey+1; %new prey
       preyIndex=preyIndex+1;
       preyPos(:,1,preyIndex) = randi([-ws,ws],2,1);
    end

    for i=1:predatorIndex %for each predator
        if predatorPos(:,1,i)~=[nan;nan] %if not dead
%             plot(predatorPos(1,1,i),predatorPos(2,1,i),'x')
%              axis([-ws ws -ws ws])
%             hold on
            predatorPos(:,1,i)=move(predatorPos(1,1,i),predatorPos(2,1,i)); %new position

            for k=1:preyIndex %prey encountered?
                d=sqrt((predatorPos(1,1,i)-preyPos(1,1,k))^2+(predatorPos(2,1,i)-preyPos(2,1,k))^2); %distance to prey
                if d==0
                    preyPos(:,1,k)= [nan;nan]; %prey is dead
                    prey = prey-1;
                    predator = predator+1; %new predator
                    predatorIndex=predatorIndex+1;
                    predatorPos(:,1,predatorIndex) = randi([-ws,ws],2,1);
                    eat = 1;
                    break;
                end
            end

            for k=1:predatorIndex %predator encountered?

                if k~=i %avoid selfkilling
                    d=sqrt((predatorPos(1,1,i)-predatorPos(1,1,k))^2+(predatorPos(2,1,i)-predatorPos(2,1,k))^2); %distance to predator
                else
                    d=1;
                end
                if d==0
                    if rand(1,1)<=c1
                        predatorPos(:,1,k)= [nan;nan]; %predator is dead
                        predator = predator -1;
                    end

                    break;
                end
            end

              if predator > 0 && eat == 0
```

```
                if rand(1,1)<=b1 %probability of prey i death
                    predator = predator-1;
                    predatorPos(:,1,i)= [nan;nan];
                end
            end

            eat = 0;
        end
    end

%     pause(0.1);
%     hold off
    t_prey(n)=prey;
    t_predator(n)=predator;
end

figure(2)
plot(t_prey)
hold on
plot(t_predator)
legend('Presa','Depredador')
title('Lotka - Volterra con desplazamiento')
ylabel('Cantidad de individuos')
xlabel('instantes de tiempo')

figure (3)
plot(t_predator,t_prey)
title('retrato de fase')
xlabel('Depredador')
ylabel('Presa')

function f=move(x,y) %new position
global ws
pos = randi([0,8],1);
    switch pos
        case 1
            f=[x+1;y];
        case 2
            f=[x+1;y+1];
        case 3
            f=[x;y+1];
        case 4
            f=[x-1;y+1];
        case 5
            f=[x-1;y];
        case 6
            f=[x-1;y-1];
        case 7
            f=[x;y-1];
        case 8
            f=[x+1;y-1];
        otherwise
            f=[x;y];
    end

    if f(1)>ws || f(1)<-ws || f(2)>ws || f(2)<-ws %out of bound movement
        f=[x;y];
    end
end
```

Lotka Volterra, 3 especies

```
%Main 3LotkaVolterra
```

```
clear all
close all
%-------------------------------------------
global a1 b1 w D a2 w1 D1 w2 D2 c w3 D3
a1 = 2;
b1 =0.05;
w  = 1;
D  = 10;
a2 = 1; %Solo se presenta caos con a2=1
w1 = 2;
D1 = 10;
w2 = 1.45;
D2 = 10;
c  = 0.0257;
w3 = 1;
D3 = 20;

x0 = [30; 20; 8];%8
t0 = linspace(0,100,1e4);
[t,x]=ode45(@LotkaVolterraChaos,t0,x0);
presas= x(:,1);
depredadores= x(:,2);
top= x(:,3);

%% Graficas temporales:
subplot(3,2,1);  plot(x(:,1));
title('Señal Presa')
ylabel('Presa')
xlabel('Tiempo')
subplot(3,2,3); plot(x(:,2));
title('Señal D-SEC');
ylabel('D-SEC');
xlabel('Tiempo')
subplot(3,2,5); plot(x(:,3));
title('Señal D-Top')
ylabel('D-Top');
xlabel('Tiempo')
% Atractor
subplot(3,2,2); plot(x(:,1),x(:,2));
title('Atractor Presa-D-Sec')
xlabel('Presa')
ylabel('D-Sec')
subplot(3,2,4); plot(x(:,1),x(:,3));
title('Atractor Presa-D-Top')
xlabel('Presa')
ylabel('D-Top')
subplot(3,2,6);plot(x(:,2),x(:,3));
title('Atractor D-Sec- D-Top')
xlabel('D-Sec')
ylabel('D-Top')
%%
figure()
plot(t,x(:,1),'b',t,x(:,2),'r',t,x(:,3),'g')
legend('presa','depredador','depredador tope')
title('Presa - 2 depredadores')
xlabel('Unidades de tiempo [ut]')
ylabel('Cantidad de individuos')

figure()
plot3(x(:,2),x(:,1),x(:,3))
title('Espacio de fase')
xlabel('Depredadores')
ylabel('Presas')
zlabel('Depredador tope')
```

```
%% Guardar Vector de datos en forma columna para ser utilizado en estimacion
%Lyapunov.
clear VECT
VECT = fopen('prey_chaos.txt','w');
fprintf(VECT,'%i \r\n',presas);
fclose(VECT);

clear VECT
VECT =fopen('predator_chaos.txt','w');
fprintf(VECT,'%i \r\n',depredadores);
fclose(VECT);

clear VECT
VECT =fopen('top_chaos.txt','w');
fprintf(VECT,'%i \r\n',top);
fclose(VECT);
```

Lotka-Volterra, 3 especies... CONT

```
function f=LotkaVolterraChaos(t,x)

global a1 b1 w D a2 w1 D1 w2 D2 c w3 D3

f = [a1*x(1) - b1*x(1)^2 - (w*x(1)*x(2))/(x(1)+D);
     -a2*x(2) + (w1*x(1)*x(2))/(x(1)+D1) - (w2*x(2)^2*x(3))/(x(2)^2+D2^2);
     c*x(3)^2 - (w3*x(3)^2)/(x(2)+D3)];
```

Análisis de caos

testbench.m

```
%=============================================================================================================
% Lyapunov exponent estimation from a time series. Documentation added.
% version 1.2.0.0 (2.49 MB) by Alan Wolf
% A Matlab version of the Lyapunov exponent estimation algorithm of Wolf et al. -- Physica 16D, 1985.
%=============================================================================================================

clc; clear all; close all; format compact;
%
%
% Matlab version of the algorithm by Wolf et al. for estimating the dominant Lyapunov exponent from a 1-D time series.
%
% Physica 16D (1985) 285-317 "Determining Lyapunov Exponents from a Time Series"
% Alan Wolf, Jack B. Swift, Harry L. Swinney, and John A. Vastano
%
% Appendix B of the Physica D article contains Fortran code for a concise, but highly inefficient version of the algorithm.
% I have been distributing a Fortran and C version of the efficient version of the algorithm since the 1980's.
% The efficient version of the code was converted to Matlab by Taehyeun Park, The Cooper Union, EE'15 in September, 2014.
%
% Detailed instructions for the use of this code will be posted at Matlab Central's File Exchange.
%
% This file, testbench.m, takes 16,384 points of Lorenz attractor data (dt = 0.01 seconds) and estimates the dominant
% Lyapunov exponent by first calling "basgen" (preparing a database for quickly finding nearest neighbors in reconstructed
% phase space) and then calling "fet" to estimate the Lyapunov exponent.
% The program shows both graphical output as orbital divergence is monitored and a text file showing the running exponent estimate.


%% CARGAR LA SERIE DE TIEMPO
```

```
%% ESCENARIO 2 DEPREDADORES 1 PRESA:
%fname='prey_chaos.txt'; %0.2089 en caos 10000// 0.2264 en caos 1000
%fname='predator_chaos.txt';
fname='top_chaos.txt';

datcnt = 10000;
tau = 1;
ndim = 2;
ires = 20;
maxbox = 6000;

db = basgen(fname, tau, ndim, ires, datcnt, maxbox);
dt = 1;
evolve = 3;
dismin = 0.0000;
dismax = 0.01;
thmax = 30;

[out, SUM] = fet(db, dt, evolve, dismin, dismax, thmax);

makeplot(db, out, evolve, 'NorthWest')
```

basgen.m

```
function db = basgen(fname, tau, ndim, ires, datcnt, maxbox)
% Database generator for fet.m function
% Taehyeun Park, The Cooper Union, EE'15
x = fileread(fname);
data = zeros(1,datcnt);
trck = 1;
start = 1;
fin = 0;
for ii = 1:length(x)
    if strcmp(x(ii), char(32)) || strcmp(x(ii), char(13)) || strcmp(x(ii), char(10)) || strcmp(x(ii), char(26))
        if fin >= start
            data(trck) = str2num(x(start:fin));
            trck = trck + 1;
            if trck > 8*floor(datcnt/8)
                break
            end
        end
        start = ii + 1;
    else
        fin = ii;
    end
end
delay = 0:tau:(ndim-1)*tau;
nxtbox = zeros(maxbox, ndim);
where = zeros(maxbox, ndim);
datptr = zeros(1,maxbox);
nxtdat = zeros(1,datcnt);
datmin = min(data);
datmax = max(data);
datmin = datmin - 0.01*(datmax - datmin);
datmax = datmax + 0.01*(datmax - datmin);
boxlen = (datmax - datmin)/ires;
boxcnt = 1;
for ii = 1:(datcnt-(ndim-1)*tau)
    target = floor((data(ii+delay)-datmin)/boxlen);
    runner = 1;
    chaser = 0;
```

```
    jj = 1;
    while jj <= ndim
        tmp = where(runner,jj)-target(jj);
        if tmp < 0
            chaser = runner;
            runner = nxtbox(runner,jj);
            if runner ~= 0
                continue
            end
        end
        if tmp ~= 0
           boxcnt = boxcnt + 1;

           if boxcnt == maxbox
               error('Grid overflow, increase number of box count')
           end

           for kk = 1:ndim
               where(boxcnt,kk) = where(chaser,kk);
           end
           where(boxcnt,jj) = target(jj);
           nxtbox(chaser,jj) = boxcnt;
           nxtbox(boxcnt,jj) = runner;
           runner = boxcnt;
        end
        jj = jj + 1;
    end
    nxtdat(ii) = datptr(runner);
    datptr(runner) = ii;
end
used = 0;
for ii = 1:boxcnt
    if datptr(ii) ~= 0;
        used = used + 1;
    end
end
display(['Created: ', num2str(boxcnt)]);
display(['Used: ', num2str(used)]);
db.ndim = ndim;
db.ires = ires;
db.tau = tau;
db.datcnt = datcnt;
db.boxcnt = boxcnt;
db.datmax = datmax;
db.datmin = datmin;
db.boxlen = boxlen;
db.datptr = datptr(1:boxcnt);
db.nxtbox = nxtbox(1:boxcnt, 1:ndim);
db.where = where(1:boxcnt, 1:ndim);
db.nxtdat = nxtdat(1:datcnt);
db.data = data;
```

fet.m

```
function [out, SUM] = fet(db, dt, evolve, dismin, dismax, thmax)
% Computes Lyapunov exponent of given data and parameters, generates output
% textfile, exact replica of Fortran 77 version of fet
% Taehyeun Park, The Cooper Union, EE'15
out = [];
ndim = db.ndim;
ires = db.ires;
tau = db.tau;
```

```
datcnt = db.datcnt;
datmin = db.datmin;
boxlen = db.boxlen;
datptr = db.datptr;
nxtbox = db.nxtbox;
where = db.where;
nxtdat = db.nxtdat;
data = db.data;
delay = 0:tau:(ndim-1)*tau;
datuse = datcnt-(ndim-1)*tau-evolve;
its = 0;
SUM = 0;
savmax = dismax;
oldpnt = 1;
newpnt = 1;
fileID = fopen('fetout.txt', 'w');
goto50 = 1;
while goto50 == 1;
    goto50 = 0;
    [bstpnt, bstdis, thbest] = search(0, ndim, ires, datmin, boxlen, nxtbox, where, ...
        datptr, nxtdat, data, delay, oldpnt, newpnt, datuse, dismin, dismax,...
        thmax, evolve);

    while bstpnt == 0
        dismax = dismax * 2;
        [bstpnt, bstdis, thbest] = search(0, ndim, ires, datmin, boxlen, nxtbox, where, ...
            datptr, nxtdat, data, delay, oldpnt, newpnt, datuse, dismin, dismax,...
            thmax, evolve);
    end

    dismax = savmax;
    newpnt = bstpnt;
    disold = bstdis;
    iang = -1;

    goto60 = 1;
    while goto60 == 1;
        goto60 = 0;

        oldpnt = oldpnt + evolve;
        newpnt = newpnt + evolve;

        if oldpnt >= datuse
            return
        end

        if newpnt >= datuse
            oldpnt = oldpnt - evolve;
            goto50 = 1;
            break
        end

        p1 = data(oldpnt + delay);
        p2 = data(newpnt + delay);
        disnew = sqrt(sum((p2 - p1).^2));

        its = its + 1;
        SUM = SUM + log(disnew/disold);
        zlyap = SUM/(its*evolve*dt);%*log(2));
        out = [out; its*evolve, disold, disnew, zlyap, (oldpnt-evolve), (newpnt-evolve)];

        if iang == -1
            fprintf(fileID, '%-d\t\t\t%-8.4f\t\t%-8.4f\t\t%-8.4f\n', out(end,1:4)');
        else
```

```
            fprintf(fileID, '%-d\t\t\t%-8.4f\t\t%-8.4f\t\t%-8.4f\t\t%-d\n', [out(end,1:4), iang]');
        end
        if disnew <= dismax
            disold = disnew;
            iang = -1;
            goto60 = 1;
            continue
        end
        [bstpnt, bstdis, thbest] = search(1, ndim, ires, datmin, boxlen, nxtbox, where, ...
            datptr, nxtdat, data, delay, oldpnt, newpnt, datuse, dismin, dismax,...
            thmax, evolve);
        if bstpnt ~= 0
            newpnt = bstpnt;
            disold = bstdis;
            iang = floor(thbest);
            goto60 = 1;
            continue
        else
            goto50 = 1;
            break;
        end
    end
end
fclose(fileID);
```

search.m

```
    function [bstpnt, bstdis, thbest] = search(iflag, ndim, ires, datmin,...
    boxlen, nxtbox, where, datptr, nxtdat, data, delay, oldpnt, newpnt,...
    datuse, dismin, dismax, thmax, evolve)
% Searches for the most viable point for fet.m
% Taehyeun Park, The Cooper Union, EE'15
target = zeros(1,ndim);
oldcrd = zeros(1,ndim);
zewcrd = zeros(1,ndim);
oldcrd(1:ndim) = data(oldpnt+delay);
zewcrd(1:ndim) = data(newpnt+delay);
igcrds = floor((oldcrd - datmin)./boxlen);
oldist = sqrt(sum((oldcrd - zewcrd).^2));
irange = round(dismin/boxlen);
if irange == 0;
    irange = 1;
end
thbest = thmax;
bstdis = dismax;
bstpnt = 0;
goto30 = 1;
while goto30 == 1
    goto30 = 0;
    for icnt = 0:((2*irange+1)^ndim)-1
        goto140 = 0;
        icounter = icnt;
        for ii = 1:ndim;
            ipower = (2*irange+1)^(ndim-ii);
            ioff = floor(icounter./ipower);
            icounter = icounter - ioff*ipower;
            target(ii) = igcrds(ii) - irange + ioff;
            if target(ii) < 0
                goto140 = 1;
                break;
            end
            if target(ii) > ires-1
```

```
        goto140 = 1;
        break
    end
end

if goto140 == 1;
    continue
end

if irange ~= 1
    iskip = 1;
    for ii = 1:ndim
        if abs(round(target(ii) - igcrds(ii))) == irange
            iskip = 0;
        end
    end
    if iskip == 1
        continue
    end
end

runner = 1;
for ii = 1:ndim
    goto80 = 0;
    goto70 = 1;
    while goto70 == 1;
        goto70 = 0;
        if where(runner,ii) == target(ii)
            goto80 = 1;
            break
        end
        runner = nxtbox(runner, ii);
        if runner ~= 0
            goto70 = 1;
        end
    end

    if goto80 == 1
        continue
    end
    goto140 = 1;
    break
end

if goto140 == 1
    continue
end

if runner == 0
    continue
end
runner = datptr(runner);
if runner == 0
    continue
end
goto90 = 1;
while goto90 == 1
    goto90 = 0;
    while 1;
        if abs(round(runner - oldpnt)) < evolve
            break
        end
        if abs(round(runner - datuse)) < (2*evolve)
            break
```

```
                end

                bstcrd = data(runner + delay);

                abc1 = oldcrd(1:ndim) - bstcrd(1:ndim);
                abc2 = oldcrd(1:ndim) - zewcrd(1:ndim);
                tdist = sum(abc1.*abc1);
                tdist = sqrt(tdist);
                dot = sum(abc1.*abc2);
                if tdist < dismin
                    break
                end
                if tdist >= bstdis
                    break
                end
                if tdist == 0
                    break
                end
                goto120 = 0;
                if iflag == 0
                    goto120 = 1;
                end
                if goto120 == 0
                    ctheta = min(abs(dot/(tdist*oldist)),1);
                    theta = 57.3*acos(ctheta);
                    if theta >= thbest
                        break
                    end
                    thbest = theta;
                end
                bstdis = tdist;
                bstpnt = runner;
                break;
            end
            runner = nxtdat(runner);
            if runner ~= 0
                goto90 = 1;
            end
        end
    end
    irange = irange + 1;
    if irange <= (0.5 + round((dismax/boxlen)))
        goto30 = 1;
        continue;
    end
    return
end
```

makeplot.m

```
function [] = makeplot(db, out, evolve, loc)
% Plots 2D or 3D attractor evolution by evolution, 4th parameter is the
% location of legend
% Taehyeun Park, The Cooper Union, EE'15
datcnt = db.datcnt;
ndim = db.ndim;
tau = db.tau;
dataplot = [];
freerun = 0;
delay = 0:tau:(ndim-1)*tau;
data = db.data;
for ii = 1:(datcnt-(ndim-1)*tau)
```

```
    dataplot = [dataplot; data(ii+delay)];
end
figure, bar(out(:,1),out(:,3)), hold on;
mle = max(dataplot(:)) - min(dataplot(:));
plot([0, out(end,1)], [mle, mle], 'r', 'LineWidth', 1.5), hold off;
set(gca,'YTick', [0, mle])
axis([0, out(end,1), 0, 1.1*mle])
title('d_f of evolutions scaled to the maximum linear extent of the attractor')
if ndim == 2
    figure('Position', [100, 100, 800, 500]);
    plot(dataplot(:,1), dataplot(:,2), '.', 'MarkerSize', 3), hold on;
    display('To see the next evolution, press enter')
    display('To clear the screen and then see the next evolution, type c and press enter')
    display('To proceed without stopping, type r and press enter')
    display('To terminate plot generating, type g and press enter')

    for ii = 1:size(out,1)
        if freerun == 0
            RESET = input('Next evolution?  ', 's');
            if strcmp(RESET, 'c')
                display('Screen cleared')
                hold off;
                clf;
                plot(dataplot(:,1), dataplot(:,2), '.', 'MarkerSize', 3), hold on;
            elseif strcmp(RESET, 'r')
                display('Evolving without stopping...')
                display('Press ctrl+c to terminate')
                freerun = 1;
            elseif strcmp(RESET, 'g')
                display('Plot generating stopped')
                return;
            else
                if ii > 1
                    delete(ann)
                end
            end
        end

        tmpold = out(ii,5);
        oldpnt = tmpold + evolve;
        tmpnew = out(ii,6);
        newpnt = tmpnew + evolve;

        plot(data(tmpold:oldpnt), data((tmpold+tau):(oldpnt+tau)), 'r', 'LineWidth', 1);
        plot(data(tmpnew:newpnt), data((tmpnew+tau):(newpnt+tau)), 'g', 'LineWidth', 1);
        for aa = 0:evolve;
            plot([data(tmpold+aa), data(tmpnew+aa)], [data(tmpold+aa+tau), data(tmpnew+aa+tau)], 'LineWidth', 1)
        end

        ann = legend(['Iteration: ', num2str(out(ii,1)), '/', num2str(out(end,1)), char(10)...
                      'd_i:', num2str(out(ii,2)), char(10)...
                      'd_f:', num2str(out(ii,3)), char(10)...
                      'Current Estimate:' num2str(out(ii,4))], ...
                      'location', loc);
        if freerun == 1
            drawnow
        end
    end

elseif ndim == 3
    figure('Position', [100, 100, 800, 500]);
    plot3(dataplot(:,1), dataplot(:,2), dataplot(:,3), '.', 'MarkerSize', 3), hold on;
    display('To see the next evolution, press enter')
    display('To clear the screen and then see the next evolution, type c and press enter')
```

```
    display('To proceed without stopping, type r and press enter')
    display('To terminate plot generating, type g and press enter')
    for ii = 1:size(out,1)
        if freerun == 0
            RESET = input('Next evolution?  ', 's');
            if strcmp(RESET, 'c')
                display('Screen cleared')
                hold off;
                clf;
                plot3(dataplot(:,1), dataplot(:,2), dataplot(:,3), '.', 'MarkerSize', 3), hold on;
            elseif strcmp(RESET, 'r')
                display('Evolving without stopping...')
                display('Press ctrl+c to terminate')
                freerun = 1;
            elseif strcmp(RESET, 'g')
                display('Plot generating stopped')
                return;
            else
                if ii > 1
                    delete(ann)
                end
            end
        end

        tmpold = out(ii,5);
        oldpnt = tmpold + evolve;
        tmpnew = out(ii,6);
        newpnt = tmpnew + evolve;

        plot3(data(tmpold:oldpnt), data((tmpold+tau):(oldpnt+tau)), data((tmpold+(2*tau)):(oldpnt+(2*tau))), 'r', 'LineWidth', 1);
        plot3(data(tmpnew:newpnt), data((tmpnew+tau):(newpnt+tau)), data((tmpnew+(2*tau)):(newpnt+(2*tau))), 'g', 'LineWidth', 1);
        for aa = 0:evolve;
            plot3([data(tmpold+aa), data(tmpnew+aa)], [data(tmpold+aa+tau), data(tmpnew+aa+tau)], [data(tmpold+aa+(2*tau)), data(tmpnew+aa+(2*tau)
        end

        ann = legend(['Iteration: ', num2str(out(ii,1)), '/', num2str(out(end,1)), char(10)...
                      'd_i:', num2str(out(ii,2)), char(10)...
                      'd_f:', num2str(out(ii,3)), char(10)...
                      'Current Estimate:' num2str(out(ii,4))], ...
                      'location', loc);
        if freerun == 1
            drawnow
        end
    end
end
```

A continuación se muestran las tablas de los resultados obtenidos

Tabla 4. Resultados del escenario 1 Presa- 1 Depredador en el simulado V-REP. *(PD:Depredador, PY:Presa, PDD:Predador muerto, PYD:Presa muerta)*; ** Limite de tiempo para pruebas, 300s.

No.	==	==	Tiempo (Seg)	==	==
==	Inicio**	30s.	60s.	90s.	300s.**
1	1.65s PY:30, PYD:0, PD:4, PDD:0	PY:17, PYD:10, PD:6, PDD:1	PY:2, PYD:15, PD:10, PDD:7	PY:9, PYD:4, PD:11, PDD:10	PY:0, PYD:5, PD:5, PDD:24
2	0.75s PY:29, PYD:1, PD:4, PDD:0	PY:16, PYD:10, PD:6, PDD:2	PY:2, PYD:18, PD:7, PDD:7	PY:12, PYD:11, PD:6, PDD:5	PY:4, PYD:17, PD:2, PDD:11
3	0.55s PY:28, PYD:2, PD:4, PDD:0	PY:20, PYD:7, PD:6, PDD:1	PY:1, PYD:12, PD:18, PDD:3	PY:0, PYD:1, PD:18, PDD:15	PY:0, PYD:1, PD:10, PDD:23
4	0.55s PY:29, PYD:1, PD:4, PDD:0	PY:15, PYD:9, PD:8, PDD:2	PY:0, PYD:13, PD:14, PDD:7	PY:0, PYD:7, PD:17, PDD:10	(220s) PY:0, PYD:1, PD:0, PDD:33
5	0.55s PY:28, PYD:2, PD:4, PDD:0	PY:17, PYD:10, PD:6, PDD:1	PY:4, PYD:13, PD:10, PDD:7	PY:7, PYD:5, PD:9, PDD:13	PY:0, PYD:4, PD:7, PDD:23
6	0.7s PY:29, PYD:1, PD:4, PDD:0	PY:17, PYD:8, PD:7, PDD:2	PY:5, PYD:15, PD:9, PDD:5	PY:8, PYD:8, PD:11, PDD:7	PY:0, PYD:3, PD:10, PDD:21
7	0.9s PY:29, PYD:1, PD:4, PDD:0	PY:19, PYD:8, PD:6, PDD:1	PY:3, PYD:16, PD:6, PDD:9	PY:4, PYD:5, PD:11, PDD:14	PY:0, PYD:0, PD:4, PDD:30

Tabla 5. Resultados del escenario 1 Presa- 1 Depredador en el simulado V-REP. (Ubicación de los depredadores modificada)*(PD:Depredador, PY:Presa, PDD:Predador muerto, PYD:Presa muerta)*; ** Limite de tiempo para pruebas, 300s.

No.	==	==	==	==	==	Tiempo (Seg)	==	==	==	==	==
==	Inicio**	30s.	60s.	90s.	120s	150s	180s	210s	240s	270s	300s.**
1	0.8s PY:29 PYD:1 PD:4 PDD:0	PY:23 PYD:4 PD:7 PDD:0	PY:13 PYD:14 PD:6 PDD:1	PY:18 PYD:10 PD:5 PDD:1	PY:11 PYD:16 PD:7 PDD:0	PY:3 PYD:19 PD:10 PDD:2	PY:1 PYD:18 PD:5 PDD:10	PY:0 PYD:17 PD:6 PDD:11	PY:0 PYD:14 PD:8 PDD:12	PY:0 PYD:13 PD:6 PDD:15	PY:0 PYD:13 PD:8 PDD:13
2	0.95s PY:29 PYD:1 PD:4 PDD:0	PY:23 PYD:6 PD:5 PDD:0	PY:9 PYD:14 PD:9 PDD:2	PY:11 PYD:10 PD:9 PDD:4	PY:13 PYD:7 PD:8 PDD:6	PY:10 PYD:9 PD:8 PDD:7	PY:9 PYD:10 PD:4 PDD:11	PY:6 PYD:13 PD:4 PDD:11	PY:5 PYD:14 PD:4 PDD:11	PY:10 PYD:9 PD:3 PDD:12	PY:10 PYD:9 PD:1 PDD:14
3	6.5s PY:27 PYD:3 PD:4 PDD:0	PY:23 PYD:5 PD:6 PDD:0	PY:8 PYD:18 PD:6 PDD:2	PY:11 PYD:5 PD:13 PDD:5	PY:8 PYD:6 PD:10 PDD:10	PY:9 PYD:5 PD:10 PDD:10	PY:6 PYD:7 PD:5 PDD:16	PY:4 PYD:8 PD:9 PDD:13	PY:3 PYD:9 PD:5 PDD:17	PY:5 PYD:8 PD:3 PDD:18	PY:5 PYD:8 PD:2 PDD:19
4	8.5s PY:24 PYD:6 PD:4 PDD:0	PY:20 PYD:9 PD:4 PDD:1	PY:14 PYD:16 PD:1 PDD:3	PY:20 PYD:11 PD:3 PDD:0	PY:14 PYD:16 PD:4 PDD:0	PY:7 PYD:19 PD:5 PDD:3	PY:5 PYD:22 PD:1 PDD:6	PY:3 PYD:24 PD:1 PDD:6	PY:3 PYD:25 PD:0 PDD:6	PY:3 PYD:26 PD:0 PDD:5	PY:2 PYD:27 PD:0 PDD:5

Tabla 6. Resultados para los diferentes experimentos. *(PD:Depredador, PY:Presa, PDD:Predador muerto, PYD:Presa muerta, TP: Depredador Tope, TPD: Depredador Tope Muerto, NR: No responde)*; ** Limite de tiempo para pruebas, 300s.

No.	==	==	==	==	==	Tiempo	==	==	==	==	==
==	Inicio**	30s.	60s.	90s.	120s.	150s.	180s.	210s.	240s.	270s.	300s.**
1	-0.55s- PY:19, PYD:11, PD:19, PDD:1, TP:8, TPD:0	PY:17, PYD:7, PD:13, PDD:8, TP:9 TPD:4	PY:4, PYD:10, PD:13, PDD:14, TP:8, TPD:9	PY:1, PYD:4, PD:11, PDD:14, TP:17, TPD:11	PY:0, PYD:2, PD:3, PDD:10, TP:28, TPD:15	PY:0, PYD:1, PD:3, PDD:9, TP:21, TPD:24	PY:0, PYD:1, PD:4, PDD:3, TP:28, TPD:22	PY:0, PYD:1, PD:3, PDD:3, TP:23, TPD:28	PY:0, PYD:0, PD:2, PDD:4, TP:16, TPD:36	PY:0, PYD:0, PD:2, PDD:4, TP:14, TPD:38	PY:0, PYD:0, PD:0, PDD:3, TP:10, TPD:45
2	-0.65s- NR:1 PY:27, PYD:2, PD:20, PDD:0, TP:8, TPD:0	NR:1, PY:8, PYD:13, PD:13, PDD:8, TP:14, TPD:1	NR:1, PY:0, PYD:7, PD:6, PDD:16, TP:19, TPD:9	NR:1 PY:0, PYD:1, PD:3, PDD:9, TP:21, TPD:23	NR:1, PY:0, PYD:1, PD:0, PDD:7, TP:27, TPD:22	NR:1, PY:0, PYD:0, PD:0, PDD:1, TP:26, TPD:30	NR:1, PY:0, PYD:0, PD:0, PDD:1, TP:23, TPD:33	NR:1, PY:0, PYD:0, PD:0, PDD:1, TP:21, TPD:35	NR:1, PY:0, PYD:0, PD:0, PDD:1, TP:17, TPD:39	NR:1, PY:0, PYD:0, PD:0, PDD:1, TP:10, TPD:46	NR:1, PY:0, PYD:0, PD:0, PDD:1, TP:12, TPD:44
3	-0.55s- PY:17, PYD:13, PD:20, PDD:0, TP:8, TPD:0	PY:18, PYD:7, PD:12, PDD:9, TP:10 TPD:2	PY:9, PYD:12, PD:9, PDD:12, TP:8, TPD:8	PY:9, PYD:7, PD:15, PDD:11, TP:10, TPD:6	PY:5, PYD:7, PD:11, PDD:15, TP:10, TPD:10	PY:3, PYD:6, PD:9, PDD:17, TP:11, TPD:12	PY:5, PYD:3, PD:11, PDD:10, TP:14, TPD:15	PY:4, PYD:4, PD:4, PDD:13, TP:14, TPD:19	PY:0, PYD:8, PD:3, PDD:13, TP:11, TPD:23	PY:0, PYD:8, PD:1, PDD:8, TP:18, TPD:23	PY:0, PYD:4, PD:1, PDD:4, TP:13, TPD:36

Tabla 7. Resultados para los diferentes experimentos. Cont.. *(PD:Depredador, PY:Presa, PDD:Predador muerto, PYD:Presa muerta, TP: Depredador Tope, TPD: Depredador Tope Muerto, NR: No responde)*; ** Limite de tiempo para pruebas, 300s.

No.	==	==	==	==	==	Tiempo	==	==	==	==	==
==	Inicio**	30s.	60s.	90s.	120s.	150s.	180s.	210s.	240s.	270s.	300s.**
4	-0.65s- PY:18, PYD:12, PD:20, PDD:0, TP:8, TPD:0	PY:16, PYD:10, PD:11, PDD:10, TP:10, TPD:1	PY:5, PYD:12, PD:9, PDD:14, TP:13, TPD:5	PY:2, PYD:5, PD:14, PDD:11, TP:19, TPD:7	PY:0, PYD:3, PD:7, PDD:20, TP:19, TPD:9	PY:0, PYD:1, PD:9, PDD:11, TP:28, TPD:9	PY:0, PYD:0, PD:2, PDD:16, TP:18, TPD:22	PY:0, PYD:0, PD:1, PDD:10, TP:21, TPD:26	PY:0, PYD:0, PD:0, PDD:7, TP:15, TPD:36	PY:0, PYD:0, PD:0, PDD:7, TP:14, TPD:37	PY:0, PYD:0, PD:0, PDD:5, TP:15, TPD:38
5	-0.5s- PY:22, PYD:8, PD:19, PDD:1, TP:8, TPD:0	PY:17, PYD:9, PD:12, PDD:7, TP:10 TPD:3	PY:6, PYD:13, PD:17, PDD:7, TP:9, TPD:6	PY:0, PYD:7, PD:21, PDD:13, TP:10, TPD:7	PY:0, PYD:1, PD:15, PDD:15, TP:16, TPD:11	PY:0, PYD:1, PD:9, PDD:14, TP:20, TPD:14	PY:0, PYD:1, PD:7, PDD:10, TP:19, TPD:21	PY:0, PYD:1, PD:6, PDD:10, TP:14, TPD:27	PY:0, PYD:1, PD:0, PDD:11, TP:10, TPD:36	PY:0, PYD:1, PD:0, PDD:8, TP:14, TPD:35	PY:0, PYD:1, PD:0, PDD:6, TP:11, TPD:40
6	-0.55s- PY:21, PYD:9, PD:20, PDD:0, TP:8, TPD:0	PY:24, PYD:3, PD:12, PDD:7, TP:10 TPD:2	PY:8, PYD:15, PD:12, PDD:6, TP:11, TPD:6	PY:7, PYD:5, PD:16, PDD:9, TP:12, TPD:9	PY:6, PYD:4, PD:10, PDD:16, TP:12, TPD:10	PY:2, PYD:6, PD:6, PDD:17, TP:14, TPD:13	PY:5, PYD:3, PD:6, PDD:11, TP:16, TPD:17	PY:6, PYD:2, PD:6, PDD:10, TP:14, TPD:20	PY:6, PYD:2, PD:2, PDD:9, TP:11, TPD:28	PY:4, PYD:4, PD:0, PDD:6, TP:12, TPD:32	-306s- PY:5, PYD:3, PD:0, PDD:3, TP:8, TPD:39
7	-0.65s- NR:1 ,PY:24, PYD:5, PD:19, PDD:1, TP:8, TPD:0	NR:1, PY:13, PYD:8, PD:11, PDD:10, TP:12, TPD:3	NR:1, PY:5, PYD:11, PD:7, PDD:10, TP:13, TPD:11	NR:1, PY:4, PYD:6, PD:5, PDD:11, TP:10, TPD:21	NR:1, PY:0, PYD:5, PD:7, PDD:8, TP:11, TPD:26	NR:1, PY:0, PYD:3, PD:2, PDD:6, TP:20, TPD:26	NR:1, PY:0, PYD:0, PD:1, PDD:4, TP:20, TPD:32	NR:1, PY:0, PYD:0, PD:0, PDD:4, TP:20, TPD:33	NR:1, PY:0, PYD:0, PD:0, PDD:4, TP:11, TPD:42	NR:1, PY:0, PYD:0, PD:0, PDD:3, TP:23, TPD:31	NR:1, PY:0, PYD:0, PD:0, PDD:3, TP:15, TPD:39